



Reprenez le contrôle à l'aide de Linux !

Par Mathieu Nebra (M@teo21)



*Licence Creative Commons BY-NC-SA 2.0
Dernière mise à jour le 20/02/2012*

Sommaire

Sommaire	1
Lire aussi	5
Reprenez le contrôle à l'aide de Linux !	7
Partie 1 : Découvrir et installer Linux	7
Mais c'est quoi Linux ?	8
Un système d'exploitation	8
Le boot : démarrage de l'ordinateur	9
Linux est un système d'exploitation	10
La naissance de Linux	12
L'informatique en 1984	12
Le projet GNU	12
Pendant ce temps, Linus Torvalds s'amusait	14
Résumons avec un schéma !	14
Les distributions de Linux	15
Les différentes distributions existantes	15
La distribution Debian	16
Téléchargez Linux, c'est gratuit !	19
Les deux visages de Linux	20
En mode console	20
En mode graphique	21
Schéma résumé à retenir	24
Télécharger et graver le CD	24
1/ Récupérer l'ISO	24
2/ Graver le CD	25
Tester et installer Ubuntu	27
Installation de Linux depuis Windows	28
Premier démarrage d'Ubuntu	30
Modifier l'ordre de boot	31
Essayer ou installer Ubuntu	32
Installer Ubuntu	34
Etape 1 : lancer l'installation et choisir la langue	35
Etape 2 : préparation de l'installation	35
Etape 3 : partitionnement du disque dur	37
Partitionner son disque	37
Défragmentez votre disque	38
Un disque dur, ça ressemble à quoi ?	38
L'importance de la défragmentation	39
Sous Windows XP	40
Sous Windows 7	42
Qu'est-ce que le partitionnement ?	43
Vive les schémas !	43
Les systèmes de fichiers	45
Prêts ? Partitionnez !	46
Le partitionnement manuel	46
La suite et fin de l'installation	52
Sélection du fuseau horaire	52
Le type de clavier	53
Création du compte utilisateur	54
Importation des données de Windows	55
Suite et fin de l'installation	56
GRUB au démarrage	56
Découverte du bureau Gnome	58
Bienvenue sur le bureau Gnome	58
Les options	59
Présentation du bureau Gnome	59
Nautilus, l'explorateur de fichiers	62
Ajouter / supprimer des programmes	63
Ajout et suppression des programmes	63
Mise à jour des programmes	65
Découverte du bureau KDE	67
Comment obtenir KDE ?	67
Connexion au bureau KDE	68
KDM, le programme de login de KDE (Kubuntu)	68
Le bureau et le menu K	69
Le tableau de bord	70
Les boutons à gauche du tableau de bord	70
Les boutons à droite du tableau de bord	73
L'explorateur de fichiers Dolphin	73
Partie 2 : Manipuler la console et les fichiers	76
La console, ça se mange ?	77
Pourquoi avoir inventé la console ?	77
La console, la vraie, celle qui fait peur	79
Le login	80
Les différentes consoles	81

La console en mode graphique	82
L'accès à distance en SSH avec Putty	84
Telnet et SSH	85
PuTTY	85
Entrer une commande	89
L'invite de commandes	89
Commandes et paramètres	90
Une commande simple	91
Les paramètres	91
Retrouver une commande	94
Autocomplétion de commande	94
L'historique des commandes	95
Ctrl + R : rechercher une commande tapée avec quelques lettres	96
Quelques raccourcis claviers pratiques	96
La structure des dossiers et fichiers	98
Organisation des dossiers	99
2 types de fichiers	99
La racine	99
Architecture des dossiers	99
Les dossiers de la racine	100
Schéma résumé de l'architecture	101
pwd & which : où... où suis-je ?	101
pwd : afficher le dossier actuel	101
which : connaître l'emplacement d'une commande	102
ls : lister les fichiers et dossiers	103
-a : afficher tous les fichiers et dossiers cachés	103
-F : indique le type d'élément	104
-l : liste détaillée	104
-h : afficher la taille en Ko, Mo, Go...	105
-t : trier par date de dernière modification	105
cd : changer de dossier	106
Les chemins relatifs	109
Les chemins absolus	109
Retour au répertoire home	111
Autocomplétion du chemin	111
du : taille occupée par les dossiers	112
-h : la taille pour les humains	113
-a : afficher la taille des dossiers ET des fichiers	113
-s : avoir juste le grand total	113
Manipuler les fichiers	114
cat & less : afficher un fichier	115
cat : afficher tout le fichier	116
less : afficher le fichier page par page	118
head & tail : afficher le début et la fin d'un fichier	121
head : afficher le début du fichier	121
tail : afficher la fin du fichier	122
touch & mkdir : créer des fichiers et dossiers	123
touch : créer un nouveau fichier	124
mkdir : créer un nouveau dossier	125
cp & mv : copier et déplacer un fichier	125
cp : copier un fichier	126
mv : déplacer un fichier	127
rm : supprimer des fichiers et dossiers	129
rm : supprimer un fichier	129
rm et le joker de la Mort (qui tue)	130
ln : créer des liens entre fichiers	132
Le stockage des fichiers	132
Créer des liens physiques	133
Créer des liens symboliques	134
Les utilisateurs et les droits	136
sudo : exécuter une commande en root	136
L'organisation des utilisateurs sous Linux	136
sudo : devenir root un instant	137
sudo su : devenir root et le rester	138
adduser : ajouter un utilisateur	138
adduser : ajouter un utilisateur	139
passwd : changer le mot de passe	139
deluser : supprimer un compte	140
addgroup : ajouter un groupe	140
addgroup : créer un groupe	141
usermod : modifier un utilisateur	141
delgroup : supprimer un groupe	142
chown : changer le propriétaire d'un fichier	142
chown : changer le propriétaire d'un fichier	142
chgrp : changer le groupe propriétaire d'un fichier	143
chown peut aussi changer le groupe propriétaire d'un fichier !	143
-R : affecter récursivement les sous-dossiers	143
chmod : modifier les droits d'accès	144
Le fonctionnement des droits	144
chmod : modifier les droits d'accès	146
Nano, l'éditeur de texte du débutant	149
Premiers pas avec nano	150

nano est un éditeur de texte, pas un traitement de texte !	150
Découverte de nano	151
Les raccourcis clavier de nano	153
Les paramètres de la commande nano	157
Configurer nano avec .nanorc	158
Pourquoi .nanorc ?	158
Création du .nanorc	159
Le nanorc global et la coloration syntaxique	160
Personnaliser la coloration syntaxique	163
Configurer sa console avec .bashrc	164
Edition du .bashrc personnel	164
Edition du bashrc global	168
Et aussi... le .profile	168
Installer des programmes avec apt-get	170
Les paquets et leurs dépendances	170
Des programmes livrés sous forme de paquets	170
Les dépendances, un cauchemar ?	170
Les dépôts	171
La notion de dépôt	171
Gérer ses dépôts	173
Utiliser l'outil graphique	175
Les outils de gestion des paquets	177
apt-get update : mettre à jour le cache des paquets	179
apt-cache search : rechercher un paquet	180
apt-get install : installer un paquet	180
apt-get autoremove : supprimer un paquet	184
apt-get upgrade : mettre à jour tous les paquets	185
Les autres gestionnaires de paquets	186
Bonus track	186
RTFM : lisez le manuel !	187
man : afficher le manuel d'une commande	188
Se déplacer dans le manuel	189
Les principales sections du manuel	189
La langue des pages de manuel	189
Comprendre le synopsis	191
man mkdir	191
man cp	193
man apt-get	195
Résumé de la syntaxe du SYNOPSIS	196
apropos : trouver une commande	196
D'autres façons de lire le manuel	198
Le paramètre -h (et --help)	198
La commande whatis	199
La commande info	200
Le manuel sous Konqueror	200
Rechercher man sur le Web	200
Rechercher des fichiers	201
locate : une recherche rapide	202
Utiliser locate	202
La base de données des fichiers	202
find : une recherche approfondie	203
find recherche les fichiers actuellement présents	203
Fonctionnement de la commande find	204
Utilisation basique de la commande find	204
Utilisation avancée avec manipulation des résultats	207
Partie 3 : Contrôler les processus et les flux de données	209
Extraire, trier et filtrer des données	210
grep : filtrer des données	210
Utiliser grep simplement	210
Utiliser grep avec des expressions régulières	213
sort : trier les lignes	215
wc : compter le nombre de lignes	217
uniq : supprimer les doublons	219
cut : couper une partie du fichier	220
Couper selon le nombre de caractères	220
Couper selon un délimiteur	221
Les flux de redirection	223
> et >> : rediriger le résultat dans un fichier	224
Préparatifs	224
> : rediriger dans un nouveau fichier	225
>> : rediriger à la fin d'un fichier	226
Résumé	227
2>, 2>> et 2>&1 : rediriger les erreurs	227
Rediriger les erreurs dans un fichier à part	228
Fusionner les sorties	229
Résumé	229
< et << : lire depuis un fichier ou le clavier	230
< : lire depuis un fichier	230
<< : lire depuis le clavier progressivement	231
Résumé	233
: chaîner les commandes	233
La théorie	233

La pratique	234
Résumé	238
Surveiller l'activité du système	238
w : qui fait quoi ?	239
L'heure (aussi accessible via "date")	239
L'uptime (aussi accessible via "uptime")	240
La charge (aussi accessible via "uptime" et "lload")	240
La liste des connectés (aussi accessible via "who")	241
ps & top : lister les processus	242
ps : liste des processus statique	242
top : liste des processus dynamique	245
ctrl+c & kill : arrêter un processus	247
Ctrl + C : arrêter un processus lancé en console	247
kill : tuer un processus	247
killall : tuer plusieurs processus	249
halt & reboot : arrêter et redémarrer l'ordinateur	249
halt : arrêter l'ordinateur	250
reboot : redémarrer l'ordinateur	250
Exécuter des programmes en arrière-plan	250
"&" & nohup : lancer un processus en arrière-plan	251
& : lancer un processus en arrière-plan	251
nohup : détacher le processus de la console	252
ctrl + z, jobs, bg & fg : passer un processus en arrière-plan	253
Ctrl + Z : mettre en pause l'exécution du programme	253
bg : passer le processus en arrière-plan (background)	253
jobs : connaître les processus qui tournent en arrière-plan	254
fg : reprendre un processus au premier plan (foreground)	254
Résumé des états possibles des processus	254
screen : plusieurs consoles en une	255
Ctrl + a puis ? : afficher l'aide	257
Les principales commandes de screen	257
Ctrl + a puis S : découper screen en plusieurs parties ("split")	258
Ctrl + a puis d : détacher screen	259
Un fichier personnalisé de configuration de screen	260
Exécuter un programme à une heure différée	261
date : régler l'heure	262
Personnaliser l'affichage de la date	262
Modifier la date	263
at : exécuter une commande plus tard	263
Exécuter une commande à une heure précise	264
Exécuter une commande après un certain délai	265
atq et atrm : lister et supprimer les jobs en attente	265
sleep : faire une pause	266
crontab : exécuter une commande régulièrement	267
Un peu de configuration...	267
La crontab, qu'est-ce que c'est ?	267
Modifier la crontab	268
Partie 4 : Transférer des données à travers le réseau	272
Archiver et compresser	273
tar : assembler des fichiers dans une archive	273
Regrouper d'abord les fichiers dans un même dossier	274
-cvf : créer une archive tar	275
-tf : afficher le contenu de l'archive sans l'extraire	276
-rvf : ajouter un fichier	276
-xvf : extraire les fichiers de l'archive	276
gzip & bzip2 : compresser une archive	277
gzip : la compression la plus courante	277
bzip2 : la compression la plus puissante	278
Archiver et compresser en même temps avec tar	278
zcat, zmore & zless : afficher directement un fichier compressé	279
unzip & unrar : décompresser les .zip et .rar	280
unzip : décompresser un .zip	280
unrar : décompresser un .rar	281
La connexion sécurisée à distance avec SSH	282
Se connecter à une console à distance	283
De Telnet à SSH, pourquoi faut-il sécuriser les échanges ?	284
Les protocoles	284
Le protocole Telnet : simple mais dangereux	284
Le protocole SSH : la solution pour sécuriser les données	286
Comment sont cryptés les échanges avec SSH ?	286
Quelles sont les différentes méthodes de cryptage qui existent ?	286
La création d'un tunnel sécurisé avec SSH	289
Se connecter avec SSH et PuTTY	292
Transformer sa machine en serveur	293
Se connecter via SSH à partir d'une machine Linux	294
Se connecter via SSH à partir d'une machine Windows	295
L'identification automatique par clé	299
Authentification par clé depuis Linux	300
Authentification par clé depuis Windows (PuTTY)	303
Transférer des fichiers	313
wget : téléchargement de fichiers	313
Reprendre un téléchargement arrêté	314

Lancer un téléchargement en tâche de fond	314
scp : copier des fichiers sur le réseau	314
Copier un fichier de votre ordinateur vers un autre	315
Copier un fichier d'un autre ordinateur vers le vôtre	316
Le piège du port	316
ftp & sftp : transférer des fichiers	317
Connexion à un serveur FTP	317
Se déplacer au sein du serveur FTP	318
Le transfert de fichiers	319
Les autres commandes	320
sftp : un FTP sécurisé	320
rsync : synchroniser des fichiers pour une sauvegarde	321
Sauvegarder dans un autre dossier du même ordinateur	322
Sauvegarder sur un autre ordinateur	324
Analyser le réseau et filtrer le trafic avec un firewall	325
host & whois : qui êtes-vous ?	325
Convertir une IP en nom d'hôte et inversement	326
Gérer les noms d'hôte personnalisés	327
whois : tout savoir sur un nom de domaine	328
ifconfig & netstat : gérer et analyser le trafic réseau	328
ifconfig : liste des interfaces réseau	329
netstat : statistiques sur le réseau	330
iptables : le firewall de référence	333
Iptables s'utilise en root	335
iptables -L : afficher les règles	335
Le principe des règles	336
Ajouter et supprimer des règles	336
Autoriser les pings	338
Autoriser les connexions locales et déjà ouvertes	338
Refuser toutes les autres connexions par défaut	338
Appliquer les règles au démarrage	339
Partie 5 : Programmez des scripts Bash pour automatiser vos tâches	339
Vim : l'éditeur de texte du programmeur	340
Installer Vim	340
Vim ou Emacs ? Emacs ou Vim ?	340
Installer et lancer Vim	341
vimtutor : le programme qui vous apprend à utiliser Vim !	341
Les modes d'édition de Vim	342
Opérations basiques (déplacement, écriture, enregistrement...)	344
L'ouverture de Vim	344
i : insérer du texte	345
Le déplacement	346
:w : enregistrer le fichier	347
:q : quitter	348
:wq : enregistrer puis quitter	348
Opérations standard (copier, coller, annuler...)	348
x : effacer des lettres	348
d : effacer des mots, lignes...	348
yy : copier une ligne en mémoire	349
p : coller	349
r : remplacer une lettre	350
u : annuler les modifications	350
G : sauter à la ligne n°X	350
Opérations avancées (split, fusion, recherche...)	351
/recherche : rechercher un mot	351
:s : rechercher et remplacer du texte	351
:r : fusion de fichiers	351
Le découpage d'écran (split)	351
:! : lancer une commande externe	353
Les options de Vim	353
Le fonctionnement des options	354
syntax : activer la coloration syntaxique	355
background : coloration sur un fond sombre	356
number : afficher les numéros de ligne	357
showcmd : afficher la commande en cours	358
ignorecase : ignorer la casse lors de la recherche	358
mouse : activer le support de la souris	358
Introduction aux scripts shell	360
Qu'est-ce qu'un shell ?	360
Il existe plusieurs environnements console : les shell	360
A quoi sert un shell ?	361
Installer un nouveau shell	363
Quelle importance a tout ceci lorsqu'on réalise un script shell ?	364
Notre premier script	364
Création du fichier	364
Indiquer le nom du shell utilisé par le script	364
Exécution de commandes	365
Les commentaires	365
Exécuter le script bash	366
Donner les droits d'exécution au script	366
Exécution du script	367
Exécution de débogage	367

Créer sa propre commande	368
Afficher et manipuler des variables	369
Déclarer une variable	369
echo : afficher une variable	370
Afficher une variable	371
Les quotes	372
read : demander une saisie	373
Affecter simultanément une valeur à plusieurs variables	374
-p : afficher un message de prompt	374
-n : limiter le nombre de caractères	375
-t : limiter le temps autorisé pour saisir un message	375
-s : ne pas afficher le texte saisi	376
Effectuer des opérations mathématiques	376
Les variables d'environnement	377
Les variables des paramètres	379
Les tableaux	380
Les conditions	382
if : la condition la plus simple	383
Si	383
Sinon	384
Sinon si	386
Les tests	387
Les différents types de tests	387
Effectuer plusieurs tests à la fois	390
Inverser un test	391
case : tester plusieurs conditions à la fois	391
Les boucles	394
while : boucler "tant que"	394
for : boucler sur une liste de valeurs	395
Parcourir une liste de valeurs	395
Un for plus classique	397
TP : Générateur de galerie d'images	398
Objectifs	399
Le rendu final	399
Le code HTML de base	400
Comment générer des miniatures d'images ?	400
Les paramètres	400
Solution	400
Améliorations	402



Reprenez le contrôle à l'aide de Linux !

Par  Mathieu Nebra (M@teo21)

Mise à jour : 20/11/2011

Difficulté : Facile  Durée d'étude : 1 mois, 15 jours



76 354 visites depuis 7 jours, classé 6/778

"Linux c'est trop compliqué, c'est pour les pros"

(Dire qu'il y a des gens qui croient ça !)

... Comment ça... c'est ce que vous croyez vous aussi ? 🤔

Halte-là, malheureux ! Ne faites pas un pas de plus, vous faites fausse route !

Linux n'est pas compliqué, et je vais vous le prouver.

Vous ne savez pas ce qu'est Linux ? Ce n'est pas grave, c'est un cours pour débutants : les explications commencent dès le premier chapitre !



A qui s'adresse ce cours ?

- Aux utilisateurs de Windows qui veulent découvrir Linux
- Aux linuxiens débutants qui cherchent à mieux maîtriser leur OS
- Aux webmasters qui doivent administrer un serveur dédié sous Linux
- Aux curieux comme vous qui se demandent juste comment Linux fonctionne 🤔

Grâce à Linux, vous avez la possibilité aujourd'hui de *reprendre le contrôle* de votre ordinateur et de découvrir tout un nouveau monde passionnant, le tout sans dépenser un sou ! 😊



Ce cours vous plaît ?

Si vous avez aimé ce cours, vous pouvez retrouver le livre "*Reprenez le contrôle à l'aide de Linux*" du même auteur, en vente [sur le Site du Zéro](#), en librairie et dans les boutiques en ligne. Vous y trouverez ce cours adapté au format papier avec une série de chapitres inédits.

[Plus d'informations](#)

Partie 1 : Découvrir et installer Linux

Dans cette première partie, nous allons découvrir Linux en douceur. Nous commencerons par voir ce que c'est, parce que ça ne sert à rien de l'installer juste "pour la beauté du geste" sans savoir à quoi ça va nous servir 🤪

📄 Mais c'est quoi Linux ?

Linux ? Difficile de ne pas en entendre parler aujourd'hui. A moins d'être resté enfermé dans un bunker anti-nucléaire coupé de tout lien avec le reste du monde, vous avez forcément rencontré ce mot quelque part sur le net ou au cours d'une conversation entre amis.

Ce n'est pas étonnant : **Linux est partout**. Par exemple : sans Linux, le Site du Zéro n'existerait pas. En fait, sans Linux, beaucoup de choses n'existeraient pas.



❓ Mais bon sang c'est QUOI Linux ?

C'est la question à laquelle ce premier chapitre va répondre, pas de panique 😊

Nous n'allons rien installer pour le moment, juste découvrir ce que c'est et comment on en est arrivé là... parce que c'est vraiment important ! Alors installez-vous confortablement et commençons par le commencement 😊

Un système d'exploitation

Est-ce que vous avez déjà entendu parler de **Windows** ? (*non je vous prends pas pour des idiots, je commence à zéro !*)

Aujourd'hui, la quasi-totalité des PC (si ce n'est plus !) est "livrée avec Windows". Mais savez-vous ce que ça signifie ?

❓ Oui moi je sais ! Ça signifie qu'on voit marqué "Windows" au démarrage de l'ordinateur ! Non c'est pas ça ? 🤪

Oui allez, disons que c'est un bon début 😊

En effet, une des premières choses que vous voyez lorsque vous allumez votre ordinateur, c'est un écran comme celui-ci :



L'écran de démarrage de Windows 7

Cet écran peut changer en fonction des versions de Windows, mais l'idée est là et vous venez de dire le mot-clé : Windows se lance **au démarrage** de l'ordinateur.

Le boot : démarrage de l'ordinateur

En fait, Windows se lance *presque* en premier. Si vous regardez bien, les toutes premières secondes vous avez autre chose qui s'affiche à l'écran. Cette "autre chose", c'est ce qu'on appelle **l'écran de boot**. Je ne vais pas vous faire de capture d'écran comme pour Windows car cet écran de boot varie beaucoup selon les ordinateurs.

Pourquoi ? Parce qu'il dépend du matériel dont est constitué votre ordinateur. C'est en effet la **carte mère** qui affiche l'écran de boot. La carte mère est le composant fondamental de tout ordinateur, c'est elle qui fait travailler le processeur, les disques durs, le lecteur de CD-Rom etc.

On a donc dans l'ordre :

1. Ecran de boot
2. Démarrage de Windows

Et c'est seulement une fois que Windows est chargé que vous pouvez enfin utiliser vos programmes : jeux, internet, logiciels de dessin, d'e-mail, de musique...



Mais pourquoi faut-il que Windows se charge d'abord ? Pourquoi on ne pourrait pas lancer des jeux dès le démarrage de l'ordinateur ?

Parce que... votre ordinateur a besoin d'une sorte de "super-logiciel" qui soit le chef d'orchestre. C'est lui qui doit gérer la mémoire de votre ordinateur, la répartir entre tous les programmes. Il fait le lien entre votre matériel (carte graphique, mémoire, imprimante) et vos logiciels. Et c'est un sacré boulot, croyez-moi ! 😊

Ce "super-logiciel", on l'appelle le **système d'exploitation**. Windows est donc un système d'exploitation.

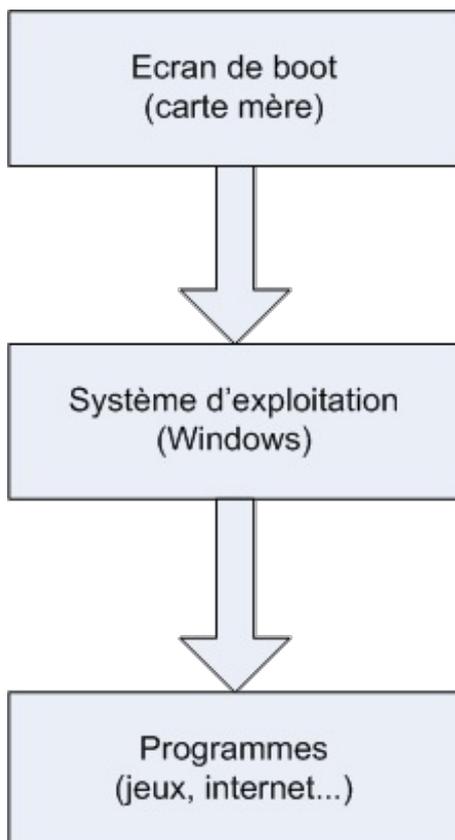


Un système d'exploitation se dit "Operating System" en anglais, que l'on abrège "OS". J'utiliserai souvent cette abréviation par la suite, ne soyez donc pas surpris !

Si on résume l'ordre des choses, on a donc ça au final :

1. Ecran de boot
2. Démarrage du système d'exploitation (Windows)
3. Lancement des programmes (jeux, internet, e-mail...)

Ou, si vous préférez les schémas :



(habituez-vous à mes schémas parce que je risque d'en utiliser pas mal par la suite 😊)

Linux est un système d'exploitation

Et Linux dans tout ça ?

Rassurez-vous je ne l'ai pas oublié ! Maintenant que vous savez un peu mieux ce qu'est un système d'exploitation (OS), je peux vous dévoiler la vérité : **Linux est un système d'exploitation**, au même titre que Windows ou encore Mac OS (pour ceux qui ont un Macintosh 😊).

Il est réputé entre autres pour sa sécurité et pour ses mises à jour plus fréquentes que Windows. Mais tout ça, vous allez le découvrir petit à petit.

Ce qu'il faut retenir pour le moment, c'est le principe de base de Linux : c'est *vous* qui contrôlez votre ordinateur. Ce n'est donc pas par hasard si ce cours s'appelle "**Reprenez le contrôle avec Linux !**". Vous allez enfin comprendre ce que vous faites, et donc mieux comprendre comment l'informatique fonctionne !



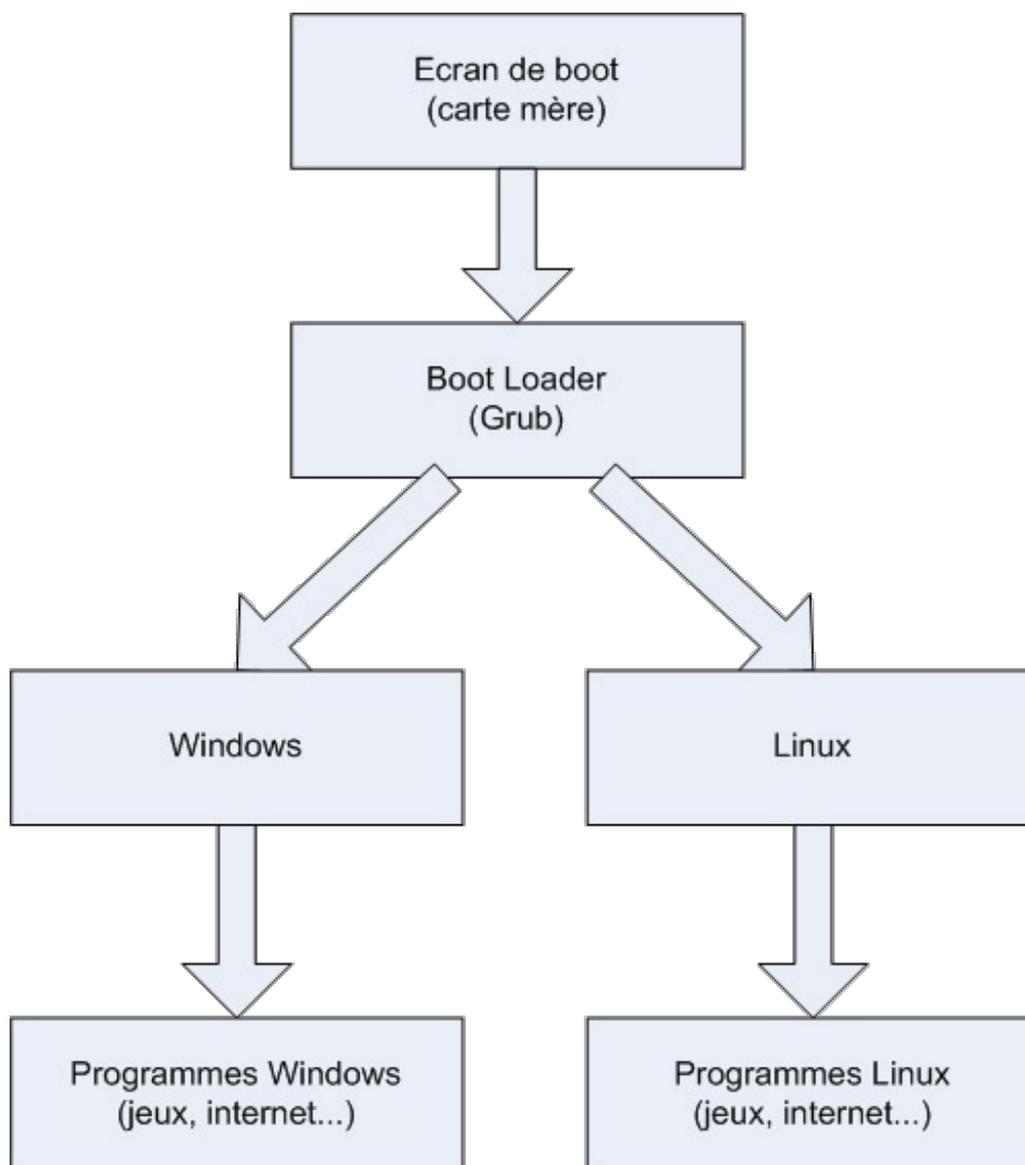
Peut-on avoir 2 OS installés sur son ordinateur en même temps ? Je n'ai pas envie de supprimer Windows pour mettre Linux à la place !

Beaucoup de gens croient qu'il faut faire un choix : Linux OU Windows. Rien n'est plus faux : vous pouvez très bien avoir 2 OS (ou plus !) installés à la fois sur votre ordinateur.

Dans ce cas, juste après l'écran de boot, vous aurez un programme appelé **Boot Loader** qui s'affichera pour que vous puissiez faire votre choix. Le boot loader dont nous parlerons ici porte le doux nom de Grub.

Grub vous proposera de choisir entre Windows et Linux à chaque démarrage de l'ordinateur. Il lancera l'OS par défaut si vous ne faites pas de choix avant quelques secondes. C'est vous qui choisissez l'OS par défaut bien entendu, on ne vous impose rien 😊

Le schéma du démarrage de l'ordinateur change donc un peu, voici le nouveau :



Comme vous pouvez le voir déjà sur ce schéma, lorsque vous êtes sous Linux vous utilisez des programmes faits pour Linux, et non pas les programmes de Windows. En effet, les programmes Windows ne fonctionnent pas sous Linux, et inversement.



Sachez quand même qu'il est possible de faire tourner des programmes Windows depuis Linux à l'aide d'un programme appelé wine. Toutefois, même si ça marche bien la plupart du temps, il est largement préférable d'utiliser des programmes faits pour Linux une fois dessus car ils tournent plus vite.

Vous devez déjà vous dire "Aïe, si je peux plus utiliser les programmes de Windows auxquels je suis habitué, je vais pas m'en sortir". Pourtant, il ne faut pas longtemps pour s'adapter (2-3 jours suffisent, voire même une soirée si vous êtes curieux !). Et les programmes sous Linux ont d'énormes avantages :

- Ils sont **gratuits** : vous verrez que sous Linux, la quasi-totalité des programmes sont gratuits.
- Les logiciels sont **mis à jour souvent**, et toujours gratuitement ! Vous verrez donc souvent vos logiciels préférés évoluer, et vous n'aurez pas à payer 300 euros pour vous mettre à jour !
- Certains de ces logiciels sont **meilleurs** que ceux que l'on trouve sous Windows. D'ailleurs, certains logiciels n'existent même pas sous Windows ! Vous découvrirez de nouvelles fonctionnalités et vous finirez par gagner du temps tout en utilisant plus efficacement votre ordinateur.



Mais pourquoi les programmes sont-ils gratuits ? Ce n'est pas complètement suicidaire financièrement ?

C'est justement ce que je vais vous expliquer maintenant 😊

La naissance de Linux

Voyons les choses en face :

- **Windows** coûte environ 200-300 euros
- **Linux** est gratuit, soit 0 euros TTC

On ne peut pas s'empêcher de se dire : "*Mais si c'est gratuit, c'est que ça doit être quelque chose de vite fait et de moins bonne qualité !*"

Grossière erreur 😬

Si Linux est gratuit (ainsi que quasiment tous ses logiciels), il y a des raisons. Pour comprendre, il faut remonter à 1984.

L'informatique en 1984

Nous sommes donc en 1984. A cette époque, l'informatique n'est pas très développée. Microsoft vient de sortir son premier OS : **MS-DOS**, mais il est encore loin d'être abouti.



Si vous avez utilisé les premières versions de Windows, vous avez forcément entendu parler de MS-DOS. En effet, toutes les versions de Windows 95 à Windows Me étaient basées sur MS-DOS (MS-DOS se lançait au démarrage juste avant Windows). Ce n'est que depuis Windows XP que MS-DOS a complètement disparu. Il n'est aujourd'hui plus développé.

```

INTERLAK EXE 17197 11-17-94 1:00p
XDFCOPY EXE 31737 11-17-94 1:00p
JOIN EXE 10279 11-17-94 1:00p
PKUNZIP EXE 29378 4-03-95 4:09p
DRVLOCK EXE 6501 11-17-94 1:00p
FIND EXE 5814 11-17-94 1:00p
RAMSETUP EXE 89649 11-17-94 1:00p
POWER EXE 8806 11-17-94 1:00p
ACALC EXE 22851 11-17-94 1:00p
NLSFUNC EXE 5609 11-17-94 1:00p
MEM EXE 16231 11-17-94 1:00p
APPEND EXE 7735 11-17-94 1:00p
SMARTDRV EXE 44121 11-17-94 12:00p
ZIP EXE 125964 9-13-93 3:36a
ZIPNOTE EXE 22942 9-07-93 8:42a
UNZIPSPFX EXE 26331 10-09-95 7:59p
UNZIP EXE 166332 10-09-95 7:59p
REXXDUMP EXE 968 11-17-94 12:00p
CPSCHED EXE 4946 11-17-94 1:00p
IBMAVSP EXE 158977 11-17-94 12:00p
RAMBOOST EXE 164272 11-17-94 1:00p
59 file(s) 2980199 bytes used
113414144 bytes free
C:\DOS>

```

MS-DOS, l'ancêtre de Windows



Mais MS-DOS était-il le seul OS qui existait à l'époque ?

Non ! Il y en avait d'autres, mais bien moins connus du grand public.

Celui qui était considéré comme le meilleur s'appelait "**Unix**". Il était beaucoup plus puissant que MS-DOS et aussi plus compliqué à utiliser, ce qui explique pourquoi seuls les informaticiens professionnels l'utilisaient.

Il est aussi beaucoup plus ancien : ses origines remontent à 1969 !

Graphiquement, Unix ressemblait beaucoup à MS-DOS : du texte blanc sur un fond noir. Il faut dire qu'à l'époque les ordinateurs n'étaient pas vraiment capables de faire mieux 😊

Le projet GNU



*Le gnou,
emblème de GNU*

C'est justement à cette époque, en 1984, que Richard Stallman créa le projet GNU. Richard Stallman était alors chercheur en intelligence artificielle au MIT. Il voulait créer un nouveau système d'exploitation fonctionnant comme Unix (les commandes restant les mêmes).



*Richard Stallman, fondateur du projet GNU
(non, tous les programmeurs ne sont pas barbus !)*

Pourquoi vouloir créer une "copie" d'Unix ?

Parce qu'Unix était payant et devenait de plus en plus cher ! Richard Stallman a voulu réagir en proposant une alternative gratuite : le projet GNU était né 😊



Bon à savoir : Mac OS X est lui aussi basé sur Unix. En revanche, MS-DOS et Windows sont complètement à part.

GNU est un système d'exploitation libre

GNU ne devait pas seulement être un OS gratuit. Il devait être aussi un OS "libre".



Quelle différence ?

Un programme **libre** est un programme dont on peut avoir le code source, c'est-à-dire la "recette de fabrication".

Au contraire, Windows est un OS **propriétaire** dont le code source est jalousement gardé par Microsoft. Imaginez que c'est un peu comme le Coca-Cola : personne ne connaît la recette de fabrication (y'a bien des gens qui essaient d'imiter, mais bon 😊).

On ne peut donc pas le modifier ou regarder comment il fonctionne à l'intérieur.

Un programme libre est donc la plupart du temps un programme gratuit. Mais c'est aussi un programme qu'on a le droit de copier, modifier, redistribuer.

C'est une véritable idéologie en informatique : ces gens pensent qu'il vaut mieux donner le code source des programmes que l'on fait car cela permet le partage des connaissances et que ça aide l'informatique à évoluer plus vite. Le slogan du monde du Libre pourrait être : "*L'union fait la force*".



On dit aussi souvent que le programme est "Open Source", car son code source est ouvert. Tout le monde peut le voir. Il existe quelques légères différences entre un programme "Open Source" et un programme "libre", mais nous



n'entrerons pas dans les détails ici (ce n'est vraiment pas intéressant).

Pendant ce temps, Linus Torvalds s'amusait

En 1991, **Linus Torvalds**, un étudiant de l'Université de Helsinki en Finlande, entreprend de créer son propre système d'exploitation sur son temps libre.

Ce système a pris le nom de Linux, en référence au nom de son créateur (Linux est la contraction de Linus et Unix).



Linus Torvalds, créateur de Linux

Quel rapport avec GNU ? Eh bien il se trouve que ces 2 projets étaient complémentaires : tandis que Richard Stallman créait les programmes de base (programme de copie de fichier, suppression de fichier, éditeur de texte), Linus s'était lancé dans la création du "coeur" d'un système d'exploitation (le *noyau*).

Le projet GNU (programmes libres) et Linux (noyau d'OS) ont fusionné pour créer **GNU/Linux**.

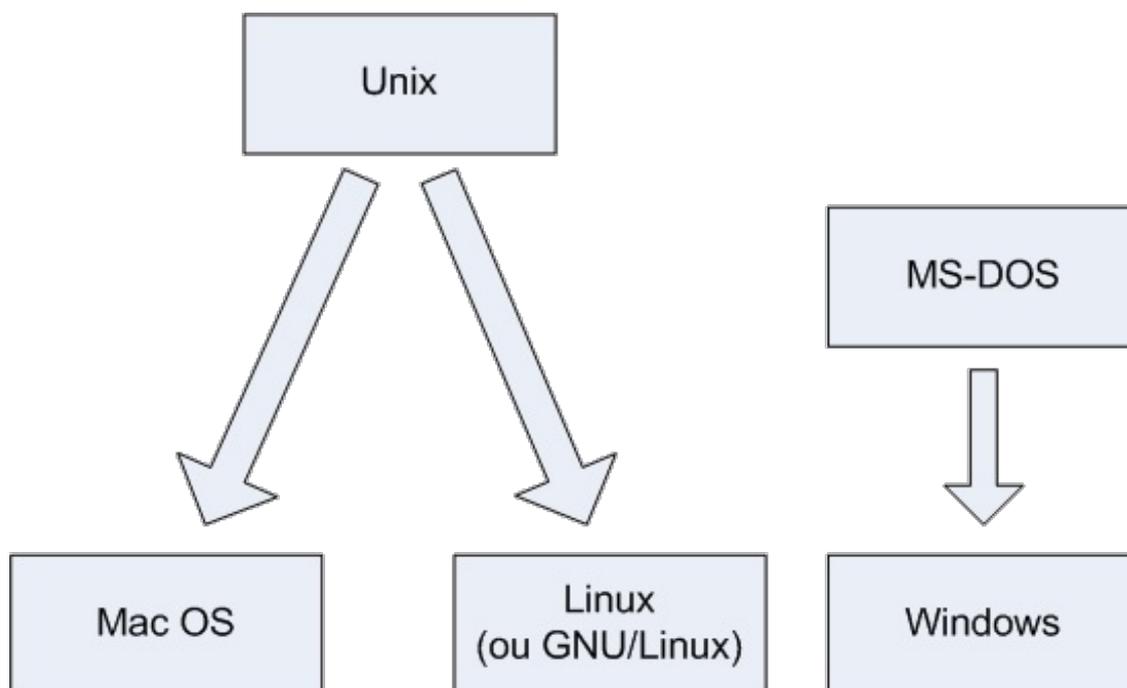


Théoriquement, on doit donc parler de GNU/Linux. C'est toutefois un peu difficile à écrire et prononcer, et par abus de langage on dit souvent juste "Linux". C'est donc pour cela que je continuerai à parler de "Linux" dans la suite du cours, même si le nom *politiquement correct* est "GNU/Linux" puisqu'il s'agit de la fusion de 2 projets complémentaires.

Résumons avec un schéma !

Ca va, vous êtes pas trop embrouillés ? 😊

Je pense qu'un petit schéma pour mettre de l'ordre dans les idées est indispensable ! S'il y a une chose que vous devez retenir, c'est ça 😊



Voilà, vous devriez maintenant avoir une meilleure idée de l'origine des 3 principaux systèmes d'exploitation qui existent aujourd'hui : Mac OS, Linux et Windows 😊

Ainsi, Mac OS et Linux sont tous les deux basés sur Unix, l'ancêtre des systèmes d'exploitation, tandis que Windows est une branche à part et est issu de MS-DOS. C'est tout ce que vous avez besoin de retenir en gros.

On dit que Mac OS et Linux sont basés sur Unix car ils ont "copié" son mode de fonctionnement. Ce n'est pas péjoratif bien au contraire, cela fait même honneur à Unix.



Les programmes Linux n'utilisent pas du tout le même code source qu'Unix (celui-ci était d'ailleurs propriétaire, donc privé). Ils ont été complètement réécrits mais fonctionnent de la même manière.

Si je vous ai raconté tout ça, c'est parce que j'estime que connaître l'**origine** de Linux est important. Cela vous permettra de comprendre bon nombre de choses par la suite.

Les distributions de Linux

Linux est un système d'exploitation très riche, vous allez le voir. On peut y trouver de nombreux logiciels différents et il existe des centaines de façons différentes de l'installer.

Pour simplifier la vie des utilisateurs et pour leur permettre de faire un choix, on a créé différentes *distributions de Linux*. C'est un concept qui n'existe pas vraiment sous Windows. C'est un peu comme la différence entre Windows XP Familial et Windows XP Professionnel, mais ça va bien plus loin que ça.

Voici ce qui peut différer d'une distribution à l'autre :

- L'installation (elle peut être très simplifiée comme très compliquée)
- La gestion de l'installation des programmes. Si elle est bien faite et centralisée, elle peut rendre l'installation de nouveaux logiciels plus simple que sous Windows comme nous le verrons plus loin !
- Les programmes préinstallés sur l'ordinateur (par exemple Windows est livré avec Internet Explorer et Windows Media Player)

En fait, une distribution c'est un peu l'**emballage** de Linux. Le cœur, lui, reste le même sur toutes les distributions.

Quelle que soit la distribution que vous installez, vous avez un Linux compatible avec les autres. Certaines distributions sont juste plus ou moins faciles à prendre en main 😊

Les différentes distributions existantes

Il existe un grand nombre de distributions Linux différentes. Dur de faire un choix vous allez me dire : en effet, quand on débarque la première fois on ne sait pas trop laquelle choisir... surtout que toutes sont gratuites ! Rassurez-vous, je vais vous aider à faire votre choix 😊

Je ne vais pas vous faire la liste de toutes les distributions qui existent, mais voici au moins les principales :

- **Slackware** : une des plus anciennes distributions de Linux. Elle existe toujours aujourd'hui !
- **Mandriva** : éditée par une entreprise française, elle se veut simple d'utilisation.
- **Red Hat** : éditée par une entreprise américaine "Red Hat", cette distribution est célèbre et très répandue, notamment sur les serveurs.
- **SuSE** : éditée par l'entreprise Novell.
- **Debian** : la seule distribution qui soit gérée par des développeurs indépendants au lieu d'une entreprise. C'est une des distributions les plus populaires.

Comme je vous l'ai dit, quelle que soit la distrib' (abréviation de distribution 🤖) que vous choisirez, vous aurez un Linux. Vous aurez "juste" grosso modo un fond d'écran différent au premier démarrage et des logiciels préinstallés différents (je simplifie un peu beaucoup, mais l'idée est là 😊).

La distribution Debian

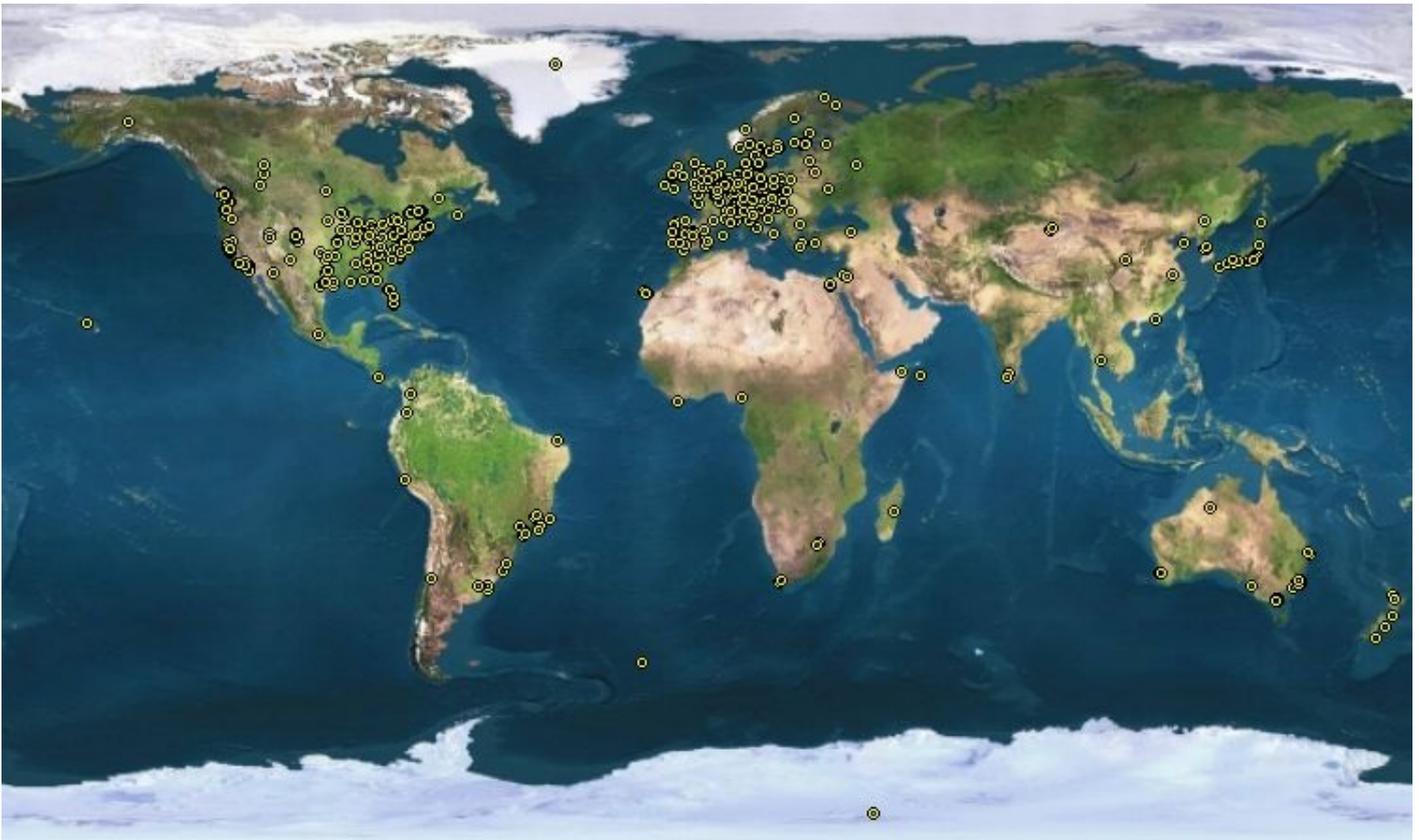
Nous, nous allons nous concentrer sur la distribution **Debian**.



Pourquoi Debian ? Ben déjà, parce qu'il faut bien faire un choix 🤖

Ensuite, parce que c'est la seule distribution qui soit gérée par des gens comme vous et moi (enfin assez doués en programmation quand même 🤖). Les autres distributions sont gérées par des entreprises, ce qui ne les empêche pas d'être Open Source et gratuites, même si on peut aussi les acheter pour avoir droit à une assistance (hotline...).

Debian est donc la seule distribution éditée par des particuliers bénévoles à travers le monde. Jetez un oeil à cette carte pour vous faire une idée :



*La carte des développeurs de Debian
Chaque point représente une personne participant à la création de Debian*

Un autre gros avantage de Debian, c'est le gestionnaire de paquets *apt-get*. C'est un programme qui gère tous les logiciels installés. Vous pouvez les désinstaller en un rien de temps. D'autre part, tous les logiciels sont centralisés en un même endroit, ce qui fait que vous n'avez pas à parcourir tout le Web pour retrouver un programme 😊

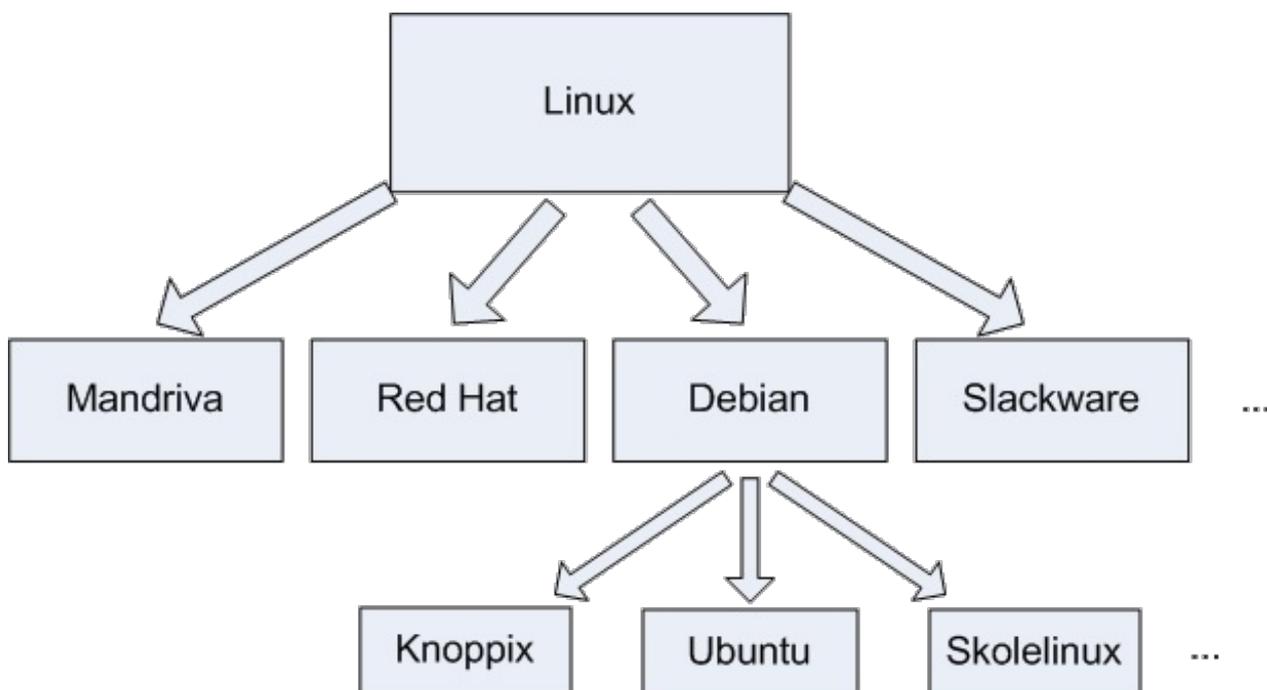
En fait, vous avez juste à indiquer le nom du logiciel que vous voulez et Debian ira le télécharger et l'installer pour vous. C'est extrêmement simple d'utilisation, je vous montrerai ça !

Debian a tellement de succès que de nombreuses distributions sont **basées sur Debian** :

- Knoppix
- Skolelinux
- Ubuntu
- ...

Ce sont donc des euh... distributions de distributions 🤔

Ok vous trouvez que c'est pas clair. Très bien très bien, je vais pas le prendre mal, je vais vous faire... un nouveau schéma ouaiiii 😊



Certaines distributions sont spécialisées. Par exemple, Skolelinux est faite pour être utilisée dans des écoles et est livrée avec de nombreux logiciels éducatifs (gratuits bien sûr !).

Quant à **Ubuntu**, c'est la distribution qui a créé la surprise. En peu de temps, elle est devenue très populaire. Pourquoi ? Il y a plusieurs raisons à cela :

- Elle est prévue pour le **grand public**, c'est-à-dire des gens comme vous et moi qui n'ont pas envie de se prendre la tête pour utiliser leur ordinateur. Le slogan est "Linux for human beings", ce qui signifie "Linux pour des êtres humains". Ça veut tout dire.
- **Les mises à jour sont fréquentes** : les développeurs travaillent d'arrache-pied sur Ubuntu et une nouvelle version sort tous les 6 mois environ, ce qui vous permet de disposer des dernières nouveautés.
- Il y a beaucoup d'utilisateurs, donc **beaucoup de gens pour vous aider** si vous avez des questions (un point à ne pas négliger !).



Logo de Ubuntu

C'est entre autres pour toutes ces raisons que nous allons utiliser Ubuntu dans la suite du cours. Nous verrons dans le prochain chapitre comment l'essayer sans l'installer, puis comment l'installer tout court si vous êtes décidé et conquis 😊

J'espère que vous avez appris pas mal de choses dans ce premier chapitre 😊

J'estime qu'il est vraiment important de savoir comment est né Linux et pourquoi il est gratuit. Maintenant que vous savez un peu ce que sont les distributions de Linux, vous vous sentirez moins perdu à l'avenir si vous en entendez parler 😊

Au fait, vous avez dû voir un manchot au début du chapitre, il n'est pas trop tard pour que je vous le présente :



Tux, la mascotte de Linux

Son nom est Tux, c'est la mascotte de Linux 😊

Il existe aussi en version bébé avec Baby Tux !



Baby Tux

Bon, le blabla est terminé, dès le prochain chapitre **on passe à l'action** ! Je prendrai pour exemple la distribution Ubuntu qui est, comme je vous l'ai dit, une très bonne distribution, idéale pour débiter 😊

Téléchargez Linux, c'est gratuit !

Le premier chapitre vous aura permis, je l'espère, de vous mettre un peu dans le bain du monde de Linux. Nous avons vu ce qu'est Linux, comment il est né et ce que sont les *distributions*.

Je vous ai dit en particulier que, sous Linux, on a énormément de choix. Il existe en effet de très nombreuses distributions qui proposent des versions différentes de Linux, qui sont fort heureusement toutes compatibles entre elles. Parmi toutes les distributions qui existent, j'ai choisi de vous présenter **Ubuntu** dans ce cours car c'est une distribution très populaire et facile à utiliser. En tant que débutant sous Linux, autant éviter de compliquer les choses de suite 😊

Maintenant, comme promis, on passe à la pratique dans ce chapitre. Nous allons dans un premier temps découvrir ce que sont les **gestionnaires de bureau** et choisir en conséquence la version de Ubuntu qui nous convient le mieux (eh oui, on va encore devoir faire un choix! 😊).

Les deux visages de Linux

A quoi ressemble Linux ?

Si vous vous êtes déjà posé cette question, vous avez peut-être pu observer de nombreuses captures d'écran, toutes très différentes les unes des autres. Il faut dire que Linux est très personnalisable, mais ça je crois que vous commencez à le comprendre à force que je le répète 😊

Si vous n'avez jamais vu de capture d'écran, ou si vous n'avez jamais vraiment fait attention, vous vous posez sûrement cette question :



Linux, c'est plus joli ou moins joli que Windows ?

Il n'y a pas de bonne réponse : ça peut être très beau comme très moche. Comme on peut très facilement changer l'apparence de son Linux, vous arriverez sans problème à trouver une apparence qui vous plaît. C'est un peu comme sous Windows XP, où, vous le savez peut-être, on peut changer l'apparence du système (le site <http://www.themexp.org> est d'ailleurs très célèbre). La différence, c'est que sous Linux la personnalisation va plus loin qu'un simple changement de couleurs. Nous allons voir ça un peu plus en détails.

Ce que je veux que vous sachiez ici, c'est que quelle que soit la distribution, il y a 2 façons d'utiliser Linux :

- En mode console (équivalent à DOS)
- En mode graphique (équivalent à Windows)

En mode console

Le mode "console" est un mode qui a tendance à faire peur aux petits nouveaux. Et pour cause, il n'a pas une tête franchement accueillante, jugez plutôt :

```

ubuntu@ubuntu:~/Desktop$ ls
Examples  ubiquity-kdeui.desktop
ubuntu@ubuntu:~/Desktop$ cd Examples
ubuntu@ubuntu:~/Desktop/Examples$ ls
book                logo-Kubuntu.png          oo-maxwell.odt
book-toc.html       logo-Ubuntu.png           oo-payment-schedule.ods
Experience ubuntu.ogg          oo-about-these-files.odt  oo-presenting-kubuntu.odp
fables_01_01_aesop.spx oo-about-ubuntu-ru.rtf    oo-presenting-ubuntu.odp
gimp-ubuntu-splash.xcf oo-access.odt              oo-trig.xls
kubuntu-leaflet.png  oo-cd-cover.odg           oo-welcome.odt
logo-Edubuntu.png    oo-derivatives.doc        ubuntu Sax.ogg
ubuntu@ubuntu:~/Desktop/Examples$ pwd
/home/ubuntu/Desktop/Examples
ubuntu@ubuntu:~/Desktop/Examples$ w
 22:44:02 up 15 min,  7 users,  load average: 0,07, 0,29, 0,26
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
ubuntu    tty1     -               22:30   0.00s  2.93s  0.02s w
ubuntu    tty2     -               22:30   15:25m 0.17s  0.14s -bash
ubuntu    tty3     -               22:30   15:25m 0.15s  0.12s -bash
ubuntu    tty4     -               22:30   15:25m 0.17s  0.14s -bash
ubuntu    tty5     -               22:30   15:25m 0.15s  0.13s -bash
ubuntu    tty6     -               22:30   15:25m 0.17s  0.15s -bash
ubuntu    :0       -               22:30   ?xdm?  50.06s 0.15s /bin/sh /usr/bi
ubuntu@ubuntu:~/Desktop/Examples$ _

```

Oui oui, c'est bien un Linux du XXIème siècle 🤖

En console, le fond est généralement noir (mais ce n'est pas une règle). Comme vous pouvez le voir sur cette capture, tout n'est pas qu'en noir et blanc : on profite aujourd'hui de nos écrans couleur pour coloriser certaines parties de la console afin qu'on puisse s'y repérer plus facilement.

Autre point important : en console, pas de souris. Tout se fait au clavier.



Quelle horreur ! Je vais être obligé d'utiliser ça ?

Pas du tout ! Comme je vous l'ai dit plus haut, il y a deux modes. La console n'est qu'un des deux "visages" de Linux, il y a aussi un mode graphique (encore heureux 😊).

Vous vous demandez à coup sûr ce que fait encore le mode console dans Linux. C'est vrai quoi, on a inventé des écrans plats gigantesques pouvant afficher des milliards de couleurs avec un contraste de 10000:1, ce n'est pas pour retomber à l'âge de pierre tout d'un coup !

Et pourtant... la console est un outil très puissant, pratiquement incontournable. Elle est toujours utilisée aujourd'hui par les Linuxiens, et vous en ferez bientôt partie.

Comment je peux être aussi sûr de moi ?... Parce que je compte bien vous en expliquer le fonctionnement ! 😊

(en partant de zéro bien sûr, c'est pas le Site du Zéro pour rien hein 🤖)

En mode graphique

Le mode graphique a une tête beaucoup plus accueillante pour quelqu'un qui débarque de Windows. En fait, ça ressemble un peu à Windows : il y a des fenêtres et on clique sur des croix pour fermer les fenêtres. Standard quoi 😊

Le truc... c'est qu'il y a plusieurs modes graphiques. Tous les modes graphiques sont basés sur un programme appelé **X** (voilà un nom court et facile à retenir 🤖). X est en fait la brique de base du mode graphique sous Linux.

Par-dessus X vient se greffer un programme appelé le **gestionnaire de bureau**. Le rôle du gestionnaire de bureau est de gérer les fenêtres, leur apparence, leurs options, etc.



Le concept de gestionnaire de bureau n'existe pas sous Windows. C'est nouveau pour vous.

Certes, sous Windows on peut changer l'apparence (le "skin"), mais ça s'arrête là. Le bureau reste le même, il y a toujours une barre des tâches avec le menu démarrer, toujours un Poste de Travail etc.

Sous Linux en revanche, d'un gestionnaire de bureau à l'autre le fonctionnement peut radicalement changer !

Présentation des principaux gestionnaires de bureau

Je ne vais pas vous faire la liste de tous les gestionnaires de bureau qui existent (pas fou), mais commencez déjà par retenir ces 3 là, qui sont probablement les plus célèbres :

- Gnome
- KDE
- XFCE

Gnome et KDE sont de "gros" gestionnaires de bureau, très riches en fonctionnalités. XFCE est une alternative plus légère qui peut fonctionner sur des PC plus anciens.

Si votre PC peut faire tourner Windows, il ne rencontrera aucun problème avec Gnome ou KDE. En revanche, s'il est un peu plus ancien, vous devrez peut-être vous pencher sur XFCE qui est moins gourmand en ressources.

Pourquoi je vous parle des gestionnaires de bureau ? Parce qu'il va falloir en choisir un ! Ubuntu est en effet déclinée en plusieurs versions, chacune basée sur un gestionnaire de bureau différent.

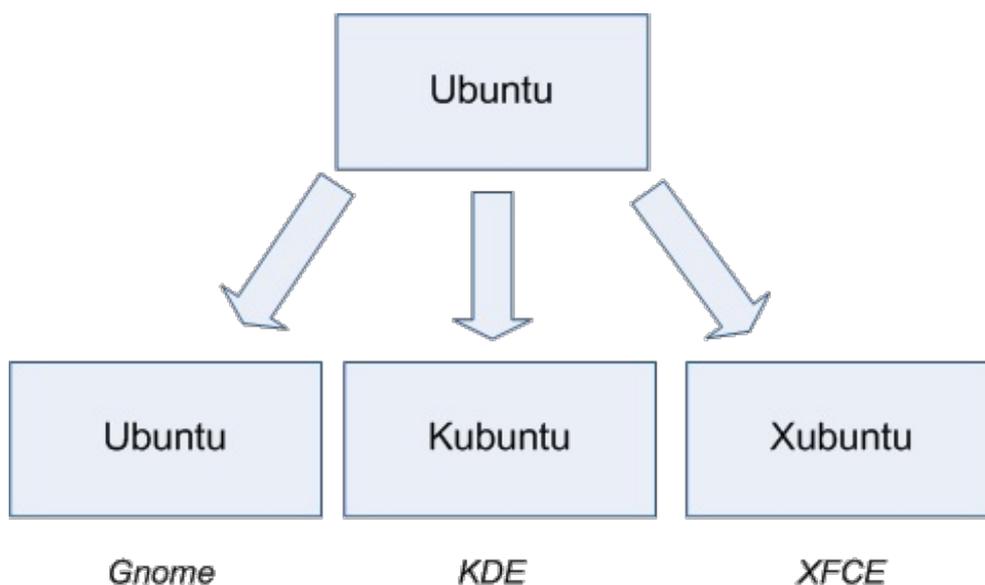
Voici une capture d'écran de chacun de ces gestionnaires de bureau :

Gestionnaire de bureau	Nom de la distrib' Ubuntu	Capture	Commentaires
Gnome	Ubuntu		<p>Gnome est un des plus célèbres gestionnaires de bureau de Linux. Le système est contrôlé par un menu présent en haut de l'écran, un peu à la manière de Mac OS (mais la ressemblance s'arrête là). Gnome est très complet et fait partie des gestionnaires de bureau les plus utilisés.</p> <p>A la base, Ubuntu ne proposait que Gnome, mais il a ensuite été décliné en plusieurs versions pour que chacun puisse choisir son gestionnaire de bureau favori.</p> <p>Si vous suivez mon cours de programmation en C / C++ en parallèle, sachez que Gnome est à la base de la librairie GTK+ permettant de créer des fenêtres dans des programmes.</p>
KDE	Kubuntu		<p>KDE est un autre poids lourd des gestionnaires de bureau. C'est un des plus anciens, et il a énormément évolué au fil des années. Très complet lui aussi, il possède une sorte de "barre des tâches" un peu comme sous Windows en bas de l'écran (le menu "K" étant ici l'équivalent du menu "Démarrer"). A cause de cette vague (j'ai bien dit vague !) ressemblance avec Windows, on a tendance à être plus attiré par KDE quand on débute sous Linux.</p> <p>Toujours pour ceux qui suivent mon cours de programmation en C / C++, il faut savoir que KDE est à la base de la librairie Qt de création de fenêtres.</p>
XFCE	Xubuntu		<p>XFCE est une alternative plus légère que Gnome et KDE. Il est donc, en toute logique, un peu moins pourvu en fonctionnalités. Ça ne veut pas dire qu'il est simplet, loin de là. Il se révèle très agréable à utiliser.</p> <p>Au niveau de l'apparence, il est proche de Gnome mais il peut aussi tout à fait ressembler à KDE. Il utilise GTK+, la même librairie de programmation de fenêtres que Gnome.</p>

La première version d'Ubuntu était basée sur Gnome. Le succès d'Ubuntu grandissant, les utilisateurs de KDE et de XFCE ont voulu eux aussi voir des versions d'Ubuntu basées sur leur gestionnaire de bureau favori. De là sont nées Kubuntu (basée sur

KDE) et Xubuntu (basée sur XFCE).

Tiens, ça fait longtemps que j'ai pas fait un schéma, ça me manque 😊



C'est tout ce que vous avez besoin de retenir pour le moment.

Ubuntu, Kubuntu et Xubuntu sont strictement identiques. Seul le gestionnaire de bureau installé par défaut change. Quand on parle d'Ubuntu, on fait donc généralement référence à toutes les versions d'Ubuntu à la fois.

i Le choix du gestionnaire de bureau n'est pas définitif. On peut sans problème avoir plusieurs gestionnaires de bureau installés à la fois (il vous faudra alors choisir au démarrage, lorsque l'on vous demande votre identifiant et votre mot de passe, le gestionnaire de bureau que vous voulez utiliser). Vous pourrez donc tester et installer d'autres gestionnaires de bureau par la suite.

Sachez d'ailleurs que certains gestionnaires de bureau moins répandus existent et qu'ils permettent d'avoir un bureau vraiment très différent de Windows, comme en témoigne ce [screenshot](#) (un bureau mis à jour en temps réel en fonction de la position de la Terre par rapport au Soleil !)

Enfin, une information importante à retenir : tous ces gestionnaires de bureau sont compatibles entre eux. Les programmes fonctionnent donc tous quel que soit le gestionnaire de bureau que vous utilisez 😊

Alors... Gnome, KDE ou XFCE ? Ubuntu, Kubuntu ou Xubuntu ?

A vous de choisir. Votre première expérience sous Linux sera différente selon que vous choisissiez Ubuntu, Kubuntu ou Xubuntu. Il sera toujours possible par la suite de changer de gestionnaire de bureau comme je vous l'ai dit précédemment, donc si vous faites une "erreur" ce n'est pas un drame 😊

De mon côté aussi, il va bien falloir que je fasse un choix. Et là, c'est délicat. Il y a des utilisateurs de Linux qui ne jurent que par Gnome, d'autres que par KDE... et d'autres que par XFCE. Je ne veux rien vous imposer. Pour ma part, je change assez souvent de gestionnaire de bureau. A l'heure où j'écris ces lignes je suis sous XFCE, mais juste avant j'étais sous Gnome, et encore avant j'utilisais KDE. Bref, ce n'est pas la peine d'en choisir un pour "faire comme moi" parce que je change trop souvent 😊

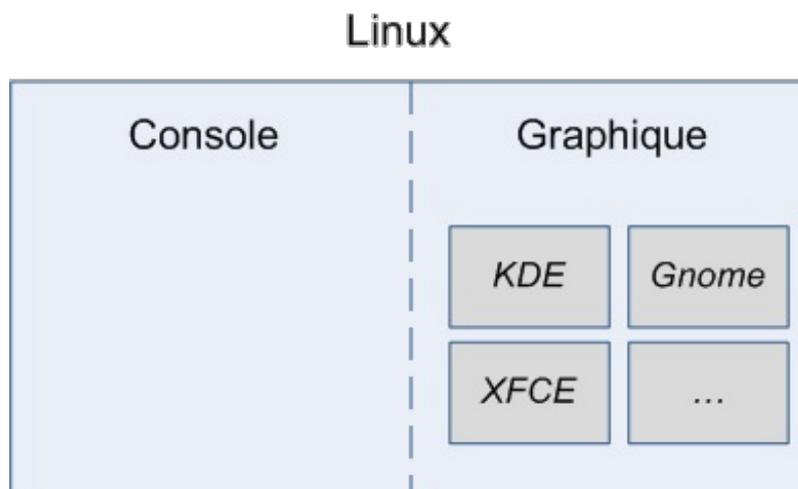
Dans la suite de ce tutoriel, je vais faire le choix de **Gnome** (donc Ubuntu). Il y a diverses raisons à cela, la première étant que Gnome est le gestionnaire de bureau par défaut d'Ubuntu. Les autres distributions (Kubuntu, Xubuntu...) ne sont que des déclinaisons. Vous avez aussi en pratique plus de chances de tomber sur quelqu'un qui utilise Ubuntu sous Gnome le jour où vous avez besoin d'aide.

i Ne vous focalisez pas trop sur le gestionnaire de bureau. Ce qui compte en fait, c'est que la console reste strictement identique que vous utilisiez Gnome ou KDE ou XFCE. En effet, la console est une "constante" : elle ne change pas d'un Linux à l'autre. Je pourrai donc dans la suite de ce cours vous expliquer le fonctionnement de la console, peu importe le

 gestionnaire de bureau que vous aurez choisi 😊

Schéma résumé à retenir

Allez, un petit schéma pour être sûr que ça rentre et on pourra passer à la suite 😊



Ce qu'il faut retenir donc, c'est que Linux peut être utilisé dans 2 modes différents : console ou graphique. Le fonctionnement de la console est le même d'un Linux à un autre, par contre l'aspect graphique peut radicalement changer selon le gestionnaire de bureau que l'on choisit : KDE, Gnome, XFCE, etc.

Télécharger et graver le CD

Intéressons-nous maintenant au concret : comment obtenir Ubuntu sur CD pour le tester et peut-être l'installer ?

Nous allons voir comment télécharger Ubuntu.

1/ Récupérer l'ISO

Linux se télécharge sous la forme d'un gros fichier .iso d'environ 700 Mo. Ce fichier correspond à l'image d'un CD, et vous permet donc de graver un CD complet de Linux.

La première étape consiste à récupérer le fichier ISO. Là, tout dépend si vous avez choisi Ubuntu, Kubuntu ou Xubuntu, car ce n'est pas le même ISO.

Rendez-vous sur une de ces pages en fonction de la version d'Ubuntu que vous désirez :

- [Télécharger Ubuntu](#)
- [Télécharger Kubuntu](#)
- [Télécharger Xubuntu](#)

Cliquez tout simplement sur le lien de téléchargement. Vous allez récupérer un fichier .iso.



Il vous faut au moins 192 Mo de mémoire vive pour pouvoir utiliser Ubuntu. Si votre ordinateur en possède moins, vous devrez cocher "Utiliser l'Alternate CD" qui est plus compliqué à installer et que je ne pourrai pas vous expliquer ici. Si vous êtes intéressé, vous pouvez lire [ce tutoriel sur l'Alternate CD rédigé par Renesis-3](#).

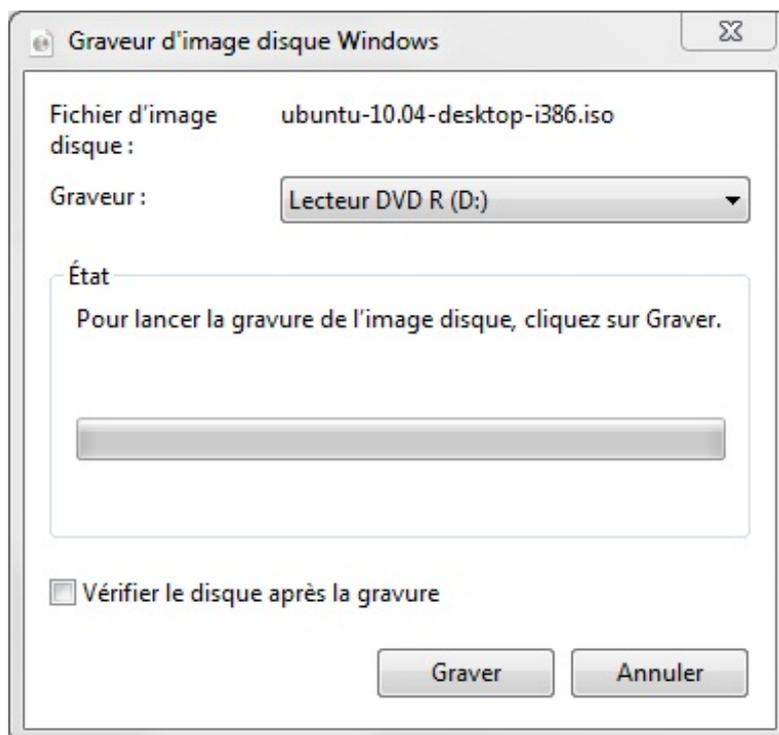
Eh oui, Linux a besoin d'un minimum de mémoire vive pour fonctionner. Beaucoup de gens croient à tort que Linux peut transformer votre vieil ordinateur en bête de course mais c'est faux : comme avec Windows, si vous voulez un beau système avec plein de fonctionnalités, il vous faut un minimum de puissance 😊

2/ Graver le CD

Il vous faut maintenant graver le gros fichier .iso que vous venez de télécharger.

Sous Windows 7

Si vous avez Windows 7, un outil de gravure d'images disque .iso est déjà inclus. Il vous suffit de double-cliquer sur le fichier .iso, ce qui aura pour effet d'ouvrir cette fenêtre :



Insérez un CD vierge dans votre graveur et cliquez tout simplement sur "Graver".

Sous d'anciennes versions de Windows

Il vous faut un logiciel de gravure pour graver le fichier ISO car les versions antérieures à Windows 7 ne savent pas graver des images disque.

Si vous avez déjà un programme comme Nero ou Easy CD Creator et que vous savez comment graver un ISO, c'est très bien. Sinon, je vais vous montrer comment faire à l'aide du logiciel de gravure gratuit **CDBurnerXPro**.

Tout d'abord, commencez par télécharger [CDBurnerXPro](#).

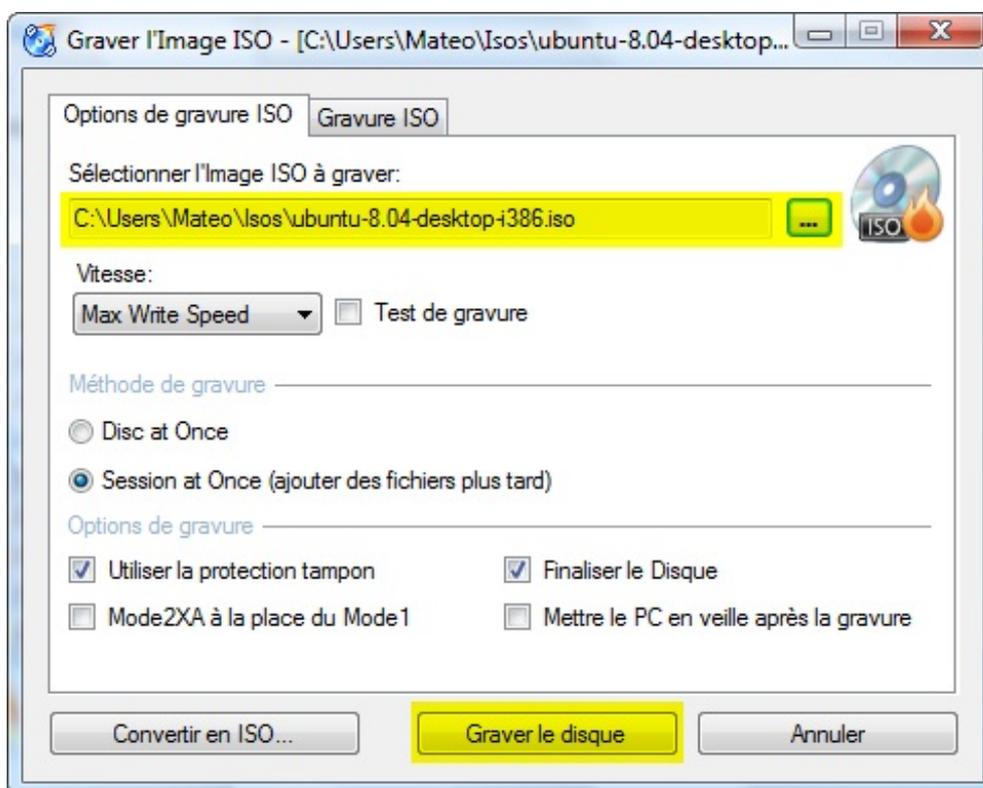
Le logiciel est en français 😊

Commencez par insérer un CD-R (CD vierge) dans votre graveur.

Lancez ensuite le logiciel CDBurnerXPro et, lors du démarrage, cliquez sur "Créer un CD/DVD de données" :



La fenêtre principale s'ouvre. Allez dans le menu Fichier / Graver le disque à partir du fichier ISO. Une nouvelle fenêtre s'ouvre alors :



Commencez par indiquer en haut où se trouve le fichier ISO que vous venez de télécharger. Vous pouvez graver à vitesse maximale, mais cela peut provoquer des erreurs parfois, comme une coupure pendant l'installation de Linux. Si vous êtes du genre prudent, je vous recommande de réduire la vitesse de gravure (vous pouvez mettre 2X ou même 1X).

Cliquez ensuite sur le bouton "Graver le disque", patientez quelques minutes, c'est prêt ! 😊

Bah voilà, c'était pas bien compliqué 😊

Vous avez maintenant un CD d'Ubuntu flambant neuf, gratuit, légal, qui n'attend que d'être essayé 😊

Nous verrons justement dans le prochain chapitre comment essayer Linux.

Nous arrivons au terme du second chapitre.

Si vous lisez ces lignes, c'est que vous avez un CD d'Ubuntu entre vos mains. Félicitations ! Nous allons pouvoir tester Linux dans le chapitre suivant !

Que faut-il retenir de ce chapitre ?

- **Linux peut être utilisé de deux manières différentes** : soit via une console, qu'on manipule uniquement au clavier, soit via une interface graphique avec des fenêtres comme sous Windows.
- **La console a l'air compliquée, mais c'est un outil puissant** dont vous ne pourrez plus vous passer une fois que je vous aurai appris à vous en servir.
- **Le mode graphique vous permet d'utiliser votre ordinateur aussi facilement que sous Windows**. Il est géré à la base par un programme appelé X. Par-dessus ce programme vient se greffer ce qu'on appelle un gestionnaire de bureau qui définit comment fonctionne votre mode graphique. Les gestionnaires de bureau les plus connus sont Gnome, KDE et XFCE, mais ce ne sont pas les seuls qui existent.
- **Ubuntu est décliné en 3 versions : Ubuntu (Gnome), Kubuntu (KDE), Xubuntu (XFCE)**.
- Vous pourrez installer plusieurs gestionnaires de bureau à la fois si vous le voulez.
- Quel que soit le gestionnaire de bureau choisi, tous les programmes fonctionneront sans problème de la même manière. **Il n'y a pas d'incompatibilité entre les gestionnaires de bureau.**

Tester et installer Ubuntu

Nous y voici enfin 😊

Dans ce chapitre, vous allez peut-être pour la première fois de votre vie voir à quoi ressemble Linux sur votre ordinateur et découvrir qu'en fait... bah c'est beaucoup plus simple qu'on ne le pensait 😊

Souvenez-vous de ce que je vous ai dit dans le chapitre précédent : Linux a deux visages. Il y a une partie console, et une partie graphique (avec Gnome, KDE ou XFCE). Si la partie console requiert un peu de travail pour savoir s'en servir, la partie graphique, elle, est très simple à utiliser. Je vous le dis tout à fait objectivement : si vous n'utilisez Linux qu'en mode graphique, c'est aussi simple que Windows voire même parfois plus simple !

Mais ce serait dommage de se limiter au mode graphique, car une grande partie de la puissance de Linux (qui justifie notamment son intérêt), c'est la console.

Bon trêve de blabla, où en étions-nous ?

Ah, tester Linux ! Vous saviez qu'on peut tester Linux sur son ordinateur sans rien installer sur son disque dur ? On va voir comment faire maintenant justement 😊

Puis, ensuite, je vous montrerai comment faire pour installer Linux sur votre disque dur si vous êtes décidé 😊



Si vous rencontrez le moindre problème pendant l'installation ou si vous vous posez une question, rendez-vous sur [les forums du site](#). Des personnes expérimentées en Linux vous y répondront 😊

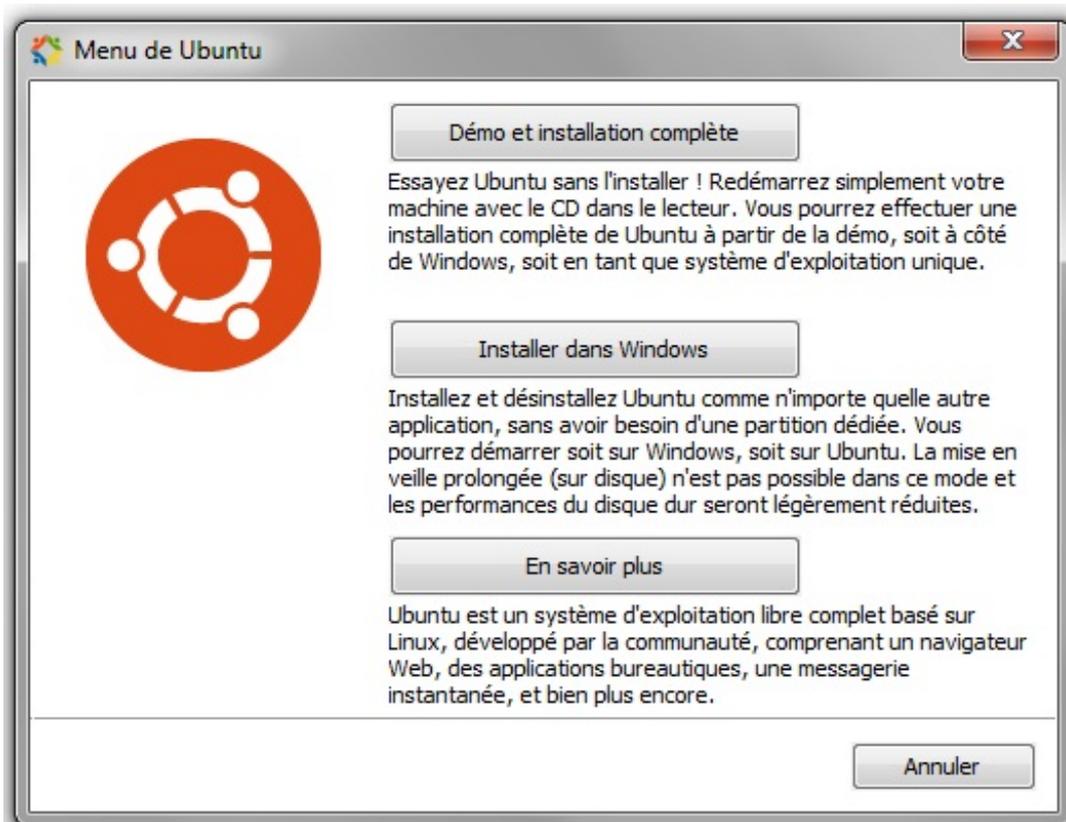
Installation de Linux depuis Windows

Depuis quelques temps, l'installation de Linux a fait des progrès étonnants pour faire en sorte d'être la plus simple possible.

Vous avez aujourd'hui 2 possibilités pour installer Linux :

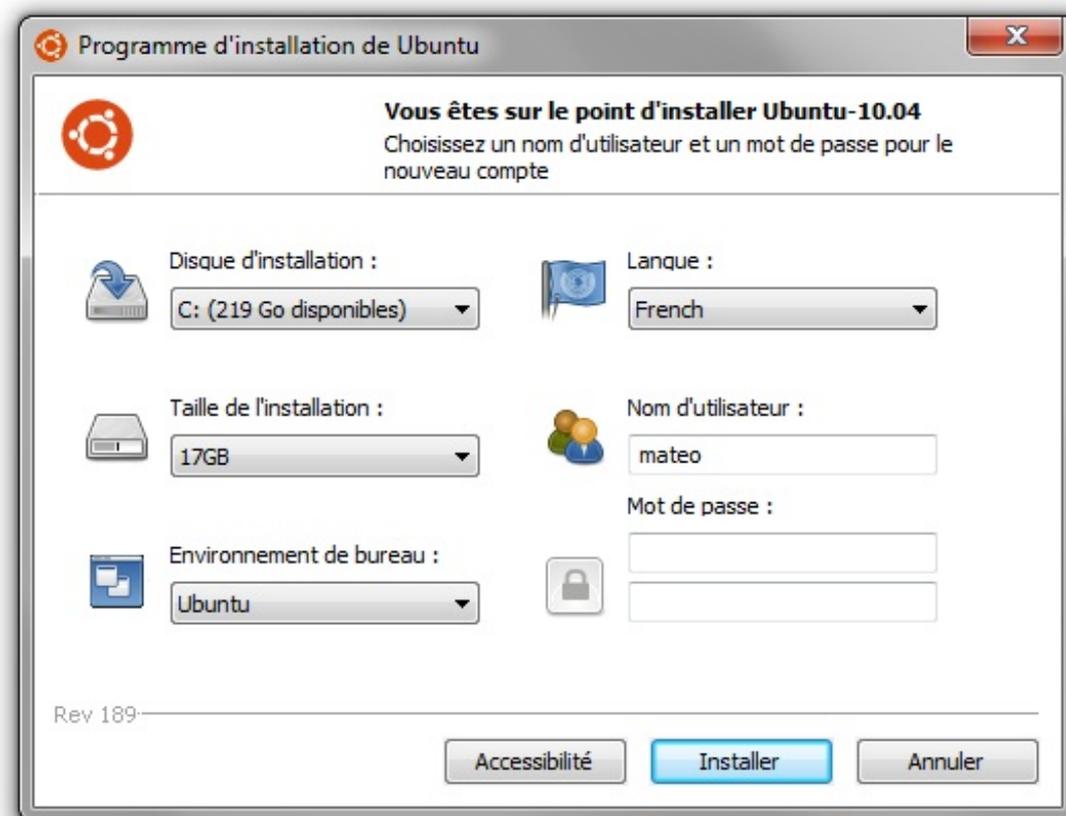
- Vous pouvez l'installer **depuis Windows** (c'est ce qu'on va voir ici), dans ce cas l'installation sera un peu particulière mais Linux fonctionnera parfaitement.
- Soit vous **redémarrez votre ordinateur** avec le CD de Linux dans votre lecteur, ce qui est la méthode la plus classique pour l'installer. On verra comment faire cela dans la suite de ce chapitre.

L'installation de Linux depuis Windows est une petite prouesse technologique qui vous apporte un certain nombre d'avantages. En effet, il suffit de lancer le CD depuis Windows pour voir apparaître cette fenêtre :



Si vous cliquez sur "Démon et installation complète", on vous invitera à redémarrer votre ordinateur pour installer Linux de manière "classique". Nous verrons cette méthode juste après.

Si vous cliquez en revanche sur "Installer dans Windows", vous pourrez installer Linux depuis Windows. La fenêtre suivante devrait alors apparaître :



Dans cette fenêtre, choisissez combien d'espace disque vous voulez réserver à Ubuntu (en Go). Choisissez aussi un nom d'utilisateur et un mot de passe, puis cliquez sur Installer 😊

L'installation se fait de manière classique depuis Windows. Une fois que ce sera fait, vous pourrez redémarrer votre ordinateur, et lancer Ubuntu (il faudra faire un choix au démarrage).

Lors du premier lancement, l'installation d'Ubuntu devra se compléter, puis ce sera bon, vous serez sous Linux 😊



Un autre gros avantage de cette méthode est que vous pourrez ensuite désinstaller Ubuntu le plus simplement du monde en allant dans... "*Ajout / Suppression de programmes*" du panneau de configuration de Windows !

Cette méthode a toutefois quelques défauts. Ubuntu sera un peu moins performant (car il sera installé *dans* Windows) et nécessitera plus de mémoire vive (512 Mo).

Dans la mesure du possible, je vous conseille d'installer Ubuntu en utilisant la "vraie" méthode classique que nous allons voir ci-dessous.

Premier démarrage d'Ubuntu



Je vous conseille d'imprimer ces informations pour pouvoir continuer à lire ce tuto lorsque vous lancerez Linux. En effet, la méthode d'installation "classique" que nous allons voir maintenant nécessite de redémarrer l'ordinateur, si vous n'avez pas imprimé ce tutoriel vous ne pourrez pas vous en aider lors de l'installation de Linux !

Je suppose que pour le moment vous êtes sous votre système d'exploitation habituel, c'est-à-dire Windows (ou Mac OS). Je vais vous demander de mettre le CD d'Ubuntu dans votre lecteur CD... lààà voilà très bien 😊

Maintenant, redémarrez votre ordinateur.

Vous allez voir Windows s'éteindre, puis l'ordinateur va redémarrer. Cette fois, il devrait afficher l'écran de chargement d'Ubuntu :



Si vous voyez cela, c'est très bien ! Cela signifie que votre ordinateur a démarré sur votre CD qui contient Linux au lieu de démarrer sur le disque dur sur lequel est installé Windows.



Euh, moi j'ai mis le CD dans le lecteur, j'ai redémarré, et pourtant ça a lancé Windows quand même !
Je dois jeter mon PC par la fenêtre ?

La plupart des PC démarrent maintenant sur le CD s'ils en trouvent un dans le lecteur, mais certains ordinateurs doivent être configurés pour démarrer à partir du CD. Pas de chance pour vous 😊

Heureusement, je vais vous expliquer comment faire pour changer cela 😊



Si vous n'avez pas eu de problème et que vous avez vu l'écran d'accueil d'Ubuntu dès le début, vous pouvez directement sauter à l'étape suivante.

Modifier l'ordre de boot

Si vous devez modifier l'ordre de boot pour que votre ordinateur lise le CD, redémarrez. Pendant l'écran de boot (la toute première chose que vous voyez à l'écran), pressez la touche indiquée pour accéder au Setup, aussi appelé BIOS (c'est l'écran de configuration de votre carte mère).

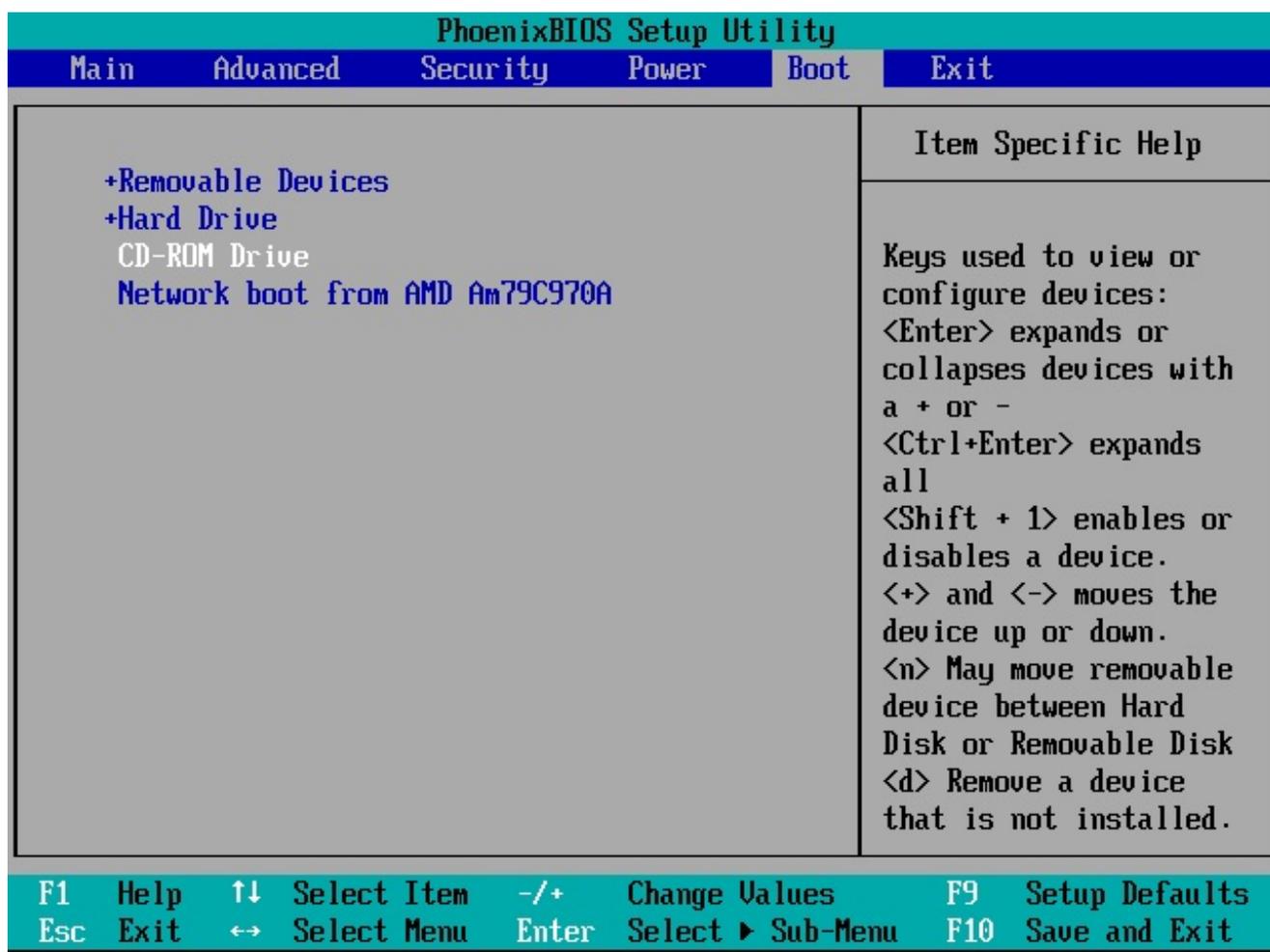
Généralement, la touche est F2 ou Suppr, mais cela peut varier selon la carte mère que vous avez.

Vous devriez alors voir le superbe menu du BIOS (sigh !). D'un ordinateur à l'autre, cet écran peut être légèrement différent.

PhoenixBIOS Setup Utility						
Main	Advanced	Security	Power	Boot	Exit	
System Time: [01]:16:12					Item Specific Help	
System Date: [10/09/2006]					<Tab>, <Shift-Tab>, or <Enter> selects field.	
Legacy Diskette A: [1.44/1.25 MB 3½"]						
Legacy Diskette B: [Disabled]						
▶ Primary Master [None]						
▶ Primary Slave [None]						
▶ Secondary Master [VMware Virtual ID]						
▶ Secondary Slave [None]						
▶ Keyboard Features						
System Memory: 640 KB						
Extended Memory: 523264 KB						
Boot-time Diagnostic Screen: [Disabled]						
F1	Help	↑↓	Select Item	-/+	Change Values	F9 Setup Defaults
Esc	Exit	↔	Select Menu	Enter	Select ▶ Sub-Menu	F10 Save and Exit

On a connu plus accueillant 😊

Repérez le menu "Boot". Il faudra généralement vous déplacer à l'aide des flèches du clavier. Le menu Boot peut être différent sur votre ordinateur, voici ce que ça donne sur le mien :



Ici, on peut définir l'ordre dans lequel l'ordinateur essaie de démarrer les éléments. A vous de changer cet ordre pour faire en sorte que votre ordinateur essaie de démarrer sur le CD *avant* de démarrer sur le disque dur.

Lisez les instructions sur le côté (certes, en anglais), pour savoir comment faire sur votre ordinateur.

Si vraiment vous êtes bloqué, n'hésitez pas à aller demander de l'aide sur les [forums](#) du site.

Essayer ou installer Ubuntu

Bien, à partir de maintenant je suppose qu'Ubuntu se lance au démarrage de l'ordinateur. Après quelques instants de chargement, vous devriez voir l'écran suivant :



Sélectionnez la langue dans le menu de gauche si les textes ne sont pas en français.

Vous voyez que vous avez 2 choix :

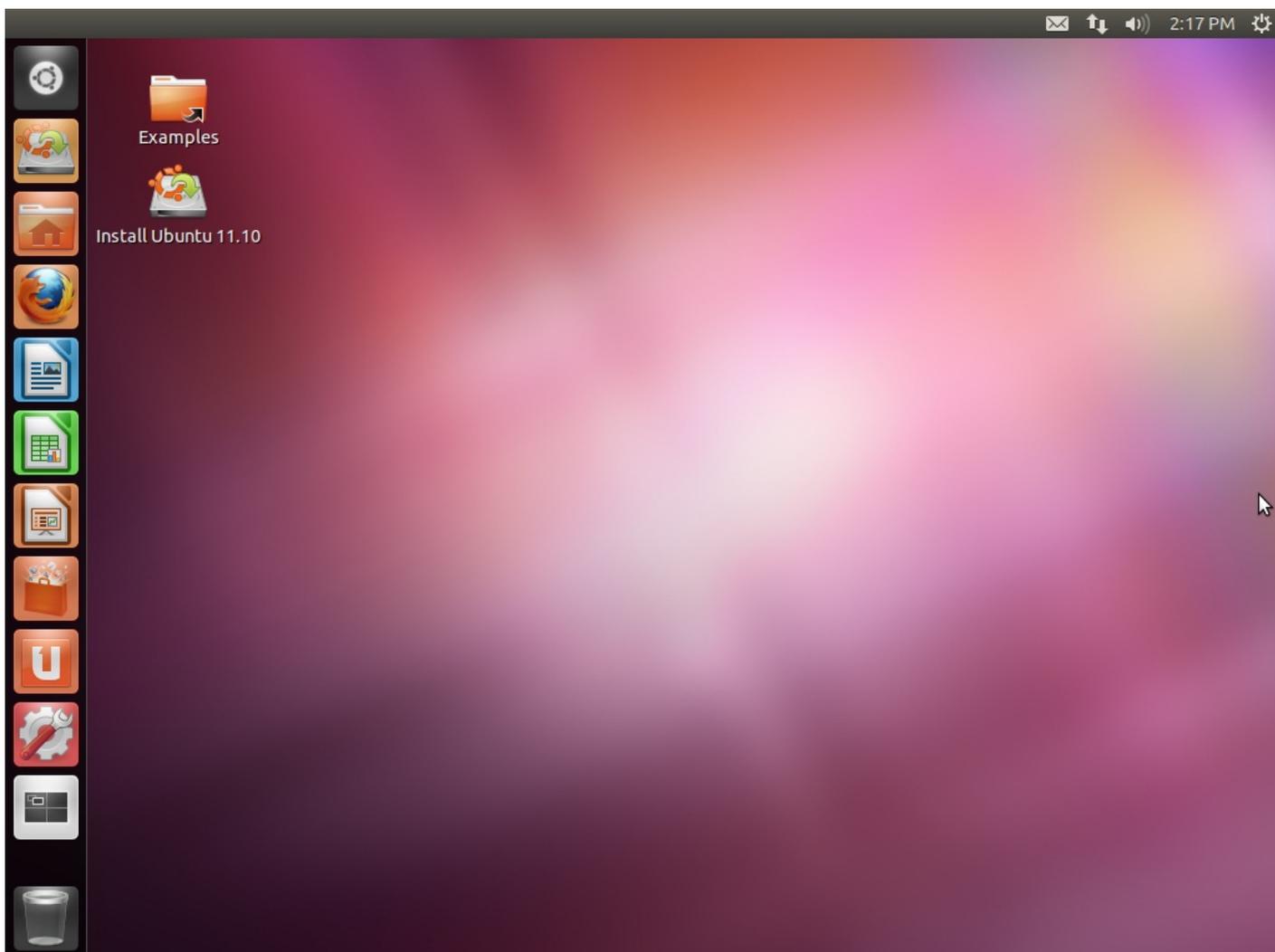
- **Essayer Ubuntu** : Ubuntu sera lancé sans toucher à votre disque dur. Vous pourrez donc l'essayer pour le tester.
- **Installer Ubuntu** : Ubuntu sera installé sur votre disque dur. Utilisez ce choix si vous êtes déjà certain de vouloir installer Ubuntu.

Dans la suite de ce tutoriel, je vais supposer que vous avez fait le premier choix (celui que tout débutant serait tenté de faire !).

Ubuntu va alors se charger à partir du CD dans la mémoire vive. En effet, dans ce mode de test appelé "Live CD", Linux ne touche pas à votre disque dur. Aucun risque d'altérer Windows, tout est fait dans la mémoire vive (et entre nous, c'est une sacrée prouesse technique !).

Une fois le chargement effectué, vous allez rapidement vous retrouver dans le gestionnaire de bureau que vous avez choisi. Si, comme moi, vous avez choisi Ubuntu, vous serez donc sous Gnome.

Voici à quoi ressemble le bureau de Gnome (il peut légèrement varier en fonction de votre version d'Ubuntu) :



Le bureau Gnome, une fois démarré

Alors, qu'en dites-vous ? C'était pas franchement la mer à boire. 😊

Retenez bien : tout ce que vous voyez là a été chargé dans votre mémoire vive. Ubuntu n'a pas touché à votre disque dur. Vous pouvez donc tester Linux en toute sécurité. N'hésitez pas à parcourir les menus et à vous familiariser un peu avec l'environnement de bureau que vous avez choisi.



Je détaillerai le fonctionnement des gestionnaires de bureau **KDE** et **Gnome** dans les prochains chapitres. Pour le moment, je vous laisse le soin de découvrir un peu par vous-même, j'estime que c'est important 😊

Installer Ubuntu

Alors, vous avez fait un petit tour dans les programmes fournis dans Ubuntu ?

Sachez que selon la version que vous avez prise (Ubuntu, Kubuntu ou Xubuntu), on retrouve des programmes différents installés par défaut.

En effet, certains programmes sont destinés à la base pour KDE, mais ils fonctionnent aussi sans problème sur Gnome. Inversement : certains programmes sont destinés à la base pour Gnome, mais on peut les utiliser sans souci sous KDE (c'est le cas de Firefox par exemple). Il n'y a donc pas d'incompatibilité entre les gestionnaires de bureau, mais on préfère généralement installer au départ les programmes prévus pour KDE sur KDE, histoire d'être... logique 😊

Allez, je vous sens chauds pour une petite installation là 😊

On y va ? 😊

Attention : même si l'installation est très détaillée et sécurisée, il y a toujours un petit "risque" que vous installiez par-



dessus Windows. Dans tous les cas, faites une sauvegarde de vos fichiers les plus importants avant de commencer l'installation, sur CD ou clé usb par exemple.

Pas de panique hein, tout va bien se passer, mais deux protections valent mieux qu'une en général 😊

Etape 1 : lancer l'installation et choisir la langue

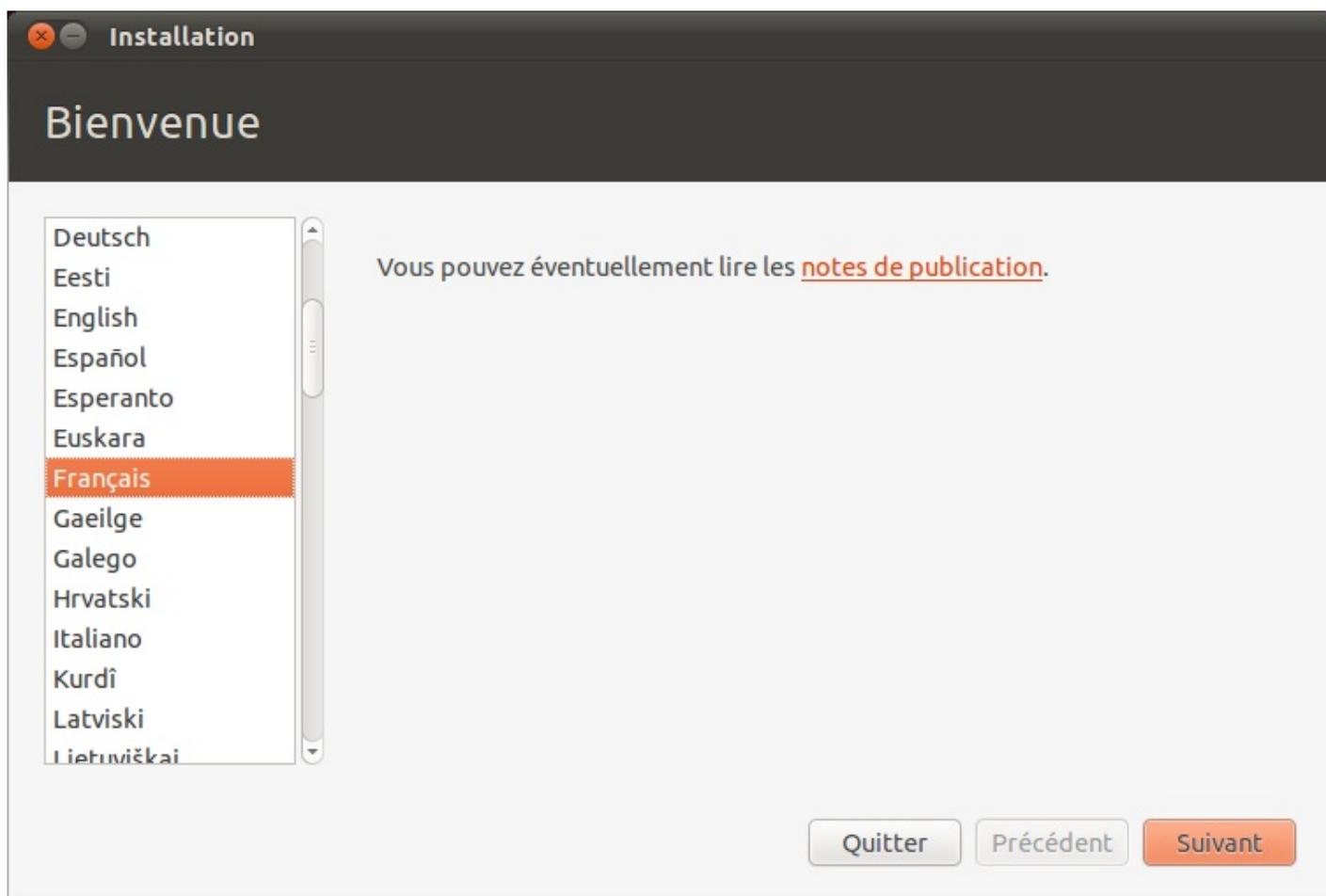
Vous allez voir, contrairement à ce que vous pensiez il y a quelques minutes à peine, installer Linux est d'une simplicité... frustrante 😞

Il faut d'abord ouvrir le programme d'installation qui se trouve sur le bureau en cliquant sur cette icône :



Mes captures d'écran sont faites sur la version 11.10 d'Ubuntu. Il est possible que vous ayez téléchargé une version plus récente, mais rassurez-vous : malgré quelques légères différences, le principe de l'installation reste le même.

La première fenêtre de l'assistant s'ouvre. Vérifiez que "Français" est bien sélectionné :

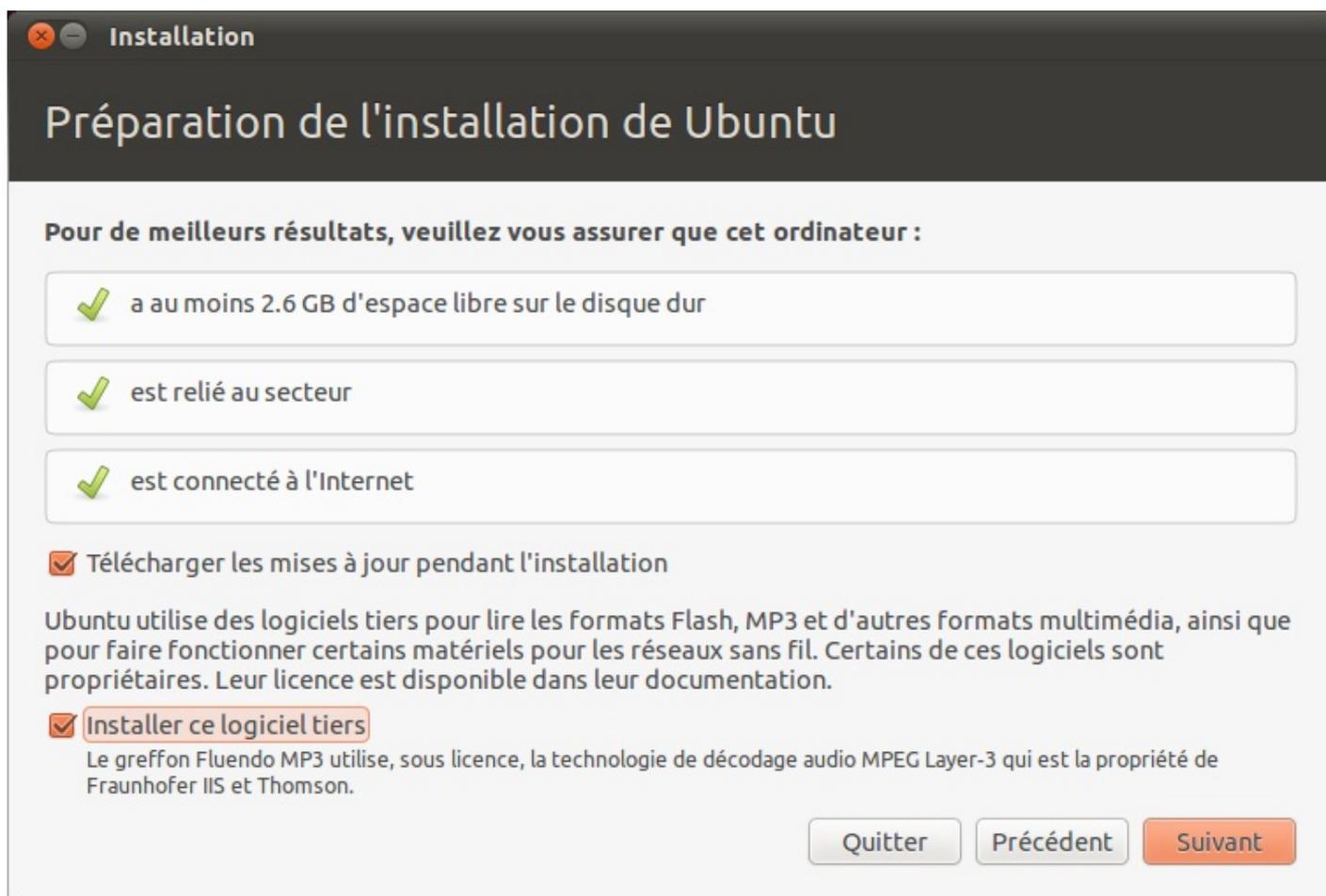


Ceux qui veulent un Linux en japonais, c'est le moment !

Cliquez ensuite sur "Suivant" pour passer à l'étape n°2.

Etape 2 : préparation de l'installation

On vous demande de vous préparer à l'installation d'Ubuntu sur votre ordinateur :



3 conditions sont posées :

- **Avoir un minimum d'espace disque disponible** : normal, si vous voulez avoir la place d'installer Ubuntu ! Ici, il s'agit du strict minimum, je vous conseille d'avoir 8-10 Go pour être suffisamment confortable. A priori, pas besoin de plus d'espace supplémentaire, sauf si vous prévoyez d'y stocker de gros fichiers : musique, photos, vidéos perso etc.
- **Etre branché sur le secteur** : cela concerne bien entendu les ordinateurs portables. Il est très fortement recommandé d'être branché, car installer un système d'exploitation sur batterie est tout simplement... suicidaire. 😊 Il serait très ennuyeux pour votre installation que celle-ci soit coupée en plein milieu à cause d'une batterie vide !
- **Etre connecté à Internet** : c'est facultatif, mais je le recommande fortement là aussi. Cela permettra à l'assistant d'installation de télécharger immédiatement les dernières mises à jour des programmes ainsi que les traductions françaises qui pourraient manquer sur le CD d'Ubuntu. Oubliez le wifi ici, qui peut être un peu compliqué à configurer : branchez-vous à Internet avec un vrai câble réseau (RJ45).

Deux options peuvent être cochées (et je recommande de les cocher toutes les deux !) :

- **Télécharger les mises à jour pendant l'installation** : cela vous assurera que les programmes sont immédiatement le plus à jour possible. C'est préférable car les mises à jour corrigent des failles de sécurité, des bugs et améliorent certaines fonctionnalités des programmes déjà présents sur votre CD d'Ubuntu. Bien entendu, il faut être connecté à Internet avec un câble réseau pour cela.
- **Installer ce logiciel tiers** : cette option vous permet d'installer certains programmes propriétaires. Pour qu'Ubuntu reste libre, ces programmes ne sont pas installés par défaut, mais vous pouvez *demandeur* leur installation en cochant cette case (ce que je vous recommande pour votre confort). Vous aurez ainsi la possibilité de lire des MP3, du Flash et d'autres fichiers multimédia protégés par des licences propriétaires. Cette option peut aussi améliorer la prise en charge de votre carte wifi. Bref, c'est forcément intéressant pour vous, sauf si vous ne voulez pas installer de programme propriétaire sur votre machine.

Etape 3 : partitionnement du disque dur

Cette étape est la plus délicate. Non pas qu'elle soit complexe, mais elle vous demandera de bien réfléchir avant de faire votre choix. Vous allez en effet décider si vous installez Ubuntu à côté de Windows, ou si vous remplacez Windows.

Le sujet mérite que l'on en parle plus longuement, voici donc pourquoi je vous invite à me retrouver dans le chapitre suivant pour parler de partitionnement ensemble, tranquillement au coin du feu. 😊

Une fois que ce sera fait, l'installation pourra démarrer. Vous aurez donc fini d'installer Linux à la fin du prochain chapitre ! Nous avons fait les premières étapes de l'installation, il reste encore quelques options à définir et Ubuntu sera installé avant que vous ayez eu le temps de dire ouf !

Allez hop hop hop, on ne s'arrête pas en si bon chemin, rendez-vous au chapitre suivant ! 😊

Partitionner son disque

L'étape du partitionnement est probablement une des plus importantes de l'installation de Linux. Elle consiste à découper votre disque en plusieurs parties afin, par exemple, de ne pas mélanger Linux et Windows (ça ferait désordre 🤪).

Avant de partitionner votre disque, nous allons découvrir ce qu'est le partitionnement et à quoi ça sert. Ensuite, nous verrons comment effectuer le partitionnement à l'aide de l'outil proposé lors de l'installation d'Ubuntu. Soyez attentifs, car il faut que cette étape soit réalisée au mieux. Il est en effet délicat de faire marche arrière une fois le partitionnement effectué.



Bien que les risques soient minimes si vous suivez pas à pas mes instructions, je vous recommande quand même de faire une sauvegarde de vos données importantes sous Windows avant de commencer ce chapitre. Le partitionnement, si vous l'effectuez mal, pourrait affecter la partition Windows. Pas de panique hein, tout sera expliqué, mais on n'est jamais trop prudent non plus 😊

Défragmentez votre disque

Dans ce chapitre, on va beaucoup parler de votre disque dur. En fait, on ne va parler que de lui. C'est son organisation qui nous intéresse.

Un disque dur, ça ressemble à quoi ?

Alors pour bien commencer et pour mériter notre nom de "Site du Zéro", je pense qu'il serait bien que je vous montre à quoi ressemble un disque dur pour ceux qui n'en ont jamais vu de leur vie. Voici un disque dur tel qu'on peut en acheter dans le marché, vous en avez forcément un dans votre ordinateur actuellement :



Un disque dur, vu de l'extérieur

Un disque dur ne doit jamais être ouvert, vous risqueriez d'endommager vos données. Cependant, et pour des raisons purement pédagogiques, on va en ouvrir un pour bien comprendre ce qu'il y a à l'intérieur :



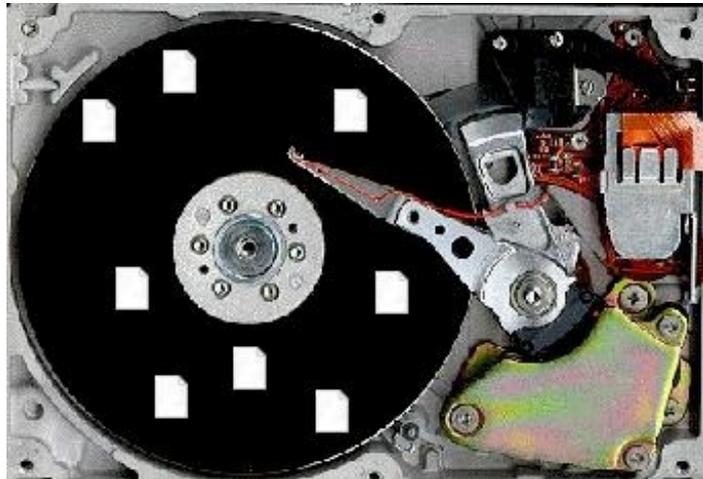
L'intérieur d'un disque dur

Comme vous le voyez, un disque dur est un empilement de disques, un peu comme des CD. Ils sont lus par une tête de lecture qui n'est pas sans rappeler la tête de lecture des disques vinyle.

L'importance de la défragmentation

Avant d'aller plus loin, il est très vivement conseillé d'effectuer une **défragmentation**. C'est une opération qui consiste en gros à mieux organiser les fichiers sur votre disque dur, à les rassembler pour éviter qu'ils ne soient éparpillés.

On dirait pas comme ça, mais vos fichiers sont parfois placés un peu n'importe comment à la surface de votre disque dur ! 🤪
Voici un petit schéma pour bien comprendre dans quel état est votre disque :



Avant la défragmentation, c'est le bazar sur votre disque

Sur la surface du disque, j'ai représenté une multitude de fichiers : ce sont les fichiers tels qu'ils sont placés sur votre disque actuellement. Un beau bordel. Parfois, certains fichiers sont coupés en plusieurs morceaux et éparpillés sur votre disque ! On dit que les fichiers sont **fragmentés** (coupés en plusieurs fragments).

Comment résoudre cela ? Votre ordinateur sait le faire, mais ça demande du temps. Grâce à un outil intelligent, appelé défragmenteur, il peut partir à la recherche des fichiers fragmentés et les rassembler tous au même endroit :



Après la défragmentation, c'est bien plus propre !

Les avantages seront les suivants :

- Comme vos fichiers seront près les uns des autres, le disque dur mettra moins de temps à les récupérer quand on les lui demandera. Votre Windows sera au final sensiblement plus rapide (surtout si vous n'avez jamais défragmenté et que vous utilisez votre ordinateur depuis des années !).
- Et surtout, c'est ce qui nous intéresse ici, cette organisation "prépare" votre disque au partitionnement que vous allez faire. Si vous ne le faites pas, il y a un risque (j'ai bien dit un "risque") que certains bouts de fichiers disparaissent lors du partitionnement, et vous pourriez vous retrouver à la fin avec un Windows instable !

Bon, vous avez compris, faut pas chercher à discuter : défragmentez votre disque, vous n'en tirerez que des avantages 😊

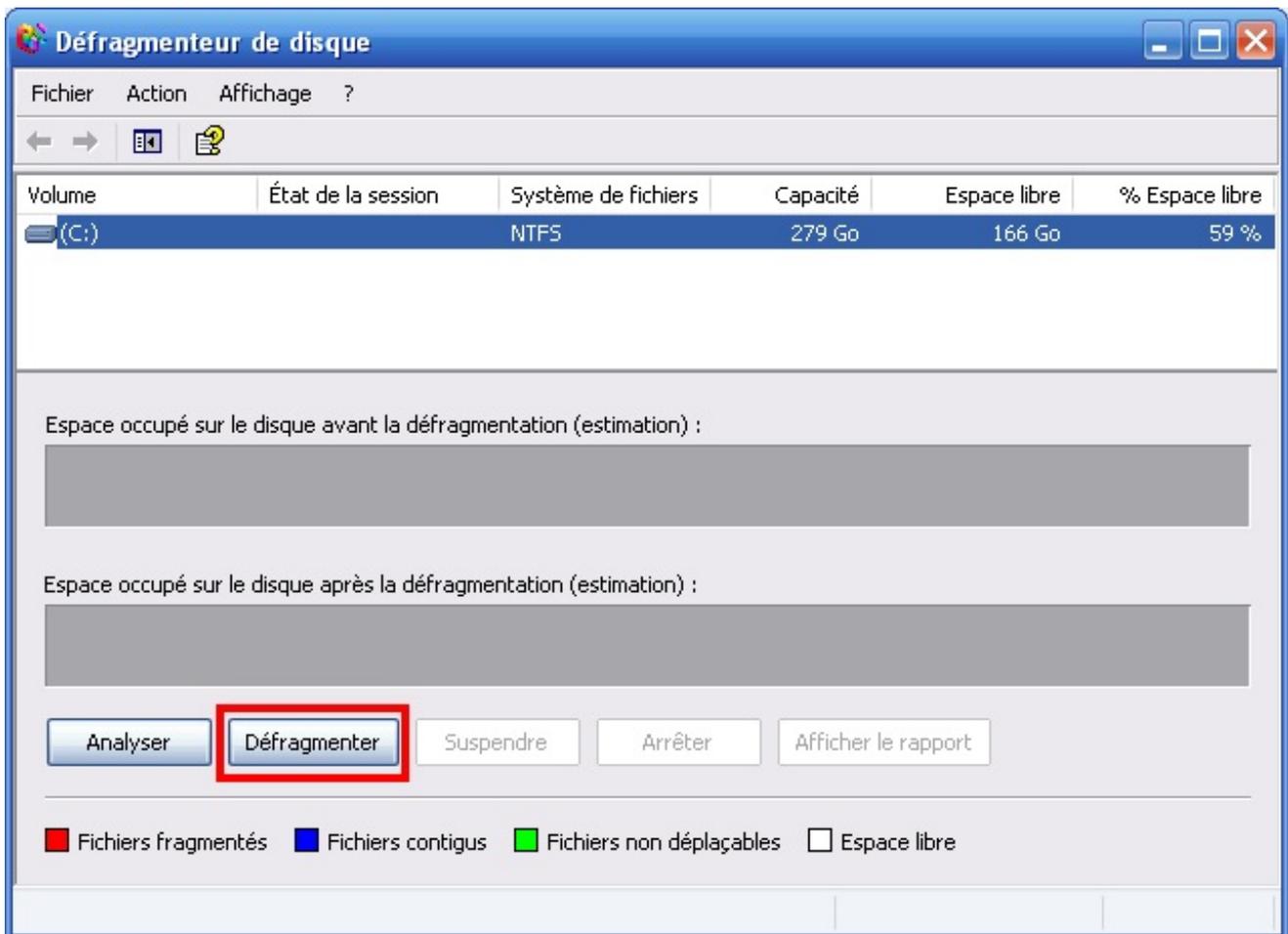
Pour défragmenter, un utilitaire est livré avec Windows. Retournez donc sous Windows pour effectuer la défragmentation si vous ne l'avez pas faite auparavant, c'est vraiment une étape importante.

Pour lancer cet utilitaire allez dans :

Démarrer / Tous les programmes / Accessoires / Outils Système / Défragmenteur de disque

Sous Windows XP

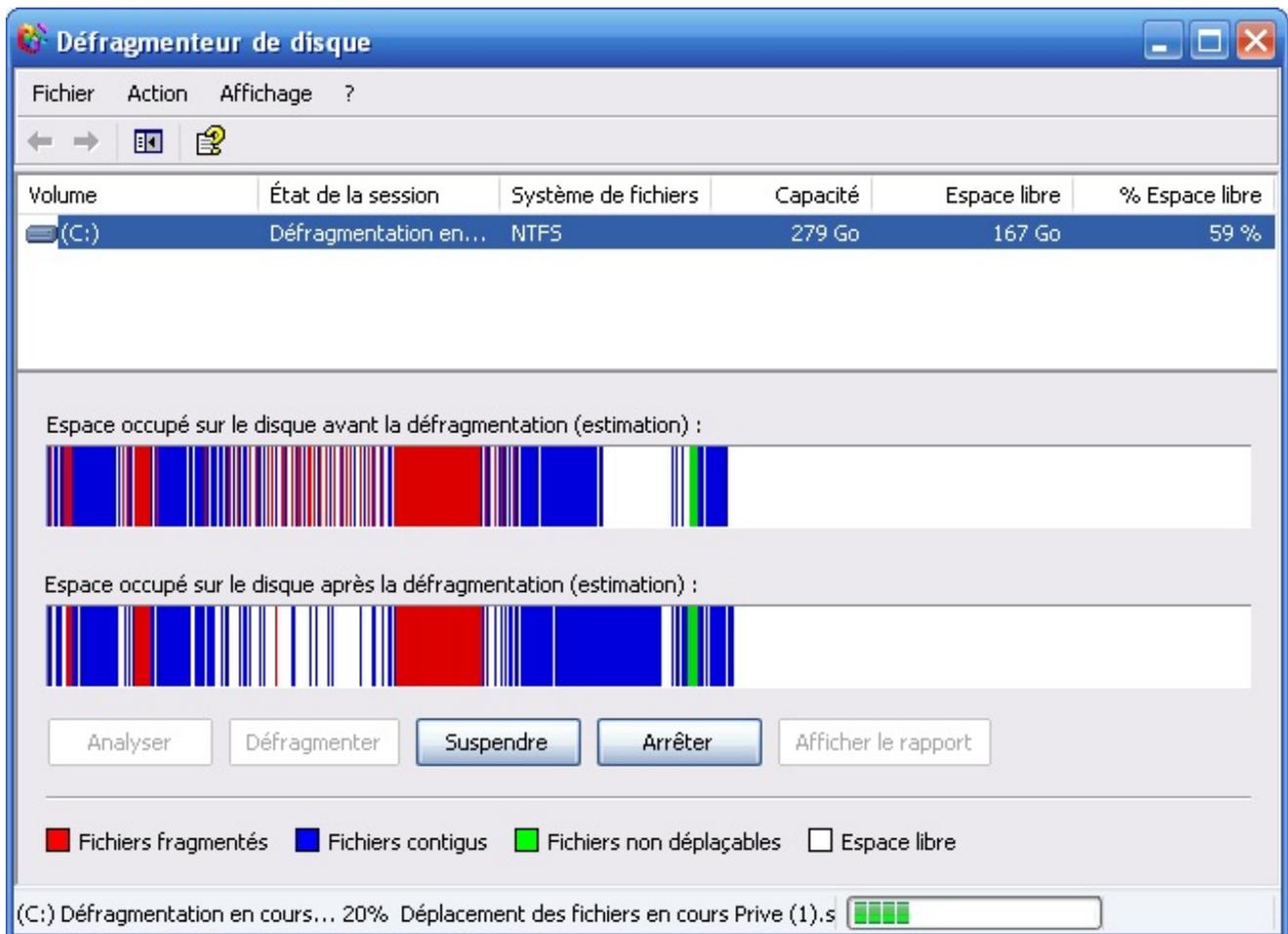
La fenêtre suivante s'ouvre :



Indiquez le disque dur que vous voulez défragmenter

Sélectionnez le disque dur que vous voulez défragmenter (celui qui contient Windows, généralement C:) et cliquez sur le bouton Défragmenter.

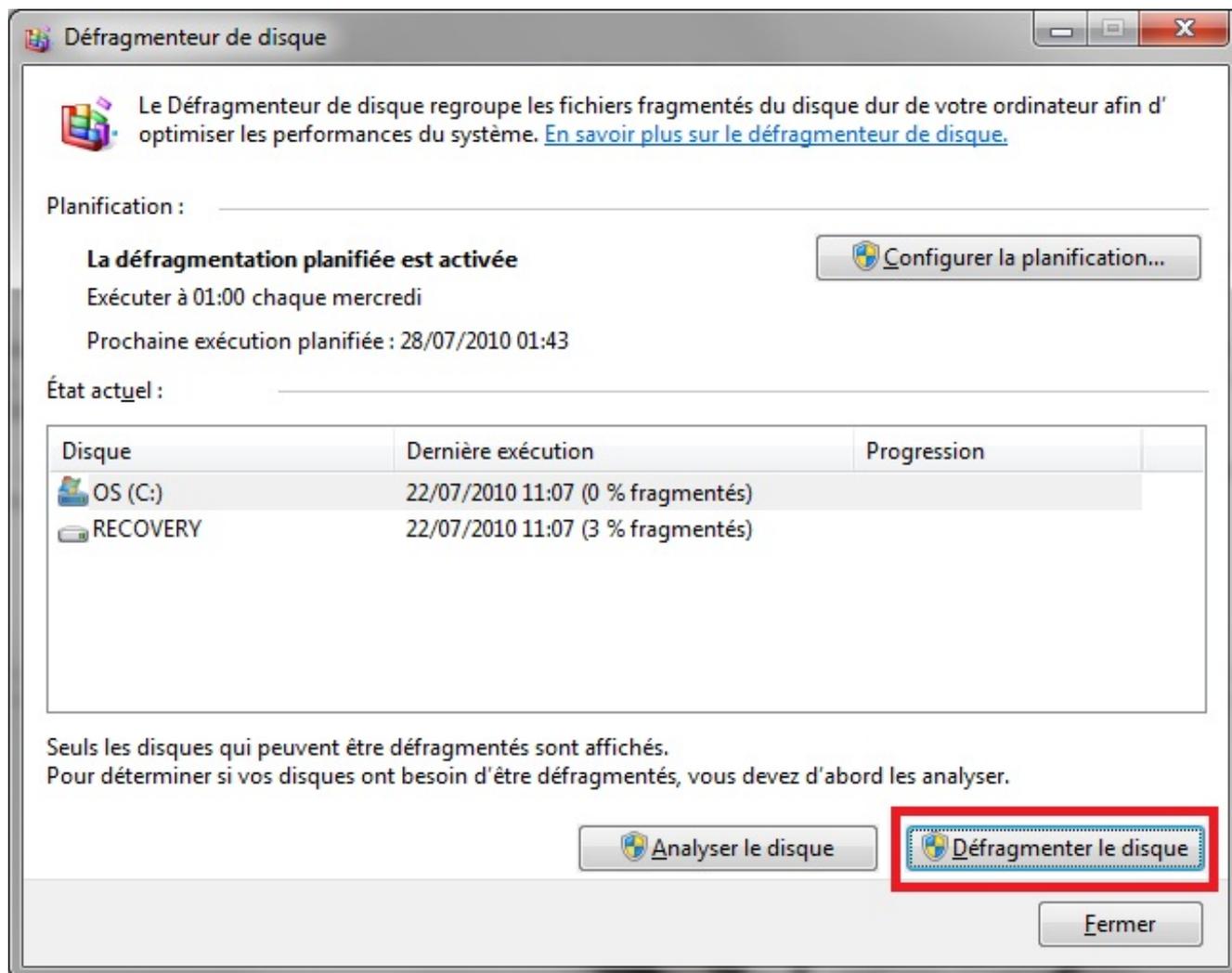
Vous pouvez sortir prendre l'air, parce que la défragmentation peut prendre un moment (plusieurs heures si vous ne l'avez jamais fait 🤖).



Pendant la défragmentation. L'avancement est indiqué en bas de la fenêtre

Sous Windows 7

La fenêtre est sensiblement différente, mais le principe est le même :



Il suffit de sélectionner le disque à défragmenter (si vous en avez plusieurs). Il est conseillé de tous les défragmenter. Cliquez ensuite sur "Défragmenter le disque".

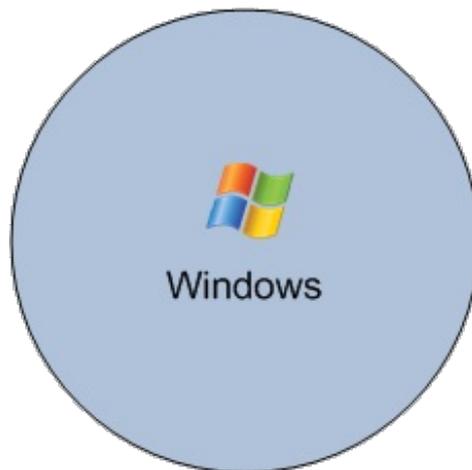
Ne coupez pas votre PC pendant la procédure et attendez sagement. Idéalement, évitez de trop toucher à votre ordinateur pendant la défragmentation pour ne rien perturber (ce n'est pas interdit, mais c'est déconseillé).

Une fois que c'est terminé, vous aurez un disque dur tout propre et bien organisé 😊

Qu'est-ce que le partitionnement ? Vive les schémas !

Pour expliquer le principe du partitionnement du disque dur, je vais avoir recours à quelques schémas 😊

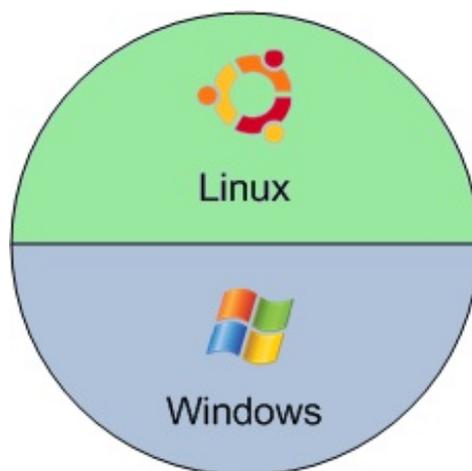
Je vais représenter votre disque dur par un cercle. Et qu'y a-t-il dessus actuellement ? Il y a de fortes chances pour qu'il n'y ait que Windows ! Votre disque dur ressemblerait donc à ça :



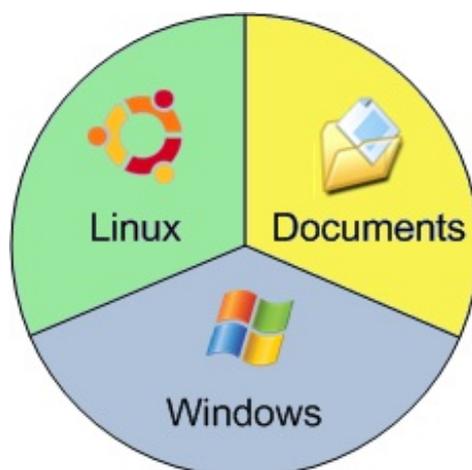
Votre disque "appartient" grosso modo à Windows. Il s'est installé dessus et il considère que tout lui appartient, donc qu'il peut mettre des fichiers où il veut sur le disque.

Supposons maintenant qu'on introduise Linux. Il est impossible de mettre 2 OS (ici Windows et Linux) ensemble au même endroit. Cela ferait trop de conflits. Pour résoudre le problème, on a inventé le partitionnement. Cela consiste à découper son disque en plusieurs parties (virtuellement hein, pas pour de vrai 🤪).

Si on donnait 50% de l'espace à Linux, et 50% à Windows, le schéma ressemblerait alors à ça :



Mieux encore, le fin du fin serait de créer une partition où l'on placerait nos documents :



Ainsi, Linux et Windows pourraient s'échanger vos fichiers (photos, documents texte, etc) sans interférer entre eux. Autre intérêt de cette technique : si par hasard vous devez réinstaller Linux ou Windows, vous ne perdriez pas vos documents car seules les partitions Linux ou Windows seraient formatées !

En quelque sorte, mettre vos documents dans une partition séparée, c'est les mettre à l'abri 😊

Les systèmes de fichiers

Sur chaque partition, les fichiers sont organisés selon ce qu'on appelle un **système de fichiers**. C'est en quelque sorte une façon d'organiser les fichiers : ils sont tous référencés dans une sorte de gigantesque annuaire.

Le système de fichiers permet aussi de dire qui a le droit de voir tel ou tel fichier. D'autre part, les systèmes de fichiers récents sont dits "journalisés", c'est-à-dire qu'en cas de crash (votre pc est éteint brusquement), le système est capable de retrouver ses fichiers sans trop de problèmes.

Règle importante : il ne peut y avoir qu'un seul système de fichiers par partition.

Il existe un nombre important de systèmes de fichiers différents, en voici quelques-uns à connaître.

Systèmes de fichiers Microsoft (DOS et Windows)

- **FAT 16** : un très vieux système de fichiers, capable de gérer jusqu'à 4 Go de données. Il est donc impossible de faire une partition en FAT 16 de plus de 4 Go. Il était très utilisé à l'époque du DOS et aux débuts de Windows 95.
- **FAT 32** : une évolution du FAT 16, qui pousse la limite de taille à 2 To (2000 Go). Le FAT 16 et le FAT 32 ont la particularité de beaucoup fragmenter les fichiers, d'où la nécessité de défragmenter régulièrement, sinon on prend le risque de voir son disque ressembler à un véritable champ de bataille.
- **NTFS** : apparu avec Windows NT, puis réutilisé par Windows XP, il permet de créer des partitions d'une taille allant jusqu'à 16 Eo (16 Exaoctets, soit 16 000 000 000 de Gigaoctets). Mais ce n'est pas tant la taille maximale qui est intéressante (on en est franchement loin) que les avantages que le NTFS procure à côté. Contrairement au FAT 32, c'est un système de fichiers journalisé qui récupère beaucoup mieux les données en cas de crash du disque. D'autre part, on peut donner des droits sur certains fichiers, les crypter, les compresser, etc. Enfin, les fichiers sont censés moins se fragmenter, mais ils se fragmentent cependant toujours.

Systèmes de fichiers Linux

- **Ext2** : c'est le système de fichiers qui a longtemps été utilisé sous Linux. Il a été développé par un français (Rémy Card) et présente la particularité *de très peu se fragmenter*. Ainsi, sous Linux et depuis longtemps, il n'y a pas besoin de faire de défragmentation.
- **Ext3** : l'ext3 est très proche de l'ext2, à une différence majeure près, la journalisation. En effet, ext2 n'était pas journalisé, et en cas de crash disque on risquait plus facilement une perte de données. Ce n'est plus le cas avec l'ext3.
A noter que l'ext2 et l'ext3 sont parfaitement compatibles entre eux, dans un sens comme dans l'autre.
- **Ext4** : une amélioration de l'ext3, relativement récente, qui améliore la prise en charge des gros disques durs et diminue les problèmes de fragmentation des fichiers.

Lequel choisir ?

Actuellement, et pour faire simple, sous Windows la plupart des gens sont en NTFS, et sous Linux la plupart sont en ext3. Nous allons choisir ext4, qui est plus récent et plus performant.

Je peux vous proposer ceci pour vos partitions :

- **Linux** : ext4
- **Windows** : NTFS (ça ne bouge pas)
- **Documents** : ext4

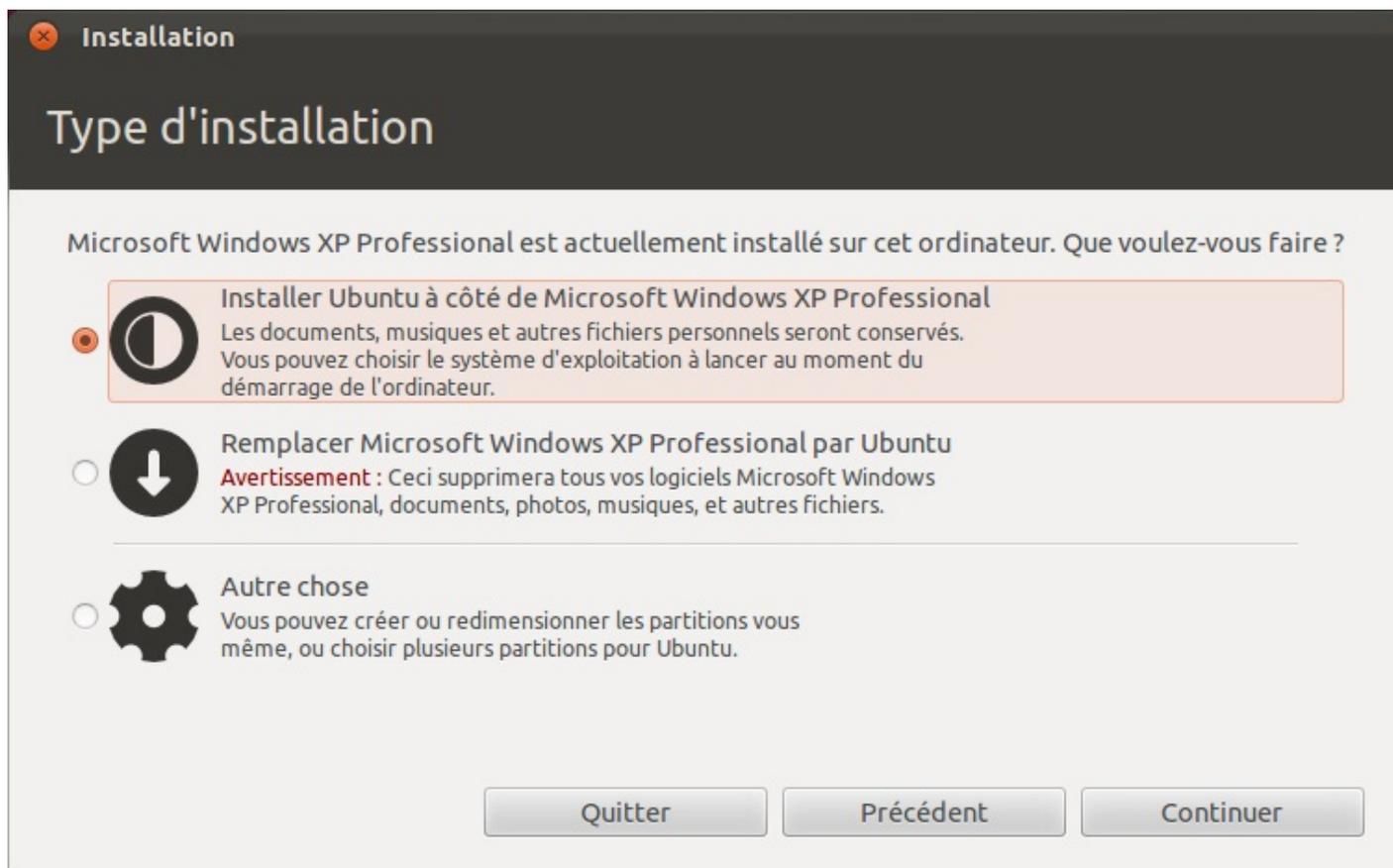
Le but est que la partition Documents puisse être lue et écrite depuis Windows et Linux, afin que vous puissiez accéder à vos documents que vous soyez sous Windows ou Linux.

L'ext4 fonctionne parfaitement sous Linux comme vous le savez. Cependant, Windows ne le reconnaît pas... à moins d'installer un programme qui lui permettra de l'utiliser. Je peux vous recommander [FS Driver](#), qui rend utilisable l'ext2, l'ext3 et l'ext4 sous Windows.

Prêts ? Partitionnez !

Revenons à l'installation d'Ubuntu, et plus précisément à l'étape du partitionnement.

La fenêtre vous propose plusieurs options dans le cas où Windows est déjà installé :



- **Installer à côté d'autres systèmes d'exploitation** : Ubuntu va se faire automatiquement de la place sur votre disque dur et créer les partitions pour vous. C'est la solution la plus simple que vous devriez choisir si vous ne voulez pas aller dans les détails. En revanche, vous n'aurez pas de partition spéciale pour les documents dans ce mode-ci. En bas de la fenêtre, vous pouvez déplacer le curseur pour décider de l'espace que vous attribuez à Windows et à Ubuntu.
- **Tout effacer et utiliser le disque entier** : tout le disque sera formaté, partition Windows comprise. **Ne faites cela que si vous voulez supprimer Windows !** Ubuntu sera installé sur l'ensemble du disque dur.
- **Définir les partitions manuellement (avancé)** : choisissez cette option si vous voulez créer vous-même les partitions. C'est plus complexe mais cela vous donnera plus de choix.

En résumé : si vous voulez aller vite et faire simple, choisissez l'option "Installer à côté d'autres systèmes d'exploitation". Sinon, choisissez le mode manuel : nous allons découvrir comment celui-ci fonctionne ci-dessous.

Le partitionnement manuel

Je vais supposer que vous faites un partitionnement manuel, et je vais donc vous montrer pas à pas comment ça fonctionne avec l'outil de partitionnement présent dans l'installateur d'Ubuntu. Mais avant cela, il faut que je vous fasse découvrir comment sont nommées les partitions du disque !

A propos du nom des disques

Si vous avez un seul disque dur sur votre ordinateur, pas de problème, vous ne risquez pas de vous tromper. Toutefois, si par hasard vous avez plusieurs disques, je pense que vous apprécierez que je vous explique comment les disques durs sont nommés sous Linux.

En effet, c'est très différent de Windows où on a l'habitude des sempiternels C:, D:, E: etc. On va découvrir les noms des disques sous Linux avec un exemple :

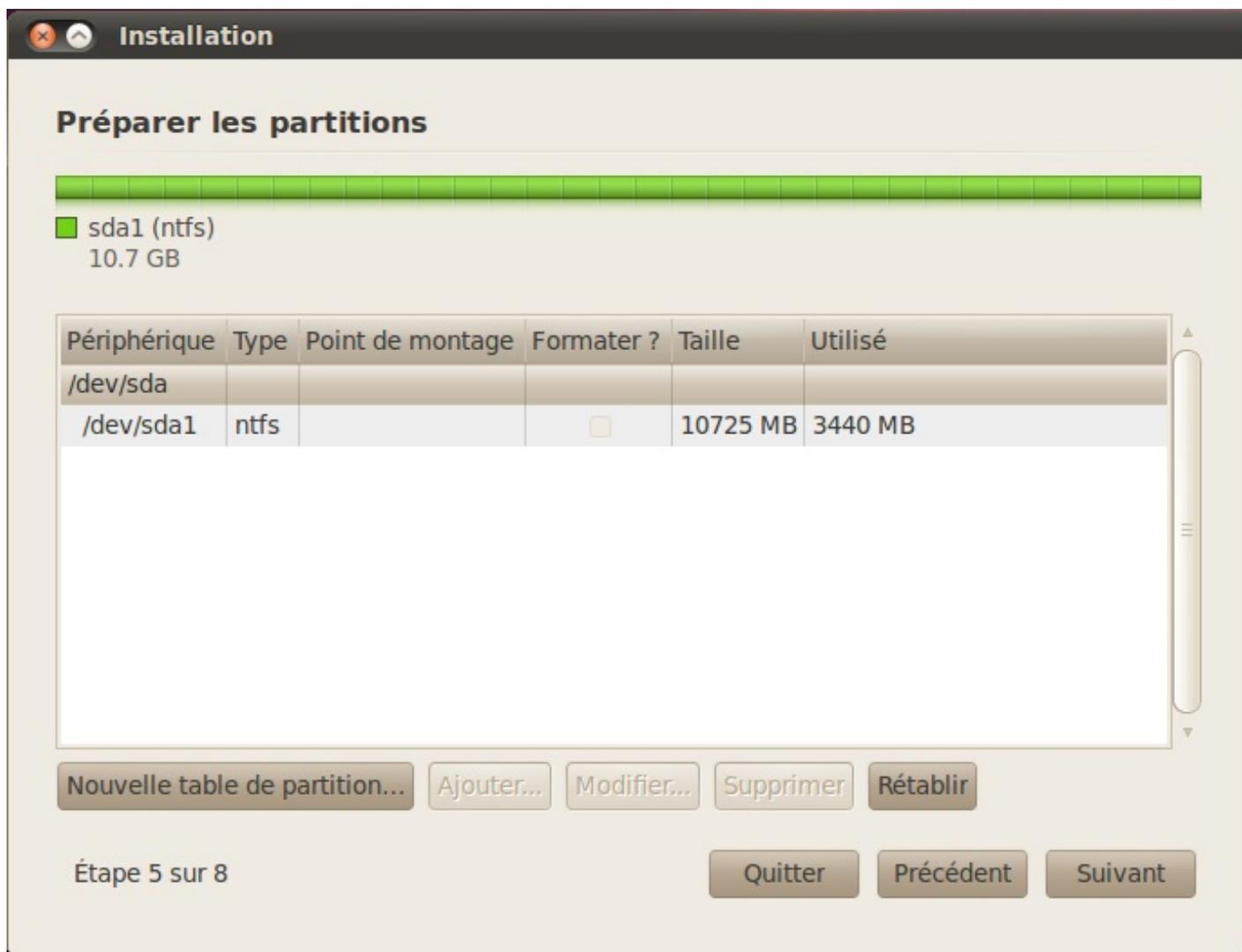
hda

- **h** : la première lettre indique si le disque est de type IDE ou SCSI (un type de connexion différent à la carte mère). Si c'est une IDE, la lettre est un h, si c'est un SCSI (ou un S-ATA), la lettre est un s.
- **d** : cette lettre ne change pas.
- **a** : c'est cette lettre qui indique les différents disques durs. hda représente le premier disque dur IDE, hdb le second, hdc le troisième etc.

Lorsqu'on crée des partitions, on ajoute généralement un chiffre représentant le numéro de la partition. Ainsi, si on a 3 partitions sur notre disque hda, elles seront nommées hda1, hda2, hda3...

L'outil de partitionnement manuel

Revenons à notre installation d'Ubuntu. Si vous avez choisi le partitionnement manuel, vous devriez voir l'écran suivant :



Dans la première ligne, vous avez le nom du disque dur : /dev/sda. C'est donc le disque dur sda. Comme je n'ai qu'un seul disque dur, il est facile à reconnaître.

En-dessous, on voit les partitions que contient le disque. Ici, il y en a une seule (qui correspond à Windows).

Actuellement, j'ai une partition nommée **sda1** car, comme je vous l'ai expliqué un peu plus haut, les partitions sont numérotées. Quand on va rajouter des partitions, vous allez voir qu'elles vont s'appeler sda2, sda3, etc.

Cette partition est de type NTFS, le système de fichiers de Windows. C'est donc la partition utilisée par Windows, et c'est là que Windows est installé.



Il se peut que vous ayez plus de partitions que moi. Par exemple, il est fréquent de voir certains PC achetés dans le commerce équipés d'une toute petite partition "restore" capable de restaurer certains logiciels de votre ordinateur en cas de formatage. Si vous en avez une, n'y touchez pas. Votre PC est peut-être "tatoué". Pour plus d'informations, lisez la [documentation Ubuntu](#) à ce sujet.

Si cela vous semble trop compliqué, vous pouvez toujours opter pour une installation simple depuis Windows qui ne pose pas ce genre de problème de partitionnement. Nous avons décrit cette procédure d'installation "simplifiée" au début du chapitre précédent.

1/ Réduire la taille de la partition Windows

La première étape consiste à limiter la place de Windows pour faire de la place pour Ubuntu.

Sélectionnez la partition Windows en cliquant dessus (/dev/sda1 de type NTFS dans mon cas), puis cliquez sur le bouton "Modifier". Dans la fenêtre qui s'ouvre, vous pourrez modifier la taille de la partition en entrant une nouvelle taille plus petite dans le premier champ, puis en cliquant sur "Valider" :

Modifier la partition

Modifier une partition

Taille de la nouvelle partition en Mo (1 000 000 octets) 5000

Utiliser comme : Ne pas utiliser la partition

Formater la partition :

Point de montage :

Annuler Valider

Redimensionnez la partition Windows pour faire de la place pour Linux

La taille est exprimée en Mo.

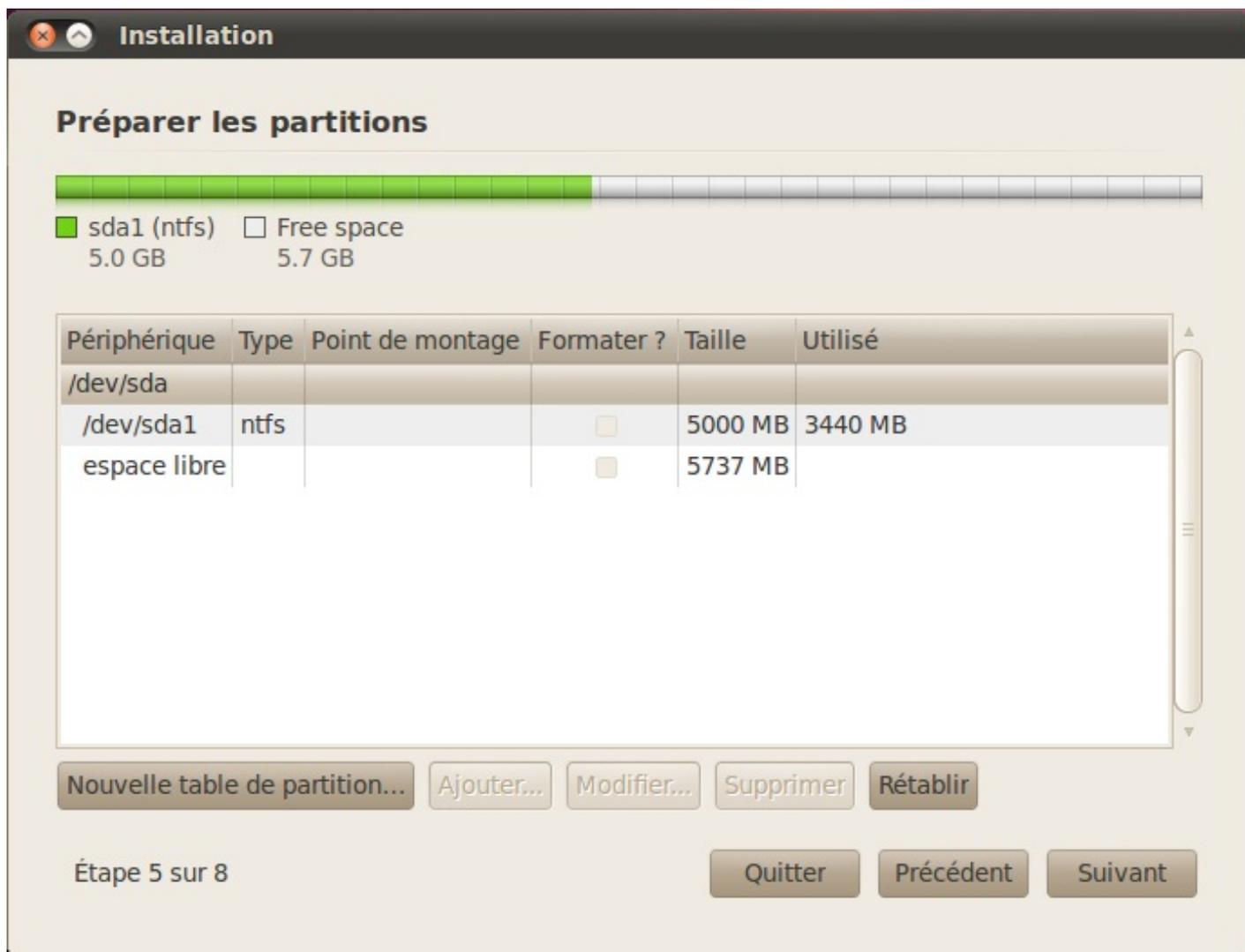
1000 Mo font environ 1 Go. Indiquez la taille que vous souhaitez laisser à Windows.



Veillez à laisser tout de même assez de place à Windows, sinon vous ne pourrez plus faire grand chose dessus (impossible d'installer un nouveau jeu s'il n'y a plus de place sur la partition par exemple).

C'est à ce moment-là que la défragmentation se révèle utile. Comme toutes les données ont été groupées au même endroit, ça évite que certains fichiers égarés soient accidentellement supprimés. Ce serait un tantinet ballot avouez 

Vous devriez alors avoir de l'espace libre comme moi :



2/ Créer une partition pour installer Ubuntu

Ubuntu vous propose de créer 2 types de partitions :

- **Primaires** : c'est la partition de base, classique. On ne peut en créer que 4 par disque.
- **Logiques** : c'est un type de partition qui peut contenir de nombreuses sous-partitions. Celles-ci n'ont pas de limite de nombre comme les partitions primaires.

Cliquez sur la partition libre du disque dur, puis cliquez sur le bouton "Nouvelle table de partition" en bas.

Créer une nouvelle partition

Type de la nouvelle partition : Primaire Logique

Taille de la nouvelle partition en Mo (1 000 000 octets) : 5000

Emplacement de la nouvelle partition : Début Fin

Utiliser comme : système de fichiers journalisé ext4

Point de montage : /

Annuler Valider

Dans la fenêtre qui s'ouvre, je vous invite à créer une partition "Primaire", de la taille que vous voulez, qui servira à installer Ubuntu et de futurs programmes. Indiquez au moins 3 ou 4 Go.

Laissez "Emplacement de la nouvelle partition : Début" pour que la partition soit créée au début de l'espace libre.

Choisissez le système de fichiers ext4.

Enfin, choisissez le point de montage /. Je ne rentrerai pas dans le détail du point de montage, il est trop tôt pour vous expliquer cela, mais sachez qu'en gros cela permet d'indiquer le dossier dans lequel la partition sera utilisée (/ étant le dossier racine, un peu comme C:\ sous Windows).

Validez. La fenêtre principale se met à jour avec les nouvelles informations.

3/ Créer une partition pour les documents

Cliquez à nouveau sur l'espace libre et rajoutez une nouvelle partition de la même manière.

Créer une nouvelle partition

Type de la nouvelle partition : Primaire Logique

Taille de la nouvelle partition en Mo (1 000 000 octets) : 6000

Emplacement de la nouvelle partition : Début Fin

Utiliser comme : système de fichiers journalisé ext4

Point de montage : /home

Annuler Valider

Cette fois, vous pouvez créer une partition bien plus grande. Ce sera la partition où vous stockerez vos documents, un peu comme le "Mes documents" de Windows qui est souvent vite rempli de musiques et de films gourmands en espace disque.

Choisissez la taille que vous voulez pour cette partition mais veillez à laisser environ 1 Go (1000 Mo) de libre sur votre disque pour que l'on puisse créer une dernière partition après.

Choisissez là encore le système de fichiers ext4.

Pour le point de montage, choisissez /home (c'est le dossier "Mes documents" équivalent de Linux).

4/ Créer une partition pour la swap

Il faut enfin créer une partition d'environ 1 Go appelée "swap". C'est une partition un peu spéciale dont je ne vous ai pas parlé jusqu'ici pour ne pas vous embrouiller 😊

Pour faire simple, il s'agit d'une extension de la mémoire vive sur votre disque dur. Lorsque votre mémoire vive est pleine, Linux continue à fonctionner mais passe par le disque dur, grâce à la partition swap. Je simplifie un peu mais en gros c'est l'idée.



Créer une partition

Créer une nouvelle partition

Type de la nouvelle partition : Primaire Logique

Taille de la nouvelle partition en Mo (1 000 000 octets) 1000

Emplacement de la nouvelle partition : Début Fin

Utiliser comme : espace d'échange (« swap »)

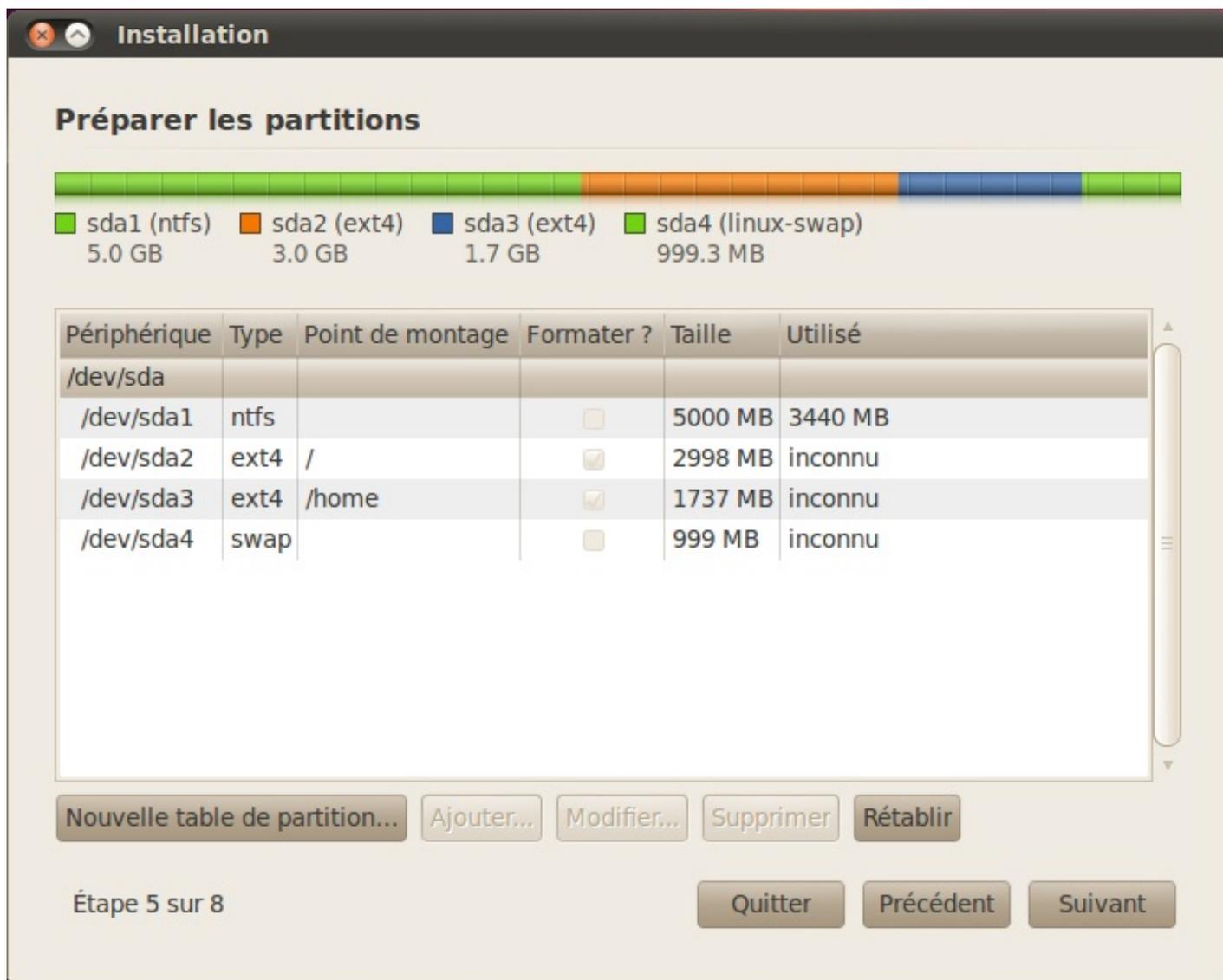
Point de montage :

Annuler Valider

Laissez tout l'espace libre restant pour cette partition, mais sachez que ça ne sert à rien qu'elle fasse plus d'1 Go en général.

Surtout, pensez à sélectionner le type "swap". Vous n'aurez pas besoin de préciser de point de montage contrairement aux autres fois.

Voici le résultat final :



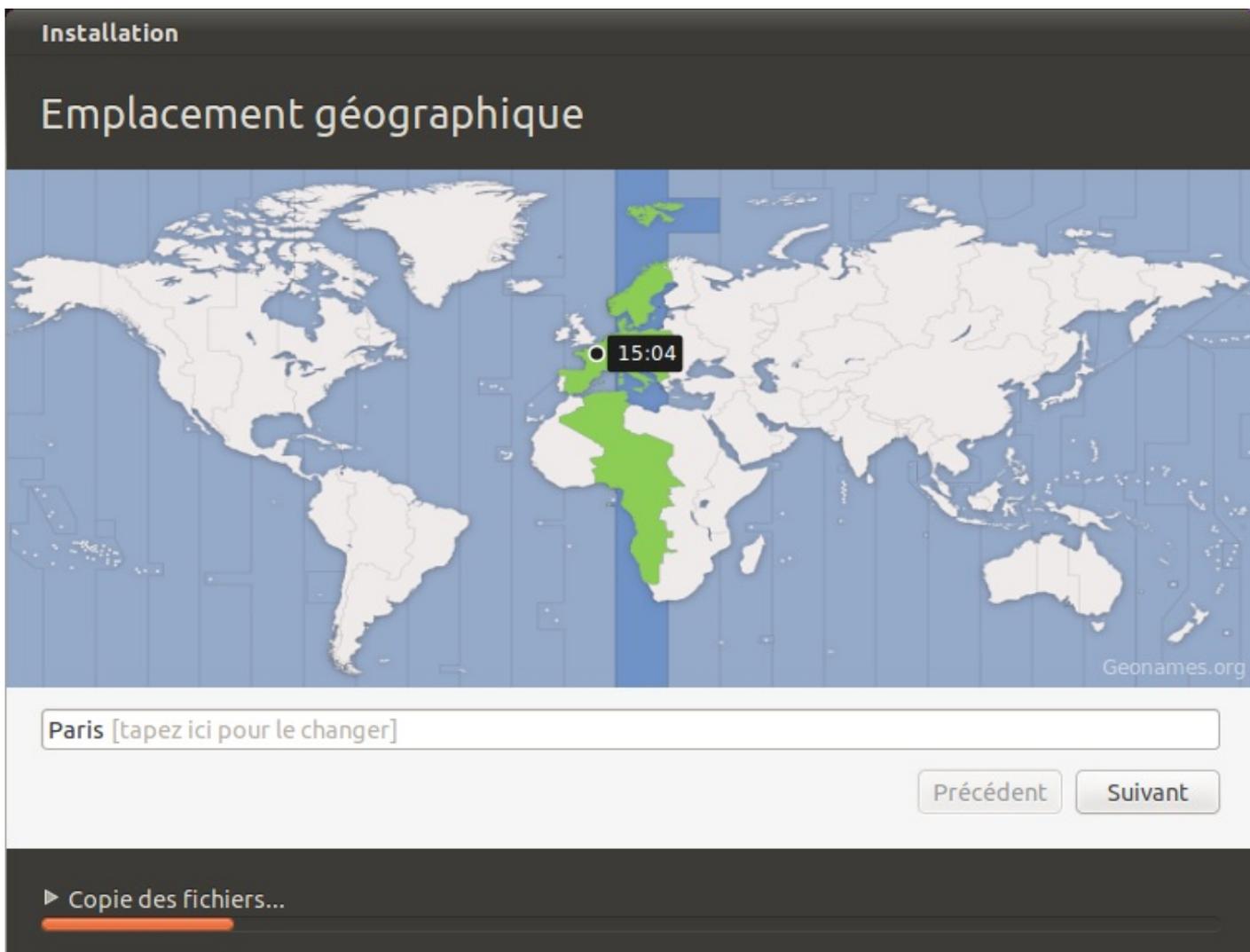
Votre disque dur est configuré. Cliquez sur "Suivant". 😊

La suite et fin de l'installation

Nous avons répondu à toutes les questions les plus importantes. L'installation démarre immédiatement après le partitionnement.

Cependant, il reste encore quelques petits détails à régler. Nous pouvons répondre à ces questions pendant le déroulement de l'installation. Allons-y !

Sélection du fuseau horaire

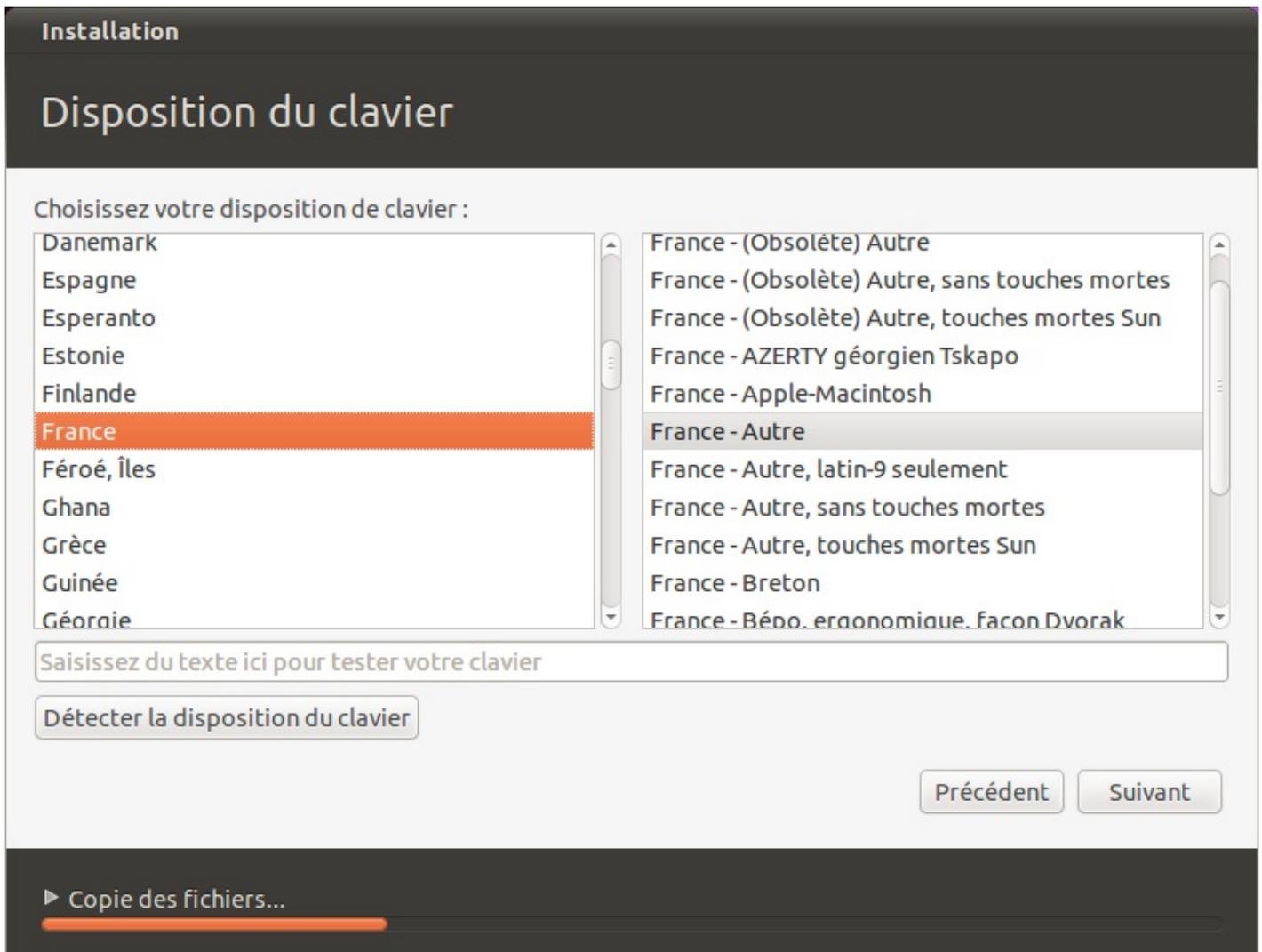


Comme vous pouvez le constater, l'installation s'effectue en tâche de fond (en bas de la fenêtre) pendant qu'on vous demande de régler quelques paramètres. Avec Ubuntu, on ne perd pas de temps ! 🤖

On vous demande sur cet écran près de quelle grande ville vous habitez pour régler le fuseau horaire. Cliquez sur la carte sur le point correspondant à la ville la plus proche.

Vérifiez bien que l'heure indiquée est la bonne.

Le type de clavier



Dans la fenêtre qui suit, on vous demande quel type de clavier vous utilisez. Si vous habitez en France, vous avez un clavier dit "AZERTY", mais il se peut que vous habitiez un pays qui possède un clavier différent, comme la Suisse ou le Canada.

Pour les français, normalement le bon type de clavier est sélectionné d'office. Vous pouvez tester dans le petit cadre de texte en bas votre clavier. Essayez de taper des symboles "spéciaux" comme é à ô ï etc. Si ces symboles s'affichent sans problème, c'est que vous avez indiqué le bon type de clavier 😊

Création du compte utilisateur

The screenshot shows the 'Identité' (Identity) screen in the Ubuntu installer. At the top, it says 'Installation' and 'Identité'. The fields are filled with the following information:

- Votre nom : Mathieu Nebra ✓
- Le nom de votre ordinateur : mateo21-desktop ✓
Le nom qu'il utilise pour communiquer avec d'autres ordinateurs.
- Choisissez un nom d'utilisateur : mateo21 ✓
- Choisissez un mot de passe : [masked] Mot de passe fort
- Confirm your password: [masked] ✓

Below the password fields, there are three radio button options:

- Ouvrir la session automatiquement
- Demander mon mot de passe pour ouvrir une session
- Chiffrer mon dossier personnel

At the bottom right, there are two buttons: 'Précédent' and 'Suivant'. At the bottom left, there is a progress bar with the text 'Copie des fichiers...' and a small play icon.

La fenêtre suivante vous demande votre nom ainsi qu'un login (pseudonyme) qui vous identifiera sur votre ordinateur. Choisissez aussi un mot de passe.

On vous demande le nom que vous voulez attribuer à votre ordinateur. On vous en propose un par défaut mais vous pouvez changer cela sans risque.

Dans mon cas, je vais indiquer "mateo21-desktop" comme nom d'ordinateur.

Vous pouvez aussi choisir comment l'ordinateur doit être démarré :

- **Ouvrir la session automatiquement** : votre ordinateur démarrera entièrement sans vous demander de saisir votre mot de passe à l'allumage. N'utilisez cette option que si vous êtes sûr que personne d'autre que vous n'a accès à votre ordinateur. Évitez de sélectionner cette option sur un ordinateur portable !
- **Demander mon mot de passe pour ouvrir une session** : on vous demandera votre mot de passe pour accéder à votre ordinateur. C'est l'option recommandée sur les ordinateurs portables en particulier comme je le disais.
- **Chiffrer mon dossier personnel** : si en plus vous êtes un peu parano et que vous avez des documents confidentiels, Ubuntu peut chiffrer vos documents, c'est-à-dire les crypter. Même un pro de l'informatique ne pourra pas lire vos documents s'il met la main sur votre ordinateur et qu'il analyse votre disque dur ! Je vous conseille de choisir un mot de passe solide pour que cela soit efficace.

Importation des données de Windows

Il se peut que l'installateur vous propose de récupérer quelques informations depuis Windows, si Windows est installé sur votre machine.

Sélectionnez ce que vous souhaitez récupérer (par exemple votre fond d'écran), puis continuez.

Suite et fin de l'installation

Votre travail à vous est terminé, vous avez indiqué toutes les informations nécessaires. L'installation se poursuit ensuite si elle n'était pas déjà terminée. Des écrans de présentation vous permettent d'en apprendre plus sur Ubuntu pour que vous ne vous ennuyiez pas. 😊

On vous proposera de redémarrer pour finaliser l'installation de Linux. C'est bon, bravo, Linux est installé ! 😊



Juste avant l'extinction du PC pour le redémarrage, un message vous demandera de retirer le CD d'installation du lecteur et d'appuyer ensuite sur la touche Entrée de votre clavier. Cette précaution permet de s'assurer qu'Ubuntu démarrera bien à partir du disque dur (et non du CD !).

GRUB au démarrage

Lors de chaque démarrage, si vous avez choisi d'installer Linux sur le même disque dur que Windows, on vous demandera quel OS vous voulez charger :

```
GNU GRUB  version 0.95  (638K lower / 186304K upper memory)

Ubuntu, kernel 2.6.8.1-3-386
Ubuntu, kernel 2.6.8.1-3-386 (recovery mode)
Memory test
Other operating systems:
Windows NT/2000/XP

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, or 'c' for a command-line.
```

Ca, c'est GRUB, le programme qui permet de choisir l'OS à lancer au démarrage (souvenez-vous, je vous en avais parlé dans le premier chapitre !).

Il y a plusieurs options, mais ne paniquez pas, c'est très simple :

- **Ubuntu** : choisissez la première ligne pour démarrer Ubuntu, c'est-à-dire Linux.
- **Ubuntu (recovery mode)** : si vous avez des problèmes au lancement d'Ubuntu, utilisez ce "mode de récupération" pour résoudre ces problèmes. N'utilisez ce mode que si on vous demande de le faire.
- **Memory Test** : pour faire un test de mémoire au cas où vous suspectiez que votre mémoire vive est défaillante. Attention le test est long, là encore ne le faites que si vous en avez vraiment besoin.
- **Other Operating Systems : Windows** : ah ben ça, c'est pour démarrer Windows 😊

Vous utiliserez les flèches de votre clavier pour sélectionner l'OS qui vous intéresse, et vous taperez Entrée pour valider. Si vous mettez trop de temps avant de vous décider, GRUB lancera l'OS sélectionné. Nous verrons plus tard comment faire pour que Windows soit sélectionné par défaut si vous le désirez.



Si vous avez installé Ubuntu sur un ordinateur équipé de Windows Vista ou Windows 7 et que vous constatez que celui-ci ne démarre plus, sachez qu'il s'agit d'un problème facile à régler. La solution se trouve ici.

Après l'installation, vous devrez certainement faire quelques petites configurations si une partie de votre matériel n'est pas reconnu immédiatement.

- Si vous avez un **problème avec le wifi** pour accéder à internet, sachez que c'est un cas très courant (ça vous rassure hein ? 😊). Il faut trouver un moyen d'installer des pilotes pour votre carte wifi sous Linux. [Ce tuto rédigé par Firemann](#) devrait vous aider, lisez-le !
- Si vous avez **une autre question**, n'hésitez surtout pas à vous rendre sur [les forums](#) du site. Vous y trouverez rapidement de l'aide 😊
Vous trouverez aussi une documentation bien fournie sur [Ubuntu-fi.org](#). Il y a beaucoup de chances pour que la solution à votre problème soit déjà expliquée, il faut juste prendre un peu le temps de chercher.

Dans les prochains chapitres, nous allons nous familiariser un peu avec les environnements de bureau KDE et Gnome ainsi qu'avec leurs principaux logiciels. Aucune difficulté n'est à prévoir, ça sera du gâteau pour vous 😊

📁 Découverte du bureau Gnome

Nous y voilà enfin ! Après avoir découvert ce qu'était Linux et comment on l'installait, vous devriez avoir maintenant un Linux qui tourne sur votre ordinateur. Parfait. Vous l'avez installé, et maintenant ?

On n'installe pas Linux juste pour la beauté du geste et pour dire "ça y est je suis sous Linux !". Si vous ne savez pas vous en servir, Linux risque plus de faire de la déco sur votre ordi qu'autre chose. 😊

Toute la suite de ce cours sera donc dédiée à la découverte et à l'étude plus en profondeur de Linux. Vous n'imaginez pas encore toutes les choses que vous allez apprendre 😊

Nous allons commencer en douceur par la découverte de Gnome. C'est un des gestionnaires de bureau de Linux les plus utilisés. C'est notamment celui que l'on trouve par défaut dans Ubuntu. Simple et clair, il conviendra à la plupart des nouveaux utilisateurs de Linux. Il est facile à prendre en main.

Ce chapitre vous propose une petite visite guidée de l'environnement Gnome. 😊

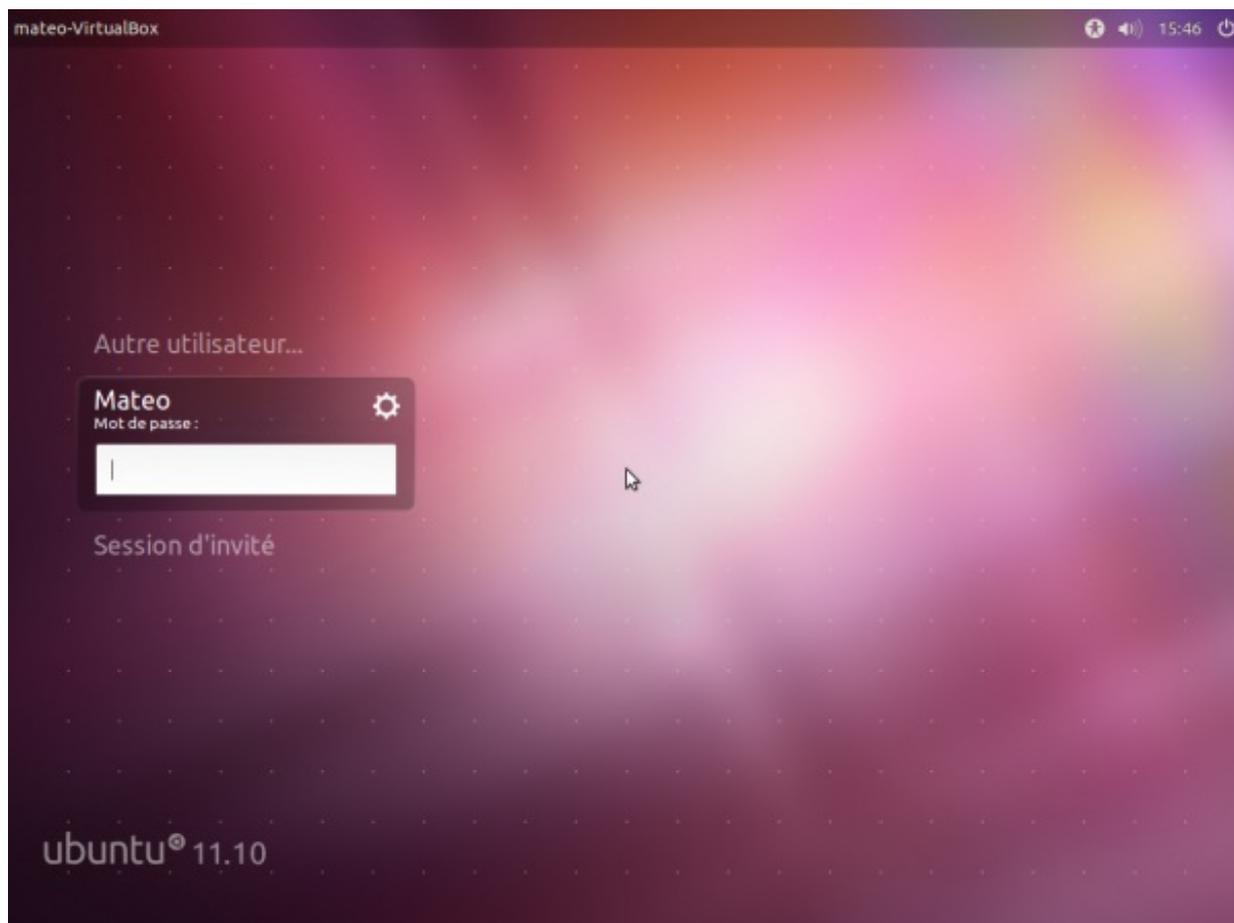
Bienvenue sur le bureau Gnome

Ce chapitre suppose soit :



- que vous avez installé Ubuntu (avec Gnome par défaut) dès le début.
- que vous avez installé autre chose (Kubuntu, Xubuntu) mais que vous avez installé le paquet *ubuntu-desktop* par la suite pour avoir Gnome. En savoir plus sur l'installation de plusieurs gestionnaires de bureau.

Lors du démarrage d'Ubuntu, vous allez être accueilli par une fenêtre de login. Cette fenêtre vérifie votre identité en vous demandant d'abord votre identifiant, puis votre mot de passe :



Bon le principe est simple, vous ne devriez pas avoir trop de mal : vous devez rentrer votre login et votre mot de passe. Ce sont les informations que vous avez rentrées lors de l'installation d'Ubuntu.



Mais pourquoi faut-il s'authentifier à chaque fois que l'on démarre Linux ? Si je suis seul, ce n'est pas la peine de mettre



Il est vrai que sous Windows on n'a pas trop l'habitude de s'authentifier, surtout sur son ordinateur de la maison... quoique, les habitudes sont en train de changer de ce côté-là.

Sous Linux, la conception est totalement différente. C'est un OS qui se veut vraiment multiutilisateur, c'est-à-dire que plusieurs personnes peuvent utiliser le même ordinateur en même temps (en le contrôlant à distance par Internet par exemple). Il y a une vraie politique de la sécurité, et c'est pour ça que même pour l'ordinateur de la maison chacun doit avoir son login et son mot de passe. Cela permet notamment de savoir à qui appartient tel ou tel fichier.

Les options

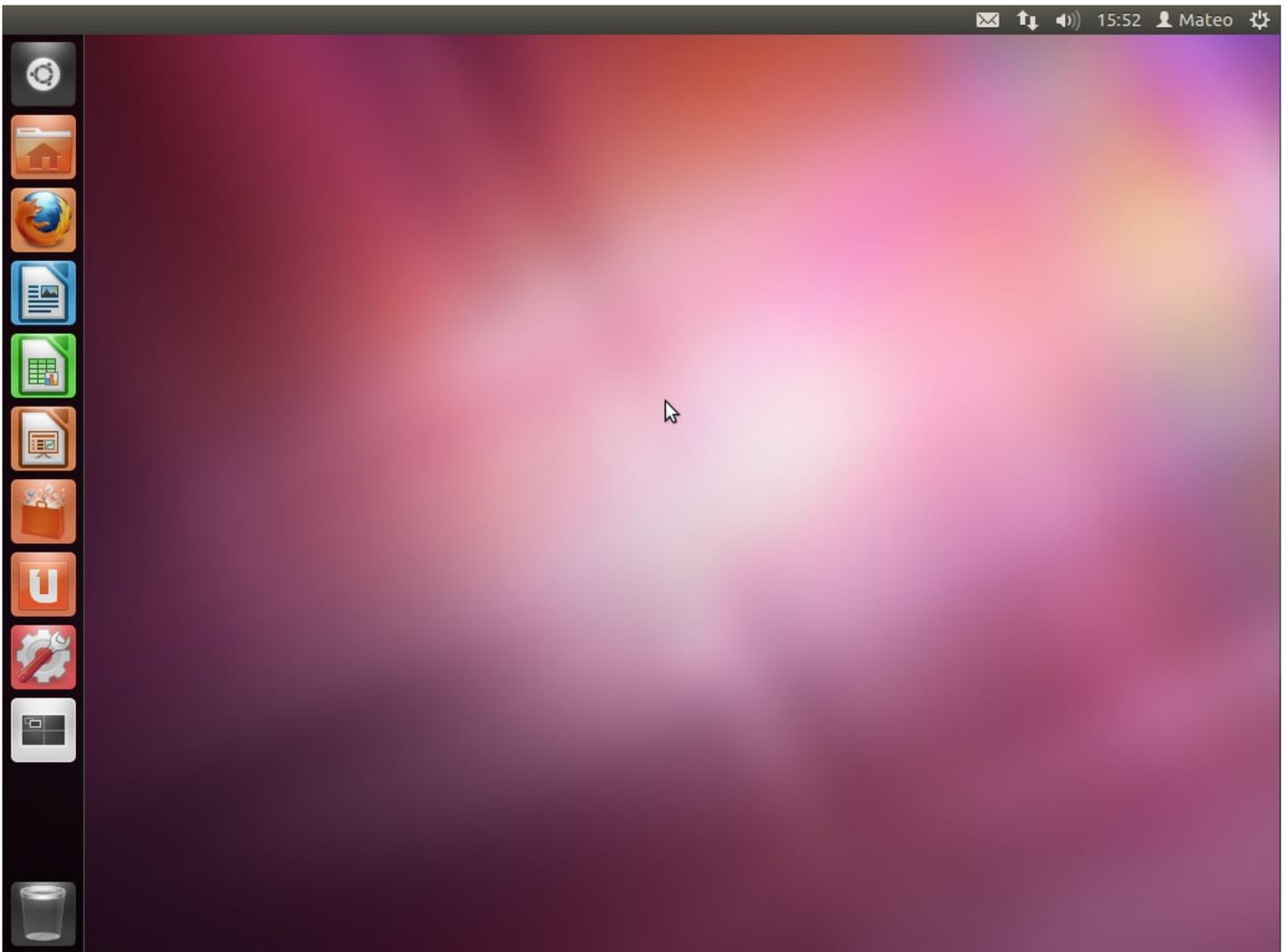
La fenêtre de login vous propose quelques options. Vous pouvez notamment y choisir votre gestionnaire de bureau :



Par défaut, seul Gnome est installé, mais par la suite si vous installez d'autres gestionnaires de bureau, vous pourrez sélectionner celui que vous souhaitez utiliser à ce moment-ci.

Présentation du bureau Gnome

Une fois loggé, vous arrivez sur le bureau de Gnome :



Le bureau est assez vide d'icônes, ce qui n'est pas pour me déplaire moi qui ai horreur des bureaux encombrés. 😊

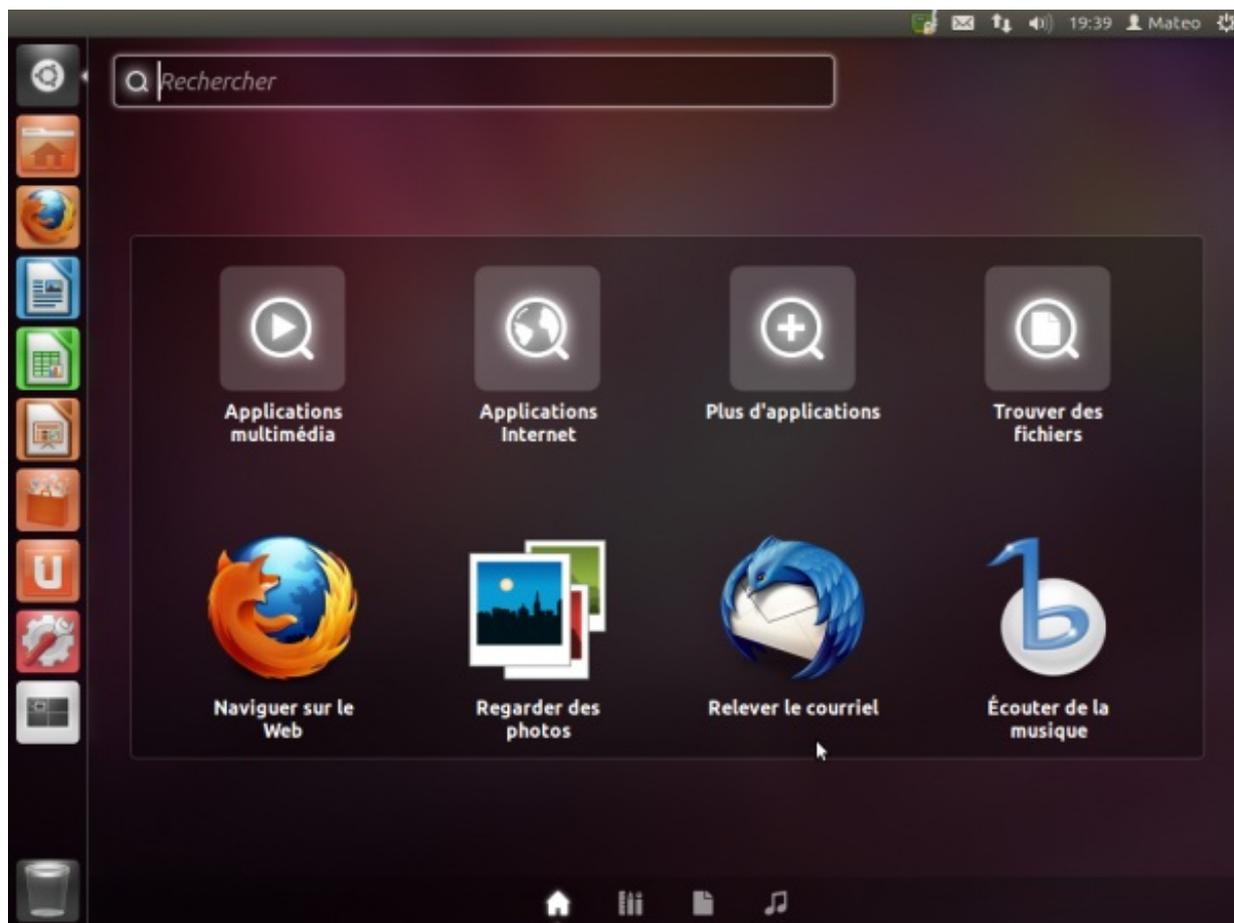
En fait, c'est un peu la philosophie de Gnome : une simplicité pour une meilleure esthétique. Ubuntu propose une version quelque peu modifiée de Gnome (appelée Unity). Nous allons la découvrir ensemble dans ce chapitre.

La barre Unity à gauche

Commençons par la barre à gauche. Très importante, vous pouvez y lancer de nouvelles applications et afficher les fenêtres déjà ouvertes. C'est une barre qui ressemble beaucoup à la barre des tâches de Windows 7 ou au dock de Mac OS X.



La première icône tout en haut ouvre le tableau de bord :



De là, vous pouvez lancer tous vos programmes. Je vous laisse le parcourir un peu, l'ensemble est plutôt intuitif vous allez voir !



Vous pouvez lancer n'importe quel programme en tapant simplement son nom. Essayez de taper "Firefox" pour voir !

Les autres icônes de la barre Unity permettent d'ouvrir vos programmes favoris. Vous pouvez ajouter et retirer des icônes dans ce menu : les options apparaissent lorsque vous faites un clic droit.

Enfin, l'icône "Espaces de travail" permet de changer de bureau. En effet, vous avez 4 bureaux disponibles sous Ubuntu ! Si vous avez beaucoup de fenêtres ouvertes, cela peut vous permettre de mieux vous organiser.

La barre en haut

Elle donne accès aux menus de la fenêtre ouverte. Pointez avec la souris pour les faire apparaître (si vous avez une fenêtre ouverte).



Sur le côté droit de cette barre, vous retrouvez quelques icônes de notification ainsi que la date. Enfin, un bouton en haut à droite vous propose d'arrêter votre ordinateur, ou bien de le redémarrer, de changer d'utilisateur, etc.

Nautilus, l'explorateur de fichiers

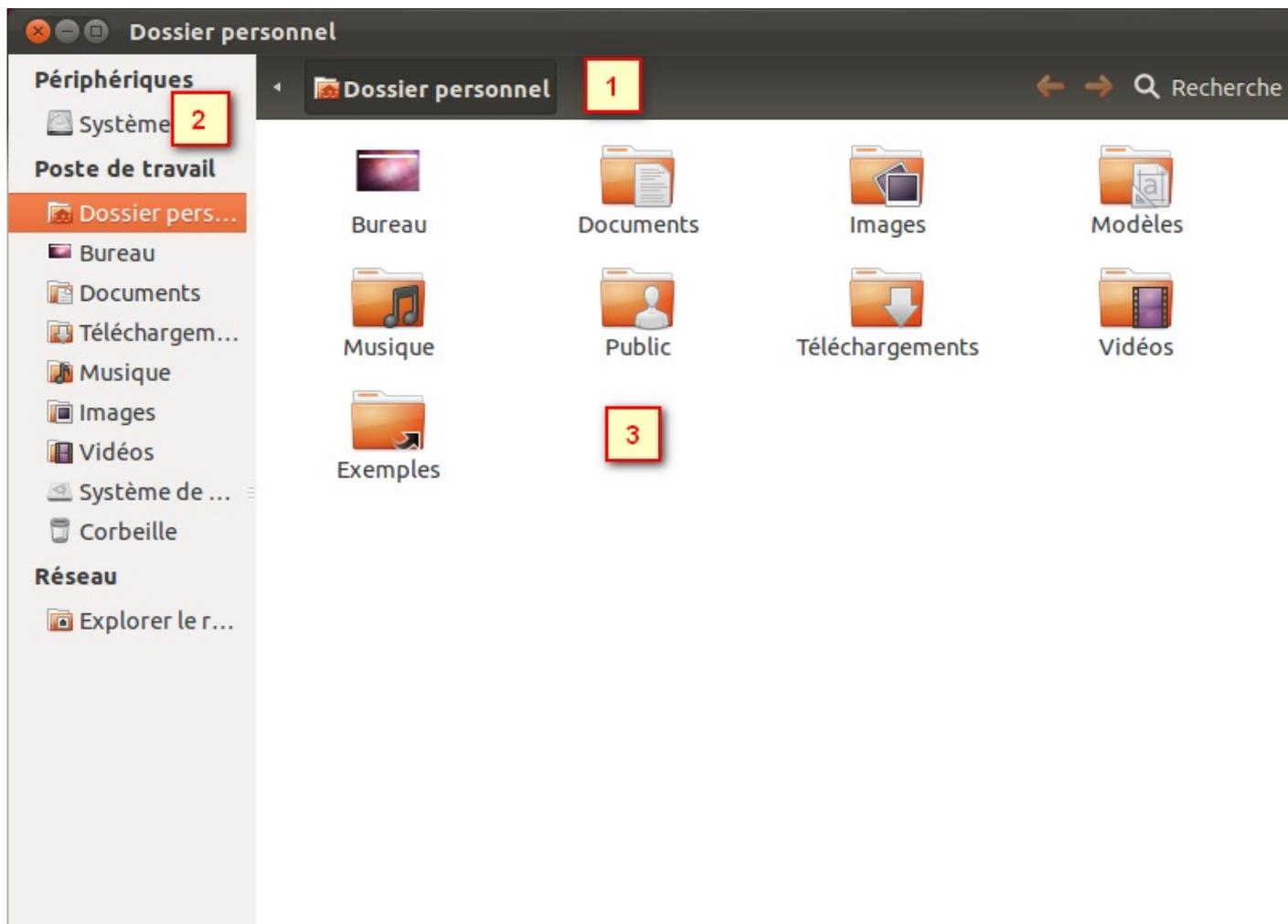
Nous allons commencer notre tour d'horizon de certains logiciels phares de Gnome par **Nautilus**. Nautilus est un explorateur de fichiers. C'est un programme du même type que l'explorateur de Windows.

Comme tout bon explorateur de fichiers qui se respecte, il vous permet de parcourir les dossiers et fichiers de votre disque dur et de les ouvrir. C'est donc un programme que vous risquez de lancer souvent. 😊

Pour ouvrir Nautilus, cliquez sur l'icône dans la barre Unity :



Nous nous trouvons ici dans le dossier personnel "Home", l'équivalent de "Mes documents" sous Linux.



Cette fenêtre est simple :

1. La première zone indique le chemin dans lequel vous vous trouvez, c'est-à-dire le nom du répertoire que vous êtes en train de visualiser. Chaque dossier est représenté par un bouton, et vous pouvez cliquer sur un des dossiers parents pour revenir en arrière. On parlera du fonctionnement des dossiers sous Linux un peu plus loin dans le cours 😊
2. Sur la gauche, une petite barre de raccourcis vous permet d'accéder à certains dossiers courant comme votre dossier personnel, le bureau, le lecteur CD, etc.
3. Enfin, la partie centrale affiche les fichiers et dossiers proprement dits.

Nautilus est donc un logiciel tout simple vous permettant de consulter les fichiers présents sur votre disque dur, mais aussi sur des lecteurs CD / DVD. Son utilisation devrait vous être familière tant il ressemble à l'outil de Windows.

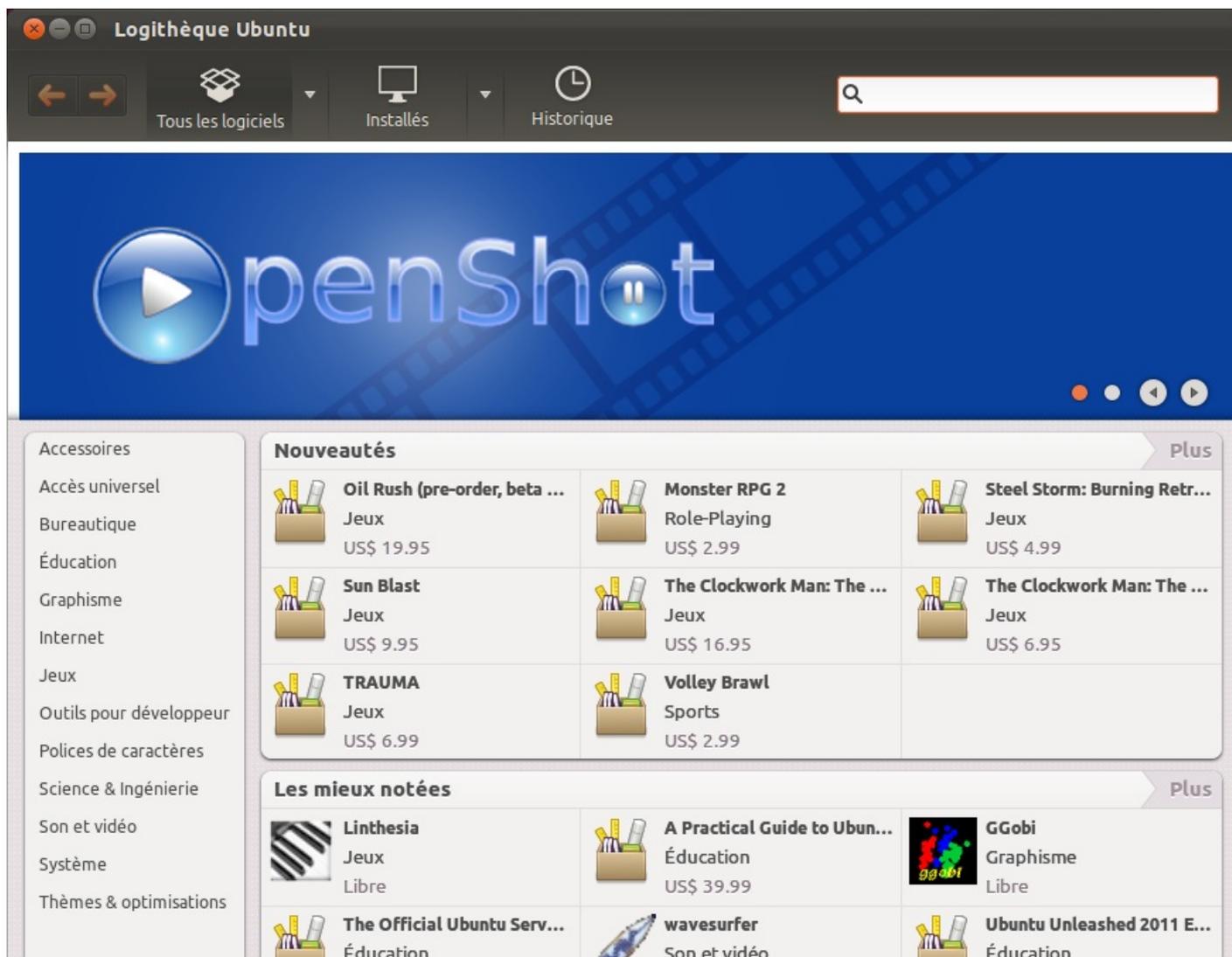
Vous mettrez un peu de temps à vous faire à l'organisation des dossiers, qui est un peu particulière sous Linux, mais vous finirez par prendre vos repères. Pour le moment, je vous conseille d'utiliser votre répertoire personnel. Vous pouvez y stocker tous vos documents, vos vidéos, votre musique, etc. 😊

Ajouter / supprimer des programmes
Ajout et suppression des programmes

L'ajout et la suppression de programme est simple et intuitive. Rendez-vous dans la logithèque Ubuntu en cliquant sur cette icône :



La fenêtre principale s'ouvre alors :

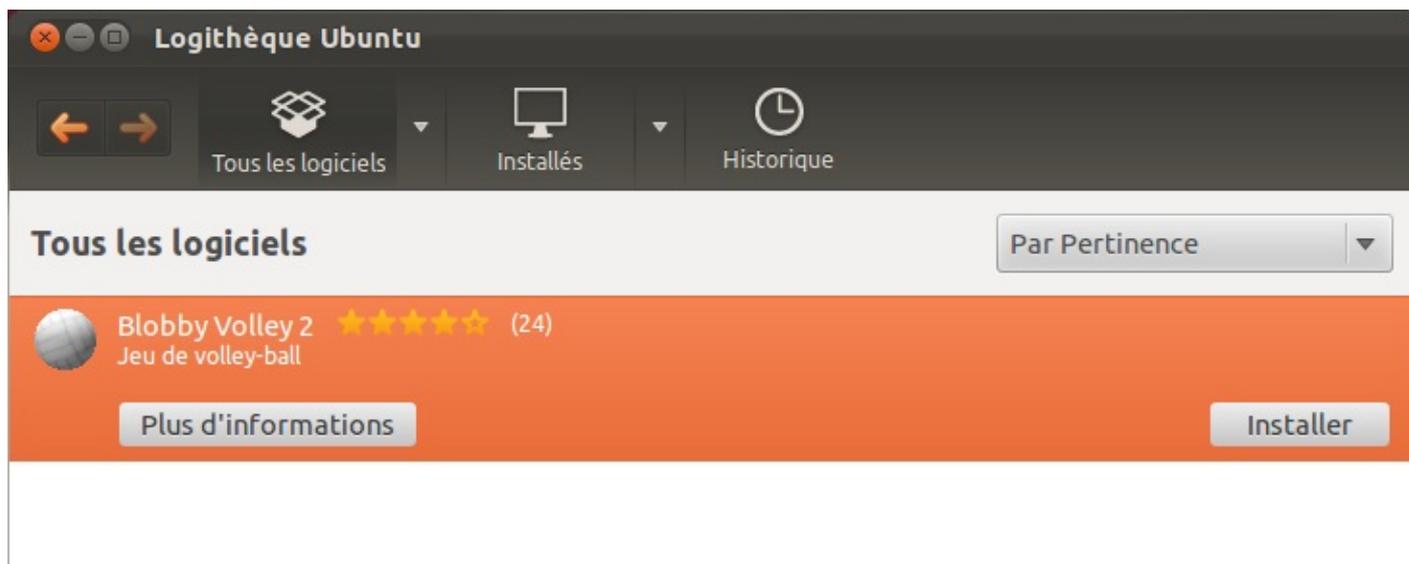


La logithèque Ubuntu est vraiment simple et agréable à utiliser. Elle fait assez penser à l'App Store des iPhone : les applications sont classées par catégorie, et vous pouvez les télécharger d'un simple clic.

La fenêtre principale vous présente une sélection d'applications, mais vous pouvez en trouver bien d'autres en faisant une recherche ou en parcourant les catégories à gauche. N'hésitez pas à en installer quelques-unes, vous y trouverez à coup sûr des programmes très intéressants.

En haut, le bouton "Installés" vous permet de parcourir les logiciels installés... et de les désinstaller si vous voulez.

Pour voir plus d'applications, revenez en arrière et sélectionnez une catégorie. Je vais par exemple aller y chercher le jeu Bloppy Volley 2 (vous pouvez aussi faire une recherche via le champ en haut à droite).



Cliquez sur le bouton "Installer". On vous demande à nouveau votre mot de passe par sécurité. C'est le mot de passe que vous utilisez pour vous connecter au lancement de Linux.



Le logiciel s'installe alors :



Il n'y a plus qu'à attendre que tout se fasse pour vous !

Mise à jour des programmes

Pouvoir ajouter et supprimer des programmes c'est bien, mais il faut aussi les mettre régulièrement à jour pour profiter des nouvelles fonctionnalités et, surtout, corriger les failles de sécurité qui sont parfois détectées.

Vous êtes automatiquement notifié dès qu'il y a des mises à jour disponibles. Il suffit de regarder la petite icône qui apparaît dans la barre Unity :



Cliquez dessus pour afficher le détail des mises à jour :



En temps normal, je vous conseille de ne pas vous prendre la tête et de tout laisser coché. Cliquez juste sur *Installer les mises à jour* et laissez le logiciel faire le reste 😊

Notre petit tour de Gnome s'arrête là, j'espère que la visite vous a plu. 😊

Dans le prochain chapitre, nous allons découvrir un autre célèbre gestionnaire de bureau : KDE. Vous verrez qu'il est assez différent car il propose de nombreuses fonctionnalités. Il ressemble d'ailleurs à première vue à Windows. 😊

Découverte du bureau KDE

Après avoir découvert Gnome, l'environnement de bureau par défaut d'Ubuntu, nous allons ici nous pencher sur KDE. Celui-ci est utilisé par défaut si vous installez Kubuntu (une variante d'Ubuntu) mais peut tout aussi bien être installé sous Ubuntu comme nous allons le voir.

L'objectif est avant tout de vous montrer la diversité qui règne dans le monde de Linux : celui-ci peut prendre plusieurs formes, selon le gestionnaire de bureau que l'on utilise. KDE est assez différent de Gnome, vous allez vite vous en rendre compte, mais il est très plaisant à utiliser. Ce chapitre sera l'occasion de vous faire une première idée de KDE qui est, avec Gnome, un des gestionnaires de bureau les plus utilisés et les plus célèbres.

Comment obtenir KDE ?

Pour essayer KDE, il faut avant toute chose l'installer. Tout dépend de votre cas :

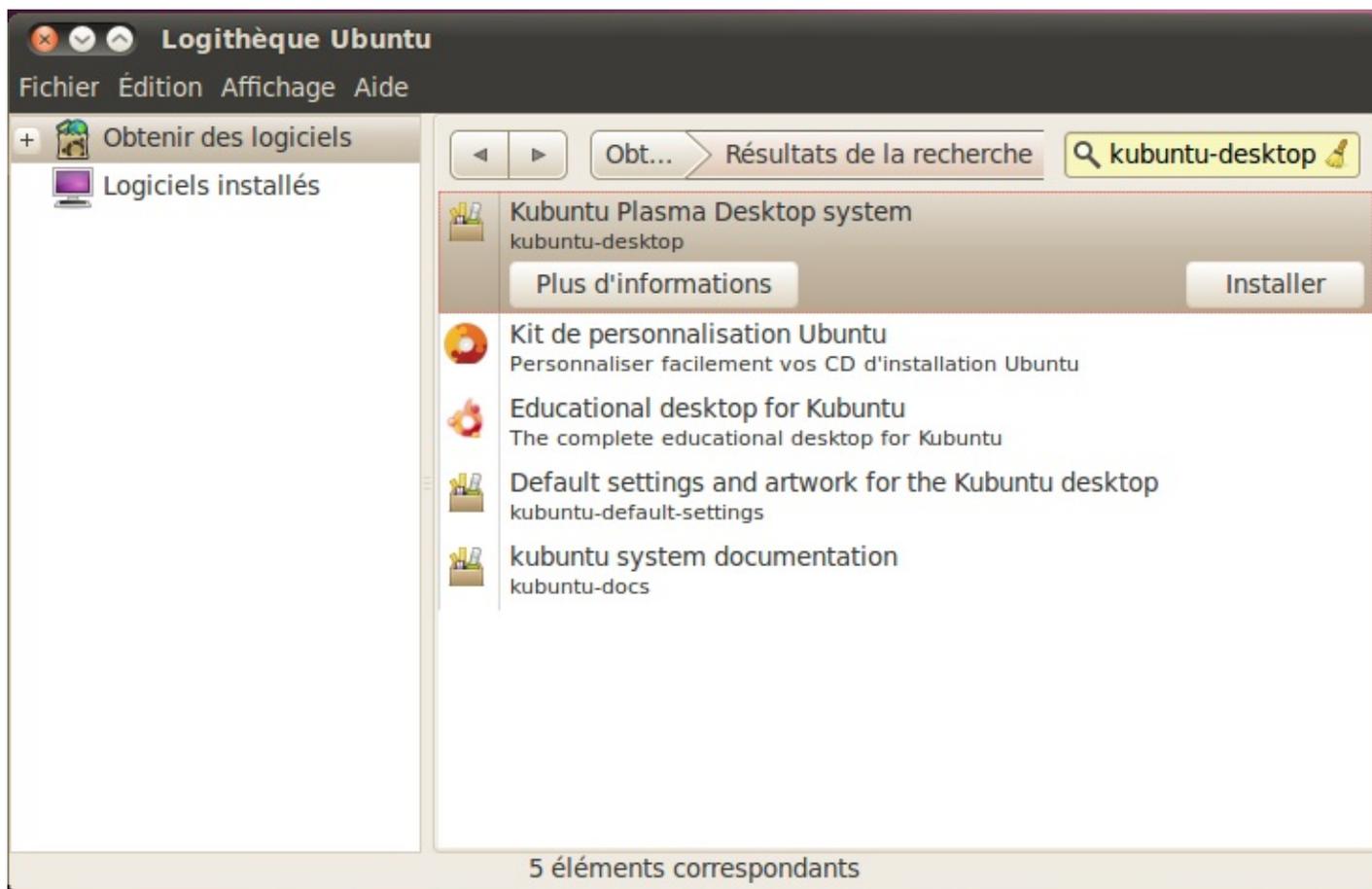
- Si vous avez téléchargé et installé Kubuntu (et non Ubuntu), KDE est installé par défaut. Vous n'avez donc rien à faire : vous serez automatiquement sous KDE !
- Si vous avez téléchargé et installé Ubuntu (ou Xubuntu), vous allez devoir installer KDE avant de continuer.

Si vous êtes sous Ubuntu et que vous voulez installer KDE, deux choix s'offrent à vous :

- **Installer le KDE "minimal"** : vous aurez KDE et les applications de base installées (navigateur, explorateur de fichiers). Pour cela, vous devez installer le programme `kde-minimal` (aussi appelé *L'environnement de bureau K, applications minimales*).
- **Installer KDE complet** : vous aurez KDE et toute une série d'applications dédiées à KDE. Pour cela, vous devez installer `kubuntu-desktop` (aussi appelé *Kubuntu Plasma Desktop System*).

Les applications de la version complète sont nombreuses et très intéressantes, quoiqu'elles font parfois doublons avec celles déjà installées. Dans les exemples qui vont suivre, je vais installer la version complète (équivalente à l'installation de Kubuntu) mais vous pouvez aussi installer la version minimale dans un premier temps si vous le désirez.

Ouvrez la logithèque Ubuntu comme vous avez appris à le faire : menu Applications / Logithèque Ubuntu. Dans le champ de recherche en haut à droite, tapez `kubuntu-desktop`, et installez le premier programme de la liste.



Patientez le temps de l'installation, qui peut être un peu plus longue que d'habitude. Une fois que c'est fait, vous devrez soit redémarrer votre ordinateur, soit vous déconnecter de votre session.

Connexion au bureau KDE

La suite de ce chapitre suppose soit :



- que vous avez installé Kubuntu (avec KDE par défaut) dès le début.
- que vous avez installé autre chose (Ubuntu, Xubuntu) mais que vous avez installé le paquet *kubuntu-desktop* ou *kde-minimal* comme expliqué précédemment pour avoir KDE.

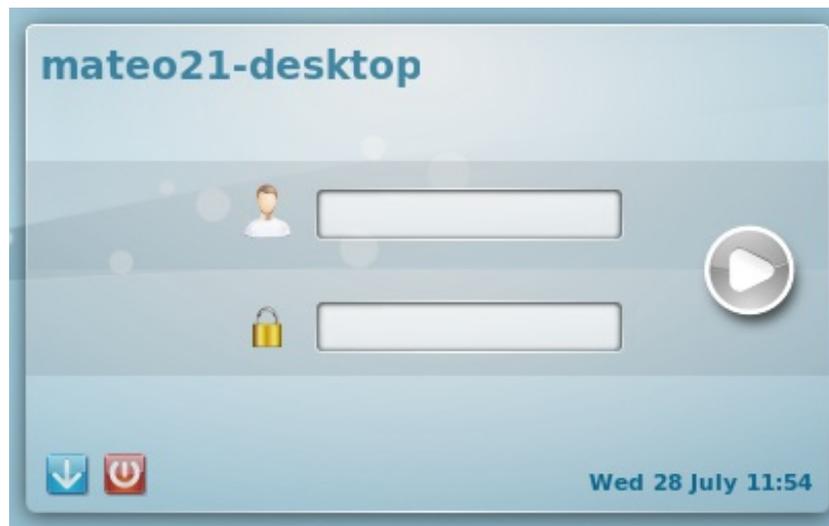
Pour accéder à KDE, tout dépend de votre cas :

- Si vous avez installé Ubuntu, puis que vous avez suivi la procédure indiquée précédemment pour installer KDE, vous devrez sélectionner KDE dans l'écran de login (voir chapitre précédent).
- Si vous avez installé Kubuntu, la fenêtre de login est différente, mais ce sera bien KDE qui sera lancé.

Je vais vous présenter l'interface de login de KDE, appelée KDM. Si vous avez toujours l'interface de login de Gnome, que nous avons découverte au chapitre précédent, sachez que cela ne change rien : vous pouvez très bien lancer KDE depuis l'interface de login de Gnome et vice-versa.

KDM, le programme de login de KDE (Kubuntu)

KDM est l'abréviation KDE Display Manager. C'est l'interface de connexion aux couleurs de KDE :



Si vous cliquez sur la petite flèche vers le bas, vous pouvez sélectionner le type de gestionnaire de bureau que vous souhaitez lancer. Vérifiez que "KDE" est bien sélectionné :

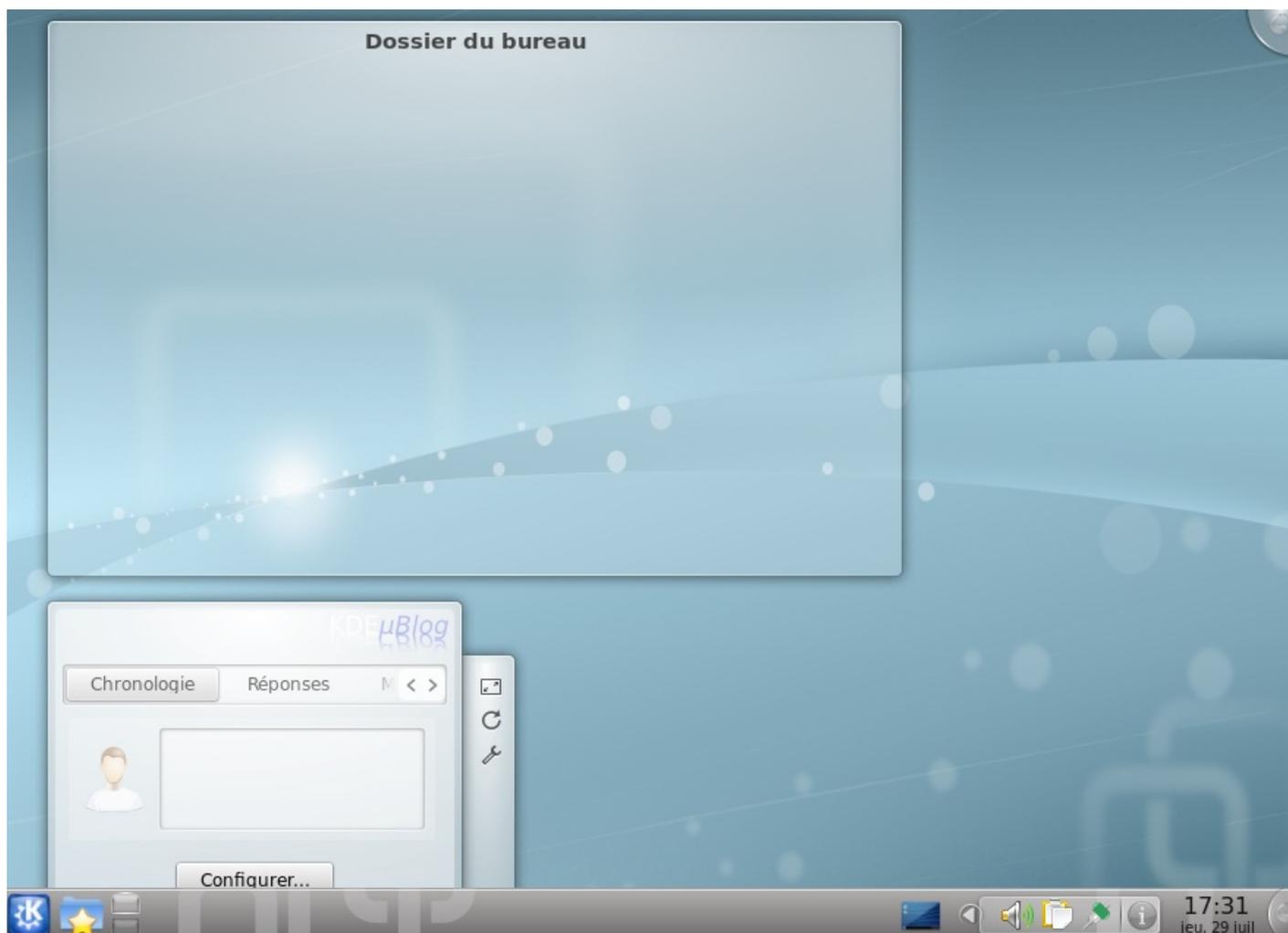


Entrez votre login et votre mot de passe : s'ils sont bons, KDE se charge alors.



Le bureau et le menu K

Après un petit instant de chargement, vous vous retrouvez sur le bureau de KDE :



Le bureau est assez riche au premier abord. En fait, il est complètement personnalisable : on peut y ajouter toute une variété de *widgets*, appelés "Plasmoides". Vous pouvez les configurer en cliquant sur l'icône tout en haut à droite de l'écran.

En bas de l'écran, on retrouve une barre des tâches qui rappelle en plusieurs points celle de Windows, vous ne devriez pas être trop dépaycé. Sous KDE, on l'appelle le **tableau de bord**.

Le tableau de bord

Le tableau de bord, normalement présent en bas de l'écran, est un outil complet qui vous permet de lancer vos applications, accéder à vos fichiers et visualiser l'état du système. Intéressons-nous dans un premier temps sur la partie gauche de ce tableau de bord.

Les boutons à gauche du tableau de bord

A gauche du tableau de bord, on retrouve les icônes suivantes :



Analysons le rôle de ces boutons, de gauche à droite.

Le menu K

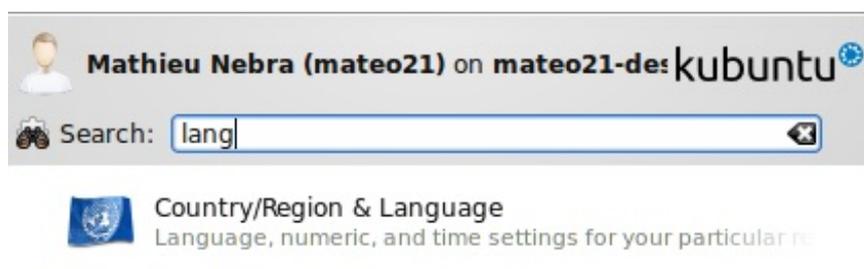
C'est le menu principal, le plus important de KDE :



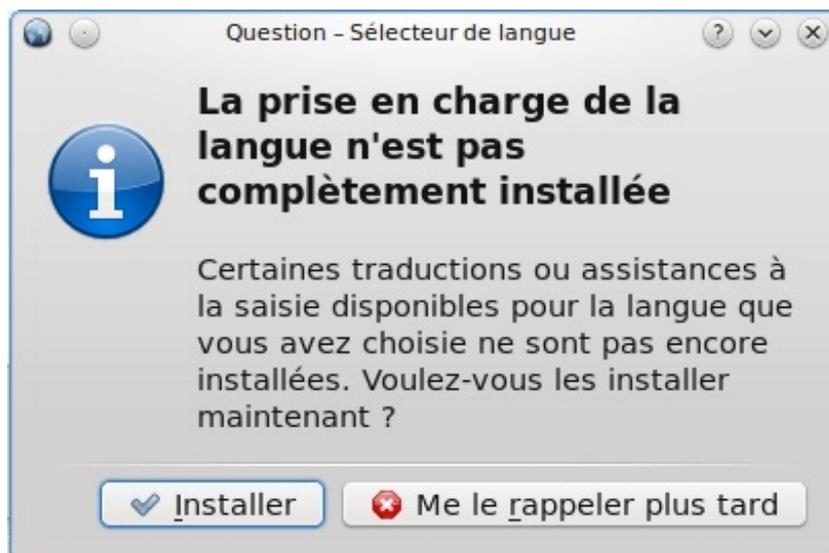
Il permet de lancer des applications, d'effectuer des configurations et d'arrêter l'ordinateur. Il ressemble à sa façon au menu Démarrer de Windows 7.

Le menu K s'ouvre sur vos applications favorites. Un clic droit sur une application vous permet de l'ajouter ou de la retirer de vos favoris pour un accès plus rapide.

Si KDE est en anglais, recherchez le programme "Country & Region Language". Vous pouvez tout simplement taper "lang" tandis que le menu K est ouvert :



Dans la fenêtre qui s'ouvre, cliquez sur "Select System Language" et installez les traductions lorsqu'on vous le demande :



Après un temps d'installation, on vous demandera de sélectionner votre langue : indiquez le français. 😊
Il faudra ensuite vous déconnecter et vous reconnecter à KDE pour que les changements soient pris en compte.

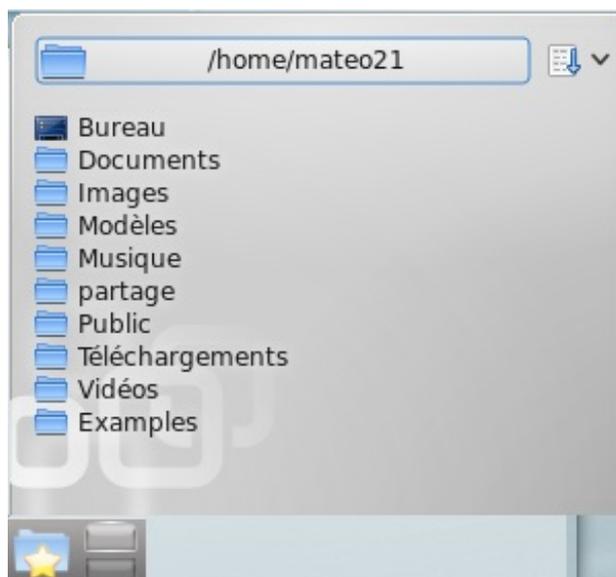
Continuez ensuite à explorer les programmes que vous pouvez lancer avec le menu K. Vous pouvez aussi accéder à votre poste de travail, vos documents récents, vous déconnecter, arrêter l'ordinateur... Je vous laisse découvrir. 😊

Jetez un oeil en particulier à :

- **Dolphin** : le programme qui permet de parcourir ses fichiers.
- **Konqueror** : le navigateur web (qui permet aussi de parcourir ses fichiers).

L'explorateur QuickAccess

Cette seconde icône du tableau de bord ouvre directement votre répertoire personnel ("Home"). Vous pouvez l'utiliser pour accéder rapidement à vos fichiers, sans passer par le programme Dolphin :



Le gestionnaire de bureaux virtuels

Comme sous Gnome, vous pouvez avoir plusieurs bureaux virtuels différents. Cela vous permet de mieux vous organiser si vous avez beaucoup de fenêtres ouvertes.



Par défaut ici, il n'y a que 2 bureaux virtuels, mais vous pouvez en ajouter (clic droit / configuration).

Les boutons à droite du tableau de bord

Passons maintenant à la droite du tableau de bord :



La première icône vous permet d'afficher à nouveau le bureau lorsque vous avez de multiples fenêtres ouvertes. Cela vous permet de réduire toutes les fenêtres d'un coup. Si vous cliquez une seconde fois, les fenêtres réapparaissent à nouveau.

Ensuite, une série d'icônes sont présentes dans ce qu'on appelle la *zone de notification*. Le principe est exactement le même que sous Windows : certains programmes qui tournent apparaissent ici et vous informent des événements en cours, du volume audio, de l'état de la batterie, etc.

Enfin, vous avez la date et, tout à droite, une petite icône qui vous permet de personnaliser complètement le tableau de bord.

Voilà pour ce rapide tour d'horizon du tableau de bord. Bien sûr, ces icônes peuvent changer d'une version à l'autre d'Ubuntu, et vous pouvez reconfigurer le tableau de bord comme bon vous semble. N'hésitez pas à le personnaliser jusqu'à ce que vous vous sentiez un peu plus "chez vous". 😊

L'explorateur de fichiers Dolphin

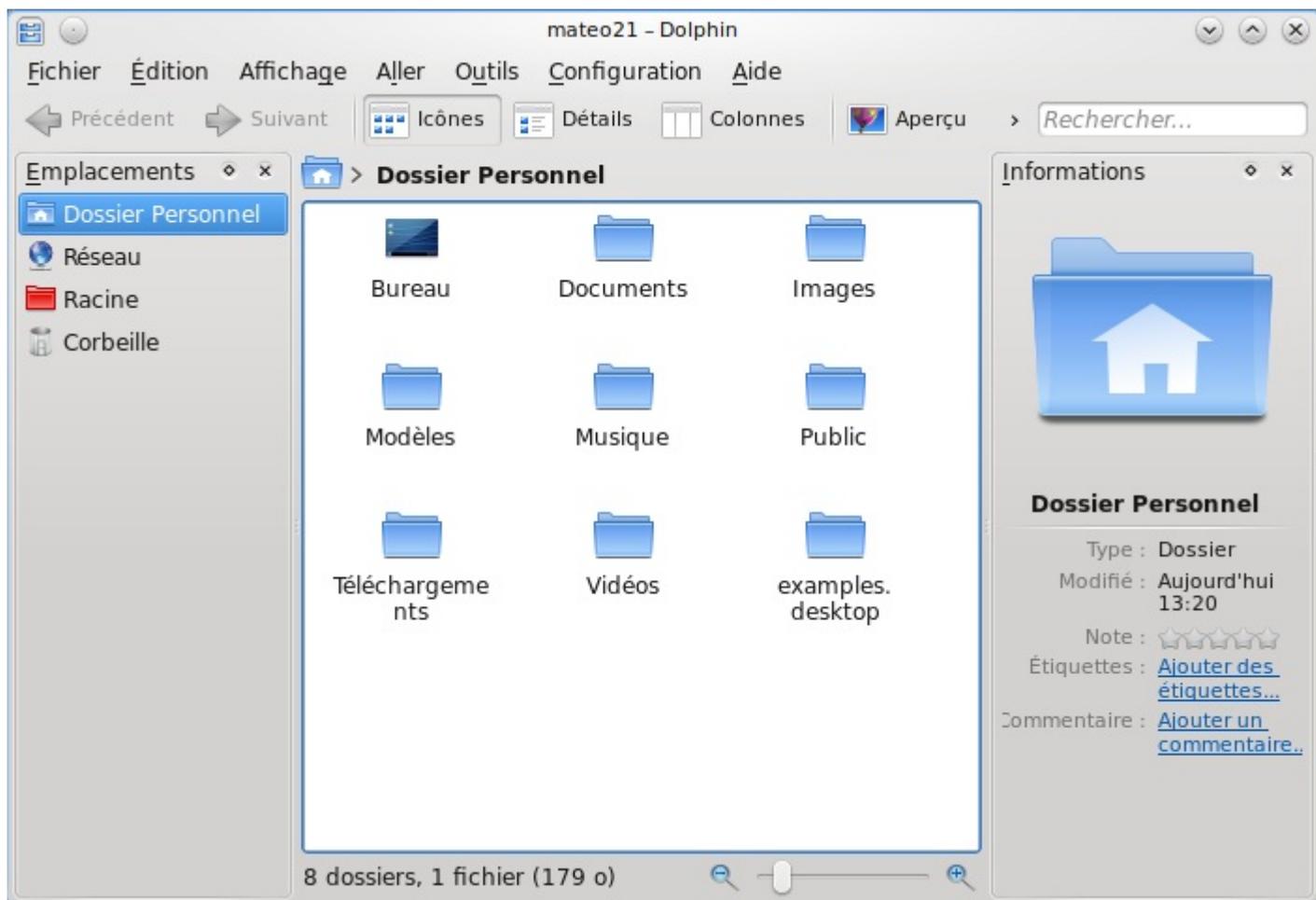
Pour ouvrir l'explorateur de fichiers, le programme qui vous permet de parcourir votre disque dur, vous avez plusieurs possibilités :

- Ouvrir le menu K et cliquer sur Dolphin dans la liste.
- Ouvrir le QuickAccess (à côté du menu K), sélectionner un dossier et cliquer sur le bouton "Ouvrir" en haut.



Notez que le navigateur web de KDE, Konqueror, peut aussi jouer le rôle d'explorateur de fichiers.

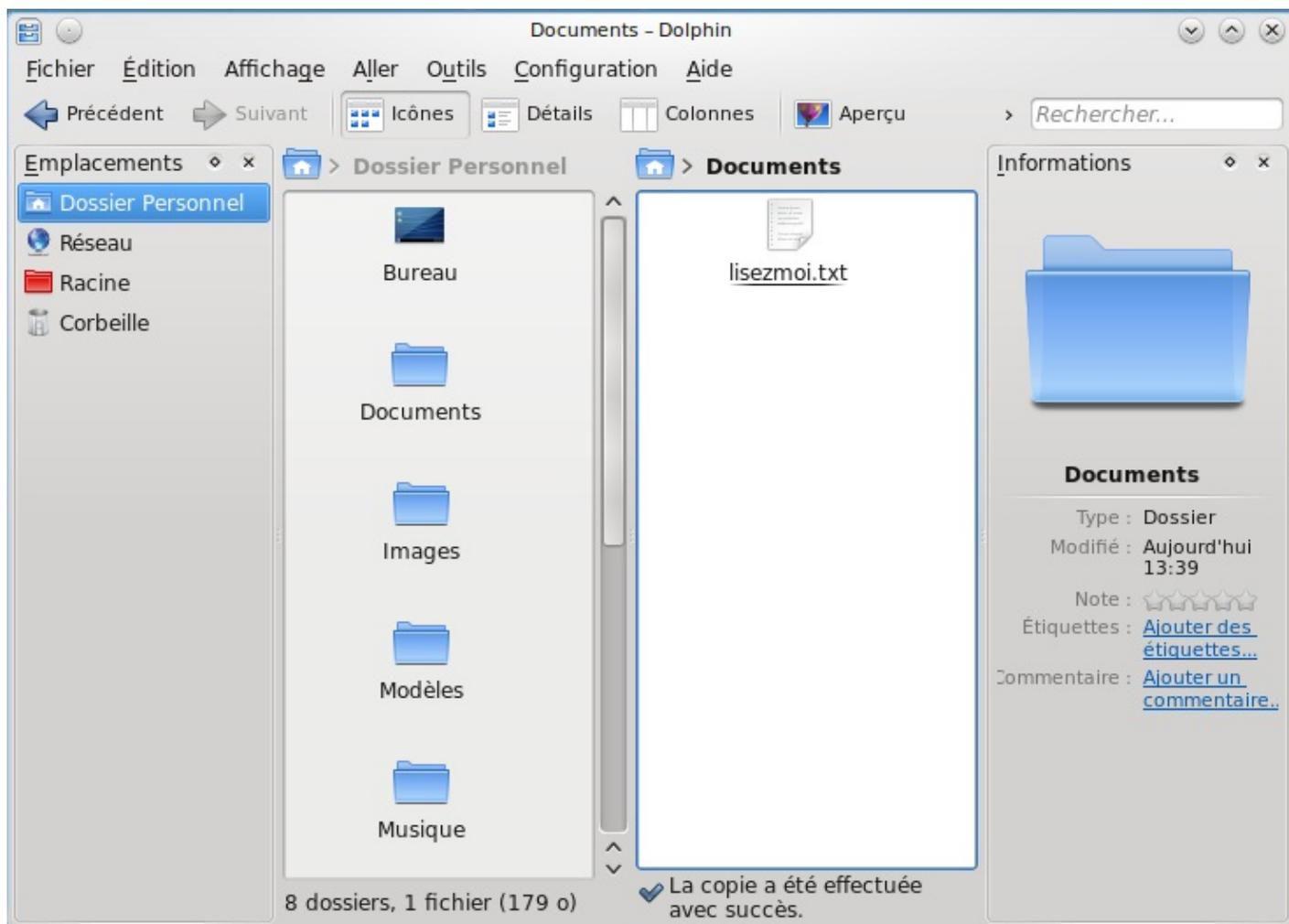
Dans un cas comme dans l'autre, la fenêtre de l'explorateur Dolphin s'ouvre :



Son fonctionnement n'est là encore pas très différent de Windows. Après quelques minutes de découverte, vous vous serez vite adaptés !

La principale particularité dans Dolphin (et KDE en général) est qu'il faut cliquer une seule fois pour ouvrir un fichier ou un dossier. Inutile donc de double-cliquer à tout va !

Vous pouvez ouvrir plusieurs onglets (comme dans le navigateur web) en effectuant la combinaison de touches Ctrl + T. Mieux encore, vous pouvez scinder la fenêtre en deux, pour voir simultanément deux dossiers différents ! Pour cela, vous pouvez appuyer sur la touche F3 ou cliquer sur le bouton "Scinder" en haut dans la barre d'outils.



Ainsi, vous pouvez facilement déplacer ou copier des fichiers d'un dossier à l'autre !

Ces particularités mises à part, vous trouverez vite vos marques sous KDE, n'ayez crainte. 😊



Je vous invite maintenant à ouvrir la "Configuration du système" (équivalent du panneau de configuration de Windows) qui vous permettra de personnaliser au mieux votre KDE. Pour l'ouvrir, vous savez ce qu'il vous reste à faire : utilisez le menu K !



Si vous souhaitez installer des programmes sous KDE, lancez le programme "Ubuntu Software Center". C'est la logithèque Ubuntu que nous avons découverte sous Gnome.

Et voilà pour ce *rapide* tour d'horizon de KDE. Vous l'aurez compris, on pourrait y passer plusieurs chapitres qu'on n'aurait toujours pas vu la moitié des programmes et des fonctionnalités.

KDE propose un environnement très riche, il ne faut pas espérer tout connaître sur le bout des doigts en quelques jours. Ce qui compte, c'est de comprendre sa logique, savoir manipuler les logiciels de base comme Dolphin, être capable d'en installer de nouveaux ou d'en désinstaller.

Voilà la fin de la partie I du cours ! Félicitations !

Vous venez de faire vos tous premiers pas dans Linux avec la distribution Ubuntu. Vous avez entrevu l'histoire de Linux et sa philosophie, puis vous avez vu la procédure pour l'installer. Enfin, vous venez de découvrir 2 de ses plus célèbres gestionnaires de bureau : Gnome et KDE.

Je vous encourage à en essayer d'autres, histoire de faire un peu plus le tour des environnements graphiques que Linux vous propose, comme XFCE (et il y en a bien d'autres, si vous êtes curieux vous en trouverez plein !).



Et maintenant, ça s'arrête là ? 🤔

Oh que non, pauvre fou ! Croyez-moi, les festivités ne font que commencer... 😈

Partie 2 : Manipuler la console et les fichiers

Un écran noir, quelques symboles blancs indéchiffrables, un curseur qui clignote... Ca a pas l'air accueillant la console hein ? 😬
Et pourtant ! La console est LE point nerveux central de Linux, c'est en grande partie ce qui lui confère toute sa puissance. C'est un monde très riche et passionnant (puisque je vous le dis !).

Seul défaut : ce n'est vraiment pas intuitif au tout début... sauf si vous lisez ce tuto en même temps bien sûr 😊

📁 La console, ça se mange ?

Nous avons passé la plupart de la première partie du cours à découvrir Linux en douceur, son bureau et son interface graphique. Par rapport à d'autres systèmes d'exploitation comme Windows, c'est un peu dépayasant au début, mais mine de rien on retrouve beaucoup de concepts similaires. Je pense donc que vous n'aurez pas plus besoin de moi pour être capable de manipuler correctement l'interface graphique.

J'aurais très bien pu partir *vraiment* de zéro et vous expliquer comment faire pour fermer une fenêtre, mais bon, je pense que vous m'en auriez plus voulu qu'autre chose 😬

J'ai donc volontairement choisi un juste milieu et considéré que vous connaissiez un tout petit peu Windows, ce qui ne me semble pas être trop fou de ma part.

En fait, les choses intéressantes commencent maintenant. C'est à partir d'ici qu'un utilisateur classique de Windows met les pieds dans un environnement totalement nouveau. Vous ne pouvez pas avoir d'a priori, et il y a de fortes chances que ce soit un domaine de l'informatique que vous n'avez jamais approché (non non, DOS ça ne compte pas 😬).

Vous n'avez pas idée de la richesse offerte par la console, qui est quasi-infinie. Personne ne peut d'ailleurs prétendre la maîtriser entièrement, c'est vous dire ! Vous aurez donc toujours quelque chose à découvrir 😊

Pourquoi avoir inventé la console ?

Avant de vous lancer à corps perdus dans l'océan de la console, ce chapitre va vous enseigner les rudiments de survie pour éviter la noyade. Parce que, bon, ça serait dommage que vous vous arrêtiez avant le meilleur moment 😬

On va commencer par répondre à cette question hautement fondamentale :



Mais pourquoi ont-ils inventé la console d'abord au lieu de l'interface graphique ? C'est quand même plus pratique une interface graphique avec une souris, c'est plus intuitif ! C'est juste pour faire pro, faire compliqué pour faire compliqué et s'assurer que l'informatique reste seulement à la portée de quelques initiés ? Pourquoi ne pas avoir supprimé la console, c'est archaïque !

(que de questions dites-donc !)

Que nenni ! Il y a une explication à tout, voici une réponse question par question :

- **Pourquoi avoir inventé la console d'abord, au lieu de l'interface graphique ?** Pour ça, je vous ai mis la puce à l'oreille dès le premier chapitre. La réponse est : *parce qu'on n'avait de toute façon pas le choix !* Les débuts de l'informatique et de la console remontent aux débuts des années 70, à une époque où un écran 2 couleurs était un luxe inimaginable et où la puissance de calcul de ces ordinateurs était 100 fois plus faible que celle de la calculatrice Casio de ma petite sœur. Bref, vous voyez le genre.

On dirait pas comme ça, mais gérer une interface graphique avec plusieurs couleurs, ainsi qu'une souris et un certain nombre de fonctionnalités avancées qui vous paraissent aujourd'hui "normales", ça demande de la puissance ! La console était donc à cette époque la seule façon d'utiliser un ordinateur.

```
apt          hostname    motd        tmp
bash.bashrc  hosts      motd-oid    rpc
bash_completion  hosts.allow  motd.tail   scss_id.config
bash_completion.d  hosts.deny  ncab        security
calendar     inetd.conf  nano        security
console      init.d      Net         services
console-tools  initramfs-tools  network    shadow
cron.d       inittab     networks   shadow
cron.daily   isupitre   nsswitch.conf  shells
cron.hourly  issue      apt         ssh
cron.monthly issue.net  pam.conf    ssh
crontab     kernel-img.conf  pam.d      sudoers
cron.weekly  ldap       passwd     systemctl.conf
debconf.conf  ld.so.cache  passwd-    syslog.conf
debian_version  ld.so.conf  perl       terminfo
default      ld.so.conf.d  postgresql-common  timezone
deluser.conf  ld.so.bwcappkgs  profile    udev
dhcp3        locales.gen  protocols  updatedb.conf
dpkg         localtime   rsync      vim
environment  logcheck    rsync      wgetrc
fstab        login.defs  rsync      xlibs.conf
groff        logrotate.conf  rsync      xorg.conf
group        logrotate.d  rsync      xorg.conf
lisa:/etc/
```

- **L'interface graphique avec la souris c'est quand même plus intuitif !** Alors là, tout à fait d'accord avec vous. On dira ce

qu'on voudra, mais la console n'est PAS intuitive. Quand on débute en informatique, il est de loin plus simple d'appréhender l'interface graphique. Par contre, je suis aujourd'hui persuadé que l'interface graphique de Linux (que ce soit KDE, Gnome ou une autre) est aussi intuitive que celle de Windows et de Mac OS. Ça n'a pas toujours été forcément le cas, mais un débutant total en informatique n'aura pas plus de mal à appréhender l'interface graphique de Linux que celle de Windows, ça j'en suis totalement convaincu.

- **Est-ce que c'est juste pour faire pro, inutilement compliqué ?** Les commandes de la console vont peut-être vous paraître du chinois les premiers temps, et vous allez vous demander à coup sûr si cet amas de lettres vide de sens n'est pas là juste pour faire en sorte que le moins de monde possible puisse utiliser la console (sous-entendu "*Seuls les programmeurs qui ont inventé la console devraient pouvoir l'utiliser*"). Ça par contre c'est totalement faux. Tout a été minutieusement pensé, et ce dès les années 60 :
 - **Les commandes sont courtes, abrégées ?** C'est pour gagner du temps et aller plus vite. Ecrire `pwd` est moins intuitif que `dire dans quel repertoire je suis`, mais après l'avoir écrit 200 fois dans la journée vous bénirez les programmeurs qui ont fait ce choix, croyez-moi !
 - **Les commandes ne sont pas intuitives ?** Faux, bien souvent il s'agit d'une abréviation de termes (en anglais, *of course* !). Et bien souvent, les lettres qu'il faut taper sont choisies en fonction de leur proximité les unes par rapport aux autres pour que vous ayez le moins possible à déplacer les doigts sur le clavier ! Bon, d'accord, à la base c'est plutôt fait pour les claviers QWERTY anglais qui sont, je le reconnais, plus adaptés pour accéder aux symboles du genre `{ } | #` etc. Mais vous n'en mourrez pas 😊
- **Pourquoi ne pas avoir supprimé la console, c'est archaïque !** Depuis l'invention de l'interface graphique, on pourrait se demander pourquoi on n'a pas supprimé la console. Sous-entendu : *elle ne sert plus à rien*. Et c'est là que vous vous trompez complètement : on met un petit temps à s'y faire, mais quand on sait s'en servir on va beaucoup plus vite avec la console qu'avec l'interface graphique. C'est même pire en fait : vous vous rendrez compte à un moment qu'il y a des choses que seule la console peut faire, et qu'il serait pour d'autres vraiment inutile d'en faire une interface graphique.

Un exemple ? En mode graphique, allez par exemple dans un répertoire qui contient beaucoup de fichiers en tous genre : des fichiers texte, des images, des vidéos... Vous voudriez savoir combien il y a d'images JPEG dans ce dossier.. Pas facile hein ? 😊

En console, en assemblant quelques commandes, on peut obtenir ce résultat sans problème !

Code : Console

```
ls -l | grep jpg | wc -l
510
```

La première ligne est la commande que j'ai tapée, la seconde le résultat. Il y avait donc 510 images JPEG dans le dossier, et on a obtenu le résultat en moins d'une seconde !

On peut même faire encore plus fort et enregistrer directement ce nombre dans un fichier texte :

Code : Console

```
ls -l | grep jpg | wc -l > nb_jpg.txt
```

... et on peut aussi envoyer le fichier `nb_jpg.txt` sur internet par FTP ou à un ami par e-mail, le tout en une ligne ! La console n'est pas morte donc, et n'a pas du tout prévu de l'être !

La plupart des commandes de la console de Linux sont des "copies" d'Unix, ce vieil OS dont je vous ai parlé au tout début, ancêtre parmi les ancêtres. N'allez pas croire que les programmes d'Unix ont été copiés ou "piratés" par Linux, c'est juste que leur mode d'emploi est le même. Les programmes ont été réécrits par un groupement de programmeurs, issus de ce qu'on appelle le projet GNU.

Ce projet a fusionné au bout de quelques temps avec le coeur du système d'exploitation Linux pour donner au final GNU/Linux, qu'on écrit en pratique juste "Linux" car c'est plus court. Mais tout ça je vous l'ai déjà dit dans le premier chapitre 😊

L'avantage ? Les commandes n'ont pas bougé et ne bougent pas depuis l'époque d'Unix (soit depuis les années 60). Ce sont les mêmes. Quelqu'un qui utilisait Unix dans les années 60 est capable de se débrouiller avec un Linux d'aujourd'hui. Et il y a fort à parier que ce sera pareil pour les nombreuses années à venir. Vous avez donc juste à apprendre à vous en servir une fois, ok il y

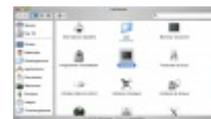
aura du boulot, mais après c'est quelque chose qui pourra vous servir toute votre vie !

Ce que vous apprendrez dans les chapitres suivants sera l'utilisation de commandes de type Unix. L'avantage, c'est que ça ne marchera pas seulement sous Linux, mais aussi sous tous les OS aussi basés sur Unix !



En effet à quelques exceptions près, tout ce que vous verrez pourra donc être fait sous les OS basés sur Unix, en particulier Mac OS X.

Si vous avez Mac OS X et que vous souhaitez ouvrir une console, faites Pomme + Shift + U pour ouvrir les Utilitaires, et sélectionnez "Terminal" comme le montre le screenshot ci-contre (merci à Makkhdyn pour le screenshot d'ailleurs 😊).



La console, la vraie, celle qui fait peur

Bien, il s'agirait maintenant de savoir comment approcher la Bête. La question est donc : comment accède-t-on à la console dans son beau Linux depuis son KDE / Gnome / XFCE / (insérez le nom de votre gestionnaire de bureau ici) ?

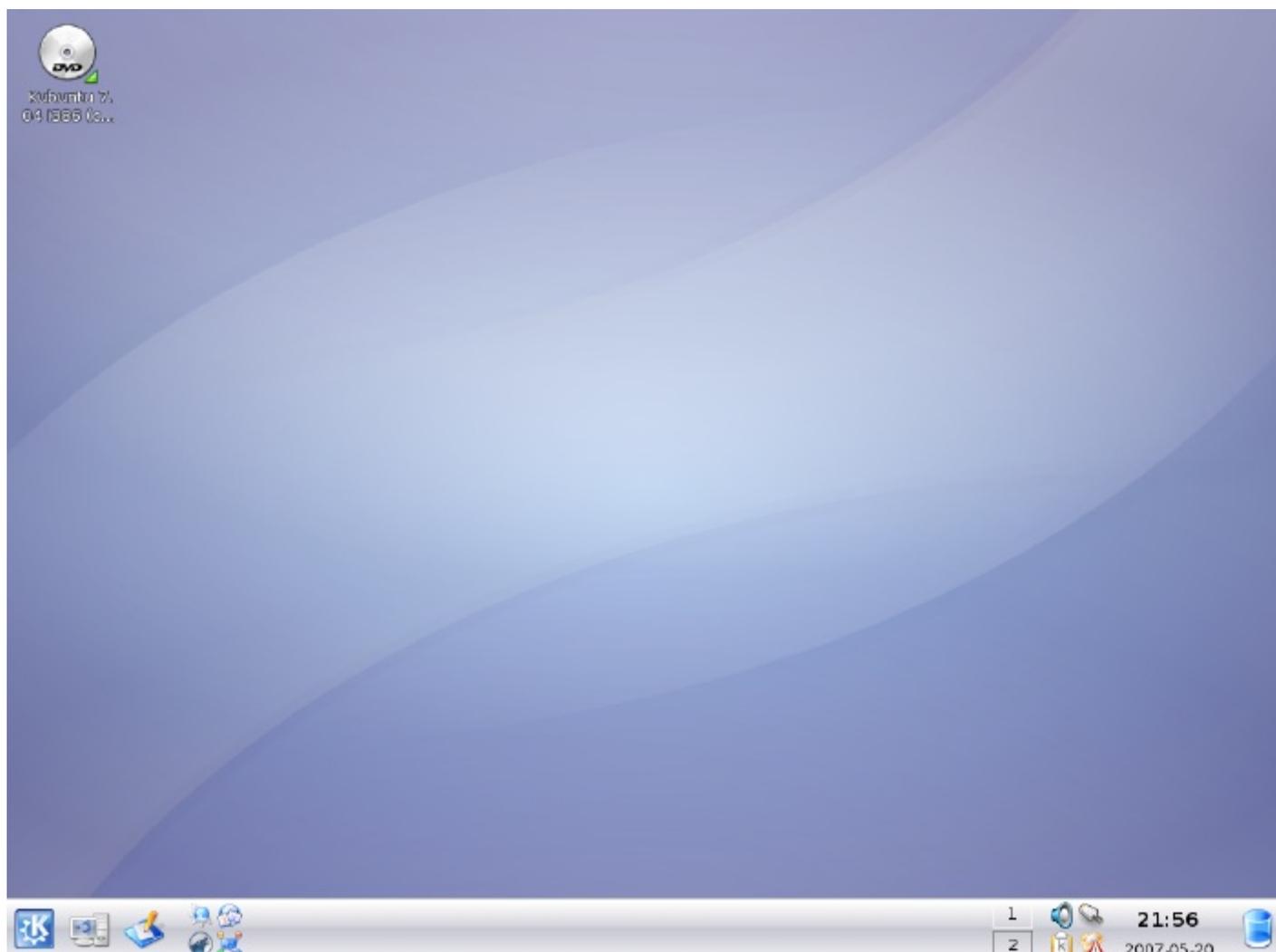
Les moyens sont variés, très variés. Il y a donc le choix, un peu comme partout sous Linux vous me direz 😊

Nous allons commencer par la solution la plus "basique", que vous utiliserez probablement le moins souvent. Elle vous permet d'accéder à la vraie console (si tant est qu'il y ait une "vraie" console) en pressant une combinaison de touches.



Lisez bien tous les prochains paragraphes avant d'exécuter les commandes que je vais vous donner. Ce n'est pas dangereux rassurez-vous, mais c'est juste que si vous vous retrouvez en console avant d'avoir lu comment en sortir, vous serez bien embêté 😬

Je vais supposer que vous vous êtes connecté, c'est-à-dire que vous avez rentré votre login et votre mot de passe. Vous êtes donc sur votre gestionnaire de bureau (ici KDE) :



Vous trouvez qu'il y a trop de couleurs ? Que ça manque de mots compliqués ?
Pas de problème ! Voici les raccourcis à connaître pour accéder à la console :

- Ctrl + Alt + F1 : terminal 1 (tty1)
- Ctrl + Alt + F2 : terminal 2 (tty2)
- Ctrl + Alt + F3 : terminal 3 (tty3)
- Ctrl + Alt + F4 : terminal 4 (tty4)
- Ctrl + Alt + F5 : terminal 5 (tty5)
- Ctrl + Alt + F6 : terminal 6 (tty6)
- Ctrl + Alt + F7 : retour au mode graphique (ouf !)



Terminal est un autre nom pour "Console".



Attention, pensez bien, si vous testez, que vous serez alors en mode console. Vous devrez donc utiliser Ctrl+Alt+F7 pour revenir en mode graphique. N'oubliez pas !

Pour tester, tapez Ctrl + Alt + F1 pour voir 😊

Votre écran va peut-être clignoter quelques instants, ne paniquez pas. Vous allez ensuite voir ceci :

```
Starting up ...
Loading, please wait...
kinit: name_to_dev_t(/dev/disk/by-uuid/890a63fb-b2e0-4e1f-841c-b176acae2e27) = s
da5(8,5)
kinit: trying to resume from /dev/disk/by-uuid/890a63fb-b2e0-4e1f-841c-b176acae2
e27
kinit: No resume image, doing normal boot...

Ubuntu 7.04 mateo21-desktop tty1
mateo21-desktop login:
```

Vous êtes en plein écran en mode console. C'est beau hein ? 😊

Le login

Vous ne pouvez pas utiliser de suite la console : il faut d'abord vous logger. C'est ce que vous demande la dernière ligne :

Code : Console

```
mateo21-desktop login:
```

mateo21-desktop : c'est le nom que vous avez donné à votre ordinateur lors de l'installation. Votre ordinateur se présente et vous rappelle où vous êtes en quelque sorte.

Ça a l'air inutile comme ça, mais avec Linux on peut se connecter à un autre PC facilement en console (on en parlera plus loin) et parfois on ne sait plus si on est dans la console de son PC ou de celle d'un autre PC 😊

Bon, rentrez votre login, dans mon cas c'est mateo21.

On vous demande ensuite votre mot de passe :

Code : Console

```
Password:
```

Là, vous rentrez votre mot de passe, dans mon cas c'est ~~oops~~ euh j'ai rien dit 🤔

Ne soyez pas étonné si les lettres que vous tapez n'apparaissent pas. En fait, il n'y a même pas d'étoiles qui apparaissent à l'écran. Ça permet d'éviter qu'une personne derrière vous compte le nombre de caractères de votre mot de passe. Euh oui, sous Linux on est un peu parano 😁

Si tout va bien, vous devriez voir l'écran suivant :

```
Starting up ...
Loading, please wait...
kinit: name_to_dev_t(/dev/disk/by-uuid/890a63fb-b2e0-4e1f-841c-b176acae2e27) = s
da5(8,5)
kinit: trying to resume from /dev/disk/by-uuid/890a63fb-b2e0-4e1f-841c-b176acae2
e27
kinit: No resume image, doing normal boot...

Ubuntu 7.04 mateo21-desktop tty1

mateo21-desktop login: mateo21
Password:
Last login: Sat Aug 18 17:44:52 2007 on :0
Linux mateo21-desktop 2.6.20-16-generic #2 SMP Thu Jun 7 20:19:32 UTC 2007 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
mateo21@mateo21-desktop:~$ _
```

La console devrait afficher en bas une ligne similaire à :

Code : Console

```
mateo21@mateo21-desktop:~$ _
```

C'est bon, vous y êtes 😊

Les différentes consoles

Sous toute machine Linux, il y a donc non pas une mais 6 consoles qui fonctionnent en simultané (d'où les 6 raccourcis différents de Ctrl + Alt + F1 à Ctrl + Alt + F6).

Vous pouvez savoir dans quel terminal vous êtes lors du chargement. Il y a en effet marqué "tty1" si vous êtes sur le terminal n°1, regardez de plus près mon screenshot de tout à l'heure :

```

Starting up ...
Loading, please wait...
kinit: name_to_dev_t(/dev/disk/by-uuid/890a63fb-b2e0-4e1f-841c-b176acae2e27) = s
da5(8,5)
kinit: trying to resume from /dev/disk/by-uuid/890a63fb-b2e0-4e1f-841c-b176acae2
e27
kinit: No resume image, doing normal boot...

Ubuntu 7.04 mateo21-desktop tty1

mateo21-desktop login: mateo21
Password:
Last login: Sat Aug 18 17:44:52 2007 on :0
Linux mateo21-desktop 2.6.20-16-generic #2 SMP Thu Jun 7 20:19:32 UTC 2007 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
mateo21@mateo21-desktop:~$ _

```

L'information est un peu cachée mais elle est là 😊

Au pire, vous changez de terminal jusqu'à retrouver celui où vous êtes. Et dès que vous en avez marre, vous pouvez retourner au mode graphique avec **Ctrl + Alt + F7**.

A noter qu'une combinaison similaire permet de redémarrer le serveur X, c'est-à-dire l'interface graphique. Contrairement aux apparences, ça ne redémarre pas l'ordinateur mais juste l'élément graphique. Les consoles continuent à tourner derrière.

Cette combinaison spéciale, c'est **Ctrl + Alt + Backspace** (la touche "Retour arrière"). Attention c'est assez radical, donc pensez à enregistrer vos documents avant d'essayer 😊



Vous ne devriez pas avoir à le faire souvent, sauf si l'interface graphique est plantée (rare) ou si on vous demande de le faire pour prendre en compte une nouvelle configuration de X, mais je vous dirai ça en temps voulu.

Notez enfin que sous les dernières versions d'Ubuntu, ce raccourci pourtant commun a été remplacé par **Alt + Impr.Écran + K**. La combinaison **Ctrl + Alt + Backspace** a été considérée comme trop fréquemment utilisée de manière inappropriée par les débutants habitués à une combinaison similaire présente sous Windows.

La console en mode graphique

Bon, la console en plein écran c'est bien joli mais pas franchement folichon. Je voulais vous montrer ça parce que c'est la vieille technique qui marche partout, même quand il n'y a pas d'interface graphique installée (c'est le cas sur la plupart des serveurs tournant sous Linux par exemple).

Mais dans le cas qui nous intéresse, c'est-à-dire le vôtre, croyez-moi il y a bien mieux. Même les plus fous de Linux préfèrent ouvrir une console dans le mode graphique. Les raisons sont multiples :

- Déjà parce que c'est plus sympa et que vous pouvez en même temps continuer à utiliser d'autres applications graphiques, tout en discutant avec vos amis en ligne...
- Mais aussi parce que la résolution de l'écran est plus grande et qu'on peut afficher plus de choses à la fois dans la console...
- Parce qu'on peut personnaliser l'apparence de la console et mettre, pourquoi pas, une image de fond...
- Et qu'on peut aussi utiliser la souris pour copier / coller du texte dans la console (comme quoi, la souris sert quelques rares fois en console !).

Je vous conseille donc fortement d'utiliser cette console en mode graphique autant que possible, notamment pendant toute la durée du cours. C'est juste plus confortable 😊

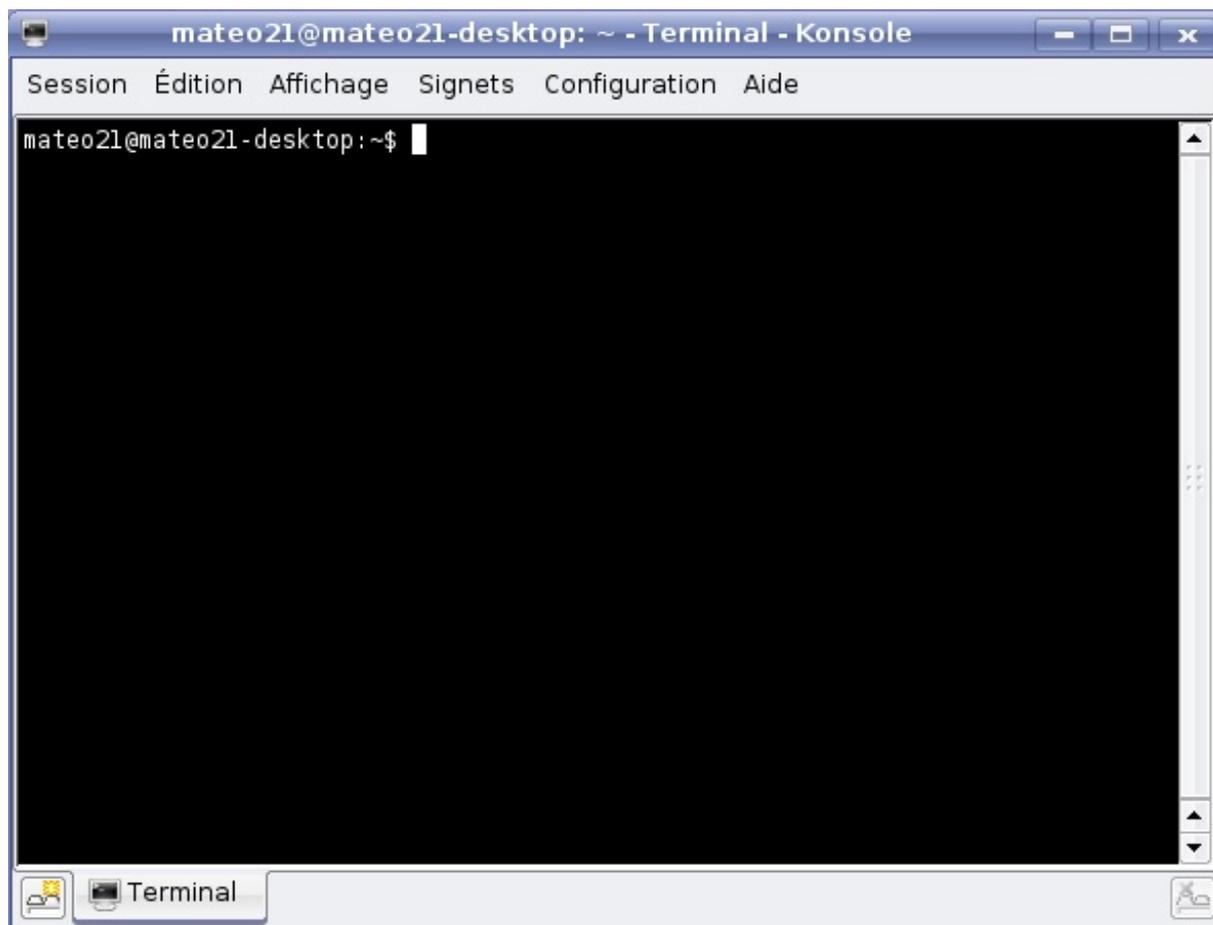
Comment accéder à la console en mode graphique ? Tout dépend de votre gestionnaire de bureau, chacun propose un programme de console différent (mais tous se valent globalement).

Sous KDE

Pour KDE, il faut se rendre dans le menu K / Système / Konsole :



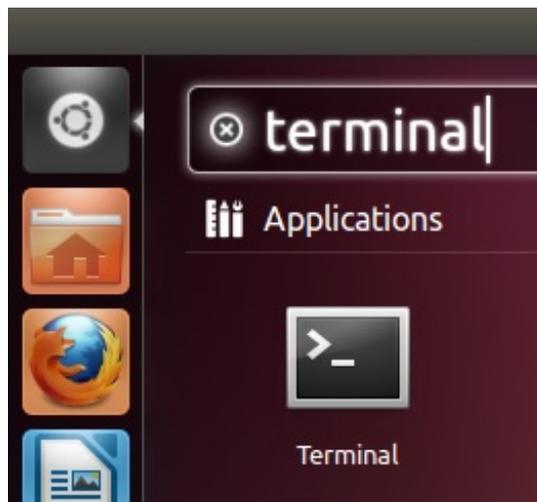
La console de KDE ressemble à ceci :



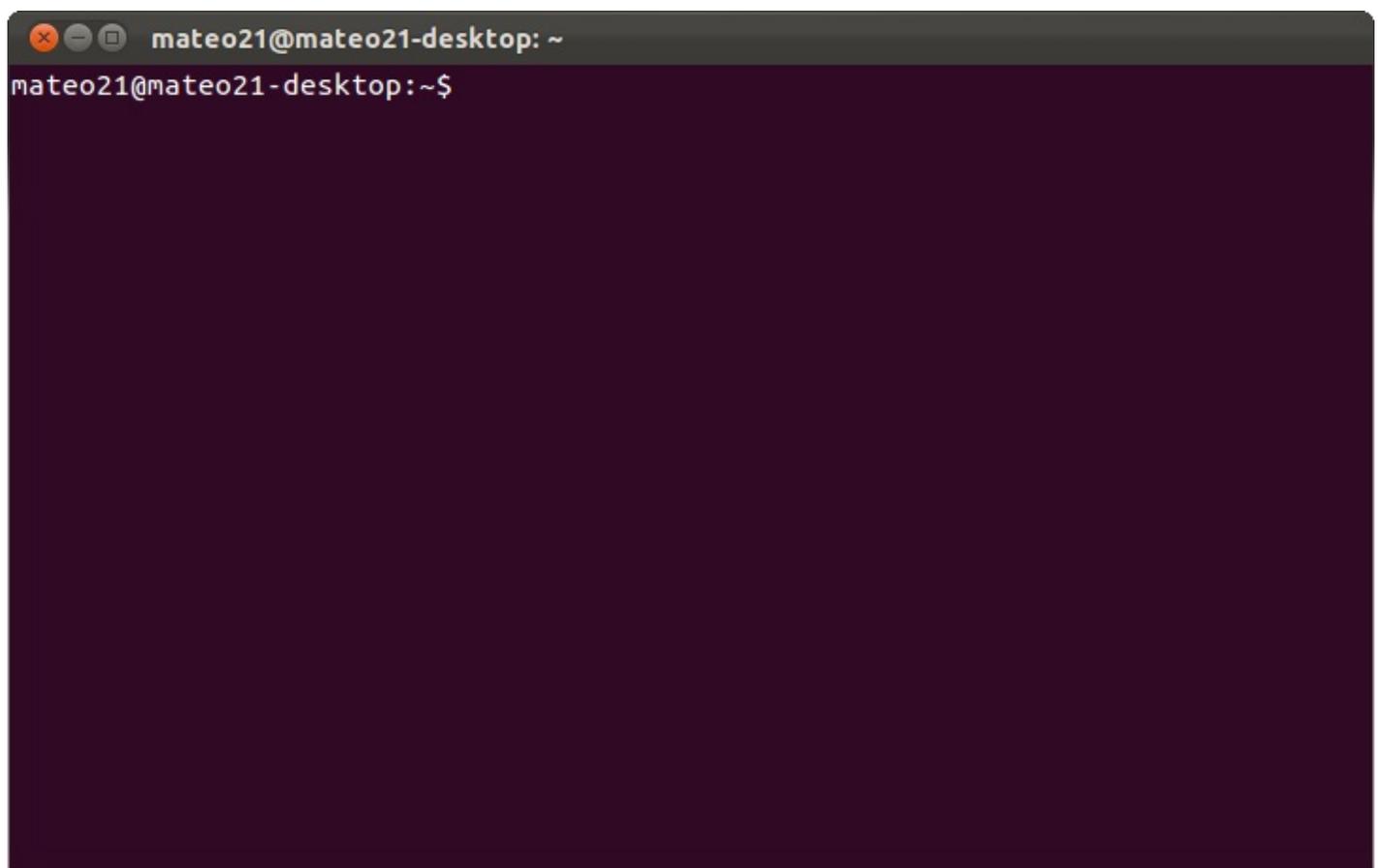
Vous pouvez vous amuser à changer la couleur de fond ou à mettre une image de fond, vous trouverez ça quelque part dans les menus. Personnellement, je suis un adepte de la console en blanc sur fond noir, le plus sobre et le plus agréable pour une utilisation hmmm... prolongée 🤔

Sous Gnome

Si vous êtes sous Gnome, rendez-vous dans le tableau de bord et tapez "Terminal" pour pouvoir cliquer sur l'icône du Terminal :



Et voilà la console de Gnome :



Vous noterez que les 2 types de console permettent d'ouvrir plusieurs onglets. Pour celle de KDE, il y a une petite icône en bas à gauche, et pour celle de Gnome il y a le menu *Fichier / Ouvrir un onglet*.

Les onglets sont très pratiques en général car ils permettent de multiplier les consoles et donc de faire plusieurs choses en même temps.

L'accès à distance en SSH avec Putty

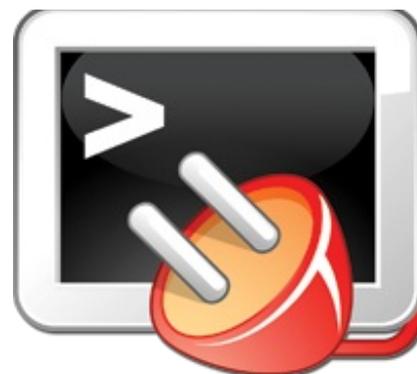


Cette partie sur l'accès à distance en SSH avec Putty ne sert qu'à vous montrer les possibilités d'utilisation de la console. N'essayez pas de faire ça pour le moment, car il y a des détails un peu compliqués.

En clair : lisez ce que j'ai à vous dire, c'est pour votre culture. Pour que vous sachiez que ça existe. On verra SSH en pratique un peu plus tard dans le cours.

Une des grosses forces de la console, c'est d'être accessible à distance par internet. Il

suffit que votre machine soit connectée à internet pour que vous puissiez vous logger de n'importe quel ordinateur dans le monde et faire comme si vous étiez chez vous ! Ça peut être pratique pour une foule de choses, comme surveiller l'état d'un téléchargement un peu long, lancer l'exécution d'un programme pour qu'il soit prêt lorsque vous serez rentré chez vous... mais surtout, c'est comme ça que l'on administre un serveur sous Linux.



Un serveur est, pour faire simple, un ordinateur tout le temps connecté à internet. Il permet d'offrir des services divers et variés aux internautes. Par exemple, il y a des serveurs web dont le rôle est de... distribuer des pages web. Si vous voyez cette page, c'est que le serveur du Site du Zéro vous l'a envoyée.

La grande majorité des serveurs tournent sous Linux. Lorsque vous allez sur un site, il y a de très fortes chances pour que ce soit un serveur Linux qui vous ait répondu. Les serveurs Windows existent aussi, mais ils sont plus rares et on apprécie en général la stabilité de Linux ainsi que la possibilité de l'administrer à distance en ligne de commande.

On aura l'occasion de reparler de serveurs Linux plus tard (c'est un vaste sujet), mais je souhaite déjà vous montrer rapidement comment on fait pour se connecter à distance.

Telnet et SSH

Pour communiquer entre votre ordinateur et le serveur, il faut un **protocole**. C'est un ensemble de règles pour que 2 ordinateurs puissent discuter entre eux... un peu comme si 2 personnes devaient parler la même langue pour avoir une conversation 😊

Il existe des tonnes et des tonnes de protocoles pour communiquer par internet, mais pour ce qui est d'accéder à la ligne de commande à distance, c'est-à-dire à la console, il y a 2 principaux protocoles :

- **Telnet** : le protocole le plus basique, qui présente le gros défaut de ne pas crypter les données échangées entre vous et le serveur. Si un pirate "écoute" vos échanges par un moyen ou un autre, il pourrait récupérer des informations sensibles, en particulier votre mot de passe lorsque vous l'envoyez à la connexion. Ce moyen de connexion reste utilisé mais peu par rapport à SSH.
- **SSH** : c'est le protocole le plus utilisé car il permet de crypter les données et de sécuriser ainsi la connexion avec le serveur. C'est de très loin celui que l'on préfère utiliser en général.

Vous l'aurez compris, vous entendrez donc plus parler de SSH que d'autre chose 😊.

PuTTY

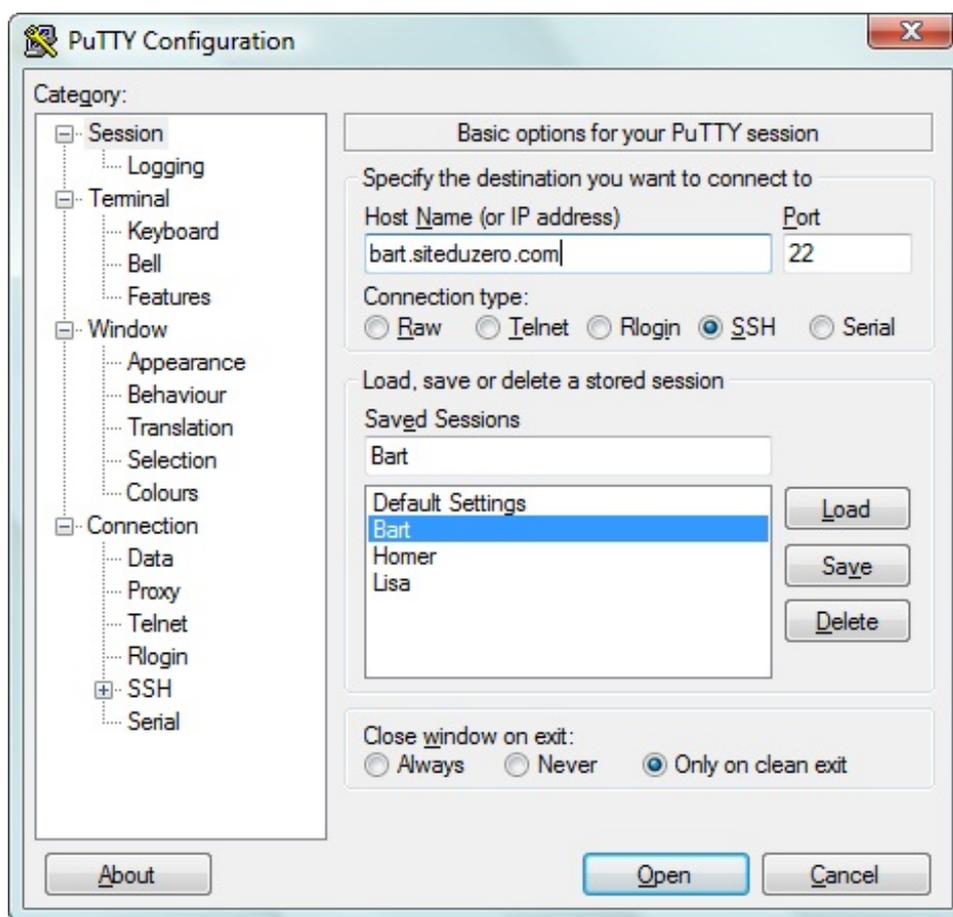
Pour accéder à distance à un ordinateur sous Linux connecté à internet, vous avez besoin d'un programme spécial capable de restituer la ligne de commande à distance. Ce qui est bien, c'est que vous n'avez pas forcément besoin d'être sous Linux pour vous connecter à un autre ordinateur sous Linux. On peut très bien le faire depuis Windows, c'est d'ailleurs la procédure que je vais vous montrer ici.

Il existe plusieurs programmes capables de se connecter en SSH à un serveur Linux. Le plus célèbre sous Windows est sûrement PuTTY : il est gratuit, léger, et ne nécessite même pas d'installation (juste un exécutable à lancer).

Vous pouvez aller sur le site web du logiciel à l'adresse suivante : <http://www.chiark.greenend.org.uk/~sgtatham/putty/>
Vous pouvez aussi télécharger directement le logiciel via ce lien :

[Télécharger PuTTY.exe \(450 Ko\)](#)

Lorsque vous le lancez, la fenêtre de configuration s'affiche :



Il y a beaucoup de pages d'options, comme le montre la section à gauche de la fenêtre. Dans la majeure partie des cas, vous n'aurez pas besoin d'y aller, heureusement 😊

Seule la première page est en fait vraiment importante : vous devez indiquer dans le champ *Host Name* en haut le nom d'hôte du serveur (dans mon cas `bart.siteduzero.com`) ou encore l'adresse IP de l'ordinateur, ça marche aussi bien (c'est juste plus difficile à retenir). Vérifiez que le type de connexion sélectionné en-dessous est bien SSH, puis cliquez sur le bouton *Open* tout en bas.

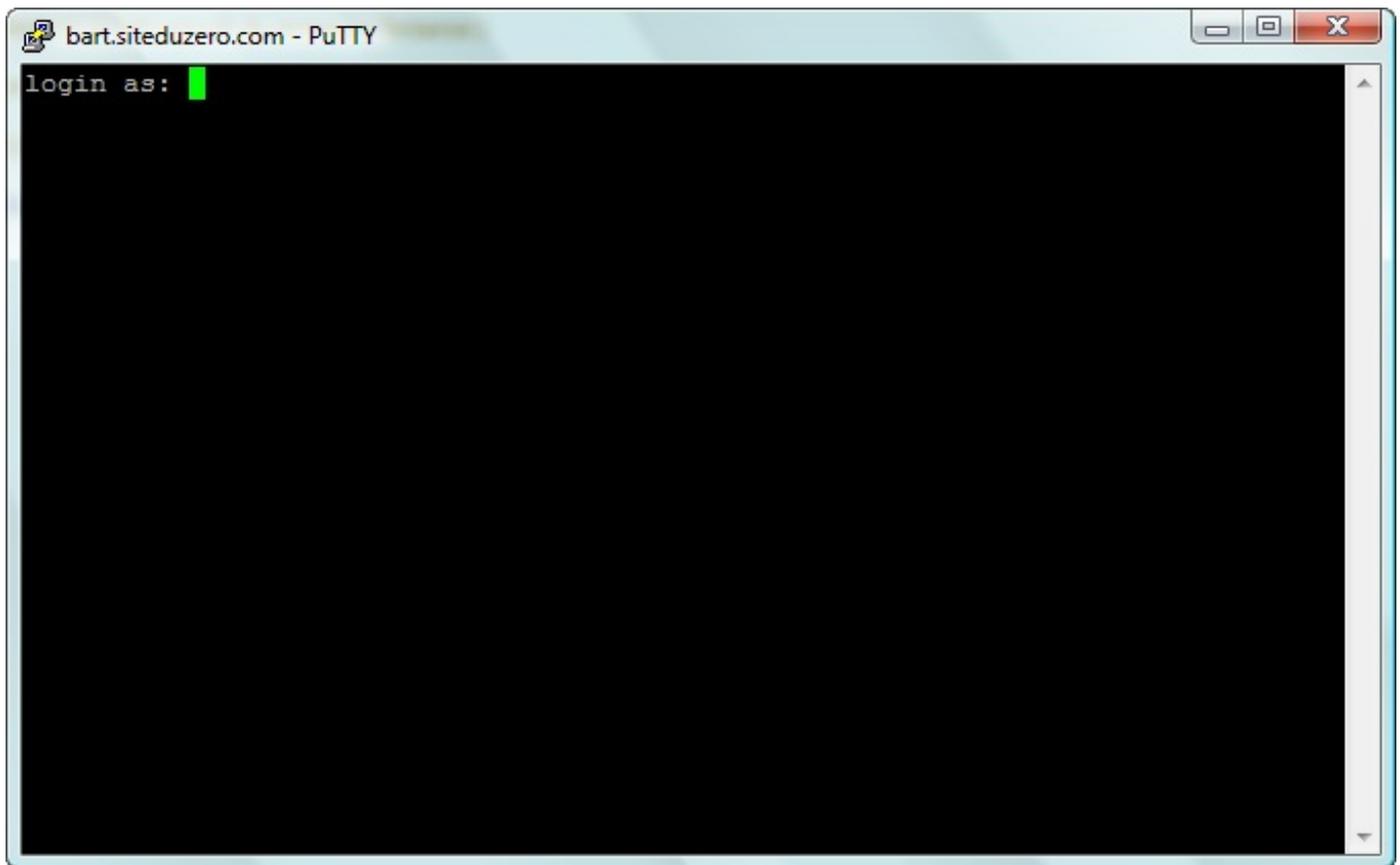


Si vous avez l'habitude de vous connecter à plusieurs serveurs différents, sachez qu'il est possible de sauvegarder les IP et configurations pour se connecter à chacun de ces serveurs. Utilisez la section "*Saved sessions*" au centre pour enregistrer ou ouvrir des connexions pré-enregistrées.

Après avoir cliqué sur *Open*, une fenêtre vous demandera lors de la première connexion au serveur si vous voulez stocker l'empreinte du serveur. C'est une sécurité pour vérifier que le serveur n'a pas changé depuis la dernière connexion et donc pour éviter que quelqu'un se fasse passer pour le serveur auquel vous avez l'habitude de vous connecter (le monde des pirates est sans pitié ! 🏴‍☠️).

On ne vous embêtera normalement plus jamais avec cette fenêtre ensuite.

La fenêtre principale de PuTTY s'affiche alors :



Voilà, vous n'avez plus qu'à vous connecter 😊

Indiquez votre login (par exemple mateo21), puis tapez Entrée.

On vous demande ensuite votre mot de passe, tapez-le puis faites Entrée.



Encore une fois, il est normal de ne pas voir d'étoiles lorsque vous tapez votre mot de passe. C'est une sécurité supplémentaire pour éviter que quelqu'un derrière vous ne compte le nombre de caractères. Soyez donc rassuré, le serveur reçoit bien votre mot de passe. Tapez-le comme si de rien n'était.

Si le login et le mot de passe sont bons, vous avez accès à la console du serveur sous Linux comme si vous étiez devant !

```

sdz@bart: /home/sdz
.....I~I=+=+~====+=+=+~+=~7+++...:
.....7=====+~+~====+I:
.....:I=====+==+?:
.....=?=====+==+I:
.....+?=====+==+?+.
.....?+=====I,
.....I+=~==~====+==+====~...
.....I+=+=+=+~==+=====.
.....,I=====????+?~==~=====.
.....,I=====+:.....~==II+??==.
.....:I==~==:.....~I:.....,~~...
.....~I==~=.
.....:?~====...Z.....=,.....:
.....,?7+~+=:.....~I~,ZZ.....:
.....~$?~==+=:.....,=?+=====.
.....:?=====++++~====I$?.
.....?+~==~==~==~==+?I~7~...
.....+~==~==~==~==~==~==I~...
.....=+~==~78$~+=~==~==~==~==I,
.....$$ZZ+~+=+~77I=====+?....
...IIIII7$Z7~==~+=?7I+=+====+++,
,~$77777IIII$7?=?=?,.....,.....
+$7I777I77I77I7Z:... ..
sdz@bart:~$ █

```

Bienvenue sur Bart !



Ne soyez pas surpris par tous les caractères que vous voyez sur ma capture d'écran. On a juste personnalisé le message de bienvenue du serveur "Bart" du Site du Zéro pour qu'il affiche la tête de Bart en lettres à la connexion 😊

Changer le message de bienvenue se fait facilement mais n'est en général pas très utile. Je vous dirai un peu plus tard comment faire.

Dans l'immédiat, vous ne devriez pas avoir à vous connecter en SSH à votre ordinateur à distance. Tout ce qu'on va faire, on va plutôt le faire en local, c'est-à-dire directement sur votre machine. On utilisera la méthode décrite plus haut, à savoir la console en mode graphique.

Au moins, vous savez désormais qu'il est aussi possible de communiquer à distance. Nous reparlerons plus en détail du protocole SSH plus tard dans ce cours, lorsque nous nous intéresserons à l'administration de serveurs.

Vous avez réussi à approcher la console ? Bravo ! C'est un premier pas 😊

Le plus intéressant reste à venir. Nous allons dès le prochain chapitre taper nos premières lignes de commande dans la console de Linux.

Je suis prêt à parier que dans quelques heures vous aurez l'impression d'avoir appris une nouvelle langue 😊

Au fait, je vous conseille de prendre une feuille de papier et un crayon pour prendre des notes. Ça ne mange pas de pain, et ça risque de vous être sacrément utile par la suite. Vous allez en effet devoir retenir pas mal de commandes qui seront pour vous totalement nouvelles !

📄 Entrer une commande

Chers amis, le grand jour est enfin arrivé ! Vous allez rentrer votre première commande de console ! 😊

Ca va, pas trop stressés ? 😊

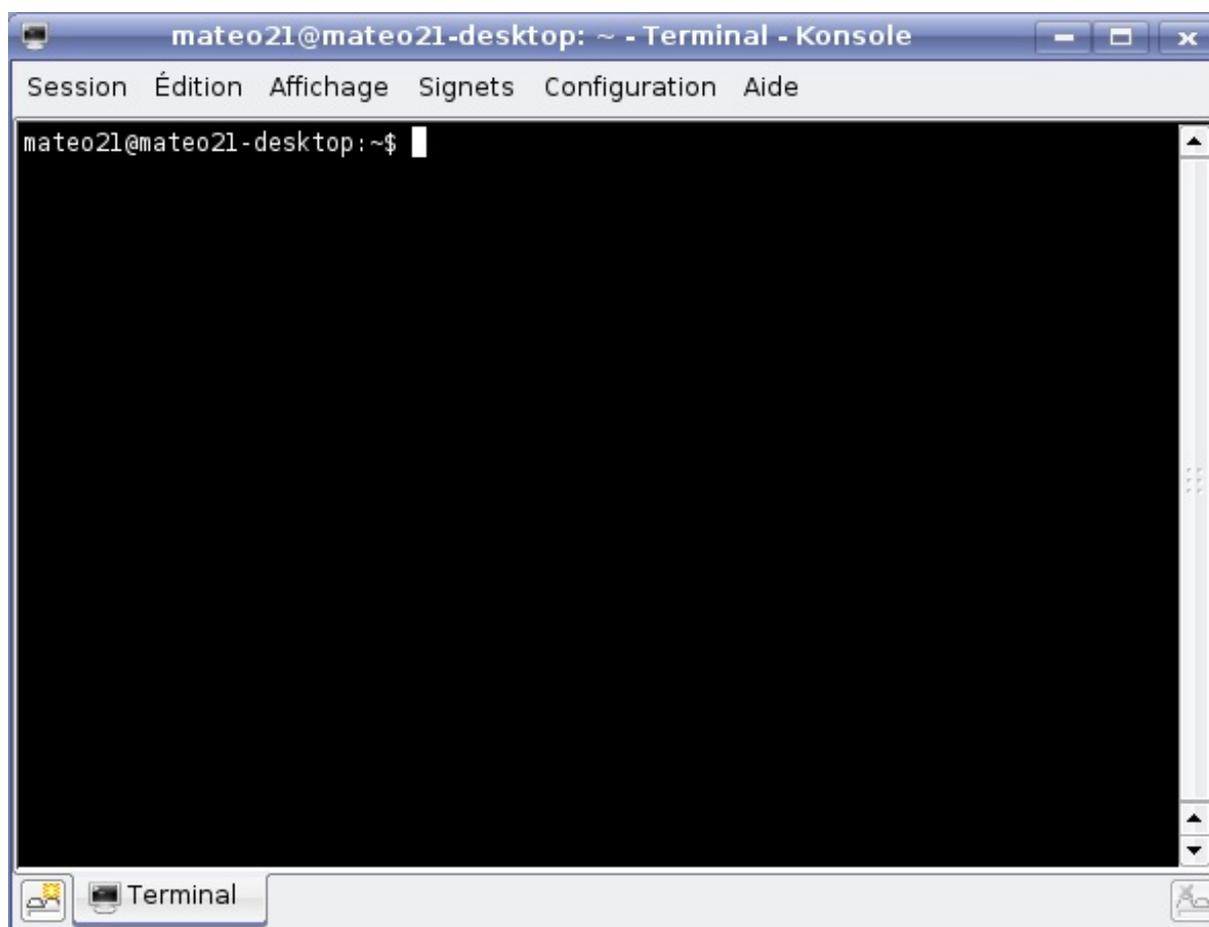
Je vous rassure, nous allons commencer par des choses très simples pour nous familiariser avec la console. Nous allons vraiment voir le B.A.-BA, le guide de survie élémentaire en quelque sorte.

Il va falloir retenir beaucoup de nouvelles choses. Je ne saurais trop vous conseiller de vous munir d'un crayon et d'une feuille de papier pour noter les commandes et raccourcis que je vais vous montrer. Ca sera plus simple que de rechercher 10 fois la commande dans le tuto par la suite 😊

L'invite de commandes

Je suppose à partir de maintenant que vous avez ouvert une console. Si vous ne savez pas faire, c'est que vous n'avez pas lu le chapitre précédent 😊

Le mieux, comme je vous l'ai dit, est d'ouvrir une console dans le mode graphique. Le programme Konsole sous KDE ou Terminal sous Gnome fera donc très bien l'affaire :



Pour simplifier la vie à votre modem, je ne vais pas faire une capture d'écran à chaque fois que je veux vous montrer le résultat d'une commande, ce serait un peu long à télécharger. Je vais utiliser des encadrés "Console" comme celui-ci :

Code : Console

```
mateo21@mateo21-desktop:~$
```

Ca c'est ce que vous voyez à l'écran. Vous n'avez encore rien écrit, mais l'ordinateur vous dit bonjour à sa manière (bon ok c'est une façon très spéciale de dire bonjour j'avoue 😊).

Ce que vous voyez là est ce qu'on appelle l'**invite de commandes**. C'est un message qui vous *invite* à rentrer une commande en vous donnant par la même occasion une foule d'informations. Cette invite s'affiche avant chaque commande que vous tapez.

Bien, décortiquons cette invite de commandes parce qu'elle est très intéressante :

- **mateo21** : le premier élément est votre pseudonyme. C'est le pseudo sous lequel vous vous êtes loggé. En effet, rappelez-vous : on peut créer plusieurs comptes utilisateur sous Linux. Il est en général conseillé d'en faire un par personne susceptible d'utiliser l'ordinateur (un pour chaque membre de la famille par exemple). Nous verrons plus tard comment rajouter des comptes utilisateurs.
- **@** : ce symbole n'indique rien de particulier. C'est le symbole "at" qui signifie "chez". Si on lit l'invite de gauche à droite, on doit donc comprendre "*mateo21 chez*".
- **mateo21-desktop** : ça c'est le nom de l'ordinateur sur lequel vous êtes en train de travailler. Dans mon cas il s'appelle mateo21-desktop, mais j'aurais pu l'appeler du nom que je voulais lors de l'installation. Par exemple, on a l'habitude de donner le nom d'un membre des Simpson à chacun des serveurs du Site du Zéro : Lisa, Bart, Itchy, Scratchy... Cela permet de savoir de quelle machine on parle quand on dit "Oulah Bart est surchargé, il faudrait voir quel est le programme qui ralentit tout". Si vous suivez toujours, la ligne d'invite de commandes se lit donc "*mateo21 chez mateo21-desktop*". En d'autres termes, je suis identifié en tant que mateo21 sur la machine mateo21-desktop.
- **:** : ce symbole à nouveau ne veut rien dire de spécial, c'est un séparateur.
- **~** : ça, c'est le dossier dans lequel vous vous trouvez actuellement. Vous pouvez naviguer de dossier en dossier dans la console et il est très utile qu'on vous rappelle systématiquement avant chaque commande où vous êtes. Pour information, le symbole ~ signifie que vous êtes dans votre dossier personnel, ce qu'on appelle le "Home" sous Linux. C'est l'équivalent du dossier "Mes documents" de Windows. Nous étudierons plus en détail le fonctionnement des dossiers sous Linux dans le prochain chapitre.
- **\$** : ce dernier symbole est très important, il indique votre niveau d'autorisation sur la machine. Il peut prendre 2 formes différentes :
 - **\$** : signifie que vous êtes en train d'utiliser un compte utilisateur "normal", avec des droits limités (il ne peut pas modifier les fichiers système les plus importants). Mon compte mateo21 est donc un compte normal avec des droits limités.
 - **#** : signifie que vous êtes en mode super-utilisateur, c'est-à-dire que vous êtes connecté sous le pseudonyme "root". Le root est l'utilisateur maître qui a le droit de tout faire sur sa machine (même de la détruire 🤖). Nous verrons le mode root plus en détails plus tard, pour l'instant nous restons dans un compte utilisateur limité car comme ça nous ne risquons pas de faire de bêtise 😊

Comme vous le voyez, une fois qu'on parle la même langue que l'invite de commandes on comprend ce qu'elle veut dire ! "*Bonjour et bienvenue, vous êtes mateo21 sur la machine mateo21-desktop. Vous vous trouvez actuellement dans votre dossier Home et possédez des droits utilisateur limités. La température extérieure est de*"



Comme un peu tout sous Linux, l'invite de commandes est totalement paramétrable. Vous pouvez la raccourcir si vous trouvez qu'elle est trop longue ou la rallonger si vous trouvez qu'elle ne donne pas assez d'informations. Vous pouvez en théorie mettre vraiment tout ce que vous voulez dans l'invite, comme par exemple l'heure actuelle (par contre pour la température extérieure il faudra repasser 🤖).

Nous verrons comment changer cela lorsque vous aurez appris à vous servir d'un éditeur de texte !

Commandes et paramètres

On travaille dans la console en tapant ce qu'on appelle des **commandes**. Les commandes sont nombreuses, vous ne pourrez jamais toutes les connaître dans votre vie. Et ce n'est pas le but : le but c'est que vous sachiez vous servir par coeur de la plupart des commandes "courantes" et, pour les moins courantes, que vous soyez capable d'apprendre à vous en servir en lisant leur **manuel d'utilisation**.

Le manuel d'utilisation est la véritable bible de tous les linuxiens. Vous verrez rapidement qu'ils ne jurent que par ça. Pourquoi ? Parce que c'est tout simplement un outil de référence, là où l'on peut trouver la réponse à TOUTES ses questions pour peu qu'on prenne la peine de le lire et qu'on sache le lire. Un chapitre entier du cours vous apprendra à lire le manuel, c'est très très important.



Pour information, il est courant de voir un linuxien dire "*RTFM*" aux débutants qui posent des questions qu'ils estiment "simples". RTFM est l'abréviation de "*Read the fucking manual !*", ce qui veut grosso modo dire "*Lis le putain de manuel !*". Ce n'est pas vraiment une insulte en fait, mais plutôt une sorte de... soupir d'exaspération.

Pour éviter qu'on vous dise ça un jour, je vous conseille donc de bien ouvrir vos oreilles quand je vous expliquerai comment utiliser le manuel 😊

Une commande simple

Bon trève de bavardages, on va rentrer une commande ! Par exemple, tapez `date` puis appuyez sur la touche "Entrée" du clavier. Le résultat devrait ressembler à cela :

Code : Console

```
mateo21@mateo21-desktop:~$ date
vendredi 14 septembre 2007, 18:59:54 (UTC+0200)
```

La première ligne contient l'invite de commandes suivie de la commande que j'ai tapée. La seconde ligne est la réponse de l'ordinateur à cette commande.

Je suppose que vous avez deviné comme des grands ce qu'on vient de faire : on a demandé quelles étaient la date et l'heure !

Vous en voulez encore ? Ok alors essayons une toute autre commande : tapez `ls`. C'est l'abréviation de "list", qui signifie "*Lister les fichiers et dossiers du répertoire actuel*".

Code : Console

```
mateo21@mateo21-desktop:~$ ls
Desktop Examples Images
```

Cela signifie que le répertoire actuel est constitué de 3 dossiers : Desktop Examples et Images. En général, le système colore les éléments pour que l'on puisse distinguer facilement les dossiers des fichiers.

Si vous n'avez aucune réponse, c'est que vous êtes dans un dossier qui ne contient aucun fichier ou dossier.

Voilà, c'est aussi simple que cela. Une commande est constituée d'un mot et ne contient pas d'espace. Dans un cas très simple comme ceux qu'on vient de voir, il suffit de taper juste la commande pour avoir une réponse. Mais dans la quasi-totalité des cas on peut (et parfois on DOIT) rentrer des options, qu'on appelle paramètres.

Les paramètres

Les paramètres sont des options que l'on écrit à la suite de la commande. La commande et les paramètres sont séparés par un espace, comme ceci :

Code : Console

```
mateo21@mateo21-desktop:~$ commande parametres
```

Les paramètres peuvent eux-mêmes contenir des espaces, des lettres, des chiffres, en fait un peu de tout. Il n'y a pas de règle véritable sur la forme des paramètres, mais heureusement les programmeurs ont adopté une sorte de "convention" pour que l'on puisse reconnaître les différents types de paramètres.

Les paramètres courts (une lettre)

Les paramètres les plus courants sont constitués d'une seule lettre, précédée d'un tiret. Par exemple :

Code : Console

```
commande -d
```

Si on doit donner plusieurs paramètres, on peut faire comme ceci :

Code : Console

```
commande -d -a -U -h
```

Ou, plus court :

Code : Console

```
commande -daUh
```



Attention à la casse des paramètres (majuscules / minuscules) ! Si vous écrivez -u, cela n'a en général pas du tout le même sens que -U !

Faisons un essai avec la commande `ls` et rajoutons-lui le paramètre "a" (en minuscule) :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -a
.                .gconfd          .mozilla-thunderbird
..               .gimp-2.2        .nautilus
.bash_history    .gksu.lock       .profile
.bash_logout     .gnome           .recently-used
.bashrc          .gnome2          .recently-used.xbel
.config          .gnome2_private  .ssh
Desktop          .gstreamer-0.10 .sudo_as_admin_successful
.dmrc            .gtkrc-1.2-gnome2 .themes
.esd_auth        .ICEauthority    .thumbnails
.evolution       .icons           .Trash
Examples         .lessshst        tutos
.face            .local           .update-manager-core
.fontconfig      .macromedia      .update-notifier
.gaim            .metacity        .Xauthority
.gconf           .mozilla         .xsession-errors
```

Cela affiche tout le contenu du dossier, même les fichiers cachés.

Un "fichier caché" sous Linux est un fichier qui commence par un point. Normalement, si vous vous trouvez dans votre répertoire Home, vous devriez avoir une bonne floppée de fichiers cachés 😊 Ce sont en général des fichiers de configuration de programmes.

Les paramètres longs (plusieurs lettres)

Les paramètres constitués de plusieurs lettres sont précédés de 2 tirets, comme ceci :

Code : Console

```
commande --parametre
```

Cette fois pas le choix, si vous voulez mettre plusieurs paramètres longs il faudra mettre un espace entre chacun d'eux :

Code : Console

```
commande --parametre1 --parametre2
```

On peut aussi combiner les paramètres longs et les paramètres courts dans une commande :

Code : Console

```
commande -daUh --autreparametre
```



Il y a parfois 2 écritures possibles pour un paramètre de commande : une version courte et une version longue. Cela permet de vous laisser le choix selon si vous préférez la version courte ou la version longue. Notez que c'est la commande qui décide des paramètres qu'elle accepte, et parfois certaines ne proposent pas le choix entre une version courte et une version longue.

Testons sur la commande `ls` avec le paramètre `--all`, qui signifie "tout" en anglais :

Code : Console

```
mateo21@mateo21-desktop:~$ ls --all
.          .gconfd          .mozilla-thunderbird
..         .gimp-2.2        .nautilus
.bash_history .gksu.lock       .profile
.bash_logout .gnome           .recently-used
.bashrc     .gnome2         .recently-used.xbel
.config     .gnome2_private .ssh
Desktop    .gstreamer-0.10 .sudo_as_admin_successful
.dmrc      .gtkrc-1.2-gnome2 .themes
.esd_auth  .ICEauthority    .thumbnails
.evolution .icons           .Trash
Examples   .lessht         tutos
.face      .local          .update-manager-core
.fontconfig .macromedia     .update-notifier
.gaim      .metacity       .Xauthority
.gconf     .mozilla        .xsession-errors
```

Comme vous le voyez, `--all` est un synonyme de `-a`. Cela illustre ce que je vous disais à l'instant, comme quoi parfois une commande propose 2 façons d'utiliser un paramètre : une courte et une longue.

Les valeurs des paramètres

Certains paramètres nécessitent que vous les complétiez avec une valeur. Cela fonctionne différemment selon si vous travaillez avec un paramètre long ou avec un paramètre court.

Avec un paramètre court :

Code : Console

```
commande -p 14
```

Cela indique que l'on associe la valeur 14 au paramètre p. Avec ce genre de technique on peut par exemple faire comprendre à l'ordinateur "Je veux voir la liste de tous les fichiers de plus de 14 Mo".

Si c'est un paramètre long, on fait en général comme ceci :

Code : Console

```
commande --parametre=14
```

Le résultat sera le même, il est juste plus lisible mais aussi plus long à écrire.

Les autres paramètres

Je vous l'ai dit : il n'y a pas de règle absolue au niveau des paramètres et vous rencontrerez sûrement des paramètres qui fonctionnent différemment. Heureusement les "conventions" que je viens de vous donner sont valables dans la grande majorité des cas, ce qui devrait vous permettre de vous repérer.

Certains paramètres sont donc un peu différents et dépendent vraiment des commandes. Par exemple avec ls, si on ajoute le nom d'un dossier (ou sous-dossier) cela affichera le contenu de ce dossier au lieu du contenu du dossier courant :

Code : Console

```
mateo21@mateo21-desktop:~$ ls Examples
Experience ubuntu.ogg      logo-Ubuntu.png          oo-payment-schedule.ods
fables_01_01_aesop.spx    oo-about-these-files.odt oo-presenting-
kubuntu.odp
gimp-ubuntu-splash.xcf    oo-about-ubuntu-ru.rtf   oo-presenting-
ubuntu.odp
kubuntu-leaflet.png      oo-cd-cover.odg          oo-trig.xls
logo-Edubuntu.png        oo-derivatives.doc       oo-welcome.odt
logo-Kubuntu.png         oo-maxwell.odt           ubuntu Sax.ogg
```

Retrouver une commande

Linux propose tellement de commandes différentes qu'il est facile de s'y perdre et d'en oublier une. Ça m'arrive très régulièrement personnellement, et heureusement ce n'est pas un drame. En effet, Linux vous propose toute une série de façons de retrouver une commande que vous avez oubliée 😊

Autocomplétion de commande

Le premier "truc" à connaître, c'est l'autocomplétion de commande. Prenons la commande "date" par exemple : vous êtes un peu tête en l'air et vous ne savez plus comment elle s'écrit. Par contre, vous êtes sûr des premières lettres de la commande.

Lister les commandes correspondantes

Tapez juste "da" dans la console, puis **tapez 2 fois sur la touche "Tabulation"** située à gauche de votre clavier. Le résultat sera le suivant :

Code : Console

```
mateo21@mateo21-desktop:~$ da
```

```
dash date
mateo21@mateo21-desktop:~$ da
```

En tapant 2 fois sur Tabulation, vous avez demandé à l'ordinateur la liste des commandes qui commencent par "da". On vous a répondu "dash" et "date". Il y a donc 2 commandes qui commencent par "da", et vous venez de retrouver celle que vous cherchiez, c'est-à-dire "date" 😊

Bien sympa, l'ordinateur a réécrit l'invite de commandes en-dessous ainsi que le début de la commande que vous aviez tapée. Vous n'avez plus qu'à compléter avec les lettres "te" qui manquent et à taper Entrée, et ce sera bon 😊

L'autocomplétion

Plus sympa encore, s'il n'y a qu'un seul résultat correspondant à votre recherche, l'ordinateur complètera avec les lettres qui manquent et vous n'aurez plus qu'à taper sur Entrée !

Par exemple, il n'y a qu'une commande qui commence par "dat". Tapez donc `dat` dans la console, puis **tapez 1 seule fois sur Tabulation**. La commande se complète comme par magie 😊

Trop de commaaaaandes !

Parfois, il y a trop de commandes correspondant à votre recherche. Faites un essai un peu brutal : ne rentrez aucun début de commande et faites 2 fois Tab (Tabulation). Cela demande de faire la liste de toutes les commandes disponibles sur votre ordinateur.

Code : Console

```
mateo21@mateo21-desktop:~$
Display all 2173 possibilities? (y or n)
```

Brutal hein ? 😊

Il y a 2173 commandes disponibles sur mon ordinateur. Plus j'installerai de programmes, plus j'aurai de commandes utilisables. N'espérez donc pas toutes les connaître, de nouveaux programmes sortent tous les jours 😊

A cette question vous pouvez répondre "y" (yes), et la liste s'affichera page par page. Quelques raccourcis à connaître quand une liste s'affiche page par page :

- Tapez Espace pour passer à la page suivante
- Tapez Entrée pour aller à la ligne suivante
- Tapez q pour arrêter la liste

Si vous répondez "n" (no), rien ne se passera. C'est dans le cas où vous vous diriez "Oulah, 2173 possibilités autant chercher une aiguille dans une botte de foin, je vais peut-être affiner ma recherche" 😊

L'historique des commandes

On a très souvent besoin de retrouver une commande qu'on vient de taper il y a 5 minutes (ou même 5 secondes). Parfois c'est parce qu'on a oublié la commande, mais souvent c'est aussi parce qu'on a un énoorme poil dans la main comme moi et qu'on a vraiment la flemme de réécrire la commande en entier nous-même 😊

Ce raccourci vaut de l'or : appuyez sur la flèche directionnelle vers le haut (). Vous verrez apparaître la dernière commande que vous avez tapée.

Si vous réappuyez sur la flèche vers le haut, vous verrez l'avant-dernière commande, puis l'avant-avant-dernière etc.

Si vous appuyez sur la flèche vers le bas () , vous reviendrez vers les commandes les plus récentes.

C'est ainsi que je peux successivement retrouver les commandes que je viens de taper, dans l'ordre inverse :

- ls --all
- ls -a
- ls
- date
- etc.

Si vous voulez "remonter" très longtemps en arrière dans l'historique de vos commandes, pas la peine de taper 100 fois sur la flèche vers le haut comme un forcené 🤪

Il existe la commande "history" qui vous rappelle l'historique des commandes :

Code : Console

```
152 date
153 ls
154 ls -a
155 ls --all
156 history
```

La dernière commande tapée sera toujours history, forcément 🤪

Vous remarquerez que les commandes sont numérotées : ainsi, on peut savoir que date est la 152^{ème} commande que j'ai tapée dans le terminal.

Ctrl + R : rechercher une commande tapée avec quelques lettres

Dans le cas où la flèche vers le haut et la commande history ne suffiraient pas à retrouver une vieille commande que vous avez tapée, il y a un raccourci super utile : Ctrl + R. Appuyez donc sur les touches Ctrl et R en même temps, et l'ordinateur se mettra en mode "recherche d'une commande tapée" (R comme Recherche).

Là, vous pouvez taper n'importe quelle suite de lettres correspondant à une vieille commande. Par exemple, faites Ctrl + R puis tapez "all". Linux retrouve la commande "ls --all" qui contenait justement le mot "all". Vous n'avez plus qu'à taper Entrée pour relancer la commande ! 😊

Code : Console

```
(reverse-i-search) `all': ls --all
```

Si ce n'est pas la commande que vous cherchiez, appuyez à nouveau sur Ctrl + R pour remonter dans la liste des commandes qui contiennent "all".

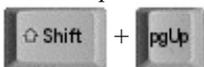
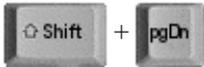
Ça a peut-être l'air bête sur une commande comme ça, mais certaines sont vraiment très longues et c'est un vrai bonheur de ne pas avoir à les réécrire en entier !

Quelques raccourcis claviers pratiques

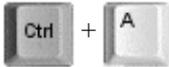
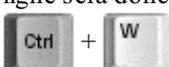
On ne dirait pas comme ça, mais la console de Linux propose une quantité incroyable de raccourcis clavier. Ce sont des raccourcis qu'on ne peut pas deviner, qu'on a un peu de mal à retenir au début, mais quand on les connaît... Waouh ! Ben on devient un peu comme Neo dans Matrix en fait, on va très vite 😊

Ces raccourcis ne sont pas intuitifs, mais ça vaut vraiment le coup de les retenir. Les premiers temps vous reviendrez sûrement souvent ici pour les consulter (et encore, si vous vous faites un petit pense-bête comme je vous ai dit, ça ne devrait pas arriver), mais petit à petit vous les connaîtrez par coeur !

Commençons par quelques raccourcis généraux à connaître :

-  : efface le contenu de la console. Utile pour faire un peu de ménage quand votre console est encombrée, ou quand votre boss passe derrière et que vous n'aimeriez pas qu'il voie ce que vous étiez en train de faire 🤪
A noter qu'il existe aussi une commande, `clear`, qui fait exactement la même chose.
-  : envoie le message EOF (fin de fichier) à la console. Si vous tapez ce raccourci dans une ligne de commande vide (c'est-à-dire sans avoir écrit un début de commande avant), cela fermera la console en cours. A noter qu'il existe aussi la commande `exit` qui a le même effet.
-  : vous permet de "remonter" dans les messages envoyés par la console. En mode graphique, la molette de la souris fait aussi très bien ça. La touche "Page Up" est généralement représentée sur votre clavier par une flèche vers le haut barrée par plusieurs petites barres horizontales.
-  : pareil, mais pour redescendre.

Les raccourcis suivants sont utiles lorsque vous êtes en train d'écrire une longue commande :

-  : ramène le curseur au début de la commande. La touche "Origine" a le même effet (elle est située à côté de la touche fin et représentée par une flèche pointant en haut à gauche).
-  : ramène le curseur à la fin de la ligne de commande. La touche "Fin" a le même effet.
-  : supprime tout ce qui se trouve à gauche du curseur. Si le curseur est situé à la fin de la ligne, toute la ligne sera donc supprimée.
-  : supprime tout ce qui se trouve à droite du curseur. Si le curseur est situé au début de la ligne, toute la ligne sera donc supprimée.
-  : supprime le premier mot situé à gauche du curseur. Un "mot" est séparé par des espaces. On s'en sert en général pour supprimer le paramètre situé à gauche du curseur.
-  : si vous avez supprimé du texte avec une des commandes Ctrl + U, Ctrl + K ou Ctrl + W qu'on vient de voir, alors le raccourci Ctrl + Y "collera" le texte que vous venez de supprimer. C'est un peu comme un couper-coller donc 😊

On va arrêter là la liste pour le moment. Ca vous fait déjà pas mal de choses à retenir 🤪

Il existe en fait beaucoup d'autres raccourcis clavier, mais je vous les montrerai au fur et à mesure du cours, quand on en aura besoin.

Même si c'est un peu barbant, je vous conseille de vous entraîner à les retenir par coeur ! Vous serez vraiment beaucoup plus efficaces lorsque vous les connaîtrez !

Eh ben dites donc ! Je crois qu'on vient de mettre un pied dans la console, et qu'on commence même à s'embourber jusqu'au genou 😊

Beaucoup, beaucoup, beaucoup de nouveautés pour vous dans ce chapitre. Faites-vous un pense-bête, ça va vraiment vous resservir. Prenez le temps de bien expérimenter tous les raccourcis pour que vous sachiez lequel utiliser lorsque vous en avez besoin.

Et dès que vous vous sentez au point, on continue 😊

Dans le prochain chapitre nous commencerons à travailler en détail sur des commandes concrètes un peu plus complexes, qui nécessitent en particulier l'utilisation de plusieurs paramètres. Nous découvrirons comment sont organisés les fichiers et dossiers sous Linux et comment on se déplace dans tous ces dossiers.

La structure des dossiers et fichiers

Ahhh, les fichiers sous Linux, tout un programme 😊

Vous croyez savoir ce que sont les fichiers et dossiers ? Vous croyez que votre disque dur s'appelle C: ? Que le lecteur CD c'est D:, ou peut-être E: ? 😊

Les choses ne fonctionnent pas du tout de la même manière sous Linux et sous Windows. Or, savoir comment se déplacer de dossier en dossier, et savoir faire la liste des fichiers, c'est quand même sacrément important ! C'est pour cela que nous allons voir ensemble le fonctionnement des fichiers sous Linux dès maintenant.

Organisation des dossiers

Le système qui gère les fichiers sous Linux est un peu déroutant au début, surtout quand on est habitué à celui de Windows. En effet, ici vous ne trouverez pas de "C:\", "D:\" ou que sais-je encore. Les fichiers sont organisés d'une manière complètement différente.

Au lieu de séparer chaque disque dur, lecteur cd, lecteur de disquettes, lecteur de carte mémoire... Linux place en gros tout au même endroit.



Mais comment on fait pour savoir si le dossier dans lequel on est appartient au premier disque dur, au second disque dur, au lecteur CD... ? C'est le bordel non ?

C'est ce qu'on pourrait croire au premier abord, mais en fait c'est juste une autre façon de penser la chose 😊

2 types de fichiers

Pour faire simple, il existe 2 grands types de fichiers sous Linux :

- **Les fichiers classiques** : ce sont les fichiers que vous connaissez, ça comprend les fichiers texte (.txt, .doc, .odt...), les sons (.wav, .mp3, .ogg), mais aussi les programmes. Bref, tout ça ce sont des fichiers que vous connaissez et que vous retrouvez dans Windows.
- **Les fichiers spéciaux** : certains autres fichiers sont spéciaux car ils *représentent* quelque chose. Par exemple, votre lecteur CD est un fichier pour Linux. Là où Windows fait la distinction entre ce qui est un fichier et ce qui ne l'est pas, Linux lui dit que tout est un fichier. C'est une conception très différente, un peu déroutante comme je vous l'ai dit, mais pas de panique vous allez vous y faire.

La racine

Dans un système de fichiers, il y a toujours ce qu'on appelle une racine, c'est-à-dire un "gros dossier de base qui contient tous les autres dossiers et fichiers".

Sous Windows, il y a en fait plusieurs racines. "C:\" est la racine de votre disque dur, "D:\" est la racine de votre lecteur CD (par exemple).

Sous Linux, il n'y a qu'une et une seule racine : "/". Comme vous le voyez, il n'y a pas de lettre de lecteur car justement Linux ne donne pas des noms aux lecteurs comme le fait Windows. Il dit juste "La base, c'est /".



Il n'y a pas de dossier de plus haut niveau que /, c'est-à-dire qu'il n'existe pas de dossier qui contienne le dossier /. Quand on est à la racine, on ne peut pas remonter en arrière car... on est déjà tout au début 😊

Architecture des dossiers

Sous Windows, un dossier peut être représenté comme ceci : "C:\Program Files\Winzip". On dit que "Winzip" est un sous-dossier du dossier "Program Files", lui-même situé à la racine.

Vous noterez que c'est l'antislash "\" (aussi appelé backslash) qui sert de séparateur aux noms de dossiers.

Sous Linux, c'est au contraire le "/" qui sert de séparateur.

Comme je vous l'ai dit, il n'y a pas de "C:" sous Linux, la racine (le début) s'appelant juste "/".

Le dossier de notre super programme ressemblerait plutôt à quelque chose comme cela : "/usr/bin/". On dit que "bin" est un sous-dossier du dossier "usr", lui-même situé à la racine.



Linux gère sans problème les noms de fichiers et dossiers contenant des espaces, des accents et des majuscules. Toutefois, vous remarquerez que la plupart du temps on préfère les éviter. On trouve ainsi plutôt des noms tout en minuscules sans accents ni espaces, comme "usr", "bin", "apache", etc. Souvenez-vous qu'il n'est pas obligatoire de nommer vos fichiers en suivant la même règle, mais la plupart des programmes que vous installerez préfèrent utiliser des noms tout en minuscules sans espaces ni accents, donc ne soyez pas surpris 😊

Les dossiers de la racine

Sous Windows, on a l'habitude de trouver souvent les mêmes dossiers à la racine : "Documents and Settings", "Program Files", "Windows"...

Sous Linux, vous vous en doutez, les dossiers sont complètement différents (et on risque pas de trouver de dossier qui s'appelle Windows ! 🤖).

Je vais vous faire ici la liste des dossiers les plus courants que l'on retrouve à chaque fois à la racine de Linux. La description de chaque dossier sera rapide, mais c'est juste pour que vous puissiez vous repérer au début 😊



Il n'est PAS nécessaire de retenir cette liste par coeur. D'ailleurs je n'ai mis que les dossiers principaux, et elle est quand même longue.

Servez-vous en juste si vous avez besoin de savoir à quoi correspond grosso modo tel ou tel dossier, mais ne vous en faites pas si vous ne maîtrisez pas à fond le sens de chacun de ces dossiers (moi-même je ne connais pas vraiment le sens détaillé de chacun d'eux).

- **bin** : contient des programmes (exécutables) qui sont susceptibles d'être utilisés par tous les utilisateurs de la machine.
- **boot** : fichiers permettant le démarrage de Linux.
- **dev** : fichiers contenant les périphériques. En fait, on en reparlera plus tard, mais ce dossier contient des sous-dossiers qui "représentent" chacun un périphérique. On y retrouve ainsi par exemple le fichier qui représente le lecteur CD.
- **etc** : fichiers de configuration.
- **home** : répertoires personnels des utilisateurs. On en a déjà parlé un peu avant : c'est dans ce dossier que vous placerez vos fichiers personnels, à la manière du dossier "Mes documents" de Windows. Chaque utilisateur de l'ordinateur possède son dossier personnel. Par exemple, dans mon cas mon dossier personnel se trouve dans "/home/mateo21/". S'il y avait un autre utilisateur (appelons-le Patrick) sur mon ordinateur, il aurait eu droit lui aussi à son propre dossier : "/home/patrick/".
- **lib** : dossier contenant les bibliothèques partagées (généralement des fichiers .so) utilisées par les programmes. C'est en fait là qu'on trouve l'équivalent des .dll de Windows.
- **media** : lorsqu'un périphérique amovible (comme une carte mémoire SD ou une clé USB) est inséré dans votre ordinateur, Linux vous permet d'y accéder à partir d'un sous-dossier de "media". On parle de *montage*. C'est un peu compliqué, on en reparlera dans un chapitre plus tard.
- **mnt** : c'est un peu pareil que media, mais pour un usage plus temporaire.
- **opt** : répertoire utilisé pour les add-ons de programmes.
- **proc** : contient des informations système.
- **root** : c'est le dossier personnel de l'utilisateur "root". Normalement, les dossiers personnels sont placés dans "home", mais celui de root fait exception. En effet, comme je vous l'ai dit dans le chapitre précédent root est le super-utilisateur, le "chef" de la machine en quelque sorte. Il a droit à un espace spécial 😊
- **sbin** : contient des programmes système importants.
- **tmp** : dossier temporaire utilisé par les programmes pour stocker des fichiers.
- **usr** : c'est un des plus gros dossiers, dans lequel vont s'installer la plupart des programmes demandés par l'utilisateur.
- **var** : ce dossier contient des données "variables", souvent des fichiers, des logs (traces écrites de ce qui s'est passé récemment sur l'ordinateur), etc.

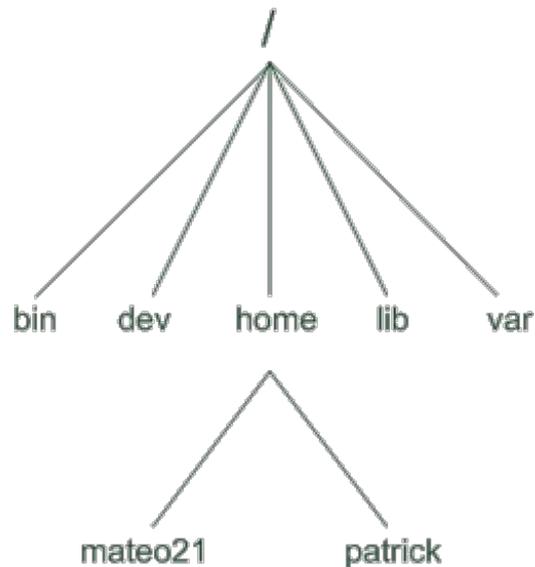
Cette liste de dossiers est en fait présente sur tous les OS de type Unix, et pas seulement sous Linux.

Encore une fois, ne retenez pas tout ça. C'est juste pour vous donner une idée de ce que contiennent les dossiers à la racine de

Linux, car je sais que c'est une question qu'on se pose souvent quand on débute.

Schéma résumé de l'architecture

Pour bien que vous vous repérez, sachez qu'on peut présenter l'organisation des dossiers de Linux de cette manière :



La racine tout en haut est /, elle contient plusieurs dossiers, qui contiennent chacun eux-mêmes plusieurs dossiers, qui contiennent des dossiers et fichiers, etc etc. 😊

pwd & which : où... où suis-je ?

Le nombre de dossiers et de fichiers présents après l'installation d'Ubuntu est tellement grand qu'il serait facile de s'y perdre. Un grand nombre de programmes sont en effet préinstallés pour que vous puissiez profiter rapidement des possibilités de Linux. Ne comptez donc pas sur moi pour vous faire la liste complète des dossiers et fichiers que vous possédez, ce n'est pas réaliste 😊

En revanche, je vais vous apprendre maintenant à vous repérer dans l'arborescence des dossiers. Vous saurez alors à tout moment où vous êtes sur votre disque. C'est un peu comme avoir une carte routière en quelque sorte !

pwd : afficher le dossier actuel

Lorsque vous ouvrez la console pour la première fois, Linux vous place dans votre dossier personnel, votre "home". En l'occurrence dans mon cas, le dossier dans lequel je serai placé sera /home/mateo21.

Normalement, l'invite de commandes vous indique le nom du dossier dans lequel vous vous trouvez :

Code : Console

```
mateo21@mateo21-desktop:~$
```

Si vous vous souvenez bien, le nom du dossier est situé entre le ":" et le "\$". Donc ici, on se trouve dans le dossier "~".



Rappel : je l'ai dit dans le chapitre précédent mais ça ne fait pas de mal de le répéter, sous Linux le symbole "~" est un synonyme de votre dossier personnel. Chez moi cela signifie donc "/home/mateo21".

Cette indication de l'invite de commandes est pratique mais il faut savoir qu'il y a un autre moyen de connaître le nom du dossier

actuel. C'est la commande `pwd`.

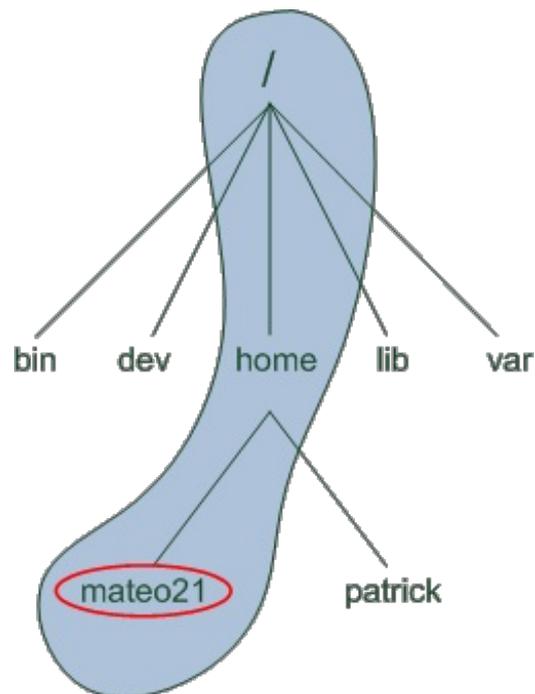
`pwd` est l'abréviation de "Print Working Directory", c'est-à-dire "Afficher le dossier actuel".

C'est une commande très simple qui ne prend aucun paramètre (on commence doucement hein 🤪), vous pouvez la tester :

Code : Console

```
mateo21@mateo21-desktop:~$ pwd
/home/mateo21
```

Cela confirme bien ce que je vous disais : je me trouve en ce moment dans le dossier `/home/mateo21`.



A tout moment, si vous vous sentez perdu dans les méandres des dossiers, souvenez-vous de la commande `pwd`, elle vous dira où vous êtes ! 🤪

which : connaître l'emplacement d'une commande

Même si cette commande ne nous est pas indispensable, j'ai pensé que c'était une bonne idée de vous la montrer dès le début afin que vous puissiez vous familiariser un peu plus encore avec le système de fichiers de Linux 🤪

Alors que fait cette commande ? Elle vous permet de localiser la position du programme correspondant à une commande. Je m'explique : chaque commande sous Linux correspond à un programme. Ainsi, `pwd` qu'on vient de voir *est* un programme.

Une commande n'est rien d'autre qu'un programme qu'on peut appeler n'importe quand n'importe où dans la console.

La commande `which` prend un paramètre : le nom de la commande dont vous voulez connaître l'emplacement. Testons sur `pwd` :

Code : Console

```
mateo21@mateo21-desktop:~$ which pwd
/bin/pwd
```

pwd se trouve donc dans le dossier /bin/ ! Le "pwd" à la fin n'est pas un dossier mais le nom du programme lui-même.



Vous noterez que les programmes sous Linux ne possèdent en général pas d'extension (contrairement à Windows où l'extension utilisée est en général .exe).

Tous les programmes ne sont pas situés dans un même dossier. Pour vous en rendre compte, testez l'emplacement d'une autre commande... tenez, par exemple la commande which !

On va donc devoir écrire which which dans la console (oui je sais, je suis un gros tordu 🤪) :

Code : Console

```
mateo21@mateo21-desktop:~$ which which
/usr/bin/which
```

Cette fois, le programme ne se trouve pas dans /bin/ mais dans /usr/bin/ !

ls : lister les fichiers et dossiers

ls est une des toutes premières commandes que nous avons essayées dans le chapitre précédent. Nous allons rentrer ici plus dans le détail de son fonctionnement (et de ses nombreux paramètres 🤪).

Commençons par taper "ls" sans paramètre depuis notre dossier personnel :

Code : Console

```
mateo21@mateo21-desktop:~$ ls
Desktop  Examples  images  log  tutos
```

(oui j'ai créé quelques dossiers pour mon usage personnel, ne vous étonnez pas si vous n'avez pas les mêmes 😊)

Ubuntu active la coloration des fichiers et dossiers par défaut, vous devriez donc voir des couleurs chez vous. Les dossiers apparaissent en bleu foncé. Vous remarquerez que le dossier Examples est en bleu clair : cela signifie que c'est un raccourci vers un dossier qui se trouve en fait ailleurs sur le disque.



Si la couleur ne s'affiche pas, vous pouvez rajouter le paramètre `--color=auto`, comme ceci : `ls --color=auto`. Si vous ne voulez pas de la couleur au contraire, essayez le paramètre `--color=none`. Pour éviter d'avoir à retaper à chaque fois ce long paramètre, il faut modifier un fichier de configuration, mais on verra cela plus tard.

La commande "ls" accepte un grand nombre de paramètres. Ca ne sert à rien que je vous fasse la liste complète ici (ce serait bien trop long), par contre je vais vous faire découvrir les paramètres les plus utiles. Ca vous permettra de vous entraîner à utiliser et combiner des paramètres !

-a : afficher tous les fichiers et dossiers cachés

Sous Linux, on peut "cacher" des fichiers et dossiers. Ce n'est pas une protection, car on peut toujours les réafficher si on veut, mais ça évite d'encombrer l'affichage de la commande "ls".

Votre dossier home est un très bon exemple car il est rempli de fichiers et dossiers cachés. En ajoutant le paramètre -a, on peut voir tous ces fichiers et dossiers cachés :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -a
.                .gnome          .nano_history
..              .gnome2        .nautilus
.armagetron     .gnome2_private .openoffice.org2
.bash_history   .gnome_private  .pgadmin3
.bash_logout    .gstreamer-0.10 .pgpass
.bashrc         .gtkrc-1.2-gnome2 .profile
.blender       .gweled         .qt
.config         .ICEauthority   .recently-used
.DCOPserver_mateo21-desktop__0 .icons          .recently-used.xbel
.DCOPserver_mateo21-desktop_:0 images          .ssh
Desktop        .inkscape       .sudo_as_admin_successful
.dmrc          .java           .themes
.emilia        .jedit          .thumbnails
.esd_auth      .kde            .Trash
.evolution     .lessshst      .tsclient
Examples       .lgames        tutos
.face          .local          .update-manager-core
.fontconfig    log             .update-notifier
.gaim          .macromedia    .vlc
.gconf         .mcp            .wormux
.gconfd       .mcpirc        .Xauthority
.geany         .metacity      .xine
.gimp-2.2     .mozilla       .xsession-errors
.gksu.lock    .mozilla-thunderbird
```

Vous comprenez peut-être mieux maintenant pourquoi tous ces fichiers et dossiers sont cachés : c'est encombrant 😊
Certains éléments commençant par un point "." sont des dossiers, d'autres sont des fichiers. La meilleure façon de faire la distinction est de comparer les couleurs : les dossiers en bleu, le reste dans la couleur par défaut (par exemple le blanc ou le noir).

Les deux premiers éléments sont assez intrigants "." et "..". Le premier représente en fait le dossier actuel, et ".." représente le dossier parent, c'est-à-dire le dossier précédent dans l'arborescence. Par exemple, là je suis dans /home/mateo21, "." représente donc le dossier /home.



Le paramètre -A (un A majuscule au lieu d'un a minuscule) a pratiquement la même signification : ça affiche la même chose sauf ces éléments "." et "..". Comme quoi il faut faire attention aux majuscules !

-F : indique le type d'élément

Ce paramètre est surtout utile pour ceux qui n'ont pas (ou ne veulent pas) afficher la couleur dans la console. Il rajoute à la fin des éléments un symbole pour qu'on puisse faire la distinction entre les dossiers, fichiers, raccourcis...

Code : Console

```
mateo21@mateo21-desktop:~$ ls -F
Desktop/  Examples@  images/  log/  tutos/
```

Grâce à ça on peut voir que tous les éléments sont des dossiers, sauf Examples qui est un raccourci (d'où la présence du @).

-l : liste détaillée

Le paramètre "-l" (la lettre L en minuscule) est un des plus utiles. Il affiche une liste détaillant chaque élément du dossier :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -l
```

```
total 16
drwxr-xr-x 2 mateo21 mateo21 4096 2007-09-24 17:22 Desktop
lrwxrwxrwx 1 mateo21 mateo21 26 2007-09-19 18:31 Examples -
> /usr/share/example-content
drwxr-xr-x 2 mateo21 mateo21 4096 2007-09-25 15:17 images
drwxr-xr-x 3 mateo21 mateo21 4096 2007-09-25 11:11 log
drwxr-xr-x 3 mateo21 mateo21 4096 2007-09-19 19:51 tutos
```

Il y a un élément par ligne.

Chaque colonne a sa propre signification. De gauche à droite :

1. Droits sur le fichier (on fera un chapitre entier pour expliquer comment fonctionnent les droits sous Linux).
2. Nombre de liens physiques (cela ne nous intéresse pas ici).
3. Nom de la personne propriétaire du fichier (là c'est moi !). Si le fichier avait été créé par quelqu'un d'autre, par exemple patrick, on aurait vu son nom à la place.
4. Groupe auquel appartient le fichier (on en reparlera dans le chapitre sur les droits). Il se peut que le nom du groupe soit le même que celui du propriétaire.
5. Taille du fichier, en octets.
6. Date de dernière modification.
7. Nom du fichier (ou dossier).

Vous noterez aussi que dans le cas du raccourci (on parle de *lien symbolique*), la commande nous précise vers où pointe le raccourci (en l'occurrence `/usr/share/example-content`).

-h : afficher la taille en Ko, Mo, Go...

Quand on fait un `ls -l`, la taille est affichée en octets. Seulement, ce n'est parfois pas très lisible. Par exemple :

Code : Console

```
mateo21@mateo21-desktop:~/Examples$ ls -l
total 9500
-rw-r--r-- 1 root root 3576296 2007-04-03 17:05 Experience ubuntu.ogg
-rw-r--r-- 1 root root 229674 2007-04-03 17:05 fables_01_01_aesop.spx
-rw-r--r-- 1 root root 848013 2007-04-03 17:05 gimp-ubuntu-splash.xcf
-rw-r--r-- 1 root root 1186219 2007-04-03 17:05 kubuntu-leaflet.png
-rw-r--r-- 1 root root 47584 2007-04-03 17:05 logo-Edubuntu.png
```

Si vous rajoutez le paramètre "h" (h pour Human Readable, c'est-à-dire "lisible par un humain"), vous obtenez des tailles de fichier beaucoup plus lisibles (normal, vous êtes des humains 🤪) :

Code : Console

```
mateo21@mateo21-desktop:~/Examples$ ls -lh
total 9,3M
-rw-r--r-- 1 root root 3,5M 2007-04-03 17:05 Experience ubuntu.ogg
-rw-r--r-- 1 root root 225K 2007-04-03 17:05 fables_01_01_aesop.spx
-rw-r--r-- 1 root root 829K 2007-04-03 17:05 gimp-ubuntu-splash.xcf
-rw-r--r-- 1 root root 1,2M 2007-04-03 17:05 kubuntu-leaflet.png
-rw-r--r-- 1 root root 47K 2007-04-03 17:05 logo-Edubuntu.png
```

Grâce à ça, on voit alors bien que le fichier `Experience ubuntu.ogg` fait 3,5 Mo, `logo-Edubuntu.png` fait 47 Ko, etc.

-t : trier par date de dernière modification

Voilà une option dont l'intérêt est sous-estimé ! `-t` permet en effet de trier par date de dernière modification, au lieu de trier par ordre alphabétique comme cela est fait par défaut.

On voit ainsi en premier le dernier fichier que l'on a modifié, et en dernier celui auquel on n'a pas touché depuis le plus longtemps :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -lt
total 16
drwxr-xr-x 2 mateo21 mateo21 4096 2007-09-25 15:17 images
drwxr-xr-x 3 mateo21 mateo21 4096 2007-09-25 11:11 log
drwxr-xr-x 2 mateo21 mateo21 4096 2007-09-24 17:22 Desktop
drwxr-xr-x 3 mateo21 mateo21 4096 2007-09-19 19:51 tutos
lrwxrwxrwx 1 mateo21 mateo21 26 2007-09-19 18:31 Examples -
> /usr/share/example-content
```

De toute évidence, le dernier fichier (ici dossier) modifié est `images`. En revanche, comme je n'ai jamais touché à `Examples`, il est normal qu'il apparaisse en dernier.

En pratique, je combine `-t` avec `-r` qui renverse l'ordre d'affichage des fichiers. Je préfère en effet avoir le dernier fichier modifié en bas de la liste, c'est plus pratique à l'usage dans la console.

Et comme je suis un gros bourrin, je combine un peu tous les paramètres que l'on vient de voir, ce qui donne un beau `ls -larth` qui contient toutes les options que j'aime 🤪

Code : Console

```
mateo21@mateo21-desktop:~$ ls -larth
total 380K
-rw----- 1 mateo21 mateo21 26 2007-09-19 16:40 .dmrc
-rw-r--r-- 1 mateo21 mateo21 89 2007-09-19 16:40 .gtkrc-1.2-gnome2
-rw----- 1 mateo21 mateo21 16 2007-09-19 16:40 .esd_auth
drwx----- 2 mateo21 mateo21 4,0K 2007-09-19 16:40 .update-notifier
lrwxrwxrwx 1 mateo21 mateo21 26 2007-09-19 18:31 Examples -
> /usr/share/example-content
-rw-r--r-- 1 mateo21 mateo21 220 2007-09-19 18:31 .bash_logout
drwxr-xr-x 4 root root 4,0K 2007-09-19 18:31 ..
drwxr-xr-x 10 mateo21 mateo21 4,0K 2007-09-25 16:03 .jedit
-rw-r--r-- 1 mateo21 mateo21 1,1K 2007-09-25 16:03 .pgadmin3
drwxr-xr-x 47 mateo21 mateo21 4,0K 2007-09-25 16:03 .
-rw----- 1 mateo21 mateo21 1,8K 2007-09-25 16:38 .bash_history
-rw----- 1 mateo21 mateo21 17K 2007-09-25 16:52 .recently-used
drwx----- 2 mateo21 mateo21 4,0K 2007-09-25 16:54 .gconfd
-rw----- 1 mateo21 mateo21 39 2007-09-25 17:18 .lessht
-rw-r--r-- 1 mateo21 mateo21 53K 2007-09-25 17:21 .xsession-errors
```

Note : j'ai volontairement réduit cette liste car il y a beaucoup de fichiers dans mon home. En pratique la liste est beaucoup plus grande 🤪

Le fichier caché `.xsession-errors` est donc le dernier qui a été modifié dans ce dossier sur mon ordinateur.



Plutôt que d'avoir à réécrire "`ls -larth`" à chaque fois (c'est un peu long), on peut créer un alias, c'est-à-dire une commande synonyme. Par exemple, j'ai créé l'alias "`ll`" (deux fois L) qui est automatiquement transformé par Linux en "`ls -larth`".

On verra comment créer des alias lorsqu'on saura se servir d'un éditeur de fichiers.

cd : changer de dossier

Bon, mine de rien depuis tout à l'heure on est coincé dans notre dossier "`home`" et on aimerait bien bouger de là.

Le moment est venu de déplacer le navire moussaillon ! 🚢

La commande que nous allons étudier ici s'appelle `cd`, abréviation de "Change Directory" (changer de dossier). C'est une commande très importante que vous allez utiliser quelques milliers de fois dans votre vie (au moins 🤖).

Contrairement à `ls`, la commande `cd` ne prend pas plein de paramètres mais juste un seul : le nom du dossier dans lequel vous souhaitez aller.

Si on veut aller à la racine, il suffit de taper `cd /` :

Code : Console

```
mateo21@mateo21-desktop:~$ cd /
mateo21@mateo21-desktop:/$ pwd
/
```

Après avoir tapé `cd /`, on se retrouve à la racine. L'invite de commandes a changé et le `~` a été remplacé par un `/`. Si vous êtes sceptique, un petit coup de `pwd` devrait vous confirmer que vous êtes bien dans `/` 😊

Bon listons les fichiers et dossiers contenus dans `/` :

Code : Console

```
mateo21@mateo21-desktop:/$ ls -F
bin/      dev/      initrd/      lib/          mnt/  root/  sys/  var/
boot/     etc/      initrd.img@  lost+found/  opt/   sbin/  tmp/  vmlinuz@
cdrom@    home/    initrd.img.old@  media/        proc/  srv/   usr/  vmlinuz.old@
```

Vous y retrouvez un grand nombre de dossiers que je vous ai décrits au début du chapitre. Allons dans le sous-dossier `usr` :

Code : Console

```
mateo21@mateo21-desktop:/$ cd usr
```

Voyons voir ce qu'il y a là-dedans...

Code : Console

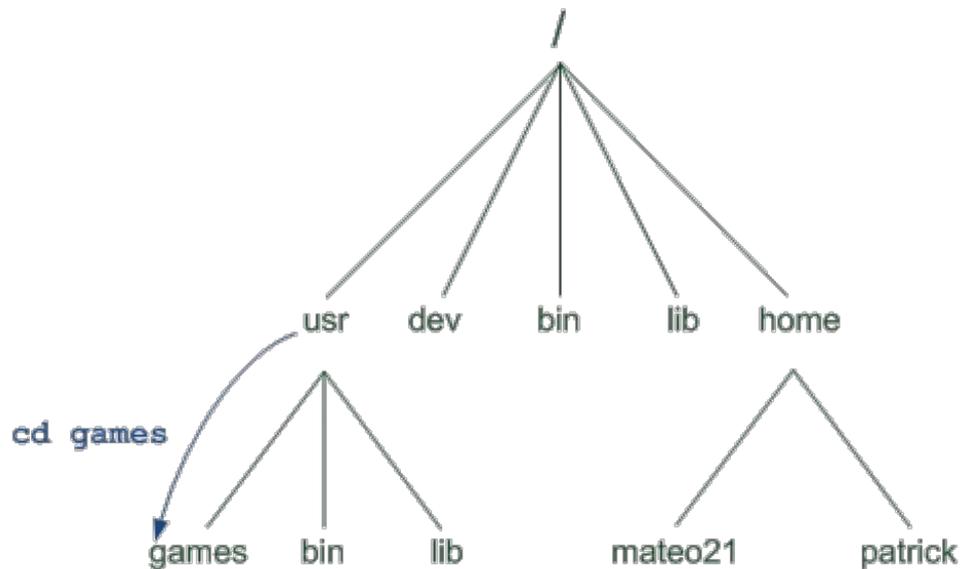
```
mateo21@mateo21-desktop:/usr$ ls -F
bin/  games/  include/  lib/  local/  sbin/  share/  src/  X11R6/
```

Chez moi, il n'y a que des dossiers. Hummm, le dossier `games` m'intrigue, voyons voir ce que j'ai comme jeux :

Code : Console

```
mateo21@mateo21-desktop:/usr$ cd games
mateo21@mateo21-desktop:/usr/games$
```

Schématiquement, on vient de faire ça :



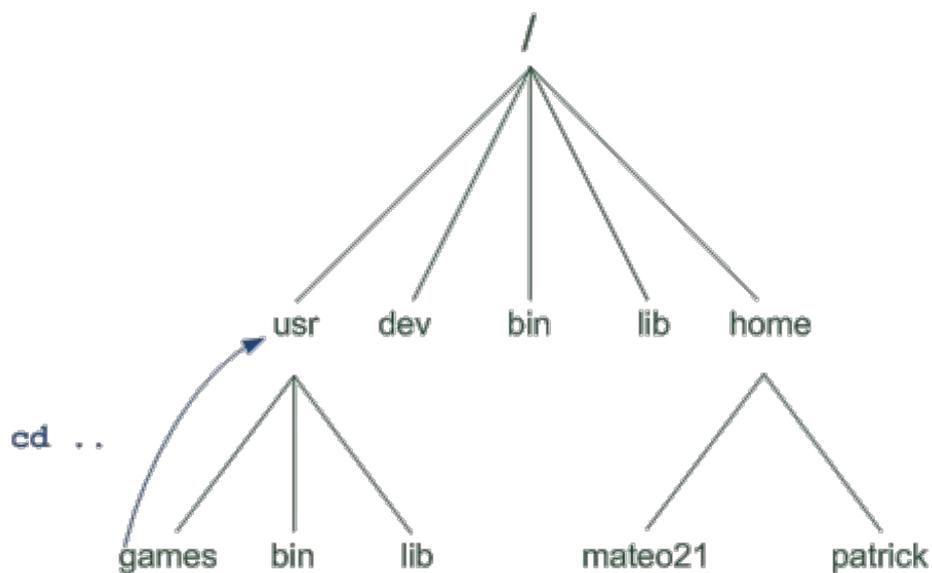
Bon, mais maintenant supposons que j'aie envie de revenir au dossier précédent, aussi appelé dossier parent, c'est-à-dire "/usr". Comment je fais ?

Il faut utiliser les deux points comme ceci :

Code : Console

```
mateo21@mateo21-desktop:/usr/games$ cd ..  
mateo21@mateo21-desktop:/usr$
```

Et hop là, on est revenu au dossier parent !



Si on avait voulu reculer de 2 dossiers parents, on aurait écrit ../../ (reviens en arrière, puis reviens en arrière). Ca nous aurait ramené à la racine :

Code : Console

```
mateo21@mateo21-desktop:/usr/games$ cd ../../
```

```
mateo21@mateo21-desktop:/$
```



Eh mais en fait, il y a plusieurs façons d'aller dans un dossier non ? Tout à l'heure on est allé à la racine en tapant `cd /`, et maintenant en tapant `cd ../..`. C'est quoi cette affaire ? 🤔

Il y a en fait 2 façons de changer de dossier : en indiquant un chemin relatif, ou en indiquant un chemin absolu.

Les chemins relatifs

Un chemin relatif est un chemin qui dépend du dossier dans lequel vous vous trouvez. Tout à l'heure, on est allé dans le sous-dossier `games` de `/usr` en tapant juste son nom :

Code : Console

```
mateo21@mateo21-desktop:/usr$ cd games
```

En faisant cela, on utilise un chemin relatif, c'est-à-dire relatif au dossier actuel. Quand on met juste le nom d'un dossier comme ici, cela indique que l'on veut aller dans un sous-dossier.

Si on fait `cd games` depuis la racine, ça va planter :

Code : Console

```
mateo21@mateo21-desktop:/$ cd games
bash: cd: games: Aucun fichier ou répertoire de ce type
```

Je crois que le message d'erreur est assez clair : il n'y a aucun dossier `games` dans `/` 🤔

Pour se rendre dans `games`, il faut d'abord indiquer le dossier qui le contient (`usr`) :

Code : Console

```
mateo21@mateo21-desktop:/$ cd usr/games
mateo21@mateo21-desktop:/usr/games$
```

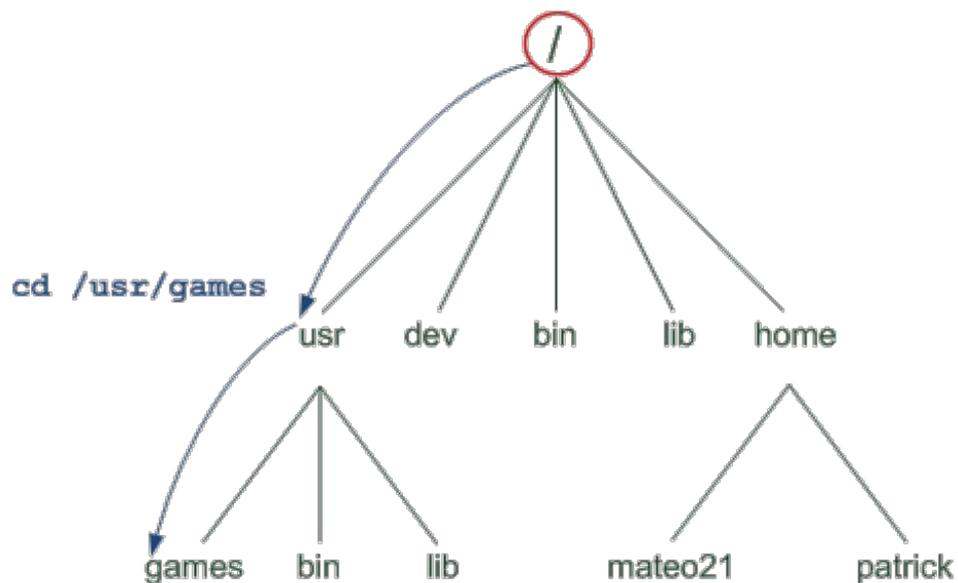
Les chemins absolus

Contrairement aux chemins relatifs, les chemins absolus fonctionnent quel que soit le dossier dans lequel on se trouve.

Un chemin absolu est facile à reconnaître : il commence toujours par la racine (`/`). Vous devez faire ensuite la liste des dossiers dans lesquels vous voulez entrer. Par exemple, supposons que je sois dans `/home/mateo21` et que je souhaite aller dans `/usr/games`. Avec un chemin absolu :

Code : Console

```
mateo21@mateo21-desktop:~$ cd /usr/games
mateo21@mateo21-desktop:/usr/games$
```



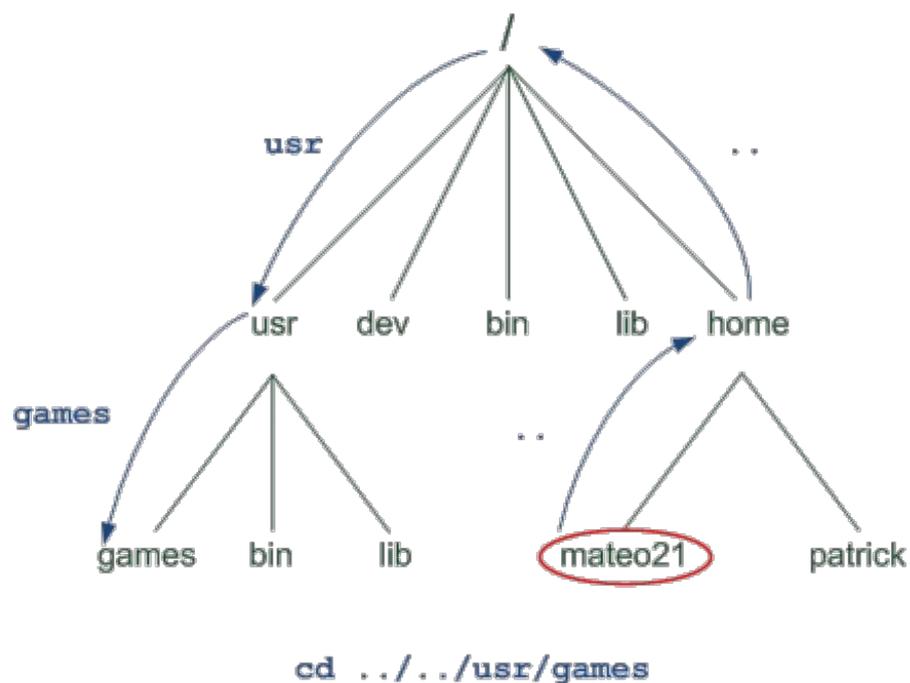
Le schéma montre bien qu'on *part* de la racine `/` pour indiquer où on veut aller.

Si on avait voulu faire la même chose à coup de chemin relatif, il aurait fallu écrire :

Code : Console

```
mateo21@mateo21-desktop:~$ cd ../../usr/games/
mateo21@mateo21-desktop:/usr/games$
```

Ce qui signifie "reviens en arrière (donc dans `/home`) puis reviens en arrière (donc dans `/`), puis va en avant dans `usr`, puis va en avant dans `games`".



Ici, comme c'est un chemin relatif, on *part* du dossier dans lequel on se trouve (`/home/mateo21`) et on indique à la machine le chemin à suivre à partir de là pour aller dans le dossier qu'on veut.



Un chemin absolu est donc facile à reconnaître, car on part toujours de la racine /.
Un chemin relatif peut aussi s'avérer très pratique et plus court (ça dépend des cas).
Ce sera à vous de choisir à chaque fois comment vous voulez écrire votre chemin. Vous avez le choix 😊

Retour au répertoire home

Si vous voulez retourner dans votre répertoire home personnel, plusieurs solutions s'offrent à vous :

- **La brutale** : il suffit d'écrire le chemin absolu en entier. Cela donne :

Code : Console

```
mateo21@mateo21-desktop:/usr/games$ cd /home/mateo21/  
mateo21@mateo21-desktop:~$
```

- **La maligne** : plus court et plus pratique, vous pouvez utiliser l'alias ~ qui signifie la même chose. Cela donne :

Code : Console

```
mateo21@mateo21-desktop:/usr/games$ cd ~  
mateo21@mateo21-desktop:~$
```

- **La super-maligne** : si vous ne mettez aucun paramètre à la commande cd, ça vous ramène aussi dans votre répertoire personnel 😊

Code : Console

```
mateo21@mateo21-desktop:/usr/games$ cd  
mateo21@mateo21-desktop:~$
```

Autocomplétion du chemin

Cette astuce est vitale, si vous ne vous en servez pas, vous passez à côté d'une des plus importantes astuces de la console.

L'idée est simple : taper `cd /usr/games/trucbidule` c'est bien, mais c'est parfois un peu long de tout écrire. On a la flemme. Vous avez la flemme. Oui vous. Alors vous allez justement demander à l'ordinateur de compléter le chemin tout seul !

L'autocomplétion de chemin fonctionne de la même manière que l'autocomplétion de commande qu'on a vue dans le chapitre précédent : avec la touche Tab (Tabulation). Faites le test avec moi. Commencez par vous placer dans /usr :

Code : Console

```
mateo21@mateo21-desktop:~$ cd /usr  
mateo21@mateo21-desktop:/usr$
```

Tapez ensuite juste `cd ga`, puis appuyez sur Tab. C'est magique, le nom du dossier a été automatiquement complété !

Code : Console

```
mateo21@mateo21-desktop:/usr$ cd games/
```

Revenez maintenant dans /usr (en faisant `cd ..` par exemple) et essayez de taper juste `cd l`, puis faites Tab. Rien ne se passe : cela signifie que l'ordinateur n'a pas trouvé de dossier qui corresponde au début de votre recherche, ou alors qu'il y en a plusieurs qui commencent par "l". Refaites encore Tab :

Code : Console

```
mateo21@mateo21-desktop:/usr$ cd l
lib/  local/
mateo21@mateo21-desktop:/usr$ cd l
```

On vient de vous donner la liste des dossiers qui commencent par "l" ! Cela signifie qu'il faut préciser votre recherche parce que sinon l'ordinateur ne peut pas deviner dans quel dossier vous voulez entrer. Ça tombe bien, la commande a été réécrite en-dessous, vous n'avez plus qu'à ajouter une lettre plus précise, par exemple "o" pour que Linux devine que vous voulez aller dans le dossier "local". Tapez donc "o", puis à nouveau Tab, et le nom sera complété !

Code : Console

```
mateo21@mateo21-desktop:/usr$ cd local/
```

Faites des tests pour vous entraîner à utiliser l'autocomplétion, c'est vraiment très important. Vous allez voir c'est intuitif et vraiment pratique ! 😊

du : taille occupée par les dossiers

La commande "du", pour Disk Usage (utilisation du disque) vous donne des informations sur la taille qu'occupe les dossiers sur votre disque.

Placez-vous pour commencer dans /usr/games, et tapez "du" :

Code : Console

```
mateo21@mateo21-desktop:~$ cd /usr/games
mateo21@mateo21-desktop:/usr/games$ du
5732  .
```

Comme ce dossier ne contient pas de sous-dossier, la commande "du" nous renvoie la taille totale que font les fichiers contenus dans le dossier.

Si vous allez dans votre home en revanche, celui-ci contient beaucoup de sous-dossiers. Dans ce cas, la commande "du" va renvoyer la taille de chacun des sous-dossiers, puis la taille totale à la fin (".") :

Code : Console

```
mateo21@mateo21-desktop:/usr/games$ cd
mateo21@mateo21-desktop:~$ du
400  ./Trash
4    ./themes
32   ./mozilla-thunderbird/8vyw6pqp.default/Mail/Local Folders
36   ./mozilla-thunderbird/8vyw6pqp.default/Mail
12   ./mozilla-thunderbird/8vyw6pqp.default/US
...
...
264  ./jedit/jars
4    ./jedit/macros
380  ./jedit/settings-backup
856  ./jedit
82484 .
```

(j'ai volontairement coupé la liste car elle est très longue 😊)

-h : la taille pour les humains

Ce qui est bien, c'est que les commandes reprennent souvent les mêmes paramètres. Ainsi, on avait vu -h pour ls, eh bien ce paramètre est le même pour avoir des tailles "humaines" avec "du" !

Code : Console

```
mateo21@mateo21-desktop:~$ du -h
400K   ./Trash
4,0K   ./themes
32K    ./mozilla-thunderbird/8vyw6pqp.default/Mail/Local Folders
36K    ./mozilla-thunderbird/8vyw6pqp.default/Mail
12K    ./mozilla-thunderbird/8vyw6pqp.default/US
...
...
264K   ./jedit/jars
4,0K   ./jedit/macros
380K   ./jedit/settings-backup
856K   ./jedit
81M    .
```

Mon dossier home prend donc 81 Mo d'espace disque, son sous-dossier caché ".jedit" prend 856 Ko, etc etc.

-a : afficher la taille des dossiers ET des fichiers

Par défaut, "du" n'affiche que la taille des dossiers. Pour avoir aussi la taille des fichiers qu'ils contiennent, rajoutez l'option -a (all) :

Code : Console

```
mateo21@mateo21-desktop:~$ du -ah
...
8,0K   ./jedit/settings-backup/abbrevs~5~
24K    ./jedit/settings-backup/history~1~
8,0K   ./jedit/settings-backup/abbrevs~4~
380K   ./jedit/settings-backup
44K    ./jedit/pluginMgr-Cached.xml.gz
856K   ./jedit
81M    .
```

-s : avoir juste le grand total

Pour n'avoir que l'espace total occupé par le dossier et donc ne pas afficher le détail des sous-dossiers, utilisez -s (que je combine à -h pour plus de lisibilité) :

Code : Console

```
mateo21@mateo21-desktop:~$ du -sh
81M    .
```

Je vois ainsi que mon dossier home fait 81 Mo (rappel : le symbole point "." signifie "*le dossier actuel*").

Félicitations moussaillon, vous voilà désormais capable de naviguer partout sur votre disque 🤖

N'hésitez pas à aller farfouiller un peu partout sur votre disque pour vous faire la main, vous ne risquez pas de faire quoi que ce soit de mal avec ces commandes, elles ne font que lire le disque et ne l'altèrent pas 😊

Prenez en particulier l'habitude d'utiliser l'autocomplétion de chemin (touche Tab), on gagne vraiment un temps fou avec ça !

Dans le prochain chapitre, nous apprendrons à visualiser le contenu des fichiers, à les déplacer, à les copier et même à les supprimer. Il faudra faire un peu plus attention 😊

Manipuler les fichiers

Après avoir vu comment étaient organisés les fichiers sous Linux, nous allons apprendre à les manipuler !

Par exemple, comment faire pour afficher le contenu d'un fichier ?
Comment le déplacer, le copier, le supprimer ?

C'est donc un chapitre à la fois simple et riche qui vous attend, durant lequel vous allez apprendre beaucoup de nouvelles commandes basiques de Linux qu'il vous faut connaître absolument ! 😊

cat & less : afficher un fichier

Nous allons d'abord voir comment afficher le contenu d'un fichier. Il y a en gros 2 commandes basiques sous Linux qui permettent de faire cela :

- cat
- less

Aucune de ces commandes ne permet d'éditer un fichier, elles permettent juste de le **voir**. Nous étudierons l'édition plus tard, ça mérite au moins un autre chapitre entier 😊



Mais... pourquoi 2 commandes pour afficher un fichier ? Une seule n'aurait pas suffit ?

En fait, chacune a ses spécificités ! Nous allons les voir dans le détail.

Pour nos exemples, nous allons travailler sur un fichier qui existe déjà : syslog. Il se trouve dans le dossier /var/log. Commencez par vous y rendre :

Code : Console

```
mateo21@mateo21-desktop:~$ cd /var/log
```

Ce dossier contient plusieurs fichiers de *log*, c'est-à-dire des fichiers qui gardent une trace de l'activité de votre ordinateur. Vous pouvez faire la liste si vous voulez, en tapant ls :

Code : Console

```
mateo21@mateo21-desktop:/var/log$ ls
acpid                daemon.log.0        kern.log.0          scrollkeeper.log.2
acpid.1.gz           daemon.log.1.gz     kern.log.1.gz       syslog
acpid.2.gz           daemon.log.2.gz     kern.log.2.gz       syslog.0
acpid.3.gz           daemon.log.3.gz     kern.log.3.gz       syslog.1.gz
acpid.4.gz           debug               lastlog             syslog.2.gz
apparmor             debug.0             lpr.log             syslog.3.gz
apport.log           debug.1.gz          mail.err            syslog.4.gz
apport.log.1         debug.2.gz          mail.info           syslog.5.gz
apport.log.2.gz      debug.3.gz          mail.log            syslog.6.gz
apport.log.3.gz      dist-upgrade        mail.warn           udev
apport.log.4.gz      dmesg               messages           unattended-
upgrades
apport.log.5.gz      dmesg.0            messages.0         user.log
apt                  dmesg.1.gz         messages.1.gz      user.log.0
auth.log             dmesg.2.gz         messages.2.gz      user.log.1.gz
auth.log.0           dmesg.3.gz         messages.3.gz      user.log.2.gz
auth.log.1.gz        dmesg.4.gz         news                user.log.3.gz
auth.log.2.gz        dpkg.log            popularity-contest  uucp.log
auth.log.3.gz        dpkg.log.1          popularity-contest.0 wttmp
bittorrent           dpkg.log.2.gz      popularity-contest.1.gz wttmp.1
boot                 faillog             popularity-contest.2.gz wvdialconf.log
bootstrap.log        fontconfig.log     popularity-contest.3.gz Xorg.0.log
btmtp                fsck                pycentral.log       Xorg.0.log.old
```

btm.p.1	gdm	samba
cups	installer	scrollkeeper.log
daemon.log	kern.log	scrollkeeper.log.1

Le fichier sur lequel nous allons travailler, syslog, contient des informations de log de ce qui s'est passé récemment sur l'ensemble de votre ordinateur.



Vous noterez qu'il est fréquent de voir des fichiers sans extension sous Linux. Notre fichier s'appelle syslog tout court, et non pas syslog.txt ou syslog.log comme on pourrait avoir l'habitude de le voir sous Windows. Un fichier sans extension peut être ouvert et lu sans aucun problème comme n'importe quel autre fichier.

cat : afficher tout le fichier

La commande cat permet d'afficher tout le contenu d'un fichier dans la console d'un coup.

Il vous suffit d'indiquer en paramètre le nom du fichier que vous voulez afficher, en l'occurrence syslog :

Code : Console

```
mateo21@mateo21-desktop:/var/log$ cat syslog
Nov 14 00:44:23 mateo21-desktop syslogd 1.4.1#21ubuntu3: restart.
Nov 14 00:44:23 mateo21-desktop anacron[6725]: Job `cron.daily' terminated
Nov 14 00:44:23 mateo21-desktop anacron[6725]: Normal exit (1 job run)
Nov 14 00:44:25 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:44:51 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:45:08 mateo21-
desktop NetworkManager: <debug> [1194997508.332093] nm_device_802_11_wireless_get_a
Nov 14 00:45:08 mateo21-
desktop NetworkManager: <info> User Switch: /org/freedesktop/NetworkManager/Device
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Deactivating device eth1.
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1): cancelli
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1) cancellat
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1): waiting
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1) cancellat
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1): cancelle
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: sending command 'DISAB
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: response was 'OK'
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: sending command 'AP_SC
Nov 14 00:45:16 mateo21-desktop NetworkManager: nm_act_request_get_ap: assertion `r
Nov 14 00:45:16 mateo21-desktop NetworkManager: nm_act_request_get_stage: assertion
Nov 14 00:45:16 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:45:21 mateo21-desktop NetworkManager: nm_act_request_get_ap: assertion `r
Nov 14 00:45:21 mateo21-desktop NetworkManager: ap_is_auth_required: assertion `ap
Nov 14 00:45:21 mateo21-
desktop NetworkManager: <info> Activation (eth1/wireless): association took too lc
Nov 14 00:45:21 mateo21-desktop NetworkManager: nm_dbus_get_user_key_for_network: a
Nov 14 00:47:45 mateo21-desktop init: tty4 main process (4517) killed by TERM signa
Nov 14 00:47:45 mateo21-desktop init: tty5 main process (4518) killed by TERM signa
Nov 14 00:47:45 mateo21-desktop init: tty2 main process (4520) killed by TERM signa
Nov 14 00:47:45 mateo21-desktop init: tty3 main process (4522) killed by TERM signa
Nov 14 00:47:45 mateo21-desktop init: tty1 main process (4524) killed by TERM signa
Nov 14 00:47:45 mateo21-desktop init: tty6 main process (4525) killed by TERM signa
Nov 14 00:47:46 mateo21-desktop avahi-daemon[5390]: Got SIGTERM, quitting.
Nov 14 00:47:48 mateo21-desktop exiting on signal 15
Nov 14 00:48:42 mateo21-desktop syslogd 1.4.1#21ubuntu3: restart.
Nov 14 00:48:42 mateo21-desktop kernel: Inspecting /boot/System.map-2.6.22-14-gener
Nov 14 00:48:42 mateo21-desktop kernel: Loaded 25445 symbols from /boot/System.map-
Nov 14 00:48:42 mateo21-desktop kernel: Symbols match kernel version 2.6.22.
Nov 14 00:48:42 mateo21-desktop kernel: No module symbols loaded - kernel modules n
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] Linux version 2.6.22-14-
generic (buildd@palmer) (gcc version 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-
16ubuntu2)) #1 SMP Sun Oct 14 23:05:12 GMT 2007 (Ubuntu 2.6.22-14.46-generic)
```

```
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-provided physical RAM m
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 0000000000000000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 0000000000009f800
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 000000000000e0000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 00000000001000000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 0000000007f680000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 0000000007f68e000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 0000000007f700000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 000000000e0000000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 00000000fec000000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 00000000fed000000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 00000000fed140000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 00000000fed1c0000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 00000000fee000000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-e820: 00000000ff0000000
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] 1142MB HIGHMEM available.
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] 896MB LOWMEM available.
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] found SMP MP-table at 000f77
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] Entering add_active_range(0,
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] Zone PFN ranges:
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   DMA             0 ->    4
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   Normal          4096 ->   229
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   HighMem       229376 ->   521
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] early_node_map[1] active PFN
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   0:             0 -> 521856
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] On node 0 totalpages: 521856
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   DMA zone: 32 pages used for
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   DMA zone: 0 pages reserved
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   DMA zone: 4064 pages, LIFC
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   Normal zone: 1760 pages us
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   Normal zone: 223520 pages,
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   HighMem zone: 2285 pages u
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000]   HighMem zone: 290195 pages
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] DMI present.
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: RSDP signature @ 0xc00
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: RSDP 000F7650, 0014 (r
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: RSDT 7F6889F5, 004C (r1 MSTEST TESTONLY 60400
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: FACP 7F68DCAA, 0074 (r1 INTEL CALISTGA 60400
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: DSDT 7F68A0EC, 3BBE (r1 INTEL CALISTGA 60400
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: FACS 7F68EFC0, 0040
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: APIC 7F68DD1E, 0068 (r1 INTEL CALISTGA 60400
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: HPET 7F68DD86, 0038 (r1 INTEL CALISTGA 60400
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: MCFG 7F68DDBE, 003C (r1 INTEL CALISTGA 60400
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: SLIC 7F68DDFA, 0176 (r1 MSTEST TESTONLY 60400
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: APIC 7F68DF70, 0068 (r1 PTLTD ^I APIC 6040
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: BOOT 7F68DFD8, 0028 (r1 PTLTD $SBFTBL$ 60400
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: SSDT 7F689A9D, 064F (r1 SataRe SataPri 10
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: SSDT 7F68940B, 0692 (r1 SataRe SataSec 10
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] ACPI: SSDT 7F688A41, 04F6 (r1 PmRef CpuPm 30
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: BIOS bug: multiple API
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: If "acpi_apic_instance
acpi@vger.kernel.org
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: PM-Timer IO Port: 0x10
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: Local APIC address 0xf
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: LAPIC (acpi_id[0x00] 1
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] Processor #0 6:15 APIC versi
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: LAPIC (acpi_id[0x01] 1
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] Processor #1 6:15 APIC versi
```

```
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: LAPIC_NMI (acpi_id[0x0
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: LAPIC_NMI (acpi_id[0x0
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: IOAPIC (id[0x01] addre
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] IOAPIC[0]: apic_id 1, versic
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: INT_SRC_OVR (bus 0 bus
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] ACPI: INT_SRC_OVR (bus 0 bus
...

```

Comme le fichier est très gros (il fait plusieurs centaines de lignes), je n'ai pas copié tout ce qui s'est affiché dans ma console 😊
Ne vous étonnez pas si vous voyez tout s'afficher d'un coup : c'est normal, c'est le but. La commande `cat` vous envoie tout le fichier à la figure. Elle est plus adaptée lorsqu'on travaille sur des petits fichiers que sur des gros, car dans un cas comme celui-là on n'a pas le temps de lire tout ce qui s'affiche à l'écran 😊

Il y a peu de paramètres vraiment intéressants à utiliser avec la commande `cat`, car c'est une commande somme toute très basique. On notera quand même le paramètre `-n` qui permet d'afficher les numéros de ligne :

Code : Console

```
mateo21@mateo21-desktop:/var/log$ cat -n syslog
 1      Nov 14 00:44:23 mateo21-desktop syslogd 1.4.1#21ubuntu3: restart.
 2      Nov 14 00:44:23 mateo21-desktop anacron[6725]: Job `cron.daily' termi
 3      Nov 14 00:44:23 mateo21-desktop anacron[6725]: Normal exit (1 job run
 4      Nov 14 00:44:25 mateo21-desktop NetworkManager: <info> eth1: link ti
 5      Nov 14 00:44:51 mateo21-desktop NetworkManager: <info> eth1: link ti
 6      Nov 14 00:45:08 mateo21-
desktop NetworkManager: <debug> [1194997508.332093] nm_device_802_11_wireless_get_a
 7      Nov 14 00:45:08 mateo21-
desktop NetworkManager: <info> User Switch: /org/freedesktop/NetworkManager/Device
 8      Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Deactivating
 9      Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (e
10      Nov 14 00:45:08 mateo21-
desktop NetworkManager: <info> Activation (eth1) cancellation handler scheduled...
11      Nov 14 00:45:08 mateo21-
desktop NetworkManager: <info> Activation (eth1): waiting for device to cancel act
12      Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (e
13      Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (e
14      Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: sending
15      Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: response
16      Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: sending
17      Nov 14 00:45:16 mateo21-desktop NetworkManager: nm_act_request_get_ap
18      Nov 14 00:45:16 mateo21-
desktop NetworkManager: nm_act_request_get_stage: assertion `req != NULL' failed
19      Nov 14 00:45:16 mateo21-desktop NetworkManager: <info> eth1: link ti
20      Nov 14 00:45:21 mateo21-desktop NetworkManager: nm_act_request_get_ap
21      Nov 14 00:45:21 mateo21-desktop NetworkManager: ap_is_auth_required:
...

```

less : afficher le fichier page par page

La commande `cat` est rapide. Trop rapide. Tout le fichier est lu et affiché d'un coup dans la console, ce qui fait qu'on n'a pas le temps de le lire s'il est très gros.

C'est là qu'une autre commande comme `less` devient vraiment indispensable. La grosse différence entre `less` et `cat`, c'est que `less` affiche progressivement le contenu du fichier, page par page. Ça vous laisse le temps de le lire dans la console 😊

Notez qu'il existe aussi une commande très proche : `more`. Pour faire simple, la différence entre `more` et `less` c'est que



more est vieux et possède peu de fonctionnalités, tandis que less est beaucoup plus puissant et rapide. Bref, utilisez less, mais si vous voyez quelqu'un utiliser more un jour ne soyez pas surpris 😊
D'ailleurs, j'avoue qu'il m'arrive encore d'utiliser more aujourd'hui, mais j'essaie de perdre cette mauvaise habitude, il n'y a vraiment que des avantages à utiliser less 😊

Comment ça marche ? Eh bien la commande est très simple : less nomdufichier.

Code : Console

```
mateo21@mateo21-desktop:/var/log$ less syslog
Nov 14 00:44:23 mateo21-desktop syslogd 1.4.1#21ubuntu3: restart.
Nov 14 00:44:23 mateo21-desktop anacron[6725]: Job `cron.daily' terminated
Nov 14 00:44:23 mateo21-desktop anacron[6725]: Normal exit (1 job run)
Nov 14 00:44:25 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:44:51 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:45:08 mateo21-
desktop NetworkManager: <debug> [1194997508.332093] nm_device_802_11_wireless_get_a
Nov 14 00:45:08 mateo21-
desktop NetworkManager: <info> User Switch: /org/freedesktop/NetworkManager/Device
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Deactivating device eth1.
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1): cancelli
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1) cancellat
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1): waiting
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1) cancellat
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1): cabelle
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: sending command 'DISAB
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: response was 'OK'
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> SUP: sending command 'AP_SC
Nov 14 00:45:16 mateo21-desktop NetworkManager: nm_act_request_get_ap: assertion `r
Nov 14 00:45:16 mateo21-desktop NetworkManager: nm_act_request_get_stage: assertion
Nov 14 00:45:16 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:45:21 mateo21-desktop NetworkManager: nm_act_request_get_ap: assertion `r
Nov 14 00:45:21 mateo21-desktop NetworkManager: ap_is_auth_required: assertion `ap
Nov 14 00:45:21 mateo21-
desktop NetworkManager: <info> Activation (eth1/wireless): association took too lo
Nov 14 00:45:21 mateo21-desktop NetworkManager: nm_dbus_get_user_key_for_network: a
syslog
```

Ce qui est intéressant pour nous ici, c'est que la commande less a arrêté la lecture du fichier au bout de quelques lignes (la taille d'un écran de console). Cela vous laisse le temps de lire le début du fichier.

On n'a lu pour le moment que les toutes premières lignes du fichier.



Et comment lire la suite ?

Il y a quelques raccourcis clavier à connaître 😊

Les raccourcis basiques indispensables

Commençons par les quelques raccourcis clavier les plus indispensables, à connaître absolument :

Touche	Effet
Espace	Affiche la suite du fichier. La touche Espace fait défiler le fichier vers le bas d'un "écran" de console. C'est celle que j'utilise le plus souvent. Vous pouvez aussi utiliser la touche "Page Down".

Entrée	Affiche la ligne suivante. Cela permet donc de faire défiler le fichier vers le bas ligne par ligne. Vous pouvez aussi utiliser la touche "Flèche vers le bas"
d	Affiche les 11 lignes suivantes (soit une moitié d'écran). C'est un peu l'intermédiaire entre Espace (tout un écran) et Entrée (une seule ligne).
b	Retourne en arrière d'un écran. Vous pouvez aussi appuyer sur la touche "Page Up".
y	Retourne d'une ligne en arrière. Vous pouvez aussi appuyer sur la touche "Flèche vers le haut".
u	Retourne en arrière d'une moitié d'écran (11 lignes).
q	Arrête la lecture du fichier. Cela met fin à la commande less.



La casse des caractères est importante. Ainsi, si je vous dis qu'il faut appuyer sur la touche "d", ce n'est pas un "D" majuscule (si vous essayez "D" majuscule vous verrez que ça ne marche pas). Sous Linux on fait souvent la différence entre les majuscules et minuscules, souvenez-vous en !

Si on tape Espace, on avance donc d'un écran dans le fichier :

Code : Console

```
Nov 14 00:47:45 mateo21-
desktop init: tty4 main process (4517) killed by TERM signal
Nov 14 00:47:45 mateo21-
desktop init: tty5 main process (4518) killed by TERM signal
Nov 14 00:47:45 mateo21-
desktop init: tty2 main process (4520) killed by TERM signal
Nov 14 00:47:45 mateo21-
desktop init: tty3 main process (4522) killed by TERM signal
Nov 14 00:47:45 mateo21-
desktop init: tty1 main process (4524) killed by TERM signal
Nov 14 00:47:45 mateo21-
desktop init: tty6 main process (4525) killed by TERM signal
Nov 14 00:47:46 mateo21-desktop avahi-
daemon[5390]: Got SIGTERM, quitting.
Nov 14 00:47:48 mateo21-desktop exiting on signal 15
Nov 14 00:48:42 mateo21-desktop syslogd 1.4.1#21ubuntu3: restart.
Nov 14 00:48:42 mateo21-desktop kernel: Inspecting /boot/System.map-
2.6.22-14-generic
Nov 14 00:48:42 mateo21-
desktop kernel: Loaded 25445 symbols from /boot/System.map-2.6.22-14-
generic.
Nov 14 00:48:42 mateo21-
desktop kernel: Symbols match kernel version 2.6.22.
Nov 14 00:48:42 mateo21-desktop kernel: No module symbols loaded -
kernel modules not enabled.
Nov 14 00:48:42 mateo21-
desktop kernel: [ 0.000000] Linux version 2.6.22-14-
generic (buildd@palmer) (gcc version 4.1.3 20070929 (prerelease) (Ubu
ntu 4.1.2-16ubuntu2)) #1 SMP Sun Oct 14 23:05:12 GMT 2007 (Ubuntu 2.6.22-
14.46-generic)
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
provided physical RAM map:
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
e820: 0000000000000000 - 000000000009f800 (usable)
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
e820: 000000000009f800 - 00000000000a0000 (reserved)
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
e820: 00000000000e0000 - 0000000000100000 (reserved)
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
e820: 0000000000100000 - 0000000007f680000 (usable)
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
e820: 0000000007f680000 - 0000000007f68e000 (ACPI data)
```

```
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
e820: 000000007f68e000 - 000000007f700000 (ACPI NVS)
Nov 14 00:48:42 mateo21-desktop kernel: [ 0.000000] BIOS-
e820: 000000007f700000 - 0000000080000000 (reserved)
:
```

Quelques raccourcis plus avancés

Ce ne sont pas des raccourcis que l'on utilise tous les jours, mais ça vaut le coup de savoir qu'ils existent 😊

Touche	Effet
=	Indique où vous en êtes dans le fichier (numéro des lignes affichées et pourcentage).
h	Affiche l'aide (toutes les commandes que je vous apprends ici, je les tire de là 😊). Tapez "q" pour sortir de l'aide.
/	Tapez "/" suivi du texte que vous recherchez pour lancer le mode recherche. Faites Entrée pour valider. Pour ceux qui savent s'en servir, sachez que les expressions régulières sont acceptées (je ne vais pas vous faire un cours sur les expressions régulières ici, ce serait trop long, mais il y en a un dans le cours PHP).
n	Après avoir fait une recherche avec "/", la touche "n" vous permet d'aller à la prochaine occurrence de votre recherche. C'est un peu comme si vous cliquiez sur le bouton "Résultat suivant" 😊
N	Pareil que "n", mais pour revenir en arrière.

Comme vous le voyez, la commande less est très riche. On peut utiliser beaucoup de touches différentes pour se déplacer dans le fichier.

Prenez le temps de vous familiariser avec : c'est un peu perturbant au début, mais lorsque vous aurez appris à vous en servir vous aurez déjà fait un grand pas en avant... et puis ça vous sera très pratique plus tard, croyez-moi 😊

head & tail : afficher le début et la fin d'un fichier

❓ Quoi ? Encore des commandes pour lire un fichier ? 😊

Eh oui 😊

Et figurez-vous que celles-là aussi ça vaut le coup de les connaître. Comme quoi on en fait des commandes rien que pour lire un fichier !

Ces deux commandes sont un peu le contraire l'une de l'autre : la première permet d'afficher le début du fichier, la seconde permet d'afficher la fin.

head : afficher le début du fichier

La commande head ("tête" en anglais) affiche seulement les premières lignes du fichier. Elle ne permet pas de se déplacer dans le fichier comme less, elle permet juste de récupérer les premières lignes.

Code : Console

```
mateo21@mateo21-desktop:/var/log$ head syslog
Nov 14 00:44:23 mateo21-desktop syslogd 1.4.1#21ubuntu3: restart.
Nov 14 00:44:23 mateo21-desktop anacron[6725]: Job `cron.daily' terminated
Nov 14 00:44:23 mateo21-desktop anacron[6725]: Normal exit (1 job run)
Nov 14 00:44:25 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:44:51 mateo21-desktop NetworkManager: <info> eth1: link timed out.
Nov 14 00:45:08 mateo21-
desktop NetworkManager: <debug> [1194997508.332093] nm_device_802_11_wireless_get_a
```

```
Nov 14 00:45:08 mateo21-
desktop NetworkManager: <info> User Switch: /org/freedesktop/NetworkManager/Device
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Deactivating device eth1.
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1): cancelli
Nov 14 00:45:08 mateo21-desktop NetworkManager: <info> Activation (eth1) cancellat
```

Si vous avez juste besoin de récupérer les premières lignes d'un fichier, head est donc la commande qu'il vous faut. Simple, net, efficace 😊

Comment ? Vous voulez des paramètres ?

Je n'en ai pas beaucoup à vous offrir, mais celui-là au moins est à connaître : -n, suivi d'un nombre. Il permet d'afficher le nombre de lignes que vous voulez. Par exemple, si vous ne voulez que les 3 premières lignes, tapez :

Code : Console

```
mateo21@mateo21-desktop:/var/log$ head -n 3 syslog
Nov 14 00:44:23 mateo21-desktop syslogd 1.4.1#21ubuntu3: restart.
Nov 14 00:44:23 mateo21-
desktop anacron[6725]: Job `cron.daily' terminated
Nov 14 00:44:23 mateo21-desktop anacron[6725]: Normal exit (1 job run)
```

Et voilà le travail ! 😊

tail : afficher la fin du fichier

Très intéressante aussi (voire même plus), la commande tail vous renvoie la fin du fichier, donc les dernières lignes.

Code : Console

```
mateo21@mateo21-desktop:/var/log$ tail syslog
Nov 14 22:42:10 mateo21-desktop NetworkManager: <info> Clearing nscd hosts cache.
Nov 14 22:42:10 mateo21-desktop NetworkManager: <WARN> nm_spawn_process(): nm_spaw
i hosts'): could not spawn process. (Failed to execute child process "/usr/sbin/nsc
Nov 14 22:42:10 mateo21-desktop NetworkManager: <info> Activation (eth1) Finish ha
Nov 14 22:42:10 mateo21-
desktop NetworkManager: <info> Activation (eth1) Stage 5 of 5 (IP Configure Commit
Nov 14 22:42:10 mateo21-desktop NetworkManager: <info> Activation (eth1) successfu
Nov 14 22:41:57 mateo21-desktop ntpdate[8422]: step time server 91.189.94.4 offset
Nov 14 22:41:59 mateo21-desktop avahi-
daemon[5385]: Registering new address record for fe80::219:d2ff:fe61:900a on eth1.*
Nov 14 22:42:08 mateo21-desktop kernel: [ 7870.160000] eth1: no IPv6 routers presen
Nov 14 23:11:26 mateo21-desktop -- MARK --
Nov 14 23:17:01 mateo21-desktop /USR/SBIN/CRON[8515]: (root) CMD ( cd / && run-pa
```

On peut là encore utiliser -n suivi d'un nombre pour afficher les X dernières lignes :

Code : Console

```
mateo21@mateo21-desktop:/var/log$ tail -n 3 syslog
Nov 14 22:42:08 mateo21-
desktop kernel: [ 7870.160000] eth1: no IPv6 routers present
Nov 14 23:11:26 mateo21-desktop -- MARK --
Nov 14 23:17:01 mateo21-
desktop /USR/SBIN/CRON[8515]: (root) CMD ( cd / && run-parts --
```

```
report /etc/cron.hourly)
```

Mais ce n'est pas tout ! Il y a un autre paramètre à côté duquel vous ne pouvez pas passer : -f (f pour "follow", "suivre" en anglais).

Ce paramètre magique ordonne à tail de "suivre" la fin du fichier au fur et à mesure de son évolution.

C'est extrêmement utile pour suivre un fichier de log qui évolue souvent. Vous pouvez tester sur syslog par exemple :

Code : Console

```
mateo21@mateo21-desktop:/var/log$ tail -f syslog
Nov 14 23:11:26 mateo21-desktop -- MARK --
Nov 14 23:17:01 mateo21-desktop /USR/SBIN/CRON[8515]: (root) CMD ( cd / && run-pa
report /etc/cron.hourly)
Nov 14 23:27:52 mateo21-
desktop kernel: [10614.344000] ata2.00: exception Emask 0x0 SAct 0x0 SErr 0x0 actio
Nov 14 23:27:52 mateo21-
desktop kernel: [10614.344000] ata2.00: cmd a0/00:00:00:00:20/00:00:00:00:00/a0 tag
Nov 14 23:27:52 mateo21-
desktop kernel: [10614.344000]          res 40/00:03:00:00:00/00:00:00:00:00/a0 Ema
Nov 14 23:27:57 mateo21-
desktop kernel: [10619.388000] ata2: port is slow to respond, please be patient (St
Nov 14 23:28:02 mateo21-desktop kernel: [10624.392000] ata2: device not ready (errn
16), forcing hardreset
Nov 14 23:28:02 mateo21-desktop kernel: [10624.392000] ata2: soft resetting port
Nov 14 23:28:02 mateo21-desktop kernel: [10624.928000] ata2.00: configured for UDMA
Nov 14 23:28:02 mateo21-desktop kernel: [10624.928000] ata2: EH complete
```

Le problème de syslog c'est qu'il n'évolue pas forcément toutes les secondes. Mais si vous êtes patient et que vous regardez votre console, vous devriez le voir écrire de nouvelles lignes sous vos yeux au bout d'un moment 😊

Faites Ctrl + C (Ctrl et C en même temps) pour arrêter la commande tail.



A connaître : la combinaison de touche Ctrl + C est utilisable dans la plupart des programmes console pour demander leur arrêt. C'est un peu l'équivalent du Alt + F4 de Windows.

Pour tout vous dire, tail -f est une de mes commandes préférées sous Linux. En effet, elle a un petit côté Matrix qui impressionne pas mal les gens, surtout si vous le faites sur un fichier de log qui évolue très vite. Et à part pour impressionner les gens, c'est aussi un bon moyen pour vous de surveiller ce qui se passe en temps réel sur un ordinateur (si vous êtes assez rapide pour suivre 😊)

Par exemple, les logs Apache du Site du Zéro permettent de voir en temps réel qui se connecte sur le site, avec quelle IP, quel fichier a été chargé, à quelle heure etc.

Aux heures d'affluence du site, ce fichier évolue tellement vite qu'il est pratiquement impossible de le suivre pour un humain 😊

Je vous ai fait une petite vidéo pour que vous vous rendiez compte de la chose :

Voir la vidéo d'un tail -f en action

Notez que par défaut, tail -f recherche les nouveaux changements dans le fichier toutes les secondes. Si vous voulez, vous pouvez rajouter le paramètre -s suivi d'un nombre. Par exemple, tail -f -s 3 syslog recherchera les changements toutes les 3 secondes (au lieu de toutes les secondes). Les nombres décimaux sont acceptés, à condition d'utiliser le point "." à la place de la virgule.

touch & mkdir : créer des fichiers et dossiers

Assez lu de fichiers, maintenant voyons voir comment on les crée ! 😊

Nous allons d'abord voir comment créer un fichier, puis comment créer un dossier, car ce n'est pas la même commande...

touch : créer un nouveau fichier

En fait, il n'existe aucune commande spécialement faite pour créer un nouveau fichier vide sous Linux (ce n'est pas très utile). En général, on se contente d'ouvrir un éditeur de texte et d'enregistrer, ce qui provoque la création d'un nouveau fichier comme sous Windows.

La commande touch est à la base faite pour modifier la date de dernière modification d'un fichier. D'où son nom : on "touche" le fichier pour faire croire à l'ordinateur qu'on vient de le modifier alors qu'on n'a rien changé. Ca peut se révéler utile dans certains cas précis qu'on ne verra pas ici.

L'intérêt de touch pour nous dans ce chapitre, c'est que si le fichier n'existe pas, il sera créé ! On peut donc *aussi* utiliser touch pour créer de nouveaux fichiers, même s'il n'a pas vraiment été fait pour ça à la base 🤖

La commande attend un paramètre : le nom du fichier à créer.

Commencez par vous rendre dans votre dossier personnel, ce n'est pas une bonne idée de mettre le bazar dans /var/log, le dossier personnel est là pour ça 😊

Si vous vous souvenez bien, il suffit de taper cd :

Code : Console

```
mateo21@mateo21-desktop:/var/log$ cd
mateo21@mateo21-desktop:~$
```

Pour le moment, mon dossier personnel ne contient que des sous-dossiers :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -F
Desktop/   Examples@  Images/    Modèles/   Public/    Vidéos/
Documents/ images/     log/       Musique/    tutos/
```

Créons un fichier appelé "fichierbidon" :

Code : Console

```
mateo21@mateo21-desktop:~$ touch fichierbidon
mateo21@mateo21-desktop:~$ ls -F
Desktop/   Examples@  images/    log/        Musique/    tutos/
Documents/ fichierbidon Images/    Modèles/    Public/     Vidéos/
```

Comme le montre la commande ls -F que j'ai tapée ensuite, un fichier appelé "fichierbidon" (sans extension) a été créé. Bien entendu, vous pouvez créer un fichier de l'extension que vous voulez :

Code : Console

```
mateo21@mateo21-desktop:~$ touch autrefichierbidon.txt
mateo21@mateo21-desktop:~$ ls -F
autrefichierbidon.txt Examples@  Images/    Musique/    Vidéos/
Desktop/        fichierbidon log/        Public/
Documents/      images/     Modèles/    tutos/
```

Autre information intéressante : vous pouvez créer plusieurs fichiers en une seule commande. Il vous suffit de les lister l'un après l'autre, séparés par des espaces.

Ainsi, on aurait pu créer nos 2 fichiers comme ceci :

Code : Console

```
touch fichierbidon autrefichierbidon.txt
```



Et si je veux que mon fichier contienne un espace, je fais comment ?

Entourez-le de guillemets !

Code : Console

```
touch "Fichier bidon"
```

mkdir : créer un nouveau dossier

La commande mkdir, elle, est faite pour créer un nouveau dossier. Elle fonctionne de la même manière que touch.

Code : Console

```
mkdir mondossier
```

On peut créer 2 dossiers (ou plus !) en même temps en les séparant là aussi par des espaces :

Code : Console

```
mkdir mondossier autredossier
```

Si vous faites un ls, vous verrez que les dossiers ont bien été créés 😊

Il y a un paramètre utile avec mkdir : -p. Il sert à créer tous les dossiers intermédiaires. Par exemple :

Code : Console

```
mkdir -p animaux/vertebres/chat
```

... créera le dossier animaux, puis à l'intérieur le sous-dossier vertebres, puis à l'intérieur encore le sous-dossier chat !

cp & mv : copier et déplacer un fichier

Parmi les opérations de base que l'on veut pouvoir faire avec les fichiers, il y a la copie et le déplacement de fichier. C'est un peu le genre de truc que l'on fait tous les jours, donc il est important de savoir s'en servir 😊

cp : copier un fichier

La commande cp (abréviation de "CoPy", "copier" en anglais) vous permet comme son nom l'indique de copier un fichier... mais aussi de copier plusieurs fichiers à la fois, et même de copier des dossiers !

Si on essayait de copier le fichier `fichierbidon` qu'on a créé tout à l'heure ?
Ça fonctionne comme ceci :

Code : Console

```
cp fichierbidon fichiercopie
```

Le premier paramètre est le nom du fichier à copier, le second est le nom de la copie du fichier à créer.
En faisant cela, on aura donc 2 fichiers identiques dans le même répertoire : `fichierbidon` et `fichiercopie`.



N'oubliez pas d'utiliser l'autocomplétion avec la touche Tab ! Lorsque vous avez écrit "`cp fic`", tapez Tab, et `fichierbidon` devrait se compléter tout seul !

Copier dans un autre dossier

On n'est pas obligé de copier le fichier dans le même dossier bien sûr. On peut très bien utiliser le système de répertoires relatifs et absolus qu'on a vu dans le chapitre précédent.

Par exemple, si je veux copier `fichierbidon` dans le sous-dossier "`mondossier`" que j'ai créé tout à l'heure :

Code : Console

```
cp fichierbidon mondossier/
```

Le fichier `fichierbidon` sera copié dans `mondossier` sous le même nom.



Notez que mettre le `/` à la fin n'est pas obligatoire. Si vous le voyez là, c'est parce que l'autocomplétion me l'a automatiquement ajouté lorsque j'ai appuyé sur Tab. Ehhh oui, je suis tellement flemmard que je n'écris même pas "`mondossier`" en entier, j'écris juste "`mon`" suivi de Tab, et hop là c'est écrit en entier ! Ça va beaucoup plus vite lorsqu'on prend ce réflexe 😊

Si vous voulez copier `fichierbidon` dans `mondossier` sous un autre nom, faites comme ceci :

Code : Console

```
cp fichierbidon mondossier/fichiercopie
```

... avec cette commande, on aura créé une copie de `fichierbidon` dans "`mondossier`" sous le nom "`fichiercopie`" !

Enfin, là j'utilise des répertoires relatifs, mais je peux aussi écrire un répertoire en absolu :

Code : Console

```
cp fichierbidon /var/log/
```

... copiera fichierbidon dans le dossier /var/log.

Copier des dossiers

Avec l'option -R (un R majuscule !), vous pouvez copier un dossier, ainsi que tous les sous-dossiers et fichiers qu'il contient !

Tout à l'heure, on a créé un dossier "animaux" qui contenait un autre dossier "vertebres", qui lui-même contenait le dossier "chat". Si vous tapez cette commande :

Code : Console

```
cp -R animaux autresanimaux
```

... cela aura pour effet de copier animaux et tous ses sous-dossiers sous le nom "autresanimaux".

Faites des ls après pour vérifier que les sous-dossiers sont bien là et que je ne vous mène pas en bateau ! 😊

Utiliser le joker *

Le symbole * est appelé joker, ou encore *wildcard* en anglais sous linux.

Il vous permet de copier par exemple tous les fichiers image .jpg dans un sous-dossier :

Code : Console

```
cp *.jpg mondossier/
```

Vous pouvez aussi vous en servir pour copier tous les fichiers dont le nom commence par "so" :

Code : Console

```
cp so* mondossier/
```

Le joker est un atout très puissant, n'hésitez pas à l'utiliser !

C'est avec des outils comme le joker que la console devient pour vous progressivement plus puissante que l'explorateur de fichiers que vous manipulez à la souris.

mv : déplacer un fichier

Très proche de cp, la commande mv (MoVe, "déplacer" en anglais) a en fait 2 utilités :

- Déplacer un fichier (ou un dossier)
- Renommer un fichier (ou un dossier)

Vous allez comprendre pourquoi 😊

Déplacer un fichier

La commande mv s'utilise pratiquement comme cp :

Code : Console

```
mv fichierbidon mondossier/
```

Au lieu de copier fichierbidon dans mondossier comme on l'a fait tout à l'heure, ici on a juste déplacé le fichier. Il n'existe plus dans son dossier d'origine.

Vous pouvez déplacer des dossiers aussi simplement :

Code : Console

```
mv animaux/ mondossier/
```

... déplacera le dossier animaux(et tous ses sous-dossiers) dans mondossier.

Vous pouvez aussi utiliser les jokers :

Code : Console

```
mv *.jpg mondossier/
```

Renommer un fichier

La commande mv permet de faire quelque chose d'assez étonnant : renommer un fichier. En effet, il n'existe pas de commande spéciale pour renommer un fichier en console sous linux, c'est la commande mv qui est utilisée pour ça.

Par exemple :

Code : Console

```
mv fichierbidon superfichier
```

... renommra "fichierbidon" en "superfichier". Après cette commande, fichierbidon n'existe plus, il a été renommé.

Déplacer et renommer un fichier à la fois

Vous pouvez aussi déplacer fichierbidon dans "mondossier" tout en lui affectant un nouveau nom :

Code : Console

```
mv fichierbidon mondossier/superfichier
```

Et voilà le travail !

Je vous conseille fortement de vous entraîner à utiliser cp et mv dans tous les sens : avec ou sans joker, en déplaçant, renommant des dossiers, en déplaçant/renommant à la fois, en utilisant des chemins relatifs et absolus, etc etc. C'est assez intuitif normalement, mais il faut pratiquer et pas juste se contenter de lire ce que j'écris pour que ça rentre.



N'oubliez pas d'utiliser l'autocomplétion de fichiers et dossiers avec la touche Tab, si vous ne le faites pas dès maintenant vous perdrez du temps et vous trouverez la console nulle alors que vous devriez la trouver géniale 😊
Autre chose : le symbole ".." signifie "dossier précédent", et "." signifie "dossier dans lequel je me trouve". Vous pourriez en avoir besoin lorsque vous copiez ou déplacez un fichier.

Si vous avez la tête qui tourne à force de copier et déplacer des fichiers dans des dossiers, c'est normal. Ça commence à devenir un beau bordel dans vos dossiers d'ailleurs, non ? 🤔

Il est temps de faire un peu de ménage avec la commande permettant de SUPPRIMER : rm ! 🧑🏻🔪

rm : supprimer des fichiers et dossiers

On attaque la commande qui fâche : rm.

Pourquoi est-ce qu'elle fâche ? Parce qu'il n'existe pas de corbeille dans la console de linux : le fichier est directement supprimé sans possibilité de récupération !

rm : supprimer un fichier

La commande rm (pour "ReMove", "supprimer" en anglais) peut supprimer un fichier, 2 fichiers, plusieurs fichiers, des dossiers, voire même votre ordinateur entier si vous le voulez 🤖

Il faut l'utiliser avec précaution donc. Commençons par des choses simples, supprimons ce fichierbidon :

Code : Console

```
rm fichierbidon
```

Normalement, on ne vous demande pas de confirmation, on ne vous affiche rien. Le fichier est supprimé sans autre forme d'avertissement. Brutal, hein ? 🧑🏻🔪

Vous pouvez aussi supprimer plusieurs fichiers en séparant leurs noms par des espaces :

Code : Console

```
rm fichierbidon fichiercopie
```

-i : demander confirmation

La commande -i permet de vous demander une confirmation pour chacun des fichiers :

Code : Console

```
mateo21@mateo21-desktop:~$ rm -i fichierbidon  
rm: détruire fichier régulier vide `fichierbidon'?
```

Lorsqu'on vous demande une confirmation de type oui/non comme ici, vous devez répondre par une lettre :

- o : signifie Oui. Sur certains systèmes anglais, il faudra peut-être utiliser "y" de Yes.

- n : signifie Non.

Tapez ensuite sur Entrée pour valider.

-f : forcer la suppression, quoiqu'il arrive

-f, c'est un peu le contraire de -i : c'est le mode des gros bourrins 🤪

Ce paramètre force la suppression, ne demande pas confirmation, même s'il y a un problème potentiel.

En raison des risques que cela comporte, utilisez-le aussi rarement que possible.

Code : Console

```
rm -f fichierbidon
```

-v : dis-moi ce que tu fais, petit cachotier

Le paramètre -v ("Verbose", verbeux en anglais, c'est-à-dire "parler beaucoup") est un paramètre que l'on retrouve dans beaucoup beaucoup de commandes sous Linux. Il permet de demander à la commande de dire ce qu'elle est en train de faire.

Comme vous l'avez vu, par défaut la commande rm est silencieuse. Si vous supprimez de très nombreux fichiers, ça peut prendre du temps. Pour éviter que vous vous impatientiez, pensez à utiliser -v :

Code : Console

```
mateo21@mateo21-desktop:~$ rm -v fichierbidon fichiercopie
détruit `fichierbidon`
détruit `fichiercopie`
```

Vous voyez au fur et à mesure de l'avancement ce qui est en train d'être fait. Très pratique !

-r : supprimer un dossier et son contenu

Le paramètre -r peut être utilisé pour supprimer un dossier (au lieu d'un fichier) ainsi que tout ce qu'il contient : fichiers et dossiers !

C'est un paramètre assez dangereux, donc faites bien attention de l'utiliser sur un dossier dont vous ne voulez vraiment plus, car tout va disparaître à l'intérieur :

Code : Console

```
rm -r animaux/
```

... supprime le dossier "animaux" ainsi que tout ce qu'il contenait (sous-dossiers "vertebres", et "chat").



Notez qu'il existe aussi la commande rmdir. La grosse différence avec rm -r, c'est que rmdir ne peut supprimer un dossier que s'il est vide ! Il faudra avoir fait le ménage dedans auparavant.

rm et le joker de la Mort (qui tue)

Bon, vous êtes grands, je crois que le moment est venu de vous révéler un terrible secret : ~~les enfants ne naissent pas dans les choux~~.

Euh pardon, je voulais dire : la commande `rm` est vraiment dangereuse. Très dangereuse. Vous pouvez potentiellement bousiller tout votre système avec ! 😬



Je vais vous montrer quelque chose d'horrible, d'affreux, d'interdit aux moins de 18 ans, bref vous m'avez compris, le truc à ne faire sous aucun prétexte, même pas en cauchemar.

Code : Console

```
NON NON NON NE FAITES JAMAIS CA !!! => rm -rf /*
```

Je me suis permis de mettre du texte avant pour vous éviter la tentation de recopier bêtement la commande pour "rigoler", pour "voir ce que ça fait". Je vais vous l'expliquer dans le détail, parce que c'est quand même l'erreur n°1 à ne pas faire sous Linux.

- `rm` : commande la suppression
- `-r` : suppression récursive de tous les fichiers et dossiers
- `-f` : force la suppression sans demander la moindre confirmation
- `/*` : supprime tous les fichiers et dossiers qui se trouvent à la racine (/) quel que soit leur nom (joker *).

En clair, cette commande supprime tout votre disque dur depuis la racine, sous-dossiers compris, et ne demande aucune confirmation. Aucune possibilité de récupération, votre PC est foutu. Vous êtes bon pour une réinstallation de Linux, et aussi de Windows si la partition de Windows était accessible depuis Linux.



Mais ils sont cons les gens qui ont créé cette commande ! Pourquoi autoriser de faire une chose aussi risquée ? 😬

En fait, il y a plusieurs mécanismes de protection. On en apprendra plus dans le prochain chapitre (qui traitera des utilisateurs et de leurs droits).

Par exemple, les fichiers à la racine ne vous "appartiennent" pas, ils appartiennent au super-utilisateur "root". Moi je me suis loggé en tant que mateo21, je n'ai donc théoriquement pas le droit de supprimer ces fichiers. La suppression sera refusée.

Seulement, pour peu que vous soyez loggé en tant que root (on verra comment le faire dans le chapitre suivant), vous aurez le droit de le faire, et là plus rien ne vous arrêtera !

On apprendra plus tard comment utiliser les alias de commande pour éviter qu'une commande aussi dangereuse ne s'exécute. En attendant, ne jouez pas avec le feu, car vous y perdriez les mains, les pieds, la tête et tout ce qui va avec 😬

Le joker reste quand même très utile, mais lorsque vous l'utilisez avec `rm`, triplez d'attention.

Par exemple :

Code : Console

```
rm -rf *
```

... supprime tous les fichiers et sous-dossiers du dossier dans lequel je me trouve. Il m'arrive de l'utiliser, d'en avoir besoin, mais à chaque fois je fais très très attention à ce qu'il n'y ait plus rien dans ce dossier (et dans les sous-dossiers) qui m'intéresse. Comme vous pouvez le voir, il n'y a qu'un seul caractère de différence (le /) avec la commande de la Mort que je vous ai montrée un peu plus haut.

Une erreur est vite arrivée. J'ignore combien de gens se sont pendus après avoir exécuté cette commande, mais ça méritait au moins un **GROS** avertissement !

In : créer des liens entre fichiers

Bien qu'un peu moins courante, la commande `ln` vous sera certainement utile un jour ou l'autre. Elle permet de créer des liens entre des fichiers, c'est-à-dire (pour employer des mots que vous connaissez) qu'elle permet de **créer des raccourcis**.

Ces "raccourcis", qu'on appelle des liens sous Linux, sont un peu plus complexes que ceux que vous avez l'habitude de voir sous Windows. En effet, on peut créer 2 types de liens :

- Des liens physiques
- Des liens symboliques

Ces 2 types ne fonctionnent pas de la même manière. Pour comprendre ce qui les différencie, il faut savoir comment un OS tel que Linux gère les fichiers sur le disque dur.

Allons allons ne faites pas cette tête-là, un peu de théorie sur le fonctionnement des OS, c'est toujours très intéressant ! 😊

Le stockage des fichiers

Sur le disque dur, chaque fichier est grosso-modo séparé en 2 parties :

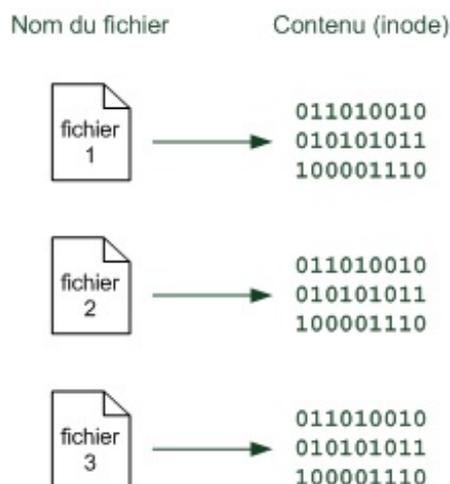
- Son nom
- Son contenu

Vous avez bien entendu : la liste des noms de fichiers est stockée à un autre endroit que leur contenu. Cette séparation aide Linux à s'organiser.



Je simplifie ici volontairement les choses. En pratique, c'est (toujours) un peu plus compliqué. Il y en fait 3 parties : le nom, les informations de gestion (droits d'accès) et le contenu. Mais nous allons faire simple car notre but est juste de comprendre l'idée générale du fonctionnement. 😊

Chaque contenu de fichier se voit attribuer un numéro d'identification appelé **inode**. Chaque nom de fichier est donc associé à un inode (son contenu).



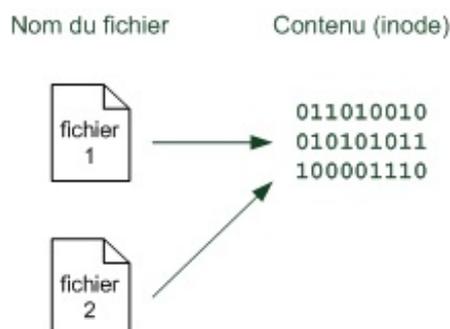
C'est tout ce que vous avez besoin de savoir pour comprendre la suite. 😊

Nous allons maintenant découvrir comment créer des liens physiques puis des liens symboliques.

Créer des liens physiques

Ce type de lien est plus rarement utilisé que le lien symbolique, mais il faut tout de même le connaître car il peut se révéler pratique.

Un lien physique permet d'avoir 2 noms de fichiers qui partagent exactement le même contenu, c'est-à-dire le même inode :



Ainsi, que vous passiez par fichier1 ou par fichier2, vous modifiez exactement le même contenu. En quelque sorte, le fichier est le même. On peut juste y accéder via 2 noms de fichiers différents.



On ne peut pas créer de liens physiques sur des répertoires. Cela ne fonctionne qu'avec les fichiers.
Il existe des options pour que ça fonctionne avec des répertoires mais c'est un peu particulier et on n'en parlera pas.
Pour faire un "raccourci" vers un répertoire, on préférera utiliser un lien symbolique.

Pour créer un lien physique, nous allons utiliser la commande `ln`. Je vous propose tout d'abord de créer un répertoire pour nos tests :

Code : Console

```
mkdir tests  
cd tests
```

Une fois dans ce dossier, créez un fichier avec la commande `touch` par exemple :

Code : Console

```
touch fichier1
```

Nous voulons maintenant créer un lien physique : nous allons créer un "fichier2" qui partagera le même inode (le même contenu) que "fichier1". Tapez :

Code : Console

```
ln fichier1 fichier2
```

Si vous listez les fichiers du répertoire, vous avez l'impression d'avoir 2 fichiers différents :

Code : Console

```
mateo21@mateo21-desktop:~/tests$ ls -l
total 0
-rw-r--r-- 2 mateo21 mateo21 0 2008-07-31 13:55 fichier1
-rw-r--r-- 2 mateo21 mateo21 0 2008-07-31 13:55 fichier2
```

A priori, rien ne nous permet ici de deviner que ces fichiers modifient le même contenu. Le lien physique est donc un lien dur, pas évident à détecter au premier coup d'oeil.



La seconde colonne de la liste (qui indique "2" pour chacun des fichiers) correspond au nombre de fichiers qui partagent le même inode. C'est le seul indice qui vous permet de savoir que quelqu'un a fait un lien physique, mais vous ne pouvez pas savoir lequel. Le seul moyen de vérifier que ces fichiers partagent le même contenu, c'est de faire `ls -li` pour afficher les numéros d'inode correspondants et de vérifier que ces 2 fichiers sont associés au même inode. En temps normal, sur la plupart des fichiers la seconde colonne indique donc "1". Si c'est un dossier, ce nombre indique en revanche le nombre de fichiers à l'intérieur.

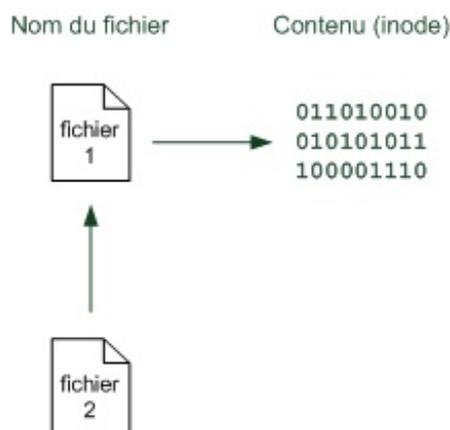
Si vous supprimez un des deux fichiers, l'autre fichier reste en place et le contenu sera toujours présent sur le disque. L'inode est supprimé uniquement quand plus un seul nom de fichier ne pointe dessus.

En clair, supprimez "fichier1" pour voir. Vous verrez que "fichier2" existe toujours et qu'il affiche toujours le même contenu. Il faut supprimer fichier1 ET fichier2 pour supprimer le contenu.

Créer des liens symboliques

Les liens symboliques ressemblent plus aux "raccourcis" dont vous avez peut-être l'habitude sous Windows. La plupart du temps, on crée des liens symboliques sous Linux pour faire un raccourci, et non des liens physiques qui sont un peu particuliers.

Le principe du lien symbolique est que l'on crée un lien vers un autre nom de fichier. Cette fois, on pointe vers le nom de fichier et non vers l'inode directement :



Supprimez le "fichier2" que nous avons créé tout à l'heure (sous forme de lien physique) :

Code : Console

```
rm fichier2
```

Créons maintenant un nouveau fichier2, cette fois sous forme de lien symbolique. On utilise là encore la commande `ln`, mais cette fois avec le paramètre `-s` (s comme symbolique) :

Code : Console

```
ln -s fichier1 fichier2
```

Et voilà. 😊

Cette fois, la commande détaillée `ls -l` sera beaucoup plus précise :

Code : Console

```
mateo21@mateo21-desktop:~/tests$ ls -l
total 0
-rw-r--r-- 1 mateo21 mateo21 0 2008-07-31 13:55 fichier1
lrwxrwxrwx 1 mateo21 mateo21 8 2008-07-31 14:15 fichier2 -> fichier1
```

On note 2 choses :

- La toute première lettre de la seconde ligne est un "l" (comme link, c'est-à-dire lien).
- Tout à la fin de la seconde ligne, une flèche montre clairement que `fichier2` pointe vers `fichier1`.

Bref, les liens symboliques sont beaucoup plus faciles à repérer que les liens physiques !



Ok, mais quelles différences à part ça sinon ? Le résultat revient au même non ? Qu'on ouvre `fichier1` ou `fichier2`, on éditera le même contenu au final !

Tout à fait. Il y a quand même quelques subtilités :

- Par exemple, si vous supprimez `fichier2`, il ne se passe rien de mal. Par contre, si vous supprimez `fichier1`, `fichier2` pointera vers un fichier qui n'existe plus. Le lien symbolique sera cassé et ne servira donc plus à rien. On parle de "lien mort".
- D'autre part, l'avantage des liens symboliques est qu'ils fonctionnent aussi sur des répertoires, contrairement aux liens physiques.

Je vous conseille d'avaler un peu de citrate de Bétaïne pour digérer tout ça 😊

Non, sans rire, il y avait beaucoup à retenir dans ce chapitre sur la manipulation de fichiers. Le QCM ne peut pas vous questionner sur tout, mais sa taille devrait déjà vous faire comprendre que vous avez vu beaucoup de choses.

Tout ça, ce sont des commandes vraiment basiques que vous utiliserez régulièrement par la suite. Autant vous dire qu'il vaut mieux les digérer correctement.

Prochaine étape : le fonctionnement des utilisateurs, des groupes et de leurs droits sous Linux. On est encore dans le B.A.-BA, mais que voulez-vous, c'est le début. Il fallait bien commencer quelque part, et je vous avais prévenu que le monde de la console était riche et que vous aviez beaucoup à apprendre. Je ne vous ai pas menti vous voyez 😊

Les utilisateurs et les droits

La grande puissance de Linux, c'est d'être :

- Multitâche
- Multi-utilisateurs

Multitâche signifie qu'on peut démarrer plusieurs programmes à la fois. Ça on a déjà l'habitude de le faire avec Windows.

Multi-utilisateurs signifie que plusieurs personnes peuvent travailler sur le même OS en même temps. Ça normalement c'est nouveau pour vous. En effet, lorsque vous utilisez l'ordinateur, vous avez l'habitude d'être "seul" dessus. Sous Linux, on peut très bien être 15 personnes à utiliser la même machine en même temps.



Mais... il n'y a qu'un écran, un clavier, une souris, comment on peut être à 15 dessus en même temps ???

En se connectant à la machine par le réseau via SSH, comme je vous l'ai montré un peu plus tôt.

Tous ces gens qui utilisent la machine, parfois en même temps, ça a demandé à Linux d'être bien organisé à ce niveau dès le départ. Ainsi, chaque personne a **son** propre compte utilisateur, et il existe un ensemble de règles qui disent qui a le droit de faire quoi.

Je vous propose de découvrir tous ces mécanismes dans ce chapitre 😊

sudo : exécuter une commande en root

Lorsque vous avez installé Ubuntu, on vous a demandé le nom du compte utilisateur que vous vouliez créer. Moi j'ai créé l'utilisateur "mateo21" par exemple.

Dans la plupart des distributions Linux, on vous proposera de créer un compte utilisateur avec des **droits limités**, comme c'est le cas pour mon compte "mateo21".



Attends, c'est moi qui ai installé Linux mais j'ai pas le droit de faire tout ce que je veux dessus ?

Oui, et c'est une sécurité. Bien sûr, comme c'est vous l'patron, vous pouvez à tout moment dire "Bon allez je passe en mode chef-qui-peut-tout-faire". Mais c'est une sécurité de ne pas avoir le droit de tout faire par défaut, car certaines commandes peuvent être dangereuses pour la stabilité et la sécurité de votre ordinateur. Avoir des droits limités, ça veut dire aussi par exemple qu'on s'empêche d'exécuter la commande de la Mort qui tue qu'on a vue dans le chapitre précédent (`rm -rf /*`).

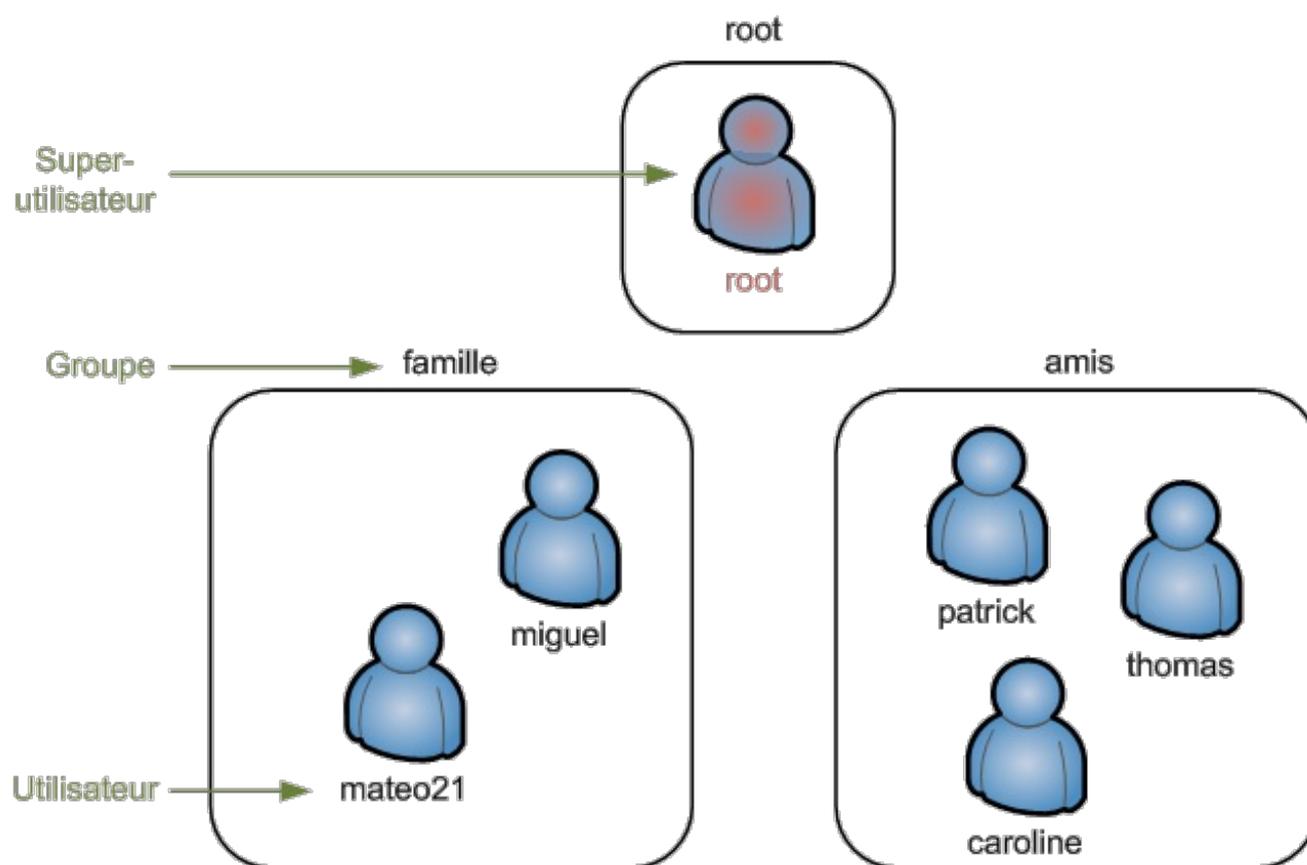
On va d'abord commencer par voir comment sont organisés les utilisateurs sous Linux, puis comment devenir le "chef". Plus loin dans le chapitre, nous apprendrons à créer et supprimer des utilisateurs en ligne de commande.

L'organisation des utilisateurs sous Linux

On peut créer autant d'utilisateurs que l'on veut, eux-mêmes répartis dans des groupes.

Il y a un utilisateur "spécial" appelé root, aussi appelé super-utilisateur. Il a tous les droits sur la machine.

Voyons ce que ça donnerait dans un schéma :



Au départ chez moi, 2 utilisateurs sont créés : root et mateo21.

On ne se connecte en root que très rarement, lorsque c'est nécessaire. Certaines commandes de Linux qu'on va voir dans ce chapitre ne sont accessibles qu'à root.

Tout le reste du temps, on utilise le compte "limité" qu'on a créé (mateo21 dans mon cas).

Cette simple protection permet de largement limiter les dégâts en cas de fausse manipulation, de virus sur votre PC, etc. En effet, un virus ne peut rien faire de plus que vous quand vous êtes connecté avec des droits limités. En revanche, si vous êtes en root il pourra tout faire, même détruire votre ordinateur.

Sous Windows, vous êtes toujours connecté en administrateur par défaut (équivalent de root) ce qui explique pourquoi les virus y sont si dangereux.



Exception : Ubuntu est une des rares distributions à ne pas autoriser de se connecter (logger) en root. Le compte root existe mais vous n'y avez pas accès directement. Ce n'est pas un problème comme on va le voir, car on peut y accéder indirectement.

Les développeurs d'Ubuntu justifient ce choix car ils considèrent (à juste titre) qu'il est dangereux de laisser le compte root entre les mains d'un débutant. Moi-même sur d'autres distributions j'ai tendance à désactiver l'accès direct à l'utilisateur root.

sudo : devenir root un instant

Par défaut, vous êtes connecté sous votre compte limité (mateo21 pour ma part).

Il est impossible sous Ubuntu de se connecter directement en root au démarrage de l'ordinateur. Comment faire alors pour exécuter des commandes que seul root a le droit d'exécuter ?

On peut devenir root *temporairement* à l'aide de la commande `sudo`.

Cette commande signifie "Faire en se substituant à l'utilisateur" : Substitute User **DO**.

Ecrivez donc `sudo` suivi de la commande que vous voulez exécuter. Ce qui donne :

Code : Console

```
sudo commande
```

On vous demandera normalement votre mot de passe (au moins la première fois) pour exécuter la commande. Ce mot de passe est le même que celui de votre compte utilisateur limité.

Par exemple, vous pouvez exécuter un simple `ls` avec les droits root (vous ne risquez rien rassurez-vous) :

Code : Console

```
mateo21@mateo21-desktop:/home$ sudo ls
[sudo] password for mateo21:
autres dossiers Desktop  Exemples  Images  Modèles  Musique  tutos
autres animaux Documents images    log      mondossier Public  Vidéos
```

Comme vous le voyez, on vous demande d'abord le mot de passe par sécurité.

Faire un `ls` en tant que root n'apporte rien de bien spécial, mais c'était pour avoir un exemple "sûr" dans lequel vous ne risquez pas de bousiller votre ordinateur 😊

sudo su : devenir root et le rester

Si vous tapez `sudo su` (tout court), vous passerez root indéfiniment.

Code : Console

```
mateo21@mateo21-desktop:/home$ sudo su
[sudo] password for mateo21:
root@mateo21-desktop:/home#
```

Le symbole `#` à la fin de l'invite de commandes vous indique que vous êtes devenu super-utilisateur.

Vous pouvez alors exécuter autant de commandes en root que vous le voulez.

Pour quitter le "mode root", tapez `exit` (ou faites la combinaison `Ctrl + D`).

Code : Console

```
root@mateo21-desktop:/home/mateo21# exit
exit
mateo21@mateo21-desktop:~$
```

Et vous voilà redevenu Simple Mortel 😊



Sous les autres distributions qu'Ubuntu, écrire "`su`" suffit à passer root.

Il est néanmoins recommandé dans ce cas d'ajouter un tiret en paramètre, c'est-à-dire d'écrire "`su -`". L'ajout du tiret a pour effet de rendre accessibles certains programmes destinés seulement à root. Par ailleurs, cela vous place directement dans le dossier personnel de root (`/root`).

adduser : ajouter un utilisateur

Maintenant que vous savez passer root (temporairement ou indéfiniment), nous allons pouvoir découvrir des commandes qui sont réservées à root.

`adduser` et `deluser` sont de celles-là. Si vous essayez de les appeler avec votre utilisateur normal, on vous dira que vous n'avez pas le droit de les utiliser. Seul root peut gérer les utilisateurs.

adduser : ajouter un utilisateur

La commande `adduser` permet d'ajouter un utilisateur. Vous devez au minimum fournir un paramètre : le nom de l'utilisateur à créer. Par exemple, pour créer un compte pour `patrick` :

Code : Console

```
root@mateo21-desktop:/home# adduser patrick
Ajout de l'utilisateur « patrick »...
Ajout du nouveau groupe « patrick » (1001)...
Ajout du nouvel utilisateur « patrick » (1001) avec le groupe « patrick »...
Création du répertoire personnel « /home/patrick »...
Copie des fichiers depuis « /etc/skel »...
```

Pensez à rajouter un `sudo` devant la commande si vous n'êtes pas déjà `root`, c'est-à-dire tapez `sudo adduser patrick`. Moi je n'ai pas eu à le faire car j'ai choisi de rester `root` indéfiniment en tapant `sudo su` auparavant.

Si vous tentez d'exécuter la commande avec votre compte limité, vous aurez une erreur de ce genre : "*adduser : Seul le superutilisateur peut ajouter un utilisateur ou un groupe sur le système*".

Le répertoire personnel de `patrick` est automatiquement créé (`/home/patrick`) et son compte est préconfiguré.

On vous demande ensuite de taper son mot de passe :

Code : Console

```
Entrez le nouveau mot de passe UNIX :
Retapez le nouveau mot de passe UNIX :
passwd : le mot de passe a été mis à jour avec succès
```

Tapez le mot de passe de `Patrick`, puis faites Entrée. Retapez le mot de passe pour valider.

Encore une fois, si vous ne voyez pas d'étoiles `*` quand vous tapez le mot de passe, c'est normal. C'est une sécurité pour qu'on ne puisse pas compter le nombre de caractères derrière votre épaule.

On vous propose ensuite de rentrer quelques informations personnelles sur `Patrick`, comme son nom, son numéro de téléphone... Si vous voulez le faire, faites-le, mais sinon sachez que vous pouvez taper Entrée sans rien écrire, on ne vous embêtera pas 🤪

Code : Console

```
Modification des informations relatives à l'utilisateur patrick
Entrez la nouvelle valeur ou « Entrée » pour conserver la valeur proposée
  Nom complet []:
  N° de bureau []:
  Téléphone professionnel []:
  Téléphone personnel []:
  Autre []:
Ces informations sont-elles correctes ? [o/N] o
```

On vous demande à la fin de confirmer par un "o" (oui) que tout est bon. Tapez Entrée, et ça y est, `patrick` est créé !

passwd : changer le mot de passe

S'il était nécessaire de changer le mot de passe de `patrick` par la suite, utilisez la commande `passwd` en indiquant en paramètre le

nom du compte à modifier.

Code : Console

```
root@mateo21-desktop:/home# passwd patrick
Entrez le nouveau mot de passe UNIX :
Retapez le nouveau mot de passe UNIX :
passwd : le mot de passe a été mis à jour avec succès
```



Attention ! Si vous appelez `passwd` sans préciser de compte en paramètre, c'est le mot de passe de root que vous allez changer !!!

deluser : supprimer un compte

Patrick vous ennuie ? Patrick est parti ? Si son compte n'est plus nécessaire (ou que vous voulez vous venger 🐱) vous pouvez le supprimer avec `deluser`.

Code : Console

```
deluser patrick
```

Aucune confirmation ne vous sera demandée !



Surtout, ne supprimez pas votre compte utilisateur ! Par exemple, je ne dois surtout pas supprimer le compte `mateo21`. En effet, si je le fais, il n'y aura plus que root sur la machine... et Ubuntu interdit de se logger en root. Par conséquent, au prochain démarrage de la machine vous ne pourrez pas vous connecter... et vous serez complètement coincé !

Toutefois, cette commande seule ne supprime pas le répertoire personnel de Patrick. Si vous voulez supprimer aussi son home et tous ses fichiers personnels, utilisez le paramètre `--remove-home` :

Code : Console

```
deluser --remove-home patrick
```



`adduser` et `deluser` sont des commandes qui n'existent que dans Debian et tous ses descendants, dont Ubuntu. Partout ailleurs, on doit utiliser `useradd` et `userdel` qui sont les commandes Unix traditionnelles qui marchent partout. Elles font en gros la même chose mais de manière beaucoup plus basique : si vous n'appelez pas `passwd` vous-même le compte ne sera pas activé et n'aura pas de mot de passe.

addgroup : ajouter un groupe

Je vous l'ai dit au début : chaque utilisateur appartient à un groupe.



Oui mais dans ce cas, à quel groupe appartiennent les utilisateurs `mateo21` et `patrick` ? On n'a rien défini nous !

En effet, si vous ne définissez rien, un groupe du même nom que l'utilisateur sera automatiquement créé. Ainsi, `mateo21` appartient au groupe `mateo21`, et `patrick` au groupe `patrick`.

On peut le vérifier en regardant à qui appartiennent les dossiers dans `/home` via un `ls -l` :

Code : Console

```
root@mateo21-desktop:~# cd /home
root@mateo21-desktop:/home# ls -l
total 24
drwx----- 2 root    root    16384 2007-09-19 18:22 lost+found
drwxr-xr-x 65 mateo21 mateo21 4096 2007-11-15 22:40 mateo21
drwxr-xr-x 2 patrick patrick 4096 2007-11-15 23:00 patrick
```

Souvenez-vous : la 3ème colonne indique le propriétaire du fichier ou dossier, et la 4ème indique le groupe qui possède ce fichier ou dossier.

Ainsi, le dossier mateo21 appartient à l'utilisateur mateo21 et au groupe mateo21.

De même pour patrick.

On constatera par ailleurs que lost+found appartient à root, et qu'il y a un groupe root (root fait donc partie du groupe root).

Bon, si tout le monde est dans son propre groupe, quel intérêt me direz-vous ?

Vous pourriez très bien vous contenter de ce fonctionnement (un utilisateur = un groupe), mais au cas où vous ayez beaucoup d'utilisateurs, je vais quand même vous montrer comment créer des groupes.

addgroup : créer un groupe

La commande addgroup crée un nouveau groupe. Vous avez juste besoin de spécifier le nom du groupe en paramètre :

Code : Console

```
root@mateo21-desktop:/home# addgroup amis
Ajout du groupe « amis » (identifiant 1002)...
Terminé.
```

Cool. Mais personne ne fait partie de ce groupe encore 😞

usermod : modifier un utilisateur

La commande usermod permet d'éditer un utilisateur. Elle possède plusieurs paramètres, on va en retenir 2 :

- **-l** : renomme l'utilisateur (le nom de son répertoire personnel ne sera pas changé par contre)
- **-g** : change de groupe

Si je veux mettre patrick dans le groupe amis, je ferai donc comme ceci :

Code : Console

```
usermod -g amis patrick
```

Et pour remettre patrick dans le groupe patrick comme il l'était avant :

Code : Console

```
usermod -g patrick patrick
```



Il est aussi possible de faire en sorte qu'un utilisateur appartienne à plusieurs groupes. Pour ce faire, utilisez le paramètre `-G` (majuscule).

Exemple : `usermod -G amis,paris,collegues patrick`

Séparez les noms des groupes par une virgule, sans espace entre chaque nom de groupe.



Faites très attention en utilisant `usermod` ! Lorsque vous utilisez `-G`, l'utilisateur change de groupe peu importe les groupes auxquels il appartenait auparavant.

Si vous voulez *ajouter* des groupes à un utilisateur (sans perdre les groupes auxquels il appartenait auparavant), utilisez `-a` :

```
usermod -aG amis patrick
```

delgroup : supprimer un groupe

Si vous voulez supprimer un groupe, c'est tout simple :

Code : Console

```
delgroup amis
```



`addgroup` et `delgroup` n'existent pas ailleurs que sous Debian et ses dérivés (même remarque que pour `adduser` et `deluser`).

Les commandes "traditionnelles" qui marchent partout sont `groupadd` et `groupdel`, mais elles offrent moins d'options.

chown : changer le propriétaire d'un fichier

L'utilisateur `root`, et seulement lui, peut changer le propriétaire d'un fichier.

Par exemple, supposons que `mateo21` possède dans son répertoire personnel un fichier appelé "rapport.txt".

Voici le résultat d'un `ls -l` pour ce fichier :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -l rapport.txt
-rw-r--r-- 1 mateo21 mateo21 0 2007-11-15 23:14 rapport.txt
```



Petite astuce : comme vous venez de le voir, si on précise un nom de fichier en dernier paramètre de la commande `ls`, on ne verra que ce fichier dans les résultats.

Le joker `*` est là aussi utilisable : `ls -l *.jpg` afficherait uniquement les images JPEG contenues dans ce dossier.

Ce fichier, je souhaite le "donner" à `patrick`. C'est là qu'intervient la commande `chown`.

chown : changer le propriétaire d'un fichier

La commande `chown`, qui doit être utilisée en tant que root, attend 2 paramètres au moins :

- Le nom du nouveau propriétaire
- Le nom du fichier à modifier

Cela donne donc :

Code : Console

```
chown patrick rapport.txt
```

On peut voir ensuite que patrick est bien le nouveau propriétaire du fichier :

Code : Console

```
root@mateo21-desktop:/home/mateo21# ls -l rapport.txt  
-rw-r--r-- 1 patrick mateo21 0 2007-11-15 23:14 rapport.txt
```

Seulement... il appartient toujours au groupe mateo21 !

chgrp : changer le groupe propriétaire d'un fichier

chgrp s'utilise exactement de la même manière que chown, à la différence près qu'il affecte cette fois le groupe propriétaire d'un fichier.

Code : Console

```
chgrp amis rapport.txt
```

... affectera le fichier rapport.txt au groupe "amis".

Un petit ls -l nous confirmera que rapport.txt appartient désormais à patrick et au groupe amis :

Code : Console

```
root@mateo21-desktop:/home/mateo21# ls -l rapport.txt  
-rw-r--r-- 1 patrick amis 0 2007-11-15 23:14 rapport.txt
```

chown peut aussi changer le groupe propriétaire d'un fichier !

Eh oui ! C'est d'ailleurs l'astuce que j'utilise le plus souvent :

Code : Console

```
chown patrick:amis rapport.txt
```

... affectera le fichier à l'utilisateur patrick et au groupe amis.

Il suffit de séparer par un symbole deux-points ":" le nom du nouvel utilisateur (à gauche) et le nom du nouveau groupe (à droite).

-R : affecter récursivement les sous-dossiers

Très utile aussi, l'option -R de chown. Elle modifie tous les sous-dossiers et fichiers contenus dans un dossier pour y affecter un nouvel utilisateur (et un nouveau groupe si on utilise la technique du deux-points qu'on vient de voir).

Par exemple, si je suis un beau salaud et que je veux donner tout le contenu du dossier personnel de patrick à mateo21 (et au groupe mateo21), c'est très simple 🤪

Code : Console

```
chown -R mateo21:mateo21 /home/patrick/
```

Résultat :

Code : Console

```
root@mateo21-desktop:/home# ls -l
total 24
drwx----- 2 root    root    16384 2007-09-19 18:22 lost+found
drwxr-xr-x 62 mateo21 mateo21 4096 2007-11-15 23:19 mateo21
drwxr-xr-x 2 mateo21 mateo21 4096 2007-11-15 23:00 patrick
```

Tous les fichiers à l'intérieur du dossier de patrick appartiennent maintenant aussi à mateo21.

Je suis vraiment trop diabolique 🤪

chmod : modifier les droits d'accès

On attaque maintenant la partie la plus "coton" du chapitre si je puis dire : les droits d'accès.

Le fonctionnement des droits

Chaque fichier et chaque dossier possèdent une liste de droits. C'est une liste qui dit qui a le droit de voir le fichier, de le modifier et de l'exécuter.

Vous avez déjà vu des listes de droits, oui oui ! Lorsque vous faites un ls -l, c'est la première colonne :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -l
total 40
drwxr-xr-x 2 mateo21 mateo21 4096 2007-11-13 21:53 Desktop
drwxr-xr-x 2 mateo21 mateo21 4096 2007-11-13 13:46 Documents
lrwxrwxrwx 1 mateo21 mateo21 26 2007-09-19 18:31 Examples -
> /usr/share/example-content
drwxr-xr-x 2 mateo21 mateo21 4096 2007-09-25 20:28 images
drwxr-xr-x 2 mateo21 mateo21 4096 2007-10-19 01:21 Images
drwxr-xr-x 3 mateo21 mateo21 4096 2007-09-25 11:11 log
drwxr-xr-x 2 mateo21 mateo21 4096 2007-10-19 01:21 Modèles
drwxr-xr-x 2 mateo21 mateo21 4096 2007-10-19 01:21 Musique
drwxr-xr-x 2 mateo21 mateo21 4096 2007-10-19 01:21 Public
-rw-r--r-- 1 mateo21 mateo21 0 2007-11-15 23:14 rapport.txt
drwxr-xr-x 3 mateo21 mateo21 4096 2007-09-19 19:51 tutos
drwxr-xr-x 2 mateo21 mateo21 4096 2007-10-19 01:21 Vidéos
```

Vous voyez tous ces d, r, w et x au début ? Ce sont ce qu'on appelle les droits d'accès du fichier ou dossier.

On peut voir 5 lettres différentes. Voici leurs significations :

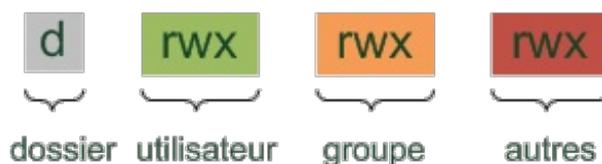
- **d** (Directory) : indique si l'élément est un dossier.
- **l** (Link) : indique si l'élément est un lien (raccourci).
- **r** (Read) : indique si on peut lire l'élément.
- **w** (Write) : indique si on peut modifier l'élément.
- **x** (eXecute) : si c'est un fichier, "x" indique qu'on peut l'exécuter. Ce n'est utile que pour les fichiers exécutables (programmes et scripts).
Si c'est un dossier, "x" indique qu'on peut le "traverser", c'est-à-dire qu'on peut voir les sous-dossiers qu'il contient si on a le droit de lecture dessus.

Si la lettre apparaît, c'est que le droit existe. S'il y a un tiret à la place, c'est qu'il n'y a pas de droit.



Pourquoi est-ce qu'on voit parfois plusieurs fois r, w et x ?

Les droits sont découpés en fonction des utilisateurs :



Le premier élément ("d") mis à part, on constate que r, w et x sont répétés 3 fois en fonction des utilisateurs :

- Le premier triplet rwx indique les droits que possède le **propriétaire** du fichier sur ce fichier.
- Le second triplet rwx indique les droits que possèdent les autres membres du **groupe** sur ce fichier.
- Enfin, le dernier triplet rwx indique les droits que possèdent tous les **autres** utilisateurs de la machine sur ce fichier.

Prenons un cas concret, le fichier rapport.txt :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -l rapport.txt
-rw-r--r-- 1 mateo21 mateo21 0 2007-11-15 23:14 rapport.txt
```

Ses droits sont :

-rW-r--r--

- **-** : le premier tiret indique qu'il ne s'agit pas d'un dossier. S'il y avait eu un "d" à la place, cela aurait indiqué qu'il s'agissait d'un dossier.
- **rW-** : indique que le propriétaire du fichier, mateo21 en l'occurrence, peut lire et modifier (et donc supprimer) le fichier. En revanche, il ne peut pas l'exécuter car il n'a pas de x à la fin. Je rappelle que quiconque peut modifier un fichier a aussi le droit de le supprimer.
- **r--** : tous les gens qui font partie du groupe "mateo21" mais qui ne sont pas "mateo21" peuvent seulement lire le fichier. Ils ne peuvent ni le modifier, ni l'exécuter. Je reconnais qu'avoir un nom de groupe identique au nom d'utilisateur peut embrouiller : si vous êtes aussi bien organisé que sur mon premier schéma, on parlerait plutôt du groupe "famille".
- **r--** : tous les autres (ceux qui ne font pas partie du groupe "mateo21") peuvent seulement lire le fichier.

En résumé, ces droits nous apprennent que l'élément est un fichier, que mateo21 peut le lire et le modifier, et que tous les autres utilisateurs peuvent seulement le lire.



Et root ?
Il a quels droits ?

Souvenez-vous d'une chose : root a TOUS les droits. Il peut tout faire : lire, modifier, exécuter n'importe quel fichier.

chmod : modifier les droits d'accès

Maintenant que nous savons voir et comprendre les droits d'accès d'un fichier, nous allons apprendre à les modifier à l'aide de la commande **chmod**.

Une précision importante pour commencer : contrairement aux commandes précédentes, vous n'avez pas besoin d'être root pour utiliser chmod. Vous devez juste être propriétaire du fichier dont vous voulez modifier les droits d'accès.

chmod est un petit peu délicat à utiliser. En effet, on peut attribuer les droits sur un fichier / dossier via plusieurs méthodes différentes, la plus courante étant celle des chiffres.

Attribuer des droits avec des chiffres (chmod absolu)

J'espère que vous êtes chauds pour un peu d'addition 🤔

Il va falloir faire un petit peu de calcul mental. En effet, on attribue un chiffre à chaque droit :

Droit	Chiffre
r	4
w	2
x	1

Si vous voulez combiner ces droits, il va falloir additionner les chiffres correspondants.

Ainsi, pour attribuer le droit de lecture et de modification, il faut additionner $4 + 2$, ce qui donne 6. Le chiffre 6 signifie donc "Droit de lecture et d'écriture".

Voici la liste des droits possibles et la valeur correspondante :

Droits	Chiffre	Calcul
---	0	$0 + 0 + 0$
r--	4	$4 + 0 + 0$
-w-	2	$0 + 2 + 0$
--x	1	$0 + 0 + 1$
rw-	6	$4 + 2 + 0$
-wx	3	$0 + 2 + 1$
r-x	5	$4 + 0 + 1$
rwX	7	$4 + 2 + 1$

C'est compris ? 😊

Avec ça, on peut calculer la valeur d'un triplet de droits. Il faut faire le même calcul pour les droits qu'on veut attribuer au propriétaire, au groupe et aux autres.

Par exemple, "640" indique les droits du propriétaire, du groupe et des autres (dans l'ordre).

- 6 : droit de lecture et d'écriture pour le propriétaire.
- 4 : droit de lecture pour le groupe.
- 0 : aucun droit pour les autres.

Le droit maximal qu'on puisse donner à tout le monde est 777 : droit de lecture, d'écriture et d'exécution pour le propriétaire, pour son groupe et pour tous les autres. Bref, avec un tel droit tout le monde peut tout faire sur ce fichier. Au contraire, avec un droit de 000, personne ne peut rien faire. A part root bien sûr.

Pour changer les droits sur le fichier rapport.txt, et être le seul autorisé à le lire et l'éditer, je dois exécuter cette commande :

Code : Console

```
chmod 600 rapport.txt
```

Un petit ls -l pour voir le résultat :

Code : Console

```
mateo21@mateo21-desktop:~$ ls -l rapport.txt
-rw----- 1 mateo21 mateo21 0 2007-11-15 23:14 rapport.txt
```

Bingo ! 😊

On a bien confirmation que seul le propriétaire du fichier, c'est-à-dire moi, peut lire et modifier le fichier !

Attribuer des droits avec des lettres (chmod relatif)

Il existe un autre moyen de modifier les droits d'un fichier. Il revient un peu au même mais permet parfois de paramétrer plus finement, droit par droit.

Dans ce mode, il faut savoir que :

- **u** = user (propriétaire)
- **g** = group (groupe)
- **o** = other (autres)

... et que :

- + signifie "ajouter le droit".
- - signifie "supprimer le droit".
- = signifie "affecter le droit".

Maintenant que vous savez cela, vous pouvez écrire :

Code : Console

```
chmod g+w rapport.txt
```

Signifie : "Ajouter le droit d'écriture au groupe".

Code : Console

```
chmod o-r rapport.txt
```

Signifie : "Enlever le droit de lecture aux autres".

Code : Console

```
chmod u+rx rapport.txt
```

Signifie : "Ajouter les droits de lecture et d'exécution au propriétaire".

Code : Console

```
chmod g+w,o-w rapport.txt
```

Signifie : "Ajouter le droit d'écriture au groupe et l'enlever aux autres".

Code : Console

```
chmod go-r rapport.txt
```

Signifie : "Enlever le droit de lecture au groupe et aux autres".

Code : Console

```
chmod +x rapport.txt
```

Signifie : "Ajouter le droit d'exécution à tout le monde".

Code : Console

```
chmod u=rwx,g=r,o=- rapport.txt
```

Signifie : "Affecter tous les droits au propriétaire, juste la lecture au groupe, rien aux autres".

Voilà ouf ! J'ai préféré vous expliquer le fonctionnement à travers des exemples concrets plutôt que de faire un cours théorique sur la syntaxe d'une des utilisations possibles de chmod 🤪

Normalement si vous suivez mes exemples, vous devriez être capable de tout faire !

Et toujours... -R pour affecter récursivement

Le paramètre -R existe aussi dans chmod. Si vous affectez des droits sur un dossier avec -R, tous ses fichiers et sous-dossiers récupéreront le même droit.

Si je veux être le seul à pouvoir lire, éditer et exécuter les fichiers de mon répertoire personnel et de tous ses fichiers, j'ai juste besoin d'écrire :

Code : Console

```
chmod -R 700 /home/mateo21
```

C'est tout !

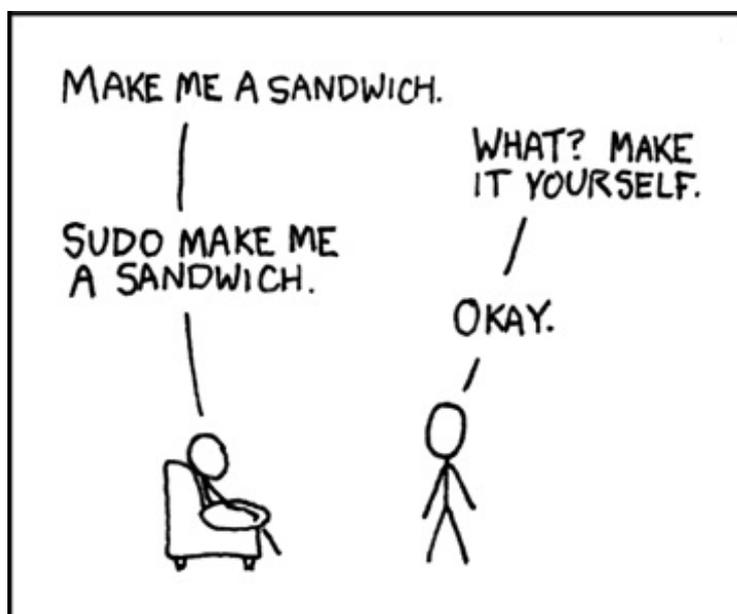
Vous êtes rentrés de plein pied dans le monde merveilleux des droits Unix !

Ce fonctionnement que vous avez découvert là existe depuis bien avant l'apparition du DOS. Il a l'avantage d'être puissant et répandu : tous les systèmes dérivés d'Unix, Linux compris, fonctionnent sur ce principe.

Vous avez découvert en particulier comment devenir root un moment.

J'ai peut-être déjà insisté dessus mais on ne le fait jamais trop : root n'est pas un jeu. Avoir tous les pouvoirs sur sa machine est dangereux, et même une personne expérimentée n'est pas à l'abri de faire une bêtise. Mon conseil serait donc : passez root uniquement lorsque c'est nécessaire, le reste du temps utilisez le compte avec les droits limités que vous avez créé à l'installation.

Ah au fait, il se trouve que vous êtes maintenant capables de comprendre une petite blague qui a fait le tour du Web (certains la portent même en T-Shirt !). La voici 😊



Crédits : xkcd.com

Pour ceux qui ne comprennent pas l'anglais, voici la traduction :

- Fais-moi un sandwich.
- Quoi ? Fais-le toi-même.
- Sudo fais-moi un sandwich.
- Ok.

Nano, l'éditeur de texte du débutant

Nous avons découvert plusieurs façons de voir le contenu d'un fichier en console. Mais... aucune des commandes que nous avons étudiées ne nous permettait d'éditer un fichier.

Pourquoi est-ce que j'ai repoussé le moment où je vous parlerais des éditeurs de texte ? Parce que c'est un des domaines les plus riches de la console ! Parmi les plus célèbres éditeurs de texte console de Linux, on compte :

- nano
- vim
- emacs

Que de jolis noms n'est-ce pas ? 😊

Des 3 que j'ai cités là, nano est de loin le plus simple à utiliser. Ce n'est pas pour rien si ce chapitre s'intitule "*Nano, l'éditeur de texte du débutant*". Tout un chapitre pour découvrir nano ne sera pas de trop.

Quant à vim (prononcez "vie aille ème") et emacs (prononcez "i mackse"), ils sont tous les deux très riches et très complets. En fait, ce sont très probablement les éditeurs de texte les plus complets et puissants qui existent au monde. Et ils sont en console eh oui.

Nous découvrirons vim plus tard dans ce cours. Quant à emacs, je ne le connais pas suffisamment bien moi-même pour me permettre d'en parler, je préfère donc ne pas en parler pour le moment plutôt que mal en parler 😊

Premiers pas avec nano

En sciences, le terme "nano" représente une toute petite unité. Par exemple, un atome a une taille d'environ 0.1 nanomètre.

Si l'éditeur de texte que je vais vous présenter s'appelle nano, c'est parce qu'il est tout petit. Il s'agit d'un programme très simple comparé à vi et emacs. Il nous conviendra tout à fait pour démarrer. Il possède assez peu de fonctions par rapport à vi et emacs (qui peuvent devenir de véritables machines de guerre), mais suffisamment pour commencer à vous débrouiller avec un éditeur de texte.

nano est un éditeur de texte, pas un traitement de texte !

Savez-vous vraiment ce qu'est un éditeur de texte ? Ne le confondez-vous pas avec un traitement de texte ?

Un **éditeur de texte** est un programme qui permet de modifier des fichiers de texte brut, sans mise en forme (gras, italique, souligné...). Sous Windows, on dispose d'un éditeur de texte très basique : le Bloc-Notes. Sous Linux, on a le choix entre nano, vi, emacs (et bien d'autres), sachant qu'au moins un de ceux-là est installé par défaut sur la plupart des distributions Linux.

Un **traitement de texte** est fait pour rédiger des documents mis en forme. Sous Windows, Word est le plus célèbre traitement de texte. Sous Linux, on possède l'équivalent : Open Office Writer. Ces programmes ne peuvent être utilisés qu'en mode graphique, car on ne peut pas vraiment faire de la mise en forme dans une console.



Quand a-t-on besoin d'un éditeur de texte ?

A chaque fois que vous devez éditer un fichier de texte brut. Sous Windows, vous avez l'habitude de voir des fichiers de texte brut au format .txt. Sous Linux, vous savez que l'extension importe peu (on peut trouver des fichiers en texte brut sans extension).

Les éditeurs de texte sont parfaits pour les programmeurs en particulier : ils permettent d'éditer des fichiers .c, .cpp, .h, .rb, .py (en fonction de votre langage de programmation).

Même si vous ne programmez pas, vous aurez besoin d'utiliser un éditeur de texte pour modifier des fichiers de configuration. Ces fichiers n'ont pas d'extension particulière, mais vous apprendrez à les reconnaître à force.

Après avoir appris à utiliser nano, nous nous ferons les dents sur nos premiers fichiers de configuration : le nanorc et le bashrc. Ce sera l'occasion pour vous de personnaliser votre nano et votre console 😊

Découverte de nano

Le nom complet de nano est "GNU nano", en référence au projet GNU dont je vous ai parlé dans le tout premier chapitre. Il s'agit d'un éditeur qui s'inspire de "pico", un éditeur de texte plus ancien qui se voulait lui aussi très simple d'utilisation.

Voici le logo de GNU nano, tiré du [site officiel](#) :

Code : Console

```

.
                                The
  iLE88Dj.  :jD88888Dj:
.LGitE888D.f8GjjjL8888E;      .d8888b.  888b   888 888   888
iE   :8888Et.   .G8888.      d88P  Y88b 88888b  888 888   888
;i   E888,      ,8888,      888   888 888888b 888 888   888
      D888,      :8888:      888   888888888 Y88b 888   888
      D888,      :8888:      888 888888 888 Y88b888 888   888
      D888,      :8888:      888   888 888 Y888888 888   888
      D888,      :8888:      Y88b d88P 888   Y8888 Y88b. .d88P
      888W,      :8888:      "Y8888P88 888   Y888   "Y88888P"
      W88W,      :8888:
      W88W:      :8888:      888888b.  8888b.  888888b.  .d88b.
      DGGD:      :8888:      888 "88b   "88b 888 "88b d88""88b
                        :8888:      888 888 .d8888888 888 888 888 888
                        :8888:      888 888 888 888 888 888 Y88..88P
                        :8888:      888 888 "Y8888888 888 888 "Y88P"
                        E888i
                        tW88D

```

Vous verrez souvent que les programmeurs sont friands d'images ASCII, c'est-à-dire d'images réalisées à partir de texte. Oui je sais ils sont fous ces programmeurs 🤪

Bon, si on démarrait nano ?

Pour ce faire, c'est très simple, il vous suffit de taper nano dans la console :

Code : Console

```
nano
```

L'éditeur nano s'ouvre immédiatement :

```
GNU nano 2.0.6          Nouvel espace

^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper    ^C Pos. cur.
^X Quitter  ^J Justifier ^W Chercher  ^V Page suiv.^U Coller   ^T Orthograp.
```

(Nota : je suis obligé de prendre des captures d'écran car la disposition de la console est particulière, c'est en quelque sorte un programme "semi-graphique")

(Nota bis : j'ai configuré ma console pour qu'elle s'affiche en vert sur noir, je trouve ça plus lisible, mais la couleur importe peu bien entendu)

Dès lors, vous pouvez commencer à taper du texte. Par exemple :

```
GNU nano 2.0.6          Nouvel espace          Modifié

Salut les Zéros, ça va ?
Je teste nano ! Un deux... un deux... █

^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper    ^C Pos. cur.
^X Quitter  ^J Justifier ^W Chercher  ^V Page suiv.^U Coller   ^T Orthograp.
```

C'est aussi simple que cela !

Ne rigolez pas, je précise qu'il "suffit de taper du texte" car ce n'est pas aussi simple sous d'autres éditeurs de texte comme "vi"



Les raccourcis clavier de nano

En bas de votre écran, vous pouvez voir un espace d'aide. Que signifie-t-il exactement ?

Il s'agit d'un aide-mémoire pour vous rappeler à tout moment les commandes principales que vous pouvez lancer sous nano.

```
^G Aide      ^O Ecrire    ^R Lire fich.^Y Page préc.^K Couper     ^C Pos. cur.
^X Quitter   ^J Justifier^W Chercher  ^V Page suiv.^U Coller    ^T Orthograp.
```

Le symbole ^ signifie "Ctrl" (la touche Contrôle de votre clavier). Ainsi, pour quitter nano il suffit de taper Ctrl + X.

Voici les raccourcis les plus importants :

- Ctrl + G : afficher l'aide
- Ctrl + K : couper la ligne de texte (et la mettre dans le presse-papier)
- Ctrl + U : coller la ligne de texte que vous venez de couper
- Ctrl + C : afficher à quel endroit du fichier votre curseur est positionné (numéro de ligne...)
- Ctrl + W : rechercher dans le fichier
- Ctrl + O : enregistrer le fichier (écrire)
- Ctrl + X : quitter nano

Vous pouvez vous déplacer dans le fichier avec les flèches du clavier ainsi qu'avec les touches "Page Up" et "Page Down" pour avancer de page en page (les raccourcis Ctrl + Y et Ctrl + V fonctionnent aussi).

Si l'aide-mémoire vous encombre, vous pouvez gagner de la place en appuyant sur Echap puis sur X. Vous pouvez le réafficher avec la même suite de touches.

La recherche

La combinaison de touches Ctrl + W lance une recherche dans le fichier :

```
GNU nano 2.0.6          Nouvel espace          Modifié
Salut les Zéros, ça va ?
Je teste nano ! Un deux... un deux...

Recherche:
^G Aide      ^Y Prem. lig.^R Remplacer ^W Début paraM-C Resp.cassM-R Exp. ratio.
^C Annuler   ^V Dern. Lig.^T Aller lig.^O Fin para. M-B ->Arrière^P Précédente
```

Il vous suffit d'écrire le mot que vous recherchez..

```
GNU nano 2.0.6          Nouvel espace          Modifié
Salut les Zéros, ça va ?
Je teste nano ! Un deux... un deux...

Recherche: deux
^G Aide      ^Y Prem. lig.^R Remplacer ^W Début paraM-C Resp.cassM-R Exp. ratio.
^C Annuler   ^V Dern. Lig.^T Aller lig.^O Fin para. M-B ->Arrière^P Précédente
```

... puis de taper Entrée :

```
GNU nano 2.0.6          Nouvel espace          Modifié
Salut les Zéros, ça va ?
Je teste nano ! Un deux... un deux...

[ La recherche a fait le tour ]
^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper    ^C Pos. cur.
^X Quitter   ^J Justifier ^W Chercher  ^V Page suiv.^U Coller    ^T Orthograp.
```

Le curseur est automatiquement positionné à la première occurrence trouvée. Si le curseur est à la fin, la recherche recommence du début.



Si vous voulez sortir du mode recherche, tapez Ctrl + C (Annuler)

Si vous voulez aller au résultat suivant (au "deux" suivant), refaites Ctrl + W pour lancer une recherche. La recherche précédente est sauvegardée et apparaît entre crochets. Si vous voulez rechercher le même mot (et donc aller au résultat suivant), tapez juste Entrée sans écrire de mot à rechercher :

```
GNU nano 2.0.6          Nouvel espace          Modifié
Salut les Zéros, ça va ?
Je teste nano ! Un deux... un deux...

Recherche [deux]:
^G Aide      ^Y Prem. lig.^R Remplacer ^W Début paraM-C Resp.cassM-R Exp. ratio.
^C Annuler   ^V Dern. Lig.^T Aller lig.^O Fin para. M-B ->Arrière^P Précédente
```

Enregistrer et quitter

Pour enregistrer à tout moment, faites Ctrl + O.

Si vous essayez de quitter (Ctrl + X) sans enregistrer auparavant, un message vous demandera si vous voulez enregistrer :

```
GNU nano 2.0.6          Nouvel espace          Modifié
Salut les Zéros, ça va ?
Je teste nano ! Un deux... un deux...

Sauver l'espace modifié (RÉPONDRE « Non » EFFACERA LES CHANGEMENTS) ?
O Oui
N Non      ^C Annuler
```

Si vous appuyez sur la touche "o", vous passerez en mode enregistrement.

Si vous appuyez sur la touche "n", nano quittera sans enregistrer.

Si vous faites la combinaison "Ctrl + C", vous annulerez votre demande de sortie de nano. Vous ne quitterez donc pas nano.

En appuyant sur "o", vous vous retrouvez en mode enregistrement. Tapez juste le nom du fichier que vous voulez créer, puis faites Entrée :

```
GNU nano 2.0.6          Nouvel espace          Modifié
Salut les Zéros, ça va ?
Je teste nano ! Un deux... un deux...

Nom du fichier à écrire: salut.txt
^G Aide                ^T Parcourir          M-M Format Mac        M-P Ajout (au début)
^C Annuler             M-D Format DOS        M-A Ajout (à la fin) M-B Copie de sécu.
```

Après ça, nano sera fermé et vous retrouverez votre bonne vieille ligne de commande 😊

Les paramètres de la commande nano

Lorsque vous appelez nano dans la ligne de commande, vous pouvez spécifier plusieurs paramètres. Le plus courant est d'indiquer en paramètre le nom du fichier qu'on veut ouvrir :

Code : Console

```
nano salut.txt
```

... ouvrira le fichier salut.txt qu'on vient de créer.

Si le fichier n'existe pas, il sera automatiquement créé par nano lors du premier enregistrement.

A part ça, la commande nano accepte de nombreux paramètres. J'en ai sélectionné 3 pour vous qui me semble parmi les plus utiles :

- **-m** : autorise l'utilisation de la souris sous nano. En console, oui oui. Vous pouvez vous en servir pour cliquer avec votre souris sur la zone de texte où vous voulez placer votre curseur.
- **-i** : indentation automatique. L'alinéa (tabulations) de la ligne précédente sera respecté lorsque vous irez à la ligne. Très utile lorsque vous éditez un fichier de code source.
- **-A** : active le retour intelligent au début de la ligne. Normalement, lorsque vous appuyez sur la touche Origine (aussi

connue sous le nom de Home), située à côté de la touche Fin, le curseur se repositionne au tout début de la ligne. Avec cette commande, il se positionnera après les alinéas. Comme -i, il s'agit d'une option surtout utile pour les programmeurs.

Si je veux lancer nano avec toutes ces options à la fois, je peux donc écrire :

Code : Console

```
nano -miA salut.txt
```

Configurer nano avec .nanorc

Vous savez maintenant utiliser nano. Comme vous avez pu le voir, ce n'est pas très compliqué. Il suffit d'apprendre un peu les raccourcis clavier les plus utiles, et on peut rapidement s'en servir.

Justement... si on utilisait nano pour quelque chose d'utile ? Non parce que bon, le fichier "salut.txt" c'est mignon, mais ça va pas nous faire avancer 😊

Alors pour l'occasion, je me suis dit que j'allais vous faire éditer quelques fichiers de configuration. Par exemple, il existe un fichier de configuration de nano qui indique toutes vos préférences. Celui-ci s'appelle ".nanorc".

Pourquoi .nanorc ?

La plupart des fichiers de configuration commencent par un point. Cela permet de "cacher" le fichier quand on fait un "ls". Bien entendu, comme vous devriez le savoir maintenant, les fichiers cachés peuvent toujours être affichés en utilisant le paramètre -a :
`ls -a.`

Chaque utilisateur de la machine peut créer son propre fichier de configuration .nanorc dans son répertoire personnel (home). Chez moi, ce fichier doit être placé à la position : /home/mateo21/.nanorc. Ce fichier est lu par nano à chaque fois que vous le démarrez.



Je viens de regarder la liste des fichiers de mon home, mais même en incluant les fichiers cachés avec -a je ne vois pas de fichier appelé .nanorc !

En effet, il se peut que ce fichier n'existe pas chez vous. Si le fichier .nanorc n'existe pas, nano sera chargé avec les options par défaut.

Code : Console

```
mateo21@mateo21-desktop:~$ ls -a
.                .gimp-2.4       .nautilus
..               .gksu.lock     nautilus-debug-log.txt
.armagetron     .gnome         .openoffice.org2
.armagetronad  .gnome2       .pgadmin3
.assistant     .gnome2_private .pgpass
.bash_history  .gnome_private .profile
.bash_logout   .gstreamer-0.10 Public
.bashrc        .gtk-bookmarks .purple
.blender       .gtkrc-1.2-gnome2 .putty
.cache         .gweled        .qt
.config        .ICEauthority  .recently-used
.DCOPserver_mateo21-desktop_0 .icons         .recently-used.xbel
.DCOPserver_mateo21-desktop_0:0 images        salut.txt
.designer      Images         .ssh
Desktop       .inkscape     .sudo_as_admin_successful
.dmrc         .java         .themes
Documents     .jedit        .thumbnails
.emilia       .kde          .Trash
.esd_auth     .lessht       .tsclient
.evolution    .lgames       tutos
Examples     .local        .update-manager-core
```

```
.face          log          .update-notifier
.fbrc         .macromedia  Vidéos
.filezilla    .mcp         .vlc
.fontconfig   .mcp         .w3m
.gaim         .metacity    .wine
.gconf        Modèles     .wormux
.gconfd       .mozilla     .Xauthority
.geany        .mozilla-thunderbird .xine
.gimp-2.2     Musique     .xsession-errors
```

Comme vous pouvez le voir, il n'y a pas de `.nanorc` chez moi !

Création du `.nanorc`

Pas de `.nanorc` ? Pas de problème, il suffit de le créer. On peut par exemple faire :

Code : Console

```
nano .nanorc
```

Cette commande ouvre nano. Comme le fichier `.nanorc` n'existe pas, un document vide est ouvert. Le fichier `.nanorc` sera créé lorsque vous enregistrerez.

```
GNU nano 2.0.6      Fichier : .nanorc
[ Nouveau fichier ]
^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper    ^C Pos. cur.
^X Quitter   ^J Justifier ^W Chercher  ^V Page suiv.^U Coller    ^T Orthograp.
```

Un `.nanorc` neuf (et vide)

Notez que le nom du fichier est déjà indiqué en haut de l'écran

Dans ce fichier, vous devez écrire une commande par ligne.

Chaque commande commence par un `set` (pour activer) ou un `unset` (pour désactiver) suivi de l'option qui vous intéresse.

Par exemple, pour activer la souris, écrivez :

Code : Console

```
set mouse
```

Ainsi, à chaque fois nano sera chargé avec la prise en charge de la souris. Vous n'aurez pas besoin de réécrire systématiquement le paramètre -m qu'on a vu tout à l'heure 😊

On peut faire de même pour éviter d'avoir à réécrire les paramètres -i et -A à chaque fois avec d'autres séries de set. Au final, on écrira ceci :

Code : Console

```
set mouse
set autoindent
set smarthome
```

Enregistrez le fichier avec Ctrl + O. Le nom du fichier est automatiquement écrit pour vous (car on l'a donné en paramètre en ouvrant nano) :



```
GNU nano 2.0.6      Fichier : .nanorc      Modifié
set mouse
set autoindent
set smarthome

Nom du fichier à écrire: .nanorc
^G Aide          ^T Parcourir    M-M Format Mac  M-P Ajout (au début)
^C Annuler      M-D Format DOS  M-A Ajout (à la fin) M-B Copie de sécu.
```

Vous pouvez ensuite faire Ctrl + X pour quitter nano.

Je vous rappelle que pour que ces options soient prises en compte, il faut démarrer une nouvelle session de nano (c'est pour ça que la souris n'a pas automatiquement marché dès que vous avez enregistré le fichier).

Si vous relancez nano ensuite, vous pouvez constater que la souris fonctionne, et que les options d'indentation automatique et de retour à la ligne intelligent fonctionnent elles aussi 😊

Le nanorc global et la coloration syntaxique

Ce fichier `.nanorc` dans votre home est très pratique, car il vous permet de définir vos propres options. Mais si vous avez 10 utilisateurs sur votre machine et que vous voulez activer le support de la souris pour tout le monde, vous n'allez quand même pas créer 10 fichiers `.nanorc` pour chacun !

Il existe un fichier `nanorc` "global" qui est pris en compte pour tout le monde. Celui-ci est situé dans `/etc/nanorc` (attention il n'y a pas de point devant cette fois).

Ce fichier ne peut être modifié que par root. Je vous conseille donc de l'ouvrir avec un `sudo` (ou dans une console en root si vous avez fait `sudo su` avant) :

Code : Console

```
sudo nano /etc/nanorc
```

Normalement, ce fichier existe déjà. Comme vous pouvez le constater, il est bien rempli :

```
GNU nano 2.0.6 Fichier : /etc/nanorc
## Sample initialization file for GNU nano.
##
## Please note that you must have configured nano with --enable-nanorc
## for this file to be read! Also note that this file should not be in
## DOS or Mac format, and that characters specially interpreted by the
## shell should not be escaped here.
##
## To make sure a value is disabled, use "unset <option>".
##
## For the options that take parameters, the default value is given.
## Other options are unset by default.
##
## Quotes inside string parameters don't have to be escaped with
## backslashes. The last double quote in the string will be treated as
## its end. For example, for the "brackets" option, ""')>]]" will match
## ", ', ), >, ], and }.
##
## Use auto-indentation.
# set autoindent
[ Lecture de 263 lignes ]
^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper     ^C Pos. cur.
^X Quitter   ^J Justifier^W Chercher  ^V Page suiv.^U Coller     ^T Orthograp.
```

Il sert en fait de fichier d'exemple. Toutes les options disponibles dans un `nanorc` sont présentes, mais elles sont précédées d'un `#` qui signifie qu'il s'agit d'un commentaire. Les commentaires sont ignorés par nano.

Le début du fichier vous explique (en anglais) que c'est un fichier d'initialisation d'exemple de nano.

Après le petit blabla d'introduction, vous avez la liste des options disponibles. Toutes sont commentées. La première est `autoindent`.

Code : Console

```
# set autoindent
```

Supprimez juste le `#` pour décommenter la ligne et donc pour activer l'indentation automatique pour tous les utilisateurs.

Code : Console

```
set autoindent
```

Vous pouvez parcourir le fichier à la recherche d'options intéressantes que vous voulez activer.

Vers la fin, vous verrez une section appelée "color setup", qui commence par ces lignes-là :

Code : Console

```
## Nanorc files
# include "/usr/share/nano/nanorc.nanorc"

## C/C++
# include "/usr/share/nano/c.nanorc"

## HTML
# include "/usr/share/nano/html.nanorc"
```

Je vous invite à décommenter toutes les lignes d'include. Cela permettra d'activer la coloration "intelligente" de vos fichiers selon leur type. Vous pourrez ainsi avoir des fichiers HTML colorés, des fichiers C colorés, des fichiers nanorc colorés, etc. 😊

Enregistrez le fichier, puis quittez nano.



Si vous avez une erreur à l'enregistrement, cela signifie que vous n'avez pas ouvert le fichier en root. Seul root a le droit de modifier ce fichier. Fermez nano et relancez-le avec un sudo cette fois.

Relancez nano en ouvrant par exemple un fichier HTML :

```
GNU nano 2.0.6      Fichier : index.html

<ul>
  <li><strong>Qu'est-ce que c'est que ce site ?</strong> : c'est un site $
  <li><strong>Qui appelle-t-on "les Zéros" ?</strong> : c'est vous, les v$
  <li><strong>Ça coûte combien ?</strong> : tout cela est gratuit. En eff$
</ul>

<div id="enbref">
<div id="calendar">
  <h3><a href="evenements.html?cal_m=12&cal_y=2007">Evènements</a></h3>
  <table>
    <thead>
      <tr>
        <th colspan="7">
          <a href="index.html?cal_m=11&cal_y=2007">&l$
          <a href="index.html?cal_m=12&cal_y=2007">Dé$
          <a href="index.html?cal_m=1&cal_y=2008">&ra$
        </th>
      </tr>
    </thead>
  </table>

^G Aide          ^O Écrire        ^R Lire fich.   ^Y Page préc.  ^K Couper      ^C Pos. cur.
^X Quitter      ^J Justifier   ^W Chercher   ^V Page suiv. ^U Coller     ^I Orthograp.
```

Les fichiers HTML sont maintenant automatiquement colorés

Les balises sont en bleu, le texte dans la couleur par défaut (en vert chez moi).

Même le fichier `/etc/nanorc` est maintenant coloré de manière intelligente :

```
GNU nano 2.0.6      Fichier : /etc/nanorc

## Fix numeric keypad key confusion problem.
# set rebindkeypad

## Do extended regular expression searches by default.
# set regexp

## Make the Home key smarter.  When Home is pressed anywhere but at the
## very beginning of non-whitespace characters on a line, the cursor
## will jump to that beginning (either forwards or backwards).  If the
## cursor is already at that position, it will jump to the true
## beginning of the line.
set smarthome

## Use smooth scrolling as the default.
# set smooth

## Use this spelling checker instead of the internal one.  This option
## does not properly have a default value.
##

[ Lecture de 263 lignes ]
^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper     ^C Pos. cur.
^X Quitter   ^J Justifier ^W Chercher  ^V Page suiv.^U Coller    ^T Orthograp.
```

Une coloration spéciale... pour les fichiers nanorc !

Nano se base sur l'extension du fichier pour activer la coloration syntaxique spéciale pour ce type de fichier.

Bon, honnêtement la coloration de nano n'est pas très performante (comparée à celle de "vi" par exemple), mais on s'en contentera pour le moment 😊

Personnaliser la coloration syntaxique

Vous pouvez, si vous le désirez, modifier les couleurs de la coloration syntaxique.

Rendez-vous dans le dossier `/usr/share/nano`. Vous y trouverez plusieurs fichiers `nanorc`, un par langage.

Vous pouvez par exemple ouvrir celui du langage HTML et l'éditer (attention il faut être root là encore).

```
GNU nano 2.0.6      Fichier : html.nanorc
## Here is a short example for HTML.
##
syntax "html" "\.(html|htm|tpl)$"
color brightblue start="<" end=">"
color white "&[^;[[:space:]]*;"

[ Lecture de 5 lignes ]
^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper    ^C Pos. cur.
^X Quitter   ^J Justifier ^W Chercher  ^V Page suiv.^U Coller    ^T Orthograp.
```

La première ligne, qui commence par "syntax", indique les extensions de fichiers qui déclencheront la coloration HTML. Pour ma part j'ai défini "html", "htm" et "tpl". Vous pouvez en ajouter si vous le désirez, il suffit de séparer chaque extension par une barre verticale "|" (Alt Gr + 6 sur un clavier français).

Ensuite... Il faut savoir utiliser les expressions régulières pour paramétrer les éléments que vous voulez colorer. Je ne vais pas vous faire un cours là-dessus ici, ce serait un peu long 😊 J'en ai parlé dans le [tuto PHP](#) pour ceux que ça intéresse.

Pour les couleurs, vous pouvez indiquer des noms de couleur en anglais : red, blue, green, black, white... Vous pouvez rajouter le mot "bright" devant pour utiliser une couleur plus claire : "brightgreen" signifie donc "vert clair".

Voilà, avec ça vous pouvez déjà commencer à personnaliser un peu mieux votre nano ! 😊

Configurer sa console avec .bashrc

Tout comme il existe un fichier de configuration de nano, il existe un fichier de configuration de l'ensemble de la console : le .bashrc. Il se situe dans votre répertoire personnel et celui-ci existe déjà normalement.

Code : Console

```
mateo21@mateo21-desktop:/usr/share/nano$ cd
mateo21@mateo21-desktop:~$ nano .bashrc
```

Bon, nano ne colore pas ce fichier et c'est dommage. Il faudrait paramétrer un peu plus finement la coloration syntaxique mais on ne va pas s'attarder à ça pour le moment (vous aurez tout le loisir de le faire à la fin du tuto si vous vous ennuyez).

Edition du .bashrc personnel



Le fichier .bashrc est un peu complexe pour les simples mortels que nous sommes (pour le moment 🙄), donc attention à ne pas éditer n'importe quoi au risque de tout casser. Bref, faites juste un peu attention et tout ira bien.

Nous n'allons pas nous intéresser au `.bashrc` dans le détail ligne par ligne. Nous allons seulement voir quelques lignes faciles à éditer qui vous permettent de personnaliser un peu votre console.

Personnaliser l'invite de commande

Le fichier `.bashrc` vous permet entre autres choses de personnaliser l'invite de commandes. Vous savez, ce petit message qui s'affiche devant votre curseur dans la console :

Code : Console

```
mateo21@mateo21-desktop:~$
```

Rendez-vous plus bas dans le fichier, jusqu'à tomber sur ces lignes :

Code : Console

```
# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
xterm-color)
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;32m\]*\s$ '
    ;;
*)
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
    ;;
esac

# Comment in the above and uncomment this below for a color prompt
# PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;32m\]*\s$ '

```

Dans les commentaires, on vous dit que vous pouvez activer l'invite de commande colorée en commentant les lignes du dessus et en décommentant la dernière ligne.

Rajoutez donc un `#` devant les 2 premiers `PS1`, et enlevez le `#` devant le dernier `PS1` pour que la coloration de l'invite de commande puisse fonctionner :

Code : Console

```
# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
xterm-color)
#    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;32m\]*\s$ '
    ;;
*)
#    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
    ;;
esac

# Comment in the above and uncomment this below for a color prompt
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;32m\]*\s$ '

```

Enregistrez.

Il faudra ouvrir une nouvelle console pour que la modification soit prise en compte. Admirez 😊

```
mateo21@mateo21-desktop:~$ cd /var/log/  
mateo21@mateo21-desktop:/var/log$ █
```

Si vous êtes en forme, vous pouvez vous éditer la ligne que vous venez de décommenter :

Code : Console

```
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;3
```

C'est en fait elle qui indique ce que l'invite de commande doit afficher. Les séquences de type "\033" servent à paramétrer la couleur (c'est pas simple, je vous l'accorde 😊).

Le symbole \u au milieu indique le nom de l'utilisateur (mateo21 par exemple) et \h indique le nom de la machine hôte (mateo21-desktop). Vous pouvez repérer dans cette ligne le @ qui sépare les deux, le ".", le "\$", etc.

Vous pouvez essayer de personnaliser un peu ces éléments et leur ordre si ça vous amuse, mais faites quand même attention à ne pas mettre le bordel là-dedans hein 😊

Créer des alias

Les alias sont des commandes que vous créez qui sont automatiquement transformées en d'autres commandes. Descendez un peu plus bas dans le fichier, vous trouverez des lignes commentées commençant par "alias".

Je vous invite à les personnaliser comme moi pour commencer :

Code : Console

```
# enable color support of ls and also add handy aliases  
if [ "$TERM" != "dumb" ]; then  
    eval "`dircolors -b`"  
    alias ls='ls --color=auto'  
    #alias dir='ls --color=auto --format=vertical'
```

```
#alias vdir='ls --color=auto --format=long'
fi

# some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'
```

Vous avez déjà probablement un alias créé :

Code : Console

```
alias ls='ls --color=auto'
```

Celui-ci active la coloration des résultats d'un `ls` à chaque fois que vous tapez `ls`. En fait, `ls` est transformé par la console en "`ls --color=auto`" automatiquement à chaque fois. C'est quand même plus rapide que de réécrire ces paramètres tout le temps 🤖

Il y a un autre alias que j'ai l'habitude d'utiliser, c'est "`ll`" (deux fois la lettre `L` minuscule). Cela permet de faire un `ls` en mode détaillé.

Personnellement, j'ai un peu complété l'alias pour utiliser plus d'options à la fois, comme j'en ai parlé dans le chapitre sur `ls` :

Code : Console

```
alias ll='ls -lArth'
```

... signifie que la commande "`ll`" fera appel à `ls` avec les options qui permettent d'afficher le détail de chaque fichier, d'afficher les fichiers cachés, d'afficher les fichiers dans l'ordre inverse de dernière modification (le fichier le plus récent sera en bas) et d'afficher des tailles de fichiers lisibles pour un humain (`-h`).

La commande `ls` appellera automatiquement l'alias "`ls --color=auto`", ce qui fait qu'un "`ll`" sera aussi coloré. Bref, c'est un peu un alias en chaîne 😊



Si vous tapez la commande "`alias`" dans la console, vous verrez la liste de tous les alias définis pour votre utilisateur.

Vous pouvez définir vos propres alias vous aussi, comme vous pouvez le voir c'est très simple. Cela fonctionne sur le modèle :

Code : Console

```
alias nom='commande'
```

Attention à ne pas mettre d'espace autour du symbole "`=`".

On peut par exemple en profiter pour sécuriser un peu nos `rm` pour éviter qu'on puisse supprimer tout le système depuis la racine `/`. Il y a en effet un paramètre de sécurité disponible avec `rm` : `--preserve-root`. Mais ce serait un peu long de l'écrire à chaque fois, et surtout on risquerait d'oublier. En définissant un alias sur `rm`, vous ne pourrez pas oublier comme ça :

Code : Console

```
alias rm='rm --preserve-root'
```



Ne testez pas l'efficacité de cette commande en faisant un `rm -rf /` en root ! En effet, il faut relancer une console pour que les modifications soient prises en compte, et si vous avez fait une faute de frappe dans votre alias vous ne serez pas protégé... mais pendant ce temps votre système sera détruit lui ! Bref, même pour "vérifier", ne vous amusez pas à balancer le `rm` de la mort...

Edition du `bashrc` global

Si vous voulez définir des alias ou modifier l'invite de commandes pour tous vos utilisateurs, vous pouvez le faire en une seule fois en éditant le fichier `bashrc` global situé dans `/etc/bash.bashrc`
Ce `bashrc` doit être édité en root.

Ce fichier propose un peu moins d'exemples commentés que celui présent dans votre home. Vous pouvez y copier vos alias et la ligne définissant l'invite de commandes (commençant par `PS1`).



Les éléments du `bashrc` personnel ont la priorité sur ceux du `bashrc` global. Si un même alias est défini dans les deux, c'est celui du `bashrc` personnel qui sera pris en compte.

Et aussi... le `.profile`

De même qu'il existe un `~/bashrc` et un `/etc/bash.bashrc`, il existe un `~/profile` et `/etc/profile`. Quelle est la différence ?

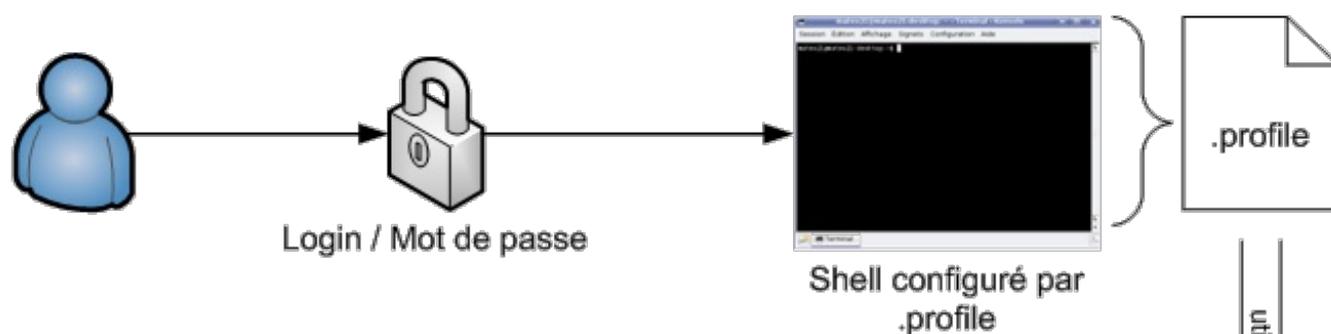
En gros, le **.profile** est lu à chaque nouvelle console dans laquelle vous vous loggez (vous rentrez votre login / mot de passe). C'est le cas des consoles que vous ouvrez avec `Ctrl + Alt + F1` à `F6` (`tty1` à `tty6`).

Le **.bashrc** est lu lorsque vous ouvrez une console dans laquelle vous ne vous loggez pas. C'est le cas des consoles que vous ouvrez en mode graphique (Terminal sous Gnome, Konsole sous KDE).

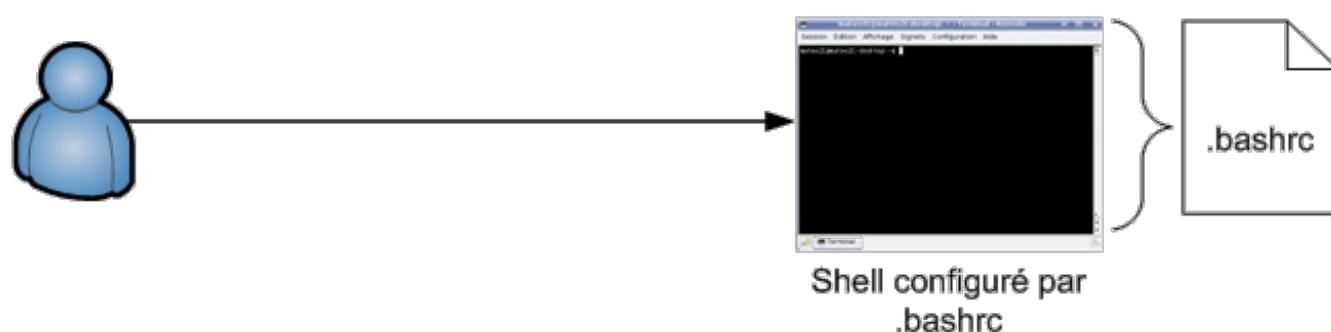
C'est un peu compliqué pour pas grand chose au final. Dans la pratique, le `.profile` fait par défaut appel au `.bashrc`... Donc il suffit d'éditer votre `.bashrc`, et vous modifierez ainsi les options de toutes vos consoles : celles avec et sans login. Voilà pourquoi je vous ai fait éditer dès le début le `.bashrc` 😊

Résumons ça dans un schéma pour bien comprendre comment ça fonctionne :

Shell avec login :



Shell sans login :



Le shell est le programme qui interprète les commandes que vous tapez (vous pouvez considérer que c'est un synonyme de "console").

On a, sur ce schéma, 2 types de shells possibles :

- Soit on a ouvert un shell qui demande un mot de passe, et dans ce cas c'est le .profile qui est lu pour la configuration.
- Soit on a ouvert un shell qui ne demande pas de mot de passe (c'est le cas quand on ouvre une console en mode graphique en général), et dans ce cas c'est le .bashrc qui servira à la configuration.

La particularité, comme le montre le schéma, c'est que le .profile fait appel au .bashrc... ce qui signifie que vous pouvez faire toutes vos configurations dans le .bashrc pour qu'elles soient valables quel que soit le type de shell que vous ouvrez 😊

Eh ben voilà, vous maîtrisez maintenant nano ! Avouez que c'était pas bien dur camarades 🤪

Nano est vraiment un tout petit programme et on en fait vite le tour. Il n'en sera pas de même pour "vi" qui est vraiment riche en fonctionnalités, mais qu'on verra bien plus tard (nous ne sommes pas pressés, et nous avons beaucoup d'autres choses à découvrir avant).

On aura en particulier profité de ce chapitre pour éditer nos premiers fichiers de configuration : .nanorc et .bashrc. Vous serez très probablement amenés à éditer beaucoup d'autres fichiers de configuration à l'avenir. Vous verrez que le format de ces fichiers est en général semblable mais il peut y avoir parfois quelques différences de l'un à l'autre : attention donc à la syntaxe.

Bon ben... Quand vous aurez fini de faire joujou avec les alias et les couleurs, on pourra passer au chapitre suivant 🤪

Installer des programmes avec apt-get

Quand vous êtes sous Windows et que vous voulez télécharger un nouveau programme, que faites-vous ?

En général une petite recherche Google, un tour sur les sites de téléchargement comme telecharger.com, clubic.com... et vous trouvez votre bonheur. Vous récupérez un programme d'installation, vous faites "Suivant", "Suivant", "Suivant", "Terminer" et c'est installé. Parfois il faut répondre à des questions un peu techniques comme "Dans quel répertoire voulez-vous installer ce programme ?".

Sous Linux (et notamment sous Ubuntu), ça ne fonctionne pas du tout comme ça. C'est encore plus simple. Mieux : vous allez vite vous rendre compte que c'est un vrai plaisir d'installer de nouveaux programmes et que c'est même un des points forts d'un système comme Ubuntu par rapport à Windows.

Vous allez voir 😊

Les paquets et leurs dépendances

Tout d'abord, il faut savoir que ce dont je vais vous parler ici concerne uniquement les distributions Linux basées sur Debian (Ubuntu en fait partie).

En effet, l'installation de programmes fonctionne différemment d'une distribution à une autre. C'est justement une des différences majeures entre les distributions.

Des programmes livrés sous forme de paquets

Sous Windows, vous connaissez ce qu'on appelle des "Programmes d'installation". En général il s'agit d'un .exe à lancer qui s'exécute et extrait les fichiers du programme dans un dossier "Program Files".



Exemple : le programme d'installation du jeu Trackmania Nations sous Windows

Sous Ubuntu, on n'a pas de programmes d'installation. On a ce qu'on appelle des **paquets**.

Un paquet est une sorte de dossier zippé qui contient tous les fichiers du programme. Il se présente sous la forme d'un fichier .deb, en référence à DEBian. Il contient toutes les instructions nécessaires pour installer le programme.



Mais alors... Un paquet .deb c'est un peu comme un programme d'installation .exe sous Windows non ?

Ca y ressemble, mais ça fonctionne en fait très différemment. Je citerai 2 différences notables :

- Il y a une gestion des **dépendances** du programme.
- On n'a pas besoin de faire une recherche Google pour trouver un .deb. Tous les .deb sont rassemblés au même endroit sur un même serveur appelé **dépôt** (repository).

Ces deux points méritent plus d'explications.

Nous allons dans un premier temps voir ce que sont les dépendances. Nous traiterons ensuite les dépôts.

Les dépendances, un cauchemar ?

Il est très rare qu'un programme puisse fonctionner seul sous Linux. Très souvent, il utilise d'autres programmes, ou d'autres "bouts de programmes" appelés **bibliothèques**. On dit que les programmes dépendent d'autres programmes pour fonctionner : ils ont des **dépendances**.

Par exemple, le programme de dessin The GIMP (équivalent de Photoshop) ne peut pas fonctionner seul. Il dépend de bibliothèques de lecture des images (qui lui disent comment lire une image JPEG) par exemple. Et parfois, ces dépendances ont elles-mêmes des dépendances !



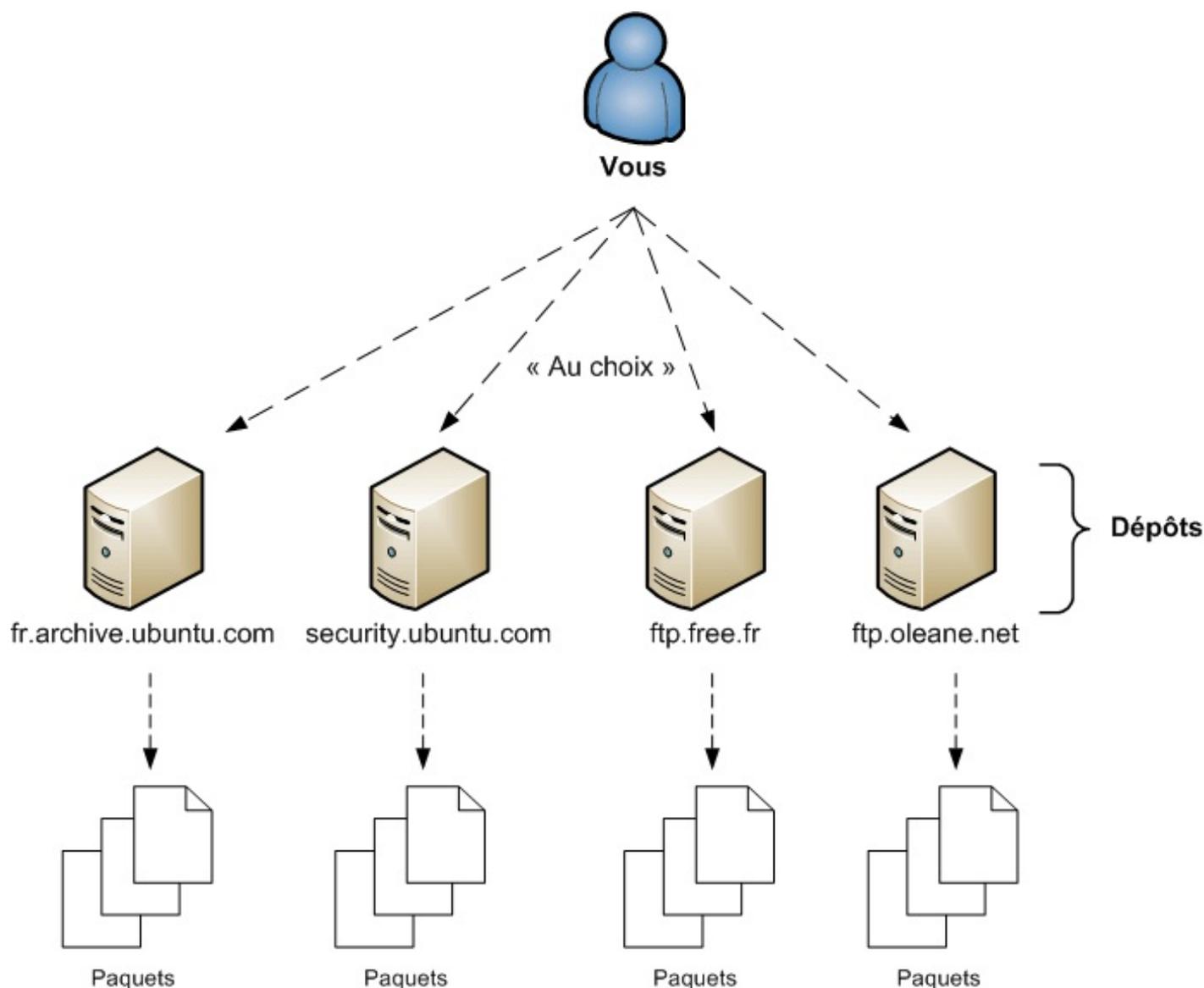
Vous avez certainement déjà rencontré un peu ce problème sous Windows. Par exemple, quand vous installez un jeu, on

Bonne remarque, vous avez tout à fait raison. On peut certes mettre un gros serveur avec une grosse bande passante (qui permet à plusieurs centaines de personnes de télécharger en même temps), mais on peut difficilement imaginer que tous les Linuxiens de la planète aillent se servir au même endroit en même temps !

C'est pour cela qu'il existe en fait un grand nombre de dépôts. La plupart des dépôts proposent exactement les mêmes paquets (les dépôts sont donc une copie les uns des autres). Certains dépôts spéciaux proposent toutefois des programmes que l'on ne trouve nulle part ailleurs. Il est cependant rare que l'on ait besoin de se servir dans ces dépôts spéciaux.

 En règle générale avoir un seul "bon" dépôt suffit à installer la plupart des programmes qui existent sous Linux! 😊

Schématiquement, je représenterais les dépôts comme ceci :



C'est donc à vous de choisir le dépôt que vous voulez utiliser. Chacun de ces dépôts est identique, peu importe celui que vous choisissez, vous devriez retrouver les mêmes paquets.

Comme vous allez probablement beaucoup télécharger depuis votre dépôt, il est conseillé de choisir un serveur qui soit proche de chez vous (chez lequel vous téléchargez suffisamment vite).

Par défaut en France, Ubuntu est configuré pour travailler avec le dépôt `fr.archive.ubuntu.com`. Ce n'est pas toujours une bonne idée de garder le dépôt par défaut, car en cas de nouvelle version d'Ubuntu et de ses logiciels, celui-ci est surchargé et devient alors très lent.

Si vous êtes chez Free, je vous recommande d'utiliser le dépôt de Free.

Si vous êtes chez Wanadoo / Orange, je vous recommande d'utiliser le dépôt Oleane (appartenant à Orange).

Nous allons voir comment changer de dépôt 😊

Gérer ses dépôts

Par défaut, quand vous installez Ubuntu, celui-ci utilise les dépôts officiels d'Ubuntu. Seulement, comme je vous l'ai dit plus tôt, ces serveurs risquent d'être souvent encombrés.

Il n'y a aucun mal à utiliser les dépôts officiels, mais il peut être bien de savoir en changer. Justement, la liste des dépôts que votre ordinateur utilise est stockée dans un fichier. Pour éditer ce fichier, il faut utiliser un éditeur de texte comme... nano, que l'on a vu au chapitre précédent justement (j'avais tout calculé qu'est-ce que vous croyez 🤔).

C'est le moment de mettre en pratique ce que vous avez appris au chapitre précédent. Le fichier à ouvrir qui contient la liste des dépôts que vous utilisez est :

```
/etc/apt/sources.list
```

Ce fichier ne peut être modifié que par root, l'administrateur de la machine. Pour que vous puissiez modifier ce fichier, il faut donc passer root.

Pour cela, vous avez 2 possibilités. Soit vous faites un sudo juste avant :

Code : Console

```
sudo nano /etc/apt/sources.list
```

... et vous modifierez le fichier en tant que root.

Soit vous passez root "définitivement" en faisant `sudo su` d'abord.

Le fichier que vous avez devrait ressembler à cela :

Code : Autre

```
# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.

deb http://fr.archive.ubuntu.com/ubuntu/ hardy main restricted
deb-src http://fr.archive.ubuntu.com/ubuntu/ hardy main restricted

## Major bug fix updates produced after the final release of the
## distribution.
deb http://fr.archive.ubuntu.com/ubuntu/ hardy-updates main restricted
deb-src http://fr.archive.ubuntu.com/ubuntu/ hardy-updates main restricted

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## universe WILL NOT receive any review or updates from the Ubuntu security
## team.
deb http://fr.archive.ubuntu.com/ubuntu/ hardy universe
deb-src http://fr.archive.ubuntu.com/ubuntu/ hardy universe

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## multiverse WILL NOT receive any review or updates from the Ubuntu
## security team.
```

```
deb http://fr.archive.ubuntu.com/ubuntu/ hardy multiverse
deb-src http://fr.archive.ubuntu.com/ubuntu/ hardy multiverse

## Uncomment the following two lines to add software from the 'backports'
## repository.
## N.B. software from this repository may not have been tested as
## extensively as that contained in the main release, although it includes
## newer versions of some applications which may provide useful features.
## Also, please note that software in backports WILL NOT receive any review
## or updates from the Ubuntu security team.
# deb http://fr.archive.ubuntu.com/ubuntu/ hardy-
backports main restricted universe multiverse
# deb-src http://fr.archive.ubuntu.com/ubuntu/ hardy-
backports main restricted universe multiverse

deb http://security.ubuntu.com/ubuntu hardy-security main restricted
deb-src http://security.ubuntu.com/ubuntu hardy-security main restricted
deb http://security.ubuntu.com/ubuntu hardy-security universe
deb-src http://security.ubuntu.com/ubuntu hardy-security universe
deb http://security.ubuntu.com/ubuntu hardy-security multiverse
deb-src http://security.ubuntu.com/ubuntu hardy-security multiverse
```



Ne copiez-collez pas ce code dans votre fichier `sources.list`, il est différent en fonction de la version de la distribution utilisée.

Les lignes commençant par un # sont des lignes de commentaires. Elles seront ignorées.
Normalement, chaque ligne du fichier commence par une de ces 2 directives :

- **deb** : pour télécharger la version compilée (binaire) des programmes. C'est ce que vous voudrez faire dans la plupart des cas, car c'est la version "prête à l'emploi".
- **deb-src** : permet de récupérer le code source du programme. Généralement, vous n'en avez pas besoin, sauf si vous êtes curieux et que vous voulez voir la source d'un programme (c'est l'avantage du logiciel libre de pouvoir voir la source des programmes !).

A priori, seules les lignes `deb` nous intéressent, on pourrait même supprimer (ou commenter) les lignes `deb-src` car récupérer les sources n'a pas d'intérêt pour nous.

Voici une ligne "type" :

```
deb http://fr.archive.ubuntu.com/ubuntu/ hardy universe
```

En premier paramètre, on a l'adresse du dépôt. Ici, <http://fr.archive.ubuntu.com/ubuntu/> est le dépôt par défaut français. Ensuite, on a le nom de la version de la distribution qu'on utilise, "hardy" dans mon cas mais votre version est peut-être plus récente 😊

Enfin, le dernier paramètre (et tous les paramètres suivants s'il y en a) correspondent à la "section" du dépôt dans laquelle vous voulez regarder.

Ouf, c'est un peu compliqué tout ça.

En fait, la seule chose que vous devriez avoir à faire, c'est remplacer toutes les adresses (`http...`) par celle du nouveau dépôt que vous voulez utiliser.



Mais comment je connais l'adresse des autres dépôts qui existent ?

Bonne question... à laquelle je ne peux pas répondre 🤔

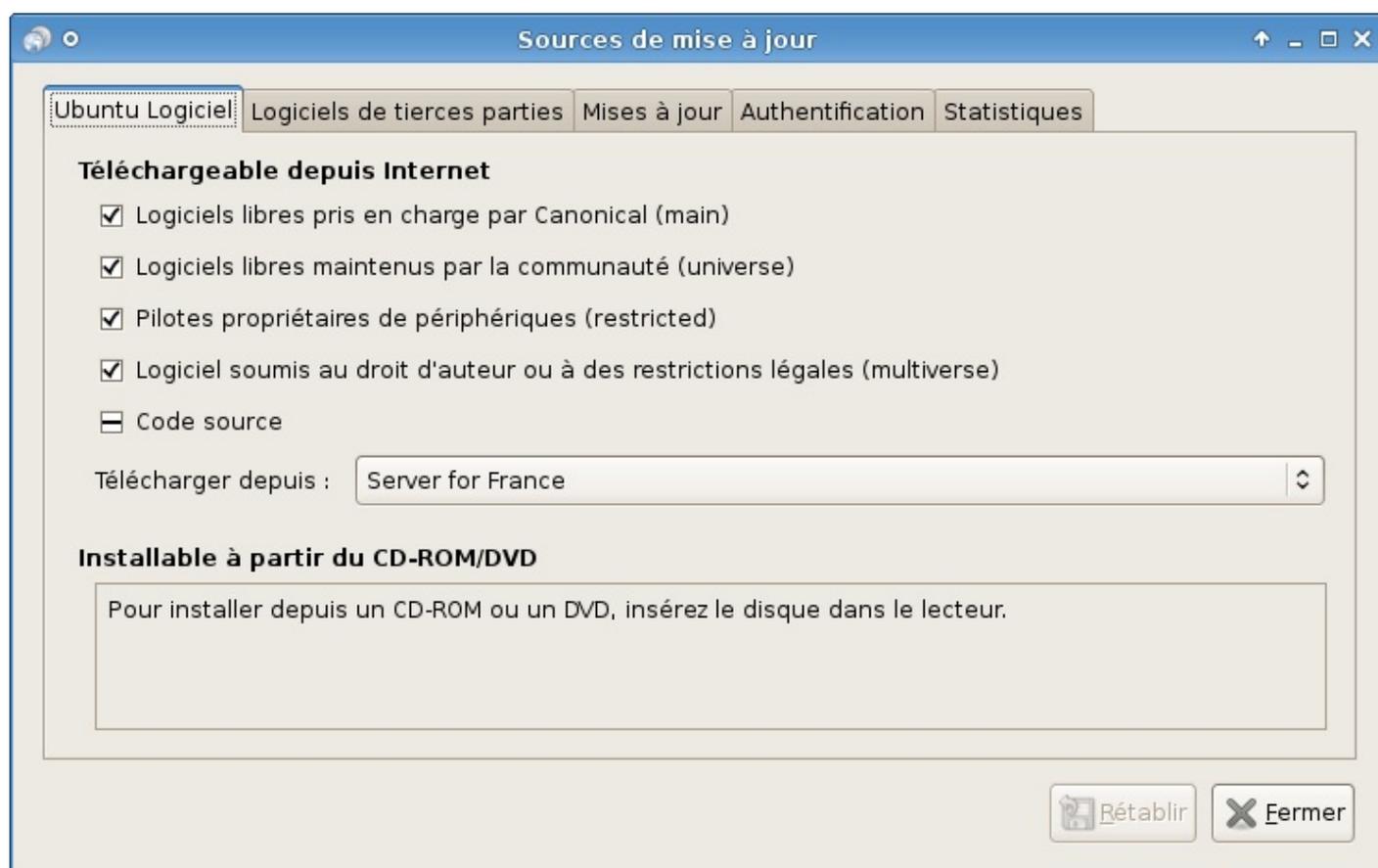
A priori tout le monde peut créer un dépôt, donc il peut très bien exister des dizaines voire des centaines de dépôts différents que vous pourriez utiliser. Je ne les connais pas tous et je ne vais pas me risquer à faire une liste ici, cependant si vous recherchez "dépôts ubuntu" sous Google vous devriez trouver des réponses.

Utiliser l'outil graphique

Le plus simple je pense est de passer par l'outil graphique fourni par Ubuntu. L'outil en question dépend de votre Ubuntu. Si vous avez :

- **Ubuntu** : allez dans Système > Administration > Sources de logiciels
- **KUbuntu** : allez dans Menu K > Système > Gestionnaire Adept > Adept > Gérer les dépôts
- **XUbuntu** : allez dans Applications > Système > Sources de mises à jour

Voici la fenêtre par exemple sous Xubuntu :



Notez la liste déroulante "Télécharger depuis : Server for France", qui signifie que vous utilisez les dépôts français officiels d'Ubuntu.

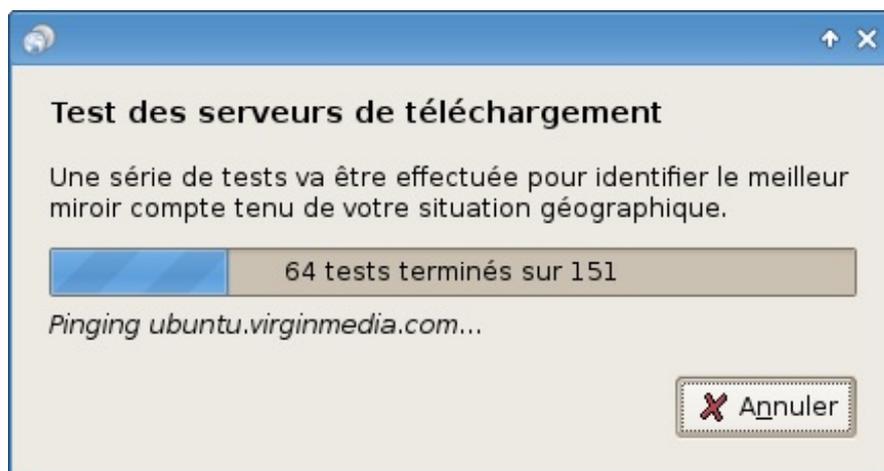
Cette liste vous offre le choix pour le moment entre "Main server" (le serveur américain officiel) et le serveur français. Cliquez sur "Autre..."



Cette fenêtre recense les dépôts les plus connus regroupés par pays. Certains sont fournis par des universités (u-picardie.fr), d'autres par des FAI (Free, Olean pour Orange...).

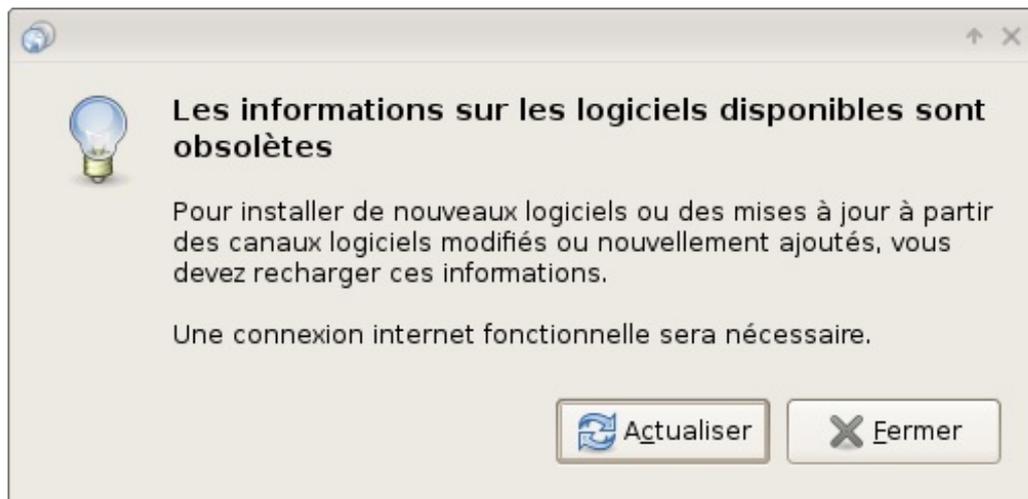


Pour savoir lequel est le plus rapide, cliquez sur le bouton "Sélectionne le meilleur serveur (le plus proche)". Une fenêtre va s'ouvrir et tester la vitesse de tous les serveurs :



Le serveur qui sera sélectionné à la fin sera le serveur que le logiciel aura détecté comme étant le plus rapide pour vous. Cliquez sur "Choisir un serveur" pour valider.

Ensuite, le logiciel vous fera remarquer que votre cache n'est pas à jour. En effet, pour des raisons de performances, Ubuntu avait téléchargé la liste des paquets proposés par l'ancien serveur. Comme vous avez changé de serveur, il se peut que la liste des paquets change. Il faut récupérer la liste des paquets proposés par le nouveau serveur et la mettre en cache. Cliquez sur "Actualiser" pour actualiser la liste des logiciels disponibles :



Et voilà ! 😊

L'outil graphique est plus pratique pour mettre à jour votre fichier sources.list, il faut bien le reconnaître.

Maintenant que nous avons choisi notre dépôt, nous sommes prêts à télécharger à gogo ! 😊

Les outils de gestion des paquets

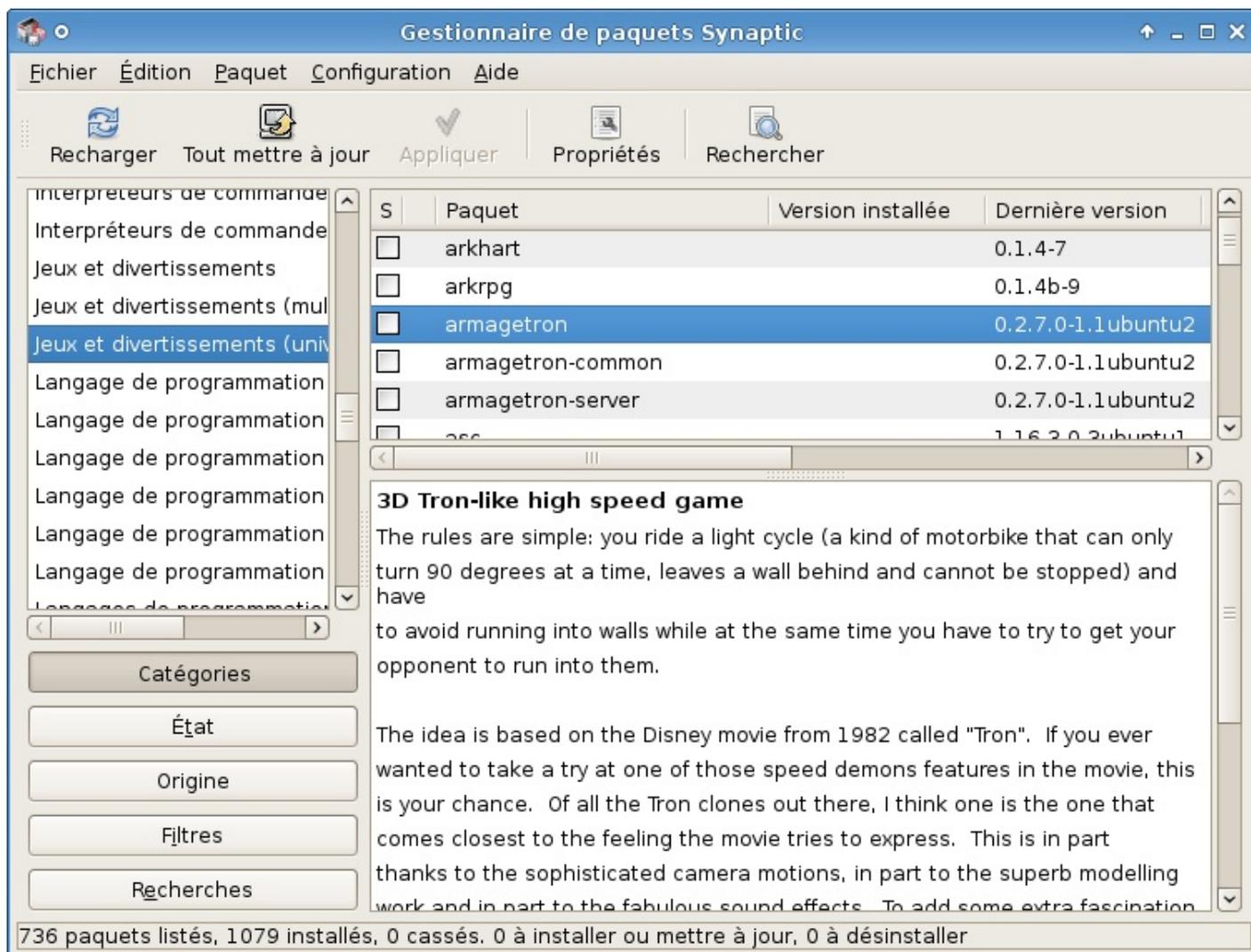
Résumons.

Nous avons découvert 3 nouveaux termes jusqu'ici :

- **paquet** : c'est un programme "prêt à l'emploi", en quelque sorte l'équivalent des programmes d'installation sous Windows.
- **dépendance** : un paquet peut avoir besoin de plusieurs autres paquets pour fonctionner, on dit qu'il a des dépendances.
- **dépôt** : c'est le serveur sur lequel on va télécharger nos paquets.

Je vous ai proposé de changer de dépôt mais sachez que ce n'est pas obligatoire, vous pouvez vous contenter du dépôt par défaut.

Sous Ubuntu, on utilise un programme qui gère les paquets pour nous. Il existe des programmes graphiques, comme Synaptic :



Ce programme est assez simple d'utilisation : vous cochez les programmes que vous voulez installer, et décochez ceux que vous voulez désinstaller. Une fois que vous avez fait votre choix, cliquez sur "Appliquer" pour télécharger tous les programmes que vous avez coché et pour désinstaller tous ceux que vous avez décoché.

Bon, Synaptic c'est bien beau, mais nous nous intéressons ici aux programmes console. Les 2 programmes console les plus connus de gestion des paquets sont :

- apt-get
- aptitude

Lequel des deux utiliser ?

Le premier est sûrement le plus célèbre, le second est généralement reconnu comme étant plus efficace lors de la désinstallation de paquets (il supprime aussi les dépendances inutilisées). Cependant, apt-get sous Ubuntu a évolué aussi et peut supprimer les dépendances inutilisées.

Après, c'est un peu une question d'habitude. Pour ma part, j'ai toujours été habitué à apt-get, c'est donc celui que j'utilise et que je vais vous montrer. Que vous utilisiez l'un ou l'autre ne fera pas beaucoup de différence.

Nous devons généralement suivre 3 étapes pour télécharger un paquet :

- **apt-get update** (optionnel) : pour mettre notre cache à jour si ce n'est pas déjà fait.
- **apt-cache search monpaquet** (optionnel) : pour rechercher le paquet que nous voulons télécharger si nous ne connaissons pas son nom exact.
- **apt-get install monpaquet** : pour télécharger et installer notre paquet.

Vous allez voir ça va être très simple 😊

Nous verrons aussi comment supprimer un paquet et comment mettre tous nos paquets à jour en une seule commande !

apt-get update : mettre à jour le cache des paquets

Commençons par la mise à jour du cache des paquets (`apt-get update`).

Comme nous l'avons vu précédemment, il est nécessaire de mettre à jour son cache des paquets. Cela correspond à télécharger la nouvelle liste des paquets proposés par le dépôt.

Toutefois, il n'est pas nécessaire de mettre à jour son cache à chaque fois que l'on veut télécharger un paquet.



Ah bon ? Quand est-ce que je sais si je dois mettre à jour mon cache alors ?

Il y a 2 cas où vous avez besoin de mettre à jour votre cache :

- Quand vous changez/ ajoutez un dépôt à votre liste de dépôts.
- Si vous n'avez pas mis à jour votre cache depuis un moment (quelques semaines).

Pour mettre à jour votre cache, tapez dans la console (en tant que root) :

Code : Console

```
apt-get update
```



Pensez à rajouter un `sudo` si vous n'êtes pas déjà root.

Après avoir tapé cette commande, vous allez automatiquement télécharger la dernière liste des paquets proposés par vos dépôts :

Code : Console

```
root@mateo21-desktop:~# apt-get update
Réception de : 1 http://wine.budgetdedicated.com hardy Release.gpg [191B]
Ign http://wine.budgetdedicated.com hardy/main Translation-fr
Atteint http://wine.budgetdedicated.com hardy Release
Atteint ftp://ftp.free.fr hardy Release.gpg
Ign http://wine.budgetdedicated.com hardy/main Packages
Atteint ftp://ftp.free.fr hardy/restricted Translation-fr
Atteint http://wine.budgetdedicated.com hardy/main Sources
Atteint ftp://ftp.free.fr hardy/main Translation-fr
Atteint http://wine.budgetdedicated.com hardy/main Packages
Atteint ftp://ftp.free.fr hardy/universe Translation-fr
Atteint ftp://ftp.free.fr hardy/multiverse Translation-fr
Atteint ftp://ftp.free.fr hardy-updates Release.gpg
Réception de : 2 ftp://ftp.free.fr hardy-updates/restricted Translation-fr
Ign ftp://ftp.free.fr hardy-updates/restricted Translation-fr
Réception de : 3 ftp://ftp.free.fr hardy-updates/main Translation-fr
Ign ftp://ftp.free.fr hardy-updates/main Translation-fr
Réception de : 4 ftp://ftp.free.fr hardy-security Release.gpg [191B]
Réception de : 5 ftp://ftp.free.fr hardy-security/restricted Translation-fr
Ign ftp://ftp.free.fr hardy-security/restricted Translation-fr
Réception de : 6 ftp://ftp.free.fr hardy-security/main Translation-fr
Ign ftp://ftp.free.fr hardy-security/main Translation-fr
Réception de : 7 ftp://ftp.free.fr hardy-security/universe Translation-fr
Ign ftp://ftp.free.fr hardy-security/universe Translation-fr
Réception de : 8 ftp://ftp.free.fr hardy-security/multiverse Translation-fr
Ign ftp://ftp.free.fr hardy-security/multiverse Translation-fr
Réception de : 9 ftp://ftp.free.fr hardy Release [57,2kB]
Réception de : 10 ftp://ftp.free.fr hardy-updates Release [50,9kB]
Réception de : 11 ftp://ftp.free.fr hardy-security Release [50,9kB]
```

```
Atteint ftp://ftp.free.fr hardy/restricted Packages
Atteint ftp://ftp.free.fr hardy/main Packages
Atteint ftp://ftp.free.fr hardy/restricted Sources
Atteint ftp://ftp.free.fr hardy/universe Packages
Atteint ftp://ftp.free.fr hardy/universe Sources
Atteint ftp://ftp.free.fr hardy/multiverse Packages
Atteint ftp://ftp.free.fr hardy/multiverse Sources
Atteint ftp://ftp.free.fr hardy-updates/restricted Packages
Atteint ftp://ftp.free.fr hardy-updates/main Packages
Atteint ftp://ftp.free.fr hardy-updates/restricted Sources
Réception de : 12 ftp://ftp.free.fr hardy-security/restricted Packages [5990B]
Réception de : 13 ftp://ftp.free.fr hardy-security/main Packages [120kB]
Réception de : 14 ftp://ftp.free.fr hardy-security/restricted Sources [956B]
Réception de : 15 ftp://ftp.free.fr hardy-security/universe Packages [78,6kB]
Réception de : 16 ftp://ftp.free.fr hardy-security/universe Sources [11,8kB]
Réception de : 17 ftp://ftp.free.fr hardy-security/multiverse Packages [5395B]
Réception de : 18 ftp://ftp.free.fr hardy-security/multiverse Sources [1042B]
382ko réceptionnés en 7s (50,1ko/s)
Lecture des listes de paquets... Fait
```

Voilà, ça fait un peu peur la première fois mais en général c'est assez rapide 😊

Je vous rappelle que vous n'avez pas besoin d'exécuter cette commande à chaque fois que vous voulez installer un paquet, mais seulement de temps en temps pour être sûr d'avoir la liste la plus à jour.

apt-cache search : rechercher un paquet

A moins que vous ne connaissiez déjà le nom exact du paquet que vous voulez, il va falloir faire une petite recherche. On utilise pour cela la commande suivante :

Code : Console

```
apt-cache search votrerecherche
```

Cette commande effectue une recherche de paquet dans votre cache. Cela évite d'avoir à aller sur internet pour faire la recherche, ce qui aurait été lent.

Bon, que veut-on télécharger ? Un éditeur de texte ? Un navigateur ?

Allez, pour s'amuser je vous propose de rechercher un jeu, par exemple un jeu de casse-briques (breakout en anglais).

Faites la recherche suivante :

Code : Console

```
root@mateo21-desktop:~# apt-cache search breakout
briquolo - Fast paced 3d Breakout
briquolo-data - Fast paced 3d Breakout data files
circuslinux - The clowns are trying to pop balloons to score points!
circuslinux-data - data files for circuslinux
gnome-breakout - Clone of the classic game Breakout, written for GNOME
lbreakout2 - A ball-and-paddle game with nice graphics
lbreakout2-data - A ball-and-paddle game with nice graphics (DATA FILES)
libfreebob0 - FreeBoB API
libfreebob0-dev - FreeBoB API - development files
tecnoballz - breaking block game ported from the Amiga platform
```

La commande `apt-cache search breakout` a listé tous les paquets qui avaient un rapport avec les casse-briques. A gauche vous avez le nom du paquet, et à droite une courte description.



Si vous voulez une plus ample description d'un paquet, utilisez `apt-cache show nomdupaquet`. Exemple :
`apt-cache show lbreakout2`

apt-get install : installer un paquet

Pour ma part j'aime beaucoup le jeu lbreakout2.

Je vous propose donc de le télécharger. Ca se fait très simplement (**en tant que root toujours**, rajoutez un sudo devant la commande si vous n'êtes pas déjà root) :

Code : Console

```
apt-get install lbreakout2
```

La commande apt-get install attend que vous lui donniez le nom du paquet à installer.



Astuce : vous pouvez installer plusieurs paquets d'un coup en les listant un à un :
`apt-get install paquet1 paquet2 paquet3`

Essayons donc d'installer lbreakout2 :

Code : Console

```
root@mateo21-desktop:~# apt-get install lbreakout2
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture de l'information d'état... Fait
Les paquets supplémentaires suivants seront installés :
  lbreakout2-data libSDL-mixer1.2 libsmpeg0
Les NOUVEAUX paquets suivants seront installés :
  lbreakout2 lbreakout2-data libSDL-mixer1.2 libsmpeg0
0 mis à jour, 4 nouvellement installés, 0 à enlever et 153 non mis à jour.
Il est nécessaire de prendre 2943ko dans les archives.
Après dépaquetage, 5358ko d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer [O/n] ? O
Réception de : 1 ftp://ftp.free.fr feisty/main libsmpeg0 0.4.5+cvs20030824-
1.9build1 [105kB]
Réception de : 2 ftp://ftp.free.fr feisty/main libSDL-mixer1.2 1.2.6-
1.1build1 [145kB]
Réception de : 3 ftp://ftp.free.fr feisty/universe lbreakout2-data 2.5.2-
2.1ubuntu1 [2444kB]
Réception de : 4 ftp://ftp.free.fr feisty/universe lbreakout2 2.5.2-
2.1ubuntu1 [249kB]
2943ko réceptionnés en 6s (484ko/s)
Sélection du paquet libsmpeg0 précédemment désélectionné.
(Lecture de la base de données... 123350 fichiers et répertoires déjà installés.)
Dépaquetage de libsmpeg0 (à partir de ../libsmpeg0_0.4.5+cvs20030824-
1.9build1_amd64.deb) ...
Sélection du paquet libSDL-mixer1.2 précédemment désélectionné.
Dépaquetage de libSDL-mixer1.2 (à partir de ../libSDL-mixer1.2_1.2.6-
1.1build1_amd64.deb) ...
Sélection du paquet lbreakout2-data précédemment désélectionné.
Dépaquetage de lbreakout2-data (à partir de ../lbreakout2-data_2.5.2-
2.1ubuntu1_all.deb) ...
Sélection du paquet lbreakout2 précédemment désélectionné.
Dépaquetage de lbreakout2 (à partir de ../lbreakout2_2.5.2-
2.1ubuntu1_amd64.deb) ...
Paramétrage de libsmpeg0 (0.4.5+cvs20030824-1.9build1) ...

Paramétrage de libSDL-mixer1.2 (1.2.6-1.1build1) ...

Paramétrage de lbreakout2-data (2.5.2-2.1ubuntu1) ...
Paramétrage de lbreakout2 (2.5.2-2.1ubuntu1) ...
```

Si vous obtenez le message d'erreur suivant :



```
E: Impossible d'ouvrir le fichier verrou /var/lib/dpkg/lock - open (13
Permission non accordée)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you root?
... c'est que vous n'avez pas exécuté la commande en tant que root. Pensez à passer root avant d'exécuter cette
commande, en utilisant sudo.
```

Il s'en est passé des choses dites-donc ! 😬

Comme vous pouvez le voir, au début apt-get a vérifié si le paquet existait et les dépendances dont il avait besoin. Il se trouve que lbreakout2 avait besoin de dépendances qu'on n'a pas, comme libsdlmixer-1.2 par exemple. C'est pour cela qu'on nous a demandé confirmation au milieu :

```
Souhaitez-vous continuer [O/n] ?
```

Répondez par un "O" majuscule (comme "Oui") et tapez Entrée pour que l'installation se poursuive.

C'est alors que la magie d'apt-get opère : le programme va aller télécharger tout seul le paquet sur le dépôt, ainsi que toutes les dépendances dont il a besoin et que nous n'avons pas.

Puis, il "dépaquete" les fichiers qui étaient contenus à l'intérieur du paquet, les installe et effectue les paramètres tout seul.

Ce qui est fort là-dedans, c'est qu'apt-get ne vous demande rien ! Il installe tout ce qu'il faut tout seul, dans les bons répertoires, et crée même le raccourci pour lancer le jeu dans la section "Jeux" !

Vérifiez par vous-mêmes. Le menu des applications d'Ubuntu comporte désormais une section Jeux (qu'il a créée si elle n'existait pas) et possède un raccourci vers le jeu que nous venons d'installer.



C'est bon, vous pouvez jouer ! 😊



Le jeu libre lbreakout2

Recommencez l'opération autant de fois que vous voulez, tous les paquets que vous devriez voir sont des logiciels libres que vous pouvez télécharger à volonté.

Ci-dessous, OpenArena, basé sur le célèbre jeu Quake III Arena dont le code source est devenu libre :



Le jeu libre openarena

Pour l'obtenir, c'est très simple :

Code : Console

```
apt-get install openarena
```

C'est aussi simple que ça 😊

Notez que vous devrez avoir installé les pilotes de votre carte graphique pour pouvoir y jouer.

apt-get autoremove : supprimer un paquet

Si vous voulez désinstaller un paquet, vous pouvez utiliser la commande apt-get remove :

Code : Console

```
apt-get remove lbreakout2
```

Le paquet sera alors désinstallé de votre ordinateur.

Toutefois, cela ne supprime pas les dépendances du paquet devenues inutiles. Pour demander à apt-get de supprimer *aussi* les dépendances inutiles, on utilise autoremove :

Code : Console

```
apt-get autoremove lbreakout2
```

Faisons un autoremove, comme ça nous serons sûrs de libérer un maximum d'espace disque.

Code : Console

```
root@mateo21-desktop:~# apt-get autoremove lbreakout2
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture de l'information d'état... Fait
Les paquets suivants ont été automatiquement installés mais ne sont plus nécessaire
  libSDL-mixer1.2 libsmpeg0
Les paquets suivants seront ENLEVÉS :
  lbreakout2 lbreakout2-data libSDL-mixer1.2 libsmpeg0
0 mis à jour, 0 nouvellement installés, 4 à enlever et 0 non mis à jour.
Il est nécessaire de prendre 0o dans les archives.
Après dépaquetage, 5358ko d'espace disque seront libérés.
Souhaitez-vous continuer [O/n] ?
```

apt-get analyse le paquet, vérifie les dépendances qu'il peut supprimer sans risque (c'est-à-dire les dépendances qui ne sont plus utilisées par aucun autre paquet) et calcule la taille d'espace disque libérée (un peu plus de 5 Mo dans mon cas).

Faites "O" pour confirmer que vous voulez bien désinstaller le paquet et ses dépendances. Vous voyez alors les paquets se faire désinstaller par apt-get :

Code : Console

```
Suppression de lbreakout2 ...
Suppression de lbreakout2-data ...
Suppression de libSDL-mixer1.2 ...
Suppression de libsmpeg0 ...
```

C'est fini, le paquet et ses dépendances sont proprement désinstallés 😊

apt-get upgrade : mettre à jour tous les paquets

Une autre fonctionnalité particulièrement géniale d'apt-get est sa capacité à mettre à jour tous les paquets installés sur votre système d'un seul coup. Le programme ira chercher les nouvelles versions de tous vos programmes et les mettra à jour s'il y a une nouvelle version de disponible :

Code : Console

```
apt-get upgrade
```



Pensez à faire un `apt-get update` pour mettre à jour votre cache des paquets avant de lancer un upgrade. En effet, apt-get compare la version de vos paquets installés avec ceux présents dans le cache. Si votre cache est "ancien", apt-get se dira "Oh bah, y'a rien de nouveau, pas besoin de mise à jour". Veillez donc à faire régulièrement un `apt-get update` pour être 100% sûr que votre cache est à jour.

On vous demandera une confirmation après avoir listé tous les paquets qui ont besoin d'une mise à jour. Vous n'avez pas besoin de faire autre chose. Tous vos paquets installés seront mis à jour (ça peut être un peu long par contre 😊).

Si tous vos paquets sont déjà à jour, vous verrez le message suivant :

Code : Console

```
root@mateo21-desktop:~# apt-get upgrade
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture de l'information d'état... Fait
```

```
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
```

Tous ces "0" de la dernière ligne nous indique que rien ne s'est passé car il n'y avait rien à faire. Cela signifie que votre système est à jour.



Il est conseillé de faire un `apt-get upgrade` régulièrement pour avoir le système le plus à jour possible. Cela vous permet de bénéficier des dernières fonctionnalités des logiciels, mais cela corrige aussi les failles de sécurité qui auraient pu être découvertes dans les programmes (et on en trouve tous les jours, même chez les logiciels libres !).

L'installation de programmes sous Ubuntu pourrait se résumer à une seule commande console très simple :

Code : Console

```
apt-get install paquet
```

Je vous rappelle que seul root a le pouvoir d'installer de nouveaux programmes sur la machine. Vous devez donc passer root auparavant à l'aide de `sudo`.

Les autres gestionnaires de paquets

La commande `apt-get` n'existe que sous Debian et ses dérivés, dont Ubuntu fait partie. Les autres distributions possèdent en général leur propre système de gestion des paquets. Citons :

- **rpm** : le système de gestion de paquets utilisé par la distribution Red Hat, qui reste très utilisé, mais qui ne gère malheureusement pas les dépendances (pour l'avoir utilisé pendant un temps, je peux vous dire que c'est galère 😞).
- **yum** : une surcouche de rpm gérant les dépendances, utilisé par la distribution Fedora.
- **urpmi** : une surcouche de rpm gérant les dépendances, utilisé par la distribution Mandriva.
- **emerge** : le gestionnaire de paquets de Gentoo, qui compile toujours à partir des sources (il ne télécharge jamais le programme binaire directement).

Parfois, il se peut que vous ayez besoin d'un programme que votre gestionnaire de paquets ne trouve pas. Si même en rajoutant des dépôts vous ne le trouvez toujours pas, il vous restera la solution ultime : recompiler le programme à partir du code source. C'est en général ce que vous devez faire pour certains programmes encore peu connus. L'installation de programmes à partir des sources fera l'objet d'un chapitre ou d'une annexe plus tard.

Bonus track

Le mot "moo" veut dire "meuh" en anglais. Essayez de taper la commande `apt-get moo` :

Code : Console

```
mateo21@mateo21-desktop:~$ apt-get moo
      (  _ )
      (oo)
 /-----\ /
/ |       ||
* / \----/ \
  ~~     ~~
...."Have you mooded today?"...
```

C'est un message caché dans la commande apt-get qui... est censé représenter une vache en console.
Sûrement un des programmeurs d'apt-get qui devait s'ennuyer 🐮

RTFM : lisez le manuel !

Quand on vient de Windows, on n'a pas trop l'habitude de lire des documentations. Parfois les logiciels sont livrés avec des modes d'emploi, mais honnêtement, qui ici prend la peine de les lire ? 🤔

Sous Linux, lire la documentation doit devenir un **réflexe**. En effet, bien que cela fasse un peu peur au premier abord, la documentation est vraiment le meilleur endroit pour en savoir plus sur les commandes que vous utilisez.

Les tutoriels que j'écris ne pourront jamais rivaliser avec la documentation. Je peux vous montrer les commandes et les paramètres qui me semblent les plus utiles, mais pour connaître certains paramètres dont vous avez besoin moins souvent, vous n'y couperez pas : vous aurez besoin de lire la doc.

Ce chapitre est justement là pour "démystifier" la doc et vous apprendre à la lire. C'est peut-être un des chapitres les plus importants du cours, car si vous savez lire la doc, vous êtes capables d'apprendre tout ce dont vous avez besoin, et donc vous pourrez tout faire !



Le titre de ce chapitre est une insulte "amicale" entre Linuxiens. C'est en général ce qu'on dit à un débutant qui pose trop de questions sur les forums, alors que la réponse était dans la doc.

"RTFM" est une abréviation qui signifie "Read The Fucking Manual", ce qui se traduirait en français par quelque chose "Lis ce p*** de manuel". Mais oui, c'est amical ! Puisque je vous le dis... 🤔

man : afficher le manuel d'une commande

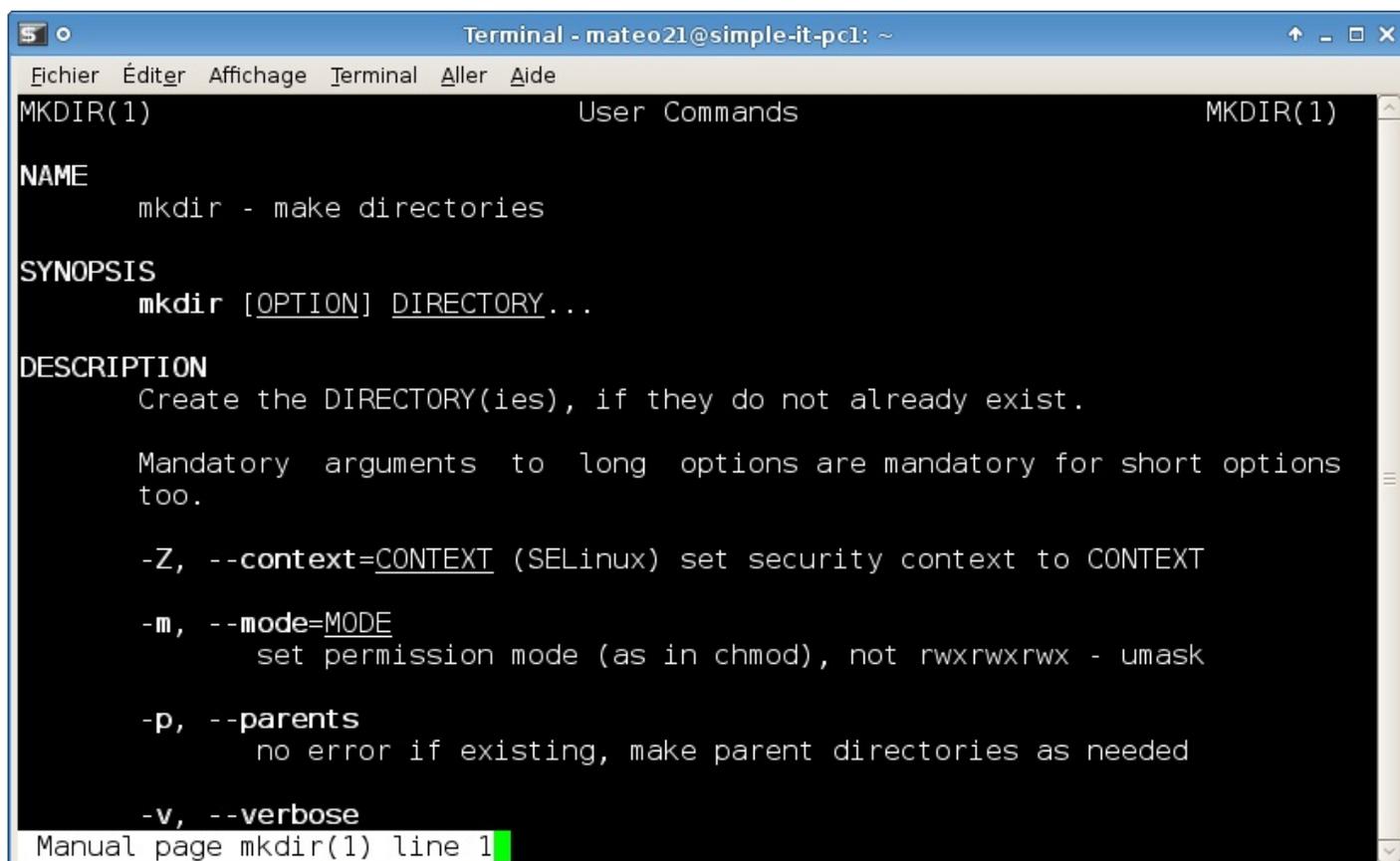
La commande magique que nous allons expérimenter durant tout ce chapitre est man, qui est l'abréviation de MANual.

La commande man s'utilise très simplement : elle prend en paramètre le nom de la commande dont vous voulez lire la doc. Par exemple, si je veux tout savoir sur la commande `mkdir`, je dois écrire :

Code : Console

```
man mkdir
```

Votre console devrait alors ressembler à ceci :



```
Terminal - mateo21@simple-it-pc1: ~
Fichier Éditer Affichage Terminal Aller Aide
MKDIR(1) User Commands MKDIR(1)
NAME
  mkdir - make directories
SYNOPSIS
  mkdir [OPTION] DIRECTORY...
DESCRIPTION
  Create the DIRECTORY(ies), if they do not already exist.

  Mandatory arguments to long options are mandatory for short options
  too.

  -Z, --context=CONTEXT (SELinux) set security context to CONTEXT
  -m, --mode=MODE
      set permission mode (as in chmod), not rwxrwxrwx - umask
  -p, --parents
      no error if existing, make parent directories as needed
  -v, --verbose
Manual page mkdir(1) line 1
```

Il s'agit de la documentation de la commande `mkdir`. Là-dedans, il y a tout ce qu'il faut savoir sur `mkdir`. La doc de la commande étant généralement un peu longue, celle-ci s'affiche page par page, à la manière de `less` qu'on a vue dans un chapitre précédent.

Se déplacer dans le manuel

Quelques commandes à connaître pour se déplacer dans le man :

- Utilisez les **touches fléchées** du clavier (vers le haut et vers le bas) pour vous déplacer ligne par ligne.
- Vous pouvez utiliser les touches "**Page Up**" et "**Page Down**" (ou **Espace**) pour vous déplacer de page en page.
- Appuyez sur la touche **Home** (aussi appelée **Origine**) pour revenir au début du manuel, et sur "**Fin**" pour aller à la fin.
- Appuyez sur la touche / (slash) pour effectuer une recherche. C'est très pratique ! Tapez ensuite le mot que vous recherchez dans le manuel, puis appuyez sur Entrée. Si la recherche renvoie un résultat, vous serez automatiquement placé sur le premier résultat trouvé. Pour passer au résultat suivant, retapez / puis Entrée directement (sans retaper votre recherche).
- Appuyez sur la touche **Q** pour quitter le manuel à tout moment, comme vous le faisiez avec `less`.



Si vous avez ouvert votre manuel dans une console graphique comme moi, vous pouvez aussi utiliser la molette de la souris !

Les principales sections du manuel

Comme vous pouvez le voir, le manuel de la commande est découpé en plusieurs sections (leur nom est écrit en gras et est aligné à gauche de l'écran).

Voici leur signification :

- **NAME** : le nom de la commande dont vous êtes en train d'afficher le manuel, ainsi qu'une courte description de ce à quoi elle sert.
- **SYNOPSIS** : c'est la liste de toutes les façons d'utiliser la commande. Nous y reviendrons un peu plus loin car il est vital de comprendre ce qui est écrit dans cette section.
- **DESCRIPTION** : une description plus approfondie de ce que fait la commande. On y trouve aussi la liste des paramètres et leur signification. C'est en général la section la plus longue.
- **AUTHOR** : l'auteur du programme. Parfois, il y a de nombreux auteurs, c'est souvent le cas d'ailleurs avec le logiciel libre.
- **REPORTING BUGS** : si vous rencontrez un bug dans le logiciel, on vous donne l'adresse de la personne à contacter pour rapporter le bug.
- **COPYRIGHT** : le copyright, c'est-à-dire la licence d'utilisation de la commande. La plupart des programmes que vous utilisez sont certainement des programmes open-source sous licence GPL, ce qui vous donne le droit de voir la source et de redistribuer le programme librement.
- **SEE ALSO** : cette section vous propose de "voir aussi" d'autres commandes en rapport avec celle que vous êtes en train de regarder. C'est une section parfois intéressante.

Pour faire simple, les 3 premières sections sont vraiment les plus importantes (**NAME**, **SYNOPSIS** et **DESCRIPTION**), c'est celles que nous regarderons dans la plupart des cas et c'est aussi pour cela qu'elles sont au début du manuel 😊

La langue des pages de manuel



Eh mais... c'est tout en anglais ! Tout le monde n'est pas un crack en anglais, comment je fais moi si j'ai du mal à lire l'anglais ??? 😞

Mon premier conseil, même si vous n'allez pas l'apprécier, je tiens à le donner car c'est vraiment le meilleur conseil : familiarisez-vous avec l'anglais. Si vous voulez travailler dans l'informatique c'est de toute façon une langue incontournable, il est impossible de l'ignorer ou alors ce serait du pur suicide.

Bon, malgré tout, il y en a peut-être parmi vous qui ne comptent pas forcément travailler dans l'informatique mais qui aimeraient éviter d'avoir à apprendre l'anglais pour se servir de Linux, ce que je peux très bien comprendre. Si l'évocation du mot "anglais" provoque chez vous des éruptions cutanées inexplicables, alors voici une bonne nouvelle : il existe une version française des pages de manuel !

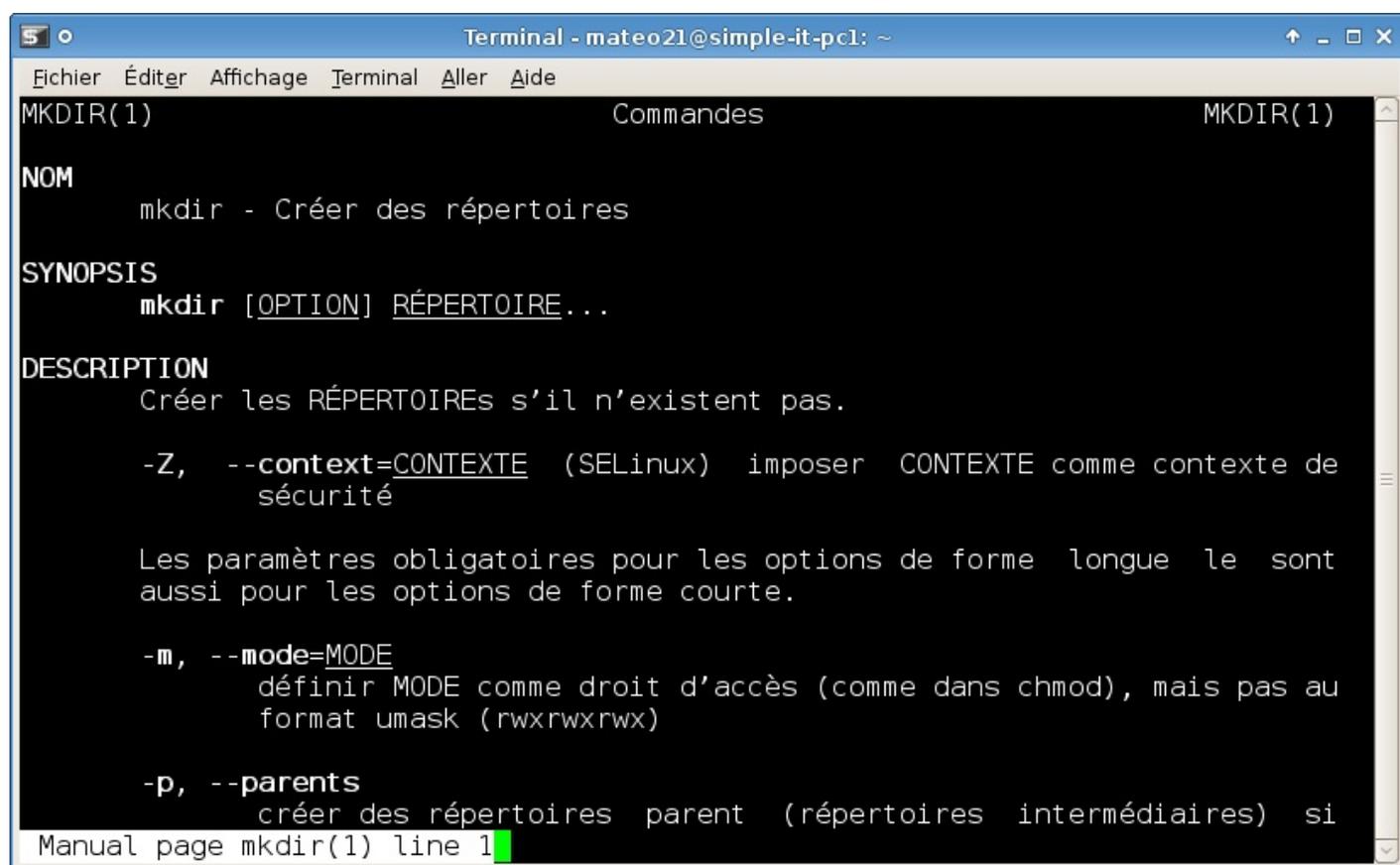
Vous pouvez l'installer grâce à la commande `apt-get` qu'on a justement appris dans le chapitre précédent (rien n'est laissé au hasard, je vous l'ai dit 🤖) :

Code : Console

```
apt-get install manpages-fr
```

Le paquet à installer est `manpages-fr` comme vous pouvez le voir 🤖

Maintenant, si vous tapez `man mkdir`, vous verrez la page s'afficher en français :



```
Terminal - mateo21@simple-it-pc1: ~
Fichier Éditer Affichage Terminal Aller Aide
MKDIR(1)                               Commandes                               MKDIR(1)
NOM
  mkdir - Créer des répertoires
SYNOPSIS
  mkdir [OPTION] RÉPERTOIRE...
DESCRIPTION
  Créer les RÉPERTOIREs s'il n'existent pas.
  -Z, --context=CONTEXTE (SELinux) imposer CONTEXTE comme contexte de
  sécurité
  Les paramètres obligatoires pour les options de forme longue le sont
  aussi pour les options de forme courte.
  -m, --mode=MODE
      définir MODE comme droit d'accès (comme dans chmod), mais pas au
      format umask (rwxrwxrwx)
  -p, --parents
      créer des répertoires parent (répertoires intermédiaires) si
Manual page mkdir(1) line 1
```



Utiliser les pages de manuel en français n'est pas recommandé. En effet, ces pages ne sont parfois pas à jour, certaines ne sont pas traduites, et seule la version anglaise est assurée d'être la plus à jour et de comporter le moins d'erreurs. D'ailleurs en parlant d'erreur cher lecteur, une faute d'orthographe s'est glissée dans la capture d'écran ci-dessus, sauras-tu la retrouver ? 🤖

Dans la suite de ce tutoriel, je considérerai que vous travaillez avec les pages de manuel **en anglais**. C'est vraiment ce que je recommande. Utilisez la version française uniquement si vous ne comprenez rien à l'anglais, sinon faites un petit effort je vous assure que ça vaut le coup 🤖

Je vais donc désinstaller les pages de manuel en français pour revenir à la version anglaise pour la suite de ce chapitre :

Code : Console

```
apt-get autoremove manpages-fr
```

Bien, les présentations étant faites, passons à la suite !

Nous allons apprendre à lire la section SYNOPSIS, une des sections les plus importantes.

Comprendre le synopsis

Le synopsis est une des sections les plus importantes, mais aussi une des plus difficiles à lire.



Quel est le rôle du synopsis ?

Son rôle est de **lister toutes les façons possibles d'utiliser la commande**. En clair, le synopsis vous affiche toutes les combinaisons de paramètres que l'on peut faire avec cette commande.

Certains synopsis sont simples, d'autres sont plus compliqués. Je pense que le mieux est de voir des exemples pour bien comprendre comment ça fonctionne 😊

man mkdir

Commençons par le man de mkdir. La section d'introduction du manuel nous dit : "mkdir - make directories", ce qui signifie que mkdir sert à créer des répertoires.

Synopsis

La section synopsis de mkdir contient ceci :

```
mkdir [OPTION] DIRECTORY...
```

Même si ce synopsis est court, il contient déjà beaucoup d'informations. Il vous dit comment on doit utiliser la commande. Détaillons point par point ce synopsis :

- **mkdir** : pour utiliser la commande mkdir, vous devez commencer par taper mkdir, ça c'est logique 😊
- **[OPTION]** : après mkdir, vous pouvez écrire une option. Dans le synopsis, on met des crochets pour indiquer que c'est facultatif. Vous n'êtes donc pas obligés d'écrire une option.
- **DIRECTORY** : c'est le nom du répertoire à créer. Ce paramètre est obligatoire puisqu'il n'est pas entre crochets. C'est en effet logique : la commande mkdir sert à créer un dossier, la moindre des choses est d'indiquer le nom du dossier à créer ! 😊
- ... : le terme DIRECTORY est suivi de points de suspension. Cela signifie que l'on peut répéter DIRECTORY autant de fois que l'on veut. Traduction : on peut indiquer plusieurs répertoires à la fois pour que la commande les crée tous d'un coup.



Pourquoi **mkdir** est écrit en gras tandis que OPTION et DIRECTORY sont soulignés ?

Les mots du synopsis écrits en gras sont des mots à taper tels quels.

Les mots écrits en souligné doivent être remplacés par le nom approprié.

C'est logique : on doit bel et bien écrire précisément "mkdir", par contre on ne doit pas écrire "DIRECTORY" mais le nom du répertoire. La présence d'un paramètre souligné signifie donc : "*Remplacez le mot souligné par un mot qui convient dans votre cas*".

Exemples d'utilisation

D'après le synopsis, on doit au minimum écrire le nom du dossier. Par exemple :

Code : Console

```
mkdir images
```

Comme on l'a vu, les points de suspension après DIRECTORY nous indiquent qu'on peut répéter le nom du répertoire autant de fois que l'on veut, ce qui nous permet d'en créer plusieurs d'un coup.

On peut donc aussi utiliser mkdir comme ceci :

Code : Console

```
mkdir images videos musiques
```

... ce qui aura pour effet de créer 3 dossiers : "images", "videos" et "musiques".

Maintenant, on peut aussi préciser des options facultatives. Ces options sont listées dans la section DESCRIPTION du man juste un peu plus bas :

Code : Console

```
DESCRIPTION
  Create the DIRECTORY(ies), if they do not already exist.

  Mandatory arguments to long options are mandatory for short options
  too.

  -m, --mode=MODE
        set file mode (as in chmod), not a=rwx - umask

  -p, --parents
        no error if existing, make parent directories as needed

  -v, --verbose
        print a message for each created directory

  -Z, --context=CTX
        set the SELinux security context of each created directory to
        CTX

  --help display this help and exit

  --version
        output version information and exit
```

Toutes ces options peuvent être utilisées à la place de [OPTION] dans le synopsis.

Par exemple, l'option -v (ou --verbose, c'est pareil mais plus long 🤖) affiche un message après chaque répertoire créé. On peut donc écrire :

Code : Console

```
mkdir -v images videos musiques
```

Résultat :

Code : Console

```
mateo21@mateo21-desktop:~/tests$ mkdir -v images videos musiques
mkdir: création du répertoire `images'
mkdir: création du répertoire `videos'
mkdir: création du répertoire `musiques'
```

La commande nous informe maintenant de ce qu'elle fait. Sans le `-v`, la commande n'affiche rien (on dit qu'elle est silencieuse).

Vous remarquerez d'ailleurs qu'on retrouve l'option `-v` dans beaucoup de commandes. A chaque fois elle a la même signification : elle demande à ce que la commande affiche le détail de ce qu'elle est en train de faire. On dit alors qu'on utilise la commande en mode "verbeux" (bavard), pour bien voir tout ce qu'elle fait.

man cp

Essayons une commande un peu plus complexe : `cp`. Je vous rappelle que cette commande sert à copier des fichiers et des répertoires.

Synopsis

Son synopsis est le suivant :

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
cp [OPTION]... -t DIRECTORY SOURCE...
```

Oulah, ça commence à devenir un peu plus délicat 😬

Pourquoi il y a 3 lignes déjà ? Tout simplement parce qu'on ne pouvait pas exprimer toutes les façons d'utiliser `cp` sur une seule ligne.

- Commençons par la première ligne :
`cp [OPTION]... [-T] SOURCE DEST`
La seule chose qui est obligatoire dans le cas de la première ligne, ce sont les paramètres `SOURCE` (le nom du fichier à copier) et `DEST` (le nom de la copie à créer).
Ces fichiers peuvent être précédés d'une ou plusieurs options (remarquez les points de suspension) ainsi que de l'option `-T`.
- La seconde ligne est un peu différente :
`cp [OPTION]... SOURCE... DIRECTORY`
Cette fois, on peut copier un ou plusieurs fichiers (`SOURCE...`) vers un répertoire (`DIRECTORY`). Tout cela peut encore une fois être précédé d'une ou plusieurs options.
- Enfin, la troisième ligne :
`cp [OPTION]... -t DIRECTORY SOURCE...`
... signifie qu'on peut aussi écrire le répertoire (`DIRECTORY`) dans un premier temps, suivi d'un ou plusieurs fichiers (`SOURCE...`). Attention, vous remarquez que dans ce cas il est obligatoire d'utiliser le paramètre `-t` qui n'est plus entre crochets.

Exemples d'utilisation

Ca fait beaucoup de façons d'utiliser `cp` en fait 😬

Si on se base sur la première ligne, on peut juste écrire :

Code : Console

```
cp photo.jpg photo_copie.jpg
```

... ce qui aura pour effet de créer la copie photo_copie.jpg.

On peut aussi ajouter une ou plusieurs options. Pour connaître toutes les options disponibles, vous devrez lire la section DESCRIPTION. Pour cp, il y a beaucoup de choix comme vous pouvez le voir.

Par exemple, on retrouve notre mode -v (verbeux) qui demande à ce que la commande détaille ce qu'elle fait. On pourrait aussi ajouter -i qui demande confirmation si le fichier de destination existe déjà.

On peut donc faire :

Code : Console

```
cp -vi photo.jpg photo_copie.jpg
```

Dans mon cas, le fichier photo_copie.jpg existait déjà. L'ajout de l'option -i va me demander confirmation pour savoir si je veux bien écraser le fichier. Je peux répondre par "o" ou "n" (pour oui ou non), ou "y" ou "n" (pour yes ou no).

Code : Console

```
mateo21@mateo21-desktop:~$ cp -vi photo.jpg photo_copie.jpg
cp: écraser `photo_copie.jpg'? o
`photo.jpg' -> `photo_copie.jpg'
```

Comme le fichier existait déjà, on m'a demandé confirmation. La dernière ligne est le résultat du mode verbeux qu'on a demandé.

Bien 😊

Tout ça c'était juste pour la première ligne, dans le cas où on veut copier un fichier. Essayons un peu ce que propose la seconde ligne : copier un ou plusieurs fichiers dans un dossier.

Code : Console

```
cp photo.jpg photo_copie.jpg images/
```

On exploite là la seconde façon d'utiliser cp (seconde ligne du synopsis). On copie 2 fichiers dans le sous-dossier images/.

Bien entendu, comme l'indique le synopsis, on peut là encore utiliser des options comme -v et -i que l'on vient de voir 😊

Mon conseil : vous vous demandez peut-être comment je comprends la signification de chaque mot du synopsis. Par exemple, était-il évident de savoir que SOURCE correspondait au fichier que l'on voulait copier ?

Si je sais tout ça, c'est parce que j'ai lu attentivement le début de la section DESCRIPTION du man. Je vous recommande d'en faire de même quelle que soit la commande que vous êtes en train d'analyser.

Code : Console



DESCRIPTION

```
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
```

Cette phrase est claire, pour peu que l'on comprenne l'anglais 😊

Elle signifie que l'on peut copier le paramètre appelé SOURCE vers DEST, ou encore (cas de la seconde ligne) plusieurs fichiers SOURCE vers un répertoire (DIRECTORY).

man apt-get

Plus joyeux encore : le synopsis de la commande apt-get 😊

```
apt-get [-hvs] [-o=option de configuration] [-c=fichier] {[update] |
[upgrade] | [dselect-upgrade] | [install paquet...] |
[remove paquet...] | [source paquet...] | [build-dep paquet...]
| [check] | [clean] | [autoclean]}
```

La bonne nouvelle cette fois, c'est que les créateurs d'apt-get n'ont pas eu besoin de faire tenir la commande sur 3 lignes : ils ont mis tous les cas d'utilisation possibles sur une seule ligne !

La mauvaise nouvelle c'est que c'est un peu difficile à lire comme ça.

Décortiquons.

La commande apt-get doit commencer par "apt-get" (ce mot est d'ailleurs écrit en gras). Ca c'est logique.

Ensuite vous pouvez utiliser une des options -hvs (vous pouvez utiliser juste -h, mais aussi -hv, -v, -vs, -hvs...). Ces options étant entre crochets, elles sont facultatives.

Pareil ensuite pour -o et -c. Ces options sont facultatives. En revanche, vous remarquerez qu'elles doivent être obligatoirement suivies d'une valeur. Par exemple "-o=option de configuration". Je vous rappelle que le fait que "option de configuration" soit souligné signifie que vous ne devez pas recopier ces mots tels quels dans la console : vous devez les remplacer par une valeur qui convient (lisez la section DESCRIPTION pour en savoir plus sur -o).

La section qui m'intéresse et que je voudrais qu'on analyse plus en détail arrive juste après. Elle commence et se termine par des accolades :

```
{[update] | [upgrade] | [dselect-upgrade] | [install paquet...] | [remove paquet...]
| [source paquet...] | [build-dep paquet...] | [check] | [clean] | [autoclean]}
```

Vous remarquerez qu'à l'intérieur, les mots sont séparés par des barres verticales |. Ces barres verticales signifient "OU". Cela veut dire que vous devez mettre une et une seule option parmi la liste entre accolades.

Parmi ces options possibles, il y en a que vous devez connaître maintenant, comme :

- **update** : met à jour le cache des paquets disponibles sur votre ordinateur.
- **upgrade** : met à jour tous les paquets installés si une nouvelle version est disponible.
- **install paquet...** : installe le ou les paquets demandés. La présence des points de suspension après "paquet" signifie que vous pouvez indiquer plusieurs paquets à installer d'un coup.
- etc.

Il y a bien d'autres mots-clés utilisables. Pour voir la signification de chacun d'eux, je vous invite à lire la section DESCRIPTION du man qui sert précisément à expliquer ça.

Exemples d'utilisation

Le synopsis indique donc qu'on doit choisir une des options entre accolades séparées par des barres verticales.

On peut donc écrire :

Code : Console

```
apt-get install monpaquet
```

Ou encore :

Code : Console

```
apt-get update
```

Ou encore :

Code : Console

```
apt-get autoclean
```

En revanche, on ne peut pas utiliser simultanément deux options séparées par une barre verticale :

Code : Console

```
INTERDIT :  
apt-get update install monpaquet
```

Le synopsis nous avait bien dit : "*utilisez update OU install OU upgrade (OU ...), mais pas deux éléments de cette liste à la fois*".

Souvenez-vous donc que les barres verticales signifient "OU" et tout ira bien 😊

Résumé de la syntaxe du SYNOPSIS

Voici un petit résumé de la syntaxe du synopsis pour vous souvenir de la façon dont chaque élément doit être interprété :

- **gras** : tapez le mot exactement comme indiqué.
- souligne : remplacez le mot souligné par la valeur qui convient dans votre cas.
- [-hvc] : toutes les options -h, -v et -c sont facultatives.
- a|b : vous pouvez écrire l'option a OU b, mais pas les deux à la fois.
- option... : les points de suspension indiquent que l'option peut être répétée autant de fois que vous voulez.

apropos : trouver une commande

Le man suppose que vous connaissez déjà votre commande et que vous voulez en savoir plus. Mais si vous ne connaissez pas la commande, comment faites-vous ? 😊

C'est là que la commande `apropos` intervient. Vous lui donnez en paramètre un mot-clé, et elle va le rechercher dans les descriptions de toutes les pages du manuel.

La commande `apropos` est donc un peu l'inverse de `man` : elle vous permet de retrouver une commande.

Prenons un exemple : vous recherchez une commande (que vous avez installée) en rapport avec le son, parce que vous aimeriez bien savoir comment modifier le volume en console.

Vous pouvez taper :

Code : Console

```
apropos sound
```

... ce qui va rechercher toutes les commandes qui parlent de son (sound) dans leur page de manuel.

Résultat :

Code : Console

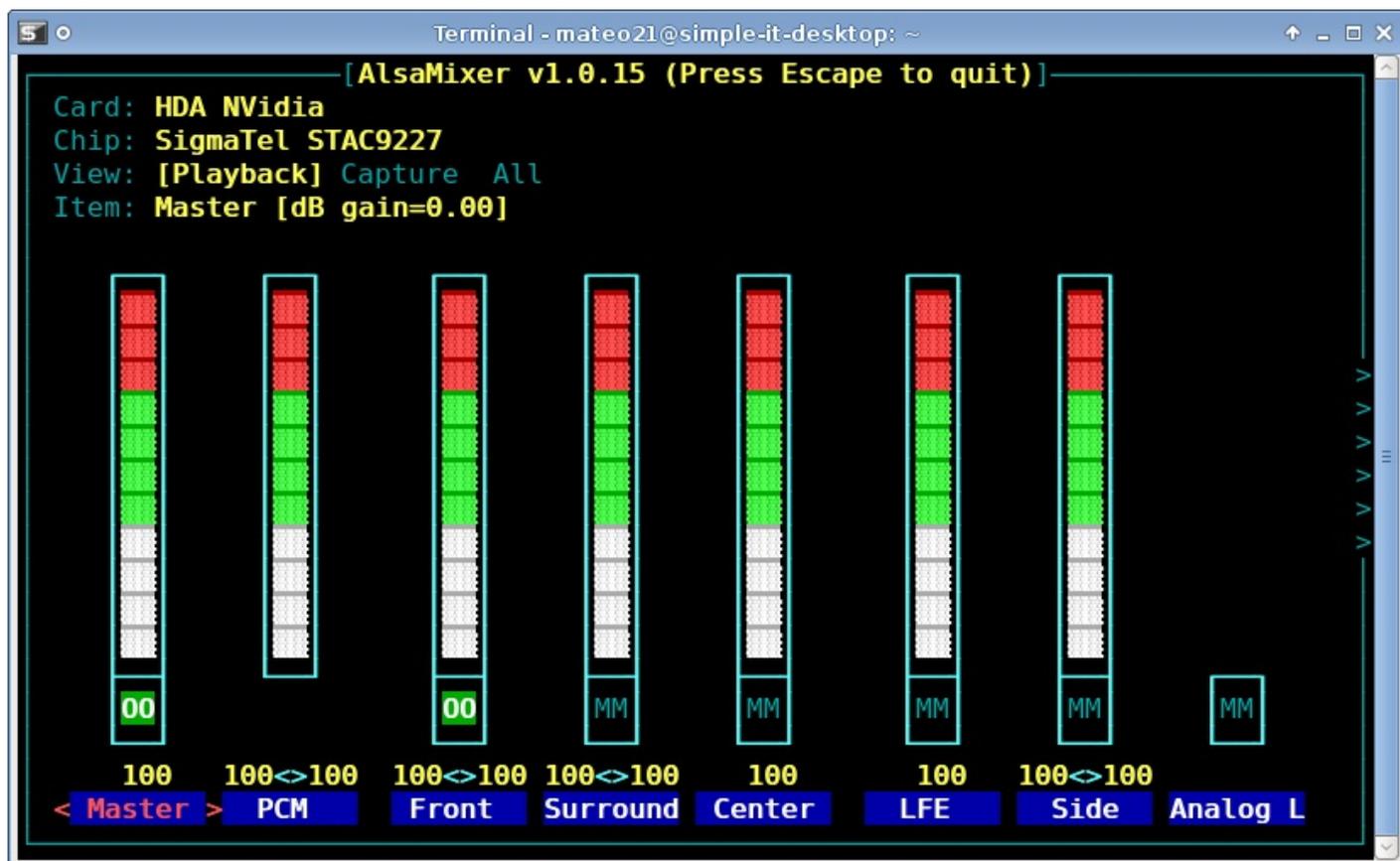
```
mateo21@mateo21-desktop:~$ apropos sound
alsactl (1)          - advanced controls for ALSA soundcard driver
alsamixer (1)       -
  soundcard mixer for ALSA soundcard driver, with ncurses...
amixer (1)          - command-line mixer for ALSA soundcard driver
aplay (1)           - command-
  line sound recorder and player for ALSA soundc...
arecord (1)         - command-
  line sound recorder and player for ALSA soundc...
artscat (1)         - pipe data to sound device
asoundconf (1)     -
  utility to read and change the user's ALSA library con...
```

A gauche la commande, à droite l'extrait de la description courte de la commande dans laquelle apropos a trouvé le mot que vous recherchez.

Il se trouve que je cherchais alsamixer. Et zou ! 😊

Code : Console

```
alsamixer
```



Et si vous voulez en savoir plus sur la commande alsamixer, vous savez maintenant comment faire ! 😊

Code : Console

```
man alsamixer
```

```
Terminal - mateo21@simple-it-desktop: ~
ALSAMIXER(1) ALSAMIXER(1)
NAME
  alsamixer - soundcard mixer for ALSA soundcard driver, with ncurses
  interface
SYNOPSIS
  alsamixer [options]
DESCRIPTION
  alsamixer is an ncurses mixer program for use with the ALSA soundcard
  drivers. It supports multiple soundcards with multiple devices.
OPTIONS
  -h, -help
      Help: show available flags.
  -c <card number or identification>
      Select the soundcard to use, if you have more than one. Cards
      are numbered from 0 (the default).
  -D <device identification>
      Select the mixer device to control.
Manual page alsamixer(1) line 1
```

Son synopsis est ridiculement simple. Pfeuh ! Même pas drôle 🤪

D'autres façons de lire le manuel

Bien que ce soit la technique la plus courante, utiliser `man` et `apropos` n'est pas le seul moyen de vous documenter. Quelles sont les alternatives à `man` ?

Le paramètre `-h` (et `--help`)

Bien que ça ne soit pas une règle, la plupart des commandes acceptent un paramètre `-h` (et parfois son équivalent plus long `--help`) qui provoque l'affichage d'une aide résumée. Parfois, cette aide est d'ailleurs plus facile à lire que celle du `man`, ce qui fait qu'il m'arrive de l'utiliser de temps en temps.

Par exemple :

Code : Console

```
apt-get -h
```

Ce qui nous donne :

Code : Console

```
mateo21@mateo21-desktop:~$ apt-get -h
apt 0.7.9ubuntu15 pour amd64 compilé sur Mar 14 2008 00:00:28
Usage: apt-get [options] command
       apt-get [options] install|remove pkg1 [pkg2 ...]
       apt-get [options] source pkg1 [pkg2 ...]

apt-get is a simple command line interface for downloading and
installing packages. The most frequently used commands are update
and install.
```

Commands:

```

update - Retrieve new lists of packages
upgrade - Perform an upgrade
install - Install new packages (pkg is libc6 not libc6.deb)
remove - Remove packages
autoremove - Remove all automatic unused packages
purge - Remove and purge packages
source - Download source archives
build-dep - Configure build-dependencies for source packages
dist-upgrade - Distribution upgrade, see apt-get(8)
dselect-upgrade - Follow dselect selections
clean - Erase downloaded archive files
autoclean - Erase old downloaded archive files
check - Verify that there are no broken dependencies

```

Options:

```

-h This help text.
-q Loggable output - no progress indicator
-qq No output except for errors
-d Download only - do NOT install or unpack archives
-s No-act. Perform ordering simulation
-y Assume Yes to all queries and do not prompt
-f Attempt to correct a system with broken dependencies in place
-m Attempt to continue if archives are unlocatable
-u Show a list of upgraded packages as well
-b Build the source package after fetching it
-V Show verbose version numbers
-c=? Read this configuration file
-o=? Set an arbitrary configuration option, eg -o dir::cache=/tmp

```

See the apt-get(8), sources.list(5) and apt.conf(5) manual pages for more information and options.

This APT has Super Cow Powers.

Le -h est parfois un bon complément au man si vous n'arrivez pas à comprendre comment utiliser la commande. On y trouve parfois des informations utiles comme ici : "*The most frequently used commands are update and install*", ce qui signifie que l'on utilise le plus souvent apt-get avec les paramètres update et install (et c'est vrai 😊).



Parfois, il n'y a pas de page de manuel pour une commande (man ne fonctionne pas sur cette commande) mais le -h ou le --help fonctionnent. Pensez-y !

La commande whatis

La commande whatis est une sorte de man très allégé. Elle donne juste l'en-tête du manuel pour expliquer en deux mots à quoi sert la commande. Par exemple pour mkdir :

Code : Console

```
whatis mkdir
```

Ca vous permet d'éviter de sortir l'artillerie lourde juste pour savoir à quoi sert la commande 🤖

Code : Console

```
mateo21@mateo21-desktop:~$ whatis mkdir
mkdir (1) - make directories
```

La commande info

La commande info est une alternative à man. Elle est réputée plus puissante mais je la connais trop peu pour vous en parler ici. Je peux donc vous dire qu'elle existe mais je ne vais pas vous faire un chapitre entier dessus 😊

En général le man est bien suffisant.

A noter toutefois que si vous tapez :

Code : Console

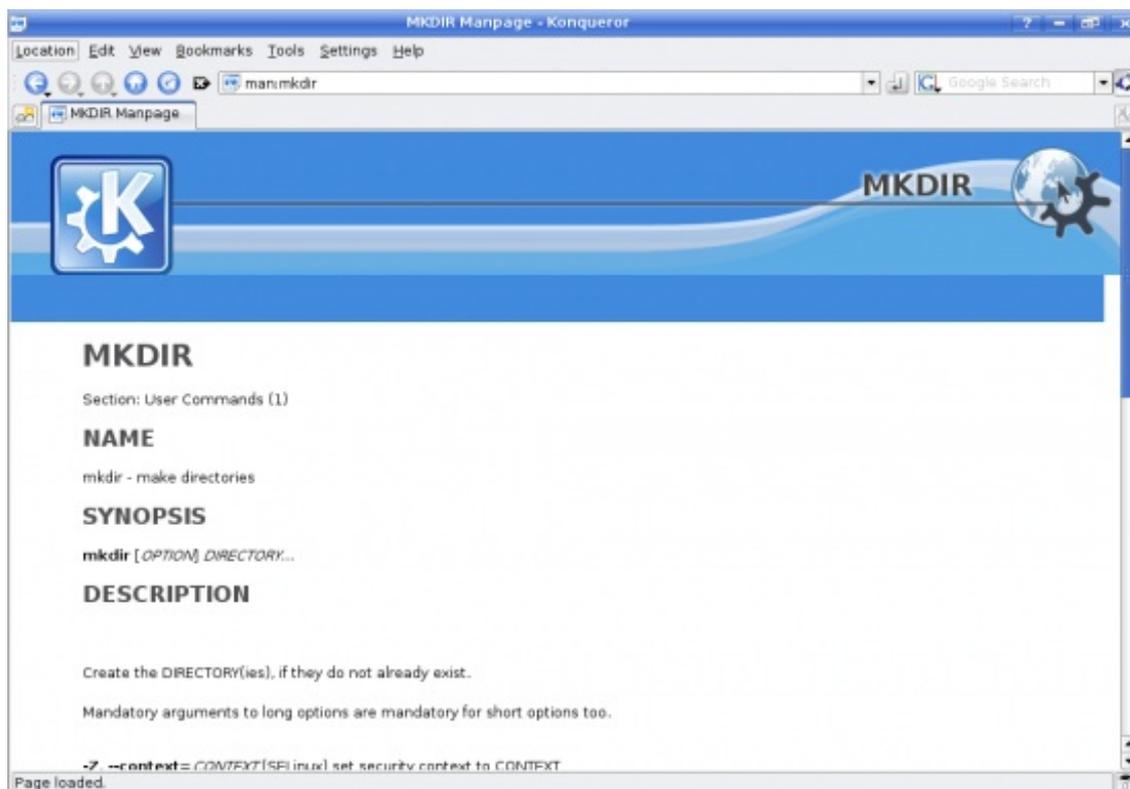
```
info coreutils
```

... cela affichera une sorte de méga-résumé des principales commandes console sous Linux (elles n'y sont pas toutes bien sûr !). Ça vous permet d'avoir une bonne vue d'ensemble 😊

Le manuel sous Konqueror

Ceux qui utilisent KDE, et qui ont donc Konqueror, seront ravis d'apprendre que l'on peut lire le manuel directement depuis Konqueror.

Pour ce faire, tapez dans la barre d'adresse : `man :votrecommande`. Par exemple : `man :mkdir`.



On profite ainsi au passage d'une jolie mise en forme dans le navigateur 😊

Rechercher man sur le Web

Enfin, il est bien de le préciser : on retrouve aussi le man sur le Web !

Si vous devez lire un manuel et que vous n'êtes pas sous Linux à ce moment-là, utilisez tout bêtement un moteur de recherche comme Google pour retrouver la doc.





Par exemple, vous pouvez taper la recherche Google : [man mkdir](#).

Je vous parie que vous trouverez le manuel dans les premiers liens qui s'offrent à vous 😊

Bonne recherche !

La commande `man` en elle-même n'est pas très complexe, mais lire une page de manuel peut faire peur aux débutants au début je le reconnais.

Ce chapitre était justement là pour démystifier la doc et vous encourager à la lire.

Lire la doc doit devenir un réflexe.

Ne pas savoir faire quelque chose sous Linux n'est pas grave. Ne pas chercher la réponse et baisser les bras est par contre beaucoup plus grave.

Les bons utilisateurs de Linux sont justement de grands amateurs de doc. Ils n'ont pas peur d'aller la lire dès qu'ils ne savent pas comment utiliser une commande.



Quoi ? Les pros de Linux ne connaissent pas toutes les commandes par coeur ? 😊

Non, personne ne connaît toutes les commandes en détail. Ça n'existe pas, c'est infaisable, et ce serait de toute façon stupide de tout vouloir apprendre par coeur.

On dit qu'un vrai professionnel connaît en moyenne 90% des commandes qu'il tape. Cela veut dire que pour les 10% restants, il ouvre la doc pour apprendre comment faire ce dont il a besoin.

Comme vous pouvez le voir, lire la doc n'est pas une honte bien au contraire. Pour écrire ce cours de Linux pour débutants, vous n'imaginez pas le nombre de fois que j'ai tapé la commande `man` ! Comme quoi, il n'y a pas de secret.

Allez, on se quitte sur une petite devinette pour terminer ce chapitre : quel est le résultat de cette commande ?

Code : Console

```
man man
```

Ok je sors... 😊

(mais vous pouvez essayer ça marche hein !)

Rechercher des fichiers

Sous Linux, les fichiers sont organisés d'une façon assez particulière. Si vous affichez la liste des répertoires à la racine avec un `ls /`, vous verrez qu'il y a une foule de dossiers aux noms tous plus obscurs les uns que les autres : var, opt, etc, bin, dev...

Une partie de ces répertoires est là pour des raisons historiques, depuis l'époque d'Unix. Et vous savez ce que c'est les vieilles habitudes, on ne change pas ça du jour au lendemain. 😞

Résultat : certes les programmes s'installent tous seuls, mais on ne sait pas forcément toujours où se trouve le fichier dont on a besoin (même si avec l'habitude on finit par trouver assez rapidement).

Heureusement, pas de panique ! On dispose sous Linux d'outils très puissants pour rechercher un fichier sur le disque dur. Certains d'entre eux sont rapides, d'autres plus lents mais aussi plus complets.

Partons à la recherche de ces fichiers ! 🕵️

locate : une recherche rapide

La première façon d'effectuer une recherche que nous allons voir est de loin la plus simple. La commande s'appelle `locate` ("localiser"). Elle est très rapide.

Utiliser locate

Son utilisation est intuitive, il suffit d'indiquer le nom du fichier que vous voulez retrouver. Par exemple :

Code : Console

```
mateo21@mateo21-desktop:~$ locate notes.txt
/home/mateo21/notes.txt
```

La commande a retrouvé notre fichier notes.txt qui était situé dans /home/mateo21.

Essayons maintenant de retrouver ces vieilles photos d'Australie...

Code : Console

```
mateo21@mateo21-desktop:/var/log$ locate australie
/home/mateo21/photos/australie1.jpg
/home/mateo21/photos/australie2.jpg
/home/mateo21/photos/australie3.jpg
```

Locate vous donne tous les résultats qui contiennent le mot "australie" dans leur nom. Que ce soient des fichiers ou des dossiers, elle ne fait pas la différence. Elle vous donne la liste complète des fichiers qu'elle a trouvés.



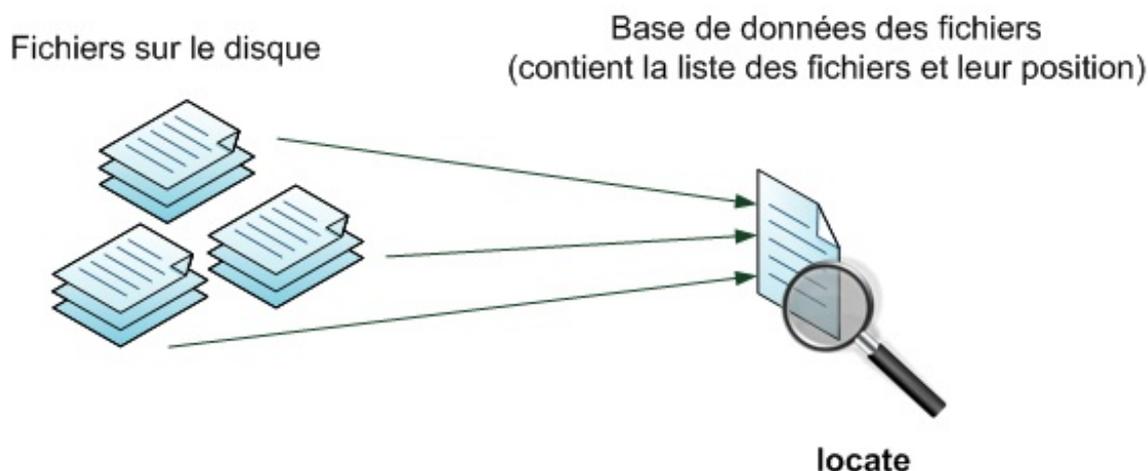
Il existe aussi la commande `slocate` qui est un peu plus récente, mais on ne la retrouve pas sur toutes les distributions par défaut. Vous pouvez toutefois l'installer rapidement avec un apt-get si vous ne l'avez pas. La commande `slocate` est identique à `locate`, à la différence près qu'elle vérifie les droits des fichiers avant de les lister. Avec `slocate`, un utilisateur ne pourra donc pas voir un nom de fichier apparaître s'il n'a pas le droit de le lire, tandis qu'avec un `locate` le nom du fichier serait apparu dans les résultats quand même.

La base de données des fichiers



Je ne comprends pas. Je viens de créer des fichiers (avec la commande `touch` par exemple), et `locate` ne me renvoie aucun résultat. Pourquoi ? 😞

C'est justement le défaut de locate dont je voulais vous parler : la commande ne fait pas la recherche sur votre disque dur entier mais sur une base de données de vos fichiers.



Votre problème, c'est que les fichiers viennent tout juste d'être créés et n'ont pas encore été répertoriés dans la base de données. Ils ne seront donc pas découverts par locate.

Une fois par jour, votre système mettra à jour la base de données. Donc, si vous réessayez demain, il est probable que locate trouve enfin votre fichier.



Mais... je ne vais pas attendre 24h pour retrouver un fichier tout de même !

Non, bien sûr !

Vous pouvez forcer la commande locate à reconstruire la base de données des fichiers du disque dur. Cela se fait avec la commande updatedb, à exécuter en root (avec sudo) :

Code : Console

```
sudo updatedb
```

La mise à jour de la liste des fichiers peut être un peu longue, il faudra patienter. Une fois que c'est fini, réessayez de faire un locate, il devrait maintenant trouver votre fichier.

En résumé, locate est pratique car rapide et facile à utiliser.

Cependant, locate donne parfois trop de résultats car elle recherche dans tous les répertoires du disque dur, elle n'est donc pas très précise. De plus, les fichiers qui viennent tout juste d'être créés ne seront pas découverts, à moins d'exécuter updatedb.

Quand locate ne suffit pas, on a besoin d'une commande plus puissante. On sort l'artillerie lourde : find. 🤖

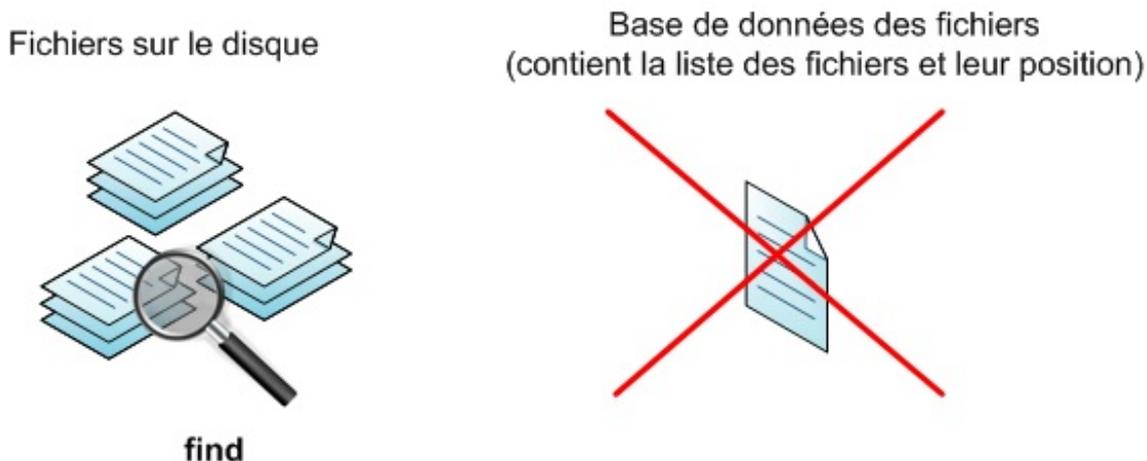
find : une recherche approfondie

`find` est la commande de recherche par excellence pour retrouver des fichiers, mais aussi pour effectuer des opérations sur chacun des fichiers trouvés. Elle est très puissante, permet de faire donc beaucoup de choses, et par conséquent... elle est un peu complexe. 😊

Savoir la manipuler est toutefois indispensable, donc découvrons-la !

find recherche les fichiers actuellement présents

Contrairement à locate, find ne va pas lire dans une base de données mais va au contraire parcourir tout votre disque dur. Cela peut être très long si vous avez plusieurs Go de données !



Avec cette méthode de recherche vous ne risquez pas de rater un fichier récent qui aurait pu être créé aujourd'hui. Et ce n'est pas le seul avantage, loin de là !

Fonctionnement de la commande find

La commande find s'utilise de la façon suivante :

find "où" "quoi" "que faire avec"
(seul le paramètre "quoi" est obligatoire)

- **Où** : c'est le nom du dossier dans lequel la commande va faire la recherche. Tous les sous-dossiers seront analysés. Contrairement à locate, il est donc possible de limiter la recherche à /home par exemple. Par défaut, si ce paramètre n'est pas précisé, la recherche s'effectuera dans le dossier courant et ses sous-dossiers.
- **Quoi** : c'est le fichier à rechercher. On peut rechercher un fichier par son nom, mais aussi en fonction de sa date de dernière création, de sa taille, etc. Ce paramètre est obligatoire.
- **Que faire avec** : il est possible d'effectuer des actions automatiquement sur chacun des fichiers trouvés (on parle de "post-traitement"). L'action la plus courante consiste à afficher simplement la liste des fichiers trouvés, mais nous verrons que nous pouvons faire bien d'autres choses. 😊
Par défaut, la commande find affiche les résultats trouvés et ne fait rien d'autre avec.

Utilisation basique de la commande find

Commençons doucement, et voyons des exemples pratiques. Nous allons tout d'abord rechercher un fichier et afficher sa position.

Recherche à partir du nom

Je vais utiliser dans un premier temps le seul paramètre obligatoire : le nom du fichier à retrouver.

Je me place dans mon répertoire home et je vais essayer de retrouver un fichier appelé "logo.png" que j'ai égaré. Je dois écrire :

Code : Console

```
find -name "logo.png"
```

Le `-name "logo.png"` est un paramètre qui demande à retrouver un fichier qui s'appelle très exactement `logo.png`.

Voici le résultat :

Code : Console

```
mateo21@mateo21-desktop:~$ find -name "logo.png"
/home/mateo21/projet/images/logo.png
```

Si la recherche n'affiche rien, cela signifie qu'aucun fichier n'a été trouvé.



Comme nous n'avons pas précisé où rechercher, `find` a cherché dans le répertoire dans lequel nous nous trouvons et dans ses sous-répertoires. Le `~` de l'invite de commande signifie que j'étais dans mon home, c'est-à-dire dans `/home/mateo21/`. Tous les sous-répertoires ont été analysés.

Maintenant, si je suis dans mon home mais que je veux rechercher dans un autre répertoire, il faudra préciser en premier paramètre le répertoire dans lequel chercher.

Par exemple, si je veux retrouver tous les fichiers qui s'appellent `"syslog"` situés dans `/var/log` (et ses sous-répertoires), je dois écrire :

Code : Console

```
find /var/log/ -name "syslog"
```

Essayons ça !

Code : Console

```
mateo21@mateo21-desktop:~$ find /var/log/ -name "syslog"
/var/log/syslog
/var/log/installer/syslog
```

Les paramètres correspondent à ceux que je vous ai présentés un peu plus tôt :

- **Où** : dans `/var/log` (et les sous-dossiers)
- **Quoi** : le fichier `"syslog"`

Il n'y a pas de paramètre "Que faire avec", donc par défaut `find` choisit d'afficher les fichiers trouvés.

Notez que, contrairement à `locate`, `find` récupère uniquement la liste des fichiers qui s'appellent exactement comme demandé. Ainsi, s'il existe un fichier nommé `"syslog2"`, il ne sera pas listé dans les résultats. Pour qu'il le soit, il faut utiliser le joker : l'étoile `*` !

Exemple :

Code : Console

```
mateo21@mateo21-desktop:~$ find /var/log/ -name "syslog*"
/var/log/syslog.3.gz
/var/log/syslog.5.gz
```

```
/var/log/syslog.4.gz
/var/log/syslog
/var/log/syslog.6.gz
/var/log/syslog.2.gz
/var/log/syslog.1.gz
/var/log/installer/syslog
/var/log/syslog.0
```

Ici, nous avons affiché tous les fichiers qui commencent par syslog.

Si on avait voulu avoir la liste des fichiers qui se terminent par syslog, on aurait écrit `"*syslog"`. De même, si on avait voulu avoir la liste des fichiers qui contiennent syslog, que ce soit au début, au milieu ou à la fin, on aurait écrit `"*syslog*"`.

L'étoile est donc un joker qui signifie *"il peut y avoir n'importe quels caractères à cet endroit"*.



Et si je veux rechercher sur tout le disque dur, et pas seulement dans un dossier ?

Facile, il suffit d'indiquer le répertoire racine /. Je vous rappelle que sous Linux, tous les dossiers sont situés dans un sous-niveau de /. C'est un petit peu l'équivalent du "C:" sous Windows.

Code : Console

```
find / -name "syslog"
```



La recherche depuis la racine prend beaucoup de temps si vous avez de nombreux fichiers. De plus, si vous n'êtes pas root vous aurez de nombreux messages d'erreur vous indiquant que vous n'avez pas le droit d'aller dans certains répertoires.

En général, à moins d'être très patient (ou désespéré), on ne fait pas de recherche depuis la racine.

Recherche à partir de la taille

Vous ne connaissez pas le nom du fichier que vous recherchez ? Pas de panique ! Il y a bien d'autres façons de retrouver des fichiers (ou des dossiers d'ailleurs).

Par exemple, on peut rechercher tous les fichiers qui font plus de 10 Mo.

Code : Console

```
mateo21@mateo21-desktop:/var$ find ~ -size +10M
/home/mateo21/souvenirs.avi
/home/mateo21/backups/backup_mai.gz
/home/mateo21/backups/backup_juin.gz
```



Rappel : le tilde "~" signifie "Rechercher dans mon home", en l'occurrence chez moi "/home/mateo21/".

Au lieu de se baser sur le nom, on se base ici sur la taille (-size). Le "+10M" indique que l'on recherche des fichiers de plus de 10Mo. On peut aussi utiliser "k" pour les Ko, "G" pour les Go, etc.

Vous pouvez aussi utiliser un moins "-" à la place du "+" pour avoir par exemple les fichiers de moins de 10 Mo. Et si vous

enlevez le "+", la commande cherchera des fichiers de 10 Mo exactement (ni plus, ni moins).

Recherche à partir de la date de dernier accès

Vous êtes sûr d'avoir accédé à votre rapport au format .odt il y a moins de 7 jours, mais vous n'arrivez pas à le retrouver ?

Avec -atime, vous pouvez indiquer le nombre de jours qui vous séparent du dernier accès à un fichier.

Code : Console

```
mateo21@mateo21-desktop:~$ find -name "*.odt" -atime -7  
/home/mateo21/ecriture/resume_infos_juin.odt
```

J'ai ici combiné une recherche avec le nom et une recherche avec la date. Si je ne me souvenais même plus de l'extension du fichier, j'aurais dû utiliser seulement -atime, mais ça m'aurait probablement donné beaucoup de fichiers (en fait tous les fichiers modifiés depuis 7 jours dans mon répertoire personnel !).

Là encore, vous pouvez utiliser un "+" à la place... ou même enlever le signe pour rechercher un fichier qui aurait été accédé il y a exactement 7 jours.

Rechercher uniquement des répertoires ou des fichiers

On peut aussi rechercher uniquement des répertoires ou des fichiers.

Utilisez :

- **-type d** : pour rechercher uniquement des répertoires (directories).
- **-type f** : pour rechercher uniquement des fichiers (files).

Par défaut, find cherche des répertoires ET des fichiers. Ainsi, si vous avez un fichier appelé syslog et un répertoire appelé syslog, les deux résultats seront affichés.

Pour avoir uniquement les répertoires qui s'appellent syslog (et pas les fichiers), tapez donc :

Code : Console

```
find /var/log -name "syslog" -type d
```

Utilisation avancée avec manipulation des résultats

Pour l'instant, n'avons pas indiqué de paramètre "que faire avec" pour effectuer une action sur les résultats trouvés. Par défaut, les noms des fichiers trouvés étaient affichés.

En fait,

Code : Console

```
find ~ -name "*.jpg"
```

...est équivalent à écrire :

Code : Console

```
find ~ -name "*.jpg" -print
```

-print signifie "Afficher les résultats trouvés".

Si le -print n'est pas écrit, la commande comprend toute seule qu'elle doit afficher la liste des fichiers.

On peut cependant remplacer ce -print par d'autres paramètres.

Afficher les fichiers de façon formatée

Par défaut, on liste juste les noms des fichiers trouvés. On peut cependant avec l'option -printf (qui rappellera à certains le langage C) manipuler un peu ce qui est affiché.

Exemple :

Code : Console

```
mateo21@mateo21-desktop:~$ find . -name "*.jpg" -printf "%p - %u\n"
./photos/australie1.jpg - mateo21
./photos/australie2.jpg - mateo21
./photos/australie3.jpg - mateo21
```

Ici, j'affiche le nom du fichier, un tiret, et le nom du propriétaire de ce fichier. Le \n permet d'aller à la ligne.

Je vous conseille fortement de lire la doc, à la section "-printf" (faites une recherche). Direction : `man find` ! Vous y trouverez tous les éléments utilisables avec -printf, en plus du %p et du %u.

Supprimer les fichiers trouvés

Un des usages les plus courants de find, à part retrouver des fichiers, consiste à les supprimer.

Si je veux faire le ménage dans mon home, et par exemple supprimer tous mes fichiers jpg, je vais écrire ceci :

Code : Console

```
find ~ -name "*.jpg" -delete
```



Soyez bien sûr de ce que vous faites ! Il n'y aura pas de confirmation !

Et voilà, toutes les images ont disparu. 😞

Appeler une commande

Avec `-exec`, vous pouvez appeler une commande qui effectuera une action sur chacun des fichiers trouvés.

Imaginons que je souhaite mettre un `chmod` à 600 pour chacun de mes fichiers `jpg`, pour que je sois le seul à pouvoir les lire :

Code : Console

```
find ~ -name "*.jpg" -exec chmod 600 {} \;
```

La commande n'affiche rien s'il n'y a pas eu d'erreur.



Euh, comment ça marche ce truc ? 🤔

Pour chaque fichier `jpg` trouvé, on exécute la commande qui suit `-exec` :

- Cette commande ne doit PAS être entre guillemets.
- Les accolades `{}` seront remplacées par le nom du fichier.
- La commande doit finir par un `\;` obligatoirement.

C'est un peu compliqué au premier abord, mais c'est très puissant ! 😊

Vous pouvez faire ce que vous voulez avec ça.

Exercice : essayez de regrouper tous les fichiers `jpg` éparpillés dans votre répertoire home dans un dossier "images".



Si le fait que la commande ne vous demande pas de confirmation vous ennuie, vous pouvez utiliser `-ok` à la place de `-exec`. Le principe est le même, mais on vous demandera une confirmation pour chaque fichier rencontré. Il faudra répondre par "y" (yes) ou "n" (no) à chaque fois.

Désormais, les fichiers ne peuvent plus se cacher ! 🕵️

En fin de compte, `locate` est très rapide et simple d'utilisation, mais `find` est un peu tout l'inverse. Je vous conseille fortement de consulter le manuel (`man find`) pour apprendre de nouvelles options de recherche. Je n'ai pas pu tout vous montrer, mais vous connaissez maintenant au moins l'essentiel !

Partie 3 : Contrôler les processus et les flux de données

Nous entrons maintenant dans une partie un peu plus avancée de la manipulation de la console sous Linux.

Nous allons nous intéresser à 2 thèmes principaux ici : les processus et les flux de données. L'objectif est de vous permettre de savoir qui fait quoi sur votre machine, d'apprendre à gérer les programmes que vous lancez mais aussi de transférer les informations qu'ils renvoient à un autre programme !

Vous allez vraiment commencer à comprendre tout ce qui fait la puissance de la console à partir de maintenant. 😊

Extraire, trier et filtrer des données

Comme vous le savez déjà, la plupart des commandes de Linux sont basées sur le modèle du système d'exploitation Unix. Ce sont les mêmes. Ces commandes s'utilisent pour certaines de la même manière depuis les années 60 ! Avantage pour les informaticiens : pas besoin de réapprendre à utiliser les mêmes commandes tous les 3 mois.

Mais la question que vous devez vous poser est la suivante : comment se fait-il que la plupart de ces commandes n'aient pas changé depuis si longtemps ? La réponse vient du fait qu'elles n'ont pas eu besoin de changer. En effet, la plupart des commandes que vous découvrez sont très basiques : elles remplissent une tâche et la remplissent bien, mais elles ne font pas plus. Ce sont des "briques de base" du système.

Dans ce chapitre, nous allons découvrir une série de commandes basiques qui permettent d'extraire, trier et filtrer des données dans des fichiers. Vous utiliserez certaines d'entre elles (comme `grep`) presque tous les jours !

grep : filtrer des données

La commande `grep` est essentielle. De toutes celles présentées dans ce chapitre, il s'agit probablement de la plus couramment utilisée.

Son rôle est de rechercher un mot dans un fichier et d'afficher les lignes dans lesquelles ce mot a été trouvé. L'avantage de cette commande est qu'elle peut être utilisée très simplement ou de manière plus complexe (mais plus précise) selon les besoins, en faisant appel aux expressions régulières.



Les expressions régulières sont un moyen très puissant de rechercher un texte. On ne les utilise pas seulement dans la ligne de commande Linux, mais aussi dans des éditeurs de texte avancés et dans de nombreux langages de programmation tel que PHP. Vous trouverez d'ailleurs 2 chapitres assez complets au sujet des expressions régulières dans le [cours PHP](#) que j'ai rédigé.

Nous allons commencer par utiliser `grep` de manière très simple, puis nous verrons ensuite comment faire des recherches plus poussées avec les expressions régulières.

Utiliser `grep` simplement

La commande `grep` peut s'utiliser de nombreuses façons différentes. Pour le moment, nous allons suivre le schéma ci-dessous :

Code : Console

```
grep texte nomfichier
```

Le premier paramètre est le texte à rechercher, le second est le nom du fichier dans lequel ce texte doit être recherché.

Essayons par exemple de rechercher le mot "alias" dans notre fichier de configuration `.bashrc`. Rendez-vous dans votre répertoire personnel (en tapant `cd`), et lancez la commande suivante :

Code : Console

```
grep alias .bashrc
```

Cette commande demande à rechercher le mot "alias" dans le fichier .bashrc et affiche toutes les lignes dans lesquelles le mot a été trouvé.

Résultat :

Code : Console

```
$ grep alias .bashrc

# ~/.bash_aliases, instead of adding them here directly.
#if [ -f ~/.bash_aliases ]; then
#   . ~/.bash_aliases
# enable color support of ls and also add handy aliases
  alias ls='ls --color=auto'
  #alias dir='ls --color=auto --format=vertical'
  #alias vdir='ls --color=auto --format=long'
# some more ls aliases
alias ll='ls -lArth'
#alias la='ls -A'
#alias l='ls -CF'
```

Pas mal n'est-ce pas ? Comme vous pouvez le voir, grep est plus un outil de filtre qu'un outil de recherche. Son objectif est de vous afficher uniquement les lignes qui contiennent le mot que vous avez demandé.

Notez qu'il n'est pas nécessaire de mettre des guillemets autour du mot recherché, sauf si vous recherchez une suite de plusieurs mots séparés par des espaces comme ceci :

Code : Console

```
grep "Site du Zéro" monfichier
```

-i : ne pas tenir compte de la casse (majuscules / minuscules)

Par défaut, grep tient compte de la casse : il fait la distinction entre les majuscules et les minuscules. Ainsi, si vous recherchez "alias" et qu'une ligne contient "Alias", grep ne la renverra pas.

Pour que grep renvoie toutes les lignes qui contiennent "alias", peu importe les majuscules et les minuscules, utilisez l'option -i :

Code : Console

```
$ grep -i alias .bashrc

# Alias definitions.
# ~/.bash_aliases, instead of adding them here directly.
#if [ -f ~/.bash_aliases ]; then
#   . ~/.bash_aliases
# enable color support of ls and also add handy aliases
  alias ls='ls --color=auto'
  #alias dir='ls --color=auto --format=vertical'
  #alias vdir='ls --color=auto --format=long'
# some more ls aliases
alias ll='ls -lArth'
#alias la='ls -A'
#alias l='ls -CF'
```

On notera que la première ligne renvoyée (ici surlignée) n'était pas présente tout à l'heure car le mot "Alias" contenait une majuscule. Avec l'option `-i`, on peut maintenant la voir.

-n : connaître les numéros des lignes

Vous pouvez afficher les numéros des lignes retournées avec `-n` :

Code : Console

```
$ grep -n alias .bashrc

49:# ~/.bash_aliases, instead of adding them here directly.
52:#if [ -f ~/.bash_aliases ]; then
53:#    . ~/.bash_aliases
56:# enable color support of ls and also add handy aliases
59:    alias ls='ls --color=auto'
60:    #alias dir='ls --color=auto --format=vertical'
61:    #alias vdir='ls --color=auto --format=long'
64:# some more ls aliases
65:alias ll='ls -lArth'
66:#alias la='ls -A'
67:#alias l='ls -CF'
```

-v : inverser la recherche : ignorer un mot

Si, au contraire, vous voulez connaître toutes les lignes qui **ne contiennent pas** un mot donné, utilisez `-v` :

Code : Console

```
$ grep -v alias .bashrc

# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-
doc)
# for examples

# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# don't put duplicate lines in the history. See bash(1) for more options
export HISTCONTROL=ignoredups
# ... and ignore same successive entries.
export HISTCONTROL=ignoreboth

# ... (renvoie beaucoup de lignes, je ne mets pas tout ici)
```

Cette fois, on récupère toutes les lignes du fichier `.bashrc` qui ne contiennent pas le mot "alias".

-r : rechercher dans tous les fichiers et sous-dossiers

Si vous ne savez pas dans quel fichier se trouve le texte que vous recherchez, vous pouvez sortir l'artillerie lourde : l'option `-r` (récursive). Cette fois, il faudra indiquer en dernier paramètre le **nom du répertoire** dans lequel la recherche doit être faite (et non pas le nom d'un fichier).

Code : Console

```
grep -r "Site du Zéro" code/
```

... recherchera la chaîne "Site du Zéro" dans tous les fichiers du répertoire "code", y compris dans les sous-dossiers.



Notez que le "/" à la fin n'est pas obligatoire. Linux comprendra très bien qu'il s'agit d'un répertoire sans cela.

Code : Console

```
$ grep -r "Site du Zéro" code/

code/intro.html: Nous vous souhaitons la bienvenue sur le Site du Zéro !
code/tpl/define.tpl: Le Site du Zéro
```

Cette fois, le nom du fichier dans lequel la chaîne a été trouvée s'affiche au début de la ligne.



A noter qu'il existe aussi la commande `rgrep` qui est équivalente à écrire `grep -r`.

Utiliser grep avec des expressions régulières

Pour faire des recherches plus poussées, pour ne pas dire des recherches très poussées, vous devez faire appel aux expressions régulières. C'est un ensemble de symboles qui va vous permettre de dire à l'ordinateur très précisément ce que vous recherchez.

Je vous propose dans un premier temps de jeter un oeil à ce tableau des principaux caractères spéciaux qu'on utilise dans les expressions régulières :

Caractère spécial	Signification
.	Caractère quelconque
^	Début de ligne
\$	Fin de ligne
[]	Un des caractères entre les crochets
?	L'élément précédent est optionnel (peut être présent 0 ou 1 fois)
*	L'élément précédent peut être présent 0, 1 ou plusieurs fois
+	L'élément précédent doit être présent 1 ou plusieurs fois
	Ou
()	Groupement d'expressions



Help ! J'ai rien compris.

Normal. Pour bien faire, il faudrait un ou deux chapitres entiers sur les expressions régulières. Là je n'ai pas vraiment la place de faire un "mini-cours" sur les expressions régulières, aussi je vous propose de jeter un oeil à ces quelques lignes pour apprendre par l'exemple.

Tout d'abord, il faut savoir qu'on doit utiliser l'option `-E` pour faire comprendre à `grep` que l'on utilise une expression régulière.

Code : Console

```
$ grep -E Alias .bashrc
# Alias definitions.
```



Notez que vous pouvez aussi utiliser la commande `egrep` qui est équivalente à écrire `grep -E`.

C'est une expression régulière très simple. Elle demande à rechercher le mot `Alias` (avec un `A` majuscule). Si le mot est présent dans une ligne, la ligne est renvoyée.

Bon jusque là rien de nouveau, ça marchait comme ça avant qu'on utilise les expressions régulières. Essayons de pimenter cela en faisant précéder "Alias" d'un accent circonflexe qui signifie que "le mot doit être placé au début de la ligne" :

Code : Console

```
$ grep -E ^Alias .bashrc
```

Résultat : `grep` ne renvoie rien. En effet, la ligne de tout à l'heure commençait par un `#` et non pas par `Alias`.

En revanche on a un résultat si on fait :

Code : Console

```
$ grep -E ^alias .bashrc
alias ll='ls -lArth'
```

Cette fois la ligne commençait bien par "alias". De même, on aurait pu utiliser un `$` à la fin pour demander à ce que la ligne se termine par `alias`.

Quelques autres exemples que vous pouvez tester :

Code : Console

```
grep -E [Aa]lias .bashrc
```

... renvoie toutes les lignes qui contiennent "alias" ou "Alias".

Code : Console

```
grep -E [0-4] .bashrc
```

... renvoie toutes les lignes qui contiennent un nombre compris entre 0 et 4.

Code : Console

```
grep -E [a-zA-Z] .bashrc
```

... renvoie toutes les lignes qui contiennent un caractère alphabétique compris entre a et z ou entre A et Z.

Je vous ai fait là une introduction très rapide, il y aurait beaucoup à dire. Si vous voulez en savoir plus sur les expressions régulières, direction le [cours PHP](#) où j'ai rédigé des explications plus complètes. Il faudra un peu s'adapter parce que ces explications sont faites pour les programmeurs PHP, mais vous devriez y trouver suffisamment d'exemples et d'explications pour en apprendre davantage.



Comme vous pourrez le constater, les expressions régulières fonctionnent aussi bien sans le -E. Pourquoi ? Normalement, cette option sert à activer la gestion des expressions régulières les plus complexes. Dans la pratique, le manuel nous dit que la version GNU de grep (celle qu'on utilise sous Linux) ne fait pas de différence que l'option soit présente ou pas. Les expressions régulières sont toujours activées. En clair, vous aurez besoin du -E si vous utilisez grep sur une autre machine de type Unix un jour, mais en attendant vous pouvez très bien vous en passer. Le -E a été conservé pour des raisons de compatibilité.

sort : trier les lignes

La commande sort se révèle bien utile lorsqu'on a besoin de trier le contenu d'un fichier.

Pour nos exemples, je vous propose de créer un nouveau fichier (avec nano par exemple) appelé "noms.txt" et d'y placer le texte suivant :

Code : Autre

```
François  
Marcel  
Albert  
Jean  
Stéphane  
patrice  
Vincent  
jonathan
```

Ensuite, exécutez la commande sort sur ce fichier :

Code : Console

```
$ sort noms.txt  
  
Albert  
François  
Jean  
jonathan  
Marcel  
patrice  
Stéphane  
Vincent
```

Le contenu du fichier est trié alphabétiquement et le résultat est affiché dans la console. Vous noterez que sort ne fait pas attention à la casse (majuscules / minuscules).

-o : écrire le résultat dans un fichier

Le fichier en lui-même n'a pas été modifié lorsque nous avons lancé la commande. Seul le résultat était affiché dans la console.

Vous pouvez faire en sorte que le fichier soit modifié en précisant un nom de fichier avec l'option `-o` :

Code : Console

```
sort -o noms_tries.txt noms.txt
```

... écrira la liste de noms triés dans `noms_tries.txt`.

-r : trier en ordre inverse

L'option `-r` permet d'inverser le tri :

Code : Console

```
$ sort -r noms.txt  
Vincent  
Stéphane  
patrice  
Marcel  
jonathan  
Jean  
François  
Albert
```

-R : trier aléatoirement

Cette option permet de trier aléatoirement les lignes d'un fichier. C'est assez marrant et ça peut se révéler utile dans certains cas :

Code : Console

```
$ sort -R noms.txt  
patrice  
François  
Marcel  
jonathan  
Jean  
Albert  
Vincent  
Stéphane
```

-n : trier des nombres

Le tri de nombres est un peu particulier. En effet, la commande `sort` ne reconnaît pas si les caractères sont des nombres et trie par défaut alphabétiquement. Par conséquent, le "mot" 129 précèdera 42, alors que ce devrait être l'inverse !

Prenons un exemple. Créez un nouveau fichier "nombres.txt" et placez-y le contenu suivant :

Code : Autre

```
36
16
42
129
27
364
```

Triez-les comme vous avez appris à le faire :

Code : Console

```
$ sort nombres.txt

129
16
27
36
364
42
```

Alphabétiquement, ces nombres sont bien triés. Tout ce qui commence par 1 est en premier, puis vient ce qui commence par 2, et ainsi de suite.

Bien sûr, quand on veut trier des nombres, c'est n'importe quoi.

C'est là que l'option `-n` intervient. Elle permet de trier en considérant le texte comme des nombres. Cette fois, le nombre 42 sera bien placé avant 129 !

Code : Console

```
$ sort -n nombres.txt

16
27
36
42
129
364
```

Magique. 😊

wc : compter le nombre de lignes

La commande `wc` signifie "Word Count". C'est donc a priori un compteur de mots, mais en fait on lui trouve plusieurs autres utilités : compter le nombre de lignes (très fréquent) et compter le nombre de caractères.

Comme les précédentes, la commande `wc` travaille sur un fichier.

Sans paramètres, les résultats renvoyés par `wc` sont un peu obscurs. Voyez plutôt :

Code : Console

```
$ wc noms.txt

8 8 64 noms.txt
```

Ces 3 nombres signifient, dans l'ordre :

1. Le nombre de lignes
2. Le nombre de mots
3. Le nombre d'octets

Il fallait le savoir ! 🤔



Dans le cas de notre fichier noms.txt, il est normal d'avoir autant de lignes que de mots car nous avons mis un seul mot par ligne.

-l : compter le nombre de lignes

Pour avoir uniquement le nombre de lignes, utilisez -l :

Code : Console

```
$ wc -l noms.txt
8 noms.txt
```

-w : compter le nombre de mots

Combien de mots différents y a-t-il dans le fichier ?

Code : Console

```
$ wc -w noms.txt
8 noms.txt
```

-c : compter le nombre d'octets

Combien d'octets compte le fichier ?

Code : Console

```
$ wc -c noms.txt
64 noms.txt
```

-m : compter le nombre de caractères

Ah, voilà une information qui ne nous a pas été donnée lorsque nous avons lancé la commande wc sans paramètres.

L'option -m renvoie le nombre de caractères :

Code : Console

```
$ wc -m noms.txt  
62 noms.txt
```

Comme vous pouvez le voir, le nombre de caractères est différent du nombre d'octets.

uniq : supprimer les doublons

Parfois, certains fichiers contiennent des lignes en double et on aimerait pouvoir les détecter ou les supprimer. La commande `uniq` est toute indiquée pour cela.

Nous devons travailler sur un fichier **trié**. En effet, la commande `uniq` ne repère que les lignes successives qui sont identiques. Je vous propose de créer un fichier `doublons.txt` contenant les noms suivants :

Code : Autre

```
Albert  
François  
François  
François  
Jean  
jonathan  
Marcel  
Marcel  
patrice  
Stéphane  
Vincent
```

Il y a des noms en double (et même en triple) dans ce fichier. Appliquons un petit coup de `uniq` là-dessus pour voir ce qu'il en reste :

Code : Console

```
$ uniq doublons.txt  
Albert  
François  
Jean  
jonathan  
Marcel  
patrice  
Stéphane  
Vincent
```

La liste de noms sans les doublons s'affiche alors dans la console ! 😊

Vous pouvez demander à ce que le résultat sans doublons soit écrit dans un autre fichier plutôt qu'affiché dans la console :

Code : Console

```
uniq doublons.txt sans_doublons.txt
```

La liste sans doublons sera écrite dans `sans_doublons.txt`.

-c : compter le nombre d'occurrences

Avec -c, la commande uniq vous affiche le nombre de fois que la ligne est présente dans le fichier :

Code : Console

```
$ uniq -c doublons.txt
  1 Albert
  3 François
  1 Jean
  1 jonathan
  2 Marcel
  1 patrice
  1 Stéphane
  1 Vincent
```

On sait ainsi qu'il y a 3 fois François, 1 fois Jean, 2 fois Marcel, etc.

-d : afficher uniquement les lignes présentes en double

L'option -d demande à afficher uniquement les lignes présentes en double :

Code : Console

```
$ uniq -d doublons.txt

François
Marcel
```

Comme seuls François et Marcel avaient des doublons, on les voit ici s'afficher dans la console.



Comme pour les autres commandes présentées dans ce chapitre, je ne vous ai pas fait la liste de toutes les options disponibles. J'ai choisi de vous présenter celles qui me paraissent les plus intéressantes ou les plus utiles, mais c'est tout à fait subjectif. Ayez le réflexe d'aller regarder le manuel (`man uniq` par exemple) pour connaître la liste exhaustive des options de la commande.

cut : couper une partie du fichier

Vous avez déjà coupé du texte dans un éditeur de texte, non ?

La commande cut vous propose de faire cela au sein d'un fichier, afin de conserver uniquement une partie de chaque ligne.

Couper selon le nombre de caractères

Par exemple, si vous souhaitez conserver uniquement les caractères 2 à 5 de chaque ligne du fichier, vous taperez :

Code : Console

```
$ cut -c 2-5 noms.txt

ran
arce
lber
ean
tép
atri
```

```
ince  
onat
```



cut a quelques soucis avec les mots contenant des accents. Comme vous pouvez le voir, certains mots ici coupés ont 4 lettres (comme prévu) et d'autres en ont 3.

C'est dû à l'encodage des caractères à cause des accents. La commande cut se base sur le nombre d'octets, et, comme nous l'avons vu plus tôt, le nombre d'octets n'est pas forcément égal au nombre de caractères. A l'heure actuelle on ne peut rien faire pour cela, c'est la commande cut qui devra être mise à jour par les programmeurs.

Pour conserver du 1er au 3ème caractère :

Code : Console

```
$ cut -c -3 noms.txt  
  
Fra  
Mar  
Alb  
Jea  
St  
pat  
Vin  
jon
```

Comme vous pouvez le voir, si on ne met pas de chiffre au début, cut comprend que vous voulez parler du premier caractère.

De même, pour conserver du 3ème au dernier caractère :

Code : Console

```
$ cut -c 3- noms.txt  
  
ançois  
rcel  
bert  
an  
éphane  
trice  
ncent  
nathan
```

Là encore, pas besoin de donner le numéro du dernier caractère (ce serait un peu ennuyant de compter à chaque fois, avouez 🤔), la commande cut comprend comme une grande qu'elle doit couper jusqu'à la fin.

Couper selon un délimiteur

Faisons maintenant quelque chose de bien plus intéressant. Plutôt que de s'amuser à compter le nombre de caractères, on va travailler avec ce qu'on appelle un délimiteur.

Prenons un cas pratique : les fichiers CSV. Vous en avez peut-être déjà vu : ils sont générés par des tableurs tels que Excel et Openoffice.org pour faciliter l'échange et le traitement de données.

Imaginons que vous ayez une (petite) classe et que vous rendiez les notes du dernier contrôle. Vous avez fait un joli tableur et vous avez enregistré le document au format CSV. Le fichier sur lequel nous allons nous baser sera le suivant :

Code : Autre

```
Fabrice,18 / 20,Excellent travail  
Mathieu,3 / 20,Nul comme d'hab  
Sophie,14 / 20,En nette progression  
Mélanie,9 / 20,Allez presque la moyenne !  
Corentin,11 / 20,Pas mal mais peut mieux faire  
Albert,20 / 20,Toujours parfait  
Benoît,5 / 20,En grave chute
```

Comme le nom CSV l'indique, les virgules servent à séparer les colonnes. Ces colonnes contiennent, dans l'ordre :

- Le prénom
- La note
- Un commentaire

C'est un exemple tout à fait fictif bien entendu. 😊

Créez un nouveau fichier avec le texte que je viens de vous donner, que vous appellerez par exemple "notes.csv".

Imaginons que nous souhaitons extraire de ce fichier la liste des prénoms. Comment nous y prendrions-nous ?

On ne peut pas utiliser la technique qu'on vient d'apprendre car les prénoms ne font pas tous la même longueur. Nous allons donc nous servir du fait que nous savons que la virgule sépare les différents champs dans ce fichier.

Vous allez avoir besoin d'utiliser 2 paramètres :

- **-d** : indique quel est le délimiteur dans le fichier
- **-f** : indique le numéro du ou des champs à couper

Dans notre cas, le délimiteur qui sépare les champs est la virgule.
Le numéro du champ à couper est 1 (c'est le premier).

Testez donc ceci :

Code : Console

```
$ cut -d , -f 1 notes.csv  
  
Fabrice  
Vincent  
Sophie  
Mélanie  
Corentin  
Albert  
Benoît
```

C'est pas beau ça ? 😞

Après le -d, nous avons indiqué quel était le délimiteur (à savoir la virgule ",").
Après le -f, nous avons indiqué le numéro du champ à conserver (le premier).

Si nous voulons juste les commentaires :

Code : Console

```
$ cut -d , -f 3 notes.csv  
  
Excellent travail  
Nul comme d'hab  
En nette progression  
Allez presque la moyenne !  
Pas mal mais peut mieux faire  
Toujours parfait  
En grave chute
```

Pour avoir les champs n°1 et n°3 (le prénom et le commentaire) :

Code : Console

```
$ cut -d , -f 1,3 notes.csv  
  
Fabrice,Excellent travail  
Vincent,Nul comme d'hab  
Sophie,En nette progression  
Mélanie,Allez presque la moyenne !  
Corentin,Pas mal mais peut mieux faire  
Albert,Toujours parfait  
Benoît,En grave chute
```



De même, il est possible de conserver toute une série de champs avec le tiret comme tout à l'heure : `cut -d , -f 2-4 notes.csv` a pour effet de conserver les champs n°2, 3 et 4.
D'autre part, `cut -d , -f 3- notes.csv` conserve les champs du n°3 jusqu'à la fin. 😊

Vous êtes bien obligés d'admettre que, quand on sait bien s'en servir, la console de Linux peut vous permettre d'effectuer des opérations vraiment puissantes que vous ne pensiez même pas pouvoir faire aussi simplement jusqu'à présent. 😊

Les commandes que nous avons découvertes dans ce chapitre et les précédents sont vraiment très basiques, comme je vous l'avais annoncé. Vous devez peut-être vous demander d'ailleurs pourquoi un système d'exploitation comme Linux propose des commandes aussi basiques telle que "sort" qui permet de trier du texte.

Je l'avoue, ces commandes utilisées seules ont assez peu d'intérêt en général. C'est en les combinant entre elles qu'elles prendront tout leur sens, comme nous allons le voir dans le chapitre suivant. Préparez-vous à en prendre plein les mirettes ! 😊

Les flux de redirection

Vous devriez maintenant avoir l'habitude d'un certain nombre de commandes que propose la console de Linux. Le fonctionnement est toujours le même :

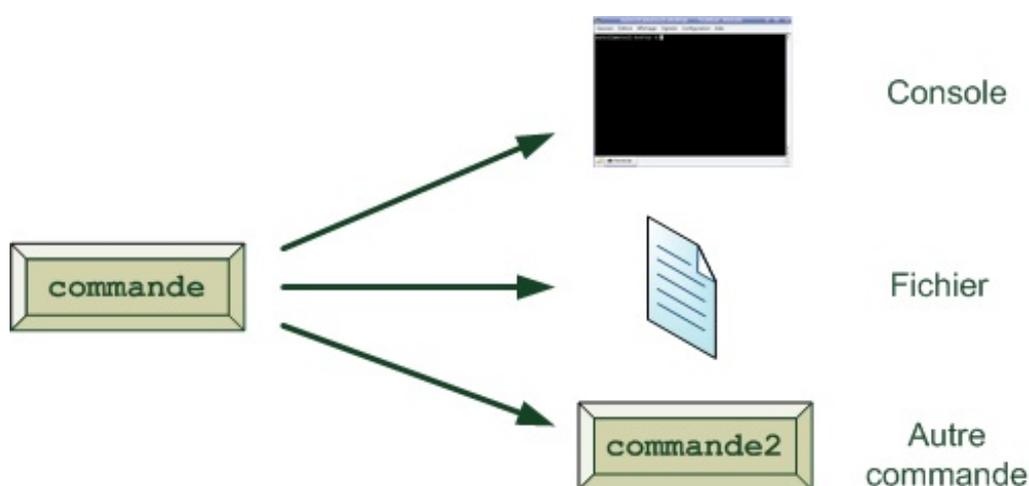
1. Vous tapez la commande (par exemple `ls`).
2. Le résultat s'affiche dans la console.

Ce que vous ne savez pas encore, c'est qu'**il est possible de rediriger ce résultat**. Au lieu que le résultat s'affiche dans la console, vous allez pouvoir l'envoyer *ailleurs*.

Où ? Dans un fichier, ou en entrée d'une autre commande pour "chaîner des commandes". Ainsi, le résultat d'une commande peut en déclencher une autre !

Comment ? A l'aide de petits symboles spéciaux, appelés flux de redirection, que vous allez découvrir dans ce chapitre.

Le principe peut être résumé dans ce schéma :



Jusqu'ici, nous n'avons donc exploité que la première possibilité (celle par défaut) : afficher le résultat dans la console. Il nous reste donc bien d'autres techniques à découvrir !



Les flux de redirection sont une composante essentielle de la console sous Linux, et ce depuis l'époque d'Unix. Ils vont très certainement changer votre façon de "voir" comment la console fonctionne et démultiplier votre contrôle sur les commandes que vous lancez. C'est dire si ce chapitre est important !

Je vais donc d'abord vous demander d'être encore plus attentifs que d'habitude. Non pas que le chapitre soit réellement "compliqué", mais il *doit* être bien compris pour que vous puissiez suivre le reste du cours convenablement.

Au pire des cas, vous pourrez toujours revenir lire ce chapitre si vous avez un trou de mémoire sur les notions que vous y avez apprises. 😊

> et >> : rediriger le résultat dans un fichier

La manipulation la plus simple que nous allons voir va nous permettre d'écrire le résultat d'une commande dans un fichier, au lieu de l'afficher bêtement dans la console.

Préparatifs

Prenons une commande au hasard. Vous vous souvenez de cut, que nous avons appris dans le chapitre précédent ?

Nous avons travaillé sur un petit fichier de type "CSV" que les tableurs peuvent générer.

Ce sont les notes des élèves d'une classe à un contrôle :

Code : Autre

```
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque la moyenne !
Corentin,11 / 20,Pas mal mais peut mieux faire
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```



Si vous ne l'aviez pas déjà fait dans le chapitre précédent, je vous recommande d'enregistrer ce fichier en copiant/collant le contenu ci-dessus dans un éditeur de texte (comme nano). Enregistrez le tout sous le nom "notes.csv".

La commande `cut` nous avait permis de "couper" une partie du fichier et d'afficher le résultat dans la console. Par exemple, nous avons demandé à `cut` de prendre tout ce qui se trouvait avant la première virgule afin d'avoir la liste des noms de tous les élèves présents à ce contrôle :

Code : Console

```
$ cut -d , -f 1 notes.csv

Fabrice
Vincent
Sophie
Mélanie
Corentin
Albert
Benoît
```

Ce résultat s'est affiché dans la console. C'est ce que font toutes les commandes par défaut... à moins que l'on utilise un flux de redirection ! 😊

> : rediriger dans un nouveau fichier

Supposons que nous souhaitions écrire la liste des prénoms dans un fichier, afin de garder sous le coude la liste des élèves présents au contrôle.

C'est là qu'intervient le petit symbole magique `>` (appelé *chevron*) que je vous laisse trouver sur votre clavier (ceux qui font du HTML le connaissent bien 😊).

Ce symbole permet de rediriger le résultat de la commande dans le fichier de votre choix. Essayez par exemple de taper ceci :

Code : Console

```
cut -d , -f 1 notes.csv > eleves.txt
```

Regardez la fin de la commande. J'y ai rajouté la petite flèche `>` qui redirige la sortie de la commande dans un fichier. Normalement, si vous exécutez cette commande rien ne s'affichera dans la console. Tout aura été redirigé dans un fichier appelé "eleves.txt" qui vient d'être créé pour l'occasion dans le dossier où vous vous trouviez.



Je le rappelle au cas où : sous Linux on se moque pas mal de l'extension des fichiers. J'aurais très bien pu créer un fichier sans extension appelé "eleves". Ici j'ai rajouté un ".txt" pour ne pas dérouter ceux qui viennent de Windows, mais il faudra vous habituer à travailler avec des noms de fichiers sans extension parfois.

Faites un petit `ls` (ou `ls -l`, comme vous préférez) pour voir que le fichier est bien présent dans le dossier :

Code : Console

```
$ ls -l
total 20
-rw-r--r-- 1 mateo21 mateo21 91 2008-04-19 19:36 doublons.txt
-rw-r--r-- 1 mateo21 mateo21 56 2008-09-26 12:01 eleves.txt
-rw-r--r-- 1 mateo21 mateo21 35 2008-04-19 17:06 fichier_trie.txt
-rw-r--r-- 1 mateo21 mateo21 20 2008-04-19 19:03 nombres.txt
-rw-r--r-- 1 mateo21 mateo21 253 2008-09-26 12:01 notes.csv
```

Comme vous pouvez le voir, un fichier vient bien d'être créé ! 😊

Vous pouvez l'ouvrir avec nano ou encore l'afficher dans la console avec la commande `cat` (pour afficher tout d'un coup s'il est court) ou `less` (pour afficher page par page s'il est long).



Attention : si le fichier existait déjà il sera écrasé sans demande de confirmation !



Parfois, vous ne voulez ni voir le résultat d'une commande ni le stocker dans un fichier. Dans ce cas, l'astuce consiste à rediriger le résultat dans `/dev/null`. C'est un peu le "trou noir" de Linux : tout ce qui va là-dedans disparaît immédiatement.

Par exemple : `commande_bavarde > /dev/null`

>> : rediriger à la fin d'un fichier

Le double chevron `>>` sert lui aussi à rediriger le résultat dans un fichier, mais cette fois à la fin de ce fichier.

Avantage : vous ne risquez pas d'écraser le fichier s'il existe déjà. Si le fichier n'existe pas, il sera créé automatiquement.

Normalement, vous devriez avoir créé un fichier `eleves.txt` lors des manipulations précédentes. Si vous faites :

Code : Console

```
cut -d , -f 1 notes.csv >> eleves.txt
```

... les noms seront ajoutés à la fin du fichier, sans écraser le résultat précédent.

Bon du coup, on a des noms en double maintenant :

Code : Console

```
$ cat eleves.txt
Fabrice
Mathieu
Sophie
Mélanie
Corentin
Albert
Benoît
Fabrice
Mathieu
Sophie
Mélanie
Corentin
Albert
Benoît
```

Heureusement, vous connaissez les commandes `sort` et `uniq` qui peuvent vous permettre de faire un peu de ménage là-dedans. Je vous laisse supprimer les doublons. 😊

N'oubliez pas qu'il faut que le fichier soit trié pour que la commande `uniq` fonctionne !



Quand utilise-t-on le double chevron pour mettre le résultat à la fin d'un fichier ?

Personnellement, j'ai des commandes qui s'exécutent automatiquement à certaines heures (on verra comment faire ça plus tard). Comme je ne suis pas devant mon ordinateur lorsque ces commandes s'exécutent, j'enregistre un log de ce qui s'est passé dans un fichier :

Code : Console

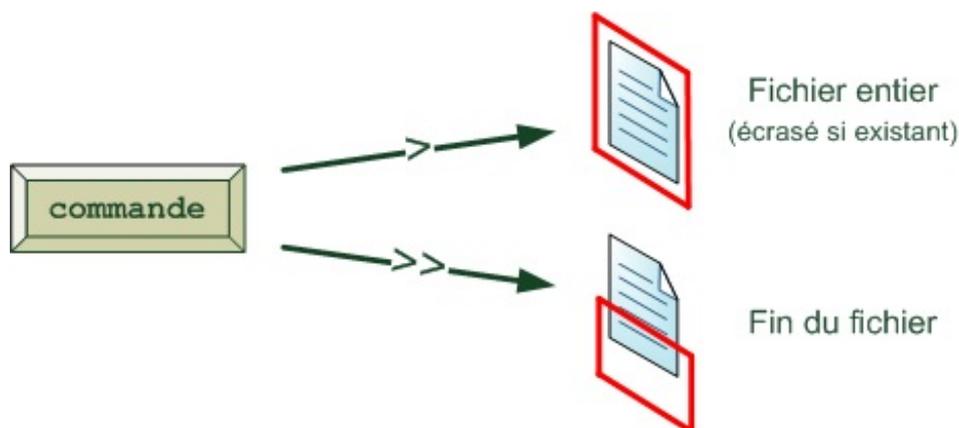
```
macommande >> resultats.log
```

Grâce à ça, si j'ai un doute sur ce qui a pu se passer lors de l'exécution d'une commande, je n'ai qu'à consulter le fichier `resultats.log`.

Résumé

Nous venons de découvrir 2 flux de redirection dans des fichiers :

- `>` : redirige dans un fichier et l'écrase s'il existe déjà.
- `>>` : redirige à la fin d'un fichier et le crée s'il n'existe pas.



2>, 2>> et 2>&1 : rediriger les erreurs

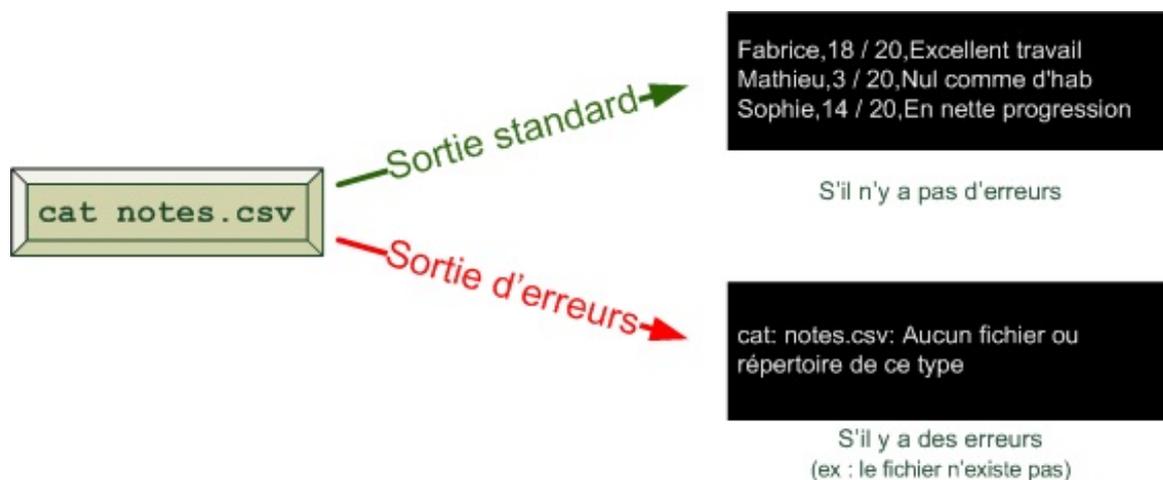
Allons un peu plus loin. Il faut savoir que toutes les commandes produisent 2 flux de données différents :

- **La sortie standard** : pour tous les messages (sauf les erreurs).
- **La sortie d'erreurs** : pour toutes les erreurs.

Prenons un exemple concret pour voir comment ça se passe.

Supposons que vous fassiez un `cat` du fichier `notes.csv` pour afficher son contenu. Il y a 2 possibilités :

- **Si tout va bien** : le résultat (le contenu du fichier) s'affiche sur la sortie standard.
- **S'il y a une erreur** : celle-ci s'affiche dans la sortie d'erreurs.



Par défaut, tout s'affiche dans la console : la sortie standard comme la sortie d'erreurs. Cela explique pourquoi vous ne faisiez pas la différence entre ces 2 sorties jusqu'ici : elles avaient l'air identiques.

Tout à l'heure, nous avons vu comment rediriger la sortie standard dans un fichier. Toutefois, les erreurs continuent d'être affichées dans la console. Faites le test :

Code : Console

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt  
cut: fichier_inexistant.csv: Aucun fichier ou répertoire de ce type
```

Le fichier "fichier_inexistant.csv" n'existe pas (comme son nom l'indique 🤖). L'erreur s'est affichée dans la console au lieu d'avoir été envoyée dans eleves.txt.

Rediriger les erreurs dans un fichier à part

On pourrait souhaiter "logger" les erreurs dans un fichier d'erreurs à part pour ne pas les oublier et pour pouvoir les analyser ensuite.

Pour cela, on utilise l'opérateur **2>**. Vous avez bien lu : c'est le chiffre 2 collé au chevron que nous avons utilisé tout à l'heure.

Faisons une seconde redirection à la fin de cette commande cut :

Code : Console

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt 2> erreurs.log
```

Il y a deux redirections ici :

- **> eleves.txt** : redirige le résultat de la commande (sauf les erreurs) dans le fichier eleves.txt. C'est la sortie standard.
- **2> erreurs.log** : redirige les erreurs éventuelles dans le fichier erreurs.log. C'est la sortie d'erreurs.

Vous pouvez vérifier : si "fichier_inexistant.csv" n'a pas été trouvé, l'erreur aura été inscrite dans le fichier "erreurs.log" au lieu d'être affichée dans la console.



Notez qu'il est aussi possible d'utiliser **2>>** pour ajouter les erreurs à la fin du fichier.

Fusionner les sorties

Parfois, on n'a pas envie de séparer les informations dans 2 fichiers différents. Heureusement, il est possible de fusionner les sorties dans un seul et même fichier. Comment ?

Il faut utiliser le code suivant : **2>&1**

Cela a pour effet de rediriger toute la sortie d'erreurs dans la sortie standard. Traduction pour l'ordinateur : "*envoie les erreurs au même endroit que le reste*".

Essayez donc ceci :

Code : Console

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt 2>&1
```

Tout ira désormais dans `eleves.txt` : le résultat (si ça a marché) de même que les erreurs (s'il y a eu un problème).

Petite subtilité : je vous ai dit tout à l'heure qu'il était possible de faire **2>>** pour rediriger les erreurs à la fin d'un fichier d'erreurs.



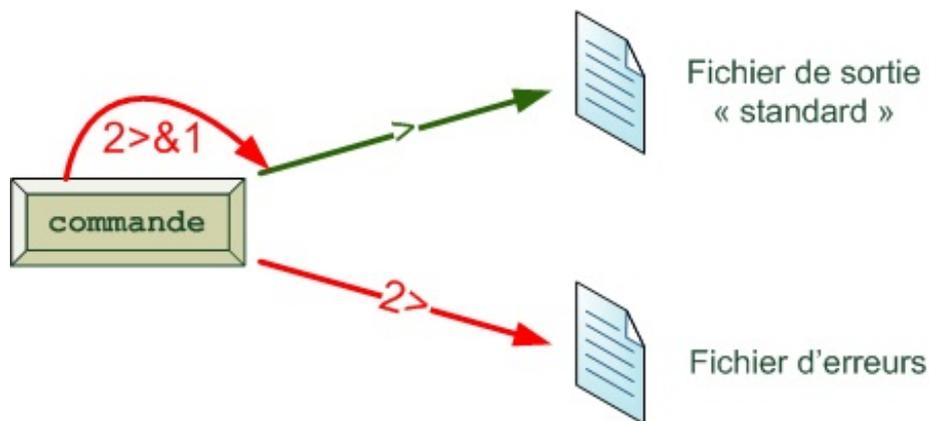
Toutefois, il n'est pas possible d'écrire : **2>>&1**. Essayez ça ne marchera pas.

En fait, le symbole **2>&1** va envoyer les erreurs dans le même fichier et *de la même façon* que la sortie standard. Donc, si vous écrivez : `cut -d , -f 1 fichier_inexistant.csv >> eleves.txt 2>&1` ... les erreurs seront *ajoutées* à la fin du fichier `eleves.txt` comme le reste des messages.

Résumé

Nous avons découvert 3 symboles :

- **2>** : redirige les erreurs dans un fichier (s'il existe déjà il sera écrasé).
- **2>>** : redirige les erreurs à la fin d'un fichier (s'il n'existe pas, il sera créé).
- **2>&1** : redirige les erreurs au même endroit et de la même façon que la sortie standard.



Comprenez-vous bien ce schéma ?

On peut choisir de rediriger les erreurs dans un fichier à part (avec `2>`) ou bien de les rediriger au même endroit que la sortie standard (avec `2>&1`).

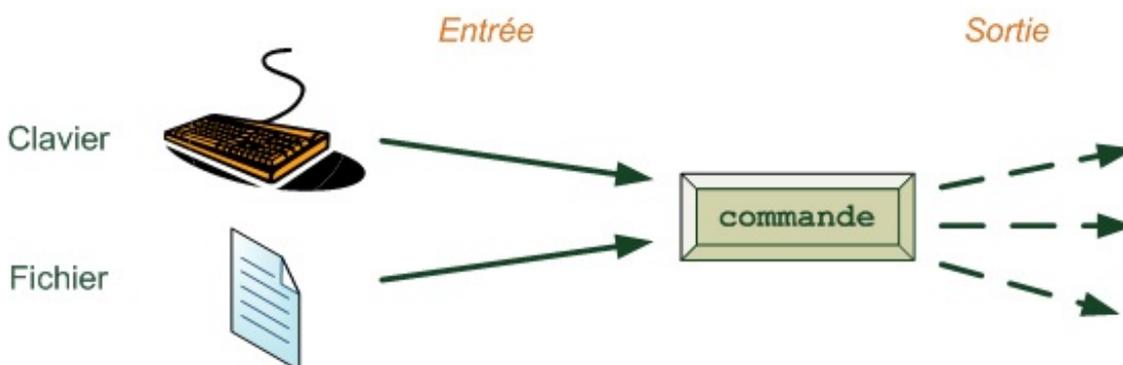


J'ai volontairement omis de parler sur ce schéma de `>>` et de `2>>` afin de ne pas le surcharger, mais le principe est le même sauf qu'on ajoute à la fin d'un fichier au lieu de l'écraser.

< et << : lire depuis un fichier ou le clavier

Pour le moment, nous avons redirigé uniquement la **sortie** des commandes. Nous avons décidé où envoyer les messages issus de ces commandes.

Maintenant, je vous propose de faire un peu l'inverse, c'est-à-dire de décider d'où vient l'**entrée** d'une commande. Jusqu'alors, l'entrée venait des paramètres de la commande... mais on peut faire en sorte qu'elle vienne d'un fichier ou d'une saisie au clavier !



< : lire depuis un fichier

Le chevron ouvrant `<` (à ne pas confondre avec le chevron fermant que nous avons utilisé tout à l'heure) permet d'indiquer d'où vient l'entrée qu'on envoie à la commande.

On va prendre un exemple tout bête : la commande `cat`.

Code : Console

```
cat < notes.csv
```

Cela aura pour effet d'afficher le contenu du fichier envoyé en entrée :

Code : Console

```
$ cat < notes.csv
```

```
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque la moyenne !
Corentin,11 / 20,Pas mal mais peut mieux faire
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```



Ouais. Trop bien.

On faisait pas pareil avant en écrivant juste `cat notes.csv` par hasard ?

Si. Le fait d'écrire `cat < notes.csv` est strictement identique à écrire `cat notes.csv` ... du moins en apparence. Le résultat produit est le même, mais ce qui se passe derrière est très différent :

- Si vous écrivez `cat notes.csv` : la commande `cat` reçoit en entrée le nom du fichier "notes.csv" qu'elle doit ensuite se charger d'ouvrir pour afficher son contenu.
- Si vous écrivez `cat < notes.csv` : la commande `cat` reçoit *le contenu* de notes.csv qu'elle se contente simplement d'afficher dans la console. C'est le shell (le programme qui gère la console) qui se charge d'envoyer le contenu de notes.csv à la commande `cat`.

Bref, ce sont 2 façons de faire la même chose mais de manière très différente.

Pour le moment, je n'ai pas d'exemple plus intéressant à vous proposer à ce sujet, mais retenez cette possibilité car vous finirez par en avoir besoin, faites-moi confiance. 😊

<< : lire depuis le clavier progressivement

Le double chevron ouvrant `<<` fait quelque chose d'assez différent : il vous permet d'envoyer un contenu à une commande avec votre clavier.

Cela peut s'avérer très utile. Je vous propose un exemple concret pour bien voir ce que ça permet de faire en pratique.

Essayez de taper ceci :

Code : Console

```
sort -n << FIN
```

La console vous propose alors de taper du texte.

Code : Console

```
$ sort -n << FIN
>
```

Comme `sort -n` sert à trier des nombres, on va justement écrire des nombres, un par ligne (en appuyant sur la touche `Entrée` à chaque fois).

Code : Console

```
$ sort -n << FIN
```

```
> 13
> 132
> 10
> 131
```

Continuez ainsi jusqu'à ce que vous ayez terminé.

Lorsque vous avez fini, tapez FIN pour arrêter la saisie.

Tout le texte que vous avez écrit est alors envoyé à la commande (ici sort) qui traite cela en entrée. Et, comme vous pouvez vous en douter, la commande sort nous trie nos nombres ! 😊

Code : Console

```
$ sort -n << FIN
> 13
> 132
> 10
> 131
> 34
> 87
> 66
> 68
> 65
> FIN
10
13
34
65
66
68
87
131
132
```

Sympa, non ? 😊

Cela vous évite d'avoir à créer un fichier si vous n'en avez pas besoin.

Vous pouvez faire la même chose avec une autre commande comme par exemple wc pour compter le nombre de mots ou de caractères.

Code : Console

```
$ wc -m << FIN
> Combien de caractères dans cette phrase ?
> FIN
42
```



Une question : ce mot FIN est-il obligatoire ?

Non, vous pouvez le remplacer par ce que vous voulez.

Lorsque vous tapez la commande, vous pouvez utiliser le mot que vous voulez. Par exemple :

Code : Console

```
$ wc -m << STOP
```

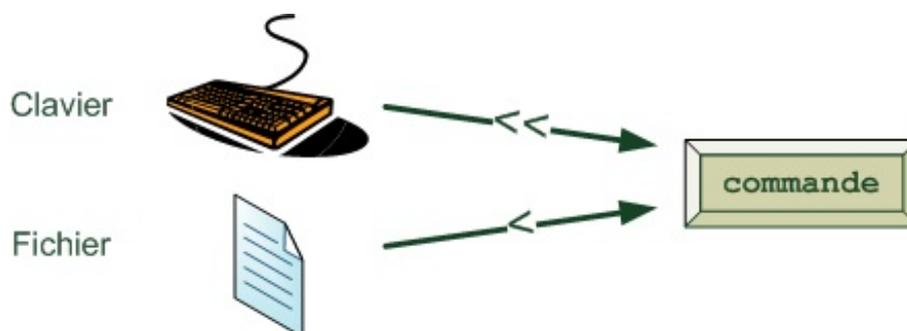
```
> Combien de caractères dans cette phrase ?
> STOP
42
```

Ce qui compte, c'est que vous définissiez un mot-clé qui servira à indiquer la fin de la saisie. 😊
Notez par ailleurs que rien ne vous oblige à écrire ce mot en majuscules.

Résumé

Nous pouvons donc "alimenter" des commandes de 2 manières différentes :

- `<` : envoie le contenu d'un fichier à une commande.
- `<<` : passe la console en mode saisie au clavier, ligne par ligne. Toutes ces lignes seront envoyées à la commande lorsque le mot-clé de fin aura été écrit.



Vous pouvez tout à fait combiner ces symboles avec ceux qu'on a vus précédemment. Par exemple :

Code : Console

```
$ sort -n << FIN > nombres_tries.txt 2>&1
> 18
> 27
> 1
> FIN
```

Les nombres saisis au clavier seront envoyés à `nombres_tries.txt`, de même que les erreurs éventuelles.

Hé, mine de rien on commence à rédiger là des commandes assez complexes.
Mais vous allez voir, on peut faire encore mieux. 😊

| : chaîner les commandes

Passons maintenant au symbole le plus intéressant que vous utiliserez le plus souvent : le *pipe* `|` (prononcez "païpe", comme un bon anglais 😊). Son but ? Chaîner des commandes.



Le pipe `|` n'est pas un symbole qu'on a l'habitude d'écrire. Pourtant, il y en a forcément un sur votre clavier (parfois représenté sous la forme d'une ligne verticale en pointillés).
Sur un clavier AZERTY français par exemple, vous pouvez l'écrire en combinant les touches `Alt Gr + 6`, et sur un clavier belge `Alt Gr + 1`. Sur un clavier Mac, c'est `Alt + Shift + L`.

La théorie

"Chaîner des commandes" ? Cela signifie connecter la sortie d'une commande à l'entrée d'une autre commande.



En gros, tout ce qui sort de la commande1 est immédiatement envoyé à la commande2. Et vous pouvez chaîner des commandes comme cela indéfiniment !

Cette fonctionnalité est vraiment une des plus importantes et elle décuple littéralement les possibilités offertes par la console. Souvenez-vous : dans le chapitre précédent *je vous disais que chaque commande Unix avait un et un seul rôle, mais qu'elle le remplissait bien*. Parfois, l'utilité de certaines commandes seules peut paraître limitée, mais celles-ci prennent en général tout leur sens lorsqu'on les combine à d'autres commandes.

La pratique

Voyons quelques cas concrets (on pourrait trouver une infinité d'exemples 😊).

Trier les élèves par nom

Si vous vous souvenez bien, nous avons toujours un fichier notes.csv qui contient la liste des élèves et leurs notes :

Code : Autre

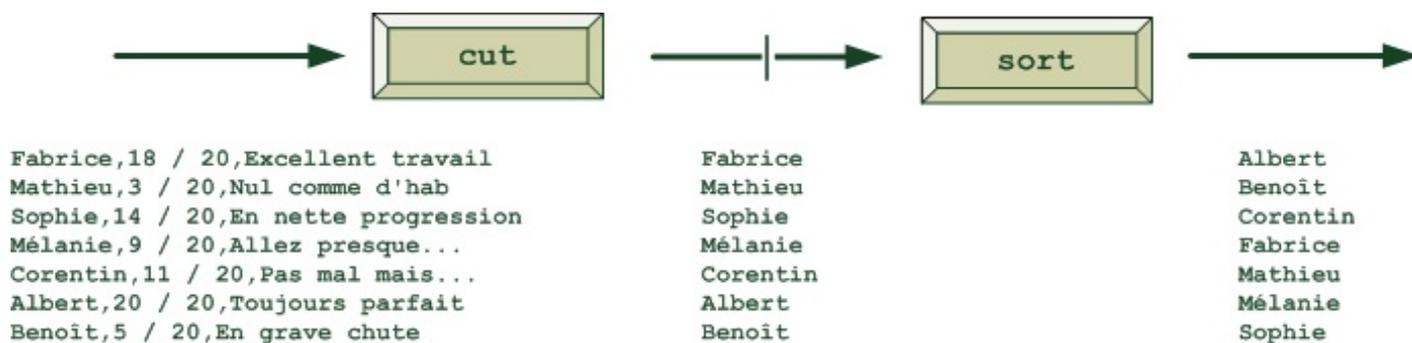
```
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque la moyenne !
Corentin,11 / 20,Pas mal mais peut mieux faire
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```

Avec cut, on peut récupérer les noms. Avec sort, on peut les trier par ordre alphabétique. Pourquoi ne pas connecter cut à sort pour avoir la liste des noms triés ?

Code : Console

```
$ cut -d , -f 1 notes.csv | sort
Albert
Benoît
Corentin
Fabrice
Mathieu
Mélanie
Sophie
```

Le pipe effectue la connexion entre la sortie de cut (des noms dans le désordre) et l'entrée de sort.



On peut même aller plus loin et écrire cette liste triée dans un fichier :



Code : Console

```
cut -d , -f 1 notes.csv | sort > noms_tries.txt
```

Trier les répertoires par taille

La commande `du` permet d'obtenir la taille de chacun des sous-répertoires du répertoire courant (je vous conseille de vous placer dans votre home en tapant d'abord `cd`) :

Code : Console

```
$ du
4      ./gnome2_private
40     ./local/share/Trash/files
4      ./local/share/Trash/info
12     ./local/share/Trash
160   ./local/share
20    ./local
...
```

2 problèmes : cette liste est parfois très longue et elle n'est pas triée.

Un problème à la fois. Tout d'abord, on aimerait par exemple avoir cette même liste dans l'ordre décroissant de taille des répertoires pour repérer plus facilement les plus gros d'entre eux qui prennent de la place sur notre disque.

Pour avoir cette liste du plus grand au plus petit, il nous suffit d'écrire :

Code : Console

```
du | sort -nr
```

On envoie tout le contenu de `du` à `sort` qui se charge de trier les nombres au début de chacune des lignes.

Code : Console

```
$ du | sort -nr
...
```

```
4      ./evolution/memos/config
4      ./evolution/calendar/config
4      ./evolution/cache
4      ./bin
```

Problème : comme les plus gros répertoires ont été affichés en premier, et que j'ai beaucoup de sous-répertoires, je dois remonter très haut dans la console pour retrouver les plus gros d'entre eux.

... Que diriez-vous de connecter cette sortie à `head` ? Cette commande permet de filtrer uniquement les premières lignes qu'elle reçoit, nous l'avons déjà étudiée dans un chapitre précédent.

Code : Console

```
$ du | sort -nr | head
120920  .
59868  ./ies4linux
43108  ./ies4linux/ie6
41360  ./ies4linux/ie6/drive_c
41248  ./ies4linux/ie6/drive_c/windows
40140  ./Desktop
34592  ./ies4linux/ie6/drive_c/windows/system32
16728  ./ies4linux/downloads
13128  ./mozilla
13124  ./mozilla/firefox
```

Vous pouvez paramétrer le nombre de résultats affichés avec l'option `-n` de `head`. Si vous avez oublié comment l'utiliser, direction le manuel ou le chapitre qui en parlait. 😊

Si vous voulez naviguer à travers tous les résultats, vous pouvez connecter la sortie à `less`. Cette commande permet d'afficher des résultats page par page, ça nous est justement utile dans le cas présent où nous avons beaucoup de résultats !

Code : Console

```
du | sort -nr | less
```

Essayez !

Vous allez vous retrouver avec un affichage de `less`, page par page.

Code : Console

```
120920  .
59868  ./ies4linux
43108  ./ies4linux/ie6
41360  ./ies4linux/ie6/drive_c
41248  ./ies4linux/ie6/drive_c/windows
40140  ./Desktop
34592  ./ies4linux/ie6/drive_c/windows/system32
16728  ./ies4linux/downloads
13128  ./mozilla
13124  ./mozilla/firefox
13112  ./mozilla/firefox/v5p4a55d.default
12604  ./ies4linux/downloads/ie6
11808  ./ies4linux/downloads/ie6/FR
5848   ./mozilla/firefox/v5p4a55d.default/Cache
3656   ./ies4linux/ie6/drive_c/windows/profiles
3616   ./ies4linux/ie6/drive_c/windows/profiles/mateo21
3496   ./ies4linux/ie6/drive_c/windows/profiles/mateo21/Local Settings
```

```
3416    ./ies4linux/ie6/drive_c/windows/profiles/mateo21/Local Settings/Tempora
ry Internet Files
3408    ./ies4linux/ie6/drive_c/windows/profiles/mateo21/Local Settings/Tempora
ry Internet Files/Content.IE5
2220    ./ies4linux/ie6/drive_c/windows/fonts
2012    ./ies4linux-2.99.0.1
:
```

Vous pouvez maintenant voir les premiers fichiers (les plus gros) et descendre progressivement vers les fichiers plus petits page par page avec la touche `Espace` ou ligne par ligne avec la touche `Entrée` (ou les flèches du clavier).

C'est pas beau ? 😊

Exercice : peut-être que vous avez toujours trop de répertoires sous les yeux et que vous vous intéressez seulement à certains d'entre eux. Pourquoi ne pas filtrer les résultats avec `grep`, pour afficher uniquement la taille des répertoires liés à... `firefox` par exemple ?

Lister les fichiers contenant un mot

Allez, un dernier exercice tordu pour finir en beauté. 😊

Avec `grep`, on peut connaître la liste des fichiers contenant un mot dans tout un répertoire (option `-r`). Le problème c'est que cette sortie est un peu trop *verbeuse* (il y a trop de texte) : il y a non seulement le nom du fichier mais aussi la ligne dans laquelle le mot a été trouvé.

Code : Console

```
/var/log/installer/syslog:Apr  6 15:14:43 ubuntu NetworkManager: <debug> [120749488
/var/log/installer/syslog:Apr  6 15:23:27 ubuntu python: log-output
```

Heureusement, le nom du fichier et le contenu de la ligne sont séparés par un deux-points. On connaît `cut`, qui permet de récupérer uniquement une partie de la ligne. Il nous permettrait de conserver uniquement le nom du fichier.

Problème : si le même mot a été trouvé plusieurs fois dans un fichier, le fichier apparaîtra en double ! Pour supprimer les doublons, on peut utiliser `uniq`, à condition d'avoir bien trié les lignes avec `sort` auparavant.

Alors, vous avez une petite idée de la ligne qu'il va falloir écrire ?

Je vous propose de rechercher les fichiers qui contiennent le mot "log" dans le dossier `/var/log`. Notez qu'il faudra passer `root` avec `sudo` pour avoir accès à tout le contenu de ce répertoire.

Voici la commande que je vous propose d'utiliser :

Code : Console

```
sudo grep log -Ir /var/log | cut -d : -f 1 | sort | uniq
```

Que fait cette commande ?

1. Elle liste tous les fichiers contenant le mot "log" dans `/var/log` (`-I` permettant d'exclure les fichiers binaires).
2. Elle extrait de ce résultat uniquement les noms des fichiers.
3. Elle trie ces noms de fichiers.
4. Elle supprime les doublons.

Et voilà le résultat !

Code : Console

```
$ sudo grep log -Ir /var/log | cut -d : -f 1 | sort | uniq
/var/log/acpid
/var/log/auth.log
/var/log/boot
/var/log/bootstrap.log
/var/log/dist-upgrade/apt-term.log
/var/log/dmesg
/var/log/dmesg.0
/var/log/gdm/
/var/log/installer/partman
/var/log/installer/syslog
/var/log/kern.log.0
/var/log/messages
/var/log/messages.0
/var/log/syslog
/var/log/syslog.0
/var/log/udev
/var/log/Xorg.0.log
/var/log/Xorg.0.log.old
/var/log/Xorg.20.log
/var/log/Xorg.20.log.old
/var/log/Xorg.21.log
```

Résumé

Le résumé est simple, et c'est dans sa simplicité qu'il tire toute sa beauté et sa puissance (non je ne suis pas fou 🤪) :



S'il y avait un schéma à retenir, ce serait celui-là. Ca tombe bien, c'est le plus simple.

Je vous laisse vous entraîner avec le pipe, nous le réutiliserons très certainement dans les prochains chapitres. Essayez d'inventer des combinaisons ! 😊



L'espace avant et après le pipe n'est en général pas obligatoire, mais je préfère le mettre ici pour une meilleure lisibilité.

Des flux, des flux dans tous les sens : voilà ce qu'il se passe dans votre console.

Les données transitent via un flux standard ou un flux d'erreurs, on peut les intercepter, les manipuler, les renvoyer à une autre commande pour créer des chaînes de commandes et ainsi réaliser des opérations parfois complexes plutôt facilement.

C'est un entraînement. Ce chapitre étant riche en nouveautés, je vous recommande de le relire au moins une fois pour bien l'assimiler.

Si vous avez en outre le sentiment que vous avez déjà oublié certaines commandes décrites ici, n'hésitez pas à relire les chapitres précédents pour vous rafraîchir la mémoire : il n'est jamais trop tard. 😊

Surveiller l'activité du système

Comme tous les OS actuels, Linux est un système **multi-tâches** : il est capable de gérer plusieurs programmes tournant en même temps.

Mieux encore, Linux est un système **multi-utilisateurs** : plusieurs personnes peuvent utiliser la même machine en même temps (en s'y connectant via internet).

Tous ces programmes et ces personnes qui sont sur votre PC peuvent vite donner le tournis. Parfois, l'ordinateur peut se retrouver surchargé à cause d'un programme. Qui a lancé ce programme ? Depuis quand ? Comment arrêter un programme qui ne répond plus ?

Sous Windows, vous avez probablement entendu parler de la commande magique Ctrl + Alt + Suppr qui peut parfois vous sortir de bien des ennuis. Sous Linux, on utilise d'autres outils et d'autres techniques que vous allez apprendre à connaître aujourd'hui.



w : qui fait quoi ?

Nous allons apprendre dans ce chapitre à utiliser une série de commandes qui nous permettront de savoir ce qui se passe actuellement dans notre ordinateur.

La première commande que je veux vous faire découvrir est très courte et facile à retenir : c'est **w** (comme la lettre oui oui 😊).

C'est la première commande que je tape en général quand je me connecte à un serveur surchargé et que je veux essayer de comprendre ce qui se passe. Cela me permet de voir d'un seul coup d'oeil si la machine est vraiment surchargée (et si oui, à quel point) et si quelqu'un d'autre est en train d'intervenir sur la machine.



Si vous utilisez Linux sur votre ordinateur personnel, tranquillement chez vous, vous êtes le seul à l'utiliser en ce moment. Pour que d'autres personnes puissent se connecter à votre ordinateur via internet, il faut avoir configuré Linux pour ça. Nous verrons comment faire cela plus tard. On en a principalement besoin sur les serveurs.

Essayons d'utiliser w pour voir comment ça marche, n'ayez pas peur c'est sans danger :

Code : Console

```
$ w
16:50:30 up 8:50, 2 users, load average: 0,08, 0,34, 0,31
USER      TTY      FROM          LOGIN@   IDLE   JCPU   PCPU   WHAT
mateo21   :0      -             19Apr08  ?xdm?
          3:38m  1.18s /usr/bin/gnome-
mateo21   pts/0   :0.0          16:49    0.00s  0.33s  0.03s  w
```

Bon, à première vue c'est court mais dense, ça n'a pas l'air très clair.

Pourtant, cette commande nous donne en fait un condensé d'informations très utiles que je vais vous présenter dans l'ordre, de gauche à droite et de haut en bas.

L'heure (aussi accessible via "date")

Ici, l'heure qui nous est donnée est **16:50:30** (16h 50m 30s).

Cette information est aussi accessible depuis la commande **date** qui nous donne... la date, l'heure et le décalage horaire.

Code : Console

```
$ date
jeudi 16 octobre 2008, 17:26:27 (UTC+0200)
```

La commande `date` permet en outre de modifier la date enregistrée dans l'ordinateur. C'est un peu particulier et pas très intéressant, nous ne verrons donc pas comment le faire ici (mais il vous suffit de lire le manuel si vous en avez vraiment besoin).

L'uptime (aussi accessible via "uptime")

Dans notre exemple plus haut, l'information d'uptime est la suivante : `up 8 : 50`. C'est la durée de fonctionnement de l'ordinateur.



L'uptime peut aussi être obtenu via la commande `uptime`.

En soi, cette information n'a pas l'air très utile mais elle permet quand même de savoir depuis combien de temps l'ordinateur travaille et donc depuis combien de temps il n'a pas été redémarré.

Notez que, contrairement à Windows, il est extrêmement rare que l'installation d'un programme nous réclame de redémarrer l'ordinateur. En fait, vous avez besoin de redémarrer principalement quand vous mettez à jour le noyau (le cœur) de Linux. Autrement, il n'est jamais nécessaire de redémarrer l'ordinateur entièrement.

Ce mode de fonctionnement est particulièrement adapté sur les serveurs qui, par définition, sont des machines qui doivent être tout le temps allumées pour servir les gens qui en ont besoin. Par exemple, les serveurs du Site du Zéro qui vous délivrent les pages du site 24h/24 7j/7 sont tout le temps allumés et nous n'avons pratiquement jamais besoin de les redémarrer. Pour preuve, l'uptime de notre serveur Lisa au moment où j'écris ces lignes :

Code : Console

```
$ uptime
17:45:58 up 211 days, 15:24, 1 user, load average: 2.44, 2.66, 2.28
```

Notre serveur est en fonctionnement depuis 211 jours. Il n'a pas eu besoin d'être redémarré depuis. Cela témoigne notamment de la robustesse de Linux et de sa capacité à "tenir le coup" pendant très longtemps.

La charge (aussi accessible via "uptime" et "load")

En haut à droite de notre exemple, nous avons la charge. Ce sont 3 valeurs décimales :

```
load average: 0,08, 0,34, 0,31
```

La charge est un indice de l'activité de l'ordinateur. Il y a 3 valeurs :

1. La première correspond à la charge moyenne depuis 1 minute (0,08).
2. La seconde à la charge moyenne depuis 5 minutes (0,34).
3. La dernière à la charge moyenne depuis 15 minutes (0,31).



Qu'est-ce que ce nombre représente ?

C'est un peu compliqué. Si vous voulez vraiment savoir, la doc nous dit qu'il s'agit du nombre moyen de processus (programmes) qui sont en train de tourner et qui réclament l'utilisation du processeur.

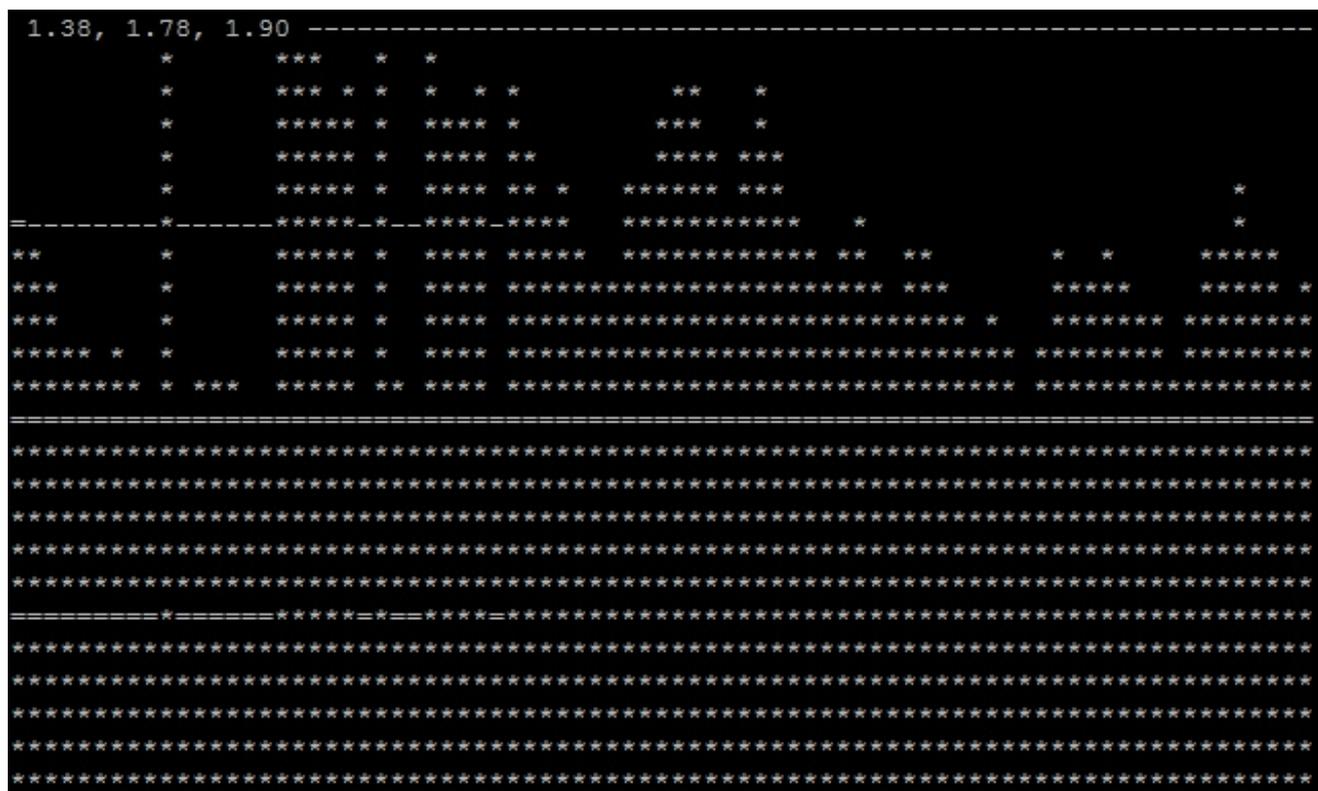
Cela veut dire que, depuis une minute, il y a en moyenne 0,33 processus qui réclament le processeur. Votre processeur est donc actif 33% du temps.

Mais ce nombre dépend du nombre de processeurs dans votre ordinateur. Un ordinateur *dual core* ne sera complètement chargé que lorsque la valeur aura atteint 2. Pour un *quad core* (4 coeurs de processeur), la valeur maximale avant surcharge sera de 4.

Bref, rien ne vous oblige à savoir ce que ce nombre signifie. Vous avez juste besoin de savoir que, lorsqu'il dépasse 1 (si vous avez un processeur), 2 ou 4, alors votre ordinateur est surchargé. J'ai déjà vu des machines avec une charge de 60, et même plus !

Quand la charge est très élevée pendant une longue période, c'est qu'il y a clairement un problème. Il y a trop de programmes qui réclament le processeur et quelque chose ne va pas dans l'ordinateur. Celui-ci aura du mal à répondre en cas de forte charge.

Notez que vous pouvez obtenir un graphique de l'évolution de la charge en console via la commande `tlload`. Le graphe évolue au fur et à mesure du temps, il faut patienter un petit peu avant d'avoir quelque chose :



Vous pouvez quitter le graphe avec Ctrl + C.

La liste des connectés (aussi accessible via "who")

Enfin, le tableau en bas qui nous est donné par `w` est surtout intéressant sur un serveur (une machine partagée par plusieurs utilisateurs). Il donne la liste des personnes connectées sur la machine, ce qu'ils sont en train de faire et depuis combien de temps.

Code : Console

```

USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
mateo21   :0      -               19Apr08  ?xdm?
          3:38m  1.18s /usr/bin/gnome-
mateo21   pts/0   :0.0            16:49   0.00s  0.33s  0.03s  w

```

Là, j'étais sur mon ordinateur personnel sous Ubuntu. Je ne l'ai pas configuré pour qu'on puisse se connecter dessus depuis internet (comme vous certainement), ce qui explique pourquoi je suis seul.

Certes, j'apparais 2 fois. Nous allons comprendre pourquoi lorsque nous aurons appris à lire le tableau.

Il n'est pas nécessaire de décrire chacune des colonnes. Sachez qu'en gros vous avez :

- **USER** : le nom de l'utilisateur (son login)
- **TTY** : le nom de la console dans laquelle se trouve l'utilisateur. Souvenez-vous que sous Linux il y a en général 6 consoles (tty1 à tty6) et qu'en plus de ça on peut en ouvrir une infinité grâce aux consoles graphiques (leur nom commence par pts en général), comme le propose le programme "Terminal" sous Gnome ou "Konsole" sous KDE.
- **FROM** : c'est l'adresse IP (ou le nom d'hôte) depuis laquelle il se connecte. Ici comme je me suis connecté en local (sur ma propre machine, sans passer par internet), il n'y a pas vraiment d'IP.
- **LOGIN@** : l'heure à laquelle cet utilisateur s'est connecté.
- **IDLE** : depuis combien de temps cet utilisateur est inactif (depuis combien de temps il n'a pas lancé de commande).
- **WHAT** : la commande qu'il est en train d'exécuter en ce moment. En général, si vous voyez "bash" cela signifie qu'il a juste un invite de commandes ouvert (il n'exécute donc pas de commande particulière).

Dans mon cas, on voit donc 2 utilisateurs (2 fois moi). Le premier correspond à la session "graphique" : on le devine notamment grâce à la dernière colonne WHAT qui indique que cet utilisateur est en train d'exécuter l'environnement graphique gnome.

L'autre utilisateur est sur une console (ici une console "graphique" lancée depuis gnome). Cet utilisateur est en train d'exécuter... la commande w ! En effet, lorsque je lance w je me "vois" en train de l'exécuter dans la liste des utilisateurs connectés, c'est parfaitement normal. 😊

ps & top : lister les processus

La commande `w` nous a permis de faire rapidement le point sur l'état du système. Allons plus loin maintenant : nous allons apprendre à lister les processus qui tournent sur votre machine.

Pour faire simple, dites-vous qu'un **processus** est un programme qui tourne en mémoire. La plupart des programmes ne font tourner qu'un processus en mémoire (une seule version d'eux-mêmes). C'est le cas de Firefox par exemple. D'autres lancent des copies d'eux-mêmes, c'est le cas du navigateur Google Chrome qui crée autant de processus en mémoire que d'onglets ouverts.



Sur un serveur web, on utilise en général le logiciel Apache qui délivre les pages web aux internautes. Ce logiciel crée beaucoup de processus pour séparer ses activités. Il en va de même pour les systèmes de gestion de base de données, comme MySQL et PostgreSQL.

Il ne faut pas s'inquiéter si un programme génère beaucoup de processus, cela n'est en général pas anormal.

Si vous faites la liste des processus qui tournent sur votre machine, vous risquez d'être surpris. Vous en reconnaîtrez certains, mais vous en verrez beaucoup d'autres qui ont été lancés par le système d'exploitation et dont vous n'avez jamais eu connaissance.

Pour lister les processus qui tournent sous Windows, on utilise `Ctrl + Alt + Suppr` et on va dans l'onglet "Processus". Sous Linux, on peut utiliser 2 commandes différentes : `ps` et `top`.

ps : liste des processus statique

`ps` vous permet d'obtenir la liste des processus qui tournent au moment où vous lancez la commande. Cette liste n'est pas rafraîchie en temps réel, contrairement à `top` qu'on verra plus tard.

Essayons d'utiliser `ps` sans paramètres :

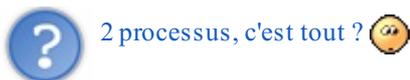
Code : Console

```
$ ps
  PID TTY          TIME CMD
 23720 pts/0    00:00:01 bash
 29941 pts/0    00:00:00 ps
```

On distingue 4 colonnes :

- **PID** : c'est le numéro d'identification du processus. Chaque processus a un numéro unique qui permet de l'identifier. Ce numéro nous sera utile plus tard lorsque nous voudrons arrêter le processus.
- **TTY** : c'est le nom de la console depuis laquelle a été lancé le processus.
- **TIME** : la durée d'exécution du processus. Plus exactement, cela correspond à combien de temps le processus a occupé le processeur depuis son lancement.
- **CMD** : le programme qui a généré ce processus. Si vous voyez plusieurs fois le même programme, c'est que celui-ci s'est dupliqué en plusieurs processus (c'est le cas de MySQL par exemple).

Dans mon cas, on distingue 2 processus : bash (qui correspond à l'invite de commandes qui gère les commandes) et ps que je viens de lancer.



En fait, quand on utilise ps sans arguments comme on vient de le faire, il affiche seulement les processus lancés par le même utilisateur (ici "mateo21") dans la même console (ici pts/0). Cela limite énormément les processus affichés, car beaucoup sont lancés par root (l'utilisateur "administrateur" de la machine) et ne sont pas lancés depuis la même console que vous.

La commande ps vous permet d'utiliser énormément d'options. Regardez le manuel pour avoir une petite idée de tout ce que vous pouvez faire avec, vous allez prendre peur. 😬

Plutôt que de faire une longue liste des paramètres possibles, je vous propose quelques combinaisons de paramètres utiles à retenir.

ps -ef : lister tous les processus

Avec ps -ef, vous pouvez avoir la liste de tous les processus lancés par tous les utilisateurs sur toutes les consoles :

Code : Console

```
$ ps -ef
UID      PID  PPID  C  STIME TTY          TIME CMD
root      1    0  0  01:01 ?           00:00:01 /sbin/init
root      2    1  0  01:01 ?           00:00:00 [migration/0]
root      3    1  0  01:01 ?           00:00:00 [ksoftirqd/0]
root      4    1  0  01:01 ?           00:00:00 [watchdog/0]
root      5    1  0  01:01 ?           00:00:00 [events/0]
root      6    1  0  01:01 ?           00:00:00 [khelper]
root      7    1  0  01:01 ?           00:00:00 [kthread]
root     30    7  0  01:01 ?           00:00:00 [kblockd/0]
root     31    7  0  01:01 ?           00:00:00 [kacpid]
root     32    7  0  01:01 ?           00:00:00 [kacpi_notify]
root     93    7  0  01:01 ?           00:00:00 [kseriod]
root    118    7  0  01:01 ?           00:00:04 [pdflush]
root    119    7  0  01:01 ?           00:00:00 [pdflush]
root    120    7  0  01:01 ?           00:00:01 [kswapd0]
root    121    7  0  01:01 ?           00:00:00 [aio/0]
root   1930    7  0  01:01 ?           00:00:00 [ksuspend_usbd]
root   1931    7  0  01:01 ?           00:00:00 [khubd]
root   2061    7  0  01:01 ?           00:00:00 [ata/0]
root   2062    7  0  01:01 ?           00:00:00 [ata_aux]
root   2094    7  0  01:01 ?           00:00:00 [scsi_eh_0]
root   2263    7  0  01:01 ?           00:00:09 [kjournald]
root   2462    1  0  01:01 ?           00:00:00 /sbin/udevd --daemon
root   3292    7  0  01:01 ?           00:00:00 [kpsmoused]
root   3448    7  0  01:01 ?           00:00:00 [kgameportd]
root   4021    1  0  01:02 tty4       00:00:00 /sbin/getty 38400 tty4
root   4022    1  0  01:02 tty5       00:00:00 /sbin/getty 38400 tty5
root   4024    1  0  01:02 tty2       00:00:00 /sbin/getty 38400 tty2
root   4027    1  0  01:02 tty3       00:00:00 /sbin/getty 38400 tty3
root   4030    1  0  01:02 tty1       00:00:00 /sbin/getty 38400 tty1
```

```

root      4040      1  0 01:02 tty6      00:00:00 /sbin/getty 38400 tty6
root      4266      1  0 01:02 ?              00:00:00 /usr/sbin/acpid -
c /etc/acpi/eve
root      4363      1  0 01:02 ?              00:00:00 /sbin/syslogd
root      4417      1  0 01:02 ?
00:00:00 /bin/dd bs 1 if /proc/kmsg of /v
klog      4419      1  0 01:02 ?              00:00:00 /sbin/klogd -
P /var/run/klogd/km
103       4440      1  0 01:02 ?              00:00:00 /usr/bin/dbus-daemon --
system
107       4456      1  0 01:02 ?              00:00:03 /usr/sbin/hald

...

```

Il y en a vraiment beaucoup, je n'ai pas recopié la liste complète ici. 😊

Vous noterez l'apparition de la colonne UID (User ID) qui indique le nom de l'utilisateur qui a lancé la commande. Il y en a beaucoup lancés par root automatiquement au démarrage de la machine dont vous n'avez jamais entendu parler.

ps -ejH : afficher les processus en arbre

Cette option intéressante vous permet de regrouper les processus dans une forme arborescente. Plusieurs processus sont des "enfants" d'autres processus, cela vous permet de savoir qui est à l'origine de quel processus.

Code : Console

```

$ ps -ejH
  PID  PGID  SID  TTY      TIME  CMD
    1     1     1  ?        00:00:01  init
    2     1     1  ?        00:00:00  migration/0
    3     1     1  ?        00:00:00  ksoftirqd/0
    4     1     1  ?        00:00:00  watchdog/0
    5     1     1  ?        00:00:00  events/0
    6     1     1  ?        00:00:00  khelper
    7     1     1  ?        00:00:00  kthread
   30     1     1  ?        00:00:00  kblockd/0
   31     1     1  ?        00:00:00  kacpid
   32     1     1  ?        00:00:00  kacpi_notify
   93     1     1  ?        00:00:00  kseriod
  118     1     1  ?        00:00:04  pdflush
  119     1     1  ?        00:00:00  pdflush
  120     1     1  ?        00:00:01  kswapd0
  121     1     1  ?        00:00:00  aio/0
 1930     1     1  ?        00:00:00  ksuspend_usbd
 1931     1     1  ?        00:00:00  khubd
 2061     1     1  ?        00:00:00  ata/0
 2062     1     1  ?        00:00:00  ata_aux
 2094     1     1  ?        00:00:00  scsi_eh_0
 2263     1     1  ?        00:00:09  kjournald
 3292     1     1  ?        00:00:00  kpsmoused
 3448     1     1  ?        00:00:00  kgameportd
 4521  4521  4521  ?        00:00:00  NetworkManager
 4538  4538  4538  ?        00:00:01  avahi-daemon
 4539  4539  4539  ?        00:00:00  avahi-daemon
 4556  4556  4556  ?        00:00:00  NetworkManagerD
 4569  4569  4569  ?        00:00:00  system-tools-ba
 4570  4569  4569  ?        00:00:00  dbus-daemon
 4593  4593  4593  ?        00:00:00  gdm
 4594  4594  4593  ?        00:00:00  gdm
 4625  4625  4625  tty7     00:05:56  Xorg
 5012  5012  5012  ?        00:00:01  gnome-session
 5057  5057  5057  ?        00:00:00  ssh-agent
 5080  5012  5012  ?        00:00:25  metacity

```

```

5083  5012  5012  ?           00:00:16          gnome-panel
5089  5012  5012  ?           00:00:31          nautilus
5098  5012  5012  ?           00:00:01          update-notifier
5102  5012  5012  ?           00:00:01          evolution-alarm
5107  5012  5012  ?           00:00:02          nm-applet
5112  5012  5012  ?           00:01:18          gnome-cups-icon
4640  4640  4640  ?           00:00:05          cupsd
4672  4672  4672  ?           00:00:00          hpiod

```

Dans cette liste, vous pouvez voir que kthread (ici surligné) a lancé lui-même de nombreux processus, comme kacpid, pdflush... Certains processus que nous connaissons mieux nous apprennent des choses sur l'organisation du système. C'est le cas ici de gdm (Gnome Desktop Manager) qui gère tout le bureau Gnome. Il a lancé Xorg qui gère l'environnement graphique, mais aussi gnome-session qui gère tous les programmes que vous avez lancés durant cette session comme nautilus (l'explorateur de fichiers), gnome-panel qui est plus ou moins l'équivalent de la barre des tâches de Windows, etc.

ps -u UTILISATEUR : lister les processus lancés par un utilisateur

Pour filtrer un peu cette longue liste, on peut utiliser -u afin d'obtenir par exemple uniquement les processus que l'on a lancé nous-même.

Code : Console

```

$ ps -u mateo21
PID TTY          TIME CMD
5012 ?           00:00:01 gnome-session
5057 ?           00:00:00 ssh-agent
5060 ?           00:00:00 dbus-launch
5061 ?           00:00:00 dbus-daemon
5063 ?           00:00:03 gconfd-2
5066 ?           00:00:00 gnome-keyring-d
5068 ?           00:00:03 gnome-settings-
5075 ?           00:00:00 sh
5076 ?           00:00:00 esd
5080 ?           00:00:25 metacity
5083 ?           00:00:16 gnome-panel
5089 ?           00:00:31 nautilus

```

Ici, j'obtiens uniquement les processus lancés par l'utilisateur mateo21, ce qui filtre déjà pas mal les autres processus "système" lancés par root.

top : liste des processus dynamique

La liste donnée par `ps` a un défaut : elle est *statique* (elle ne bouge pas). Or, votre ordinateur, lui, est en perpétuel mouvement. De nombreux processus apparaissent et disparaissent régulièrement.

Comment avoir une liste régulièrement mise à jour ? Avec la commande `top` !

Essayez-la :

Code : Console

```

top - 13:31:30 up 12:30,  3 users,  load average: 0.01, 0.07, 0.11
Tasks:  96 total,   3 running,  93 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.8%us,  0.6%sy,  0.0%ni, 97.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    515984k total,  453652k used,   62332k free,   69036k buffers
Swap:   240932k total,  31496k used,   209436k free,   246404k cached

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4625	root	15	0	38572	14m	6676	R	1.2	2.9	6:01.00	Xorg
5068	mateo21	15	0	29760	9.8m	8008	S	0.6	1.9	0:03.69	gnome-settings-
5112	mateo21	15	0	48612	8440	6844	S	0.6	1.6	1:19.45	gnome-cups-icon
1	root	18	0	2908	1848	524	S	0.0	0.4	0:01.50	init
2	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0
4	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
5	root	10	-5	0	0	0	S	0.0	0.0	0:00.66	events/0
6	root	10	-5	0	0	0	S	0.0	0.0	0:00.02	khelper
7	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
30	root	10	-5	0	0	0	S	0.0	0.0	0:00.55	kblockd/0
31	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
32	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	kacpi_notify
93	root	10	-5	0	0	0	S	0.0	0.0	0:00.02	kseriod
118	root	15	0	0	0	0	S	0.0	0.0	0:04.84	pdflush
119	root	15	0	0	0	0	S	0.0	0.0	0:00.20	pdflush
120	root	10	-5	0	0	0	S	0.0	0.0	0:01.29	kswapd0

Cette liste est **interactive** et régulièrement mise à jour.

En haut, vous retrouvez l'uptime et la charge, mais aussi la quantité de processeur et de mémoire utilisée. Nous ne rentrerons pas dans les détails à ce niveau car cela demanderait un peu trop d'explications avancées sur le fonctionnement du système d'exploitation. Néanmoins, si vous savez lire la charge et la mémoire disponible, vous pouvez déjà vous faire une idée de ce qui se passe.

En-dessous, vous avez la liste des processus.



Pourquoi y a-t-il si peu de processus ?

top ne peut pas afficher tous les processus à la fois, il ne conserve que les premiers pour qu'ils tiennent sur une "page" de la console.

Par défaut, les processus sont triés par taux d'utilisation du processeur (colonne %CPU). Les processus que vous voyez tout en haut de cette liste sont donc ceux qui sont actuellement les plus gourmands en processeur. Ce sont peut-être eux que vous devriez cibler en premier si vous sentez que votre système est surchargé.

On navigue à l'intérieur de ce programme en appuyant sur certaines touches du clavier. En voilà au moins 2 à connaître :

- **q** : ferme top.
- **h** : affiche l'aide, et donc la liste des touches utilisables.



Attention à la différence entre majuscules et minuscules ! Taper "h" n'a pas le même effet que de taper "H" !

Mise à part cela, voici quelques commandes à connaître au sein de top qui peuvent vous être utiles :

- **B** : met en gras certains éléments.
- **f** : ajoute ou supprime des colonnes dans la liste.
- **F** : change la colonne selon laquelle les processus sont triés. En général, laisser le tri par défaut en fonction de %CPU est suffisant.
- **u** : filtre en fonction de l'utilisateur que vous voulez.
- **k** : tue un processus, c'est-à-dire arrête ce processus. Ne vous inquiétez pas, en général les processus ne souffrent pas. On vous demandera le numéro (PID) du processus que vous voulez tuer. Nous reviendrons sur l'arrêt des processus un peu plus loin.
- **s** : change l'intervalle de temps entre chaque rafraîchissement de la liste (par défaut c'est toutes les 3 secondes).

Vous voilà prêts à utiliser top ! 😊

Je l'utilise principalement pour voir la charge évoluer régulièrement tout en surveillant les processus les plus gourmands qui peuvent poser un problème.

ctrl+c & kill : arrêter un processus

Parfois, rien ne va plus. Un processus s'emballe et ne veut pas s'arrêter. Cela arrive partout, même sur Linux. A la différence de Windows toutefois, vous ne devriez pas avoir le réflexe de redémarrer "pour que ça aille mieux". Tout peut être résolu en arrêtant les processus qui vous gênent et en les relançant au besoin.

Il y a plusieurs façons d'arrêter un processus, nous allons les étudier ici.

Ctrl + C : arrêter un processus lancé en console

La combinaison de touche Ctrl + C est à connaître. Cela demande (gentiment) l'arrêt du programme console en cours d'exécution à l'écran.

Prenez une commande qui n'en finit plus, comme par exemple un `find` sur l'ensemble du disque. Celui-ci va analyser tout votre disque dur à la recherche du fichier demandé. Si vous trouvez cela trop long et que vous voulez arrêter le programme en cours de route, il vous suffit de taper Ctrl + C :

Code : Console

```
# find / -name "*log*"
/dev/log
/bin/login
/sys/module/scsi_mod/parameters/scsi_logging_level
/sys/module/ehci_hcd/parameters/log2_irq_thresh
```

La liste aurait dû être beaucoup plus longue. Mais j'ai demandé l'arrêt du programme avec Ctrl + C, ce qui fait que j'ai pu "retrouver" l'invite de commande rapidement et facilement.

Taper Ctrl + C ne coupe pas le programme brutalement, cela lui demande gentiment de s'arrêter, comme si vous aviez cliqué sur la croix pour fermer une fenêtre.

kill : tuer un processus

Ctrl + C ne marche que sur un programme actuellement ouvert dans la console. De nombreux programmes tournent pourtant en arrière-plan, et Ctrl + C n'aura aucun effet sur eux.

C'est là que vous devez utiliser `kill` si vous voulez les arrêter (on dit aussi "tuer", c'est pareil même si ça a l'air violent 😊).

Pour vous en servir, il faudra auparavant récupérer le PID du ou des processus que vous voulez tuer. Pour cela, 2 solutions :

- ps
- top

Ces 2 commandes que nous venons de voir vous indiquent le PID (numéro d'identification) de chaque processus. Par exemple avec ps :

Code : Console

```
$ ps -u mateo21
PID TTY          TIME CMD
5012 ?            00:00:01 gnome-session
```

```
5057 ?      00:00:00 ssh-agent
5060 ?      00:00:00 dbus-launch
5061 ?      00:00:00 dbus-daemon
5063 ?      00:00:03 gconfd-2
5066 ?      00:00:00 gnome-keyring-d
5068 ?      00:00:03 gnome-settings-
5075 ?      00:00:00 sh
5076 ?      00:00:00 esd
5080 ?      00:00:26 metacity
5083 ?      00:00:17 gnome-panel

...

25227 pts/1   00:00:00 bash
32617 pts/1   00:00:00 man
32627 pts/1   00:00:00 pager
32703 pts/0   00:00:00 ps
```

Supposons qu'on souhaite arrêter Firefox. On peut filtrer cette longue list avec grep et un pipe que nous avons appris à utiliser.



Code : Console

```
$ ps -u mateo21 | grep firefox
32678 ?      00:00:03 firefox-bin
```

Hop là, on a filtré Firefox de cette longue liste et on a même récupéré son PID. Il ne nous reste plus qu'à le tuer, avec la commande suivante :

Code : Console

```
kill 32678
```

Si tout va bien, la commande ne renvoie rien. Sinon, une erreur devrait s'afficher dans la console.

Vous pouvez aussi tuer plusieurs processus d'un seul coup en indiquant plusieurs PID de suite :

Code : Console

```
kill 32678 2768 33071
```



Attention : même si kill est par défaut une commande "gentille" qui demande simplement au processus de s'arrêter, évitez de tuer des processus que vous ne connaissez pas. Beaucoup d'entre eux sont essentiels au bon fonctionnement de votre système, surtout ceux qui ont été lancés par root.



J'ai essayé, mais Firefox a l'air vraiment complètement planté et il refuse de s'arrêter. Il n'y a pas moyen d'être un peu plus... direct ?

Vous voulez tuer un processus sans lui laisser le choix?

C'est tout à fait possible, mais à n'utiliser que dans le cas d'un programme complètement planté que vous voulez vraiment arrêter !

Avec `kill -9` (comme le chiffre 9 oui oui) vous demandez à Linux de tuer le processus sans lui laisser le temps de s'arrêter proprement. Cela peut faire le ménage quand rien ne va plus.

Code : Console

```
kill -9 32678
```

... tuera le processus n°32678 (Firefox dans mon cas) immédiatement sans lui laisser le temps de finir.

killall : tuer plusieurs processus

Souvenez-vous : je vous ai dit que certains programmes se dupliquaient en plusieurs processus. Si vous voulez arrêter l'ensemble de ces processus, comment faire ? Heureusement, vous avez des armes pour éradiquer cette vermine. 🤖

Vous pourriez, certes, tuer tous les processus en récupérant un à un leurs PID. Mais il y a plus rapide : `killall` ("tuez-les tous !").

Contrairement à `kill`, `killall` attend le nom du processus à tuer et non son PID.

Supposons que nous ayons 3 processus "find" en cours d'exécution que nous souhaitons arrêter.

Code : Console

```
$ ps -u mateo21 | grep find
675 pts/1    00:00:01 find
678 pts/2    00:00:00 find
679 pts/3    00:00:01 find
```

Pour tous les tuer, il faudra donc taper :

Code : Console

```
$ killall find
```

Si la commande ne renvoie rien, c'est que tout s'est bien passé.

En revanche, si vous avez :

Code : Console

```
$ killall find
find: aucun processus tué
```

... cela signifie qu'il n'y avait aucun processus de ce nom à tuer. Soit le processus n'est plus là, soit vous n'avez pas écrit correctement son nom. Vérifiez ce nom à nouveau avec la commande `ps`.

halt & reboot : arrêter et redémarrer l'ordinateur

Nous venons d'apprendre à arrêter des processus avec `kill`. Je pense que le moment est bien choisi pour découvrir comment arrêter et redémarrer l'ordinateur.

Comme je vous le disais plus tôt, il est assez rare que l'on soit forcé d'arrêter ou de redémarrer l'ordinateur. A moins d'avoir mis à jour le kernel (noyau) de Linux, il n'est jamais nécessaire de redémarrer complètement. L'arrêt ou le redémarrage d'un serveur sous Linux sont réellement des opérations exceptionnelles.



Mais, j'ai installé Linux sur mon ordinateur personnel. Je n'en fais pas un serveur. J'ai le droit de l'arrêter ou de le redémarrer quand même, non ? 🤔

En effet, et je suppose que vous n'avez pas attendu ce chapitre pour le faire. 😊

Vous pouviez en effet arrêter et redémarrer l'ordinateur via l'interface graphique (Gnome, KDE, ...). Mais en console, savez-vous le faire ?

halt : arrêter l'ordinateur

La commande `halt` commande l'arrêt immédiat de l'ordinateur. Il faut être root pour arrêter la machine, donc vous devrez taper :

Code : Console

```
$ sudo halt
```

Un message sera affiché dans la console pour annoncer l'arrêt de l'ordinateur.

reboot : redémarrer l'ordinateur

De même, il existe la commande `reboot` pour redémarrer l'ordinateur. Il faut à nouveau être root :

Code : Console

```
$ sudo reboot
```

Le redémarrage prend effet immédiatement.



Les commandes `halt` et `reboot` appellent en réalité la commande `shutdown` avec des paramètres spécifiques. N'hésitez pas à lire sa page de manuel, vous verrez que vous pouvez par exemple programmer un arrêt ou un redémarrage à une heure précise ou au bout d'un certain temps.

Vous voilà maintenant capables de savoir quels processus tournent sur votre machine, d'analyser la charge et éventuellement de prendre la décision d'en tuer un ou deux. 🤔

Soyez prudents tout de même, ne tuez pas des processus à tout-va. Faites-le uniquement lorsque c'est nécessaire.

Dans le prochain chapitre, nous allons aller plus loin dans notre étude des processus. Nous allons apprendre à les mettre en pause, en arrière-plan et à les relancer. Bref, non contents de les tuer, nous allons un peu plus les torturer. 😈

Exécuter des programmes en arrière-plan

Nous avons commencé à découvrir ce qu'étaient les processus dans le chapitre précédent. Nous savons désormais comment les lister, les trier, les filtrer et enfin comment les tuer.

Ici, je vous propose d'aller plus loin et découvrir *l'exécution en arrière-plan*. A priori, la console a quelque chose de frustrant : on a l'impression qu'on ne peut lancer qu'un seul programme à la fois par console. Or, c'est tout à fait faux ! ... mais encore faut-il savoir comment faire pour faire tourner des programmes en arrière-plan.

Il existe un certain nombre de techniques plus ou moins sophistiquées. Il est recommandé de les connaître car, parfois, on souhaite tout faire au sein d'une seule et même console.

"&" & nohup : lancer un processus en arrière-plan

Lorsque vous vous apprêtez à lancer une opération un peu longue, comme une grosse copie de fichiers par exemple, vous n'avez peut-être pas envie de patienter sagement le temps que la commande s'exécute pour pouvoir faire autre chose en attendant. Certes, on peut ouvrir une autre console me direz-vous. Il y a des cas cependant où on n'a accès qu'à une seule console, ou encore où on n'a tout simplement pas envie d'en ouvrir une autre (la flemme, vous connaissez ? 🤔).

Contrairement aux apparences, plusieurs programmes peuvent tourner en même temps au sein d'une même console. Ce n'est pas parce qu'on ne peut pas afficher plusieurs fenêtres comme dans un environnement graphique qu'on est bloqué à un seul programme à la fois ! Encore faut-il connaître les techniques qui permettent de lancer une commande en tâche de fond...

& : lancer un processus en arrière-plan

La première technique que je veux vous faire découvrir est très simple : elle consiste à rajouter le petit symbole "&" à la fin de la commande que vous voulez envoyer en arrière-plan.



Le symbole "&" s'appelle la "*et commerciale*" ou encore "*esperluette*". Il est présent sur la touche 1 sur un clavier AZERTY.

Prenons par exemple la commande `cp` qui permet de copier des fichiers.

Je vous propose de copier un gros fichier vidéo (ce qui prend en général du temps), comme ceci :

Code : Console

```
$ cp video.avi copie_video.avi &  
[1] 16504
```



Notez que l'espace avant le "&" à la fin n'est pas obligatoire.

On vous renvoie 2 informations :

- **[1]** : c'est le numéro du processus en arrière-plan dans cette console. Comme c'est le premier processus que nous envoyons en arrière plan, il prend le numéro 1.
- **16504** : c'est le numéro d'identification général du processus (le fameux PID dont on a déjà parlé). Cette information vous permet de tuer le processus avec `kill` si nécessaire.

Maintenant, vous ne voyez peut-être rien, mais le processus est bel et bien en train de tourner en "tâche de fond".

Si vous essayez de faire la même chose avec d'autres commandes, par exemple sur un `find`, vous risquez d'être surpris : les messages renvoyés par la commande s'affichent toujours dans la console ! Vous pouvez certes écrire du texte et lancer d'autres

commandes pendant ce temps (essayez), mais c'est un peu frustrant de voir ces messages apparaître dans la console !

Heureusement, vous savez maintenant rediriger la sortie pour ne pas être importuné :

Code : Console

```
$ find / -name "*log" > sortiefind &  
[1] 18191
```

Les résultats seront maintenant écrits dans le fichier sortiefind au lieu d'être affichés dans la console. De plus, la commande s'exécute en fond et ne nous importune plus.

Notez que pour être sûr de ne pas être dérangé du tout, vous devrez aussi rediriger les erreurs (par exemple avec `2>&1`), ce qui peut nous donner une jolie commande comme celle-ci :

Code : Console

```
$ find / -name "*log" > sortiefind 2>&1 &  
[1] 18231
```

Il reste toutefois un problème : le processus est "attaché" à votre console. Si vous fermez la console sur laquelle vous êtes, le processus sera tué et ne s'exécutera donc pas jusqu'au bout.

nohup : détacher le processus de la console

L'option "&", bien qu'assez couramment utilisée, a ce défaut non négligeable : le processus reste attaché à la console, ce qui veut dire que si la console est fermée ou que l'utilisateur se déconnecte, le processus sera automatiquement arrêté.

Si on veut que le processus continue, il faut lancer la commande via `nohup`. Cela s'utilise comme ceci :

Code : Console

```
nohup commande
```

Par exemple, voici ce que ça donne si on lance la copie via un nohup :

Code : Console

```
$ nohup cp video.avi copie_video.avi  
nohup: ajout à la sortie de `nohup.out'
```

La sortie de la commande est par défaut redirigée vers un fichier `nohup.out`. Aucun message ne risque donc d'apparaître dans la console.

D'autre part, la commande est maintenant immunisée contre la fermeture de la console. Elle continuera de courir quoiqu'il arrive (sauf si on lui envoie un kill bien sûr 😊).



nohup est très utile par exemple lorsque vous vous connectez à un serveur.

Imaginons que vous voulez lancer un programme (comme un serveur de jeu) : celui-ci s'arrêtera de fonctionner dès que vous vous serez déconnecté de la ligne de commande du serveur. Vous n'allez pas rester connecté juste pour que le programme continue à fonctionner ! Heureusement, nohup vous épargne ce problème. 😊

ctrl + z, jobs, bg & fg : passer un processus en arrière-plan

Voyons maintenant le problème différemment : vous avez lancé la commande sans penser à rajouter un petit "&" à la fin. Malheureusement, la commande prend beaucoup plus de temps à s'exécuter que ce que vous aviez prévu. Êtes-vous condamné à attendre qu'elle soit terminée pour reprendre la main sur l'invite de commandes ? Bien sûr que non !

Il y a une série de commandes et de raccourcis qu'il vous faut absolument connaître ! Nous allons les étudier un par un dès maintenant.

Ctrl + Z : mettre en pause l'exécution du programme

Reprenons le cas de notre grosse copie de fichiers. Cette fois, je suppose que vous l'avez lancée sans le petit symbole "&" :

Code : Console

```
$ cp video.avi video_copie.avi
```

Si vous n'avez pas de gros fichier sous la main pour faire le test, vous pouvez aussi faire un `top`.

Tapez maintenant Ctrl + Z pendant l'exécution du programme. Celui-ci va s'arrêter et vous allez immédiatement reprendre la main sur l'invite de commandes.

Code : Console

```
[1]+  Stopped                  top
mateo21@mateo21-desktop:~$
```

Vous noterez que nous avons plusieurs informations : le numéro du processus en arrière-plan (ici [1]), son état ("stopped"), et le nom de la commande qui a lancé ce processus.

Le processus est maintenant dans un état de pause. Il ne s'exécute pas mais reste en mémoire.

bg : passer le processus en arrière-plan (background)

Maintenant que le processus est en "pause" et qu'on a récupéré l'invite de commandes, tapez :

Code : Console

```
$ bg
[1]+ top &
```

C'est tout, pas besoin de paramètres.

Qu'est-ce que cela fait ? Cette commande la reprise du processus, mais cette fois en arrière-plan. Il continuera à s'exécuter à nouveau, mais en tâche de fond.

En résumé, si vous avez lancé une commande par erreur en avant-plan et que vous voulez récupérer l'invite de commande, il faudra faire dans l'ordre :

- **Ctrl + Z** : pour mettre en pause le programme et récupérer l'invite de commandes.
- **bg** : pour que le processus continue à tourner mais en arrière-plan.

jobs : connaître les processus qui tournent en arrière-plan

Vous pouvez envoyer autant de processus en arrière-plan que vous voulez au sein d'une même console :

- Soit en les lançant directement en arrière-plan avec un "&" à la fin de la commande,
- Soit en utilisant la technique du Ctrl + Z suivi de bg que vous venez d'apprendre.

Comment savoir maintenant quels sont les processus qui tournent en arrière-plan ? Vous pourriez, certes, recourir à la commande `ps`, mais celle-ci vous donnera tous les processus. C'est un peu trop.

Heureusement, il existe une commande qui liste uniquement les processus qui tournent en fond au sein d'une même console : `jobs`.

Code : Console

```
$ jobs
[1]-  Stopped                  top
[2]+  Stopped                  find / -name "*log*" > sortiefind 2>&1
```

Encore une fois, vous avez le numéro du processus qui tourne en fond (à ne pas confondre avec le PID), son état et son nom.

fg : reprendre un processus au premier plan (foreground)

La commande `fg` renvoie un processus au premier plan.

Code : Console

```
$ fg
```

Si vous avez un seul processus listé dans les jobs, c'est ce processus qui sera remis au premier plan.

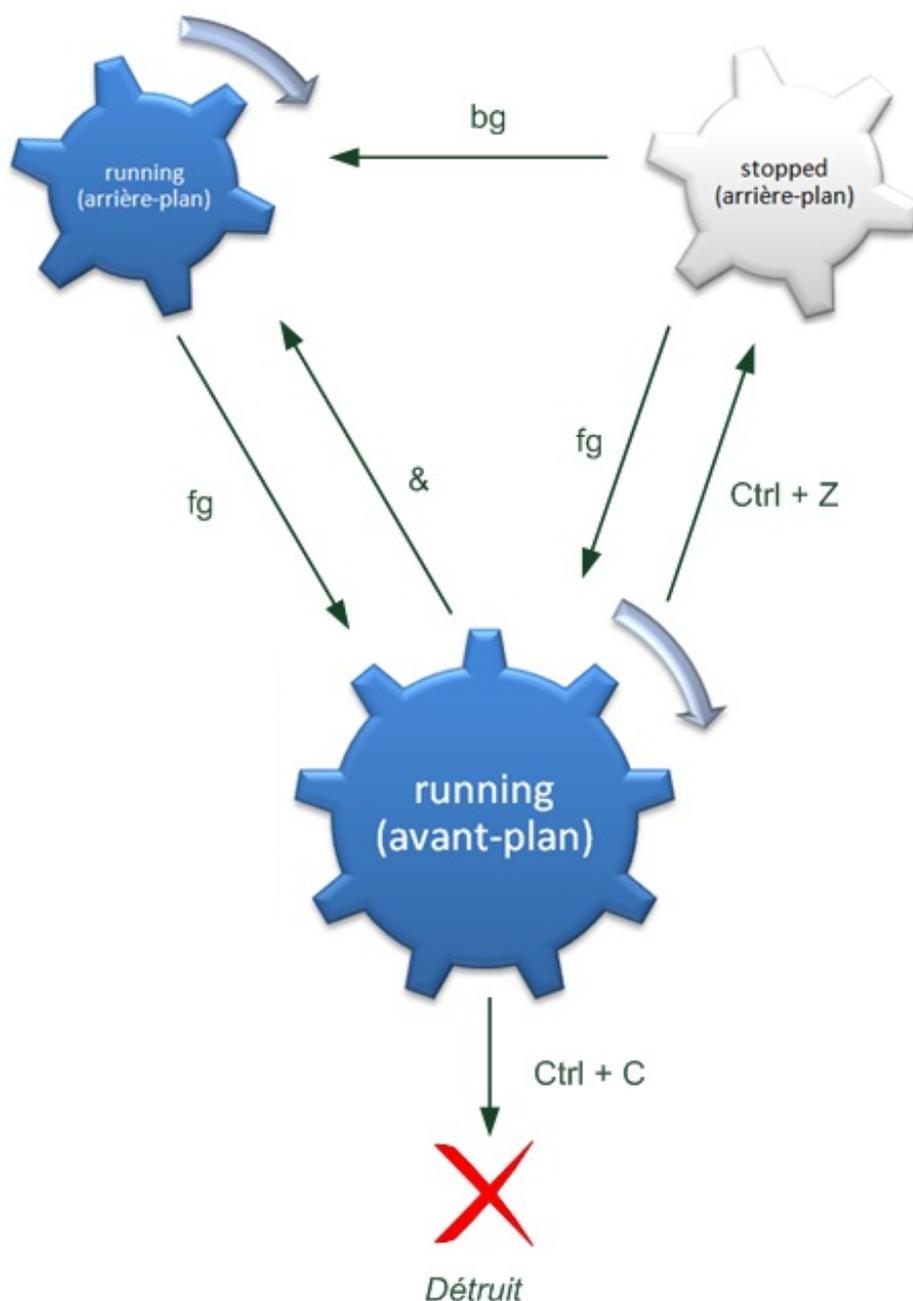
Si, comme moi tout à l'heure, vous aviez plusieurs processus en arrière-plan, il faudra préciser lequel vous voulez récupérer. Par exemple, voici comment reprendre le find qui était le job n°2 :

Code : Console

```
$ fg %2
```

Résumé des états possibles des processus

Je pense qu'un schéma s'impose maintenant. Dans celui-ci je résume tout ce que nous avons vu jusqu'ici, à l'exception de nohup qui est une commande un peu à part.



Expliquons un peu ce schéma !

Par défaut, un processus est lancé dans l'état "running" à l'avant-plan. On peut l'arrêter avec la combinaison Ctrl + C, auquel cas il sera détruit.

Mais on peut aussi l'envoyer en arrière-plan. Si on l'exécute dès le départ avec un "&", il sera à l'état "running" à l'arrière-plan. Si on choisit de faire Ctrl + Z, il passera à l'état stopped à l'arrière-plan. Il faudra taper "bg" pour le faire passer à nouveau à l'état "running" en arrière-plan.

Enfin, la commande fg renvoie un processus de l'arrière-plan vers l'avant-plan.

Prenez 5 minutes pour bien analyser ce schéma et vérifier que vous avez compris l'essentiel de ce chapitre, c'est vraiment important. Il résume à peu près tout ce qu'il faut savoir (à l'exception de nohup que j'ai mis à part comme je vous l'ai dit).

screen : plusieurs consoles en une

Il nous reste à découvrir une commande un peu particulière que j'ai volontairement réservée pour la fin : `screen`. Pourquoi j'ai attendu avant d'en parler ? Tout simplement parce que, contrairement à ce que nous avons vu jusqu'ici, ce n'est pas une commande "standard" qui est installée par défaut dans toutes les distributions Linux. Parfois, vous n'aurez pas accès à screen (parce que vous n'êtes pas root sur la machine) et il faudra vous débrouiller avec les commandes que l'on vient de voir précédemment.

Si, toutefois, vous êtes le maître de la machine (ce qui est votre cas si vous avez installé Linux chez vous), je peux vous recommander d'installer le programme screen.

Code : Console

```
$ sudo apt-get install screen
```

De quoi s'agit-il ?

screen est un multiplicateur de terminal. Derrière ce nom un peu pompeux qui peut faire peur je le reconnais, se cache en fait un programme qui est capable de gérer plusieurs consoles au sein d'une seule, un peu comme si chaque console était une fenêtre !



Imaginez que screen est un programme qui permet entre autres de faire une *mise en veille prolongée* de votre console, tout comme vous le faites peut-être avec votre ordinateur portable qui se retrouve exactement dans l'état où vous l'avez laissé en l'éteignant.

Concrètement, j'ai souvent tendance à utiliser screen sur un serveur. Cela me permet par exemple de lancer un serveur de jeu dans une console screen, de quitter le serveur, puis de revenir l'administrer plus tard au besoin en récupérant la console dans l'état où je l'ai laissée.

Lorsque vous avez installé screen, essayez-le en tapant tout simplement :

Code : Console

```
$ screen
```

Un message s'affiche pour indiquer que le programme est un logiciel libre ainsi que l'adresse e-mail de l'auteur où on peut lui envoyer, je cite "*des t-shirts, de l'argent, de la bière et des pizzas*". Bon... passons. 🤪

Code : Console

```
Screen version 4.00.03 (FAU) 23-Oct-06
```

```
Copyright (c) 1993-2002 Juergen Weigert, Michael Schroeder  
Copyright (c) 1987 Oliver Laumann
```

```
This program is free software; you can redistribute it and/or modify it under  
the terms of the GNU General Public License as published by the Free Software  
Foundation; either version 2, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful, but WITHOUT  
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS  
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License along with  
this program (see the file COPYING); if not, write to the Free Software  
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

```
Send bugreports, fixes, enhancements, t-shirts, money, beer & pizza to  
screen@uni-erlangen.de
```

```
[Press Space or Return to end.]
```

Tapez Entrée ou Espace pour passer ce message.

A première vue, il ne se passe rien de bien extraordinaire, on retrouve une console vide. Mais mine de rien, nous nous trouvons dans une console "émulée", pas dans la "vraie" console où nous étions tout à l'heure. Vous pouvez en sortir en tapant Ctrl + D ou exit, comme si vous quittiez une console normalement.

Vous retrouverez alors votre console habituelle où vous avez lancé screen :

Code : Console

```
mateo21@mateo21-desktop:~$ screen
[screen is terminating]
```

Bon, maintenant que vous savez sortir de screen, retournez-y. 😊

Il faut savoir que sous screen tout se fait avec des combinaisons de touches sous la forme suivante : Ctrl + a, suivi d'une autre touche. En fait, vous devez taper Ctrl + a, relâcher ces touches (levez les mains du clavier 🙄) et ensuite appuyer sur une autre touche.

Ctrl + a puis ? : afficher l'aide

Essayez de taper Ctrl + a, puis tapez "?". L'aide devrait alors s'afficher :

Code : Console

```
Screen key bindings, page 1 of 2.

                Command key: ^A   Literal ^A: a

break      ^B b      license      ,      removebuf   =
clear      C        lockscreen   ^X x      reset       Z
colon      :        log          H        screen      ^C c
copy       ^[ [      login       L        select      '
detach     ^D d      meta        a        silence     _
digraph    ^V        monitor     M        split       S
displays   *        next        ^@ ^N sp n suspend     ^Z z
dumftermcap .      number      N        time        ^T t
fit        F        only        Q        title       A
flow       ^F f      other       ^A       vbell       ^G
focus     ^I        pow_break   B        version     v
hardcopy   h        pow_detach  D        width       W
help       ?        prev        ^H ^P p ^? windows     ^W w
history    { }      quit        \        wrap        ^R r
info       i        readbuf     <       writebuf    >
kill       K k      redisplay   ^L l     xoff        ^S s
lastmsg    ^M m      remove     X        xon         ^Q q

                [Press Space for next page; Return to end.]
```

Il y a 2 pages de commandes. Avec Espace vous allez à la page suivante, avec Entrée vous refermez l'aide.

Comment lire cette page d'aide ? Par exemple, si vous voulez connaître la version du programme (milieu de la troisième colonne), il faudra taper Ctrl + a suivi de "v" (la lettre minuscule). Toutes les touches que vous voyez là doivent impérativement être précédées d'un Ctrl + a.

Notez par ailleurs que l'accent circonflexe "^" signifie ici "Ctrl".

Les principales commandes de screen

Je ne connais pas toutes ces commandes, mais je vais vous présenter les principales qui peuvent vous être utiles.

- **Ctrl + a puis c** : créer une nouvelle "fenêtre".
- **Ctrl + a puis w** : affiche la liste des "fenêtres" actuellement ouvertes. En bas de l'écran vous verrez apparaître par exemple : `0-$ bash 1*$ bash`. Cela signifie que vous avez 2 fenêtres ouvertes, l'une numérotée 0, l'autre 1. Celle sur laquelle vous vous trouvez actuellement contient une étoile * (on se trouve donc ici dans la fenêtre n°1).
- **Ctrl + a puis A** : renomme la fenêtre actuelle. Ce nom apparaît lorsque vous affichez la liste des fenêtres avec **Ctrl + a puis w**.
- **Ctrl + a puis n** : passer à la fenêtre suivante ("next").
- **Ctrl + a puis p** : passer à la fenêtre précédente ("previous").
- **Ctrl + a puis Ctrl + a** : revenir à la dernière fenêtre utilisée.
- **Ctrl + a puis un chiffre de 0 à 9** : passer à la fenêtre n°X.
- **Ctrl + a puis "** : choisir la fenêtre dans laquelle on veut aller.
- **Ctrl + a puis k** : ferme la fenêtre actuelle ("kill").



screen est sensible à la casse pour les commandes ! Faites donc bien la différence entre "c" et "C" par exemple.

Il nous reste 2 options très intéressantes de screen à découvrir qui méritent une attention particulière : split et detach.

Ctrl + a puis S : découper screen en plusieurs parties ("split")

Ctrl + a puis S coupe l'écran en 2 pour afficher 2 consoles à la fois ("split"). Il est possible de répéter l'opération plusieurs fois pour couper en 3, 4, ou plus (dans la mesure du possible, parce qu'après les consoles sont toutes petites 🤖).

Voici ce que vous voyez après avoir splitté l'écran une fois :

```
mateo21@mateo21-desktop:~$ ls
bin          Desktop      ies4linux-latest.tar.gz  tuto
copie_video.avi  Examples    Images                  video.avi
cours_unix.txt  ies4linux-2.99.0.1  sortiefind              video_copie.avi
mateo21@mateo21-desktop:~$ rm copie_video.avi
mateo21@mateo21-desktop:~$ █
```

```
0 bash
```

```
--
```

L'écran est bien découpé en 2, mais la "fenêtre" du bas est vide. Il n'y a même pas d'invite de commande.

Pour passer d'une fenêtre à une autre, faites **Ctrl + a puis Tab**.

Une fois le curseur placé dans la fenêtre du bas, vous pouvez soit créer une nouvelle fenêtre (**Ctrl + a puis c**) soit appeler une autre fenêtre que vous avez déjà ouverte (avec **Ctrl + a puis un chiffre** par exemple).

Vous pourrez par exemple afficher top pendant que vous faites des opérations sur la fenêtre du dessus :

```
mateo21@mateo21-desktop:~$ ls
bin          Desktop      ies4linux-latest.tar.gz  tuto
copie_video.avi  Examples    Images                   video.avi
cours_unix.txt  ies4linux-2.99.0.1  sortiefind               video_copie.avi
mateo21@mateo21-desktop:~$ rm copie_video.avi
mateo21@mateo21-desktop:~$
```

```
0 bash
top - 21:34:26 up 3:39, 3 users, load average: 0.15, 0.18, 0.12
Tasks: 99 total, 1 running, 97 sleeping, 1 stopped, 0 zombie
Cpu(s): 0.7%us, 1.0%sy, 0.0%ni, 98.0%id, 0.0%wa, 0.3%hi, 0.0%si, 0.0%st
Mem: 515984k total, 480984k used, 35000k free, 111412k buffers
Swap: 240932k total, 33820k used, 207112k free, 197040k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 5910 root        15   0  2444   984  784  S   0.7   0.2   0:17.45 vmware-guestd
 5680 root        15   0 41216  16m 6440  S   0.3   3.2   1:16.42 Xorg
20582 mateo21    15   0  2320  1144  880  R   0.3   0.2   0:00.05 top
     1 root        20   0  2912  1844  524  S   0.0   0.4   0:00.97 init

1 bash
```

La classe de geek quoi. 😎

Ah, et pour fermer une fenêtre que vous avez splittée, il faudra taper **Ctrl + a puis X**. Voilà, vous savez l'essentiel !

Ctrl + a puis d : détacher screen

Ctrl + a puis d détache screen et vous permet de retrouver l'invite de commandes "normale" sans arrêter screen. C'est peut-être une des fonctionnalités les plus utiles que nous devons approfondir, et cela nous ramène d'ailleurs à l'exécution de programmes en arrière-plan dont nous avons parlé au début du chapitre.

Concrètement, si vous détachez screen, vous retrouvez l'invite de commande classique :

Code : Console

```
mateo21@mateo21-desktop:~$ screen
[detached]
mateo21@mateo21-desktop:~$
```

L'information [detached] apparaît pour signaler que screen tourne toujours et qu'il est détaché de la console actuelle. Il continuera donc à tourner quoiqu'il arrive, même si vous fermez la console dans laquelle vous vous trouvez.



Ah, alors c'est comme nohup finalement non ?

En effet, screen se comporte comme un nohup. La différence est qu'une session screen vous permet d'ouvrir plusieurs fenêtres de consoles à la fois, contrairement à nohup qui ne peut lancer qu'un programme à la fois.

Vous pouvez donc partir, quitter la console et revenir récupérer votre session screen plus tard. Il faudra simplement taper :

Code : Console

```
$ screen -r
```

... pour retrouver votre session screen dans l'état où vous l'avez laissée.

Notez qu'il est possible de faire tourner plusieurs sessions screen en fond à la fois. Dans ce cas, `screen -r` ne sera pas suffisant car on vous demandera de préciser quelle session vous voulez récupérer :

Code : Console

```
$ screen -r
There are several suitable screens on:
    20930.pts-0.mateo21-desktop    (Detached)
    19713.pts-0.mateo21-desktop    (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
```

Pour récupérer la session n°20930, tapez simplement :

Code : Console

```
$ screen -r 20930
```

A noter aussi que `screen -ls` affiche la liste des screens actuellement ouverts :

Code : Console

```
$ screen -ls
There are screens on:
    20930.pts-0.mateo21-desktop    (Detached)
    19713.pts-0.mateo21-desktop    (Detached)
2 Sockets in /var/run/screen/S-mateo21.
```

Certaines personnes ont pris l'habitude de tout faire sur screen, notamment sur les serveurs. Il m'est arrivé de laisser tourner une session screen pendant plusieurs mois grâce à la possibilité de détachement que nous venons de découvrir.

Un fichier personnalisé de configuration de screen

Sans rentrer dans le détail car ce serait bien trop long, sachez qu'il est possible de personnaliser screen avec un fichier de configuration, comme la plupart des autres programmes sous Linux d'ailleurs.

Ce fichier s'appelle `.screenrc` et doit être placé dans votre home (par exemple `/home/mateo21`). Vous pouvez vous amuser à lire la doc à ce sujet, mais vous pouvez aussi utiliser le même fichier `.screenrc` que j'ai l'habitude d'utiliser :

[Télécharger le .screenrc personnalisé](#)



Ce fichier de configuration n'est pas de moi, merci donc à son auteur "bennyben".

Une fois placé dans votre home, exécutez screen. Vous devriez noter quelques différences :

```
mateo21@mateo21-desktop:~$ ls
bin          Examples    Images      video.avi
cours_unix.txt  ies4linux-2.99.0.1  sortiefind  video_copie.avi
Desktop      ies4linux-latest.tar.gz  tuto
```

mateo21@mateo21-desktop:~\$ █

```
12:13:54 mateo21-desktop 0,50 0,43 0,24 0$ bash 1-$ bash
```

Je trouve cette configuration plus pratique car on a toujours en bas l'heure, le nom de la machine sur laquelle on se trouve, la charge ainsi que la liste des fenêtres ouvertes. Après, libre à vous d'utiliser la configuration par défaut ou celle-là, dans tous les cas les commandes restent les mêmes. 😊

Vous avez désormais tous les outils en main pour tout faire au sein d'une seule et même console ! Si aujourd'hui on peut certes ouvrir plusieurs consoles à la fois, je mets ma main à couper (et pourtant j'en ai besoin pour écrire des tutoriels !) que vous aurez un jour besoin d'une des commandes que vous avez découvertes dans ce chapitre.

Si la combinaison Ctrl + Z, bg, fg et jobs nous permet de gérer la plupart des cas classiques, n'hésitez pas à utiliser screen si vous en avez la possibilité. C'est vraiment un programme très puissant qui mérite que l'on s'y intéresse.

Exécuter un programme à une heure différée

Nous savons pour le moment uniquement lancer une commande pour qu'elle s'exécute tout de suite. Il est cependant aussi possible de "retarder" son lancement.

Linux vous propose toute une série d'outils qui vous permettent de programmer à l'avance l'exécution d'une tâche, comme par exemple la *crontab* que nous allons étudier.

Tous les outils que nous allons découvrir dans ce chapitre feront en outre appel à la notion de date. Nous allons donc dans un premier temps nous intéresser au formatage de la date. 😊

date : régler l'heure

Nous commencerons par nous intéresser à la date et l'heure qu'il est, car tout dans ce chapitre tourne autour de la notion de date.



Je vous ai déjà présenté brièvement la commande `date`. Essayez-la à nouveau :

Code : Console

```
$ date
lundi 10 novembre 2008, 12:27:25 (UTC+0100)
```

Sans paramètre, la commande nous renvoie donc la date actuelle, l'heure et le décalage horaire.

Personnaliser l'affichage de la date

Si vous regardez le manuel (`man date`), vous verrez qu'il est possible de personnaliser l'affichage de la date : vous pouvez choisir quelles informations vous voulez afficher et dans quel ordre (vous pouvez par exemple ajouter les nanosecondes ou encore le numéro du siècle actuel).

Pour spécifier un affichage personnalisé, vous devez utiliser un symbole `+` suivi d'une série de symboles qui indiquent l'information que vous désirez. Je vous recommande de mettre le tout entre guillemets.

Prenons quelques exemples pour bien comprendre :

Code : Console

```
$ date "+%H"
12
```

Le `"+%H"` est le *format de date*. `%H` signifie "Le numéro de l'heure actuelle". Il était donc 12 heures au moment où j'ai lancé la commande.

Essayons autre chose d'un peu plus compliqué :

Code : Console

```
$ date "+%H:%M:%S"
12:36:15
```

Ici, j'ai rajouté les minutes (`%M`) et les secondes (`%S`).

J'ai séparé les nombres par des deux-points, mais j'aurais très bien pu mettre autre chose à la place :

Code : Console

```
$ date "+%Hh%Mm%Ss"  
12h41m01s
```

Seule la lettre qui suit le % est interprétée. Mes lettres "h", "m", et "s" sont donc simplement affichées.



Mais, comment tu sais que %M affiche le nombre de minutes par exemple ? 🤔

Je lis le man de date, tout simplement. 😊

C'est là que j'apprends comment afficher l'année par exemple :

Code : Console

```
$ date "+Bienvenue en %Y"  
Bienvenue en 2008
```

A vous de jouer ! 😊

Modifier la date

La commande `date` permet aussi de changer la date.



Attention, il faudra être root pour faire cela (vous devrez placer un `sudo` devant par exemple).

Il faut préciser les informations sous la forme suivante : `MMDDhhmmYYYY`. Les lettres signifient :

- MM : mois
- DD : jour
- hh : heure
- mm : minute
- YYYY : année

Notez qu'il n'est pas obligatoire de préciser l'année. On peut donc écrire :

Code : Console

```
$ sudo date 11101250  
lundi 10 novembre 2008, 12:50:00 (UTC+0100)
```

La nouvelle date s'affiche automatiquement et est mise à jour sur le système. Attention à bien respecter l'ordre des nombres : Mois - Jour - Heure - Minutes.

at : exécuter une commande plus tard

Vous souhaitez qu'une commande soit exécutée plus tard ? Pas de problème ! Il est possible de programmer l'exécution d'une commande avec `at`.



Avec `at`, le programme ne sera exécuté qu'une seule fois. Si vous voulez que l'exécution soit répétée régulièrement, il



faudra utiliser la `crontab` que nous verrons plus loin.

Exécuter une commande à une heure précise

La commande s'utilise en 2 temps :

1. Vous indiquez à quel moment (quelle heure, quel jour) vous désirez que la commande soit exécutée.
2. Vous tapez ensuite la commande que vous voulez voir exécutée à l'heure que vous venez d'indiquer.

Il faut donc d'abord indiquer à quelle heure vous voulez exécuter votre commande, sous la forme "HH:MM" :

Code : Console

```
$ at 14:17
```

Cela demande à exécuter des commandes à 14h17 aujourd'hui. Si vous tapez cela dans votre console, vous devriez voir ceci s'afficher :

Code : Console

```
$ at 14:17
warning: commands will be executed using /bin/sh
at>
```

`at` comprend que vous voulez exécuter des commandes à 14h17 et vous demande lesquelles. C'est pour cela qu'un *prompt* est affiché : on vous demande de taper les commandes que vous voulez exécuter à cette heure-là.

Pour cet exemple, nous allons demander de créer un fichier à 14h17 :

Code : Console

```
$ at 14:17
warning: commands will be executed using /bin/sh
at> touch fichier.txt
at> <EOT>
job 5 at Mon Nov 10 14:17:00 2008
```

Après avoir écrit la commande `touch`, `at` affiche à nouveau un prompt et vous demande une autre commande. Vous pouvez indiquer une autre commande à exécuter à la même heure... ou bien arrêter là. Dans ce cas, tapez `Ctrl + D` (comme si vous cherchiez à sortir d'un terminal). Le symbole `<EOT>` devrait alors s'afficher, et "at" s'arrêtera.

`at` affiche ensuite le numéro associé à cette tâche (à ce "job" comme il dit) et l'heure à laquelle il sera exécuté.

Attendez 14h17, et vous verrez que le fichier sera créé. 😊



Et si je veux exécuter la commande demain à 14h17 et non pas aujourd'hui ?

Code : Console

```
$ at 14:17 tomorrow
```

tomorrow signifie "demain".



Et si je veux exécuter la commande le 15 novembre à 14h17 ? 🤔

Code : Console

```
$ at 14:17 11/15/08
```



La date est au format américain, les numéros du jour et du mois sont donc inversés : 11/15/08. "11" correspond au mois (novembre) et 15 au numéro du jour !

Exécuter une commande après un certain délai

Il est possible d'exécuter une commande dans 5 minutes, 2 heures ou 3 jours sans avoir à écrire la date. Par exemple, pour exécuter la commande dans 5 minutes :

Code : Console

```
$ at now +5 minutes
```

... ce qui signifie "*Dans maintenant (now) + 5 minutes*". Les mots-clés utilisables sont les suivants :

- minutes
- hours (heures)
- days (jours)
- weeks (semaines)
- months (mois)
- years (années)

Un autre exemple :

Code : Console

```
$ at now +2 weeks
```

... exécutera les commandes dans 2 semaines.

atq et atrm : lister et supprimer les jobs en attente

A chaque fois qu'une commande est "enregistrée", at nous indique un numéro de job ainsi que l'heure où il sera exécuté.

Il est possible d'avoir la liste des jobs en attente avec la commande `atq` :

Code : Console

```
$ atq
13      Mon Nov 10 14:44:00 2008 a mateo21
12      Mon Nov 10 14:42:00 2008 a mateo21
```

Si vous souhaitez supprimer le job n°13 (je sais pas, parce que ça porte malheur par exemple 🙄), utilisez `atrm` :

Code : Console

```
$ atrm 13
```

sleep : faire une pause

Le saviez-vous ? Vous pouvez enchaîner plusieurs commandes à la suite en les séparant par des point-virgules comme ceci :

Code : Console

```
$ touch fichier.txt; rm fichier.txt
```

`touch` est d'abord exécuté, puis une fois qu'il a fini ce sera le tour de `rm` (qui supprimera le fichier que nous venons de créer).

Parfois, enchaîner les commandes comme ceci est bien pratique... mais on a besoin de faire une pause entre les deux. C'est là qu'intervient `sleep` : cette commande permet de faire une pause.

Code : Console

```
$ touch fichier.txt; sleep 10; rm fichier.txt
```

Cette fois, il va se passer les choses suivantes :

- fichier.txt va être créé
- sleep fait une pause de 10 secondes
- rm supprime ensuite le fichier.

Par défaut, la pause est exprimée en secondes. Il est aussi possible d'utiliser d'autres symboles pour changer l'unité :

- m : minutes
- h : heures
- d : jours

Pour faire une pause d'1 minute :

Code : Console

```
$ touch fichier.txt; sleep 1m; rm fichier.txt
```

L'intérêt de sleep ne vous sera peut-être pas évident tout de suite, mais retenez que cette commande existe car il est parfois bien pratique de faire une pause, par exemple pour s'assurer que la première commande a bien eu le temps de se terminer. 😊



Vous pouvez aussi remplacer les points-virgule par des "&&", comme ceci :

```
touch fichier.txt && sleep 10 && rm fichier.txt
```

Dans ce cas, les instructions ne s'enchaîneront que si elles se sont correctement exécutées. Par exemple, si touch renvoie une erreur pour une raison ou une autre, alors les commandes qui suivent (sleep, rm) ne seront pas exécutées.

crontab : exécuter une commande régulièrement

La "crontab" est un incontournable sous Linux : cet outil nous permet de programmer l'exécution régulière d'un programme. Contrairement à "at" qui n'exécutera le programme qu'une seule fois, crontab permet de faire en sorte que l'exécution soit répétée : toutes les heures, toutes les minutes, tous les jours, tous les 3 jours, etc.

Un peu de configuration...

Avant toute chose, nous devons modifier notre configuration (notre fichier .bashrc) pour demander à ce que nano soit l'éditeur par défaut. En général, c'est le programme "vi" qui fait office d'éditeur par défaut. C'est un bon éditeur de texte mais il est bien plus complexe que nano et je ne vous le présenterai que plus tard.

En attendant, rajoutez la ligne suivante à la fin de votre fichier .bashrc :

Code : Console

```
export EDITOR=nano
```

Vous pouvez par exemple écrire la commande suivante :

Code : Console

```
$ echo "export EDITOR=nano" >> ~/.bashrc
```

Cela aura pour effet d'écrire cette ligne à la fin de votre fichier .bashrc situé dans votre répertoire personnel. 😊

Fermez ensuite votre console et réouvrez-la pour que cette nouvelle configuration soit bien prise en compte.

Cette petite configuration étant faite, attaquons les choses sérieuses.

La crontab, qu'est-ce que c'est ?

crontab est en fait une commande qui permet de lire et de modifier un fichier appelé la "crontab".

Ce fichier contient la liste des programmes que vous souhaitez exécuter régulièrement, et à quelle heure vous souhaitez qu'ils soient exécutés.



crontab permet donc de changer la liste des programmes régulièrement exécutés. C'est toutefois le programme cron qui se charge d'exécuter ces programmes aux heures demandées.

Ne confondez donc pas crontab et cron : le premier permet de modifier la liste des programmes à exécuter, et le second les exécute. 😊

Comment utilise-t-on crontab ?

Il y a 3 paramètres différents à connaître, pas plus :

- -e : modifier la crontab.
- -l : afficher la crontab actuelle.
- -r : supprimer votre crontab. Attention, la suppression est immédiate et sans confirmation !

Commençons par afficher la crontab actuelle :

Code : Console

```
$ crontab -l
no crontab for mateo21
```

Normalement, vous n'avez pas encore créé de crontab. Vous noterez qu'il y a une crontab par utilisateur. Là j'édite la crontab de mateo21 car je suis loggé avec l'utilisateur mateo21, mais root a aussi sa propre crontab. La preuve :

Code : Console

```
$ sudo crontab -l
no crontab for root
```

Bien, intéressons-nous à la modification de la crontab (c'est là que ça devient un peu moins drôle 😊). Tapez :

Code : Console

```
$ crontab -e
```

Si vous avez bien configuré votre .bashrc tout à l'heure (et que vous avez relancé votre console), cela devrait ouvrir le programme nano que vous connaissez.

Si par hasard vous n'avez pas fait quelque chose correctement, c'est le programme "vi" qui se lancera. Comme vous ne le connaissez pas encore, tapez ":q" puis Entrée pour sortir. Vérifiez à nouveau votre configuration du .bashrc et n'oubliez pas de fermer puis réouvrir votre console.

Modifier la crontab

Pour le moment, si votre crontab est vide comme la mienne, vous devriez voir uniquement ceci (capture d'écran de nano) :

Code : Console

```
GNU nano 2.0.7      Fichier : /tmp/crontab.4u4jHU/crontab
# m h dom mon dow  command
```

```
[ Lecture de 1 ligne ]
^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper    ^C Pos. cur.
^X Quitter   ^J Justifier ^W Chercher   ^V Page suiv.^U Coller    ^T Orthograp.
```

Les champs

Le fichier ne contient qu'une seule ligne :

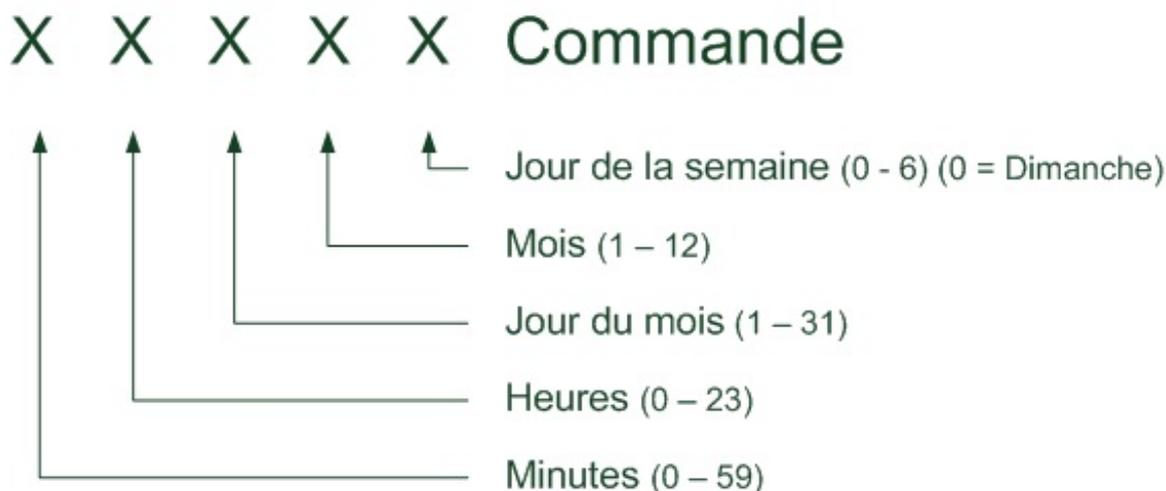
Code : Console

```
# m h dom mon dow command
```

Comme cette ligne est précédée d'un #, il s'agit d'un commentaire (qui sera donc ignoré). Cette ligne vous donne quelques indications sur la syntaxe du fichier :

- m : minutes (0 - 59)
- h : heures (0 - 23)
- dom (day of month) : jour du mois (1 - 31)
- mon (month) : mois (1 - 12)
- dow (day of week) : jour de la semaine (0 - 6, 0 étant le dimanche)
- command : c'est la commande à exécuter

Chaque ligne du fichier correspond à une commande que l'on veut voir exécutée régulièrement. Voici un schéma qui résume la syntaxe d'une ligne :



En clair, vous devez d'abord indiquer à quel moment vous voulez que la commande soit exécutée, puis ensuite vous devez écrire à la fin la commande à exécuter.

C'est un peu comme un tableau. Chaque champ est séparé par un espace.

Chaque "X" sur mon schéma peut être remplacé soit par un nombre, soit par une étoile qui signifie "tous les nombres sont

variables".

Bien comprendre la crontab n'est pas si simple, je vous propose donc de nous baser sur quelques exemples pour voir comment ça marche.

Imaginons que je veuille exécuter une commande tous les jours à 15h47. Je devrai écrire ceci :

Code : Console

```
47 15 * * * touch /home/mateo21/fichier.txt
```

Seules les 2 premières valeurs sont précisées : les minutes et les heures. A chaque fois qu'il est 15h47, la commande indiquée à la fin sera exécutée.



J'ai écrit le chemin du fichier en entier, car vous ne pouvez pas être sûr que le cron s'exécutera dans le répertoire que vous voulez. Il est donc toujours préférable d'écrire le chemin du fichier en absolu comme je l'ai fait ici : /home/mateo21/fichier.txt.



Au fait, pourquoi passer par la commande `crontab -e` pour modifier un fichier ? Il ne serait pas plus simple d'ouvrir le fichier directement avec `nano .crontab` par exemple ?

Oui, mais ce n'est pas comme cela que ça fonctionne. La crontab exige de passer par une commande, c'est comme ça. Il y a quelques avantages à cela, puisque cela permet au programme de vérifier si votre fichier est correctement écrit avant de mettre à jour la crontab. S'il y a une erreur de syntaxe on vous le dira et aucun changement ne sera apporté.

Essayez d'enregistrer et de quitter nano. Vous verrez que la crontab vous dit qu'elle "installe" les changements (elle les prend en compte en quelque sorte) :

Code : Console

```
crontab: installing new crontab
mateo21@mateo21-desktop:~$
```

Désormais, un fichier.txt sera créé dans mon répertoire personnel tous les jours à 15h47 (s'il n'existe pas déjà).

Revenez dans la crontab, nous allons voir d'autres exemples :

Crontab	Signification
<code>47 * * * * commande</code>	Toutes les heures à 47 minutes exactement. Donc à 00h47, 01h47, 02h47, etc.
<code>0 0 * * 1 commande</code>	Tous les lundis soir à minuit.
<code>0 4 1 * * commande</code>	Tous les premiers du mois à 4h du matin.
<code>0 4 * 12 * commande</code>	Tous les jours du mois de décembre à 4h du matin.
<code>0 * 4 12 * commande</code>	Toutes les heures les 4 décembre.
<code>* * * * * commande</code>	Toutes les minutes !



Est-il possible d'exécuter une commande plus fréquemment que toutes les minutes ?

Non, c'est impossible avec cron. La fréquence minimale c'est toutes les minutes.

Les différentes notations possibles

Pour chaque champ, on a le droit à différentes notations :

- 5 (un nombre) : exécuté lorsque le champ prend la valeur 5
- * : exécuté tout le temps (toutes les valeurs sont bonnes).
- 3,5,10 : exécuté lorsque le champ prend la valeur 3, 5 ou 10. Ne pas mettre d'espace après la virgule.
- 3-7 : exécuté pour les valeurs 3 à 7.
- */3 : exécuté tous les multiples de 3 (par exemple à 0h, 3h, 6h, 9h...).

Vous connaissiez déjà les 2 premières notations. Celles que nous venons de découvrir nous permettent de démultiplier les possibilités offertes par la crontab.

Voici un tableau avec quelques exemples d'utilisation :

Crontab	Signification
30 5 1-15 * * commande	A 5h30 du matin du 1er au 15 de chaque mois.
0 0 * * 1,3,4 commande	A minuit le lundi, le mercredi et le jeudi.
0 */2 * * * commande	Toutes les 2 heures (00h00, 02h00, 04h00...)
*/10 * * * 1-5 commande	Toutes les 10 minutes du lundi au vendredi.

Comme vous le voyez, la crontab offre de très larges possibilités (pour peu que l'on ait compris comment elle fonctionne 😊).

Rediriger la sortie

Pour le moment, nous avons exécuté notre commande très simplement dans la crontab :

Code : Console

```
47 15 * * * touch /home/mateo21/fichier.txt
```

Toutefois, il faut savoir que si la commande renvoie une information ou une erreur, vous ne la verrez pas apparaître dans la console. Normal : ce n'est pas vous qui exécutez la commande mais le programme cron.

Que se passe-t-il alors si la commande renvoie un message ? En fait, le résultat de la commande vous est envoyé par e-mail. Chaque utilisateur possède sa propre boîte e-mail sur les machines de type Unix, mais je ne vais pas m'attarder là-dessus. Nous allons plutôt voir comment rediriger le résultat.

Tiens, rediriger une sortie, vous savez faire ça non ? 😊

Code : Console

```
47 15 * * * touch /home/mateo21/fichier.txt >> /home/mateo21/cron.log
```

Tous les messages seront désormais ajoutés à la fin de cron.log. Tous ? Non, on oublie d'y rediriger aussi les erreurs !

Code : Console

```
47 15 * * * touch /home/mateo21/fichier.txt >> /home/mateo21/cron.log 2>&1
```

Voilà, c'est mieux. 😊

Cette fois, tout sera envoyé dans cron.log : les messages et les erreurs.



Et si je ne veux pas du tout récupérer ce qui est affiché ?

Nous avons déjà appris à le faire ! Il suffit de rediriger dans /dev/null (le fameux "trou noir" du système). Tout ce qui est envoyé là-dedans est immédiatement supprimé, hop, plus de trace, le crime parfait. 😊

Code : Console

```
47 15 * * * touch /home/mateo21/fichier.txt > /dev/null 2>&1
```

Vous voilà passés maîtres dans l'art de programmer l'exécution de tâches sur votre machine !

Le programme-clé de ce chapitre, vous l'aurez compris, c'est crontab. Vous y aurez certainement souvent recours pour automatiser des tâches, aussi vaut-il mieux s'habituer à l'utiliser. Sa syntaxe est un peu surprenante au premier abord, mais une fois qu'on sait l'utiliser on peut faire (presque) tout ce qu'on veut. 😊

Vous pouvez par exemple l'utiliser pour nettoyer des dossiers temporaires tous les soirs. Bien souvent, vous l'utiliserez pour faire appel à des scripts personnalisés que vous aurez écrits qui exécutent toute une série de tests et de tâches. Nous verrons comment faire cela dans la partie V (pas la partie suivante mais celle d'après).

Personnellement, j'ai été amené à écrire par exemple des scripts qui vérifient toutes les semaines le taux d'utilisation du disque dur et qui m'avertissent si jamais on vient à manquer de place, ou encore des scripts qui sauvegardent les données de l'ordinateur tous les soirs (*scripts de backup*).

Partie 4 : Transférer des données à travers le réseau

Depuis longtemps, les systèmes d'exploitation de type Unix (comme Linux) proposent une panoplie d'outils pour travailler en réseau. Ces outils sont aujourd'hui très perfectionnés et vous permettent d'accéder à une machine à distance, comme si vous étiez devant. Le tout de manière sécurisée et cryptée.

Comment compresser ses fichiers pour optimiser l'envoi ? Comment se connecter à une machine Linux à distance ? Qu'est-ce que SSH et le FTP ? Comment analyser le trafic réseau et se protéger à l'aide d'un firewall ?

Autant de questions auxquelles nous allons répondre dans cette partie. 😊

Archiver et compresser

Pour bien commencer cette partie sur le réseau, il me semble logique de vous présenter d'abord le fonctionnement de la compression sous Linux. En effet, si vous vous apprêtez à envoyer un ou plusieurs fichiers par le réseau (que ce soit par mail, FTP ou autre), il est toujours préférable de commencer par les compresser afin de réduire leur taille.

Vous avez sûrement déjà entendu parler du format Zip. C'est le plus connu et le plus répandu... du moins sous Windows. On peut l'utiliser aussi sous Linux, de même que le format Rar.

Cependant, on préférera utiliser des alternatives libres (et souvent plus puissantes) telles que le Gzip et le Bzip2. Toutefois, contrairement à Zip et Rar, le Gzip et le Bzip2 ne sont capables de compresser qu'un seul fichier à la fois et ne peuvent donc pas créer un "paquetage" de plusieurs fichiers.

Mais rassurez-vous, tout est prévu : on utilise pour cela un outil à part, appelé Tar, qui permet d'assembler des fichiers avant de les compresser.

Nous allons découvrir le fonctionnement de tout cela dans ce chapitre. 😊

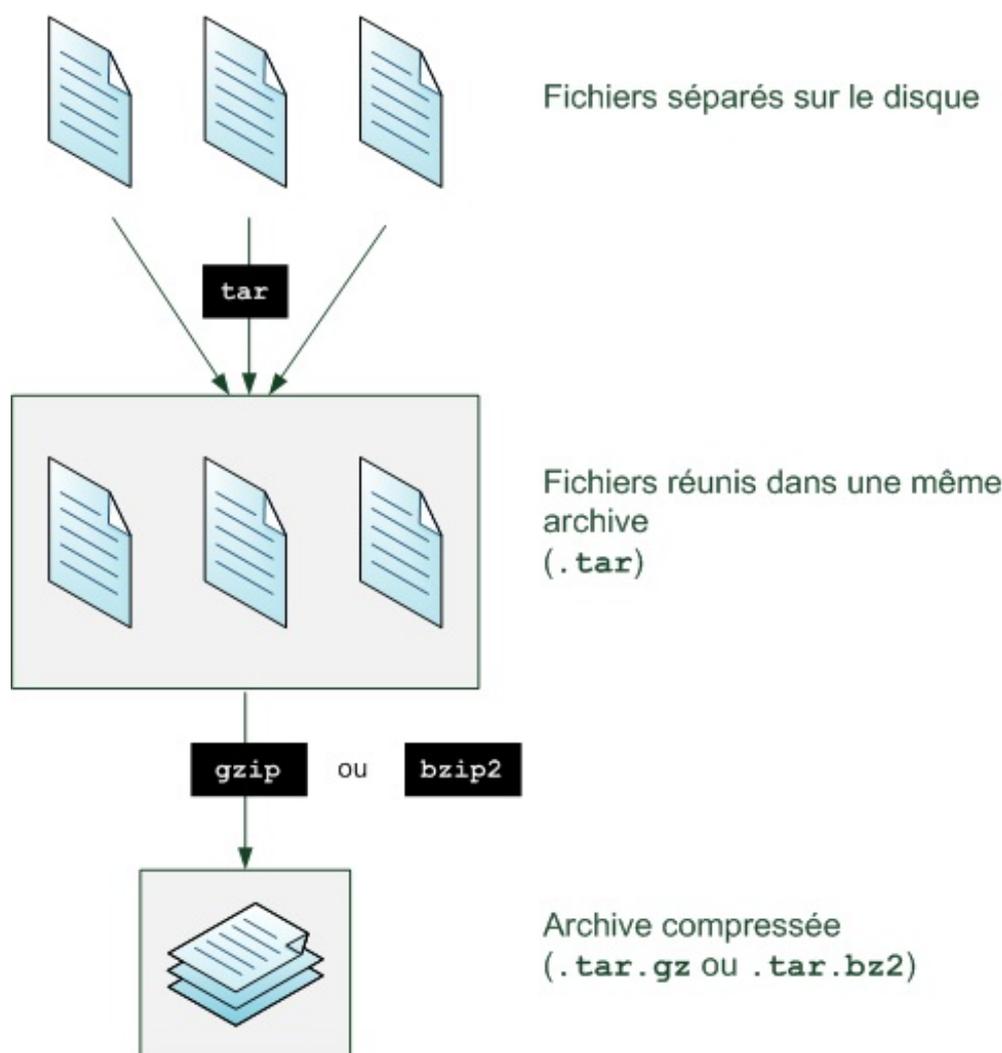
tar : assembler des fichiers dans une archive

Comme je vous le disais en introduction, aussi étonnant que cela puisse paraître les gzip et bzip2 ne permettent de compresser qu'un seul fichier à la fois. Comment faire alors si vous voulez compresser une dizaine de fichiers ?

Sous Linux, on a pris l'habitude depuis longtemps de séparer les deux choses en deux temps :

1. Réunir les fichiers dans un seul gros fichier appelé **archive**. On utilise pour cela le programme **tar**.
2. Compresser le gros fichier ainsi obtenu à l'aide de **gzip** ou de **bzip2**.

Ces 2 étapes sont résumées dans ce schéma :



Nous allons dans un premier temps apprendre à manipuler tar, puis nous verrons la compression avec gzip et bzip2, sans oublier les formats zip et rar que vos amis utilisant Windows risquent de vous envoyer un jour ou l'autre.



Les formats zip et rar ne séparent pas les étapes comme sur le schéma ci-dessus. Ils sont capables d'assembler plusieurs fichiers en une archive et de la compresser en même temps.

Comme vous le voyez, sous Linux il y a donc une méthode à suivre dans un ordre précis. Voyons voir ensemble comment faire !

Regrouper d'abord les fichiers dans un même dossier

Vous avez plusieurs fichiers que vous voulez compresser. Dans mon cas, ce sont des fichiers .tuto (qui contiennent des chapitres de tutoriels du Site du Zéro 🤖), mais vous pouvez bien entendu compresser ce que vous voulez : des textes, présentations, tableurs, logs, etc.



Certains formats de fichier sont déjà compressés. C'est le cas des images jpeg, png et gif, mais aussi de la plupart des vidéos. Vous pouvez bien entendu les assembler dans une archive tar et même les compresser, mais vous ne les rendrez en général pas plus petits car ils ont déjà été compressés.

Mes fichiers .tuto que je souhaite archiver sont pour le moment placés en vrac dans mon home :

Code : Console

```
$ ls
```

```
Bureau      Images      l-heritage.tuto  Public
Documents   la-surcharge-d-operateurs.tuto  Modèles        Vidéos
Exemples    les-principaux-widgets.tuto     Musique
```

Il est recommandé de placer d'abord les fichiers à archiver dans un seul et même dossier. Créons-le et déplaçons-y tous nos .tuto :

Code : Console

```
$ mkdir tutoriels
$ mv *.tuto tutoriels/
$ ls
Bureau      Exemples  Modèles  Public  Vidéos
Documents   Images    Musique  tutoriels
```

Voilà, nos fichiers sont réunis dans le dossier tutoriels.

-cvf : créer une archive tar

Nous allons maintenant créer une archive tar de ce dossier et de ses fichiers. Le schéma à suivre pour créer une archive est :

Code : Console

```
tar -cvf nom_archive.tar nom_dossier/
```

J'utilise 3 options :

- **-c** : signifie "créer" une archive tar.
- **-v** : signifie afficher le détail des opérations.
- **-f** : signifie assembler l'archive dans un fichier.

Essayons de faire cela sur notre dossier tutoriels :

Code : Console

```
$ tar -cvf tutoriels.tar tutoriels/
tutoriels/
tutoriels/les-principaux-widgets.tuto
tutoriels/la-surcharge-d-operateurs.tuto
tutoriels/l-heritage.tuto
```

Ici on archive le dossier "tutoriels" et donc son contenu. Grâce à -v, on voit bien la liste des fichiers qui ont été archivés.



Est-on obligé de mettre systématiquement nos fichiers dans un même dossier pour archiver ensuite ce dossier ? On ne pourrait pas archiver directement les fichiers ?

Si, c'est possible. Imaginons que nous soyons toujours dans le home avec nos fichiers .tuto. On pourrait très bien faire :

Code : Console

```
tar -cvf archive.tar fichier1 fichier2 fichier3
```

C'est possible et ça fonctionne. Toutefois, il est de coutume sous Linux de placer *d'abord* les fichiers dans un dossier avant de les mettre dans le tar. Cela permet d'éviter, lorsqu'on extrait les fichiers de l'archive, que ceux-ci aillent se mêler à d'autres fichiers. Nous allons voir ce problème maintenant.

-tf : afficher le contenu de l'archive sans l'extraire

Vous venez de recevoir une archive tar qu'on vous a envoyée. Bien. Mais que contient-elle ? Avant d'extraire quoi que ce soit, vous aimeriez peut-être voir son contenu. C'est possible avec -tf :

Code : Console

```
$ tar -tf tutoriels.tar
tutoriels/
tutoriels/les-principaux-widgets.tuto
tutoriels/la-surcharge-d-operateurs.tuto
tutoriels/l-heritage.tuto
```

Quand on fait cela, on voit que tous les fichiers sont réunis dans un même dossier tutoriels, et ça c'est très pratique. J'en reviens justement au problème dont je parlais un peu plus haut : imaginez que vous "détariez" une archive contenant plus de 400 fichiers dans votre home. Si ces fichiers n'étaient pas réunis dans un dossier, ils iraient tous se mêler aux autres fichiers du répertoire home, et alors là je vous dis pas la pagaille. 🤔

Voilà donc pourquoi je vous ai invité dès le début à réunir vos fichiers à archiver dans un même dossier. Cela permet d'éviter des mauvaises surprises pour celui qui extrait les fichiers de l'archive. Quasiment toutes les archives qu'on vous proposera de télécharger suivent ce même schéma et font attention à tout réunir dans un même dossier, mais vérifiez le contenu avant de l'extraire, on ne sait jamais !

-rvf : ajouter un fichier

Si vous avez oublié un fichier, vous pouvez toujours l'ajouter par la suite avec -rvf :

Code : Console

```
$ tar -rvf tutoriels.tar fichier_supplementaire.tuto
tutoriels/fichier_supplementaire.tuto
```

-xvf : extraire les fichiers de l'archive

Pour extraire les fichiers, on va utiliser les options -xvf (-x pour eXtract) :

Code : Console

```
$ tar -xvf tutoriels.tar
tutoriels/
tutoriels/les-principaux-widgets.tuto
tutoriels/la-surcharge-d-operateurs.tuto
```

```
tutoriels/l-heritage.tuto
```

Les fichiers s'extraient dans le répertoire dans lequel vous vous trouvez. Vérifiez donc avant de les extraire que ceux-ci sont réunis dans un même dossier (avec -tf) si vous ne voulez pas que ces fichiers aillent se mélanger à d'autres !

gzip & bzip2 : compresser une archive

Vous avez maintenant créé une belle archive .tar. Tous vos fichiers sont réunis là-dedans. Il nous faut maintenant compresser cela.

Nous disposons de 2 programmes de compression bien répandus dans le monde Linux :

- **gzip** : c'est le plus connu et le plus utilisé.
- **bzip2** : il est un peu moins fréquemment utilisé. Il compresse mieux mais plus lentement que gzip.



A noter qu'il existe aussi le vieux programme **compress**. Il n'est cependant plus vraiment utilisé car on dispose aujourd'hui de meilleurs algorithmes de compression : gzip et bzip2.

Ces programmes sont simples à utiliser. Ils prennent comme paramètre le nom du fichier à compresser. Ils le compressent et modifient ensuite son nom.

Concrètement, ils ajoutent un suffixe pour indiquer que l'archive a été compressée :

- .tar.gz : si l'archive a été compressée avec gzip.
- .tar.bz2 : si l'archive a été compressée avec bzip2.

A titre indicatif, voici les différentes tailles de l'archive, avant et après compression :

Fichier	Taille
tutoriels.tar	130 Ko
tutoriels.tar.gz	35 Ko
tutoriels.tar.bz2	29 Ko

Cela confirme ce que je vous disais : bzip2 est plus efficace... mais il est aussi plus lent à compresser et moins fréquemment utilisé .

gzip : la compression la plus courante

Concrètement, le programme gzip s'utilise de la manière la plus simple qui soit :

Code : Console

```
gzip tutoriels.tar
```

L'archive est compressée et gagne ensuite le suffixe .gz. Elle s'appelle donc désormais `tutoriels.tar.gz`. Voilà pourquoi vous voyez circuler sur internet des fichiers .tar.gz : cela signifie que ce sont des archives compressées !

Pour décompresser l'archive ensuite, il suffit d'utiliser `gunzip` :

Code : Console

```
gunzip tutoriels.tar.gz
```

L'archive retrouve son état non compressé en .tar. Vous pouvez maintenant extraire les fichiers de l'archive comme vous avez appris à le faire un peu plus tôt avec tar -xvf. 😊

bzip2 : la compression la plus puissante

Le fonctionnement de bzip2 est le même que gzip :

Code : Console

```
bzip2 tutoriels.tar
```

Une archive compressée `tutoriels.tar.bz2` sera alors créée. Pour la décompresser, utiliser `bunzip2` :

Code : Console

```
bunzip2 tutoriels.tar.bz2
```

Vous retrouvez un .tar que vous pouvez extraire avec tar -xvf.

Archiver et compresser en même temps avec tar



C'est bien beau de séparer les étapes, mais cela nous demande de taper 2 fois plus de commandes pour compresser et décompresser des fichiers ! Il n'y a pas plus rapide ?

Si on fait comme cela, c'est essentiellement pour des raisons historiques. Souvenez-vous que Linux ne fait que recopier le fonctionnement d'Unix dont l'origine remonte aux années 1960 !

Heureusement, les choses ont un peu évolué. Il faut toujours archiver puis compresser, mais le programme tar est capable d'appeler lui-même gzip ou bzip2 si vous lui donnez les bons paramètres.

-zcvf : archiver et compresser en gzip

Vous connaissez tar -cvf pour créer une archive tar. Si vous rajoutez l'option -z, l'archive sera automatiquement compressée avec gzip.

Code : Console

```
tar -zcvf tutoriels.tar.gz tutoriels/
```

Voilà comment on obtient une archive compressée en une seule commande. 😊

Pour décompresser, c'est pareil, sauf que le -c est remplacé par un -x comme tout à l'heure :

Code : Console

```
tar -zxvf tutoriels.tar.gz
```

-jcvf : archiver et compresser en bzip2

Le principe est le même avec -j à la place de -z :

Code : Console

```
tar -jcvf tutoriels.tar.bz2 tutoriels/
```

Et pour extraire :

Code : Console

```
tar -jxvf tutoriels.tar.bz2 tutoriels/
```



Vous pouvez toujours analyser le contenu de l'archive avant de la décompresser. Avec -ztf vous regarderez à l'intérieur d'une archive gzipée, et avec -jtf vous regarderez à l'intérieur d'une archive "bzipée-deux". 🤖

zcat, zmore & zless : afficher directement un fichier compressé

Parfois, on compresses non pas une archive tar mais directement un fichier. Par exemple, je peux compresser un fichier .tuto directement :

Code : Console

```
gzip l-heritage.tuto
```

Le fichier est alors compressé et renommé en l-heritage.tuto.gz.

Maintenant, supposons que nous voulions afficher le contenu de ce fichier sans le décompresser auparavant. Eh bien il existe des outils qui permettent de faire cela !

- **zcat** : équivalent de cat, capable de lire un fichier compressé (grippé).
- **zmore** : équivalent de more, capable de lire un fichier compressé (grippé).
- **zless** : équivalent de less, capable de lire un fichier compressé (grippé).

Si vous essayez de faire un `cat l-heritage.tuto.gz`, vous allez voir des caractères bizarres s'afficher à l'écran :

Ces caractères bizarres sont une représentation de votre fichier compressé. Comme vous pouvez le voir, ce n'est pas très lisible.



A ce stade, votre console est d'ailleurs buggée. Si vous tapez des caractères vous allez voir que vous allez taper n'importe quoi. Pour réinitialiser la console, tapez la commande `reset` puis appuyez sur Entrée.

Maintenant, essayez plutôt d'utiliser `zcat`. Ce programme va décompresser le fichier à la volée et l'afficher dans la console :

Code : Console

```
$ zcat l-heritage.tuto.gz
<conclusion>
<![CDATA[Ce chapitre en impose peut-être un peu par sa taille, mais ne vous y f
D'ailleurs, j'ai volontairement évité de trop montrer de codes sources complets dif
```

Les commandes `zmore` et `zless`, équivalents de `more` et `less` qui permettent d'afficher page par page, fonctionnent aussi !

unzip & unrar : décompresser les .zip et .rar

Les `.tar.gz` et `.tar.bz2` ont beau être courants dans le monde Linux, vos amis utilisant Windows ne les connaissent pas et risquent tôt ou tard de vous envoyer un superbe `.zip` ou `.rar`... que vous ne pouvez pas décompresser avec `gunzip`.

Heureusement, il existe des utilitaires de décompression pour ces formats. Ils ne sont pas toujours installés par défaut, il faudra donc les installer si vous ne les avez pas.

unzip : décompresser un .zip

Vous venez de recevoir un `.zip` ?

Pas de panique ! Le programme `unzip` est capable de l'extraire. Il est peut-être installé par défaut, mais si vous ne l'avez pas, vous savez ce qu'il vous reste à faire :

Code : Console

```
sudo apt-get install unzip
```

Ceci étant fait, l'utilisation d'unzip est très simple :

Code : Console

```
unzip archive.zip
```



Les fichiers vont s'extraire dans le dossier dans lequel vous vous trouvez ! Le problème est le même qu'avec les .tar.gz et .tar.bz2. Avant de décompresser, vérifiez si les fichiers sont réunis dans un même dossier.

Pour voir le contenu d'une archive zip sans l'extraire, utilisez -l :

Code : Console

```
$ unzip -l tutoriels.zip
Archive:  tutoriels.zip
 Length   Date      Time    Name
-----
      0   11-12-08  15:04   tutoriels/
  59515   11-12-08  14:44   tutoriels/les-principaux-widgets.tuto
  36757   11-12-08  14:43   tutoriels/la-surcharge-d-operateurs.tuto
  27685   11-12-08  14:44   tutoriels/l-heritage.tuto
-----
 123957                   4 files
```

On peut voir que les fichiers sont réunis dans un même dossier dans l'archive. C'est plutôt rare avec les .zip en général, faites donc attention avant de décompresser les fichiers pour qu'ils n'atterrissent pas n'importe où.

En général on a surtout besoin d'unzip pour décompresser un zip, mais il est plus rare que l'on soit amené à créer un fichier zip (on préférera toujours le gzip ou le bzip2). Si toutefois vous voulez vraiment créer un zip, installez le programme "zip" puis basez-vous sur la commande suivante :

Code : Console

```
zip -r tutoriels.zip tutoriels/
```

Le -r demande à compresser tous les fichiers contenus dans le dossier tutoriels (sinon seul le dossier, vide, sera compressé !).

unrar : décompresser un .rar

Il vous faut installer le paquet unrar pour pouvoir utiliser cette commande :

Code : Console

```
sudo apt-get install unrar
```

Ensuite, pour extraire :

Code : Console

```
unrar e tutoriels.rar
```

Non, vous ne rêvez pas, l'auteur du programme ne veut pas que l'on mette un tiret devant l'option "e" ! Il faut bien qu'il y ait des exceptions dans la vie. 😞

Pour lister le contenu avant décompression, utilisez l'option "l" :

Code : Console

```
$ unrar l tutoriels.rar
UNRAR 3.80 beta 2 freeware      Copyright (c) 1993-2008 Alexander Roshal
Archive tutoriels.rar

Name                Size   Packed Ratio  Date   Time   Attr      CRC   Meth Ver
-----
les-principaux-widg  59515   16191  27% 12-11-08 14:44 -rw-r--r--
6E266812 m3b 2.9
la-surcharge-d-oper  36757   11215  30% 12-11-08 14:43 -rw-r--r-
-E8474528 m3b 2.9
l-heritage.tuto     27685   8720  31% 12-11-08 14:44 -rw-r--r--
738EF121 m3b 2.9
-----
      3             123957   36126  29%
```



Et si je veux créer des .rar ?

Ce n'est pas possible. En fait, le format rar est propriétaire. La méthode de décompression a été donnée et vous pouvez donc décompresser des .rar, mais pour créer des .rar il faut... acheter le logiciel.

Vous pouvez toujours installer le paquet "rar" mais vous verrez que c'est une shareware, qu'il n'est pas libre et qu'il faudra l'acheter sous 40 jours... bref, ce n'est pas le meilleur plan. Si vraiment vous voulez rester compatible, créez plutôt des .zip. Vous voilà capables de générer des archives, de choisir la compression, de lister leur contenu et de décompresser le tout ! Maintenant que vous savez manipuler les archives, la suite logique sera de découvrir comment les envoyer par internet, en les copiant sur un serveur grâce au FTP par exemple. Nous verrons cela dans les chapitres suivants. 😊

Une petite précision : si vous êtes amenés à communiquer avec des personnes utilisant Windows, ce qui arrivera tôt ou tard, vous n'êtes pas obligés d'utiliser le format Zip. Vous pouvez très bien utiliser le Gzip et le Bzip2, mais dans ce cas il faudra que les utilisateurs de Windows installent un programme capable de décompresser ce format de fichier.

Je vous recommande de leur faire installer 7zip qui gère un très grand nombre de formats compressés.

📄 La connexion sécurisée à distance avec SSH

Nous entrons probablement maintenant dans l'un des chapitres les plus intéressants de ce cours. Nous allons découvrir comment se connecter à distance à une machine équipée de Linux.

Je vous en ai déjà un peu parlé au début du cours : toutes les machines sous Linux peuvent être configurées pour qu'on s'y connecte à distance, pour peu qu'elles restent allumées.

Prenons un cas concret : votre ordinateur chez vous est sous Linux, vous le laissez allumé. Pendant la journée au boulot, vous avez besoin de lancer un téléchargement ou de récupérer un document. Vous vous connectez à distance sur votre machine et vous ouvrez une console comme si vous étiez en face de votre PC ! Tout ce que vous avez appris à faire dans une console, vous pouvez le faire à distance depuis n'importe quelle machine dans le monde. 😊

Nous n'allons pas seulement découvrir comment se connecter à distance. Nous allons aussi essayer de comprendre comment ça fonctionne et comment les données sont sécurisées grâce au protocole SSH. Ce chapitre sera donc mi-théorique, mi-pratique. Vous apprendrez très certainement de nouvelles choses sur le monde passionnant des réseaux et de la sécurité (cryptographie).



Ce chapitre intéressera en particulier ceux qui ont besoin d'apprendre à gérer un serveur dédié (et ils sont de plus en plus nombreux). Sur le Site du Zéro par exemple, j'ai été obligé d'apprendre tout cela lorsque le site a commencé à grossir et qu'il a fallu louer un premier serveur dédié et l'administrer à distance. On peut donc être amené à gérer un serveur si on a un gros site ou si on souhaite y faire tourner toutes sortes de services, comme un serveur de jeux.

Se connecter à une console à distance

Jusqu'ici, vous avez utilisé Linux de la même façon que Windows : vous étiez *en face* de votre ordinateur. Vous étiez physiquement à côté de votre machine, vous avez par exemple appuyé sur le bouton "Power" pour l'allumer. Jusque-là rien de nouveau.

Pourtant, une des grandes forces de Linux est qu'on peut s'en servir même si on est à des centaines de kilomètres de la machine. Ce fonctionnement date de l'époque d'Unix où il était nécessaire d'administrer des machines à distance.

Aujourd'hui, si j'habite à Paris je peux très bien contrôler un ordinateur sous Linux situé à Tokyo au Japon en même temps qu'un autre ordinateur situé au fin fond du Nevada aux Etats-Unis. Je peux même ordonner à l'ordinateur de Tokyo d'envoyer un fichier à celui du Nevada.

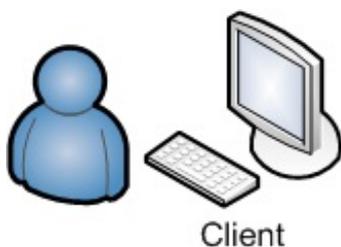
Ce genre de manipulation est désormais possible grâce à internet, et cela se fait tous les jours. Les personnes qui travaillent à gérer des machines Linux, souvent à distance, sont appelés *administrateurs système* (c'est un métier recherché !).

Heureusement qu'il n'est pas nécessaire d'être présent physiquement à côté de la machine pour travailler dessus ! Vous imaginez, devoir se payer un billet aller-retour pour Tokyo juste parce qu'on a besoin d'installer un programme sur un serveur...



Un **serveur** est un ordinateur qui reste allumé 24h/24, 7j/7. Cet ordinateur est semblable au vôtre (quoique souvent plus puissant et plus bruyant) : il possède un processeur, un ou des disques durs, etc. Le principe d'un serveur est de rester allumé et connecté à internet tout le temps. Il offre des services. Par exemple, le Site du Zéro possède plusieurs serveurs qui sont chargés de vous envoyer les pages web du site à toute heure du jour et de la nuit. 😊

Le PC qui se connecte au serveur est appelé le **client**. Nous allons les représenter comme ceci dans les prochains schémas :



Actuellement, votre petit PC chez vous n'est pas considéré comme un serveur... mais vous pouvez très facilement le transformer en serveur si vous le désirez à condition d'installer les bons programmes qui en font un serveur et de les configurer correctement. Et de le laisser allumé aussi, parce qu'un serveur éteint c'est un serveur qui ne sert à rien. 🤖

Nous allons suivre le plan suivant :

1. Pourquoi faut-il sécuriser les échanges ?
2. Comment fait SSH pour sécuriser les échanges ?
3. Comment utiliser SSH concrètement ?

De Telnet à SSH, pourquoi faut-il sécuriser les échanges ? Les protocoles

Pour communiquer entre eux en réseau, 2 ordinateurs doivent utiliser le même *protocole*. C'est un peu comme une langue : pour parler à quelqu'un vous devez parler la même langue que lui, sinon vous ne vous comprendrez pas.

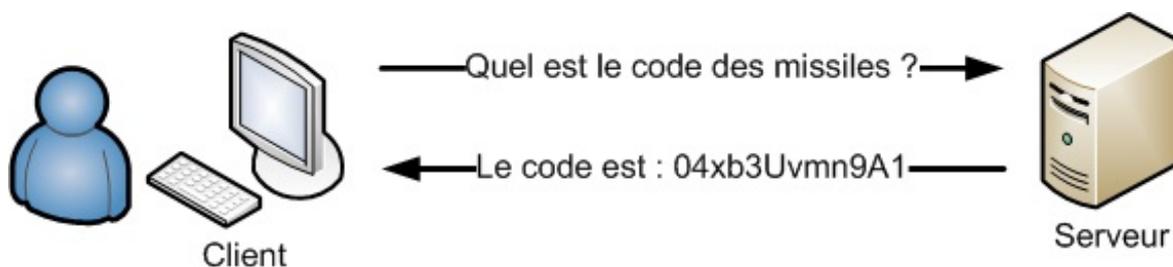
Il existe de très nombreux protocoles pour que les ordinateurs puissent communiquer entre eux. Il y en a un que vous avez forcément vu, c'est le HTTP (*HyperText Transfer Protocol*). Si si, regardez l'adresse par exemple du Site du Zéro : <http://www.siteduzero.com>. Le préfixe "http" signifie que vous communiquez avec les serveurs du Site du Zéro à l'aide du protocole HTTP. C'est le protocole utilisé sur le web pour s'échanger des pages web. Mais il existe bien d'autres protocoles ! Par exemple le FTP (*File Transfer Protocol*, protocole de transfert de fichiers), l'IMAP (*Internet Message Access Protocol*, utilisé pour s'échanger des e-mails), etc.

Le protocole Telnet : simple mais dangereux

Il y a un protocole très simple, très basique, qui a été créé dans les années 80 : c'est **Telnet**. Il sert juste à échanger des messages simples d'une machine à une autre.

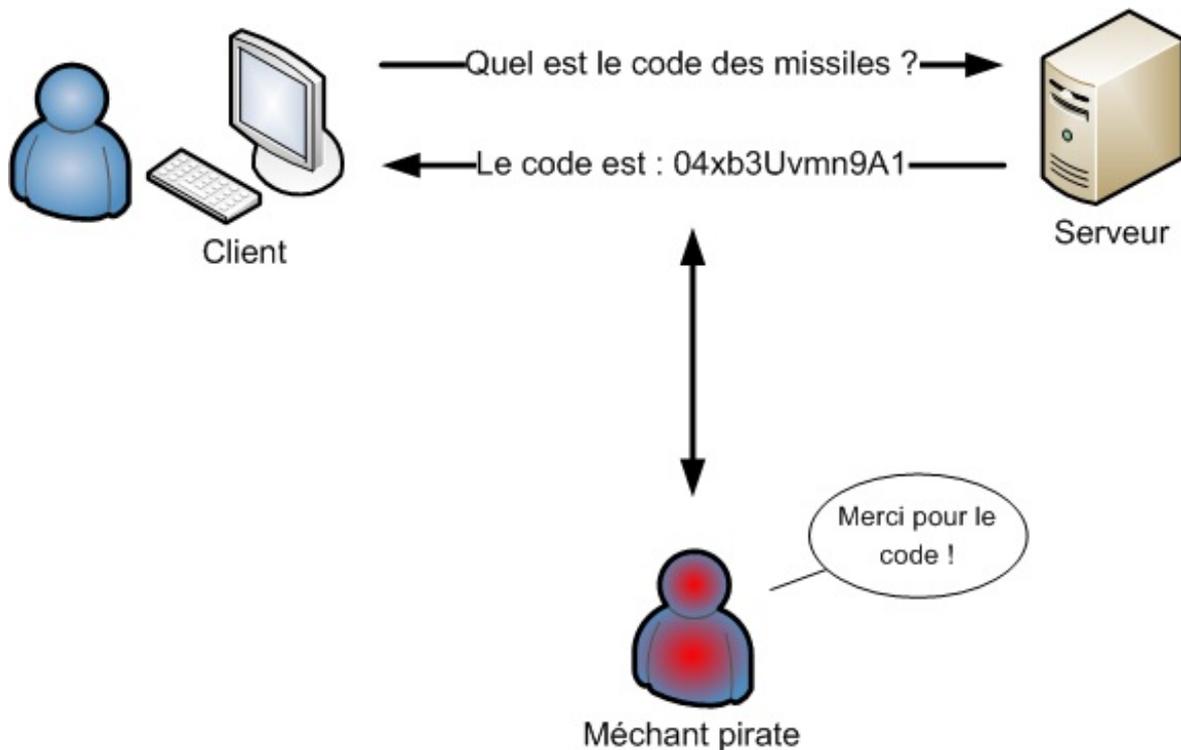
En théorie donc, on peut communiquer avec un serveur à l'aide du protocole Telnet. Le problème de ce protocole... c'est justement qu'il est trop simple : les données sont transférées en clair sur le réseau. Il n'y a aucun cryptage.

Voici ce qui pourrait se passer. J'exagère les traits mais c'est pour vous donner une idée. Imaginez qu'un PC de l'armée demande à un serveur de l'armée le code de lancement de missiles (nucléaires, soyons fous 🤖) :

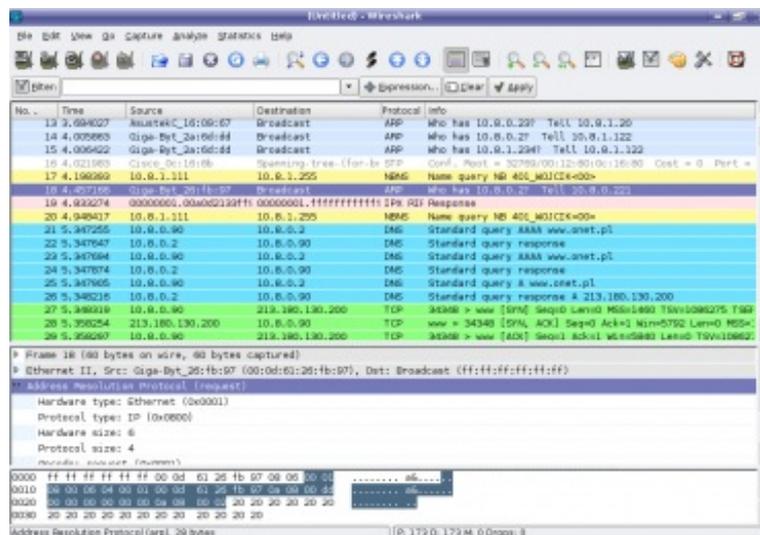


Après tout, il n'y a rien de choquant. Le message n'est envoyé qu'au client qui l'a demandé.

Mais en fait, il est possible pour un pirate d'"écouter" ce qui se passe sur le réseau, et donc d'intercepter les données en chemin :



Vous pouvez difficilement empêcher que quelqu'un intercepte les données. Intercepter les données peut être compliqué à faire, mais c'est possible (non, ce n'est pas le sujet de ce chapitre de vous expliquer comment faire 🤖). Sachez qu'il existe par exemple des programmes comme [Wireshark](#) (anciennement nommé *Ethereal*) qui sont capables d'écouter ce qui se passe sur un réseau local notamment, et donc d'intercepter les données.



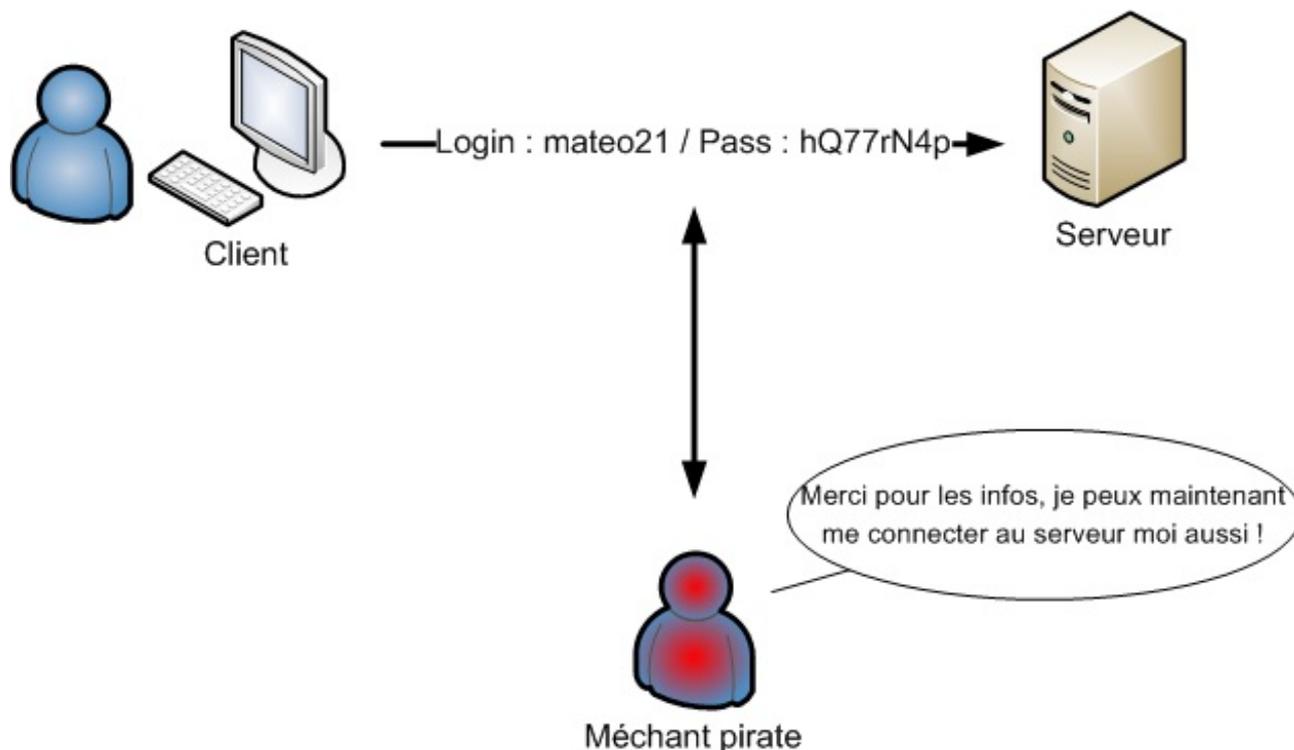
Wireshark en action



Euh attends là, moi je veux juste me connecter à distance à ma machine ou à un serveur pour avoir l'accès à la console. Je vais pas échanger le code de lancement de missiles nucléaires ! Je vois pas en quoi c'est un problème que quelqu'un sache que je suis en train de faire un "grep" sur ma machine par exemple...

Ca ne vous dérange pas qu'on vous espionne ? Soit.

Mais quand vous allez vous connecter au serveur, vous allez donner votre login et votre mot de passe. Rien que ça, c'est dangereux. Il ne faut pas que le login et le pass apparaissent en clair sur le réseau !



Rien que pour ça, il faut que les données soient cryptées. Vous ne voulez pas que quelqu'un récupère votre mot de passe !

Le protocole SSH : la solution pour sécuriser les données

Comme on ne peut pas complètement empêcher quelqu'un d'intercepter les données qui transitent sur internet, il faut trouver un moyen pour que le client et le serveur communiquent de manière sécurisée. Le cryptage sert précisément à ça : si le pirate récupère le mot de passe crypté, il ne peut rien en faire.

Mais tout cela est plus compliqué que ça en a l'air. Comment crypter les données ?

Comment sont cryptés les échanges avec SSH ?

SSH est un protocole assez complexe, mais il est vraiment intéressant de savoir comment il fonctionne. Plutôt que de l'utiliser bêtement, je vous propose de vous expliquer dans les grandes lignes son mode de fonctionnement.

Nous allons voir 2 choses ici :

1. Quelles sont les différentes méthodes de cryptage qui existent ?
2. Comment SSH utilise ces méthodes de cryptage pour garantir la sécurité ?

Quelles sont les différentes méthodes de cryptage qui existent ?

Il existe des tonnes d'algorithmes de cryptage. Je ne vais pas tous vous les présenter. Cela demanderait trop de notions mathématiques et on pourrait y passer 30 chapitres qu'on n'aurait pas tout vu.

Si on ne peut pas connaître tous les algorithmes de cryptage, il faut par contre savoir qu'on peut les classer dans 2 catégories : les cryptages symétriques et les cryptages asymétriques.

Le cryptage symétrique

C'est la méthode de cryptage la plus simple. Cela ne veut pas dire qu'elle n'est pas robuste (il existe des cryptages symétriques

très sûrs). Cela veut plutôt dire que le fonctionnement est simple à comprendre. 😊

Avec cette méthode, pour crypter un message on utilise une clé (un mot de passe secret). Par exemple, imaginons que cette clé soit "topsecret" :

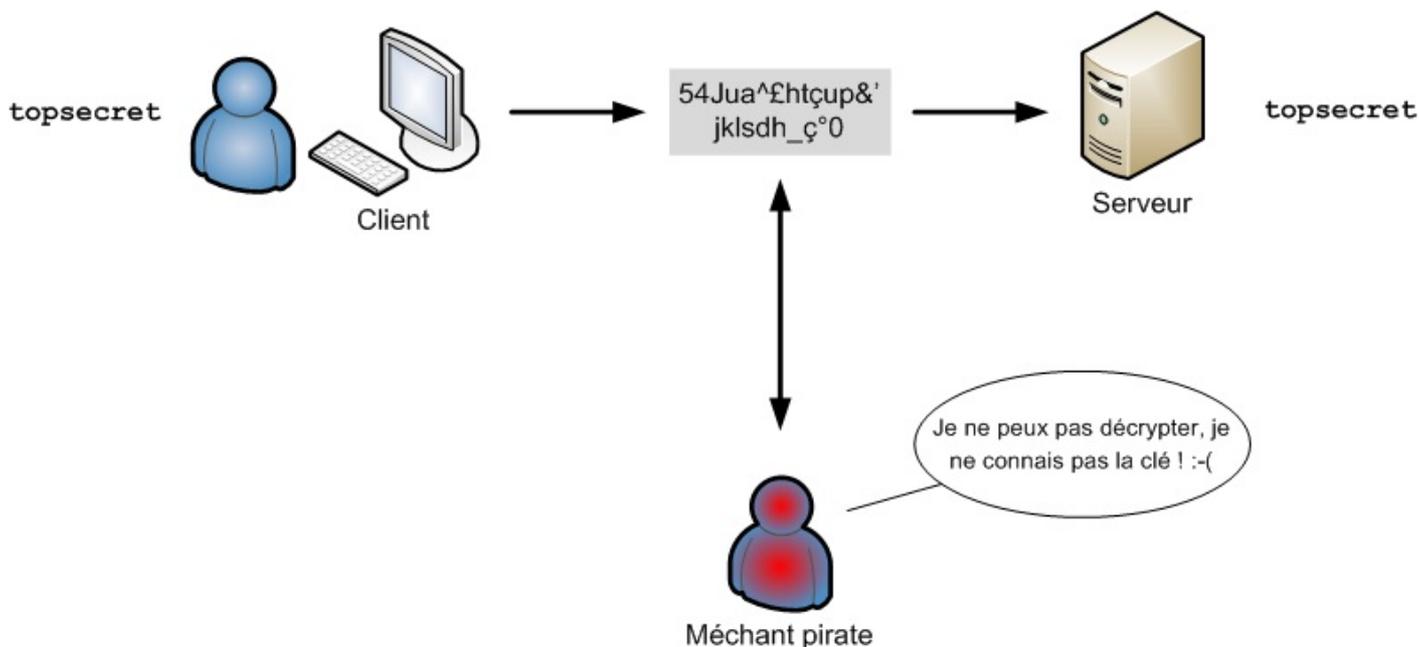


Pour décrypter ensuite le message, on utilise cette même clé :



Il faut donc que la personne qui crypte et la personne qui décrypte connaissent toutes les deux cette clé qui sert à crypter et décrypter.

Si le pirate intercepte un message crypté, il ne peut rien en faire s'il ne connaît pas la clé secrète !

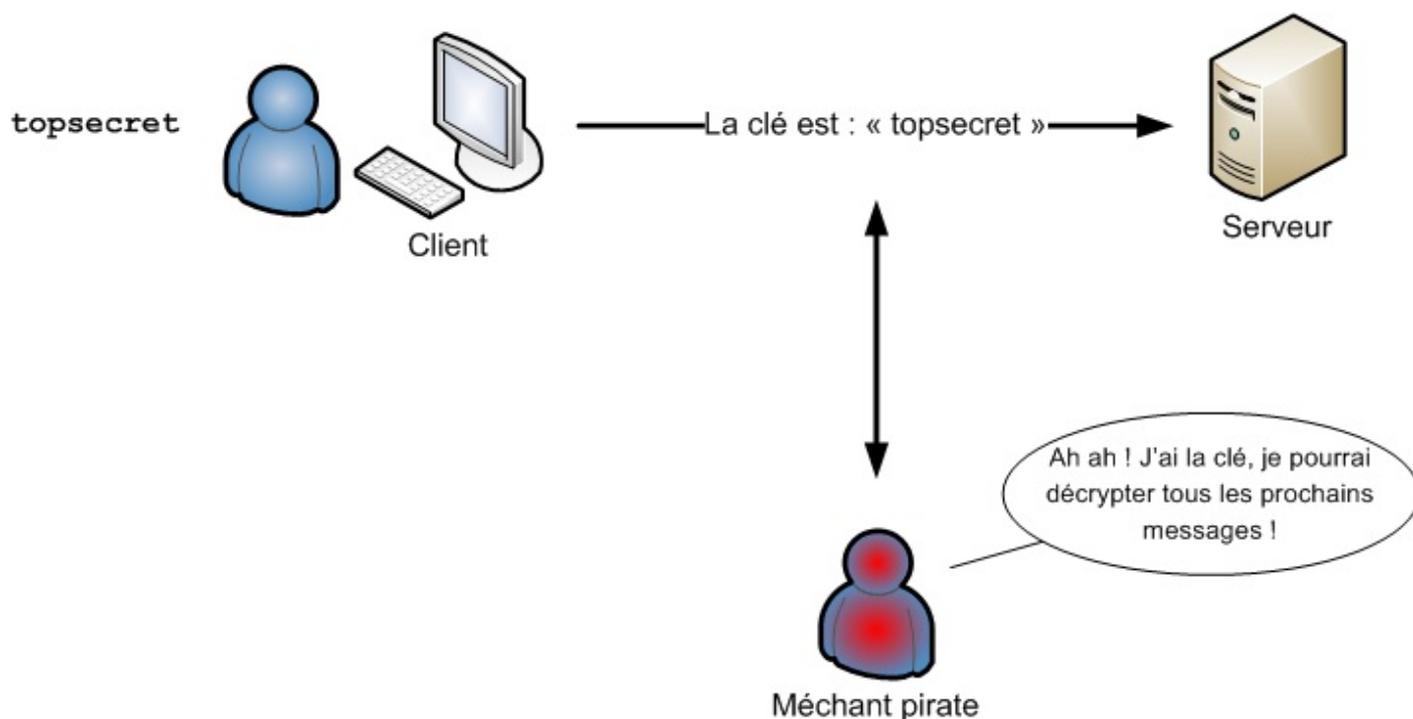


Ah c'est bien ça ! Mais il faut que le client et le serveur connaissent tous les deux la clé de cryptage. Il faut donc que le client envoie d'abord au serveur la clé pour que celui-ci puisse décrypter ses futurs messages...

Très bonne remarque, je vois que vous suivez, c'est bien. 😊

En effet, pour que le schéma qu'on vient de voir puisse fonctionner, il faut que le client et le serveur se soient communiqués

auparavant la clé magique qui sert à crypter et décrypter. Mais comment font-ils pour se l'échanger ? S'ils l'envoient en clair, le pirate va pouvoir l'intercepter et il sera ensuite capable de décrypter tous les messages cryptés qui passeront sur le réseau !



Le cryptage symétrique est donc un cryptage puissant, mais il a un gros défaut : il faut communiquer "discrètement" la clé de cryptage... mais c'est impossible : il faut bien envoyer la clé en clair au début !

...

À moins de... non...

Et pourquoi pas ? Si on cryptait la clé de cryptage lors de son envoi ? 😊

Pour crypter la clé de cryptage symétrique, on va utiliser une autre méthode : le cryptage asymétrique. Avec cette autre méthode on ne risque pas de retrouver le même problème qu'on vient de rencontrer.

Le cryptage asymétrique

Le cryptage symétrique utilise une seule clé pour crypter et décrypter.

Le cryptage asymétrique, lui, utilise une clé pour crypter, et une autre pour décrypter.

Il y a donc 2 clés :

- Une clé dite "**publique**" qui sert à **crypter**.
- Une clé dite "**privée**" qui sert à **décrypter**.

La clé publique ne sert qu'à crypter. Avec ce type d'algorithme, on ne peut décrypter un message que si on connaît la clé privée.

On demande à l'ordinateur de générer une paire de clés : une privée et une publique. Elles vont ensemble. Ne me demandez pas comment il les génère et pourquoi elles vont ensemble, c'est trop compliqué à expliquer. Acceptez juste que l'ordinateur est capable de générer aléatoirement un couple de clés qui vont ensemble.

Prenons un exemple et imaginons que :

- La clé publique est **74A48vXX**.
- La clé privée est **99o0pn9**.

Pour crypter, on utilise la clé publique :



Pour décrypter, la clé publique ne marche pas. Il faut obligatoirement utiliser la clé privée :



Voilà pourquoi on dit que c'est un cryptage asymétrique : il faut 2 clés différentes. L'une d'elle permet de crypter le message, l'autre de le décrypter. Il n'y a pas d'autre moyen.

La clé publique peut être transmise en clair sur le réseau (elle est "publique"). Ce n'est pas grave si un pirate l'intercepte. Par contre la clé privée qui permet de décrypter doit rester secrète.



L'algorithme de cryptage asymétrique le plus connu s'appelle RSA. Si vous voulez savoir comment RSA fonctionne et pourquoi il faut une clé différente pour crypter et pour décrypter, lisez ce [tutoriel sur RSA](#). Attention je vous préviens, il faut aimer les maths. 😊

La création d'un tunnel sécurisé avec SSH

SSH combine cryptage asymétrique et cryptage symétrique

SSH utilise les 2 cryptages : asymétrique et symétrique. Cela fonctionne dans cet ordre :

1. On utilise d'abord le cryptage asymétrique pour s'échanger discrètement une clé secrète de cryptage symétrique.
2. Puis ensuite on utilise tout le temps la clé de cryptage symétrique pour crypter les échanges.



Pourquoi ne pas utiliser uniquement du cryptage asymétrique tout le temps ?

Ce serait possible mais il y a un défaut : le cryptage asymétrique demande beaucoup trop de ressources au processeur. Le cryptage asymétrique est 100 à 1000 fois plus lent que le cryptage symétrique !

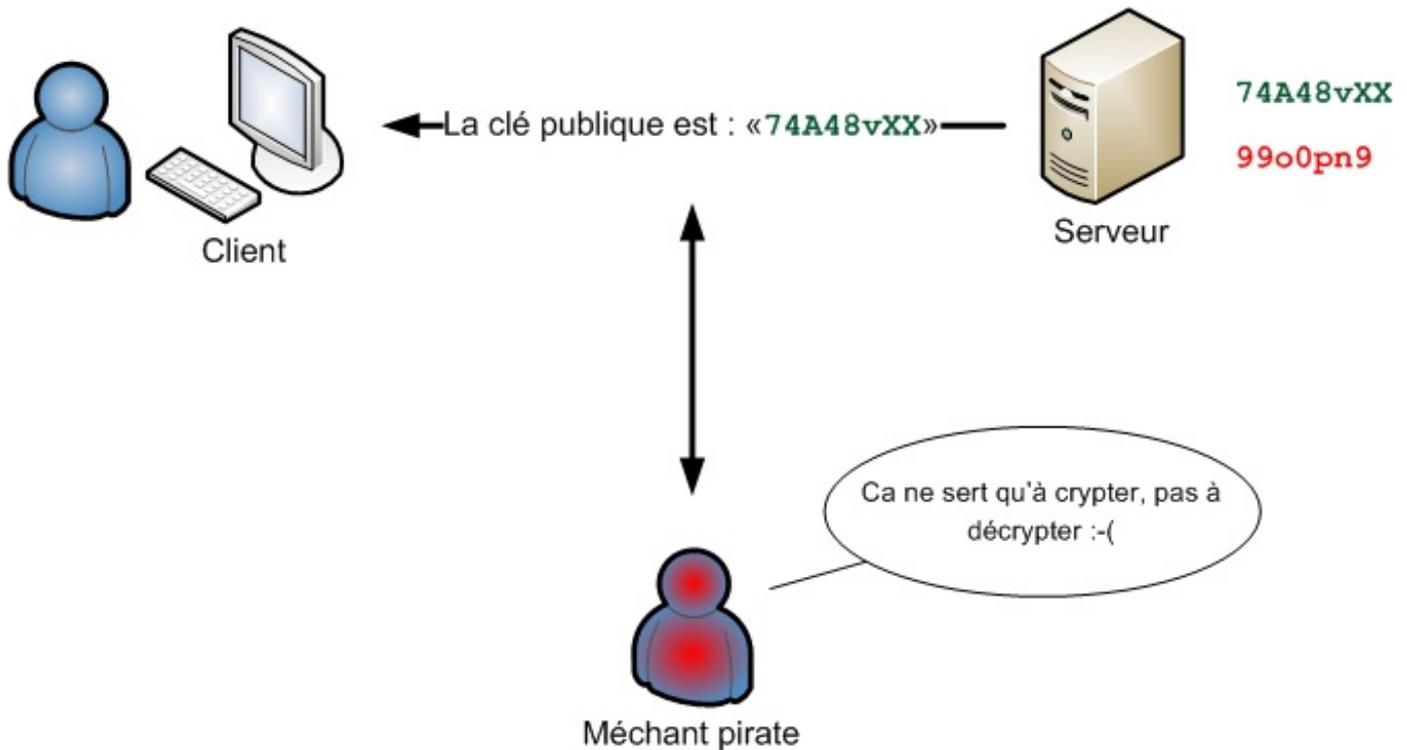
Les ordinateurs s'échangent donc la clé de cryptage symétrique de manière sécurisée (grâce au cryptage asymétrique 😊) et ils peuvent ensuite communiquer plus rapidement en utilisant tout le temps du cryptage symétrique.

Le cryptage asymétrique est donc utilisé seulement au début de la communication, afin que les ordinateurs s'échangent la clé de cryptage symétrique de manière sécurisée. Ensuite, ils ne communiquent que par cryptage symétrique.

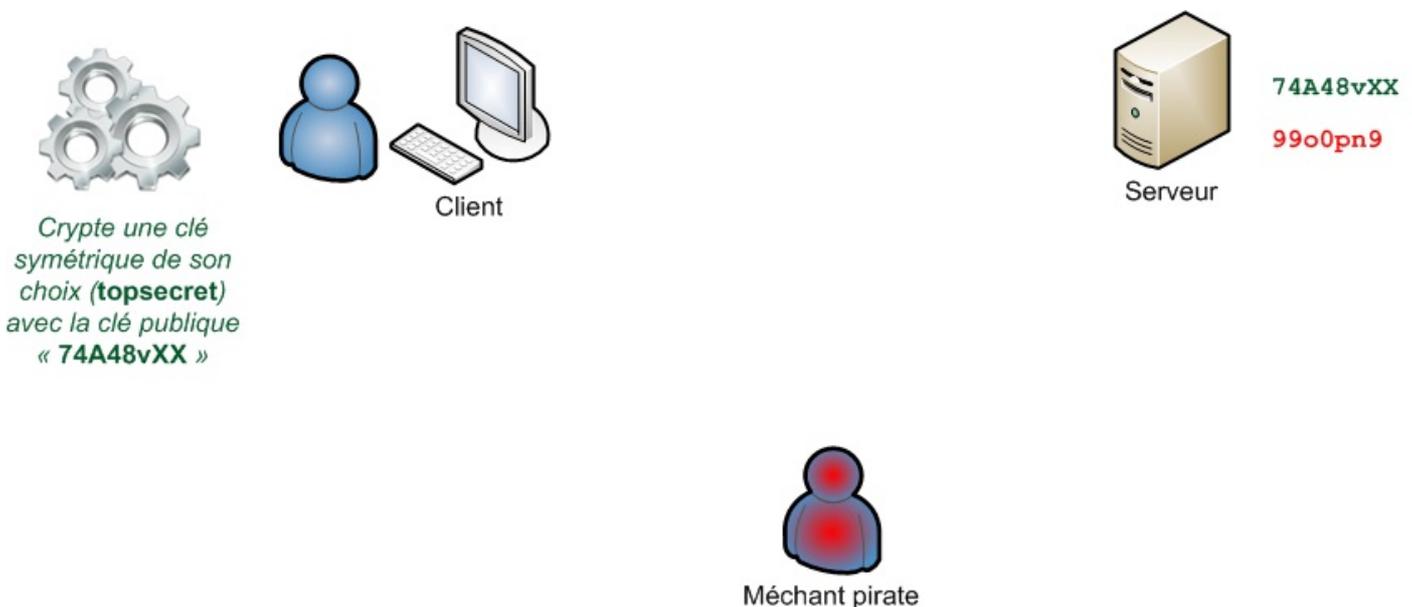
Les étapes pour créer un canal sécurisé avec SSH en images

Je résume en images. On veut s'échanger une clé de cryptage symétrique, mais on ne peut pas le faire en clair sinon le pirate peut l'intercepter. On va donc crypter la clé grâce au cryptage asymétrique.

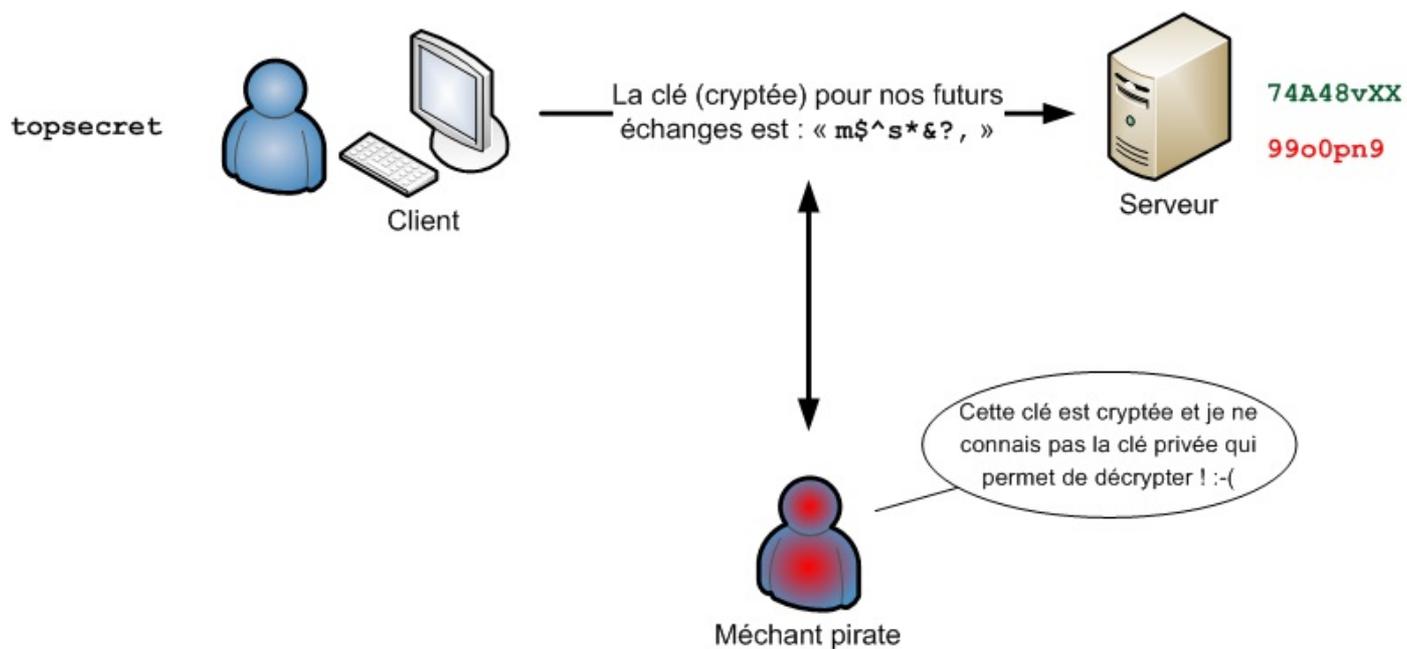
Le serveur envoie la clé publique en clair au client pour qu'il puisse crypter :



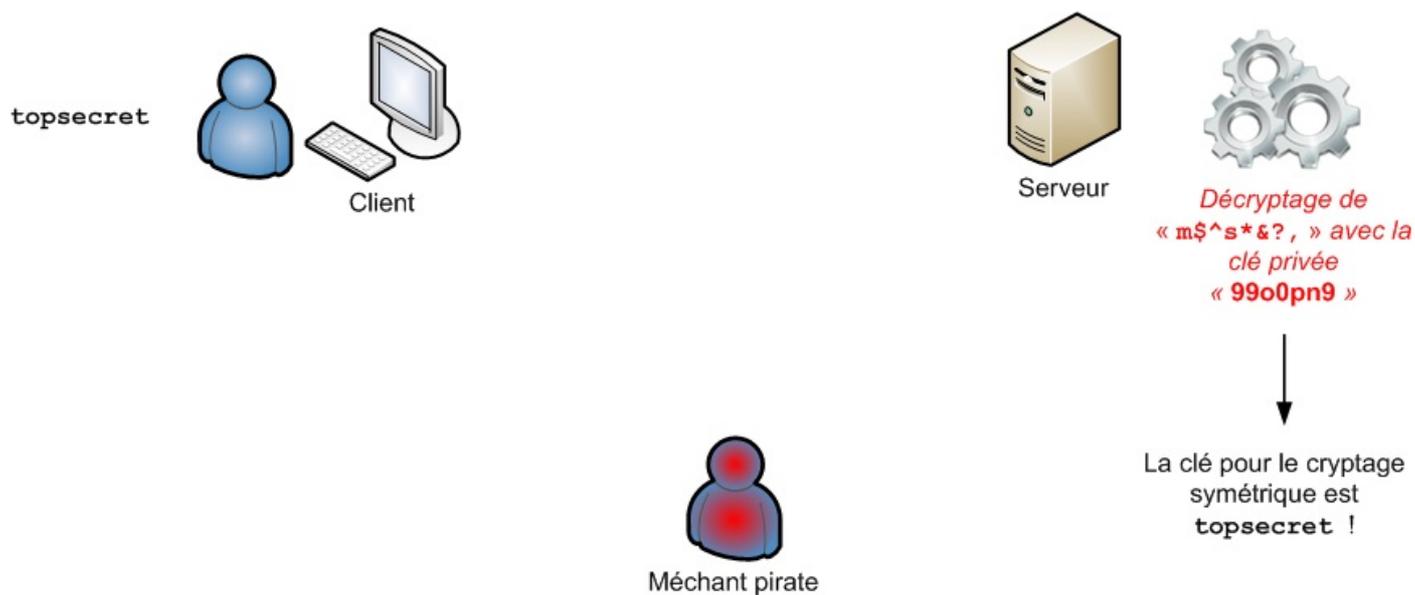
Le client génère une clé de cryptage symétrique (par exemple `topsecret`) qu'il crypte grâce à la clé publique qu'il a reçue :



Le client envoie la clé symétrique cryptée au serveur. Le pirate peut l'intercepter mais il ne peut pas la décrypter car il faut pour cela la clé privée (connue seulement du serveur) :

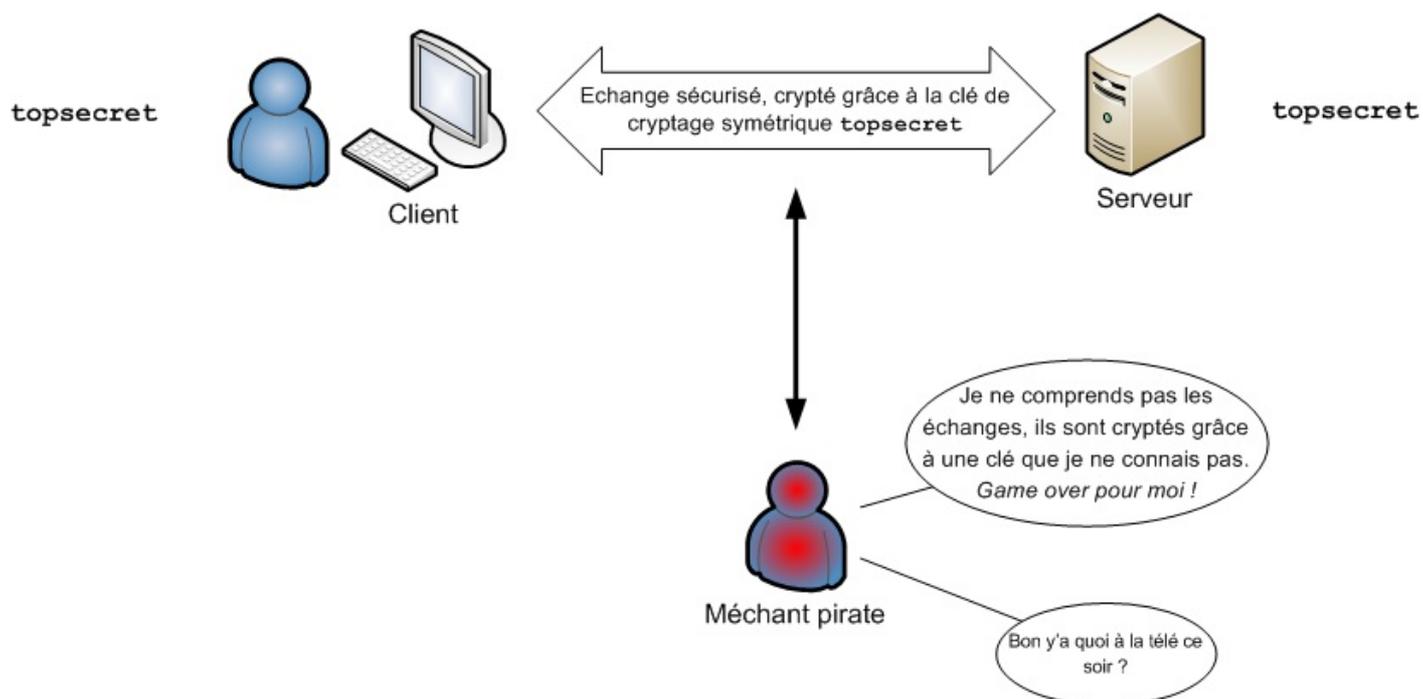


Le serveur décrypte la clé reçue grâce à sa clé privée qu'il a gardée bien au chaud chez lui :



Le client et le serveur connaissent maintenant tous les deux la clé symétrique topsecret, et à aucun moment ils ne se la sont échangée en clair sur le réseau !

Ils peuvent donc s'envoyer des messages cryptés de manière symétrique en toute tranquillité. Ce cryptage est plus rapide, et il est tout aussi sûr que le cryptage asymétrique car le pirate ne connaît pas la clé !

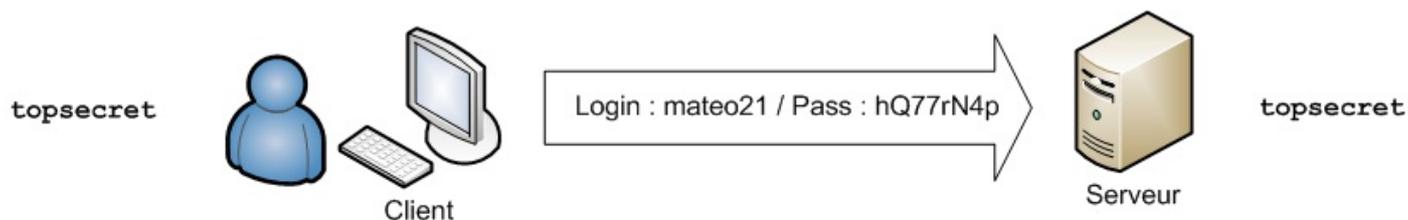


Voilà comment SSH fonctionne pour créer un canal d'échange sécurisé. Tout est crypté grâce à la clé symétrique que le client et le serveur se sont *astucieusement* communiquée.



Maintenant qu'ils discutent de manière sécurisée, que font le client et le serveur ?

Eh bien *seulement* maintenant, le client peut se logger sur le serveur : il peut donner son login et son mot de passe pour se connecter au serveur sans craindre de se les faire voler par le pirate ! 😊



Faut-il savoir tout cela pour utiliser SSH ? 🤔

Non, en fait tout se fait automatiquement. Vous allez juste avoir à rentrer un login et un mot de passe pour vous connecter à votre machine à distance.

Mais j'estime que c'était l'occasion idéale de vous expliquer comment fonctionne le protocole SSH. Ce système est utilisé partout dans le monde ! Plus personne n'envisage de se connecter en Telnet aujourd'hui.

Se connecter avec SSH et PuTTY

Assez de théorie, passons à la pratique ! Vous allez voir, ça sera beaucoup plus simple car les ordinateurs effectuent les cryptages entre eux sans nous demander d'intervenir... et c'est tant mieux. 🤖

A partir de maintenant :

- Soit **vous louez déjà un serveur dédié** (ce qui devrait être le cas d'une minorité d'entre vous). Celui-ci est déjà configuré pour faire serveur SSH, vous n'avez donc rien à faire pour le "transformer" en serveur.

Si vous voulez louer votre serveur dédié, sachez qu'il existe de très nombreux hébergeurs qui le proposent,



comme par exemple [OVH](#). Comme vous pouvez le voir, ça coûte cher (en même temps c'est un ordinateur à part entière que vous louez !).

Sachez qu'il existe aussi des serveurs *low cost* moins chers (moins puissants mais ils peuvent suffire), comme [Kimsufi](#) et [Dedibox](#).

Je vous recommande d'attendre un peu avant de louer un serveur dédié, ça coûte de l'argent et il vaut mieux être sûr d'en avoir vraiment besoin.

- Soit **vous n'avez pas de serveur dédié**, ce qui je suppose est le cas de la plupart d'entre vous. Dans ce cas, nous allons voir comment transformer votre PC en serveur dans les prochains paragraphes.

Transformer sa machine en serveur

Cette étape vous concerne si vous voulez transformer votre PC en serveur. Par exemple, si vous voulez accéder à votre PC depuis un autre endroit (et donc suivre le reste de ce chapitre), vous devez le transformer en serveur au préalable.

Il faut tout simplement installer le paquet `openssh-server` :

Code : Console

```
sudo apt-get install openssh-server
```

Lors de l'installation, vous devriez voir certaines étapes intéressantes s'effectuer automatiquement :

Code : Console

```
Creating SSH2 RSA key; this may take some time ...
Creating SSH2 DSA key; this may take some time ...
* Restarting OpenBSD Secure Shell server sshd [ OK ]
```

RSA et DSA sont 2 algorithmes de cryptage asymétrique. Comme je vous l'ai dit plus tôt, SSH peut travailler avec plusieurs algorithmes de cryptage différents.

Ce que vous voyez là est l'étape de la création d'une paire de clés publiques et privées pour chacun des 2 algorithmes (RSA et DSA).

Ensuite, le programme de serveur SSH (appelé `sshd`) est lancé.

Normalement, le serveur SSH sera lancé à chaque démarrage. Si ce n'est pas le cas, vous pouvez le lancer à tout moment avec la commande suivante :

Code : Console

```
sudo /etc/init.d/ssh start
```

Et vous pouvez l'arrêter avec cette commande :

Code : Console

```
sudo /etc/init.d/ssh stop
```

Normalement vous ne devriez pas avoir besoin de configurer quoi que ce soit, mais sachez au besoin que le fichier de configuration se trouve dans `/etc/ssh/ssh_config`. Il faudra recharger SSH avec la commande `sudo /etc/init.d/ssh reload` pour que les changements soient pris en compte.

Voilà, votre machine est désormais un serveur SSH ! Vous pouvez vous y connecter depuis n'importe quelle machine Linux ou Windows dans le monde.

Nous commencerons dans un premier temps par voir comment accéder à votre PC à distance depuis une machine Linux.

Se connecter via SSH à partir d'une machine Linux

Toutes les machines équipées de Linux proposent la commande `ssh` qui permet de se connecter à distance à une autre machine.



A partir d'ici, je suppose que vous avez installé `openssh-server` et que votre machine est allumée. L'idéal serait d'aller chez un ami qui a Linux (ou d'utiliser un autre PC de chez vous équipé de Linux).

Ouvrez une console sur le PC de votre ami, et utilisez la commande `ssh` comme ceci :

Code : Console

```
ssh login@ip
```

Il faut remplacer "login" par votre login (mateo21 dans mon cas) et "ip" par l'adresse IP de votre ordinateur.



Si vous vous connectez depuis chez un ami, il vous faut entrer l'IP internet de votre PC que vous pouvez obtenir en allant sur <http://www.whatismyip.com> par exemple.

Si vous vous connectez depuis un autre PC chez vous (sur le même réseau local), il vous faut entrer l'IP locale que vous devriez voir en tapant la commande `ifconfig` (par exemple 192.168.0.3).

Si vraiment vous n'avez ni ami sous Linux ni second PC dans la maison, vous pouvez simuler une connexion réseau en vous connectant de votre PC vers votre PC. Utilisez pour cela l'IP 127.0.0.1 (ou le mot `localhost`), ça marche toujours.

Si je suis chez un ami et que l'IP internet de mon ordinateur est 87.112.13.165, je vais taper :

Code : Console

```
ssh mateo21@87.112.13.165
```

Si, faute de mieux, vous voulez tester en vous connectant chez vous depuis chez vous, vous pouvez taper :

Code : Console

```
ssh mateo21@localhost
```

Cette seconde méthode marche toujours, mais c'est moins impressionnant parce que vous ne faites que simuler une connexion réseau. 😊

Normalement, le serveur devrait répondre au bout d'un moment et vous devriez voir quelque chose comme :

Code : Console

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.  
RSA key fingerprint is 49:d9:2d:2a:df:fd:80:ab:e9:eb:59:37:58:34:de:f7.  
Are you sure you want to continue connecting (yes/no)?
```



Si vous n'avez pas de réponse du serveur, vérifiez que vous ne vous êtes pas trompé d'IP. Vérifiez aussi que le port 22 n'est pas bloqué par un firewall, car c'est celui utilisé par SSH par défaut.

Si le serveur tourne sur un autre port, il faudra préciser le numéro de ce port comme ceci :

```
ssh mateo21@87.112.13.165 -p 12451 (si le serveur fonctionne sur le port 12451 au lieu du port 22).
```

Que se passe-t-il ? On vous dit que le fingerprint (empreinte) du serveur est : 49:d9:2d:2a:df:fd:80:ab:e9:eb:59:37:58:34:de:f7. C'est un numéro unique qui vous permet d'identifier le serveur. Si demain quelqu'un essaie de se faire passer pour le serveur, le fingerprint changera forcément et vous saurez qu'il se passe quelque chose d'anormal. Ne vous inquiétez pas, SSH vous avertira de manière très claire si cela arrive.

En attendant, tapez "yes" pour confirmer que c'est bien le serveur auquel vous voulez vous connecter. Le serveur et le client vont alors s'échanger une clé de cryptage comme je vous l'ai expliqué un peu plus tôt. Normalement, au bout de quelques secondes le serveur devrait vous demander votre mot de passe :

Code : Console

```
mateo21@localhost's password:
```

Vous pouvez entrer votre mot de passe en toute sécurité, la communication est cryptée. 😊

Si vous rentrez le bon mot de passe, la console du PC de votre ami (ou votre propre console) devrait vous afficher un message de bienvenue puis un *prompt* qui correspond à la console de votre PC. Bravo, vous êtes connecté !

Code : Console

```
mateo21@mateo21-desktop:~$
```

Si on ne vous affiche pas d'erreur, c'est que vous êtes bien loggé et que vous travaillez désormais sur votre machine à distance ! Vous pouvez effectuer toutes les opérations que vous voulez comme si vous étiez chez vous.

Essayez de parcourir les dossiers pour voir que ce sont bien les vôtres, et amusez-vous (pourquoi pas) à créer un fichier (avec nano). Lorsque vous reviendrez sur votre PC vous l'y retrouverez. 😊

Vous pouvez aussi commander l'exécution d'un programme, d'une recherche, etc. Vous savez déjà comment lancer un programme en tâche de fond pour qu'il continue même quand vous n'êtes pas connecté à la machine (vous vous souvenez de nohup et de screen ?).

Pour vous déconnecter, tapez "logout" ou son équivalent : la combinaison de touches Ctrl + D.

Se connecter via SSH à partir d'une machine Windows

Si vous voulez avoir accès à la console de votre machine Linux mais que vous n'avez pas d'autre machine Linux sous la main, pas de panique ! Il existe des programmes pour Windows faits pour ça. Le plus connu d'entre eux, et celui que j'utilise personnellement, s'appelle **PuTTY**.

Vous pouvez [télécharger PuTTY sur son site officiel](#).

Je sais, vous devez vous dire que ce n'est pas très clair et que vous ne voulez pas chercher sur quel lien cliquer sur cette page.

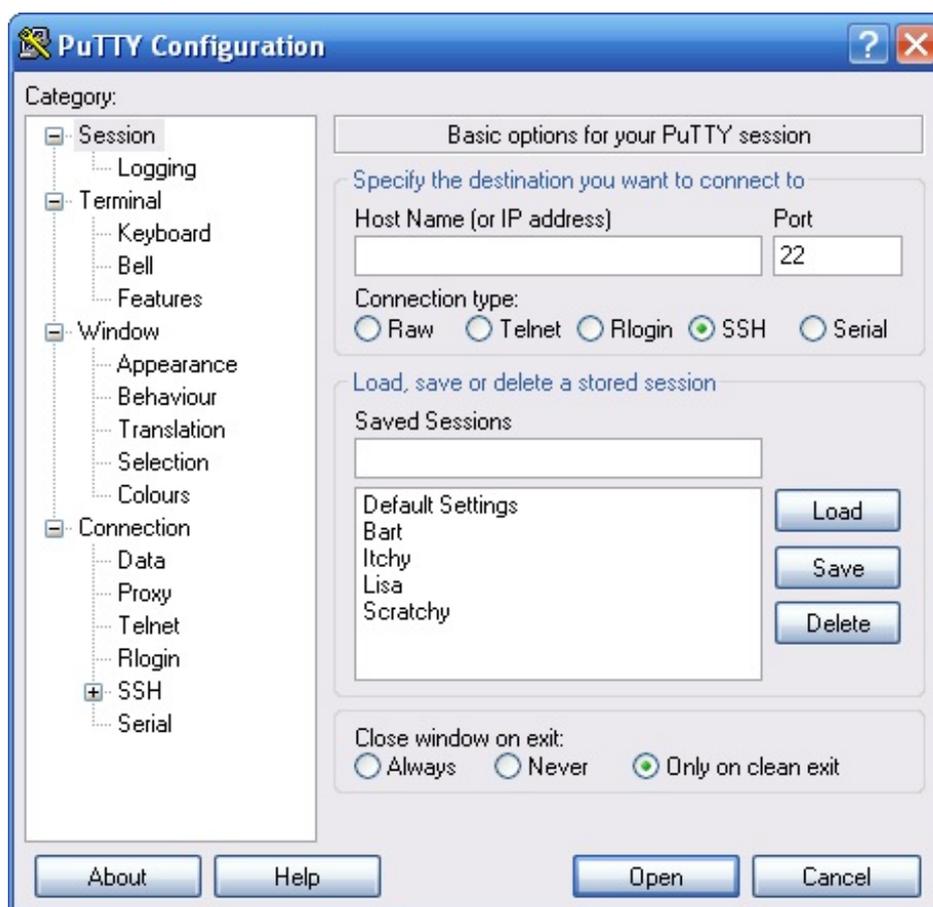


Repérez la section "Binaries". C'est un tableau. Vous avez le choix entre :

- Cliquer sur "putty.exe" pour télécharger le programme principal. Il ne nécessite pas d'installation.
- Cliquer sur le programme d'installation (par exemple "putty-0.60-installer.exe"). Celui-ci installera PuTTY et d'autres utilitaires dont vous aurez besoin dans quelques minutes.

Putty.exe suffit, mais je vous recommande donc de prendre le package complet en récupérant le programme d'installation.

Une fois que c'est fait et installé, lancez Putty. Une fenêtre comme celle-ci devrait s'afficher :

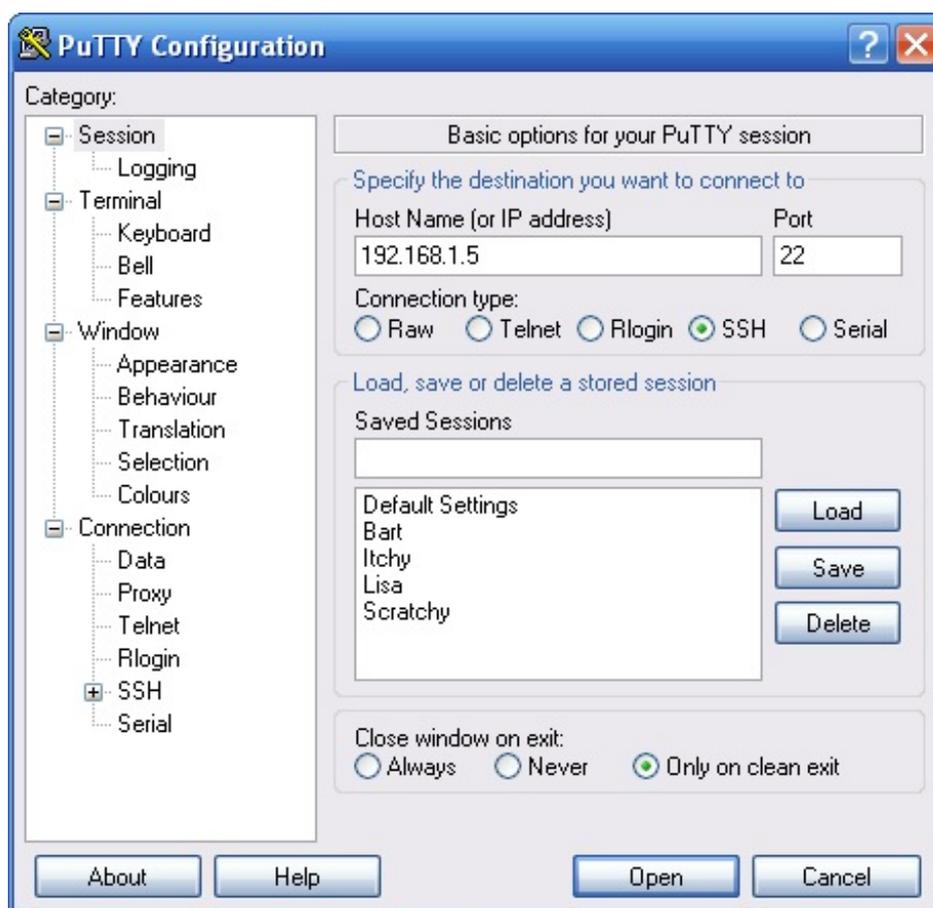


Il y a beaucoup de pages d'options, comme vous pouvez le voir au niveau de la section "Category" sur le côté. Pour le moment, pas de panique, vous avez juste besoin de remplir le champ en haut "Host Name (or IP address)". Entrez-y l'adresse IP de votre ordinateur sous Linux.



J'ai donné quelques explications à propos de l'adresse IP un peu plus haut lorsque j'ai parlé de la connexion SSH depuis Linux. Lisez donc les paragraphes précédents si vous voulez plus d'informations à ce sujet.

Dans mon cas, je vais rentrer l'adresse IP de mon PC sous Linux situé sur mon réseau local (192.168.1.5) :



Vous pouvez changer le numéro du port si ce n'est pas 22, mais normalement c'est 22 par défaut.

Ensuite, vous n'avez plus qu'à cliquer sur le bouton tout en bas "Open" pour lancer la connexion. Rien d'autre !



Si vous voulez sauvegarder l'IP et les paramètres pour ne pas retaper ça à chaque fois, donnez un nom à cette connexion (par exemple le nom de votre ordinateur) dans le champ sous "Saved Sessions", puis appuyez sur le bouton Save. La prochaine fois, vous n'aurez qu'à double-cliquer sur le nom de votre PC dans la liste pour vous y connecter directement.

La première fois que vous vous connectez à votre serveur, PuTTY devrait vous demander une confirmation comme ceci :



C'est la même chose que sous Linux : on vous donne l'empreinte (*fingerprint*) de votre serveur. Vous devez confirmer que c'est bien chez lui que vous voulez vous connecter. Cliquez sur Oui pour confirmer.

A l'avenir, on ne vous reposera plus la question. Par contre, si le fingerprint change, un gros message d'avertissement

s'affichera. Cela signifiera soit que le serveur a été réinstallé, soit que quelqu'un est en train de se faire passer pour le serveur (c'est ce qu'on appelle une attaque *man-in-the-middle*). Cela ne devrait fort heureusement pas vous arriver, du moins on l'espère.

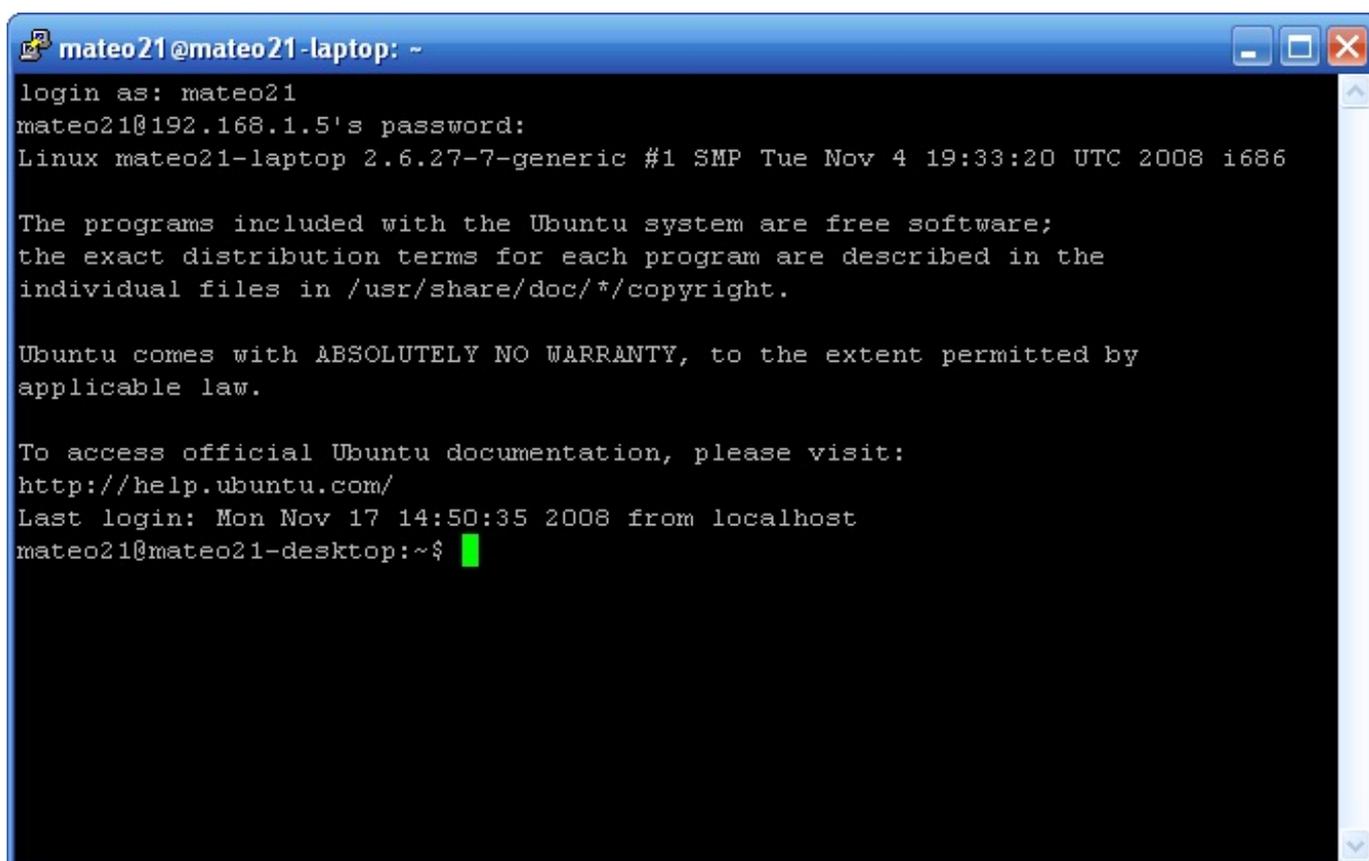


Le serveur vous demande alors le login et le mot de passe :

A screenshot of a PuTTY terminal window titled "192.168.1.5 - PuTTY". The terminal shows a login prompt: "login as: mateo21" followed by "mateo21@192.168.1.5's password:" and a green cursor. The terminal background is black, and the text is white. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner.

Rappelez-vous qu'il est normal que les caractères ne s'affichent pas quand vous tapez votre mot de passe. Il n'y a même pas d'étoiles pour des raisons de sécurité, afin que quelqu'un ne soit pas tenté de compter le nombre de caractères en regardant derrière votre épaule !

Si tout est bon, vous devriez être connecté à votre machine !



```
mateo21@mateo21-laptop: ~
login as: mateo21
mateo21@192.168.1.5's password:
Linux mateo21-laptop 2.6.27-7-generic #1 SMP Tue Nov 4 19:33:20 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
Last login: Mon Nov 17 14:50:35 2008 from localhost
mateo21@mateo21-desktop:~$
```

Et voilà, vous êtes chez vous ! Vous pouvez faire ce qui vous chante : lire vos fichiers, écrire des fichiers, lancer une recherche, exécuter un programme, bref vous êtes chez vous. 😊

Pour vous déconnecter, tapez "logout" ou son équivalent : la combinaison de touches Ctrl + D.

L'identification automatique par clé

Il y a plusieurs façons de s'authentifier sur le serveur, pour qu'il sache que c'est bien vous. Les 2 plus utilisées sont :

- Authentification par mot de passe.
- Authentification par clé publique et privée du client.

Nous avons pour le moment vu uniquement l'authentification par mot de passe (le serveur vous demandait votre mot de passe). Il est possible d'éviter que l'on vous demande votre mot de passe à chaque fois grâce à une authentification spéciale par clés. Cette méthode d'authentification est plus complexe à mettre en place, mais elle est ensuite plus pratique.



Avec cette nouvelle méthode d'authentification, c'est le client qui va générer une clé publique et une clé privée. Les rôles sont un peu inversés.

L'avantage est qu'on ne vous demandera pas votre mot de passe à chaque fois pour vous connecter. Si vous vous connectez très régulièrement à un serveur, c'est vraiment utile. Si vous faites bien les choses, cette méthode est tout aussi sûre que l'authentification par mot de passe.

Je vais là encore distinguer les 2 cas :

- Vous essayez de vous connecter depuis une machine Linux.
- Vous essayez de vous connecter depuis une machine Windows (avec Putty).

Authentification par clé depuis Linux

Pour mettre en marche ce mode d'authentification, nous allons d'abord devoir effectuer des opérations sur la machine du client puis nous enverrons le résultat au serveur.

Opérations sur la machine du client

Il faut tout d'abord vous rendre sur la machine du client et taper la commande suivante pour générer une paire de clés publique/privée :

Code : Console

```
ssh-keygen -t rsa
```

(Vous pouvez remplacer *rsa* par *dsa* si vous voulez utiliser l'autre algorithme de cryptage, mais ça n'a pas vraiment d'importance ici)

Lorsque vous tapez cette commande, vous allez voir plusieurs messages s'afficher et on va vous demander quelques petites précisions :

Code : Console

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mateo21/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mateo21/.ssh/id_rsa.
Your public key has been saved in /home/mateo21/.ssh/id_rsa.pub.
The key fingerprint is:
b7:22:94:aa:8c:fb:d3:ef:53:86:df:b9:37:40:bd:4d mateo21@mateo21-laptop
The key's randomart image is:
+--[ RSA 2048 ]-----+
|
|          .
|         . . . . E
|        o.S.. +
|       o. o.... .
|      .. .+...o
|     o... ..o o
|    oo+. oo. .o .
+-----+

```

Dans un premier temps, le client génère une paire de clés ("Generating public/private rsa key pair"). Il doit ensuite sauvegarder ces clés dans des fichiers (un pour la clé publique, un pour la clé privée). On vous propose une valeur par défaut : je vous conseille de ne rien changer et de taper simplement Entrée.

Ensuite, on vous demande une *passphrase*. C'est une phrase de passe qui va servir à crypter la clé privée pour une meilleure sécurité. Là vous avez 2 choix :

- Soit vous tapez Entrée directement sans rien écrire, et la clé ne sera pas cryptée sur votre machine.
- Soit vous tapez un mot de passe de votre choix, et la clé sera cryptée.

Tout le monde ne met pas une phrase de passe. En fait ça dépend le risque que vous avez que quelqu'un d'autre utilise la

machine du client et puisse lire le fichier contenant la très secrète clé privée. Si le PC du client est votre PC chez vous et que personne d'autre ne l'utilise, il y a assez peu de risque (à moins d'avoir un virus, un spyware...). Si c'est en revanche un PC public, je vous recommande vivement de mettre une passphrase pour chiffrer la clé qui sera enregistrée.

Si vous hésitez entre les 2 méthodes, je vous recommande de rentrer une passphrase : c'est quand même la méthode la plus sûre.

Envoyer la clé publique au serveur

Il faut maintenant envoyer au serveur votre clé publique pour qu'il puisse vous crypter des messages.

Votre clé **publique** devrait se trouver dans `~/.ssh/id_rsa.pub` (pub comme public). `~` correspond à votre home (`/home/mateo21/` dans mon cas). Notez que `.ssh` est un dossier caché.

Votre clé **privée**, elle, se trouve dans `~/.ssh/id_rsa`. Ne la communiquez à personne ! Elle est normalement cryptée si vous avez entré une passphrase, ce qui constitue une sécurité de plus.

Vous pouvez vous rendre dans le dossier `.ssh` déjà pour commencer :

Code : Console

```
cd ~/.ssh
```

Si vous faites un `ls` vous devriez voir ceci :

Code : Console

```
$ ls
id_rsa  id_rsa.pub  known_hosts
```

Les 3 fichiers sont :

- `id_rsa` : votre clé privée, qui doit rester secrète. Elle est cryptée si vous avez rentré une passphrase.
- `id_rsa.pub` : la clé publique que vous pouvez communiquer à qui vous voulez, et que vous devez envoyer au serveur.
- `known_hosts` : c'est la liste des fingerprint que votre PC de client tient à jour. Ça lui permet de se souvenir de l'identité des serveurs et de vous avertir si, un jour, votre serveur est remplacé par un autre (qui pourrait être celui du pirate !). Je vous en ai déjà parlé un peu plus tôt.

L'opération consiste à envoyer la clé publique (`id_rsa.pub`) au serveur et à l'ajouter à son fichier "authorized_keys" (clés autorisées). Le serveur y garde une liste des clés qu'il autorise à se connecter.

Le plus simple pour cela est d'utiliser la commande spéciale `ssh-copy-id`. Utilisez-la comme ceci :

Code : Console

```
ssh-copy-id -i id_rsa.pub login@ip
```

Remplacez-y votre login et l'ip de votre serveur.

Code : Console

```
$ ssh-copy-id -i id_rsa.pub mateo21@88.92.107.7
mateo21@88.92.107.7's password:
Now try logging into the machine, with "ssh 'mateo21@localhost'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

Si vous devez vous connecter au serveur par un autre port que celui par défaut, basez-vous sur la commande suivante : `ssh-copy-id -i id_rsa.pub "-p 14521 mateo21@88.92.107.7"`.

On vous demande votre mot de passe (celui de votre compte, pas la passphrase). En fait vous vous connectez par mot de passe encore une fois, pour pouvoir ajouter sur le serveur votre clé publique.

La clé est ensuite automatiquement ajoutée à `~/.ssh/authorized_keys` sur le serveur. On vous invite à vérifier si l'opération s'est bien déroulée en ouvrant le fichier `authorized_keys`, ce que vous pourrez faire plus tard si vous le voulez.

Se connecter !

Maintenant, connectez-vous au serveur comme vous le faisiez auparavant :

Code : Console

```
ssh login@ip
```

Par exemple :

Code : Console

```
$ ssh mateo21@88.92.107.7
Enter passphrase for key '/home/mateo21/.ssh/id_rsa':
```

On vous demande la phrase de passe pour décrypter votre clé privée. Rentrez-la.

Normalement, si tout va bien, vous devriez être alors connecté au serveur.



Où je suis le dernier des nuls, ou alors c'est ce système qui est nul. Auparavant on me demandait mon mot de passe. Maintenant on me demande une phrase de passe pour décrypter la clé privée. Où est le progrès ???

Je comprends votre frustration. 😞

En fait, si vous n'aviez pas mis de phrase de passe, on ne vous aurait rien demandé et vous auriez été directement connecté. Heureusement, il y a une solution pour ceux qui ont choisi la sécurité en utilisant une phrase de passe, mais qui ne veulent quand même pas avoir à la rentrer à chaque fois : l'agent SSH.

L'agent SSH

L'agent SSH est un programme qui tourne en arrière-plan en mémoire. Il retient les clés privées pendant toute la durée de votre session.

Tout ce que vous avez à faire est de lancer le programme `ssh-add` sur le PC du client :

Code : Console

```
$ ssh-add
Enter passphrase for /home/mateo21/.ssh/id_rsa:
Identity added: /home/mateo21/.ssh/id_rsa (/home/mateo21/.ssh/id_rsa)
```

Il va automatiquement chercher votre clé privée. Pour la décrypter, il vous demande la passphrase. Rentrez-la.

Maintenant que c'est fait, à chaque fois que vous vous connecterez à un serveur, vous n'aurez plus besoin de rentrer la passphrase. Essayez de vous connecter à votre serveur pour voir ! 😊



L'intérêt de l'agent SSH est qu'il ne vous demande la passphrase qu'une seule fois au début. Ensuite, vous pouvez vous connecter plusieurs fois au même serveur, ou même à plusieurs serveurs différents, le tout sans avoir besoin de retaper votre passphrase !

Authentification par clé depuis Windows (PuTTY)

Il est tout à fait possible d'utiliser l'authentification par clé avec Putty. C'est là justement qu'il est recommandé d'avoir pris l'installateur, et non pas juste le programme principal putty.exe.

Le principe est le même que sous Linux : il faut d'abord qu'on génère une paire de clés sur le PC du client, puis qu'on les envoie au serveur. Nous retrouverons aussi un équivalent de l'agent SSH pour éviter d'avoir à rentrer une passphrase à chaque fois.

Commençons par la génération des clés.

Générer une paire de clés (publique et privée) avec Puttygen

Normalement, vous devriez avoir installé un programme appelé Puttygen (il se trouvait dans l'installateur de Putty). Lancez-le :

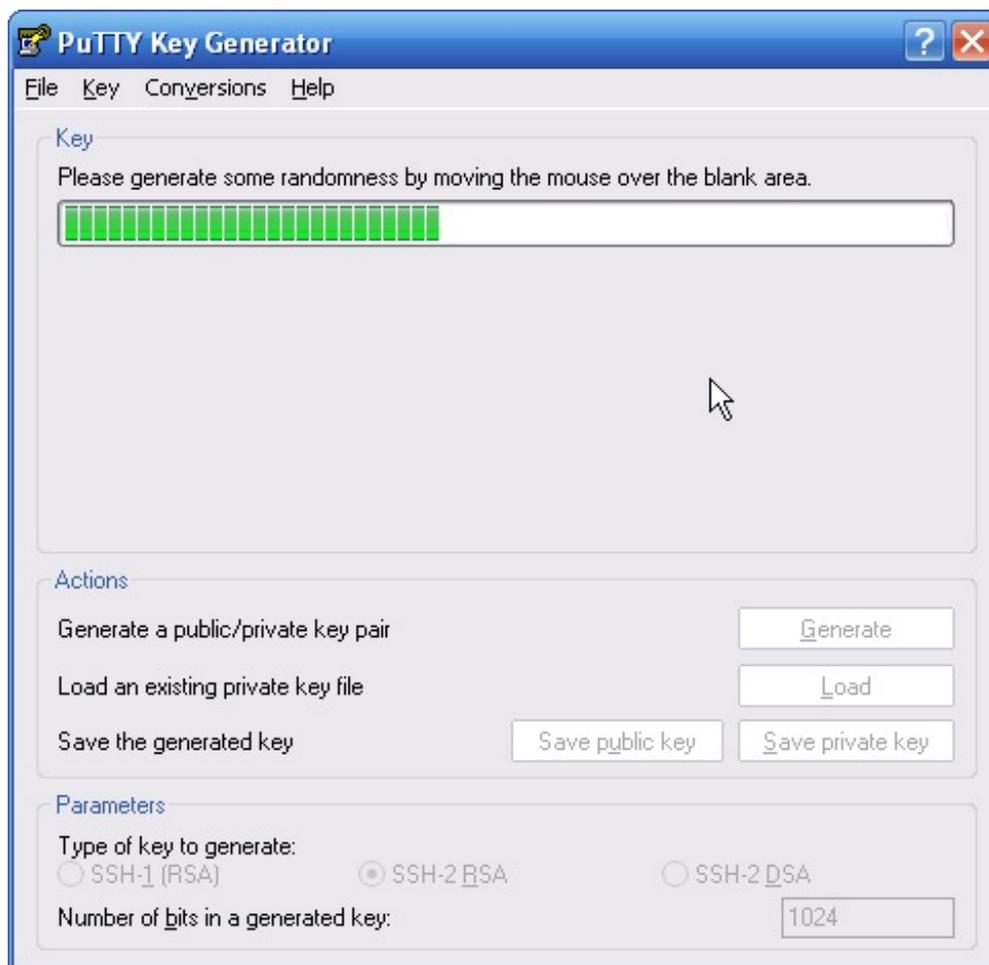


En bas de la fenêtre vous pouvez choisir vos paramètres : algorithme de cryptage et puissance du cryptage. Les valeurs par défaut ici (RSA 1024 bits) sont tout à fait convenables. Vous pouvez les changer, mais sachez qu'elles sont sûres et que vous pouvez donc vous en contenter.

Cliquez sur le bouton "Generate". Le programme va générer une paire de clés (publique et privée).



Pour aider le programme à générer cette paire, il vous propose quelque chose d'assez amusant : vous devez bouger la souris dans la fenêtre. Comme vous allez le faire aléatoirement, cela aidera Puttygen à générer des clés aléatoires. Sous Linux, on utilise d'autres méthodes pour générer des clés aléatoirement (il faut dire qu'en console on n'a pas de souris 🐒).



Génération des clés grâce aux mouvements de la souris

Une fois que c'est fait, on vous affiche la clé publique :



(Comme vous le voyez, ça ne me dérange pas que tout le monde voie ma clé publique. Le principe c'est justement que tout le monde peut voir cette clé, mais on ne peut rien en faire. Par contre la clé privée doit rester secrète.)

Vous pouvez choisir d'entrer une passphrase ou non. Comme je vous l'ai expliqué plus tôt, cela renforce la sécurité en cryptant la clé privée.

Saisissez la passphrase dans les champs "Key passphrase" et "Confirm passphrase".

Ensuite, enregistrez la clé publique dans un fichier en cliquant sur "Save public key". Vous pouvez nommer ce fichier comme vous voulez, par exemple cle.pub. Enregistrez-le où vous voulez.

Puis, enregistrez la clé privée en cliquant sur "Save private key". Donnez-lui l'extension .ppk : cle.ppk par exemple.

Ne fermez pas encore Puttygen.

Envoyer la clé publique au serveur

Comme sous Linux tout à l'heure, il faut envoyer la clé publique au serveur pour qu'il nous autorise à nous connecter par clé. Le problème, c'est qu'il n'y a pas de commande pour le faire automatiquement depuis Windows. Il va falloir ajouter la clé à la main dans le fichier authorized_keys. Heureusement ce n'est pas très compliqué.

Ouvrez Putty et connectez-vous au serveur comme auparavant (en rentrant votre mot de passe habituel). Rendez-vous dans ~/.ssh :

Code : Console

```
cd ~/.ssh
```



Si le dossier `.ssh` n'existe pas, pas de panique, créez-le : `mkdir .ssh`

Rajoutez votre clé publique à la fin du fichier `authorized_keys` (s'il n'existe pas il sera créé). Vous pouvez utiliser la commande suivante :

Code : Console

```
echo "votre_cle" >> authorized_keys
```



Rappel : votre clé publique est affichée dans Puttygen, que vous ne devriez pas avoir fermé. Pour coller la clé dans la console, utilisez la combinaison de touches `Shift + Inser` plutôt que `Ctrl + V`.

Par exemple :

Code : Console

```
echo "ssh-rsa AAAAB3NzaC1yc2E [...] AAAABJQAP++UWBOkLp0= rsa-key-20081117" >> authorized_keys
```

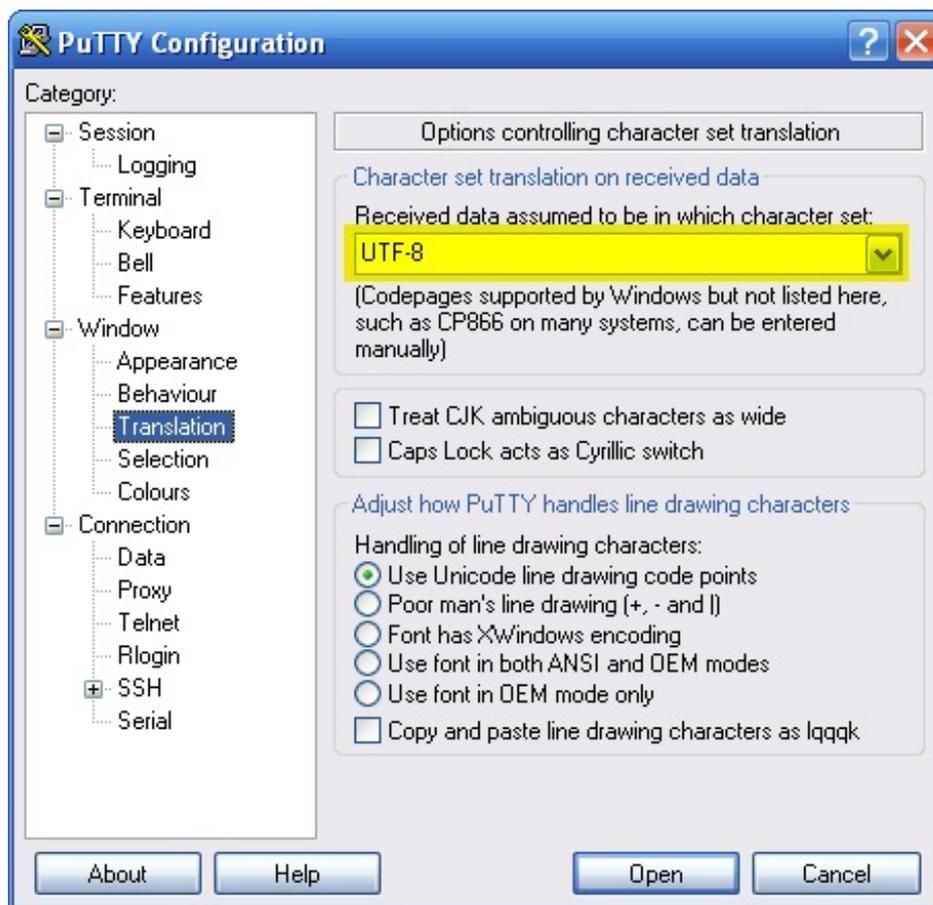
Voilà, c'est fait. 😊

Déloggez-vous, et relancez Putty. On va maintenant le configurer pour qu'il se connecte à l'aide de la clé.

Configurer Putty pour qu'il se connecte avec la clé

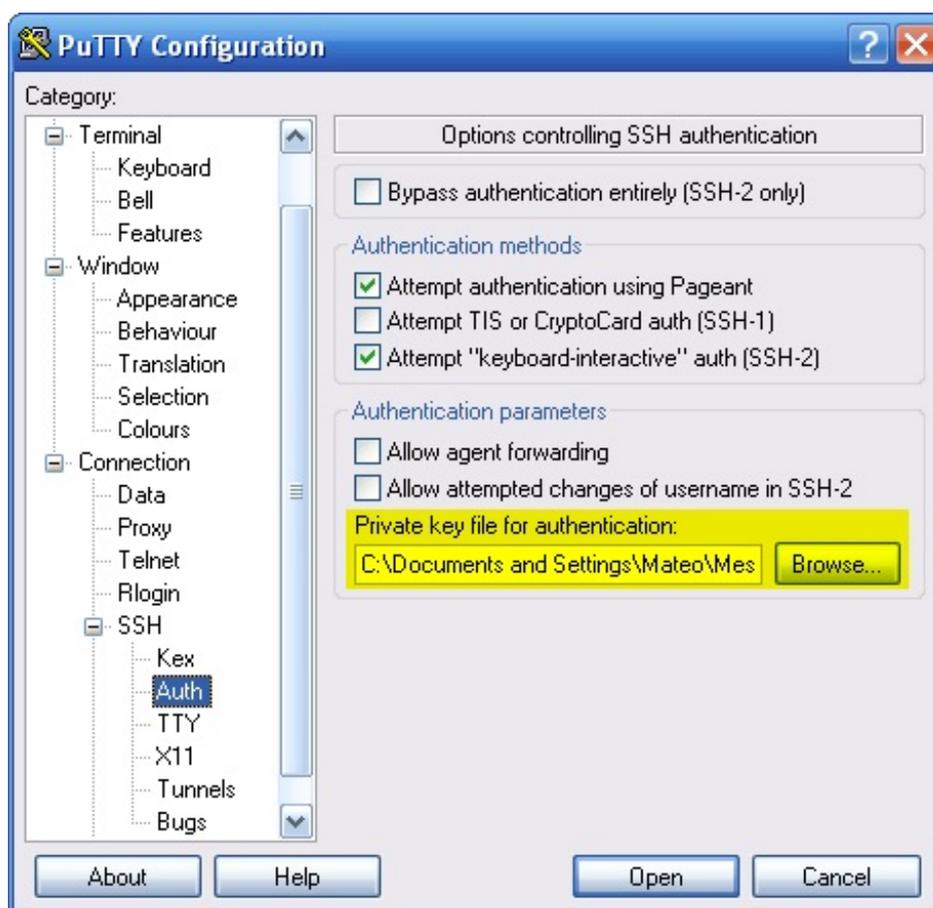
Une fois Putty ouvert, rendez-vous dans la section "Window > Translation" pour commencer. Ça n'a pas de rapport direct avec les clés, mais cela vous permettra de régler le problème des accents qui s'affichent mal dans la console si vous l'avez rencontré.

Réglez la valeur de la liste déroulante à UTF-8 :

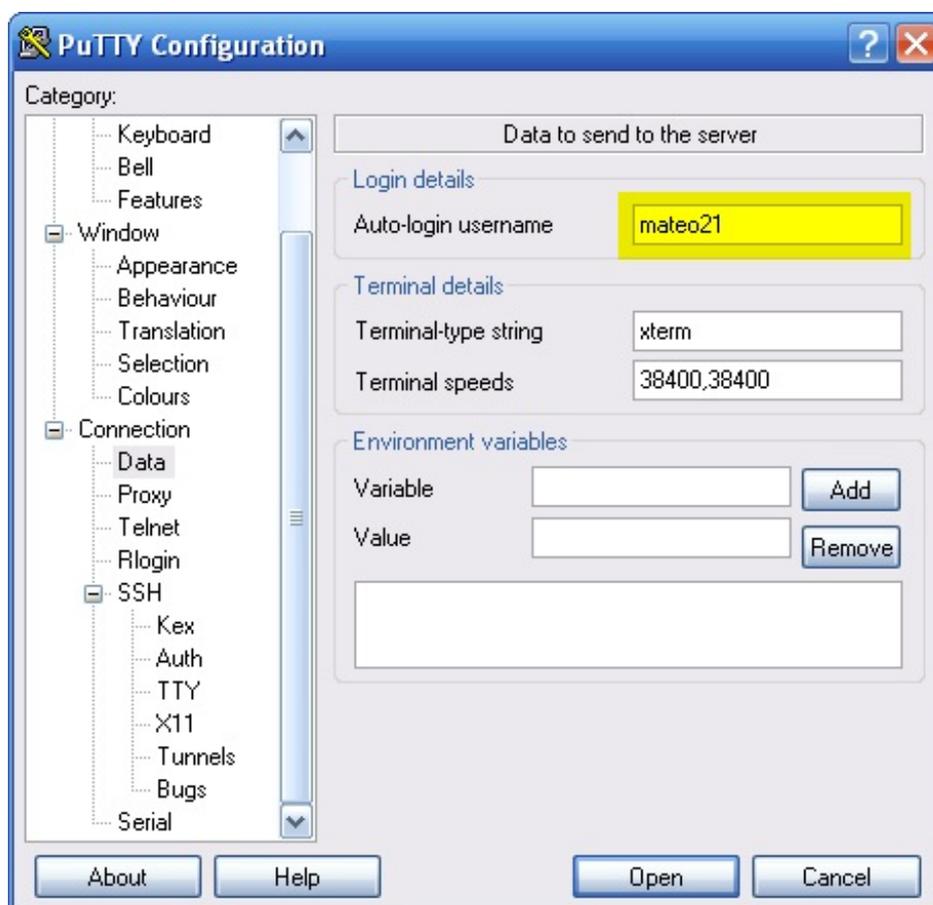


La plupart des serveurs encodent désormais les caractères en UTF-8, cela devrait donc vous éviter des soucis d'affichage.

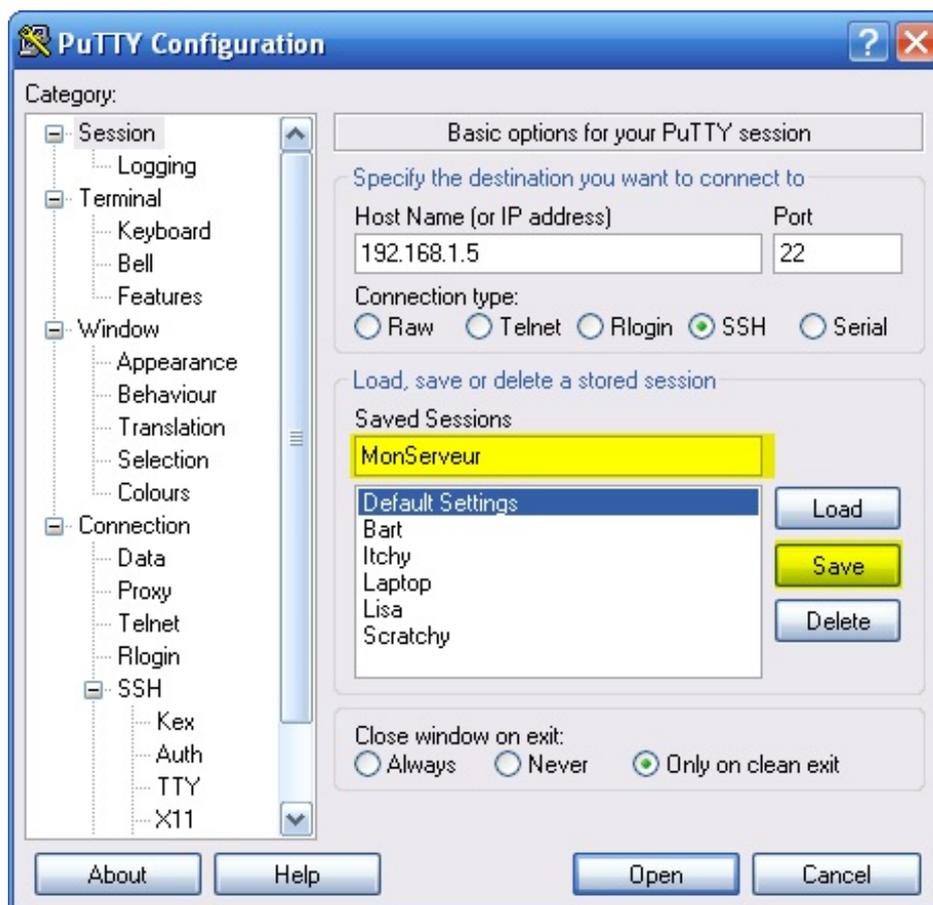
Maintenant, rendez-vous dans "Connection > SSH > Auth". Cliquez sur le petit bouton "Browse" pour sélectionner votre clé privée :



Je vous recommande aussi d'aller dans "Connection > Data" et de rentrer votre login dans "Auto-login username" :



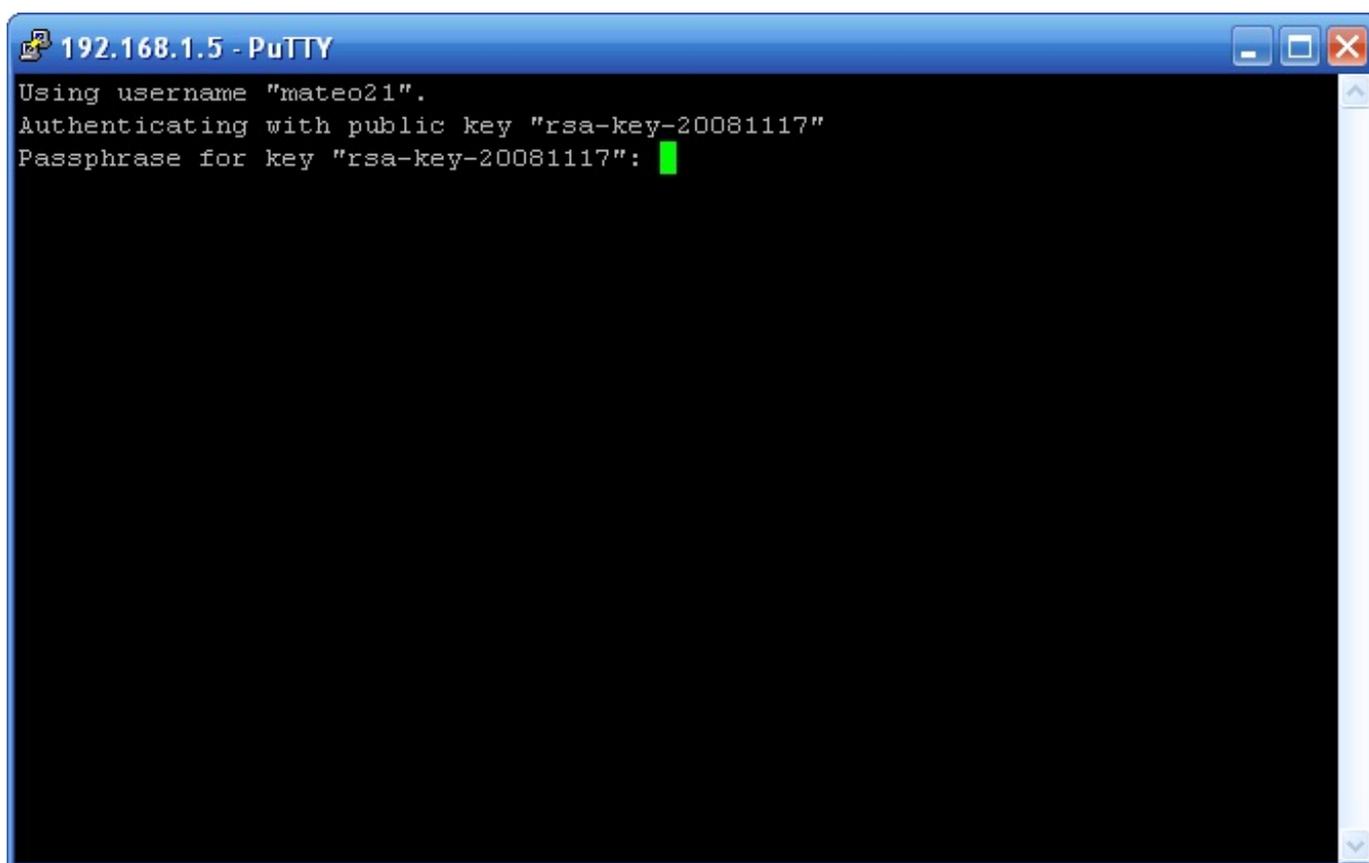
Retournez à l'accueil en cliquant sur la section "Session" tout en haut. Rentrez l'ip du serveur. Ensuite, je vous recommande fortement d'enregistrer ces paramètres.



Rentrez un nom à votre serveur (par exemple MonServeur) sous "Saved Sessions". Cliquez ensuite sur Save. A l'avenir, vous n'aurez qu'à double-cliquer sur le nom de votre serveur dans la liste pour vous y connecter directement avec les bons paramètres.

Cliquez sur "Open" pour vous connecter au serveur.

Vous devriez voir Putty utiliser automatiquement votre pseudo, puis vous demander votre passphrase. Rentrez-la pour vérifier que ça marche :



Euh, et si je veux pas avoir à rentrer la passphrase à chaque fois ? Non parce que c'est pareil que de rentrer un mot de passe là... 🤔

En effet, et ma réponse sera la même que pour ceux qui se connectent depuis Linux : il faut utiliser un agent SSH. Ce programme va rester en mémoire et retenir votre clé privée. Il ne vous demandera la passphrase qu'une fois au début, puis ensuite vous pourrez vous connecter autant de fois que vous voulez à autant de serveurs que vous voulez sans avoir à rentrer quoi que ce soit. 😊

L'agent SSH Pageant

L'agent SSH installé avec Putty s'appelle "Pageant". Je vous recommande de le lancer au démarrage de l'ordinateur automatiquement (il ne prend que 4 Mo en mémoire), en le plaçant dans le dossier "Démarrage" du menu Démarrer.

Lorsque vous lancez Pageant, une petite icône d'un ordinateur avec un chapeau s'ajoute dans la barre des tâches à côté de l'horloge :



Faites un clic droit dessus, puis cliquez sur "Add key". On vous demande où se trouve la clé privée (cle.ppk). Rentrez ensuite la passphrase.

C'est bon. Vous avez juste besoin de le faire une fois. Maintenant, vous pouvez vous connecter au serveur que vous voulez en cliquant droit sur l'icône puis en sélectionnant "Saved Sessions" :



On ne vous demandera plus votre clé. 😊



Notez que si l'agent SSH Pageant est pratique, il vaut mieux l'arrêter si vous devez vous absenter de votre ordinateur un long moment et que quelqu'un risque de l'utiliser. Sinon, n'importe qui peut se connecter à vos serveurs sans avoir à rentrer de mot de passe.

Retenez bien : l'agent SSH est un compromis entre la sécurité et le côté pratique. Il retient les clés pour vous (du moins tant que le programme tourne). Si vous êtes un utilisateur intensif de SSH, cela vous fera gagner beaucoup de temps.

Vous pouvez modifier le raccourci qui lance Pageant pour que celui-ci charge votre clé privée automatiquement dès son lancement. Faites un clic-droit sur l'icône de Pageant, allez dans "Propriétés".

Dans le champ "Cible", rajoutez à la fin en paramètre le chemin de la clé à charger. Par exemple : "`C:\Program Files\PuTTY\pageant.exe`" `c:\cle.ppk`. La clé sera alors chargée dès que vous lancerez Pageant.

Ouf !

Au terme de ce chapitre, vous devriez avoir engrangé une quantité importante de nouvelles connaissances à la fois théoriques et pratiques.

Vous savez désormais comment fonctionne l'échange de clés entre le serveur et le client. Ça paraît bête mais beaucoup d'utilisateurs de SSH ne savent pas vraiment comment ça marche. Par contre, ils savent que ça crypte les échanges, et ça c'est déjà l'essentiel.

La possibilité de se connecter à distance à une machine Linux est vraiment un de ses points forts. Vous pouvez retrouver ainsi une console dans laquelle vous pouvez faire tout ce que vous voulez, exactement comme si vous étiez en face de la machine.

Enfin, si Windows ne propose pas de fonctionnalité serveur dans ses éditions "classiques" (familiale et professionnelle), il faut savoir tout de même qu'il existe une version spéciale de Windows appelée "Windows Server" qui concurrence Linux sur le marché des serveurs.

Transférer des fichiers

Vous avez appris à vous connecter à un serveur à distance avec SSH. Désormais, vous pouvez exécuter des commandes sur un ordinateur en prenant le contrôle à distance, grâce au réseau.

On continue ici notre découverte du monde fabuleux des réseaux sous Linux. Un monde un peu particulier comme vous avez pu le découvrir : de gros efforts sont faits pour assurer la sécurité des données grâce au cryptage, ce qui permet d'éviter que l'on se fasse voler ses données personnelles comme son mot de passe.

Nous allons mettre l'accent sur le transfert de fichiers ici : comment télécharger un fichier ? Comment se connecter à un FTP, lire et télécharger des fichiers ? Et surtout, comment copier des fichiers de manière sécurisée ?

wget : téléchargement de fichiers

Nous commençons par une commande simple à utiliser, du moins en apparence : `wget`. Elle permet de télécharger des fichiers directement depuis la console.

Il suffit d'indiquer une adresse HTTP ou FTP d'un fichier à télécharger :

Code : Console

```
$ wget http://cdimage.debian.org/debian-cd/4.0_r5/i386/iso-cd/debian-40r5-i386-businesscard.iso
```

Une barre de progression du téléchargement devrait s'afficher alors dans la console :

Code : Console

```
$ wget http://cdimage.debian.org/debian-cd/4.0_r5/i386/iso-cd/debian-40r5-i386-businesscard.iso
--2008-12-05 12:43:25-- http://cdimage.debian.org/debian-cd/4.0_r5/i386/iso-cd/debian-40r5-i386-businesscard.iso
Résolution de cdimage.debian.org... 130.239.18.173, 130.239.18.137
Connexion vers cdimage.debian.org|130.239.18.173|:80... connecté.
requête HTTP transmise, en attente de la réponse... 302 Found
Emplacement: http://saimei.acc.umu.se/debian-cd/4.0_r5/i386/iso-cd/debian-40r5-i386-businesscard.iso [suivant]
--2008-12-05 12:43:25-- http://saimei.acc.umu.se/debian-cd/4.0_r5/i386/iso-cd/debian-40r5-i386-businesscard.iso
Résolution de saimei.acc.umu.se... 130.239.18.138
Connexion vers saimei.acc.umu.se|130.239.18.138|:80... connecté.
requête HTTP transmise, en attente de la réponse... 200 OK
Longueur: 34181120 (33M) [application/octet-stream]
Saving to: `debian-40r5-i386-businesscard.iso'

38% [=====] 13 208 331 117K/s eta 70s
```

Les informations au début sont assez nombreuses et ne nous intéressent pas vraiment. Elles indiquent simplement comment le programme a communiqué avec le serveur qui possédait le fichier.

En bas, vous avez dans l'ordre sur la dernière ligne :

1. Une barre de progression qui se met à jour
2. Le nombre d'octets téléchargés
3. La vitesse de téléchargement
4. Le temps restant estimé (*eta*)

Vous pouvez arrêter le téléchargement à tout moment en utilisant la combinaison Ctrl + C que vous avez déjà découverte.



Comment je récupère l'adresse du fichier à télécharger pour le donner à wget ?

Le plus simple est d'ouvrir un navigateur web tel que Firefox là où vous avez accès à un environnement graphique, et de faire un clic droit sur le lien du fichier que vous voulez télécharger, puis "Copier l'adresse du lien". Vous pouvez ensuite le coller dans la console.

Notez qu'il existe aussi des navigateurs en console tel que `lynx` (plutôt basique) et `links` (assez complet), que vous pouvez télécharger et essayer si vous voulez.

Reprendre un téléchargement arrêté

Si vous voulez reprendre un téléchargement arrêté, utilisez l'option `-c` :

Code : Console

```
$ wget -c http://cdimage.debian.org/debian-cd/4.0_r5/i386/iso-cd/debian-40r5-i386-businesscard.iso
```

Pour que ça fonctionne, il ne faut bien entendu pas supprimer le bout de fichier téléchargé sur votre disque. 😊

Si la reprise a fonctionné, vous devriez voir une barre de progression comme celle-ci :

Code : Console

```
71% [+++++++=====] 24 450 216 470K/s eta 88s
```

Les `+++` correspondent à la partie précédemment téléchargée. Cela vous confirme que la reprise a bien marché.

Lancer un téléchargement en tâche de fond

Enfin, si vous voulez que le téléchargement soit envoyé en tâche de fond dès le début, il y a la technique du `nohup` que l'on connaît qui s'applique à toutes les commandes, mais vous pouvez aussi utiliser l'option `--background` :

Code : Console

```
$ wget --background -c http://cdimage.debian.org/debian-cd/4.0_r5/i386/iso-cd/debian-40r5-i386-businesscard.iso
Poursuite à l'arrière plan, pid 8422.
La sortie sera écrite vers « wget-log ».
```

L'avancement du téléchargement sera écrit dans un fichier "wget-log" comme indiqué.



wget propose une quantité impressionnante d'options. Je ne peux pas toutes les couvrir. Sachez que vous pouvez l'utiliser pour télécharger des pages web notamment. Appelez le manuel pour en savoir plus, il y a de quoi faire :

`man wget`

Un des avantages de wget est que vous avez toujours une barre de progression, même si vous téléchargez un fichier depuis un serveur FTP. Ça paraît bête, mais le programme "ftp" qu'on verra plus loin ne donne pas l'avancement du téléchargement, contrairement à wget !

scp : copier des fichiers sur le réseau

Vous connaissez la commande `cp` ? Elle permet de copier des fichiers sur votre disque dur.

Eh bien voici `scp` (Secure CoPy), qui permet de copier des fichiers d'un ordinateur à un autre à travers le réseau ! Le tout de manière sécurisée bien sûr. 😊



Il existe aussi `rcp` (Remote CoPy) qui fait la même chose mais sans aucun cryptage. Son utilisation est déconseillée.

`scp` s'utilise quasiment comme `ssh`. D'ailleurs ce n'est pas un hasard, car `scp` se base sur `ssh` pour fonctionner. Là où `ssh` sert à ouvrir une console à distance (un *shell*), `scp` est spécialement faite pour copier des fichiers d'un ordinateur à un autre.

On l'utilise comme ceci :

Code : Console

```
scp fichier_origine copie_destination
```

Le premier élément à indiquer est la position du fichier que l'on veut copier. Le second élément correspond au répertoire de destination où il doit être copié.

Chacun de ces éléments peut s'écrire sous la forme suivante : `login@ip:nom_fichier`. Le login et l'IP sont facultatifs. Si vous n'écrivez pas de login ni d'IP, `scp` considèrera que le fichier se trouve sur votre ordinateur.



Vous pouvez remplacer l'IP par un nom d'hôte (un nom de domaine) si vous en avez un, qui est plus facile à retenir. Par exemple notre serveur *lisa* peut être appelé en écrivant `lisa.siteduzero.com` au lieu d'une IP compliquée type `85.123.10.201`.

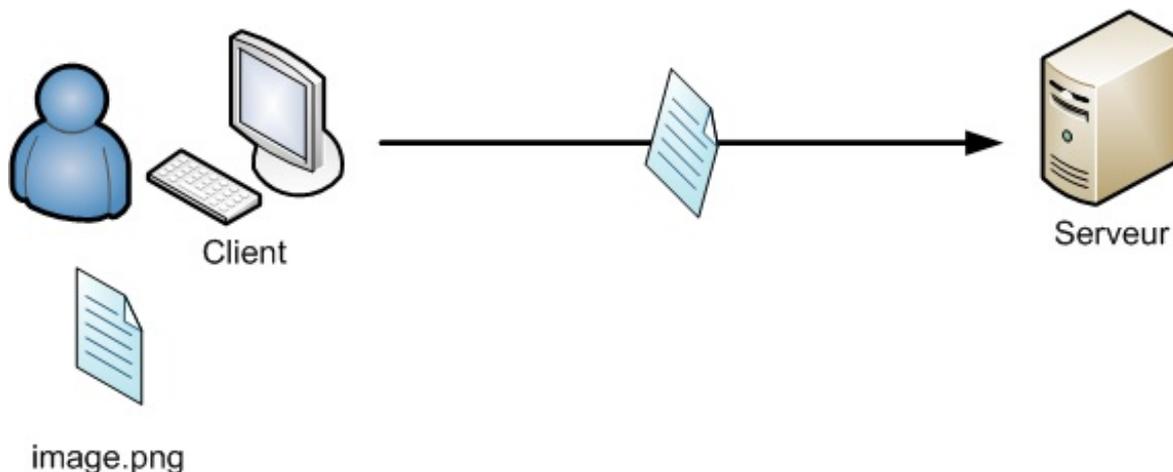
Copier un fichier de votre ordinateur vers un autre

Prenons un cas concret, vous allez mieux voir comment ça s'utilise :

Code : Console

```
scp image.png mateo21@85.123.10.201:/home/mateo21/images/
```

Ici, je demande à copier le fichier "image.png" qui se trouve sur mon ordinateur vers un autre ordinateur dont l'ip est 85.123.10.201. Le fichier sera placé dans le dossier `/home/mateo21/images/` sur cet autre ordinateur.



Notez qu'on peut utiliser le symbole ~ pour indiquer "mon répertoire personnel" (/home/mateo21/). D'autre part, si cet autre ordinateur a un nom d'hôte facile à retenir, j'aurai tendance à l'utiliser à la place de l'IP. J'aurais donc pu écrire quelque chose comme ça, qui aurait été identique :

Code : Console

```
scp image.png mateo21@lisa.siteduzero.com:~/images/
```

Lorsque vous lancez la commande, scp essaiera de se connecter au serveur ayant l'ip indiquée avec le login que vous avez demandé (mateo21 dans mon cas). On vous demandera alors votre mot de passe ou, mieux, scp utilisera votre clé privée si elle existe (on a vu que combiné à l'agent ssh, cela nous évitait d'avoir à retaper notre passphrase à chaque fois !).

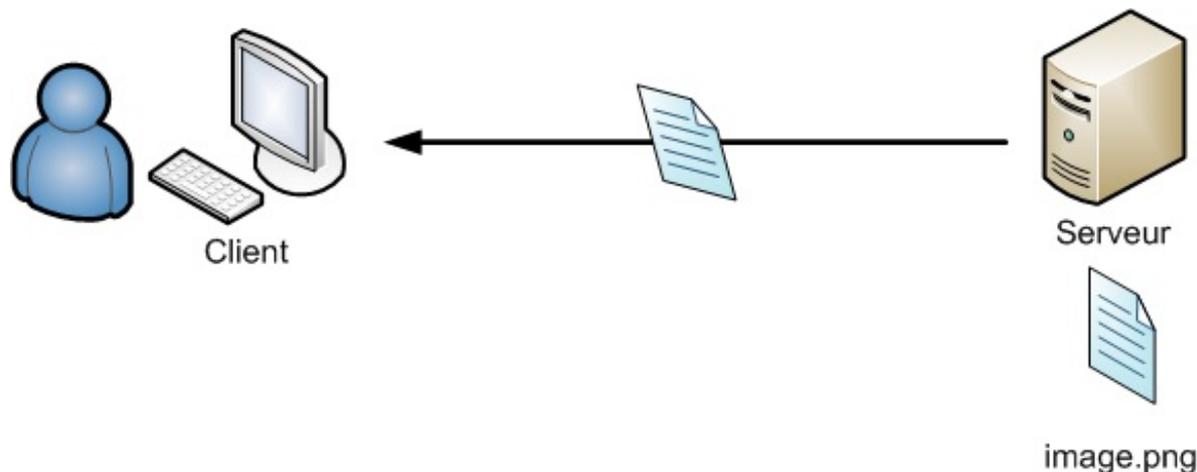
Copier un fichier d'un autre ordinateur vers le vôtre

On peut faire aussi l'inverse, c'est-à-dire récupérer un fichier qui se trouve sur un autre ordinateur et le placer chez nous :

Code : Console

```
scp mateo21@85.123.10.201:image.png copie_image_sur_mon_pc.png
```

Dans cet exemple, je copie le fichier "image.png" qui se trouve sur le serveur dont l'ip est 85.123.10.201. Je place cette copie sur mon propre ordinateur sous le nom "copie_image_sur_mon_pc.png".



Si je veux, je peux aussi copier le fichier sans changer le nom :

Code : Console

```
scp mateo21@85.123.10.201:image.png .
```

Notez le "point" à la fin. Ce point signifie "copier dans le répertoire dans lequel je me trouve". Le fichier image.png sera donc placé sur mon ordinateur dans le dossier actuel.

Le piège du port

Si le serveur SSH auquel vous essayez de vous connecter n'est pas sur le port standard (22), il faudra indiquer le numéro du port avec l'option **-P** :

Code : Console

```
scp -P 16296 mateo21@85.123.10.201:image.png .
```



La commande ssh a aussi une option pour indiquer un port, vous vous en souvenez peut-être, c'est... **-p** ! Faites attention donc, car avec ssh c'est un "p" minuscule qu'il faut utiliser, et avec scp c'est un P majuscule pour faire la même chose !

Je peux vous dire que je me suis trompé un bon nombre de fois. 🤔

ftp & sftp : transférer des fichiers

Le FTP (File Transfer Protocol) est un protocole permettant d'échanger des fichiers sur le réseau. Il est assez ancien (1985) et est aujourd'hui toujours utilisé pour transférer des fichiers.

On l'utilise en général dans 2 cas :

- Pour télécharger un fichier depuis un serveur FTP public. En général, les navigateurs web font ça de manière automatique et transparente lorsque vous cliquez sur un lien de téléchargement. La connexion se fait alors en **mode anonyme**.
- Pour transférer des fichiers vers un serveur FTP privé (et éventuellement en télécharger aussi). Lorsqu'on prend un hébergement pour son site web, l'hébergeur nous donne en général des accès FTP pour aller y déposer les fichiers de son site web. La connexion se fait donc en **mode authentifié**.

Tout le monde n'a pas forcément accès à un serveur FTP privé, aussi je vous propose pour les exemples suivants de nous connecter à un serveur FTP public (rassurez-vous, si vous voulez vous connecter à un FTP privé la méthode est la même).



Nous nous intéressons ici au fonctionnement du FTP en ligne de commande. Bien sûr, il existe des logiciels graphiques qui font la même chose, comme par exemple FileZilla.

Connexion à un serveur FTP

Essayons de nous connecter au serveur FTP de debian, qui est accessible à l'adresse ftp.debian.org :

Code : Console

```
$ ftp ftp.debian.org
```

Le serveur FTP devrait répondre en vous demandant un login et un mot de passe. Pour les serveurs FTP publics, le login à utiliser est toujours "anonymous" (anonyme).

Code : Console

```
$ ftp ftp.debian.org
Connected to ftp.debian.org.
220 saens.debian.org FTP server (vsftpd)
Name (ftp.debian.org:mateo21): anonymous
331 Please specify the password.
Password:
```

Pour le mot de passe, vous mettez ce que vous voulez, peu importe, vous serez accepté. 😊

Vous devriez alors voir un message de bienvenue s'afficher, se terminant par :

Code : Console

```
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp>
```

Vous avez maintenant un *prompt* ftp> qui vous permet de rentrer des commandes FTP.

Se déplacer au sein du serveur FTP

Vous savez quoi ? Bonne nouvelle : les commandes que vous pouvez utiliser sont pour la plupart les mêmes que celles que vous connaissez. 😊

Citons, dans le lot :

- **ls** : affiche le contenu du répertoire actuel.
- **pwd** : affiche le chemin actuel.
- **cd** : change de répertoire.

Avec ces commandes vous devriez déjà pouvoir vous balader sur le serveur FTP.
Faites un ls pour voir :

Code : Console

```
ftp> ls  
200 PORT command successful. Consider using PASV.  
150 Here comes the directory listing.  
drwxrwsr-x    7 1176    1176          4096 Dec 05 09:10 debian  
226 Directory send OK.
```

Les lignes commençant par un numéro sont des messages envoyés par le serveur FTP. Vous noterez que les fichiers s'affichent comme si on avait écrit ls -l.

Il y a seulement un répertoire, rendez-vous donc dans "debian" :

Code : Console

```
ftp> cd debian  
250 Directory successfully changed.
```

Et affichez à nouveau le contenu :

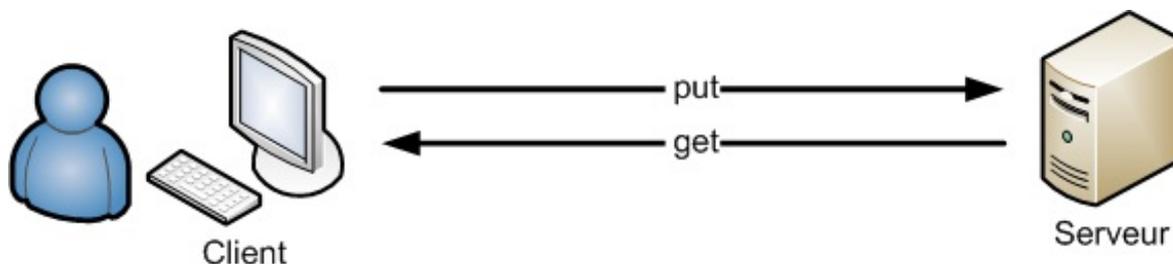
Code : Console

```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-rw-r-- 1 1176 1176 940 Oct 27 20:29 README
-rw-rw-r-- 1 1176 1176 1290 Dec 04 2000 README.CD-
manufacture
-rw-rw-r-- 1 1176 1176 2426 Oct 27 20:29 README.html
-rw-r--r-- 1 1176 1176 124286 Dec 03 19:52 README.mirrors.html
-rw-r--r-- 1 1176 1176 62059 Dec 03 19:52 README.mirrors.txt
drwxr-sr-x 9 1176 1176 4096 Nov 16 18:56 dists
drwxr-sr-x 3 1176 1176 4096 Nov 11 22:16 doc
drwxr-sr-x 3 1176 1176 4096 Dec 05 09:08 indices
-rw-rw-r-- 1 1176 1176 4557196 Dec 05 08:49 ls-lR.gz
-rw-r--r-- 1 1176 1176 154934 Dec 05 08:49 ls-lR.patch.gz
drwxr-sr-x 5 1176 1176 4096 Nov 11 22:16 pool
drwxr-sr-x 4 1176 1176 4096 Nov 18 09:04 project
226 Directory send OK.
```

Le transfert de fichiers

Si vous souhaitez récupérer un fichier ou en envoyer un, il y a 2 commandes à connaître :

- **put** : envoie un fichier vers le serveur.
- **get** : télécharge un fichier depuis le serveur.



Notez qu'il est impossible d'utiliser **put** sur les serveurs FTP publics comme celui auquel nous sommes connectés. Seul le téléchargement de fichiers est autorisé. D'autres commandes, comme celle qui permet de changer les CHMOD des fichiers, ne sont pas non plus activées.

Vous pouvez récupérer le fichier README par exemple en écrivant `get README` :

Code : Console

```
ftp> get README
local: README remote: README
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for README (940 bytes).
226 File send OK.
940 bytes received in 0.00 secs (918.9 kB/s)
```

Et voilà ! Le fichier se trouve maintenant sur votre ordinateur. 😊

Il a été téléchargé sur le dossier dans lequel vous vous trouviez sur votre ordinateur. Pour savoir dans quel dossier vous êtes *chez vous*, tapez `!pwd` :

Code : Console

```
ftp> !pwd
/home/mateo21
```

Si vous voulez changer de dossier *chez vous*, utilisez `!cd`. Pour lister les fichiers *chez vous*, utilisez `!ls`. Bref, vous m'avez compris, il suffit de faire précéder les commandes d'un point d'exclamation pour qu'elles s'exécutent chez nous. 😊

Les autres commandes

Il existe de nombreuses autres commandes FTP, nous n'allons pas toutes les voir.

Tapez `man ftp` pour avoir un aperçu des commandes disponibles. Vous noterez que toutes ne sont pas identiques à celles que vous connaissez. Par exemple, pour supprimer un fichier, ce n'est pas `rm` mais... `delete` ! D'autres vous seront familières : `mkdir` permet de créer un dossier par exemple.

Pour quitter le serveur et vous déconnecter, vous avez le choix entre la bonne vieille combinaison de touches `Ctrl + D` qui commande la fermeture de la session, ou encore les commandes `bye`, `exit` et `quit`, qui sont synonymes.

sftp : un FTP sécurisé

Le protocole FTP a un défaut : il n'est pas sécurisé. Les données ne sont pas cryptées. Quelqu'un ayant accès au réseau pourrait alors intercepter le contenu des fichiers que vous échangez ou encore votre mot de passe lors de la connexion.

Pour remédier à cela, on a inventé `sftp`, qui se base sur SSH pour sécuriser la connexion :

Code : Console

```
sftp login@ip
```

Par exemple :

Code : Console

```
sftp mateo21@lisa.siteduzero.com
```

On vous demandera alors votre mot de passe (ou bien la clé publique sera utilisée si elle est présente).

Une fois connecté, les commandes sont presque les mêmes que pour le FTP. Vous retrouverez notamment `get` et `put` pour échanger des fichiers. Sachez que les commandes sont globalement plus puissantes et pratiques en SFTP qu'en FTP. Méfiez-vous toutefois, certaines commandes changent ! Par exemple, pour supprimer un fichier, ce n'est plus `delete` mais à nouveau `rm` ! Lisez le manuel pour plus d'infos : `man sftp`.



Pour se connecter en SFTP, on utilise le même port que SSH (soit 22 par défaut). Si votre serveur SSH fonctionne sur un autre port, vous devrez le préciser comme ceci : `sftp -oPort=27401 mateo21@serveur`

A l'heure actuelle, le SFTP reste assez peu utilisé. Les hébergeurs web utilisent toujours le FTP classique, alors que la plupart

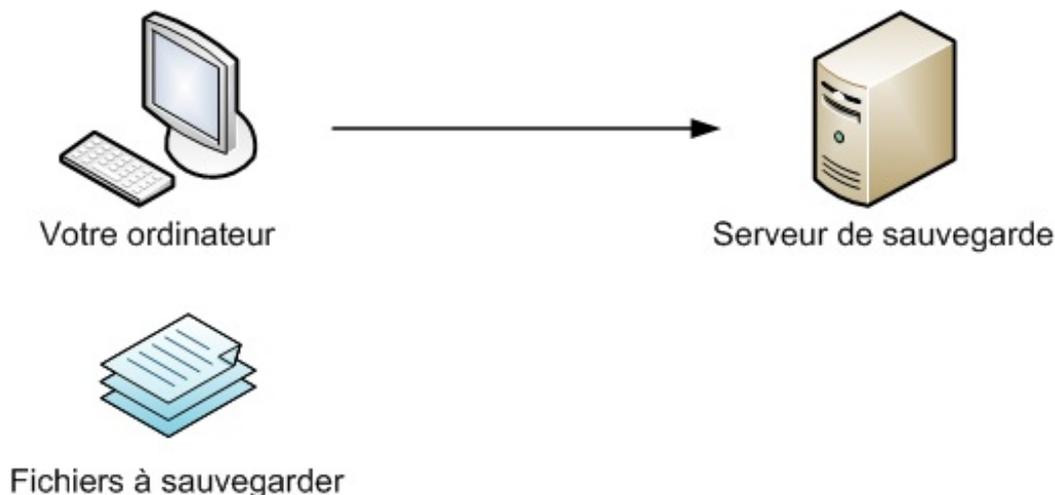
des logiciels graphiques comme FileZilla sont pourtant capables de se connecter en SFTP.

rsync : synchroniser des fichiers pour une sauvegarde

`rsync` est un programme assez simple à utiliser et pourtant très puissant. Il permet d'effectuer une synchronisation entre 2 répertoires, que ce soit sur le même PC ou entre 2 PC reliés en réseau.

Rsync est le plus souvent utilisé pour effectuer des **sauvegardes incrémentielles**. Je m'explique.

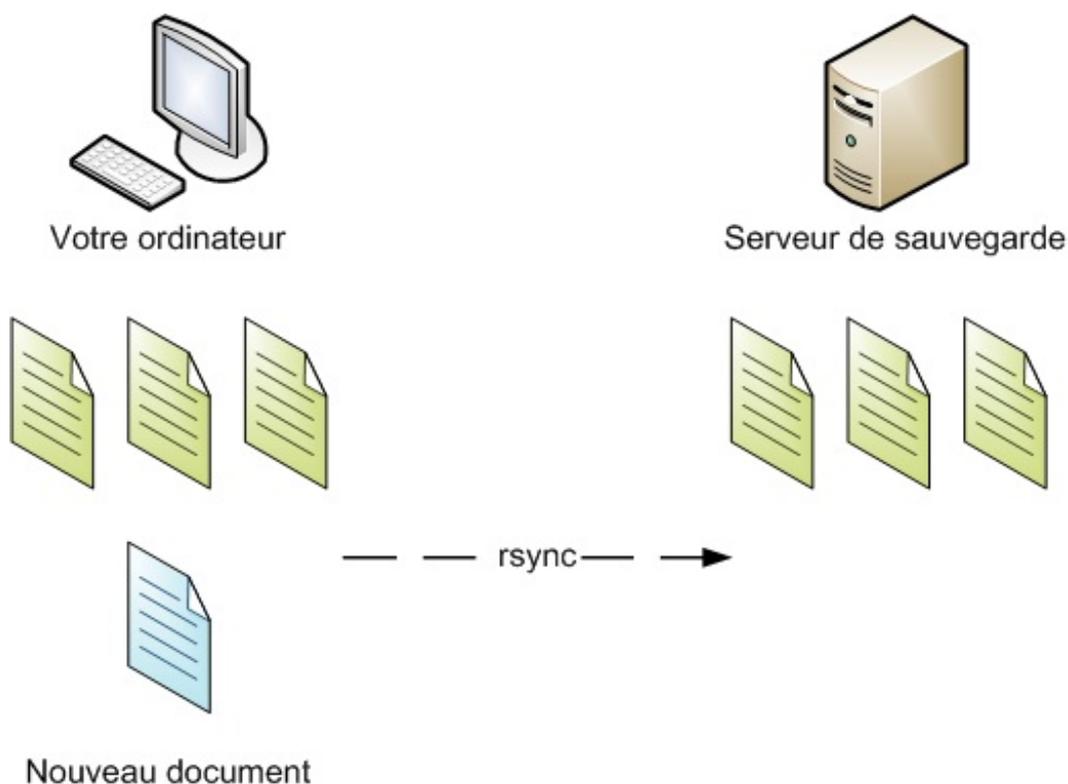
Par exemple, imaginez que vous souhaitez sauvegarder régulièrement les fichiers de votre home (répertoire personnel). Ainsi, si un jour un accident survient (perte du PC par exemple) vous aurez toujours vos documents sauvegardés au chaud sur un serveur quelque part.



Ok, vous copiez tout votre home sur ce serveur.

Oui mais voilà, votre home contient peut-être 10 Go de fichiers : entre vos vidéos, la musique, vos rapports au format texte et vos photos de vacances, ça va vite.

Sauvegarder ces 10 Go une fois, d'accord. Mais la seconde fois, plutôt que de tout renvoyer, vous souhaitez peut-être envoyer uniquement les fichiers qui ont été ajoutés ou modifiés, non ?



C'est là que rsync intervient. C'est une sorte de scp intelligent : il compare et analyse les différences entre 2 dossiers puis copie

uniquement les changements. C'est ce que veut dire le mot "incrémentiel".

Rsync peut être utilisé pour effectuer une sauvegarde entre 2 dossiers sur le même ordinateur, ou bien entre 2 dossiers sur 2 ordinateurs différents. En général, on l'utilise plutôt pour sauvegarder entre 2 ordinateurs différents bien sûr.

Nous allons dans un premier temps pour simplifier voir la sauvegarde entre 2 dossiers de votre ordinateur, puis nous effectuerons ensuite la sauvegarde sur un autre ordinateur.

Sauvegarder dans un autre dossier du même ordinateur

Dans les exemples qui vont suivre, je vais supposer que vous souhaitez sauvegarder le dossier "Images" dans un dossier "backups".

Dans le dossier Images, il y a quelques photos de vacances :

Code : Console

```
$ ls
espagne1.jpg  italie1.jpg  italie2.jpg  italie3.jpg
```

Vous pouvez créer comme moi des fichiers bidon à l'aide de la commande `touch`.

Maintenant, lancez un rsync comme ceci :

Code : Console

```
$ rsync -arv Images/ backups/
sending incremental file list
created directory backups
./
espagne1.jpg
italie1.jpg
italie2.jpg
italie3.jpg

sent 268 bytes  received 91 bytes  718.00 bytes/sec
total size is 0  speedup is 0.00
```

Quelques explications sur les paramètres :

- `-a` : conserve toutes les informations sur les fichiers, comme les droits (chmod), la date de modification, etc.
- `-r` : sauvegarde aussi tous les sous-dossiers qui se trouvent dans le dossier à sauvegarder.
- `-v` : mode verbeux, affiche des informations détaillées sur la copie en cours.

Viennent ensuite le nom du dossier à sauvegarder, et le répertoire de sauvegarde.

Rsync analyse le contenu du répertoire de sauvegarde dans un premier temps. Comme celui-ci est vide, il y a copie tous les fichiers comme vous pouvez le voir.

Maintenant, relancez la même commande une seconde fois :

Code : Console

```
$ rsync -arv Images/ backups/
sending incremental file list
```

```
sent 109 bytes  received 12 bytes  242.00 bytes/sec
total size is 0  speedup is 0.00
```

Comme vous pouvez le voir, aucun fichier n'a été envoyé cette fois ! En effet, rsync étant intelligent, il a détecté qu'il n'y avait aucun changement et donc qu'il n'y avait pas lieu de copier quoi que ce soit.

Testons un peu ce qui se passe si on ajoute un fichier :

Code : Console

```
$ touch Images/espagne2.jpg
$ rsync -arv Images/ backups/
sending incremental file list
./
espagne2.jpg

sent 172 bytes  received 34 bytes  412.00 bytes/sec
total size is 0  speedup is 0.00
```

Le nouveau fichier espagne2.jpg a bien été copié ! 😊

Vous pouvez essayer aussi de modifier un fichier, vous verrez que rsync copie bien les fichiers qui ont été modifiés.

Supprimer les fichiers



J'ai essayé de supprimer un fichier, mais celui-ci n'a pas été supprimé dans le répertoire de sauvegarde. Comment faire ?

Par défaut, rsync ne supprime pas les fichiers dans le répertoire de copie. Si vous voulez lui demander de les supprimer, pour que le contenu soit strictement identique, rajoutez `--delete`.

Par exemple, si je supprime le fichier `italie3.jpg` :

Code : Console

```
$ rm Images/italie3.jpg
$ rsync -arv --delete Images/ backups/
sending incremental file list
deleting italie3.jpg

sent 120 bytes  received 12 bytes  264.00 bytes/sec
total size is 4  speedup is 0.03
```

... rsync me supprime mon fichier `italie3.jpg` !

Sauvegarder les fichiers supprimés

Peut-être que la suppression du fichier était accidentelle. Si même votre rsync supprime le fichier dans le répertoire de sauvegarde, vous n'en aurez plus aucune trace !

Heureusement, il est possible de garder de côté les fichiers que l'on a supprimés, sait-on jamais, au cas où... 🤔

Pour cela, rajoutez l'option `--backup`. Les fichiers supprimés prendront un suffixe dans le répertoire de sauvegarde.

Vous pouvez aussi, pour éviter que ça ne fasse désordre, déplacer les fichiers supprimés dans un dossier spécial des fichiers supprimés. Rajoutez `--backup-dir=/chemin/vers/le/repertoire`.

Par exemple :

Code : Console

```
$ rsync -arv --delete --backup --backup-  
dir=/home/mateo21/backups_supprimes Images/ backups/
```



Je vous recommande d'indiquer le répertoire `backup-dir` en absolu comme je l'ai fait. Sinon, le répertoire des fichiers supprimés sera placé à l'intérieur du répertoire de sauvegarde et vous risquez d'avoir plus de problèmes lors de la synchronisation qu'autre chose.

Rsync peut faire bien d'autres choses, comme exclure un dossier de la sauvegarde (option `--exclude`). Je vous laisse lire le manuel pour savoir un peu tout ce que vous pouvez faire. 🤔

Sauvegarder sur un autre ordinateur

Intéressons-nous maintenant à la sauvegarde sur un autre ordinateur, parce que là c'est bien joli mais on se sentirait plus en sécurité si les fichiers étaient envoyés ailleurs sur un autre ordinateur.

L'avantage de rsync est qu'il peut copier les fichiers à l'aide de plusieurs méthodes différentes. La plus couramment utilisée, que nous allons choisir ici, est de passer par SSH. Comme quoi vous le voyez, SSH ça sert à tout pour sécuriser les transferts. 🤖

Code : Console

```
$ rsync -arv --delete --backup --backup-  
dir=/home/mateo21/fichiers_supprimes/ Images/ mateo21@IP_du_serveur:mes_backups/
```

Si votre serveur SSH s'utilise sur un autre port que le port par défaut, il faudra rajouter `-e "ssh -p port"` :

Code : Console

```
$ rsync -arv --delete --backup --backup-  
dir=/home/mateo21/fichiers_supprimes/ Images/ mateo21@IP_du_serveur:mes_backups/ -  
e "ssh -p 12473"
```

L'échange de fichiers est, comme vous avez pu le voir, rendu possible par une multitude de commandes. A vous d'utiliser celle qui convient en fonction de votre cas :

- **wget** : si vous avez juste besoin de télécharger un fichier dont vous connaissez l'adresse (que vous avez récupérée sur un site web par exemple).
- **scp** : pour copier des fichiers entre 2 ordinateurs auprès desquels vous avez un accès SSH.
- **ftp** : pour échanger des fichiers avec les serveurs qui ne proposent que ce mode de transfert.
- **sftp** : pour échanger des fichiers avec un serveur de façon sécurisée (si disponible).
- **rsync** : pour synchroniser 2 répertoires, généralement pour effectuer des sauvegardes incrémentielles rapides.

Analyser le réseau et filtrer le trafic avec un firewall

Ce chapitre vous propose d'apprendre à maîtriser le trafic réseau qui passe par votre ordinateur. En effet, lorsque vous êtes connecté à internet, vous avez régulièrement des applications qui vont se connecter, puis télécharger et envoyer des informations. Comment surveiller ce qui se passe, quelle application est en train de communiquer, sur quel port ?

Je vous propose de découvrir d'abord quelques outils de base qui vous permettent de bien comprendre comment une IP est associée à un nom d'hôte. Puis, nous analyserons le trafic en cours avec un outil comme netstat. Enfin, et ce ne sera pas le plus facile je vous préviens, nous apprivoiserons le célèbre firewall utilisé sous Linux : iptables. Il est assez complexe à paramétrer, mais heureusement des programmes supplémentaires peuvent nous simplifier le travail. 😊

Savoir paramétrer un firewall est essentiel, que ce soit sur votre PC à la maison ou (à plus forte raison) sur un serveur. Cela vous protège de manière solide contre les programmes qui voudraient échanger des informations sur le réseau sans votre accord. C'est une mesure de sécurité essentielle qu'il faut connaître, et dont aucun administrateur de serveur sérieux ne pourrait se passer. 😊

host & whois : qui êtes-vous ?

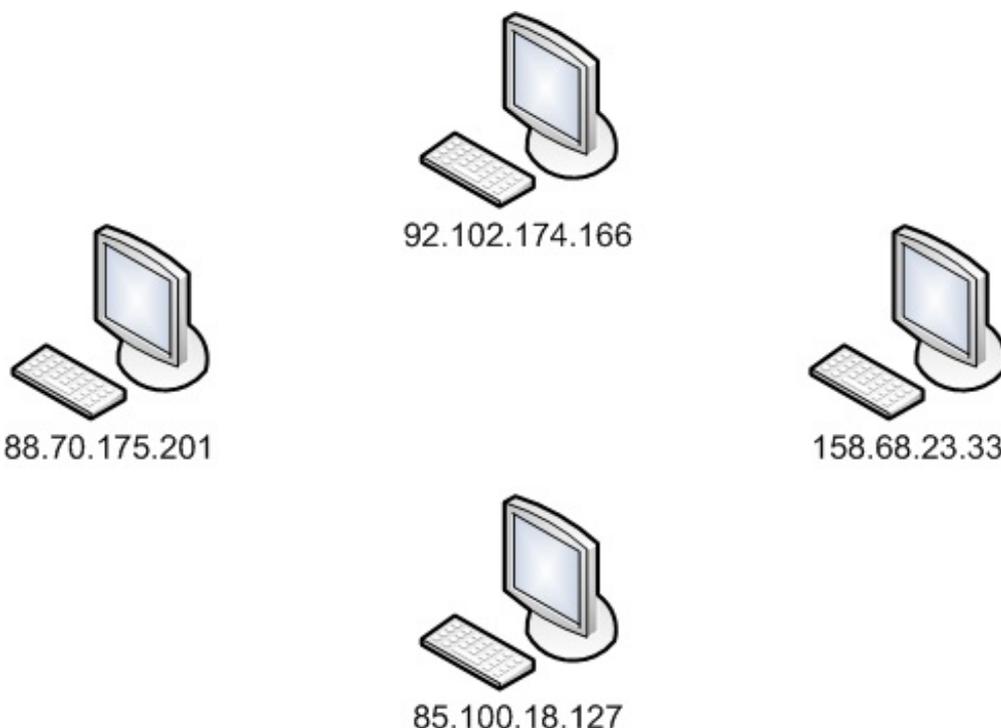
Chaque ordinateur relié à internet est identifié par une adresse IP, comme vous le savez sûrement.

Une adresse IP est un groupement de 4 nombres, par exemple :

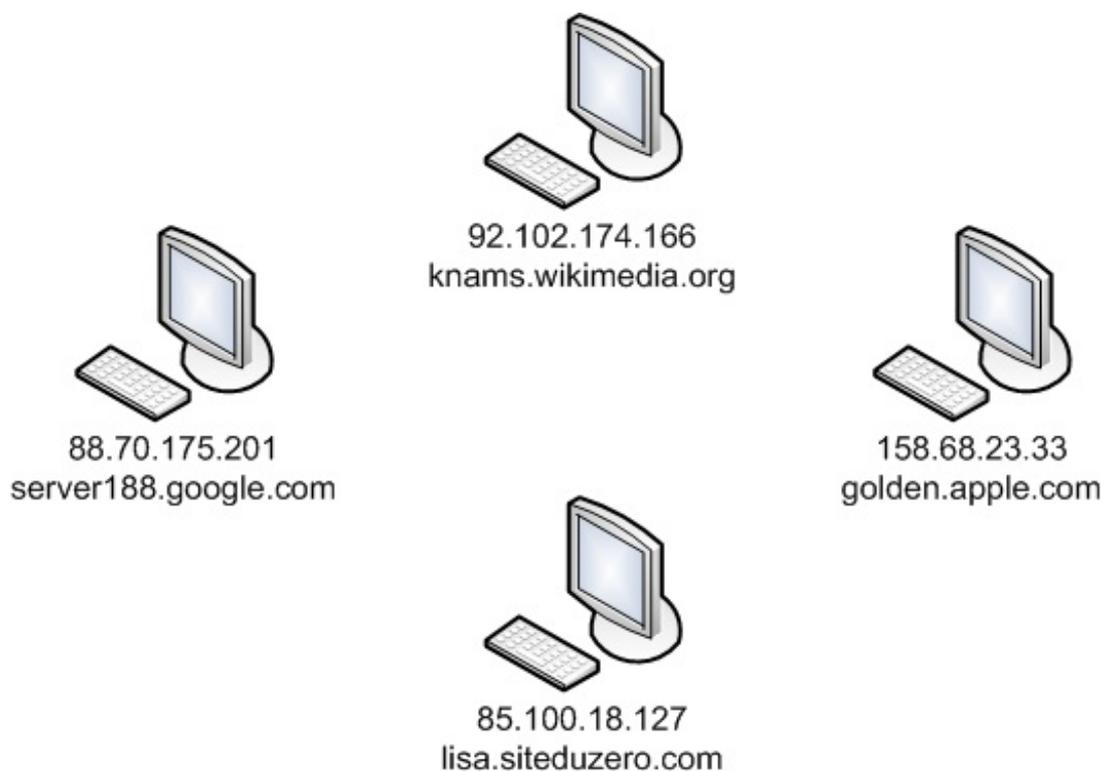
86 . 172 . 120 . 28



Cette adresse est au format IPv4. C'est encore à l'heure actuelle le type d'IP le plus utilisé, mais ces adresses sont appelées petit à petit à être remplacées par la norme IPv6. Bientôt, tout le monde aura donc une IP qui ressemblera plutôt à quelque chose comme ceci : **fe80 : : 209 : 62fa : fb80 : 29f2**



A chaque IP, on peut associer ce qu'on appelle un nom d'hôte (*hostname*). C'est un nom en toutes lettres plus facile à mémoriser, qui revient exactement au même que d'écrire l'adresse IP :



Chaque serveur peut ainsi avoir un nom d'hôte plus facile à retenir. Je retiens mieux le nom de notre serveur (lisa.siteduzero.com) que son équivalent en adresse IP. 😊

Convertir une IP en nom d'hôte et inversement

Il existe une commande qui est capable d'effectuer la conversion dans les 2 sens :

- A partir d'une IP on peut avoir le nom d'hôte correspondant
- A partir d'un nom d'hôte, on peut avoir l'IP correspondante.

Cette commande, c'est `host`. Donnez-lui en paramètre une IP ou un nom d'hôte.

Par exemple :

Code : Console

```
$ host siteduzero.com
siteduzero.com has address 80.248.219.123
siteduzero.com mail is handled by 0 mail.siteduzero.com
```

La commande nous répond que l'IP de siteduzero.com est 80.248.219.123. Elle nous indique par ailleurs le nom du serveur qui gère les e-mails.

Maintenant essayons à l'envers avec l'IP :

Code : Console

```
$ host 80.248.219.123
123.219.248.80.in-addr.arpa domain name pointer lisa.siteduzero.com.
```

On nous répond que le nom d'hôte de 80.248.219.123 est lisa.siteduzero.com.



Mais ? Je croyais que c'était "siteduzero.com" cette IP ?

Oui, en fait c'est un synonyme dans le cas présent : siteduzero.com = lisa.siteduzero.com.

Vous pouvez essayer la même manipulation avec d'autres IP et noms d'hôte, prenez des sites que vous connaissez, comme par exemple "mozilla.org", "google.fr", etc.

Gérer les noms d'hôte personnalisés

Les associations entre les IP et les noms d'hôtes sont faites sur ce qu'on appelle des serveurs DNS. Nous n'allons pas rentrer dans le détail, mais sachez en gros que chaque fournisseur d'accès met en place des serveurs DNS qui fournissent la liste des correspondances IP <-> noms d'hôte.

Ainsi, lorsque vous tapez "siteduzero.com" dans votre navigateur, vous pouvez obtenir l'adresse IP correspondante et ainsi naviguer sur le Site du Zéro. 😊

C'est quand même plus pratique que d'avoir à retenir l'IP !

Vous ne pouvez pas modifier la liste des correspondances IP <-> noms d'hôte sur le serveur DNS (puisque ce serveur est utilisé par de nombreuses personnes), mais par contre vous pouvez établir une liste de correspondances personnalisée sur votre ordinateur.

Ouvrez pour cela en root le fichier /etc/hosts :

Code : Console

```
$ sudo nano /etc/hosts
```

Dedans, vous devriez avoir des lignes ressemblant à ceci :

Code : Console

```
127.0.0.1      localhost
127.0.1.1      mateo21-laptop
```

A gauche l'IP, à droite le nom d'hôte correspondant. Ecrire "localhost" est donc équivalent à écrire "127.0.0.1".

Vous pouvez ajouter des lignes sur le même modèle pour faire correspondre une IP à un nom d'hôte.

Quel intérêt ? Ca dépend. Parfois, les DNS ne fonctionnent pas bien pendant de courtes périodes (c'est très rare, mais ça peut arriver). Dans ce cas, il est plus simple de modifier votre fichier hosts pour pouvoir continuer à aller sur votre site préféré en "forçant" l'association du nom d'hôte et de l'IP.

Vous pourriez donc ajouter :

Code : Console

```
80.248.219.123  siteduzero.com
```

Enregistrez, et ouvrez un navigateur puis tapez "siteduzero.com" pour voir si ça fonctionne.



Attention, cette technique a l'avantage de forcer l'association mais si notre serveur change d'IP un jour, votre ordinateur ne sera pas au courant ! En règle générale, il est préférable d'utiliser les serveurs DNS qui se mettent régulièrement à jour (une fois par jour en moyenne) afin d'avoir toujours une liste à jour.

Sur un réseau local à la maison, il peut être pratique d'associer un nom d'hôte à chaque PC pour pouvoir vous y connecter sans avoir à retenir l'IP :

Code : Console

```
192.168.0.5      pc-papa
```

Ainsi, écrire "pc-papa" vous permet d'accéder à cet ordinateur sans avoir à retenir l'adresse IP correspondante.

whois : tout savoir sur un nom de domaine

Chaque nom de domaine doit obligatoirement indiquer qui se trouve derrière : nom, prénom, adresse et moyens de contact. C'est une règle.

L'outil whois vous permet d'obtenir facilement ces informations pour n'importe quel nom de domaine :

Code : Console

```
$ whois siteduzero.com

[...]

domain: siteduzero.com
reg_created: 2002-06-09 21:53:29
expires: 2009-06-09 21:53:29
created: 2007-02-27 06:56:43
changed: 2007-12-09 12:49:39
transfer-prohibited: yes
ns0: NS1.DINHOSTING.NET
ns1: BART.SITEDUZERO.COM
owner-c:
  nic-hdl: PD2500-GANDI
  owner-name: Simple IT SARL
  organisation: Simple IT SARL
  person: Pierre DUBUC
  address: '110, avenue du Général Leclerc'
  zipcode: 92340
  city: BOURG LA REINE

[...]
```

Utilisez ces informations avec parcimonie lorsque vous avez besoin de contacter le propriétaire d'un nom de domaine ou d'une adresse IP, de manière générale pour régler un litige mettant en jeu le nom de domaine ou l'IP en question.

ifconfig & netstat : gérer et analyser le trafic réseau

Nous allons découvrir ici 2 commandes : `ifconfig` et `netstat`. La première permet de gérer les connexions réseau de votre machine (pour les activer / désactiver par exemple), tandis que la seconde vous permet d'analyser ces connexions, de connaître des statistiques, etc.

ifconfig : liste des interfaces réseau

Votre ordinateur possède en général plusieurs **interfaces réseau**, c'est-à-dire plusieurs moyens de se connecter au réseau.

Tapez `ifconfig` dans la console pour voir ce que ça donne :

Code : Console

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:90:f5:56:44:5a
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Packets reçus:0 erreurs:0 :0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:0 (0.0 B) Octets transmis:0 (0.0 B)
          Interruption:220 Adresse de base:0xe000

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          Packets reçus:10 erreurs:0 :0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          Octets reçus:500 (500.0 B) Octets transmis:500 (500.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:19:d2:61:90:0a
          inet adr:192.168.1.2  Bcast:192.168.1.255  Masque:255.255.255.0
          adr inet6: fe80::219:d2ff:fe61:900a/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:5238 erreurs:0 :0 overruns:0 frame:0
          TX packets:4899 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:5069449 (5.0 MB) Octets transmis:1202459 (1.2 MB)
```

On distingue ici 3 interfaces réseau. Vous en avez peut-être plus, peut-être moins, tout dépend de votre ordinateur.

Les interfaces que j'ai sont assez courantes, détaillons-les :

- **eth0** : cela correspond à la connexion par câble réseau (ce qu'on appelle en général le câble RJ45). Si votre PC est relié au réseau via un câble, c'est sûrement ce moyen de communication que vous utilisez actuellement. Notez que certains ordinateurs (et notamment les serveurs) ont plusieurs sorties réseau filaires. Dans ce cas, vous devriez voir aussi des interfaces eth1, eth2, etc.
- **lo** : c'est la boucle locale. Tout le monde devrait avoir cette interface. Elle correspond à une connexion à... vous-même. C'est pour cela qu'on l'appelle la boucle locale : tout ce qui est envoyé par là vous revient automatiquement. Cela peut paraître inutile, mais on a parfois besoin de se connecter à soi-même pour des raisons pratiques.
- **wlan0** : il s'agit d'une connexion sans-fil type wifi. Là encore, bien que ce soit plus rare, si vous avez plusieurs cartes réseau sans fil, vous aurez un wlan1, wlan2, etc.



Câble RJ45

Observez les résultats de ma commande et essayez de deviner par quelle interface réseau je me connecte à internet.

...

Vous avez trouvé ? Il ne fallait pas avoir peur de lire le détail des messages. 😊

En effet, bien que je possède une sortie réseau filaire (RJ45), j'utilise ici le wifi comme en témoigne la ligne "*Packets reçus:5238*" pour le wifi wlan0 (alors qu'il y en a 0 pour eth0). C'est donc l'interface active que j'utilise le plus.

La commande ifconfig permet aussi de faire des réglages réseau. Cependant, cela sortirait un peu du cadre de ce cours et il vous

faudrait des connaissances en réseau pour bien l'utiliser.

Je vais cependant vous donner un réglage très simple que vous pouvez faire et qui pourra vous être utile : l'activation / désactivation d'interface.

Il suffit d'écrire une commande sous cette forme :

Code : Console

```
ifconfig interface etat
```

Remplacez :

- *interface* : par le nom de l'interface que vous voulez modifier (eth0, wlan0...)
- *etat* : par up ou down selon si vous voulez activer ou désactiver l'interface.

Exemple :

Code : Console

```
$ ifconfig eth0 down
```

... désactive l'interface eth0 (filaire). Plus aucun trafic ne pourra alors passer par l'interface eth0.

Code : Console

```
$ ifconfig eth0 up
```

... la réactive à nouveau.

Vous aurez peut-être besoin de connaître ces commandes un jour ou l'autre si vous devez désactiver puis réactiver une interface pour prendre en compte des changements dans la configuration de votre réseau.

netstat : statistiques sur le réseau

La commande `netstat` va vous paraître un peu complexe si vous n'avez pas trop de notions de réseau, mais elle est incontournable quand on veut savoir ce que notre machine est en train de faire sur le réseau.

Netstat peut afficher beaucoup d'informations. Pour sélectionner celles qui nous intéressent, on a recours à de nombreux paramètres.

Plutôt que de les expliquer un par un, je vais vous montrer quelques combinaisons de paramètres qui donnent des résultats intéressants.

netstat -i : statistiques des interfaces réseau

Essayez pour commencer l'option -i :

Code : Console

```
$ netstat -i
Table d'interfaces noyau
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500 0          0          0          0 0          0          0          0          0 BMU
lo        16436 0          10         0          0 0          0          10         0          0 LRU
wlan0     1500 0          5161       0          0 0          4810       0          0          0 BMRU
```

Vous n'aurez pas nécessairement les mêmes lignes que moi, tout dépend de votre ordinateur.

Il s'agit là d'un tableau présentant, pour chaque interface réseau que vous avez, une série de statistiques d'utilisation. On retrouve ici nos interfaces eth0, lo et wlan0.

Comme vous le voyez sur la colonne RX-ERR, c'est wlan0 qui est l'interface la plus active. Et vous noterez que "lo" est un petit peu utilisée elle aussi, comme quoi ça sert de se connecter à soi-même. 😊

Je ne rentrerai pas dans le détail de ces colonnes car c'est assez technique, mais vous savez au moins détecter l'activité de vos interfaces grâce à cette commande.

netstat -uta : lister toutes les connexions ouvertes

Code : Console

```
$ netstat -uta
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat
tcp      0      0 *:ssh *: * LISTEN
tcp      0      0 localhost:ipp *: * LISTEN
tcp      0      0 mateo21-laptop.lo:60997 debian-mirror.mirro:ftp ESTABLISHED
tcp      1      0 mateo21-laptop.lo:33721 lisa.siteduzero.com:www CLOSE_WAIT
tcp6     0      0 [::]:ssh [::]:* LISTEN
udp      0      0 *:bootpc *: *
udp      0      0 *:mdns *: *
udp      0      0 *:45176 *: *
```

Les options signifient :

- -u : afficher les connexions UDP.
- -t : afficher les connexions TCP.
- -a : afficher toutes les connexions quel que soit leur état.

TCP et UDP sont 2 protocoles différents pour envoyer des données sur le réseau.

UDP est plutôt utilisé dans les jeux en réseau et pour les communications vocales (avec Skype par exemple). De manière générale sinon, TCP est le protocole le plus utilisé. Je ne rentrerai pas plus dans le détail ici, mais vous pouvez vous renseigner si le sujet vous intéresse. 😊

Pour filtrer un peu, on va enlever les connexions UDP qui sont moins importantes la plupart du temps :

Code : Console

```
$ netstat -ta
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat
tcp      0      0 *:ssh *: * LISTEN
tcp      0      0 localhost:ipp *: * LISTEN
tcp      0      0 mateo21-laptop.lo:60997 debian-mirror.mirro:ftp ESTABLISHED
tcp      0      0 mateo21-laptop.lo:33721 lisa.siteduzero.com:www ESTABLISHED
```

```
tcp6      0      0  [::]:ssh          [::]:*           LISTEN
```

Ce tableau vous indique qui, depuis l'adresse locale, est connecté à qui (à une adresse distante).

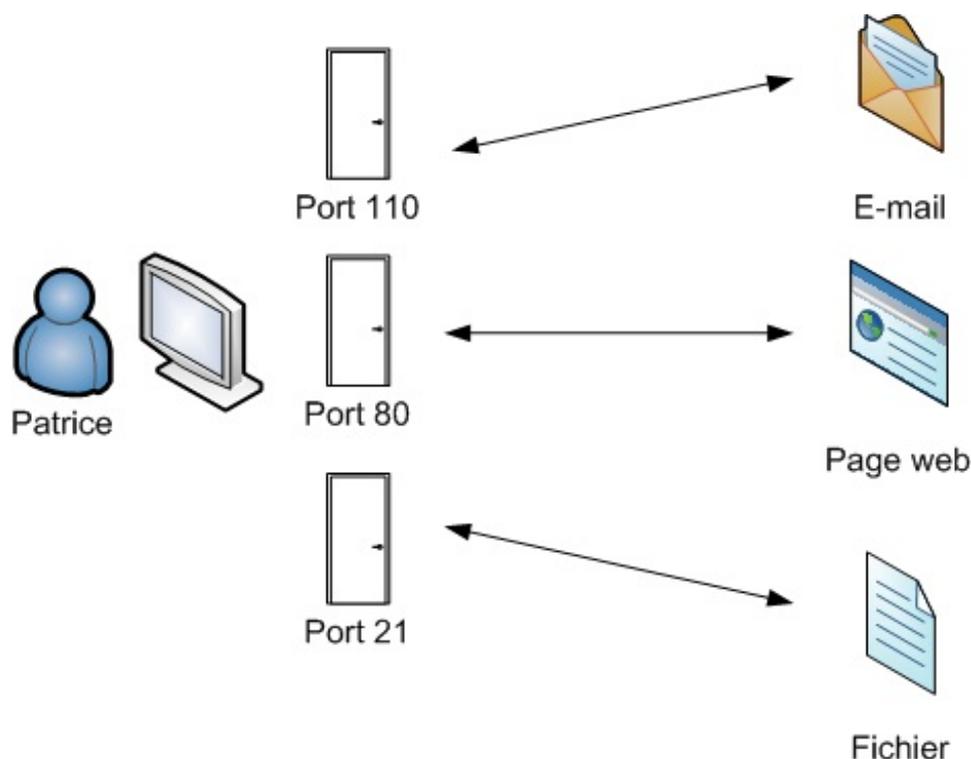
Chaque connexion a un état. Ici, on repère les états LISTEN et ESTABLISHED.

De nombreux états sont possibles, en voici quelques un à connaître :

- **ESTABLISHED** : la connexion a été établie avec l'ordinateur distant.
- **TIME_WAIT** : la connexion attend le traitement de tous les paquets encore sur le réseau avant de commencer la fermeture.
- **CLOSE_WAIT** : le serveur distant a arrêté la connexion de lui-même (peut-être parce que vous êtes resté inactif trop longtemps ?).
- **CLOSED** : la connexion n'est pas utilisée.
- **CLOSING** : la fermeture de la connexion est entamée, mais toutes les données n'ont pas encore été envoyées.
- **LISTEN** : à l'écoute des connexions entrantes.

Il y en a d'autres que vous pouvez lire dans la doc. Globalement, ce qu'il faut retenir, c'est que les connexions à l'état LISTEN ne sont pas actuellement utilisées mais qu'elles "écoutent" le réseau au cas où quelqu'un veuille se connecter à votre ordinateur.

Regardez en particulier le *port* sur lequel ces connexions écoutent (après le symbole ":"). C'est probablement l'information la plus intéressante. En effet, on peut se connecter à chaque ordinateur via différentes "portes" appelées ports. Chaque service utilise un port différent :



A la première ligne, vous avez `* : ssh`, ce qui signifie que SSH est en train d'écouter sur le port de SSH au cas où quelqu'un veuille se connecter à votre machine. C'est logique, puisque j'ai activé le serveur SSH pour pouvoir m'y connecter à distance au besoin. 😊

D'autres connexions, elles, sont déjà établies et donc en cours d'utilisation. Par exemple, au niveau de l'adresse distante, on voit que je suis connecté par FTP à `debian-mirror.mirro:ftp` et que je suis connecté à un serveur web `lisa.siteduzero.com:www` : en clair je suis en train de charger une page sur le Site du Zéro. 😊

Vous pouvez rajouter `-n` si vous voulez avoir les numéros des ports plutôt qu'une description en toutes lettres :

Code : Console

```
$ netstat -tan
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN
tcp 0 0 127.0.0.1:631 0.0.0.0:* LISTEN
tcp 15 0 192.168.1.2:60997 128.101.240.212:21 CLOSE_WAIT
tcp 0 0 192.168.1.2:54001 80.248.219.123:80 ESTABLISHED
tcp6 0 0 :::22 :::* LISTEN
```

Cela correspond aux ports qu'on connaît : 22 pour SSH, 21 pour FTP, 80 pour le web, etc.

netstat -lt : liste des connexions en état d'écoute

Très utile, l'option -l vous permet de filtrer les connexions à l'état LISTEN, et donc savoir quels ports de serveur sont susceptibles d'être utilisés en ce moment sur votre machine.

Code : Console

```
$ netstat -lt
Connexions Internet actives (seulement serveurs)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat
tcp 0 0 *:ssh *: * LISTEN
tcp 0 0 localhost:ipp *: * LISTEN
tcp6 0 0 [::]:ssh [::]: * LISTEN
```

netstat -s : statistiques résumées

Enfin, si vous êtes très friand de statistiques réseau, -s est fait pour vous :

Code : Console

```
$ netstat -s
Ip:
 7443 paquets reçus au total
 1 avec des en-têtes invalides
 8 avec des adresses invalides
 0 réacheminés
 0 paquets arrivant rejetés
 7354 paquets entrants délivrés
 7226 requêtes envoyées
Icmp:
 0 Messages ICMP reçus
 0 messages ICMP entrant échoués

[...]
```

Il y a un nombre important d'informations que je ne suis pas capable de décrypter moi-même. Vous aurez recours à ces statistiques uniquement dans des cas particuliers si vous devez déceler un comportement anormal sur le réseau.

iptables : le firewall de référence

Maintenant que nous savons analyser le trafic réseau et voir ainsi un peu ce qui se passe, nous allons nous atteler à filtrer le trafic à l'aide d'un firewall.

Le plus célèbre firewall utilisé sous Linux est `iptables`. Il permet d'établir un certain nombre de *règles* pour dire par quels ports on peut se connecter à votre ordinateur, mais aussi à quels ports vous avez le droit de vous connecter. On peut aussi filtrer par IP mais nous ne détaillerons pas cela ici.

Par exemple, si je veux empêcher toute connexion FTP (parce que je considère que le FTP n'est pas sûr), je peux souhaiter bloquer le port 21 (utilisé par FTP).

En général, la technique ne consiste pas à bloquer certains ports, mais plutôt à bloquer par défaut *tous* les ports et à en autoriser seulement quelques-uns.



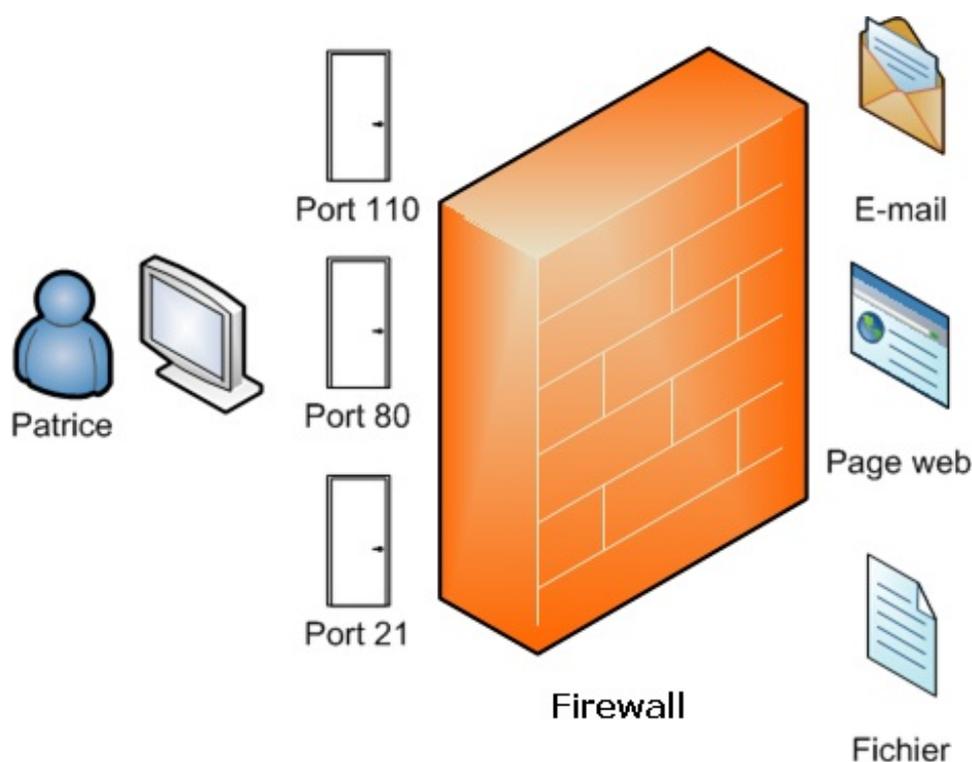
Attends, c'est quoi le but exactement ? Bloquer tout le trafic réseau ? Pour quoi faire ?

C'est avant tout une question de sécurité. Le but d'un firewall est d'empêcher que des programmes puissent communiquer sur le réseau sans votre accord. Aujourd'hui, même sous Windows depuis Windows XP SP2 un firewall est intégré par défaut, tant le problème est important.

Avoir un firewall ne vous prémunit pas contre les virus (bien que sous Linux ils restent rares). En revanche, cela rend la tâche *particulièrement difficile* aux pirates qui voudraient accéder à votre machine.

Vous vous souvenez ce que je vous ai expliqué un peu plus tôt ? Chaque ordinateur possède plusieurs portes d'entrées possibles.

Notre objectif est de bloquer par défaut toutes ces portes et d'autoriser seulement celles dont vous avez besoin, que vous considérez comme "sûres" et que vous utilisez. Par exemple, le port 80 utilisé pour le web est un port sûr que vous pouvez activer.



Notez, et c'est important, qu'il y a des portes d'entrée et des portes de sortie sur votre ordinateur (ce ne sont pas nécessairement les mêmes).



`Iptables` est un programme extrêmement puissant, mais tout aussi complexe. Nous ne verrons que des fonctionnalités basiques (et ça sera déjà pas mal 😊). Sachez qu'il peut faire bien plus que ce qu'on va voir : pour en savoir plus, comme d'habitude, lisez le manuel.

Iptables s'utilise en root

Pour manipuler iptables, vous devez impérativement être root. Pour la suite des opérations, je vous recommande donc de passer root dès maintenant :

Code : Console

```
$ sudo su
```

iptables -L : afficher les règles

Avec iptables -L (attention, un L majuscule), vous pouvez afficher les règles qui régissent le firewall actuellement :

Code : Console

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

On repère 3 sections :

- **Chain INPUT** : correspond aux règles manipulant le trafic entrant.
- **Chain FORWARD** : correspond aux règles manipulant la redirection de trafic.
- **Chain OUTPUT** : correspond aux règles manipulant le trafic sortant.

Nous ne verrons pas ici la section FORWARD. Iptables permet de rediriger le trafic, mais c'est assez compliqué et ça ne nous intéresse pas ici. Déjà avec INPUT et OUTPUT nous aurons de quoi faire. 😞

Actuellement chez moi, les règles sont vides. Il y a 3 tableaux mais aucune ligne à l'intérieur. Par ailleurs, vous noterez à chaque fois les mots (*policy ACCEPT*) qui signifient que, par défaut, tout le trafic est accepté. Donc chez moi, pour le moment, le firewall est tout simplement inactif car il ne bloque rien. Mon ordinateur est une vraie passoire. 😊

Si vous avez déjà des règles inscrites dans votre firewall (ce qui ne devrait pas être votre cas, mais on ne sait jamais), sachez que vous pouvez réinitialiser les règles.

Ne le faites que si vous êtes certain de vouloir le faire. En effet, sur un ordinateur partagé peut-être que quelqu'un a déjà configuré le firewall et il serait dommage de casser tout son travail. 😞



Code : Console

```
# iptables -F <--
Attention ! Réinitialise toutes les règles iptables !
```

Le principe des règles

Voici ce que cela pourrait donner lorsqu'on aura établi des règles, par exemple ici pour la section INPUT :

Code : Console

```
# iptables -L
Chain INPUT (policy DROP)
target      prot opt source                destination           tcp dpt:www
ACCEPT      tcp  -- anywhere              anywhere              tcp dpt:ssh
ACCEPT      tcp  -- anywhere              anywhere              tcp dpt:imap2
```

Première chose à savoir : **l'ordre des règles est important**. En effet, iptables lit les règles de haut en bas, et la position de ces règles influe sur le résultat final. Sachez donc que les règles sont numérotées.

Pour avoir les numéros, ajoutez `--line-numbers` :

Code : Console

```
# iptables -L --line-numbers
Chain INPUT (policy DROP)
num target      prot opt source                destination           tcp dpt:www
1  ACCEPT      tcp  -- anywhere              anywhere              tcp dpt:ssh
2  ACCEPT      tcp  -- anywhere              anywhere              tcp dpt:imap2
```

Ainsi, la règle filtrant SSH est la règle n°2.

Chaque ligne correspond à une règle différente qui permet de filtrer ou non une IP ou un port. Parmi les colonnes intéressantes, on note :

- **target** : ce que fait la règle. Ici c'est ACCEPT, c'est-à-dire que cette ligne autorise un port et / ou une IP.
- **prot** : le protocole utilisé (tcp, udp, icmp). Je rappelle que TCP est le plus utilisé. ICMP permet à votre ordinateur de répondre aux requêtes de type "ping".
- **source** : l'IP de source. Pour INPUT la source est l'ordinateur distant qui se connecte à vous.
- **destination** : l'IP de destination. Pour OUTPUT, c'est l'ordinateur auquel on se connecte.
- **Dernière colonne** : elle indique le port après les deux-points ":". Ce port est affiché en toutes lettres, mais avec -n vous pouvez obtenir le numéro correspondant.

Sur mon exemple, seuls les ports web, ssh et imap2 (mail) sont autorisés en entrée. Personne ne peut se connecter à la machine par un autre biais.

En effet, si vous regardez bien, par défaut j'ai configuré le firewall pour qu'il ignore tous les autres paquets : (*policy DROP*).

Nous allons apprendre à faire tout cela maintenant. 😊

Ajouter et supprimer des règles

Voici les principales commandes à connaître :

- **-A chain** : ajoute une règle en fin de liste pour la chain indiquée (INPUT ou OUTPUT par exemple).
- **-D chain rulenum** : supprime la règle n° rulenum pour la chain indiquée.
- **-I chain rulenum** : insère une règle au milieu de la liste à la position indiquée par rulenum. Si vous n'indiquez pas de position rulenum, la règle sera insérée en premier, tout en haut dans la liste.
- **-R chain rulenum** : remplace la règle n° rulenum dans la chain indiquée.
- **-L** : liste les règles (nous l'avons déjà vu).
- **-F chain** : vide toutes les règles de la chain indiquée. Cela revient à supprimer toutes les règles une par une pour cette chain.
- **-P chain regle** : modifie la règle par défaut pour la chain. Cela permet de dire par exemple que par défaut tous les ports sont fermés, sauf ceux qu'on a indiqués dans les règles.

De manière générale, l'ajout d'une règle se passe suivant ce schéma :

Code : Console

```
iptables -A (chain) -p (protocole) --dport (port) -j (décision)
```

Remplacez chain par la section qui vous intéresse (INPUT ou OUTPUT), protocole par le nom du protocole à filtrer (TCP, UDP, ICMP...) et enfin décision par la décision à prendre : ACCEPT pour accepter le paquet, REJECT pour le rejeter ou bien DROP pour l'ignorer complètement.

Le mieux est de découvrir comment on ajoute une règle par une série d'exemples. 😊

Code : Console

```
# iptables -A INPUT -p tcp --dport ssh -j ACCEPT
```

Cela ajoute à la section INPUT (donc pour le trafic entrant) une règle sur les données reçues via le protocole TCP, sur le port de ssh (vous pouvez remplacer "ssh" par le numéro du port, soit 22). Lorsque votre ordinateur recevra des données en TCP sur le port de SSH, celles-ci seront acceptées. Cela vous permettra donc de vous connecter à distance à votre PC via SSH.

Vous pouvez faire de même avec d'autres ports :

Code : Console

```
# iptables -A INPUT -p tcp --dport www -j ACCEPT
```

... pour le web (80).

Code : Console

```
# iptables -A INPUT -p tcp --dport imap2 -j ACCEPT
```

... pour les mails, etc.



Si vous ne précisez pas de port (en omettant la section dport), tous les ports seront acceptés !

Autoriser les pings

En plus d'autoriser le trafic sur ces ports, je peux vous conseiller d'autoriser le protocole ICMP (pour pouvoir faire un ping) sur tous les ports :

Code : Console

```
# iptables -A INPUT -p icmp -j ACCEPT
```

Comme je n'ai pas indiqué de section `--dport`, cette règle s'applique à tous les ports, *mais pour les pings (icmp) uniquement !*

Votre ordinateur répondra alors aux "pings" pour indiquer qu'il est bien en vie.

Vos règles Iptables pour INPUT devraient maintenant ressembler à ceci :

Code : Console

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            tcp dpt:www
ACCEPT     tcp  --  anywhere              anywhere                tcp dpt:ssh
ACCEPT     tcp  --  anywhere              anywhere                tcp dpt:imap2
ACCEPT     icmp --  anywhere              anywhere
```

Autoriser les connexions locales et déjà ouvertes

Pour l'instant, nos règles sont encore un peu trop restrictives et pas vraiment utilisables (vous risquez de ne plus pouvoir faire grand chose).

Je vous propose de rajouter 2 règles pour "assouplir" un peu votre pare-feu et le rendre enfin utilisable.

Code : Console

```
# iptables -A INPUT -i lo -j ACCEPT
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Ces 2 règles utilisent des options un peu différentes de celles que nous avons vues jusqu'ici. Voici quelques explications :

1. La première règle autorise tout le trafic sur l'interface de loopback locale, grâce à `-i lo`. Il n'y a pas de risque à autoriser votre ordinateur à communiquer avec lui-même, et il en a parfois besoin !
2. La seconde règle autorise toutes les connexions qui sont déjà à l'état ESTABLISHED ou RELATED. En clair, elle autorise toutes les connexions qui ont été demandées par votre PC. Là encore, cela permet d'assouplir le firewall et de le rendre fonctionnel pour une utilisation quotidienne.

Refuser toutes les autres connexions par défaut

Il reste un point essentiel, car pour l'instant ce filtrage ne sert à rien. En effet, nous avons indiqué quelles données nous autorisions, mais *nous n'avons pas dit que toutes les autres devaient être refusées* !

Changez donc la règle par défaut pour DROP par exemple :

Code : Console

```
# iptables -P INPUT DROP
```

Iptables devrait maintenant indiquer que par défaut tout est refusé, sauf pour les lignes acceptées indiquées dans le tableau :

Code : Console

```
# iptables -L
Chain INPUT (policy DROP)
target     prot opt source                destination            tcp dpt:www
ACCEPT     tcp  --  anywhere              anywhere               tcp dpt:ssh
ACCEPT     tcp  --  anywhere              anywhere               tcp dpt:imap2
ACCEPT     icmp --  anywhere              anywhere
```

Le filtrage est radical. Nous n'avons pas autorisé beaucoup de ports, et il se pourrait que vous vous rendiez compte que certaines applications n'arrivent plus à accéder à internet (normal, leur port doit être filtré).

A vous de savoir quels ports ces applications utilisent pour modifier les règles en conséquence.

Pensez à faire de même pour les règles de sortie (OUTPUT) si vous le désirez. 😊

Appliquer les règles au démarrage

Si vous redémarrez votre ordinateur, les règles iptables auront disparu !

Le seul moyen pour qu'elles soient chargées au démarrage consiste à créer un script qui sera exécuté au démarrage.

Justement, ça tombe bien, nous allons étudier la programmation de scripts shell sous Linux dans la prochaine partie. 😊

En attendant, si vous voulez lire un mode d'emploi rapide pour mettre les règles au démarrage, je vous invite à lire la [doc ubuntu-fr](#).



Iptables est donc un firewall assez compliqué comme vous avez pu le constater. Sachant cela, des développeurs ont travaillé sur un programme qui simplifie l'utilisation d'iptables : **ufw (Uncomplicated Firewall)**. Ce programme n'est pas disponible partout contrairement à iptables, mais on le trouve dans les versions récentes d'Ubuntu.

On ne s'improvise pas administrateur réseau du jour au lendemain comme vous avez pu le voir. 😊

Ce chapitre est une introduction aux réseaux et il faudrait un cours tout entier pour en parler correctement. J'ai donc dû me contenter de vous faire découvrir l'essentiel sur le réseau et je vous ai amené à manipuler le firewall iptables.

A propos d'iptables : c'est un sujet très complexe. Nous l'avons seulement effleuré. Je vous invite à lire des tutoriels spécialisés sur iptables, à condition que vous ayez acquis des connaissances en réseau auparavant (si on rentre dans le détail, c'est très pointu).

Ce chapitre devrait en tout cas vous avoir donné les moyens de mettre en place un certain nombre de règles basiques (mais efficaces) de filtrage pour mieux sécuriser votre ordinateur et votre serveur. 😊

Partie 5 : Programmez des scripts Bash pour automatiser vos tâches

Vous avez découvert de nombreuses commandes du monde de Linux depuis le début du cours. C'est bien !

Nous allons terminer en beauté en réutilisant tout ce que nous avons appris pour programmer des *scripts shell* (et plus exactement des *scripts Bash*). L'idée est d'assembler les commandes entre elles, d'utiliser des conditions et des boucles, afin d'automatiser certaines tâches répétitives qu'on a l'habitude de faire.

Vous voulez archiver vos travaux tous les soirs, les compresser et les envoyer par mail ou par FTP ? Vous allez apprendre à faire cela, et bien plus encore !

Vim : l'éditeur de texte du programmeur

Dans cette dernière partie, nous allons réunir toutes les connaissances que nous avons acquises sur les commandes utilisées sur Linux. Nous allons les combiner entre elles et créer ce que l'on appelle **des scripts shell**.

Le scripting shell est un mini-langage de programmation intégré à tous les systèmes Linux qui vous permet d'automatiser des tâches répétitives. Il s'agit d'un élément très puissant du système que vous devez absolument connaître.

Toutefois, pour programmer, il va vous falloir utiliser un éditeur de texte digne de ce nom. Certes, vous connaissez déjà **Nano**, mais comme je vous l'ai déjà dit il est très basique. On l'a utilisé pour simplifier les choses au début, mais maintenant il est temps de passer à quelque chose de plus complet et de plus puissant : **Vim** (prononcez "*Vi aille ème*").

Installer Vim

Sous Linux, il y a 2 éditeurs de texte puissants en console à connaître :

- **Vim** : il s'agit d'une version améliorée d'un des plus anciens éditeurs en console, qui s'appellait "Vi" (prononcez les lettres en anglais "*vi aille*").
Vim (VI iMproved, version améliorée de Vi) est largement répandu et généralement disponible par défaut sur la plupart des OS basés sur Unix, comme Linux.
- **Emacs** : développé par Richard Stallman, le fondateur du projet GNU dont je vous ai parlé au début du cours, cet éditeur concurrent a lui aussi bien des atouts. On le retrouve plus spécifiquement sous Linux et il est rarement installé par défaut (un petit apt-get suffit toutefois 😊). Il peut être complété par toute une série de plugins qui lui permettent de faire navigateur web, lecteur audio... Bref, c'est un peu un outil à tout faire.

Sachez qu'il est courant que les gens adoptent l'un ou l'autre de ces éditeurs et le défendent bec et ongles. Choisir un éditeur de texte sous Linux, c'est en fait un peu comme choisir une religion. Oui je sais, ils sont fous ces Linuxiens !



Oulah, c'est important alors ! Lequel choisir ?

En fait, rien ne vous empêche d'apprendre à utiliser les deux. Toutefois, ces logiciels sont tellement complets qu'il vous faudra du temps pour vous habituer à chacun d'eux.

Dans la pratique, on prend l'habitude d'en choisir un et de s'y tenir : il est donc rare de voir quelqu'un naviguer entre les deux.

Vim ou Emacs ? Emacs ou Vim ?

Tout cela ne répond pas à votre question, je sais. Mais ne comptez pas sur moi pour vous dire "*Utilisez celui-là, il est mieux*" : des milliers de trolleurs le font mieux que moi sur tous les forums du monde. Et je pourrais m'attirer les foudres divines des adorateurs de l'un ou l'autre éditeur si je m'y risquais. 🤔

D'ailleurs, vous devriez vous mettre en tête dès maintenant qu'il n'y en a pas un qui soit nul et l'autre génial. Ce sont juste 2 conceptions un peu différentes de ce que doit être un éditeur de texte.

Le meilleur conseil que je puisse vous donner est le suivant : **choisissez d'utiliser le même éditeur que votre ami pro de Linux ou votre collègue de bureau**. L'idéal est d'avoir quelqu'un à proximité qui peut régulièrement vous conseiller. Croyez-moi, s'il y a bien un conseil qui soit important dans ce chapitre, c'est celui-là.



Et toi, ton éditeur c'est quoi ?

Je craignais cette question mais il fallait bien qu'elle tombe un jour... 😊

Pour ma part, je n'ai jamais eu l'occasion de prendre le temps d'apprendre à utiliser Emacs. Le professeur qui m'a initié à Linux était un habitué de Vim (mais il n'a jamais dit de mal d'Emacs, je le jure, je le jure !).

Je suis donc à mon tour un habitué de Vim, et c'est lui que je vous présenterai dans ce cours. Je ne suis toutefois pas sectaire, si quelqu'un rédige un bon tutoriel sur Emacs sur le Site du Zéro, je serai ravi de vous inviter à le lire lui aussi. 😊

Installer et lancer Vim

Sur la plupart des distributions Linux, Vim est en général installé par défaut. Mais j'ai bien dit *en général*. Rien n'assure que Vim soit installé chez vous par défaut. Après tout, c'est la distribution qui choisit les programmes qu'elle vous installe au départ.

D'ailleurs, sous Ubuntu, il faut savoir que ce n'est pas Vim qui est installé mais Vim-tiny, une version allégée de Vim. Personnellement elle ne me convient pas et est limitée en possibilités. Je vous invite donc à installer le vrai Vim complet en tapant :

Code : Console

```
sudo apt-get install vim
```

Vous pourrez alors lancer Vim en tapant la commande `vim` (la commande `vi` fonctionne aussi mais il est recommandé de taper plutôt `vim`).

vimtutor : le programme qui vous apprend à utiliser Vim !

Il existe un véritable petit tutoriel intégré à Vim pour les nouveaux ! 😊

Ce programme peut être lancé en tapant :

Code : Console

```
vimtutor
```

Si vous ne l'avez pas, installez le paquet vim-common. Mais normalement il devrait déjà être installé.

En fait, vimtutor lance simplement Vim en ouvrant un fichier d'aide prédéfini. Cette introduction à Vim est d'ailleurs en français et est accessible à tout le monde, aussi je vous invite à l'essayer et à la lire en complément de mon cours à moi. 😊

Petit aperçu :

Code : Console

```
=====
= B i e n v e n u e  d a n s  l e  T u t o r i e l  d e  V I M  -  V e r s i o n  1.5.fr.2  =
=====

Vim est un éditeur très puissant qui a trop de commandes pour pouvoir
toutes les expliquer dans un cours comme celui-ci, qui est conçu pour en
```

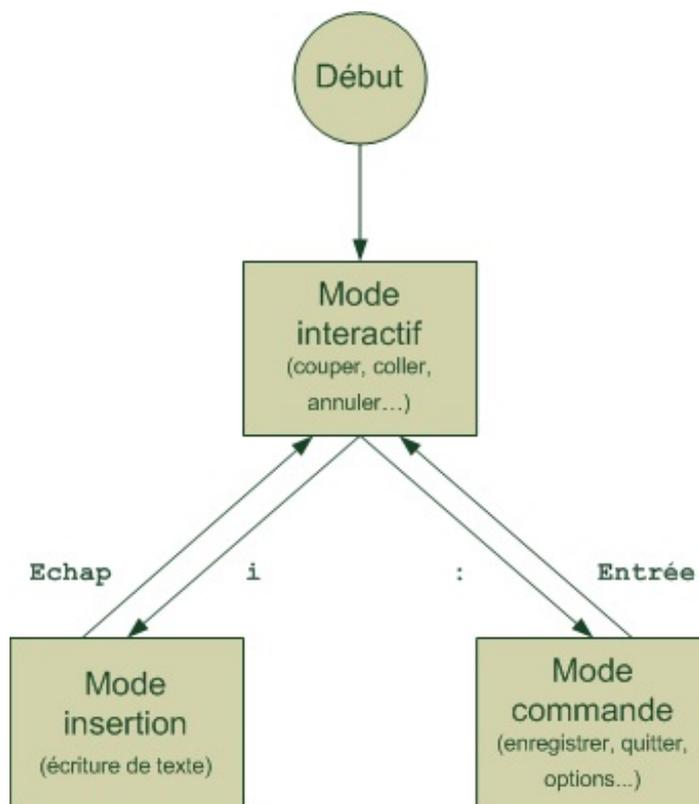

fallu un peu de temps pour m'y habituer et il vous en faudra aussi, mais le jeu en vaut la chandelle.

Si on ne vous explique rien, vous risquez d'être un peu perdus. Pire, vous aurez même du mal à sortir de Vim (ne rigolez pas, ça m'est arrivé la première fois !).

Voilà ce qu'il faut savoir... et qu'on aurait dû m'expliquer dès le départ d'ailleurs. Vim possède 3 modes de travail différents :

- **Mode interactif** : c'est le mode par défaut par lequel vous commencez. En lançant Vim, vous êtes donc en mode interactif. Dans ce mode, vous ne pouvez pas écrire de texte (oui je sais, c'est le comble pour un éditeur de texte !). N'essayez donc pas d'appuyer sur des lettres au hasard, car vous risquez de faire n'importe quoi !
Le mode interactif est un mode puissant qui permet de se déplacer dans le texte, supprimer une ligne, copier/coller du texte, aller à une ligne précise, annuler ses actions, etc. Chaque action peut être déclenchée en appuyant sur une touche du clavier (par exemple, on appuie sur "u" pour annuler la dernière action).
- **Mode insertion** : ça c'est le mode que vous connaissez ! Vous tapez du texte, et il s'insère à l'endroit où le curseur se trouve. Magique !
Pour rentrer dans ce mode, il y a plusieurs possibilités. Une des plus courantes est d'appuyer sur la touche "i" (insertion). Pour en sortir, il faut appuyer sur la touche "Echap".
- **Mode commande** : ce mode permet de lancer des commandes telles que "quitter", "enregistrer", etc. Vous pouvez aussi l'utiliser pour activer des options de Vim (comme la coloration syntaxique, l'affichage du numéro des lignes...), et vous pouvez même envoyer des commandes au shell (à la console) telles que ls, locate, cp, etc.
Pour activer ce mode, vous devez être en mode interactif et appuyer sur la touche deux-points ":". Vous validez la commande avec la touche Entrée et revenez alors au mode interactif.

Je résume. Vim possède 3 modes : interactif, insertion et commande. Vous démarrez en mode interactif. Le seul mode que vous connaissez et qui ne sera pas nouveau pour vous est le mode insertion. Les 2 autres modes (interactif et commande) vont vous surprendre un peu.



Pourquoi avoir fait autant de modes qui ont l'air si compliqués dans un éditeur de texte ? Pourquoi n'y a-t-il pas de menus ? Et pourquoi ne pas utiliser plutôt un éditeur de texte graphique, c'est quand même plus simple avec une souris !

Trop de questions à la fois. 😊

Je vais essayer de vous répondre simplement, et il va falloir me croire sur parole au début : si des gens se sont amusés à créer tous ces "modes" et tous ces raccourcis clavier, ce n'est pas juste pour le plaisir torturé de faire la même chose qu'avec une souris et des menus mais avec un clavier.

En fait, vous allez vite découvrir que vous pouvez faire des choses que vous ne soupçonniez pas avec un éditeur de texte : supprimer le mot actuel, couper le texte du curseur jusqu'à la fin de la ligne, coller 4 fois le texte qui se trouve dans le presse-papiers, sauter à la ligne n°453, sauter à la dernière ligne, etc.

Toutes ces choses-là se font au clavier et, pour la plupart d'entre elles, vous devrez retenir par coeur quelle touche correspond à quelle action. C'est un peu contraignant au départ, mais imaginez que c'est comme apprendre à taper à 10 doigts au clavier comme un dactylo : au début, c'est dur, vous avez l'impression de ramer, d'aller moins vite qu'avant, mais petit à petit vous gagnez en productivité, vous allez de plus en plus vite, et finalement vous vous demandez comment vous avez pu faire autant de temps sans connaître tout ça. 😊

Et pour ceux qui voudraient une interface graphique, sachez que Vim a été porté sous Gnome sous le nom gVim. Vous pouvez donc l'installer (même si vous utilisez KDE hein) et le lancer, le fonctionnement est identique au Vim de la console. Il est même disponible en version Windows, si c'est pas beau ça !

Cette fenêtre affiche par défaut des menus et une barre d'outils, comme un éditeur de texte classique. Un habitué du Vim console aura plutôt tendance à utiliser les raccourcis clavier qui vont plus vite, bien entendu.



gVim sous Vista

Opérations basiques (déplacement, écriture, enregistrement...)

Nous allons découvrir Vim à travers plusieurs étapes de plus en plus complexes. Nous commençons en douceur. 😊

L'ouverture de Vim

Pour le moment, si vous lancez Vim en tapant juste la commande sans paramètre :

Code : Console

```
vim
```

... il s'ouvre sur un nouveau fichier vide comme ceci que vous avez déjà vu :

```
VIM - Vi IMproved
      version 7.0.235
      by Bram Moolenaar et al.
Vim is open source and freely distributable

      Help poor children in Uganda!
type  :help iccf<Enter>      for information

type  :q<Enter>              to exit
type  :help<Enter> or <F1>  for on-line help
type  :help version7<Enter> for version info

                                0,0-1      All
```

(le texte d'introduction que vous voyez disparaîtra dès que vous écrirez les premiers caractères)

Vous pouvez aussi ouvrir un fichier en rajoutant son nom en paramètre :

Code : Console

```
vim nomdufichier
```

Si le fichier n'existe pas, il sera créé.

i : insérer du texte

Nous allons partir sur un fichier vide. Nous souhaitons commencer à rentrer du texte (quoi de plus normal pour un éditeur de texte après tout ?).

Appuyez sur "i" (i minuscule). Vous basculez alors en mode insertion. Vous pouvez maintenant taper du texte :

Code : Console

```
"monfichier" [New] 4L, 185C written          4,101-  
98      All
```

:q : quitter

Maintenant que vous avez enregistré, vous pouvez quitter Vim en tapant **:q**



J'ai essayé de quitter en ayant fait des modifications après avoir enregistré, et un message d'erreur s'affiche en rouge : *"No write since last change"*.

Vim vous interdit de quitter si vous n'avez pas enregistré vos changements. Vous pouvez toutefois forcer la fermeture de Vim en ajoutant un point d'exclamation à la fin : **:q!**. Cette fois, il n'y aura pas d'erreur.

:wq : enregistrer puis quitter

C'est la combinaison des 2 commandes que nous venons de voir. Vous enregistrez et quittez immédiatement Vim lorsque vous tapez **:wq**.

Opérations standard (copier, coller, annuler...)

Nous avons vu le strict minimum qu'il faut connaître pour se débrouiller dans Vim. Si ce n'est pas difficile, il faut bien avouer que c'est perturbant. Prenez donc le temps de vous y habituer.

Nous allons aller un peu plus loin ici, et c'est maintenant que vous allez commencer à trouver Vim pratique (et parfois même étonnant). Nous allons faire la majorité de ces actions en mode interactif : appuyez sur la touche **Echap** si vous n'y êtes pas déjà.

x : effacer des lettres

Placez le curseur sur une lettre en mode interactif, puis appuyez sur **x** pour l'effacer. Cela revient à appuyer sur "Suppr" en mode insertion.

On peut aller plus loin et effacer plusieurs lettres d'un coup. Pour cela, utilisez la formule suivante :

(nombre) x

Par exemple, si vous tapez **4x** (4 puis x), vous supprimerez les 4 prochaines lettres en partant du curseur.



Vous devez taper 4 puis x. Ne vous étonnez pas si rien ne s'affiche à l'écran lorsque vous avez tapé 4, c'est normal. Ecrivez la commande jusqu'au bout, cela fonctionnera.

d : effacer des mots, lignes...

De la même manière, on utilise aussi la touche **d** pour supprimer des mots et des lignes.

Commençons par supprimer une ou plusieurs lignes.

dd : supprimer une ligne

Appuyez 2 fois sur **d** (**dd**) pour supprimer toute la ligne sur laquelle se trouve le curseur.

Mieux : vous pouvez faire précéder cette instruction d'un nombre de lignes à supprimer. Par exemple, si vous tapez **2dd**, vous supprimerez 2 lignes d'un coup.



Encore une fois, ne vous étonnez pas si, juste après avoir tapé "2", rien ne s'affiche à l'écran. L'information est enregistrée par Vim en mémoire mais l'action ne sera vraiment exécutée que lorsque vous aurez tapé entièrement "2dd".

Note importante : la ligne ainsi supprimée est en fait "coupée" et placée en mémoire. Elle peut être collée, comme on le verra plus loin, avec la touche **p**.

dw : supprimer un mot

Placez le curseur sur la première lettre d'un mot. Tapez ensuite **dw** (*delete word*) : cela supprime le mot en entier ! Si le curseur est positionné au milieu du mot, vous ne supprimerez que les prochains caractères du mot (jusqu'à l'espace qui suit).

Vous pouvez aussi supprimer les 3 prochains mots en tapant **3dw**. Notez que le 3 peut être placé entre le d et le w, cela revient au même : **d3w** (ce qui peut se lire "*delete 3 words*").

d0 et d\$: supprimer le début ou la fin de la ligne

Vous vous souvenez de 0 et \$? Je vous avais demandé de les utiliser à la place des touches "Origine" et "Fin" car on allait en avoir besoin à nouveau par la suite. Le moment est venu de s'en resservir. 😊

- En tapant **d0**, vous supprimez du curseur jusqu'au début de la ligne.
- En tapant **d\$**, vous supprimez du curseur jusqu'à la fin de la ligne.

Pratique !

yy : copier une ligne en mémoire

yy copie la ligne actuelle en mémoire.

Cela fonctionne comme **dd** qui la "coupe". Vous pouvez aussi utiliser **yw** (pour copier un mot), **y\$** pour copier du curseur jusqu'à la fin de la ligne, etc.

p : coller

Si vous avez "coupé" du texte avec **dd** ou copié du texte avec **yy** (ou un de leurs équivalents) vous pouvez ensuite le coller avec la touche **p**.



Attention, retenez bien ceci : si vous avez copié une ligne en mémoire et que vous appuyez sur "p", elle sera collée sur la ligne située après le curseur.

On est parfois surpris de voir où se colle le texte, donc prenez le temps de vous y habituer.

Vous pouvez aussi coller plusieurs fois un texte en faisant précéder le "p" d'un nombre. Par exemple, **8p** collera 8 fois le texte en mémoire.

Si je place mon curseur sur une ligne, que je tape **yy** puis **8p**, je la collerai donc 8 fois !

```

Salut les Zéros !

J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
Mine de rien, une fois qu'on a basculé en mode insertion, écrire du texte n'est
pas compliqué. ;o)
~
~
~
~
~
~
~
~
~
~
8 more lines                                     4,1                                     All

```

r : remplacer une lettre

Si vous avez fait une faute sur seulement une lettre, vous pouvez passer en mode remplacement.

Placez le curseur sur la lettre à remplacer. Tapez **r** suivi de la lettre que vous voulez mettre à la place. Par exemple, **rs** remplace la lettre actuelle par un "s".

Si vous utilisez un **R** majuscule, cette fois vous basculerez dans le mode remplacement : vous pourrez remplacer plusieurs lettres à la fois. Vous pouvez par exemple écrire **Rbonjour** pour remplacer les caractères par "bonjour".

Pour revenir au mode interactif normal, appuyez sur **Echap**.

u : annuler les modifications

Pour annuler vos dernière modifications, appuyez sur **u** (*undo*). Si vous souhaitez annuler vos 4 dernières modifications, appuyez sur **4u** (vous commencez à connaître la formule, c'est toujours la même 😊).

Pour répéter un changement (= annuler une annulation), appuyez sur **Ctrl + R**.

G : sauter à la ligne n°X

Toutes les lignes d'un fichier ont un numéro. La numérotation commence à 1.

Regardez bien en bas à droite de Vim, vous devriez voir quelque chose comme 4,3. 4 correspond au numéro de la ligne sur laquelle se trouve le curseur, et 3 correspond au numéro de la colonne (3ème lettre de la ligne).

Vous pouvez directement sauter à la ligne n°7 par exemple en tapant **7G** (attention, c'est un G majuscule, donc pensez à laisser la touche "Maj" appuyée).

Pour sauter à la dernière ligne, tapez simplement **G**.

Pour revenir à la première ligne, tapez **gg**.

Opérations avancées (split, fusion, recherche...)

Nous avons vu l'essentiel des commandes les plus courantes. Nous allons maintenant découvrir une série de commandes un peu plus complexes (quoique, ça dépend desquelles 😊) parmi lesquelles la fusion de fichiers, la recherche, le remplacement, le découpage de l'écran (split), etc.

Toutes ces commandes se lancent depuis le mode interactif.

/recherche : rechercher un mot

Si vous tapez **/**, vous passez en mode recherche. Le curseur se place en bas de l'écran (vous indiquant que vous êtes passé en mode commande).

Ecrivez ensuite le mot que vous recherchez, par exemple "remplir" : **/remplir**. Tapez ensuite sur **Entrée** pour valider.

Le curseur se place alors sur la prochaine occurrence de "remplir" dans le fichier.

Pour passer à la prochaine occurrence du mot plus bas dans le fichier (s'il apparaît plusieurs fois), appuyez sur **n**. Pour rechercher en arrière, appuyez sur **N** (Maj + n).



Si vous souhaitez dès le départ lancer une recherche qui remonte vers le début du fichier, utilisez **?** au lieu de **/** pour lancer la recherche. Le fonctionnement reste le même.

:s : rechercher et remplacer du texte

Pour rechercher et remplacer du texte, c'est un peu plus compliqué. Il y a en effet plusieurs façons d'effectuer le remplacement.

La façon la plus simple de faire une recherche est de taper **:s/ancien/nouveau** pour rechercher "ancien" et le remplacer par "nouveau". Problème... cela ne remplacera que la première occurrence d'ancien par nouveau.

Voici toutes les variantes à connaître :

- **:s/ancien/nouveau** : remplace la première occurrence de la ligne où se trouve le curseur.
- **:s/ancien/nouveau/g** : remplace toutes les occurrences de la ligne où se trouve le curseur.
- **:#,#s/ancien/nouveau/g** : remplace toutes les occurrences dans les lignes n°# à # du fichier.
- **:%s/ancien/nouveau/g** : remplace toutes les occurrences dans tout le fichier. C'est peut-être ce que vous utiliserez le plus fréquemment.

:r : fusion de fichiers

Avec **r**, vous pouvez insérer un fichier à la position du curseur. Vous devez indiquer le nom du fichier à insérer, par exemple **:r autrefichier**.

L'auto-complétion avec Tab fonctionne là-aussi, donc pas besoin d'écrire le nom du fichier en entier !

Le découpage d'écran (split)

Vim possède une fonctionnalité pratique : il permet de découper l'écran et d'ouvrir plusieurs fichiers

:sp : découper l'écran horizontalement

Le plus simple pour commencer est de découper l'écran horizontalement. Tapez la commande `:sp` pour découper l'écran en deux :

```
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
@
monfichier [+] 3,1 66%
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
monfichier [+] 3,1 33%
:sp
```

Le fichier est ouvert une seconde fois (ce qui vous permet de voir 2 endroits différents du fichier à la fois) mais il est bien entendu possible d'ouvrir 2 fichiers différents. Pour faire cela, rajoutez le nom du fichier à ouvrir à la suite de la commande `:sp autre fichier`. Bonne nouvelle : l'auto-complétion avec la touche Tab fonctionne aussi dans Vim! 😊

Vous pouvez retaper `:sp` pour découper cette fois l'écran en 3 et ainsi de suite, mais gare à la lisibilité ensuite. 😊

`:vsp` : découper l'écran verticalement

Si le découpage horizontal par défaut ne vous convient pas, sachez que vous pouvez aussi effectuer un découpage vertical avec `:vsp`.

```

Salut les Zéros !
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
J'écris juste quelques lignes dans Vim pour remplir l'écran.
monfichier [+]      3,1      Top monfichier [+]      3,1      Top
:vsp

```

Il est bien entendu possible de répéter plusieurs fois la commande, et même de combiner des découpages verticaux et horizontaux. 🤖

Les principaux raccourcis en écran splitté

Chaque morceau de l'écran (correspondant à un fichier) est appelé *viewport*.

Voici une liste de raccourcis pratiques que vous pouvez utiliser lorsque l'écran est splitté (découpé) :

- **Ctrl + w** puis **Ctrl + w** : navigue de viewport en viewport. Répétez l'opération plusieurs fois pour accéder au viewport désiré.
- **Ctrl + w** puis **j** : déplace le curseur pour aller au viewport juste en-dessous. La même chose fonctionne avec les touches h, k et l que l'on utilise traditionnellement pour se déplacer dans Vim.
- **Ctrl + w** puis **+** : agrandit le viewport actuel.
- **Ctrl + w** puis **-** : réduit le viewport actuel.
- **Ctrl + w** puis **=** : égalise à nouveau la taille des viewports.
- **Ctrl + w** puis **r** : échange la position des viewports. Fonctionne aussi avec "R" majuscule pour échanger en sens inverse.
- **Ctrl + w** puis **q** : ferme le viewport actuel.

Voilà qui devrait vous permettre de faire ce que vous voulez en écran splitté. 😊

!: lancer une commande externe

Il est possible d'écrire des commandes traditionnelles du shell directement dans Vim. Pour cela, commencez par taper **!** suivi du nom de la commande.

Essayez par exemple de taper **!ls**. Vous afficherez alors le contenu du dossier dans lequel vous vous trouvez ! Cette fonctionnalité est bien pratique pour effectuer quelques actions sans avoir à quitter Vim.

Les options de Vim

Vim peut être personnalisé de 2 façons différentes :

- En activant ou désactivant des options. La [documentation complète des options](#) est disponible en ligne.
- En installant des plugins. Voir la [page officielle des plugins les plus téléchargés de Vim](#).

Nous n'allons pas passer en revue les plugins, mais il y a un certain nombre d'options intéressantes qu'il vaut le coup d'activer.

Le fonctionnement des options

Les options peuvent être activées après le lancement de Vim en lançant des commandes. Cependant, ces options seront "oubliées" dès que vous quitterez Vim.

Si vous voulez que les options soient activées à chaque démarrage de Vim, il faut créer un fichier de configuration `.vimrc` dans votre répertoire personnel.

Activer des options en mode commande

La première méthode consiste à activer l'option en mode commande. Une fois Vim ouvert, pour activer l'option nommée "option", tapez :

```
:set option
```

Pour la désactiver, tapez :

```
:set nooption
```

Il faut donc rajouter le préfixe "no" devant le nom de l'option pour la désactiver.

Certaines options doivent être précisées avec une valeur, comme ceci :

```
:set option=valeur
```

Pour connaître l'état d'une option :

```
:set option?
```

Activer des options dans un fichier de configuration

C'est à mon avis la meilleure façon de procéder. Commencez par copier un fichier de configuration déjà commenté qui vous servira d'exemple. Il y en a un dans `/etc/vim` qui s'appelle `vimrc`.

Copiez-le dans votre répertoire personnel en le faisant précéder d'un point (pour que ce soit un fichier caché) :

Code : Console

```
$ cp /etc/vim/vimrc ~/.vimrc
```

Ouvrez maintenant ce fichier... avec Vim bien sûr. 😊

Code : Console

```
$ vim .vimrc
```

Le début du fichier ressemble à ceci :

Code : Console

```
" All system-wide defaults are set in $VIMRUNTIME/debian.vim (usually just
" /usr/share/vim/vimcurrent/debian.vim) and sourced by the call to :runtime
" you can find below.  If you wish to change any of those settings, you should
" do it in this file (/etc/vim/vimrc), since debian.vim will be overwritten
" everytime an upgrade of the vim packages is performed.  It is recommended to
" make changes after sourcing debian.vim since it alters the value of the
" 'compatible' option.

" This line should not be removed as it ensures that various options are
" properly set to work with the Vim-related packages available in Debian.
runtime! debian.vim

" Uncomment the next line to make Vim more Vi-compatible
" NOTE: debian.vim sets 'nocompatible'.  Setting 'compatible' changes numerous
" options, so any other options should be set AFTER setting 'compatible'.
"set compatible

" Vim5 and later versions support syntax highlighting.  Uncommenting the next
" line enables syntax highlighting by default.
"syntax on

" If using a dark background within the editing area and syntax highlighting
" turn on this option as well
```

Les lignes commençant par " sont des commentaires. Je vous recommande de les lire, ils donnent des informations utiles.

Passons maintenant à l'activation de quelques commandes bien utiles. Je vous recommande de travailler avec le fichier de configuration `.vimrc` comme moi, et d'activer les options qui vous plaisent en décommentant les lignes concernées. Pour cela, la meilleure façon de procéder est de se mettre en mode interactif, de se déplacer avec `hjkl`, et d'appuyer sur `x` lorsque le curseur est sur un guillemet pour le supprimer et activer ainsi l'option. 😊

syntax : activer la coloration syntaxique

C'est clairement la première option à activer : la coloration syntaxique. En fonction du type de fichier que vous ouvrez, Vim colorera le texte.

Vim supporte un très très grand nombre de langages de programmation : C, C++, Python, Java, Ruby, Bash, Perl, etc.

Activez donc :

Code : Console

```
syntax on
```



Notez qu'il faut enregistrer, quitter et relancer Vim pour que le changement soit pris en compte... sauf bien sûr si vous activez l'option à la volée en tapant dans Vim : `set syntax=ON`

Aperçu du résultat :

```
" you can find below.  If you wish to change any of those settings, you should
" do it in this file (/etc/vim/vimrc), since debian.vim will be overwritten
" everytime an upgrade of the vim packages is performed.  It is recommended to
" make changes after sourcing debian.vim since it alters the value of the
" 'compatible' option.

" This line should not be removed as it ensures that various options are
" properly set to work with the Vim-related packages available in Debian.
runtime! debian.vim

" Uncomment the next line to make Vim more Vi-compatible
" NOTE: debian.vim sets 'nocompatible'.  Setting 'compatible' changes numerous
" options, so any other options should be set AFTER setting 'compatible'.
"set compatible

" Vim5 and later versions support syntax highlighting.  Uncommenting the next
" line enables syntax highlighting by default.
syntax on

" If using a dark background within the editing area and syntax highlighting
" turn on this option as well
"set background=dark

25,0-1 63
```

background : coloration sur un fond sombre

Par défaut, la coloration de Vim est plus adaptée aux fonds clairs. Les commentaires, par exemple, sont écrits en bleu foncé sur noir... pas très lisible.

Si votre console est sur un fond noir comme chez moi, je vous recommande d'activer la prochaine option background et de la mettre à "dark".

Code : Console

```
set background=dark
```

Les couleurs seront largement plus adaptées :

```
runtime! debian.vim

" Uncomment the next line to make Vim more Vi-compatible
" NOTE: debian.vim sets 'nocompatible'. Setting 'compatible' changes numerous
" options, so any other options should be set AFTER setting 'compatible'.
"set compatible

" Vim5 and later versions support syntax highlighting. Uncommenting the next
" line enables syntax highlighting by default.
syntax on

" If using a dark background within the editing area and syntax highlighting
" turn on this option as well
set background=dark

" Uncomment the following to have Vim jump to the last position when
" reopening a file
if has("autocmd")
  au BufReadPost * if line("\") > 0 && line("\") <= line("$")
    \ | exe "normal g'\|" | endif
endif

" Uncomment the following to have Vim load indentation rules according to the
28,1 30%
```

number : afficher les numéros de ligne

Il est possible d'afficher le numéro de chaque ligne à gauche :

Code : Console

```
set number
```

C'est assez pratique, notamment quand on programme.

Code : Console

```
set mouse=a
```

Désormais, vous pourrez cliquer avec la souris sur une lettre pour y déplacer le curseur directement. Vous pourrez utiliser la molette de la souris pour vous déplacer dans le fichier. 😊

Vous pourrez aussi sélectionner du texte avec la souris. Vous passerez alors en mode visuel.



Dans ce mode, vous pouvez supprimer le texte sélectionné (avec **x** comme d'habitude), mais aussi mettre le texte tout en majuscules (**U**), minuscules (**u**), etc.

Pour plus d'informations, lisez la [documentation du mode visuel de Vim](#).

Le surnom de Vim est "*Programmer's text editor*" (l'éditeur de texte du programmeur). On se rend compte à l'usage qu'il est en effet bien adapté à la programmation : coloration syntaxique poussée de très nombreux langages, affichage du numéro de ligne, mise en surbrillance de la parenthèse ou de l'accolade correspondante, etc. Il est même possible de compiler et lancer ses programmes directement depuis Vim !

Pourtant, il n'est pas facile à prendre en main. Ce tutoriel vous aura permis d'avoir une bonne petite introduction à Vim, afin que vous ne soyez pas perdus, mais retenez bien ceci : nous sommes très loin d'avoir vu toutes les possibilités du logiciel ! Sans plugins, on peut déjà faire des tonnes de choses, le tout étant de se renseigner et de lire la documentation.

N'hésitez pas à aller plus loin que ce chapitre. Même si sa lecture sera suffisante pour suivre la fin du cours, je vous conseille vraiment de parcourir un peu la doc et de demander des conseils. On peut faire des choses étonnantes.

Bien, maintenant que vous savez manier un éditeur de texte digne de ce nom, il est temps de passer aux choses sérieuses : la programmation ! Direction le prochain chapitre, on va faire des scripts bash ! 😊

Introduction aux scripts shell

Vous venez d'apprendre à utiliser un éditeur de texte puissant comme Vim. C'est très bien, cela va vous être utile.

Revenons maintenant dans le vif du sujet : **la programmation shell**. De quoi s'agit-il ?

Imaginez un *mini-langage* de programmation intégré à Linux. Ce n'est pas un langage aussi complet que peuvent l'être le C, le C++ ou le Java par exemple, mais cela permet d'automatiser la plupart de vos tâches. Voici un aperçu de ce qu'on peut faire avec :

- Sauvegarde de vos données
- Surveillance de la charge de votre machine
- Système de gestion personnalisé de vos téléchargements
- etc.

On aurait très bien pu faire tout cela en écrivant un programme en C par exemple. Le gros avantage des scripts shell, c'est que ce langage est totalement intégré à Linux : il n'y a rien à installer et rien à compiler. Et surtout : vous avez très peu de nouvelles choses à apprendre. En effet, toutes les commandes que l'on utilise dans les scripts shells sont des commandes du système que vous connaissez déjà : `ls`, `cut`, `grep`, `sort` ...

On parlera beaucoup de shell dans cette section. De quoi s'agit-il exactement ? Nous répondrons à cette question en premier. Ensuite, nous réaliserons notre tout premier script shell qui affiche un message à l'écran... et nous pourrons alors passer aux choses sérieuses dès le chapitre suivant !

Qu'est-ce qu'un shell ?

Dès le début de ce cours, j'ai fait la distinction entre les 2 environnements très différents disponibles sous Linux :

- L'environnement console
- L'environnement graphique

La plupart du temps sur sa machine, on a tendance à utiliser l'environnement graphique, ce qui est plus intuitif. Cependant, la console est aussi un allié très puissant qui permet d'effectuer des actions qui sont habituellement difficiles à réaliser en environnement graphique.

Je vous avais dit qu'il y avait plusieurs environnements graphiques disponibles (Gnome, KDE, XFCE...) mais qu'il n'y avait qu'une seule console. J'ai menti. 🤖

Il existe plusieurs environnements console : les shell

La différence est moins tape-à-l'oeil que dans le mode graphique (où on voit de suite que les menus ne sont pas à la même place par exemple).

La console a toujours un fond noir et un texte blanc, je vous rassure (quoique ça se personnalise ça). En revanche, les fonctionnalités offertes par l'invite de commande peuvent varier en fonction du **shell** que l'on utilise.



Les différents environnements console sont appelés des shell, c'est ça ?

C'est ça en effet. Voici les noms de quelques-uns des principaux shell qui existent :

- **sh** : Bourne Shell. L'ancêtre de tous les shells.
- **bash** : Bourne Again Shell. Une amélioration du Bourne Shell, disponible par défaut sous Linux et Mac OS X.
- **ksh** : Korn Shell. Un shell puissant assez présent sur les Unix propriétaires, mais aussi disponible en version libre, compatible avec bash.
- **csh** : C Shell. Un shell utilisant une syntaxe proche du langage C.
- **tcsh** : Tenex C Shell. Amélioration du C Shell.
- **zsh** : Z Shell. Shell assez récent reprenant les meilleures idées de bash, ksh et tcsh.

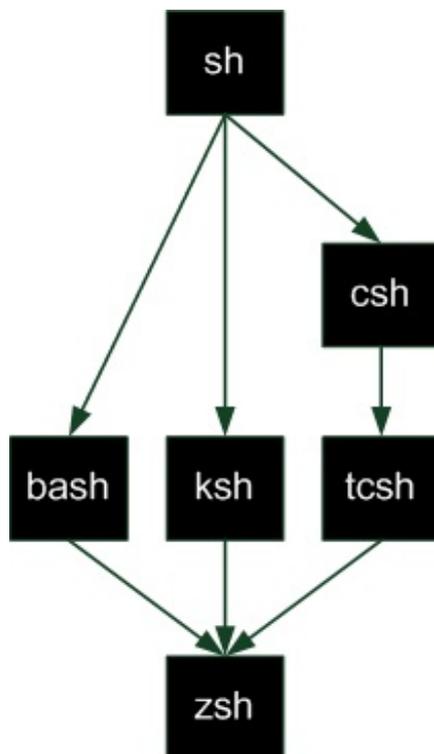
Il y a quelques autres mais vous avez là les principaux. 😊

Que faut-il savoir ? Tout d'abord que l'ancêtre de tous les shell est le sh (Bourne Shell). C'est le plus vieux. Il est installé sur tous les OS basés sur Unix. Il est néanmoins pauvre en fonctionnalités par rapport aux autres shells.

Le bash (Bourne Again Shell) est le shell par défaut de la plupart des distributions Linux, mais aussi celui du terminal de Mac OS X. Il y a fort à parier que c'est celui que vous utilisez en ce moment sous Linux.

Le bash est une amélioration du sh.

Voici dans les grandes lignes comment ont évolué les shells. Chacun hérite de la plupart des fonctionnalités de son ancêtre :

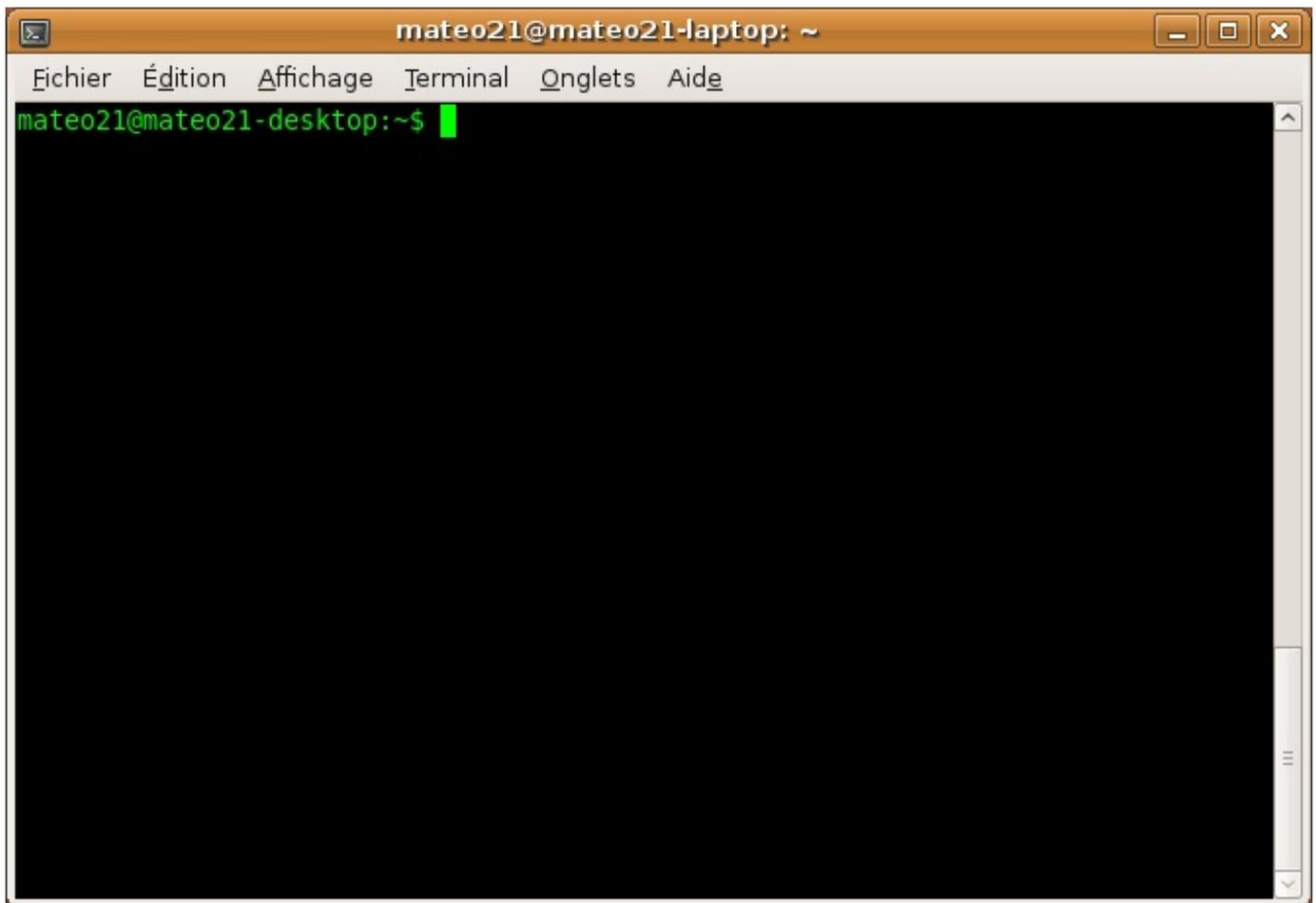


A quoi peut bien servir le sh aujourd'hui alors, si bash est par défaut sous Linux ?

Sh reste toujours plus répandu que bash. En fait, vous pouvez être sûr que tous les OS basés sur Unix possèdent sh, mais ils n'ont pas tous forcément bash. D'autres OS basés sur Unix, notamment les OS propriétaires (AIX et Solaris...), utilisent souvent d'autres types de shell. Le ksh par exemple y est très répandu.

A quoi sert un shell ?

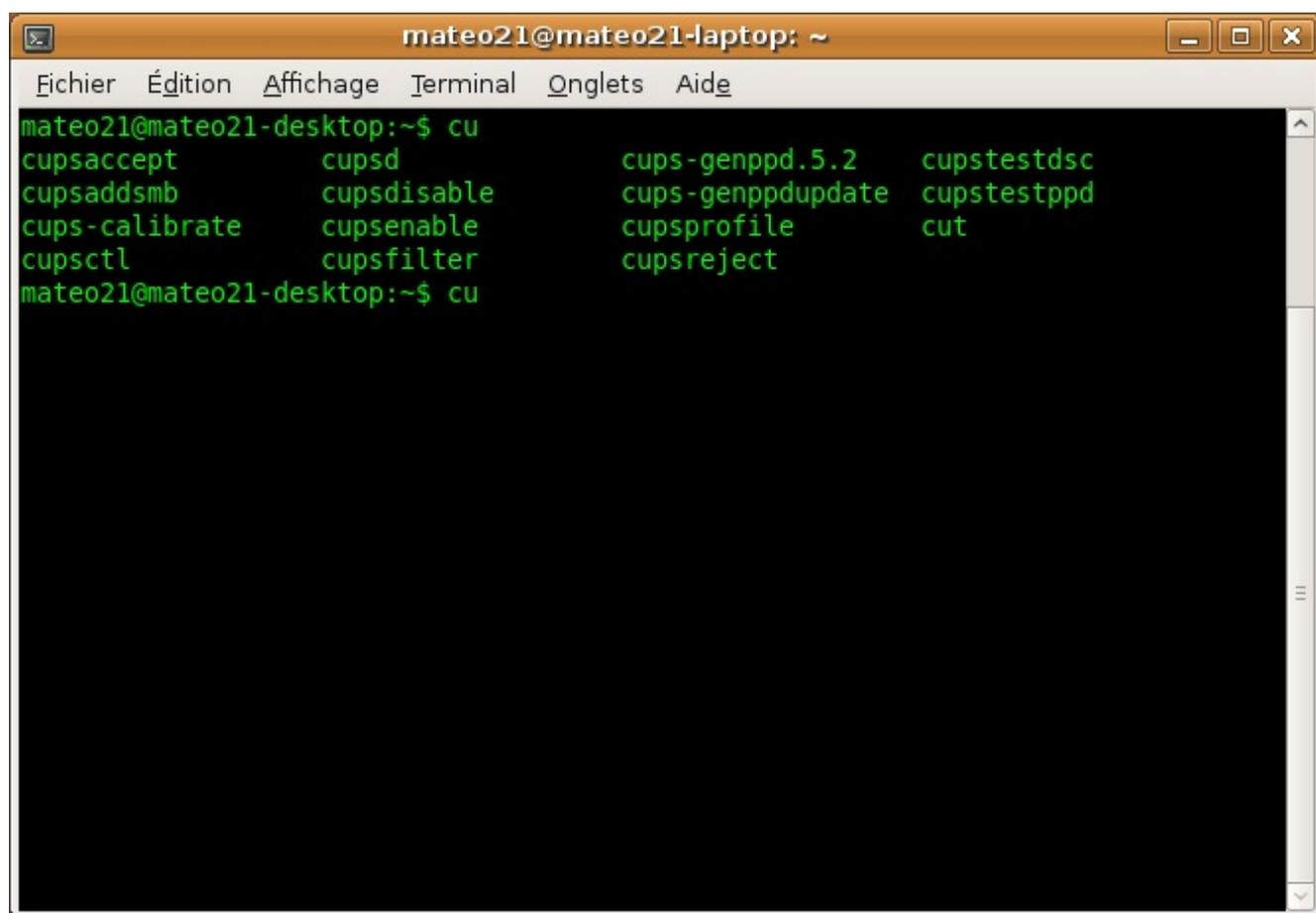
Le shell est le programme qui gère l'invite de commandes. C'est donc le programme qui attend que vous rentriez des commandes :



Le shell attend que vous rentriez une commande

C'est aussi le programme qui est capable par exemple de :

- Se souvenir quelles étaient les dernières commandes tapées (vous remontez dans votre historique en appuyant sur la flèche vers le haut ou en faisant une recherche avec un Ctrl + R).
- Auto-compléter une commande ou un nom de fichier lorsque vous appuyez sur Tab :

A screenshot of a terminal window titled 'mateo21@mateo21-laptop: ~'. The window has a menu bar with 'Fichier', 'Édition', 'Affichage', 'Terminal', 'Onglets', and 'Aide'. The terminal shows the user typing 'cu' at the prompt 'mateo21@mateo21-desktop:~\$'. The shell auto-completes the command, displaying a list of files and directories: 'cupsaccept', 'cupsd', 'cups-genppd.5.2', 'cupstestdsc', 'cupsaddsmb', 'cupsdisable', 'cups-genppdupdate', 'cupstestppd', 'cups-calibrate', 'cupsenable', 'cupsprofile', 'cut', 'cupstcl', 'cupsfilter', and 'cupsreject'. The user then types 'cu' again, and the prompt returns to 'mateo21@mateo21-desktop:~\$'.

Le shell auto-complète les commandes et les noms de fichiers

- Gérer les processus (envoi en arrière-plan, mise en pause avec Ctrl + Z...).
- Rediriger et chaîner les commandes (les fameux symboles >, <, |...).
- Définir des alias (par exemple `ll` signifie chez moi `ls -lArth`).

Bref, le shell fournit toutes les fonctionnalités de base pour pouvoir lancer des commandes.

Souvenez-vous, nous avons modifié un fichier `.bashrc` dans un des premiers chapitres (celui où nous avons [appris à utiliser nano](#)). Le `bashrc` est le fichier de configuration du bash, que Linux vous fait utiliser par défaut. Chaque personne peut avoir son `bashrc` pour personnaliser son invite de commande, ses alias, etc.

Installer un nouveau shell

Pour le moment, vous devriez avoir `sh` et `bash` installés sur votre système. Si vous voulez essayer un autre shell comme `ksh`, vous pouvez le télécharger comme n'importe quel paquet :

Code : Console

```
# apt-get install ksh
```

... pour installer `ksh` par exemple.

Une fois installé, il faut demander à l'utiliser pour votre compte utilisateur. Tapez pour cela :

Code : Console

```
$ chsh
```

`chsh` signifie "Change Shell".

On vous demandera où se trouve le programme qui gère le shell. Vous devriez indiquer `/bin/ksh` pour `ksh`, `/bin/sh` pour `sh`, `/bin/bash` pour `bash`, etc.

Quelle importance a tout ceci lorsqu'on réalise un script shell ?

Si je vous parle de cela, c'est parce qu'un script shell *dépend* d'un shell précis. En gros, le langage n'est pas tout à fait le même selon si vous utilisez `sh`, `bash`, `ksh`, etc.

Il est possible d'écrire des scripts `sh` par exemple. Ceux-là, on est sûr qu'ils fonctionnent partout car tout le monde possède un shell "`sh`".

Toutefois, c'est le plus vieux shell, et écrire des scripts en `sh` est possible mais pas franchement facile ou ergonomique. 😞



Avec quel shell va-t-on écrire nos scripts alors ?

Je propose d'étudier le `bash` dans ce cours car :

- On le trouve par défaut sous Linux et Mac OS X (cela couvre assez de monde !).
- Il rend l'écriture de scripts plus simple que `sh`.
- Il est plus répandu que `ksh` et `zsh` sous Linux.

En clair, le `bash` est un bon compromis entre `sh` (le plus compatible) et `ksh/zsh` (plus puissants).

Notre premier script

Nous allons commencer par écrire un premier script `bash` tout simple. Il ne sera pas révolutionnaire, mais il va nous permettre de voir les bases de la création d'un script et comment celui-ci s'exécute. Cela vous sera donc essentiel pour la suite.

Création du fichier

Commençons par créer un nouveau fichier pour notre script. Le plus simple est d'ouvrir `vim` en lui donnant le nom du nouveau fichier à créer :

Code : Console

```
$ vim essai.sh
```

Si `essai.sh` n'existe pas, il sera créé (ce qui sera le cas ici).



J'ai donné ici l'extension "`.sh`" à mon fichier. On le fait souvent par convention pour indiquer que c'est un script shell, mais ce n'est pas une obligation. Certains scripts shell n'ont d'ailleurs pas d'extension du tout (j'aurais pu appeler mon script "`essai`" tout court donc).

Indiquer le nom du shell utilisé par le script

`Vim` est maintenant ouvert et vous avez un fichier vide sous les yeux.

La première chose à faire dans un script shell est d'indiquer... quel shell est utilisé. En effet, comme je vous l'ai dit plus tôt, la syntaxe du langage change un peu selon si on utilise "sh", "bash", "ksh", etc.

En ce qui nous concerne, nous souhaitons utiliser la syntaxe de bash qui est le plus répandu sous Linux et qui est plus complet que sh. Nous indiquons où se trouve le programme bash :

Code : Bash

```
#!/bin/bash
```



Le #! est appelé le *sha-bang*.
/bin/bash peut être remplacé par /bin/sh si vous souhaitez coder pour sh, /bin/ksh pour ksh, etc.

Bien que non indispensable, cette ligne permet de s'assurer que le script est bien exécuté avec le bon shell. En l'absence de cette ligne, c'est le shell de l'utilisateur qui sera chargé. Cela pose un problème : si votre script est écrit pour bash et que la personne qui l'exécute utilise ksh, il y a de fortes chances que le script ne fonctionne pas correctement !

La ligne du sha-bang permet donc de "charger" le bon shell avant l'exécution du script. Vous devrez la mettre au tout début de chacun de vos scripts à partir de maintenant.

Exécution de commandes

Après le sha-bang, nous pouvons commencer à coder.

Le principe est très simple : il vous suffit d'écrire les commandes que vous souhaitez exécuter. Ce sont les mêmes que celles que vous tapez dans l'invite de commandes !

- **ls** : pour lister les fichiers du répertoire
- **cd** : pour changer de répertoire
- **mkdir** : pour créer un répertoire
- **grep** : pour rechercher un mot
- **sort** : pour trier des mots
- etc.

Bref, tout ce que vous avez appris, vous pouvez le réutiliser ici ! 😊

Allez, on va commencer par quelque chose de très simple : un ls. On va donc créer un script bash qui va juste se contenter d'afficher le contenu du dossier courant :

Code : Bash

```
#!/bin/bash  
  
ls
```

C'est tout !

Les commentaires

Notez que vous pouvez aussi ajouter des commentaires dans votre script. Les commentaires sont des lignes qui ne seront pas exécutées mais qui permettent d'expliquer ce que fait votre script.

Tous les commentaires commencent par un #. Par exemple :

Code : Bash

```
#!/bin/bash

# Affichage de la liste des fichiers
ls
```



Eh, mais la ligne du sha-bang commence aussi par un #... C'est un commentaire ?

Oui, c'est un commentaire aussi, mais considérez que c'est un commentaire "spécial".

Exécuter le script bash

Nous avons écrit un petit script sans prétention de 2-3 lignes. Notre mission maintenant est d'arriver à l'exécuter.

Commencez par enregistrer votre fichier et à fermer votre éditeur (sous Vim, il suffit de taper ":wq" ou encore ":x"). Vous retrouvez alors l'invite de commandes.

Donner les droits d'exécution au script

Si vous faites un `ls -l` pour voir votre fichier qui vient d'être créé, vous obtenez ça :

Code : Console

```
$ ls -l
total 4
-rw-r--r-- 1 mateo21 mateo21 17 2009-03-13 14:33 essai.sh
```

Ce qui nous intéresse ici, ce sont les droits sur le fichier : `-rw-r--r--`

Si vous vous souvenez un petit peu de [ce cours](#) sur les droits, vous devriez vous rendre compte que notre script peut être lu par tout le monde (r), écrit uniquement par nous (w), et n'est pas exécutable (pas de x).

Or, pour exécuter un script, il faut que le fichier ait le droit "exécutable". Le plus simple pour donner ce droit est d'écrire :

Code : Console

```
$ chmod +x essai.sh
```

Vous pouvez vérifier que le droit a bien été donné :

Code : Console

```
$ ls -l
total 4
-rwxr-xr-x 1 mateo21 mateo21 17 2009-03-13 14:33 essai.sh
```

Tout le monde a maintenant le droit d'exécuter le script. Si vous voulez, vous pouvez limiter ce droit à vous-même, mais pour cela je vous invite à revoir le cours sur les droits, je ne vais pas me répéter. 😊

Exécution du script

Le script s'exécute maintenant comme n'importe quel programme, en tapant `./` devant le nom du script :

Code : Console

```
$ ./essai.sh
essai.sh
```

Que fait le script ? Il fait juste un `ls`, donc il affiche la liste des fichiers dans le répertoire (ici, il y avait seulement `essai.sh` dans mon répertoire)

Bien entendu, ce script est inutile. Il était plus simple de taper `ls` directement. Cependant, vous vous en doutez, on va pouvoir faire beaucoup mieux que ça dans les prochains chapitres. 😊

Vous pouvez déjà modifier votre script pour qu'il vous donne auparavant aussi le nom du répertoire dans lequel vous vous trouvez :

Code : Bash

```
#!/bin/bash

pwd
ls
```

Les commandes seront exécutées l'une après l'autre :

Code : Console

```
$ ./essai.sh
/home/mateo21/scripts
essai.sh
```

Exécution de débogage

Plus tard, vous ferez probablement de gros scripts et vous risquez de rencontrer des bugs. Il faut donc dès à présent que vous sachiez comment déboguer un script.

Il faut l'exécuter comme ceci :

Code : Console

```
$ bash -x essai.sh
```

On appelle en fait directement le programme `bash` et on lui ajoute en paramètre un `-x` (pour lancer le mode débogage) et le nom de notre script à déboguer.

Le shell affiche alors le détail de l'exécution de votre script, ce qui peut vous aider à retrouver la cause de vos erreurs :

Code : Console

```
$ bash -x essai.sh
+ pwd
/home/mateo21/scripts
+ ls
essai.sh
```

Créer sa propre commande

Actuellement, le script doit être lancé via `./essai.sh` et vous devez être dans le bon répertoire (ou alors vous devez taper le chemin en entier, comme `/home/mateo21/scripts/essai.sh`).



Comment font les autres programmes pour pouvoir être exécutés depuis n'importe quel répertoire sans `./` devant ?

Ils sont placés dans un des répertoires du PATH. Le PATH est une variable système qui indique où sont les programmes exécutables sur votre ordinateur. Si vous tapez `echo $PATH`, vous aurez la liste de ces répertoires "spéciaux".

Il vous suffit donc de déplacer ou copier votre script dans un de ces répertoires, comme `/bin`, ou `/usr/bin`, ou `/usr/local/bin` (ou un autre répertoire du PATH). Notez qu'il faut être root pour pouvoir faire ça.

Une fois que c'est fait, vous pourrez alors taper simplement `essai.sh` pour exécuter votre programme, et ce quel que soit le répertoire dans lequel vous vous trouvez !

Code : Console

```
$ essai.sh
/home/mateo21/scripts
essai.sh
```

Vous connaissez désormais les bases du shell scripting. 😊

Nous n'avons pas fait grand-chose de passionnant pour le moment, mais cela va changer, et ce dès le prochain chapitre. Au menu de ce qui vous attend : création et manipulation de variables, affichage de texte, saisie au clavier, gestion des paramètres du programme et bien d'autres choses !

Afficher et manipuler des variables

Résumons ce que nous savons :

- Le **bash** est le programme qui gère l'invite de commande de la console sous Linux. C'est aussi un langage de programmation. On parle de "shell".
- D'autres shells existent, tels sh, ksh, zsh... Bash sera utilisé dans ce cours car c'est le **shell par défaut** sous la plupart des distributions Linux.
- **Toutes les commandes** que vous avez appris à taper dans la console depuis le début du cours peuvent être utilisées pour réaliser des scripts shell : grep, cut, ls, pwd... Les scripts shell ne sont donc qu'un assemblage de ces commandes !

Comme dans tous les langages de programmation, on trouve en bash ce qu'on appelle des **variables**. Elles nous permettent de stocker temporairement des informations en mémoire. C'est en fait la base de la programmation.

Les variables en bash sont assez particulières. Il faut être très rigoureux lorsqu'on les utilise. Si vous avez fait du C ou d'autres langages de programmation, vous allez être un peu surpris de leur mode de fonctionnement, donc soyez attentifs. Et si vous n'avez jamais programmé, soyez attentifs aussi. 😊

Déclarer une variable

Nous allons créer un nouveau script que nous appellerons variables.sh :

Code : Console

```
$ vim variables.sh
```

La première ligne de tous nos scripts doit indiquer quel shell est utilisé, comme nous l'avons appris tout à l'heure. Commencez donc par écrire :

Code : Bash

```
#!/bin/bash
```

Cela indique que nous allons programmer en bash.

Maintenant, définissons une variable. Toute variable a un nom et une valeur :

Code : Bash

```
message='Bonjour tout le monde'
```

Dans le cas présent :

- La variable a pour *nom* **message**
- ... et pour *valeur* **'Bonjour tout le monde'**



Ne mettez pas d'espaces autour du symbole égal "=" ! Le bash est très pointilleux sur de nombreux points, donc évitez de le vexer. 😊

Je vous signalerai systématiquement les pièges à éviter, car il y en a un certain nombre !

Si vous voulez insérer une apostrophe dans la valeur de la variable, il faut la faire précéder d'un antislash \. En effet, comme les apostrophes servent à délimiter le contenu, on est obligé d'utiliser un *caractère d'échappement* (comme on les appelle) pour pouvoir vraiment insérer une apostrophe :

Code : Bash

```
message='Bonjour c\'est moi'
```

Bien, reprenons notre script. Il devrait maintenant ressembler à cela :

Code : Bash

```
#!/bin/bash  
message='Bonjour tout le monde'
```

Exécutez-le pour voir ce qui se passe (après avoir modifié les droits pour le rendre exécutable bien sûr) :

Code : Console

```
$ ./variables.sh  
$
```

Il ne se passe rien !



Qu'est-ce que le script fait alors ?

Il met en mémoire le message *Bonjour tout le monde*, et c'est tout ! Rien ne s'affiche à l'écran !

Pour afficher une variable, il va falloir utiliser une commande dont je ne vous ai pas encore parlé...

echo : afficher une variable

Avant de commencer à parler de *variables*, il y a une commande que j'aimerais vous présenter : `echo`. J'aurais pu en parler avant qu'on commence à faire des scripts bash, mais vous n'en auriez pas vu l'utilité avant ce chapitre. 😞

Son principe est très simple : elle affiche dans la console le message demandé. Un exemple :

Code : Console

```
$ echo Salut tout le monde  
Salut tout le monde
```

Comme vous le voyez, c'est simple comme bonjour. Les guillemets ne sont pas requis.



Mais... comment ça marche ?

En fait, la commande `echo` affiche dans la console tous les paramètres qu'elle reçoit. Ici, on a envoyé 4 paramètres :

- Salut
- tout
- le
- monde

Chacun des mots était considéré comme un paramètre que echo a affiché. Si vous mettez des guillemets autour de votre message, celui-ci sera considéré comme un seul paramètre (le résultat sera visuellement le même) :

Code : Console

```
$ echo "Salut tout le monde"
Salut tout le monde
```

Si vous voulez insérer des retours à la ligne, il faudra activer le paramètre -e et utiliser le symbole \n :

Code : Console

```
$ echo -e "Message\nAutre ligne"
Message
Autre ligne
```

Afficher une variable

Pour afficher une variable, on va réutiliser son nom précédé du symbole dollar "\$" :

Code : Bash

```
#!/bin/bash

message='Bonjour tout le monde'
echo $message
```



Comparez les lignes 3 et 4 : lorsqu'on *déclare* la variable à la ligne 3, on ne doit pas mettre de \$ devant. En revanche, lorsqu'on *l'affiche* à la ligne 4, on doit cette fois mettre un \$!

Résultat :

Code : Console

```
Bonjour tout le monde
```

Maintenant, supposons que l'on veuille afficher du texte et la variable à la fois. On serait tenté d'écrire :

Code : Bash

```
#!/bin/bash

message='Bonjour tout le monde'
```

```
echo 'Le message est : $message'
```

Problème, ça ne marche pas comme on veut car ça affiche :

Code : Console

```
Le message est : $message
```

Pour bien comprendre ce qui se passe, intéressons-nous au fonctionnement de ce qu'on appelle les "quotes".

Les quotes

Il est possible d'utiliser des *quotes* pour délimiter un paramètre qui contient des espaces. Il existe 3 types de quotes :

- Les apostrophes '' (simples quotes)
- Les guillemets "" (doubles quotes)
- Les accents graves `` (back quotes), qui s'insèrent avec **Alt Gr + 7** sur un clavier AZERTY français

Selon le type de quote que vous utilisez, la réaction de Bash ne sera pas la même.

Les simples quotes ''

Commençons par les simples quotes :

Code : Bash

```
message='Bonjour tout le monde'  
echo 'Le message est : $message'
```

Code : Console

```
Le message est : $message
```

Avec des simples quotes, la variable n'est pas analysée et le \$ est affiché tel quel.

Les doubles quotes ""

Avec des doubles quotes :

Code : Bash

```
message='Bonjour tout le monde'  
echo "Le message est : $message"
```

Code : Console

```
Le message est : Bonjour tout le monde
```

... ça marche ! Cette fois, la variable est analysée et son contenu est affiché.

En fait, les doubles quotes demandent à bash d'analyser le contenu du message. S'il trouve des symboles spéciaux (comme des variables), il les interprète.

Avec des simples quotes, le contenu était affiché tel quel.

Les back quotes ``

Un peu particulières, les back quotes demandent à bash d'**exécuter** ce qui se trouve à l'intérieur.

Un exemple vaut mieux qu'un long discours, regardez la première ligne :

Code : Bash

```
message=`pwd`  
echo "Vous êtes dans le dossier $message"
```

Code : Console

```
Vous êtes dans le dossier /home/mateo21/bin
```

La commande pwd a été exécutée, et son contenu a été inséré dans la variable message ! On a ensuite affiché le contenu de la variable.

Ca peut paraître un peu tordu, mais c'est réellement utile. Nous nous en resservons dans les chapitres suivants.

read : demander une saisie

Vous pouvez demander à l'utilisateur de saisir du texte avec la commande **read**. Ce texte sera immédiatement stocké dans une variable.

La commande read propose plusieurs options intéressantes. La façon la plus simple de l'utiliser est d'indiquer le nom de la variable dans laquelle le message saisi sera stocké :

Code : Bash

```
read nomvariable
```

Adaptons notre script pour qu'il nous demande notre nom et nous l'affiche :

Code : Bash

```
#!/bin/bash  
  
read nom  
echo "Bonjour $nom !"
```

Lorsque vous lancez ce script, rien ne s'affiche, mais vous pouvez taper du texte (votre nom par exemple 🤖) :

Code : Console

```
Mathieu  
Bonjour Mathieu !
```

Notez que la première ligne correspond au texte que j'ai tapé au clavier.

Affecter simultanément une valeur à plusieurs variables

On peut demander de saisir autant de variables d'affilée que l'on souhaite. Voici un exemple de ce qu'il est possible de faire :

Code : Bash

```
#!/bin/bash  
  
read nom prenom  
echo "Bonjour $nom $prenom !"
```

Code : Console

```
Deschamps Mathieu  
Bonjour Deschamps Mathieu !
```

 `read` lit ce que vous tapez mot par mot (en considérant que les mots sont séparés par des espaces 😊). Il assigne chaque mot à une variable différente, d'où le fait que le nom et le prénom ont été correctement assignés à `$nom` et `$prenom` respectivement.

Si vous rentrez plus de mots au clavier que vous n'avez prévu de variables pour en stocker, la dernière variable de la liste récupérera tous les mots restants. En clair, si j'avais tapé pour le programme précédent "Nebra Mathieu Cyril", la variable `$prenom` aurait eu pour valeur "Mathieu Cyril".

-p : afficher un message de prompt

Bon, notre programme n'est pas très clair, on devrait afficher un message pour que l'utilisateur sache quoi faire. Avec l'option `-p` de `read`, vous pouvez faire cela :

Code : Bash

```
#!/bin/bash  
  
read -p 'Entrez votre nom : ' nom  
echo "Bonjour $nom !"
```



Notez que le message 'Entrez votre nom' a été entouré de quotes. Si on ne l'avait pas fait, le bash aurait considéré que chaque mot était un paramètre différent !

Résultat :

Code : Console

```
Entrez votre nom : Mathieu
```

```
Bonjour Mathieu !
```

C'est mieux! 😊

-n : limiter le nombre de caractères

Avec -n, vous pouvez couper au besoin au bout de X caractères si vous ne voulez pas que l'utilisateur rentre un message trop long.

Exemple :

Code : Bash

```
#!/bin/bash

read -p 'Entrez votre login (5 caractères max) : ' -n 5 nom
echo "Bonjour $nom !"
```

Code : Console

```
Entrez votre login (5 caractères max) : mathiBonjour mathi !
```

Notez que le bash coupe automatiquement au bout de 5 caractères, sans que vous ayez besoin d'appuyer sur la touche Entrée. Ce n'est pas très esthétique du coup, parce que le message s'affiche sur la même ligne. Pour éviter cela, vous pouvez faire un echo avec des \n comme vous avez appris à le faire plus tôt :

Code : Bash

```
#!/bin/bash

read -p 'Entrez votre login (5 caractères max) : ' -n 5 nom
echo -e "\nBonjour $nom !"
```

Code : Console

```
Entrez votre login (5 caractères max) : mathi
Bonjour mathi !
```

-t : limiter le temps autorisé pour saisir un message

Vous pouvez définir un "timeout" avec -t, c'est-à-dire un nombre de secondes au bout desquelles le read s'arrêtera.

Code : Bash

```
#!/bin/bash

read -p 'Entrez le code de désamorçage de la bombe (vous avez 5 secondes) : ' -t 5 code
```

```
echo -e "\nBoum !"
```

-s : ne pas afficher le texte saisi

Plus utile probablement, le paramètre `-s` masque les caractères que vous saisissez. Cela vous servira notamment si vous voulez que l'utilisateur rentre un mot de passe :

Code : Bash

```
#!/bin/bash

read -p 'Entrez votre mot de passe : ' -s pass
echo -e "\nMerci ! Je vais dire à tout le monde que votre mot de
passe est $pass ! :-)"
```

Code : Console

```
Entrez votre mot de passe :
Merci ! Je vais dire à tout le monde que votre mot de passe est supertopsecret38 !
)
```

Comme vous pouvez le constater, le mot de passe que j'ai rentré ne s'affiche pas lors de l'instruction `read`.

Effectuer des opérations mathématiques

En bash, les variables sont toutes des chaînes de caractères. En soi, le bash n'est donc pas vraiment capable de manipuler des nombres, et donc il n'est pas capable d'effectuer des opérations.

Heureusement, on peut passer par des commandes (eh oui encore). Ici, la commande à connaître est `let`.

Code : Bash

```
let "a = 5"
let "b = 2"
let "c = a + b"
```

A la fin de ce script, la variable `$c` vaudra 7. Testons :

Code : Bash

```
#!/bin/bash

let "a = 5"
let "b = 2"
let "c = a + b"
echo $c
```

Code : Console

```
7
```

Les opérations utilisables sont :

- L'addition : +
- La soustraction : -
- La multiplication : *
- La division : /
- La puissance : **
- Le modulo : % (renvoie le reste de la division)

Quelques exemples :

Code : Bash

```
let "a = 5 * 3" # $a = 15
let "a = 4 ** 2" # $a = 16 (4 au carré)
let "a = 8 / 2" # $a = 4
let "a = 10 / 3" # $a = 3
let "a = 10 % 3" # $a = 1
```

Une petite explication pour les 2 dernières lignes :

- $10 / 3 = 3$ car la division est entière (la commande ne renvoie pas de nombres décimaux)
- $10 \% 3$ renvoie 1 car le reste de la division de 10 par 3 est 1. En effet, 3 "rentre" 3 fois dans 10 (ça fait 9), et il reste 1 pour aller à 10.

Notez qu'il est possible aussi de contracter les commandes, comme cela se fait en langage C.

Ainsi :

Code : Bash

```
let "a = a * 3"
```

... est équivalent à écrire :

Code : Bash

```
let "a *= 3"
```



Actuellement, les résultats renvoyés sont des nombres entiers et non des nombres décimaux. Si vous voulez travailler avec des nombres décimaux, renseignez-vous sur le fonctionnement de la commande `bc`.

Les variables d'environnement

Actuellement, les variables que vous créez dans vos scripts bash n'existent que dans ces scripts. En clair, une variable définie dans un programme A ne sera pas utilisable dans un programme B.

Les variables d'environnement sont des variables que l'on peut utiliser dans n'importe quel programme. On parle aussi parfois de **variables globales**. Vous pouvez afficher toutes celles que vous avez actuellement en mémoire avec la commande `env` :

Code : Console

```
$ env
ORBIT_SOCKETDIR=/tmp/orbit-mateo21
GLADE_PIXMAP_PATH=/usr/share/glade3/pixmaps
TERM=xterm
SHELL=/bin/bash
GTK_MODULES=canberra-gtk-module
USER=mateo21
PATH=/home/mateo21/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
GDM_XSERVER_LOCATION=local
PWD=/home/mateo21/bin
EDITOR=nano
SHLVL=1
HOME=/home/mateo21
OLDPWD=/home/mateo21

[ ... ]
```

Il y en a beaucoup. Certaines sont très utiles, d'autres moins. Parmi celles que je peux vous commenter et qui peuvent s'avérer utiles :

- **SHELL** : indique quel type de shell est en cours d'utilisation (sh, bash, ksh...)
- **PATH** : une liste des répertoires qui contiennent des exécutable que vous souhaitez pouvoir lancer sans indiquer leur répertoire. On en a parlé un peu plus tôt. Si un programme se trouve dans un de ces dossiers, vous pourrez l'invoquer quel que soit le dossier où vous vous trouvez.
- **EDITOR** : l'éditeur de texte par défaut qui s'ouvre lorsque cela est nécessaire.
- **HOME** : la position de votre dossier home.
- **PWD** : le dossier dans lequel vous vous trouvez.
- **OLDPWD** : le dossier dans lequel vous vous trouviez auparavant.



Notez que les noms de ces variables sont, par convention, écrits en majuscules.

Comment utiliser ces variables dans vos scripts ? C'est très simple, il suffit de les appeler par leur nom !

Exemple :

Code : Bash

```
#!/bin/bash
echo "Votre éditeur par défaut est $EDITOR"
```

Code : Console

```
Votre éditeur par défaut est nano
```



Plus rarement, vous pourriez avoir besoin de définir votre propre variable d'environnement. Pour cela, on utilise la commande **export** que vous avez pu voir dans votre `.bashrc`.

Note un peu technique : un script lancé comme on le fait depuis le terminal ne peut modifier une variable d'environnement pour l'ensemble du système, car c'est un *processus enfant*. Un processus enfant ne peut pas modifier les variables d'environnement pour ses parents. Je vous laisse vous renseigner (et méditer) là-dessus si vous avez

vraiment besoin de le faire. N'hésitez pas à poser des questions sur les forums au besoin.

Les variables des paramètres

Comme toutes les commandes, vos scripts bash peuvent eux aussi accepter des paramètres. Ainsi, on pourrait appeler notre script comme ceci :

Code : Console

```
./variables.sh param1 param2 param3
```

Le problème, c'est que nous n'avons toujours pas vu comment récupérer ces paramètres dans notre script. Pourtant, c'est très simple à faire !

En effet, des variables sont automatiquement créées :

- **\$#** : contient le nombre de paramètres
- **\$0** : contient le nom du script exécuté (ici `./variables.sh`)
- **\$1** : contient le premier paramètre
- **\$2** : contient le second paramètre
- ...
- **\$9** : contient le 9ème paramètre

Essayons :

Code : Bash

```
#!/bin/bash

echo "Vous avez lancé $0, il y a $# paramètres"
echo "Le paramètre 1 est $1"
```

Code : Console

```
$ ./variables.sh param1 param2 param3
Vous avez lancé ./variables.sh, il y a 3 paramètres
Le paramètre 1 est param1
```



Et si on utilise plus de 9 paramètres ? J'ai cru voir que les variables s'arrêtaient à \$9... 🤔

Là, ça va un peu loin, mais ça peut arriver. On peut imaginer un script qui accepte une liste de fichiers en paramètres. Rien ne nous empêcherait de lui envoyer 15 paramètres dans ce cas :

Code : Console

```
./script.sh fichier1 fichier2 fichier3 fichier4 ... fichier14 fichier15
```

Pour traiter autant de paramètres, on les fera en général un par un.. On peut "décaler" les paramètres dans les variables \$1 \$2 etc. avec la commande **shift**.

Reprenons notre script :

Code : Bash

```
#!/bin/bash

echo "Le paramètre 1 est $1"
shift
echo "Le paramètre 1 est maintenant $1"
```

Code : Console

```
$ ./variables.sh param1 param2 param3
Le paramètre 1 est param1
Le paramètre 1 est maintenant param2
```

Comme vous le voyez, les paramètres ont été décalés : \$1 correspond après le shift au second paramètre, \$2 au troisième paramètre, etc.

Bien sûr, shift est généralement utilisé dans une boucle qui permet de traiter les paramètres un par un. On verra d'ailleurs comment faire des boucles dans peu de temps. 😊

Les tableaux

Le bash gère aussi les variables "tableaux". Ce sont des variables qui contiennent plusieurs cases, comme un tableau. Vous en aurez probablement besoin un jour, voyons comment cela fonctionne.

Pour définir un tableau, on peut faire comme ceci :

Code : Bash

```
tableau=('valeur0' 'valeur1' 'valeur2')
```

Cela crée une variable tableau qui contient 3 valeurs (valeur0, valeur1, valeur2).

Pour accéder à une case du tableau, on doit utiliser la syntaxe suivante :

Code : Bash

```
${tableau[2]}
```

... ceci affichera le contenu de la case n°2 (donc "valeur2").



Les cases sont numérotées à partir de 0 ! La première case a donc le numéro 0.

Notez par ailleurs que pour afficher le contenu d'une case du tableau, vous devez entourer votre variable d'accolades comme je l'ai fait pour `${tableau[2]}`.

Vous pouvez aussi manuellement définir le contenu d'une case :

Code : Bash

```
tableau[2]='valeur2'
```

Essayons tout ceci dans un script :

Code : Bash

```
#!/bin/bash

tableau=('valeur0' 'valeur1' 'valeur2')
tableau[5]='valeur5'
echo ${tableau[1]}
```

A votre avis, que va afficher ce script ?

Réponse :

Code : Console

```
valeur1
```



Comme vous pouvez le constater, le tableau peut avoir autant de cases que vous le désirez. La numérotation n'a pas besoin d'être continue, vous pouvez sauter des cases sans problème (la preuve, il n'y a pas de case n°3 ni de case n°4 dans mon script précédent).

Vous pouvez afficher l'ensemble du contenu du tableau d'un seul coup en utilisant `${tableau[*]}` :

Code : Bash

```
#!/bin/bash

tableau=('valeur0' 'valeur1' 'valeur2')
tableau[5]='valeur5'
echo ${tableau[*]}
```

Code : Console

```
valeur0 valeur1 valeur2 valeur5
```

Le bash est un langage qui propose beaucoup plus de possibilités qu'il n'en a l'air. Sa syntaxe, parfois étonnante, nécessite d'être très précis et pointilleux. Un oubli d'un caractère et plus rien ne marche. 😞

J'ai essayé de faire dans ce chapitre un bon tour d'horizon de ce qu'il est possible de faire avec les variables en bash, mais nous n'avons pas tout vu. Il y a de nombreuses autres possibilités, parfois plus rarement utilisées, que j'ai passées sous silence. Il est par exemple possible de :

- Calculer la taille d'une chaîne de caractères : `${#var}`
- Extraire des caractères précis d'une chaîne : `${var:3:2}`
- Récupérer le code de retour de la dernière commande qui a été lancée : `$?`
- etc.

N'hésitez pas à faire des recherches sur bash pour en savoir plus si vous en avez besoin un jour.

Les conditions

La prise de décision est un élément indispensable dans tout programme. Si on ne pouvait pas décider quoi faire, le programme ferait toujours la même chose... ce qui serait bien ennuyeux. 🤖

Les **branchements conditionnels** (que nous abrègerons "conditions") sont un moyen de dire dans notre script "*SI cette variable vaut tant, alors fais ceci, sinon fais cela*".

Si vous avez déjà vu un autre langage de programmation, cela doit vous être familier. Sinon, ne vous en faites pas, vous allez très vite comprendre le concept. 😊

if : la condition la plus simple

Le type de condition le plus courant est le **if**, qui signifie "si".

Si

Les conditions ont la forme suivante :

Citation

```
SI test_de_variable
ALORS
-----> effectuer_une_action
FIN SI
```

Bien entendu, ce n'est pas du bash ça. C'est juste un schéma pour vous montrer quelle est la forme d'une condition.

La syntaxe en bash est la suivante :

Code : Bash

```
if [ test ]
then
echo "C'est vrai"
fi
```

Le mot "fi" (if à l'envers !) à la fin indique que le if s'arrête là. Tout ce qui est entre le then et le fi sera exécuté uniquement si le test est vérifié.



Vous noterez, c'est très important, qu'il y a des espaces à l'intérieur des crochets. On ne doit pas écrire [test] mais [test] !

Il y a une autre façon d'écrire le if : en plaçant le then sur la même ligne. Dans ce cas, il ne faut pas oublier de rajouter un point-virgule après les crochets :

Code : Bash

```
if [ test ]; then
echo "C'est vrai"
fi
```

A la place du mot "test", il faut indiquer votre test. C'est à cet endroit que vous testerez la valeur d'une variable par exemple. Nous allons voir ici un cas simple où nous testons la valeur d'une chaîne de caractères, puis nous apprendrons à faire des tests plus compliqués un peu plus loin.

Faisons quelques tests sur un script que nous appellerons "conditions.sh" :

Code : Bash

```
#!/bin/bash

nom="Bruno"

if [ $nom = "Bruno" ]
then
echo "Salut Bruno !"
fi
```

Comme \$nom est bien égal à "Bruno", ce script affichera :

Code : Console

```
Salut Bruno !
```

Essayez de changer le test, si vous n'écrivez pas précisément "Bruno", le if ne sera pas exécuté, et donc votre script n'affichera rien.

Notez aussi que vous pouvez tester 2 variables à la fois dans le if :

Code : Bash

```
#!/bin/bash

nom1="Bruno"
nom2="Marcel"

if [ $nom1 = $nom2 ]
then
echo "Salut les jumeaux !"
fi
```

Comme ici \$nom1 est différent de \$nom2, le contenu du if ne sera pas exécuté. Le script n'affichera donc rien.

Sinon

Si vous souhaitez faire quelque chose de particulier quand la condition n'est *pas* remplie, vous pouvez rajouter un "else" qui signifie "sinon".

En français, ça s'écrirait comme ceci :

Citation

```
SI test_de_variable
ALORS
----> effectuer_une_action
SINON
----> effectuer_une_action
FIN SI
```

Code : Bash

```
if [ test ]
then
echo "C'est vrai"
else
echo "C'est faux"
fi
```

Reprenons notre script de tout à l'heure, et ajoutons-lui un else :

Code : Bash

```
#!/bin/bash

nom="Bruno"

if [ $nom = "Bruno" ]
then
echo "Salut Bruno !"
else
echo "J'te connais pas, ouste !"
fi
```

Bon, comme la variable vaut toujours la même chose, le else ne sera jamais exécuté, ce n'est pas rigolo. Je vous propose plutôt de vous baser sur le premier paramètre (\$1) envoyé au script :

Code : Bash

```
#!/bin/bash

if [ $1 = "Bruno" ]
then
echo "Salut Bruno !"
else
echo "J'te connais pas, ouste !"
fi
```

Testez maintenant votre script en lui donnant un paramètre :

Code : Console

```
$ ./conditions.sh Bruno
Salut Bruno !
```

Et si vous mettez autre chose :

Code : Console

```
$ ./conditions.sh Jean
J'te connais pas, ouste !
```



Notez que le script plante si vous oubliez de l'appeler avec un paramètre. Pour bien faire, il faudrait d'abord vérifier dans un if s'il y a au moins un paramètre. Nous apprendrons à faire cela plus loin.

Sinon si

Il existe aussi le mot-clé "elif", abréviation de "else if", qui signifie "sinon si". Sa forme ressemble à cela :

Citation

```
SI test_de_variable
ALORS
-----> effectuer_une_action
SINON SI autre_test
ALORS
-----> effectuer_une_action
SINON SI encore_un_autre_test
ALORS
-----> effectuer_une_action
SINON
-----> effectuer_une_action
FIN SI
```

C'est un peu plus compliqué hein ? 😊

Sachez qu'on peut mettre autant de "sinon si" qu'on veut. Là j'en ai mis deux. En revanche, on ne peut mettre qu'un seul "sinon", qui sera exécuté à la fin si aucune des conditions précédentes n'est vérifiée.

Bash va d'abord analyser le premier test. S'il est vérifié, il effectuera la première action indiquée, sinon il ira au premier sinon si, au second, etc., jusqu'à trouver une condition qui soit vérifiée. Si aucune condition n'est vérifiée, c'est le "sinon" qui sera lu.

Bien, voyons comment cela s'écrit en bash :

Code : Bash

```
if [ test ]
then
echo "Le premier test a été vérifié"
elif [ autre_test ]
then
echo "Le second test a été vérifié"
elif [ encore_autre_test ]
then
echo "Le troisième test a été vérifié"
else
echo "Aucun des tests précédents n'a été vérifié"
fi
```

On peut reprendre notre script précédent et l'adapter pour utiliser des elif :

Code : Bash

```
#!/bin/bash

if [ $1 = "Bruno" ]
then
echo "Salut Bruno !"
elif [ $1 = "Michel" ]
then
echo "Bien le bonjour Michel"
```

```
elif [ $1 = "Jean" ]
then
echo "Hé Jean, ça va ?"
else
echo "J'te connais pas, ouste !"
fi
```

Vous pouvez tester ce script, encore une fois n'oubliez pas d'envoyer un paramètre sinon il plantera, ce qui est normal.

Les tests

Voyons maintenant un peu quels tests on peut faire. Pour l'instant on a juste testé si 2 chaînes de caractères étaient identiques, mais on peut faire beaucoup plus de choses que ça !

Les différents types de tests

On peut faire 3 types de tests différents en bash :

- Tests sur des chaînes de caractères
- Tests sur des nombres
- Tests sur des fichiers

Nous allons découvrir tous ces types de tests maintenant et les essayer. 😊

Tests sur des chaînes de caractères

Comme vous le savez maintenant, en bash toutes les variables sont considérées comme des chaînes de caractères. Il est donc très facile de tester ce que vaut une chaîne de caractères.

Condition	Signification
\$chaine1 = \$chaine2	Teste si les 2 chaînes sont identiques. Notez que bash est sensible à la casse : "b" est donc différent de "B". Il est aussi possible d'écrire 2 "==" pour les habitués du langage C.
\$chaine1 != \$chaine2	Teste si les 2 chaînes sont différentes
-z \$chaine	Teste si la chaîne est vide
-n \$chaine	Teste si la chaîne est non vide

Testons par exemple si 2 paramètres sont différents :

Code : Bash

```
#!/bin/bash

if [ $1 != $2 ]
then
echo "Les 2 paramètres sont différents !"
else
echo "Les 2 paramètres sont identiques !"
fi
```

Code : Console

```
$ ./conditions.sh Bruno Bernard
Les 2 paramètres sont différents !
```

Code : Console

```
$ ./conditions.sh Bruno Bruno
Les 2 paramètres sont identiques !
```

On peut aussi tester si le paramètre existe avec `-z` (teste si la chaîne est vide). En effet, si une variable n'est pas définie, elle est considérée comme vide par bash. On peut donc par exemple s'assurer que `$1` existe comme ceci :

Code : Bash

```
#!/bin/bash

if [ -z $1 ]
then
echo "Pas de paramètre"
else
echo "Paramètre présent"
fi
```

Code : Console

```
$ ./conditions.sh
Pas de paramètre
```

Code : Console

```
$ ./conditions.sh param
Paramètre présent
```

Tests sur des nombres

Bien que bash gère les variables comme des chaînes de caractères pour son fonctionnement interne, rien ne nous empêche de faire des comparaisons de nombres si ces variables contiennent des nombres.

Condition	Signification
<code>\$num1 -eq \$num2</code>	Teste si les nombres sont égaux (<i>equal</i>). A ne pas confondre avec le "=" qui, lui, compare 2 chaînes de caractères.
<code>\$num1 -ne \$num2</code>	Test si les nombres sont différents (<i>non equal</i>). Encore une fois, ne confondez pas avec "!=" qui est censé être utilisé sur des chaînes de caractères.
<code>\$num1 -lt \$num2</code>	Teste si num1 est inférieur (<) à num2 (<i>lower than</i>)
<code>\$num1 -le \$num2</code>	Teste si num1 est inférieur ou égal (<=) à num2 (<i>lower or equal</i>)

<code>\$num1 -gt \$num2</code>	Teste si num1 est supérieur (>) à num2 (<i>greater than</i>)
<code>\$num1 -ge \$num2</code>	Teste si num1 est supérieur ou égal (>=) à num2 (<i>greater or equal</i>)

Testons par exemple si un nombre est supérieur ou égal à un autre nombre :

Code : Bash

```
#!/bin/bash

if [ $1 -ge 20 ]
then
echo "Vous avez envoyé 20 ou plus"
else
echo "Vous avez envoyé moins de 20"
fi
```

Code : Console

```
$ ./conditions.sh 23
Vous avez envoyé 20 ou plus
```

Code : Console

```
$ ./conditions.sh 11
Vous avez envoyé moins de 20
```

Tests sur des fichiers

Un des avantages de bash sur d'autres langages est qu'on peut très facilement faire des tests sur des fichiers : savoir s'ils existent, si on peut écrire dedans, s'ils sont plus vieux, plus récents, etc.

Condition	Signification
<code>-e \$nomfichier</code>	Teste si le fichier existe
<code>-d \$nomfichier</code>	Teste si le fichier est un répertoire. N'oubliez pas que sous Linux, tout est considéré comme un fichier, même les répertoires !
<code>-f \$nomfichier</code>	Teste si le fichier est un... fichier. Un vrai fichier cette fois, pas un dossier 😊
<code>-L \$nomfichier</code>	Teste si le fichier est un lien symbolique (raccourci)
<code>-r \$nomfichier</code>	Teste si le fichier est lisible (r)
<code>-w \$nomfichier</code>	Teste si le fichier est modifiable (w)
<code>-x \$nomfichier</code>	Teste si le fichier est exécutable (x)
<code>\$fichier1 -nt \$fichier2</code>	Teste si fichier1 est plus récent que fichier2 (<i>newer than</i>)
<code>\$fichier1 -ot \$fichier2</code>	Teste si fichier1 est plus vieux que fichier2 (<i>older than</i>)

Je vous propose de faire un script qui demande à l'utilisateur de rentrer le nom d'un répertoire, et qui vérifie si c'est bien un répertoire :

Code : Bash

```
#!/bin/bash

read -p 'Entrez un répertoire : ' repertoire

if [ -d $repertoire ]
then
echo "Bien, vous avez compris ce que j'ai dit !"
else
echo "Vous n'avez rien compris..."
fi
```

Code : Console

```
Entrez un répertoire : /home
Bien, vous avez compris ce que j'ai dit !
```

Code : Console

```
Entrez un répertoire : rienavoir.txt
Vous n'avez rien compris...
```

Notez que bash vérifie au préalable que le répertoire existe bel et bien.

Effectuer plusieurs tests à la fois

Dans un if, il est possible de faire plusieurs tests à la fois. En général, on vérifie :

- Si un test est vrai ET qu'un autre test est vrai
- Si un test est vrai OU qu'un autre test est vrai

Les 2 symboles à connaître sont :

- **&&** : signifie "et"
- **||** : signifie "ou"

Il faut encadrer chaque condition par des crochets. Prenons un exemple :

Code : Bash

```
#!/bin/bash

if [ $# -ge 1 ] && [ $1 = 'koala' ]
```

```
then
echo "Bravo !"
echo "Vous connaissez le mot de passe"
else
echo "Vous n'avez pas le bon mot de passe"
fi
```

Le test vérifie 2 choses :

- Qu'il y a au moins un paramètre ("si \$# est supérieur ou égal à 1")
- Que le premier paramètre est bien koala ("si \$1 est égal à koala")

Si ces 2 conditions sont remplies, alors le message indiquant qu'on a trouvé le bon mot de passe s'affichera.

Code : Console

```
$ ./conditions.sh koala
Bravo !
Vous connaissez le mot de passe
```



Notez que les tests sont effectués l'un après l'autre, et seulement s'ils sont nécessaires. Bash vérifie d'abord s'il y a au moins un paramètre. Si ce n'est pas le cas, il ne fera pas le second test puisque la condition sera de toute façon fausse.

Inverser un test

Il est possible d'inverser un test en utilisant la négation. En bash, c'est le point d'exclamation "!" qui exprime la négation.

Code : Bash

```
if [ ! -e fichier ]
then
echo "Le fichier n'existe pas"
fi
```

Vous en aurez besoin, donc n'oubliez pas ce petit point d'exclamation. 😊

case : tester plusieurs conditions à la fois

On a vu tout à l'heure un if un peu complexe, qui faisait appel à des elif et un else :

Code : Bash

```
#!/bin/bash

if [ $1 = "Bruno" ]
then
echo "Salut Bruno !"
elif [ $1 = "Michel" ]
then
echo "Bien le bonjour Michel"
elif [ $1 = "Jean" ]
then
echo "Hé Jean, ça va ?"
else
```

```
echo "J'te connais pas, ouste !"
fi
```

Ce genre de "gros if qui teste toujours la même variable" ne pose pas de problème mais n'est pas forcément très facile à lire pour le programmeur. On peut utiliser si on veut à la place l'instruction *case*.

Le rôle de *case* est de tester la valeur d'une même variable, mais de manière plus concise et lisible.

Voyons comment on écrirait la condition précédente avec un *case* :

Code : Bash

```
#!/bin/bash

case $1 in
  "Bruno")
    echo "Salut Bruno !"
    ;;
  "Michel")
    echo "Bien le bonjour Michel"
    ;;
  "Jean")
    echo "Hé Jean, ça va ?"
    ;;
  *)
    echo "J'te connais pas, ouste !"
    ;;
esac
```

Ca fait beaucoup de nouveautés d'un coup. 🤪

Analysons la structure du *case* !

Code : Bash

```
case $1 in
```

Tout d'abord, on indique que l'on veut tester la valeur de la variable *\$1*. Bien entendu, vous pouvez remplacer *\$1* par n'importe quelle variable que vous voulez tester.

Code : Bash

```
"Bruno")
```

Là, on teste une valeur. Cela signifie "Si *\$1* est égal à Bruno". Notez qu'on peut aussi utiliser une étoile comme joker : "*B**" acceptera tous les mots qui commencent par un B majuscule.

Si la condition est vérifiée, tout ce qui suit est exécuté jusqu'au prochain double point-virgule :

Code : Bash

```
;;
```

Important, il ne faut pas l'oublier : le double point-virgule dit à bash d'arrêter la lecture du case là. Il saute donc à la ligne qui suit le "esac" qui signale la fin du case.

Code : Bash

```
*)
```

C'est en fait le "else" du case. Si aucun des tests précédents n'a été vérifié, alors c'est cette section qui sera lue.

Code : Bash

```
esac
```

Marque la fin du case (esac, c'est "case" à l'envers !).

On peut aussi faire des "ou" dans un case. Dans ce cas, petit piège, il ne faut pas mettre deux || mais un seul ! Exemple :

Code : Bash

```
#!/bin/bash

case $1 in
  "Chien" | "Chat" | "Souris")
    echo "C'est un mammifère"
    ;;
  "Moineau" | "Pigeon")
    echo "C'est un oiseau"
    ;;
  *)
    echo "Je ne sais pas ce que c'est"
    ;;
esac
```

Vous venez d'avoir un bon tour d'horizon de ce qu'il est possible de tester en bash ! 😊

Avec ces nouvelles connaissances, vous pouvez commencer à faire des scripts intéressants, surtout si vous utilisez les paramètres (\$1, \$2...) que l'utilisateur peut rentrer.

On s'entraînera plus loin à faire des scripts plus complets car, malgré sa syntaxe un peu particulière, on peut connaître l'essentiel du bash rapidement, mais réaliser de "gros scripts" est un peu plus complexe et nécessite un peu d'expérience... d'où la nécessité je pense de faire un petit TP tout à l'heure. 😊

Bien, il nous reste à voir les boucles et nous pourrons commencer à pratiquer pour de bon !

Les boucles

Nous allons découvrir dans ce chapitre un autre élément de base de tous les langages : les boucles. Ces structures permettent de répéter autant de fois que nécessaire une partie du code. En bash, on n'y échappe pas !

Les consignes sont les mêmes que pour le chapitre sur les conditions : il faut être vigilant sur la syntaxe. Un espace de trop ou de moins, un oubli d'un caractère spécial, et plus rien ne marche. Soyez donc très rigoureux lorsque vous codez !

Si vous suivez cette simple règle, vous n'aurez pas de problème. 😊

while : boucler "tant que"

Le type de boucle que l'on rencontre le plus couramment en bash est le while.

Le principe est de faire un code qui ressemble à ceci :

Citation

```
TANT QUE test
FAIRE
-----> effectuer_une_action
RECOMMENCER
```

En bash, on l'écrit comme ceci :

Code : Bash

```
while [ test ]
do
echo 'Action en boucle'
done
```



Il est aussi possible, comme pour le if, d'assembler les 2 premières lignes en 1, à condition de mettre un point-virgule :

```
while [ test ]; do
echo 'Action en boucle'
done
```

On va demander à l'utilisateur de dire "oui", et répéter cette action tant que l'utilisateur n'a pas fait ce qu'on voulait. On va créer un script boucles.sh pour l'occasion :

Code : Bash

```
#!/bin/bash

while [ -z $reponse ] || [ $reponse != 'oui' ]
do
read -p 'Dites oui : ' reponse
done
```

On fait 2 tests :

- Est-ce que \$reponse est vide ?
- Est-ce que \$reponse est différent de "oui" ?

Comme il s'agit d'un OU (||), tant que l'un des 2 tests est vrai, on recommence la boucle. La boucle pourrait se traduire par : "Tant

que la réponse est vide ou que la réponse est différente de oui".

On est obligé de vérifier d'abord si la variable n'est pas vide, car sinon le second test plante (essayez, vous verrez).

Essayons ce script :

Code : Console

```
Dites oui : euh
Dites oui : non
Dites oui : bon
Dites oui : oui
```

Comme vous pouvez le voir, il ne s'arrête que lorsqu'on a tapé oui !



Il existe aussi le mot-clé until, qui est l'exact inverse de while. Il signifie "Jusqu'à ce que". Remplacez juste "while" par "until" dans le code précédent pour l'essayer.

for : boucler sur une liste de valeurs



Avertissement pour ceux qui ont déjà fait de la programmation : le "for" en bash ne se comporte pas de la même manière que le for auquel vous êtes habitués dans un autre langage, comme le C ou le PHP. Lisez donc attentivement.

Parcourir une liste de valeurs

La boucle "for" permet de parcourir une liste de valeurs, et de boucler autant de fois qu'il n'y a de valeurs.

Concrètement, la forme d'un for est la suivante :

Citation

```
POUR variable PRENANT valeur1 valeur2 valeur3
FAIRE
-----> effectuer_une_action
VALEUR_SUIVANTE
```

La "variable" va prendre successivement les valeurs "valeur1", "valeur2", "valeur3". La boucle va donc être exécutée 3 fois, et à chaque fois la variable vaudra une nouvelle valeur de la liste.

En bash, la boucle for s'écrit comme ceci :

Code : Bash

```
#!/bin/bash

for variable in 'valeur1' 'valeur2' 'valeur3'
do
echo "La variable vaut $variable"
done
```

Ce qui donne, si on l'exécute :

Code : Console

```
La variable vaut valeur1
La variable vaut valeur2
La variable vaut valeur3
```

Vous pouvez donc vous servir du `for` pour faire une boucle sur une liste de valeurs que vous définissez :

Code : Bash

```
#!/bin/bash

for animal in 'chien' 'souris' 'moineau'
do
    echo "Animal en cours d'analyse : $animal"
done
```

Code : Console

```
Animal en cours d'analyse : chien
Animal en cours d'analyse : souris
Animal en cours d'analyse : moineau
```

Toutefois, la liste de valeurs n'a pas besoin d'être définie directement dans le code. On peut utiliser une variable :

Code : Bash

```
#!/bin/bash

liste_fichiers=`ls`

for fichier in $liste_fichiers
do
    echo "Fichier trouvé : $fichier"
done
```

Ce script liste tous les fichiers trouvés dans le répertoire actuel :

Code : Console

```
Fichier trouvé : boucles.sh
Fichier trouvé : conditions.sh
Fichier trouvé : variables.sh
```

On pourrait faire un code plus court, sans passer par une variable `$liste_fichiers`, en écrivant :

Code : Bash

```
#!/bin/bash

for fichier in `ls`
do
    echo "Fichier trouvé : $fichier"
done
```



Bien entendu, ici on ne fait qu'afficher le nom du fichier, ce qui n'est pas très amusant ni très utile. On pourrait se servir de notre

script pour renommer chacun des fichiers du répertoire actuel en leur ajoutant un suffixe -old par exemple :

Code : Bash

```
#!/bin/bash

for fichier in `ls`
do
mv $fichier $fichier-old
done
```

Essayons de voir si l'exécution du script renomme bien tous les fichiers :

Code : Console

```
$ ls
boucles.sh conditions.sh variables.sh
$ ./boucles.sh
$ ls
boucles.sh-old conditions.sh-old variables.sh-old
```

A vous de jouer ! Essayez de créer un script multirenommage.sh basé sur ce principe qui va rajouter le préfixe -old... uniquement aux fichiers qui correspondent au paramètre envoyé par l'utilisateur !

Code : Console

```
./multirenommage.sh *.txt
```

Si aucun paramètre n'est envoyé, vous demanderez à l'utilisateur de saisir le nom des fichiers à renommer avec read.

Un for plus classique

Pour les habitués d'autres langages de programmation, le "for" est une boucle qui permet de faire prendre à une variable une suite de nombres.

En bash, comme on l'a vu, le "for" permet de parcourir une liste de valeurs. Toutefois, en trichant un peu à l'aide de la commande `seq`, il est possible de simuler un for classique :

Code : Bash

```
#!/bin/bash

for i in `seq 1 10`;
do
echo $i
done
```

Explication : seq génère tous les nombres allant du premier paramètre au dernier paramètre, donc 1 2 3 4 5 6 7 8 9 10 :

Code : Console

```
1
2
3
```

```
4
5
6
7
8
9
10
```

Si vous le voulez, vous pouvez changer le pas et avancer de 2 en 2 par exemple. Dans ce cas, il faut écrire `seq 1 2 10` pour aller de 1 à 10 en avançant de 2 en 2. Cela va donc générer les nombres 1 3 5 7 9. 😊

Vous connaissez maintenant le fonctionnement des boucles en bash, ce qui va vous être indispensable pour la réalisation de futurs scripts !

En parlant de scripts, je pense qu'il est temps que l'on s'entraîne ensemble sur un vrai script dans un TP. En effet, vous savez faire des choses assez poussées sans le savoir. Il est temps de découvrir un peu ce que le bash a dans le ventre. 😊

TP : Générateur de galerie d'images

L'intérêt du bash ne commence à se sentir que lorsqu'on fait de vrais scripts alors... Il est grand temps de pratiquer !

Vous allez devoir réutiliser dans ce TP un peu tout ce que vous avez appris jusqu'ici sur bash et sur Linux en général. N'oubliez pas que dans les scripts bash vous pouvez réutiliser toutes les commandes de la console que vous connaissez : ls, grep, cut, sort, les flux... allez-y, tous les coups sont permis. Et il risque même de falloir lire le manuel pour trouver quelques paramètres !



Votre objectif est de créer une page web présentant une galerie d'images en fonction des fichiers présents dans un dossier. Plus facile à dire qu'à faire, vous allez voir qu'il y a du challenge. Bonne chance à tous. 😊

Objectifs

Nous souhaitons réaliser dans ce TP un **générateur de galerie d'images** en bash.

Le script s'appellera `galerie.sh`. Pour sa première version, il faudra le placer dans un dossier contenant des images, desquelles il générera des miniatures et un fichier HTML présentant toutes les images du dossier.

Concrètement, le script devra donc :

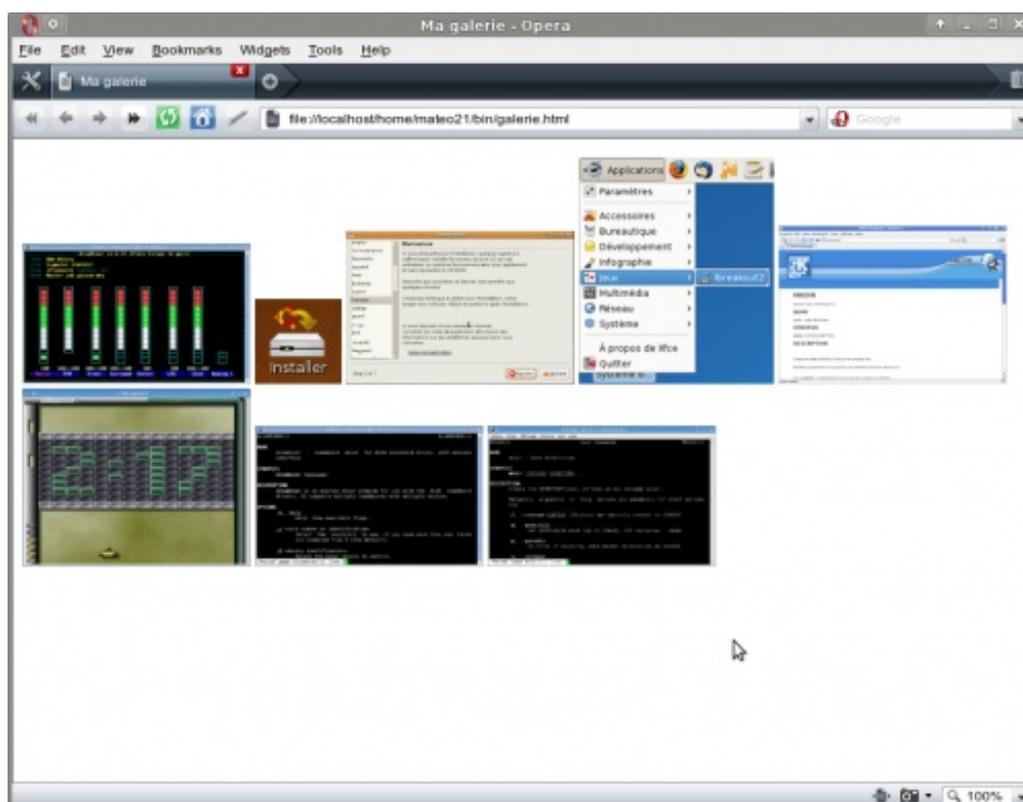
- Créer une image miniature de chaque image du dossier
- Générer un fichier HTML et y insérer ces miniatures
- Faire un lien vers les images en taille originale



Pour réaliser ce script, il est recommandé de connaître un petit peu le (X)HTML. Vous avez un [cours à votre disposition](#) sur le Site du Zéro. C'est assez facile, ceux qui ne connaissent pas ne devraient pas mettre longtemps, d'autant plus qu'il suffit de lire la première partie seulement de ce cours pour faire ce TP.

Le rendu final

La page web que vous devez arriver à générer devrait ressembler à ceci :



Bien entendu, c'est un exemple *minimal*. Il est possible de faire quelque chose de beaucoup plus joli, mais commencez déjà par faire en sorte que ça fonctionne, vous enjoliverez après. 😊

Le code HTML de base

Pour vous aider, enfin surtout pour ceux qui ne sont pas très à l'aise en HTML, je vous propose de vous baser sur le code (minimaliste) suivant :

Code : HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
  <head>
    <title>Ma galerie</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <style type="text/css">
      a img { border:0; }
    </style>
  </head>
  <body>
    <p>

<a href="alsamixer.png"></a>
<a href="icone_installer.png"></a>
<a href="installation.png"></a>

    </p>
  </body>
</html>
```

Vous pouvez réutiliser le début et la fin de ce code source pour chaque fichier HTML de galerie que vous créez. Par contre, au milieu (les images `` et les liens `<a>`) il faudra adapter automatiquement le code en fonction des images présentes dans le dossier.

Vous afficherez des miniatures sur la page et ferez un lien vers la version agrandie de l'image.

Comment générer des miniatures d'images ?

Bonne question. On n'a pas appris à faire cela en ligne de commande, tout simplement parce que ce n'est pas ce que j'appelle une commande "de base" de Linux. Néanmoins, vous devriez avoir le programme `convert`, qui est capable d'effectuer de nombreuses opérations sur des images. À vous d'afficher le manuel et de comprendre comment on l'utilise. 😊

Bon, je vous aide un peu quand même, parce que cette commande a énormément de paramètres. "Miniature" en anglais se dit "thumbnail".

Vous apprécierez peut-être [l'aide en ligne](#), plus étoffée et plus lisible peut-être que le man.

Les paramètres

Notre programme devra accepter un paramètre optionnel : le nom du fichier HTML à générer. S'il n'est pas présent, on génèrera un fichier `galerie.html` par défaut.

Solution

L'heure est venue de passer à la correction !

Avant tout, je tiens à rappeler qu'il n'y a pas une façon de faire ce script mais plusieurs. Je vous présente la mienne et, bien qu'elle fonctionne, je vous préviens qu'on peut largement l'améliorer. Je vous donnerai d'ailleurs des pistes pour améliorer ce script ensuite.

Code : Bash

```
#!/bin/bash

# Vérification des paramètres
# S'ils sont absents, on met une valeur par défaut

if [ -z $1 ]
then
sortie='galerie.html'
else
sortie=$1
fi

# Préparation des fichiers et dossiers

echo '' > $sortie

if [ ! -e miniatures ]
then
mkdir miniatures
fi

# En-tête HTML

echo '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
<head>
<title>Ma galerie</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style type="text/css">
a img { border:0; }
</style>
</head>
<body><p>' >> $sortie

# Génération des miniatures et de la page

for image in `ls *.png *.jpg *.jpeg *.gif 2>/dev/null`
do
convert $image -thumbnail '200x200>' miniatures/$image
echo '<a href="'$image'"></a>' >> $sortie
done

# Pied de page HTML

echo '</p>
</body>
</html>' >> $sortie
```

Quelques commentaires sur ce script :

- Il vérifie d'abord si un paramètre est présent. Si oui, il l'utilise comme nom de fichier de sortie, sinon il utilise galerie.html
- On doit créer un fichier vide pour galerie.html. Normalement, on peut faire ça avec touch, mais si le fichier existe déjà on veut le vider. On choisit donc de faire un echo vide dans ce fichier pour le vider.
- On crée le dossier qui accueillera les images miniatures s'il n'existe pas.

- On écrit l'en-tête HTML dans le fichier
- On fait une boucle sur tous les fichiers de type image (*.png, *.jpg, etc.) qui existent dans le dossier. Pour chacun d'eux, on crée une miniature dans le sous-dossier "miniatures" de taille maximale 200x200 pixels. Le petit symbole ">" permet, [comme indiqué dans la doc du programme](#), de ne pas générer de miniature inutilement si l'image est de base plus petite que la taille des miniatures.
- On écrit dans la page web la balise qui affichera l'image et on fait un lien vers la version agrandie.
- Enfin, on termine la page HTML en fermant les balises.

S'il vous reste des questions sur le fonctionnement de ce script, n'hésitez pas à passer sur les forums !

Améliorations

Comme je le disais plus tôt, le script que je vous ai proposé de faire est minimal. Le but était d'avoir à faire un script accessible à tous et qui produise un résultat intéressant.

Si vous voulez l'améliorer, les pistes ne manquent pas. En voici quelques-unes :

- Améliorer le design de la galerie avec un peu de CSS.
- Permettre de choisir le dossier contenant les images dont on veut générer une galerie. Actuellement, il faut que galerie.sh soit dans le bon dossier pour que ça fonctionne !
- Utiliser un paramètre pour définir la taille des miniatures à générer.
- Afficher le nom de l'image sous chaque image.
- Afficher d'autres informations, comme les dimensions de l'image originale, sous chaque miniature. Il faudra faire appel à l'outil "convert" pour obtenir ces informations.
- Afficher la date de dernière modification sous chaque image.

Pour certaines de ces améliorations, il faudra vous renseigner dans le manuel, voire poser des questions sur les forums.

Ne vous arrêtez pas en si bon chemin ! Cherchez, cherchez, et cherchez encore ! Vous allez vous habituer à faire des recherches et vous deviendrez ainsi beaucoup plus autonomes ! 😊

Ce TP sur bash s'achève ici. Il aura été pour vous l'occasion de manipuler bash pour répondre à un besoin concret, ce qui vous aura permis de faire vos premières armes en bash. C'est à vous ensuite de l'utiliser pour vos propres besoins.

Dans la pratique, on a souvent besoin de bash pour automatiser une action sur un serveur. On couple d'ailleurs souvent les scripts bash avec les crons. Par exemple, cela me permet tous les soirs de gzipper des logs, de les envoyer sur un FTP, etc. Je peux aussi m'en servir pour analyser régulièrement les logs et repérer les comportements anormaux sur le serveur (par exemple, quelqu'un qui charge un nombre trop élevé de pages, etc.).

Bonne route avec bash ! 😊

Le cours se termine ici. Bien entendu, il aurait été possible de le compléter (presque à l'infini !) mais je ne dispose pas d'autant de temps. 😊

Néanmoins, avec l'ensemble de ce cours vous avez désormais je l'espère une introduction à Linux enfin accessible aux débutants. Il reste bien des choses à découvrir, je vous invite à regarder les [tutoriels de la section Linux](#) du site pour en apprendre plus si vous le souhaitez.

Bonne continuation ! 😊