Introduction

Why Learn UNIX?

Because it's fun, obviously.

Actually, people come to UNIX with a variety of wants and needs. Here are some typical reasons:

- Web Pages - Many websites are run under UNIX. Thus, many people are learning UNIX to become better webmasters.
- Programming - UNIX is a very attractive platform for software development, and is often used as such for introductory programming courses.
- Research - Most "supercomputers" run UNIX. UNIX can handle large proccessing tasks with relative ease.
- Open Source Software - Some people believe that software should be free, convenient, and open to the public. These people usually use Linux, a type of UNIX. Most free software is written first for UNIX.

This tutorial is not going to help you with any of these tasks directly, but it will get you started in the right direction.

Dealing with Culture Shock

This tutorial assumes you are familiar with either the Windows or Macintosh operating systems. Five years ago, we could have assumed you knew at least some DOS, but those days are past. This may be the first time you deal with things like command lines, processes, and files.

Above all, you should approach this learning task with an open mind, a full pot of coffee, and plenty of time. UNIX is very difficult to learn, at first. If you use it daily (and have enough patience to overcome the inevitable frustration) you will probably master it within a couple of weeks.

Here are some of the cultural hurdles you may have to overcome:

Using the Wrong Brain

UNIX was developed by computer programmers, for computer programmers. This means that everything in UNIX will make sense, if you are a computer programmer. The rest of us are not so lucky. If your analytic abilities are strong, you should be able to understand how (and eventually why) things work the way they do.

The Horrors of the Command Line

You'll be using what's known as a command line interface. This means that your computing session will consist entirely of typing into a monochrome box on the screen, and reading the text that appears. There are no icons, noises (except for beeps), windows, menus, or taskbars. You won't be using your mouse. In time, you may actually grow to like it (sort of like broccoli).

Alphabet Soup

cd, pwd, lpr, chmod, ls, pico.... Their meaning is obvious, right? Since all commands in UNIX are typed from the keyboard, and "shorter is faster is better", many commands have short, obscure names. Most of them meant something at some time. As we show you new commands, we'll try to come up with a mnemonic device to help you remember what each command does.

The Many Wonders of Diversity

Chocolate, vanilla, strawberry, fudge-almond-bubblegum.... all flavors of ice cream. AIX, IRIX, HP-UX, linux, Solaris... all flavors of UNIX. There is no one standard UNIX operating system. Instead, each computer manufacturer (and some other groups, acting out of either goodwill or maliciousness, depending on your point of view) has their own unique brand of UNIX. This means there are differences between them, especially when it comes to things like printing. We'll concentrate on the similarities, and celebrate the differences.

Conventions Used in this Document

Since everything you do in UNIX is typed from a keyboard, you must be sure to type exactly what is intended. We'll use different fonts and styles to let you know what we want you to do.

Text that should appear on the screen (from the computer to you) will look like this:
Hello world!

Text that you should type into the computer will look like this:
how do you use this silly thing?

You should always press Enter or Return (they're the same thing, for our purposes) after typing in text like that.

Sometimes we'll need you to substitute specific information into a command (such as the year, or your username). It will look like this:
cal month year

This means you should type cal 6 1977, for example, not cal month year.

Sometimes we'll need you to type in a single key. For example,

Press a.

This means to press the a key (don't hit Enter, unless we say to). If we want you to hold down any other keys, we'll say so like this: Press Ctrl-a. This means to hold down the Control (or Ctrl, or Ctl -it's near the Shift key) and press the a key.

Getting an Account

What is an account?

An account signifies that you have permission to use a computer. An account is specific to a computer. For example, you might have an account on Steel, but not have an account on the DaVinci cluster. An account gives you the following:

- a username - a username is a name that identifies you to the computer. Examples include "joebob37" and "clwolfe". At IU, usernames are usually composed of your last name, with one or two letters from your first name (and maybe your middle name, too!). Your username is the same for all UITS computer systems.
- a password - a string of characters that only you and the computer know. This lets you prove that you are who you say you are.
- a home directory - a home directory is a place to put files on the computer.

UNIX Computers at IUB

There are many UNIX computers in the IU system. Each one has a different intended purpose, hardware and software combination, and set of allowed users. You are most likely to use one of the following groups of machines:

Common Name	Host Name	UNIX Flavor	Permitted Users	Purpose
Shakespeare	ariel.ucs.indiana.edu, iago.ucs.indiana.edu, kate.ucs.indiana.edu, lear.ucs.indiana.edu	Solaris	All IUB faculty, staff, and students	Email (only)
Steel	steel.ucs.indiana.edu	Solaris	All IUB faculty, staff, and students	General use, web pages, programming
DaVinci and Ships	davinci.ucs.indiana.edu, ships.ucs.indiana.edu	IRIX	All IUB faculty, staff, and students	Computer clusters for Graphics
Nations	nations.ucs.indiana.edu	Solaris	All IUB faculty, staff, and students	Computer cluster for programming

Research SP	sp.iu.edu	AIX	All IU faculty, and staff and students with faculty sponsors	Research (parallel supercomputing)
Solar	solar.uits.indiana.edu	Solaris	Approved IU faculty, and staff and students with faculty sponsors	Research (shared memory parallel computing)
PPCC	ppcc0.uits.indiana.edu	Linux/NT	Approved IU faculty, and staff and students with faculty sponsors	Research (parallel cluster computing)
Cobalt	cobalt.ucs.indiana.edu	HP-UX	All IU faculty, and staff and students with faculty sponsors	Research (large memory computing)

We recommend that you use Steel as your training ground for learning UNIX. Note that there are other UNIX systems on campus (such as the CS Burrow, Infogate, etc), but we won't be covering those, since they are not owned by UITS.

Creating a New Account

First, use a computer that you are already comfortable with, such as a Macintosh or a Windows computer. You may use your home computer or use a computer in one of the Student Technology Centers. If you are using your own computer, it must be connected to the Internet.

Open a web browser (such as Netscape or Internet Explorer), and go to http://iuaccts.ucs.indiana.edu.

Log in using your IUB network ID.

First, check to see if you already have an account.

1. Click on "Display Information on Existing and Requested Accounts".
2. Check to see if Steel is listed.
3. Click "Return to Services Selection Menu".

If you do not have a Steel account, you can create one.

1. Click on "Create accounts on UITS Computers".
2. Click on Steel. If it is not listed, you may not be authorized to have a Steel account. Contact the UITS Support Center at 855-6789.
3. Confirm that you wish to generate a Steel account.
4. Enter your network password, and optionally enter a password for your steel account. If you do not enter a password, your Steel password will be the same as your Network ID password.
5. Press the button to complete the application process.
6. You account will be generated within 6 hours (usually within 10 minutes).

Exit the account system when you are finished.

Connecting to a UNIX machine

Now that you have an account, you are ready for your first session with UNIX.

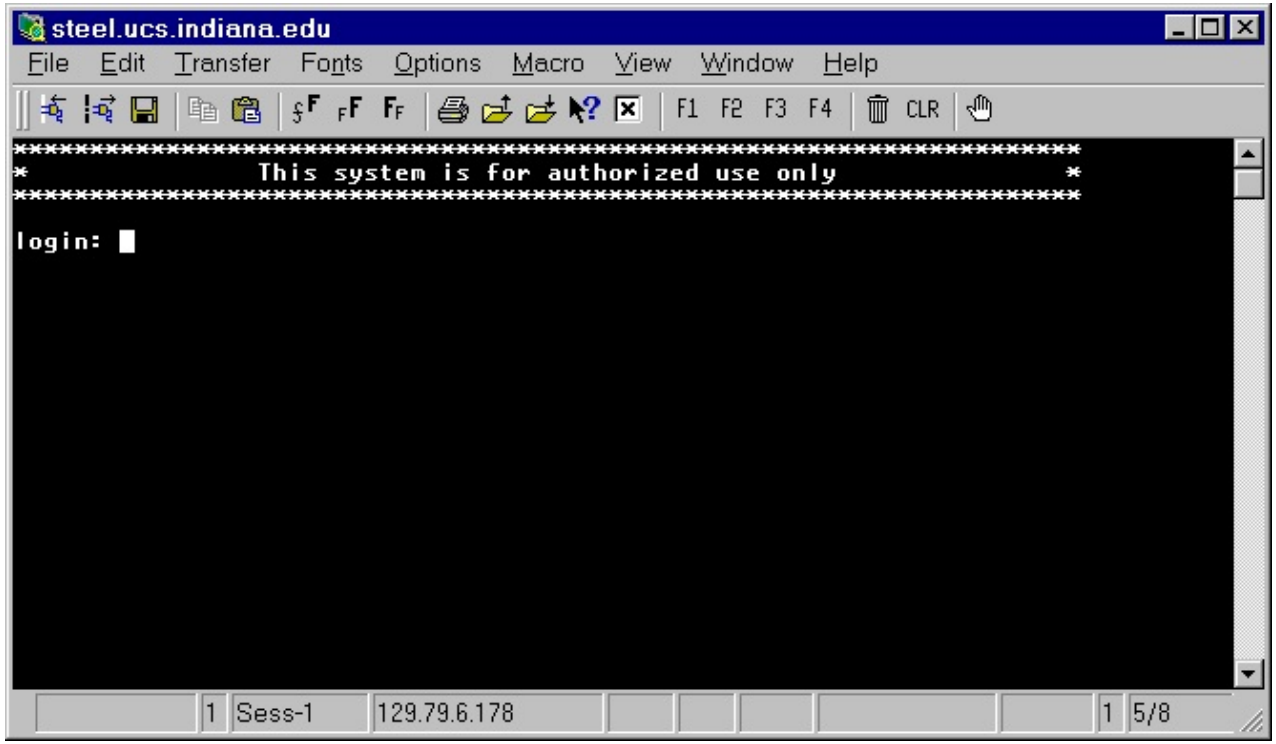
Using Telnet

We'll use Telnet to connect to Steel from your computer. Telnet is a program that lets you have a two-way conversation with a computer over the Internet.

To run Telnet:

- On a Windows computer in the STC's: Go to Start -> Programs ->Communications -> Steel.
- On a Macintosh in the STC's, go to Apple Menu -> Communications -> Telnet. When prompted for a hostname, type steel.ucs.indiana.edu .

A window that looks similar to this should pop up.



It's OK if the window doesn't match exactly. You may have different colors, or totally different menus and buttons.

See Also:

- What is Telnet?

Your First Login

The window should prompt you with the following text:
login:

Type in your username, which should appear on the screen as you type it. For example, I would type:
login: clwolfe

Next, the system will prompt you for your password. Enter it, and notice that nothing will appear on the screen as you type. This protects you from someone looking over your shoulder at the screen (and thus knowing your password).

password:

If you mistype your password, the system will let you try again a certain number of times (usually 3, though this does vary). If you enter the incorrect password more than this number, your account may be disabled.

The first time you log into your account, you will see a statement informing you of your responsibilities as a computer user at IU. Read it carefully and type YES if you agree to accept the stated responsibilities.

You might be prompted to change your password. If you want to keep your password the same, when prompted for a new password, enter your old password again.

Next, the system may ask you what your terminal type is. Accept the default value (probably vt100) by pressing Enter.

Finally, the system banner will appear. It should look something like this:

```
Last login: Mon Feb 21 09:51:03 2000 from nowhere.ucs.indiana.edu
You have mail.
Your home directory on the NFS server resides on the /N/fs1 file system.

Your disk quota is set to 10000 blocks.
**************************************************************************
  SUN ULTRA 2/512 MB      *   Indiana University   *        Node Name STEEL
**************************************************************************
                    This system is for authorized use only.
**************************************************************************
For software problems with UITS UNIX systems post a message to "ucs.system".
For problems with UITS networks call 855-6789.  To contact the machine room
operator call 855-9910 or send mail to opr@indiana.edu.
**************************************************************************
For user support, contact the UITS Support Center (5-6789, ITHELP, IMU M084)
**************************************************************************
The UITS Ombudsman can be reached at: E-mail: ombuds@indiana.edu Phone: 5-5752
**************************************************************************
   The IUPUI and regional campus Internet provider, UUNET, will be performing
   network maintenance on Tuesday, February 22 from 3am to 6am. Campuses
   should expect the possibility of periodic outages and/or slow response
   times for the commodity Internet during this time frame.
                                    ****
   IU began filtering Napster.com servers on February 12th. For more
   information see the Knowledge Base article at
   http://kb.indiana.edu/data/aifq.html
                                    ****
   UITS has extended weekend hours for IUIS. The expanded hours offer access
   to Insite and the touch-tone registration system on Saturdays from 7am
   until 5pm and on Sundays from 10:30am until 5pm.
**************************************************************************
steel /N/fs1/clwolfe/Steel $
```

This information is called the Message of the Day, or motd. It is where important information about the system is posted. You should read over it carefully every time you log in, so you will know about system downtime and other news.

Now let's look at the last line again:

steel /N/fs1/clwolfe/Steel $

This is called the *prompt*. This is where you type commands into UNIX. Your prompt won't look exactly like mine. It may end with a % or a $. The important thing is that there is a cursor

at the end of it. This is where commands that you type will appear.

See Also:

- In Unix, how do I log in and out of my account?
- In Unix, how do I change my password?
- If I forgot my password on an IUB shared system, what should I do?

Logging Out Gracefully

Now that you know how to log in, you should learn how to log out. It's a lot like calling someone on the phone - you should hang up when you are finished. Properly logging out is important from a security point of view, as well - if you leave your telnet session running, anyone could start using your account as you!

Logging out is easy. All you have to do is type the logout command and press enter.

steel /N/fs1/clwolfe/Steel $ logout

This will log you out of the system, and close your telnet connection. Your telnet client might pop up a window at this point to inform you of this.

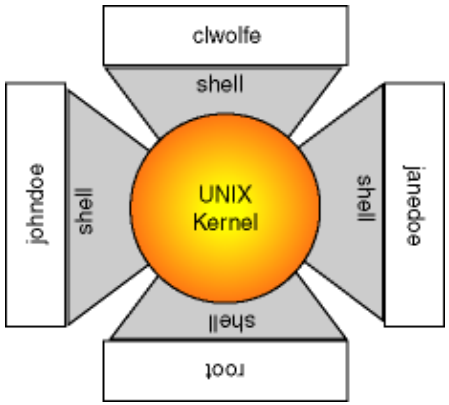
Now log back in to the system (you need the practice, right?) to continue with the tutorial.

The Shell

What is the Shell?

In Windows and the MacOS, you access files and programs by clicking on icons and menus. The computer shows you pretty pictures, and you respond by pushing buttons and moving the mouse.

Every operating system has a *interface* - the point of contact between the human and the machine. This interface is called a GUI (Graphical User Interface) on Windows and MacOS. In UNIX, the interface is called the *shell*.



Notice that there is more than one user using the UNIX machine. Each user gets their own copy of a shell.

The shell is your interface to UNIX. It is the component of UNIX that acts as an intermediary on your behalf. If you and your shell get along, your UNIX experience will be a good one.

Specifically, the shell handles the following for you:

- Finds and executes programs
- Provides you with access to your files
- Makes it easy (or hard) to recall earlier commands and edit command lines

See also:

- In UNIX, what is the Shell?

Types of Shells

Since UNIX culture is so diverse, there are many shells to choose from. They differ mainly in terms of how much pain the user likes, and which lineage they come from.

UNIX name	English name	Decendant of	Pain level	Features
sh	Bourne Shell	(First shell)	Extremely high	Easy to script, great to use if you are made of silicon
csh	C Shell	(Second shell)	High	Command aliasing, job control
tcsh	TC Shell	csh	Tolerable	Filename completion, history, aliasing
ksh	Korn Shell	sh	Low	Filename completion, command line editing
bash	Bourne Again Shell	sh	Pain-free, a real pleasure to use	Cursor keys, filename completion, reliable backspace

Pain refers to the amount of frustration you will experience as a new user. This frustration stems mostly from having "broken keys" - keys that do not function as expected, such as the cursor keys and backspace.

Shell choice is a very personal matter. While I recommend bash to new users, you may find that you like a different shell.

See Also:

- In Unix, where can I get information on differences between the various shells?
- In Unix, what is chsh, and how do I use it to change my shell?

Environment Variables

You are probably curious about what shell you are using. In fact, you may be wondering where information like this is stored - information about your environment.

Your shell has a number of things called *environment variables*. These are variables (ie, a placeholder for information) that many programs look for in order to know how to behave. Let's look at an example.

Type the following into your UNIX session:

steel /N/fs1/clwolfe/Steel $ echo $SHELL

You should see something like this:

/usr/local/bin/bash

We just used the echo command to access the value of the $SHELL environment variable. The vaule turned out to be /usr/local/bin/bash. This means that I am using the bash shell.

There are many other environment variables. To get a full listing, try this:

steel /N/fs1/clwolfe/Steel $ printenv

```
TZ=US/East-Indiana
HOSTNAME=steel.ucs.indiana.edu
MANPATH=/usr/share/man:/usr/local/man:/usr/local/gnu/man:/opt/SUNWspro/man:/usr/X/man
PS1=\h \w $
USER=clwolfe
MACHTYPE=sparc-sun-solaris2.6
MAIL=/var/mail/clwolfe
```

```
LINES=23
EDITOR=pico
LOGNAME=clwolfe
SHLVL=1
COLUMNS=80
SHELLOPTS=braceexpand:hashall:histexpand:monitor:interactive-comments:emacs
SHELL=/usr/local/bin/bash
PRINTER=none
HOSTTYPE=sparc
OSTYPE=solaris2.6
HOME=/N/u/clwolfe/Steel
TERM=vt100
PATH=/N/u/clwolfe/Steel/bin:/usr/sbin:/usr/local/bin:/usr/local/gnu/bin:/bin:/usr/bin:/usr/ucb:/opt/SUNWspro/bin:.:/usr/X/bin
SSH_TTY=/dev/pts/22
EXINIT=set redraw wm=10
```

Most of this information isn't very useful right now, but we'll need it later. Now, let's try setting a new variable of our own creation. Suppose we would like the variable $MYVAR to have the value "ImportantStuff".

The first thing to do is to find out what shell you are using, because (sigh) the syntax is different for the various shells. Then follow the instructions below.

sh	MYVAR="ImportantStuff"; export MYVAR
bash, ksh	export MYVAR="ImportantStuff"
csh, tcsh	setenv MYVAR "ImportantStuff"

Now let's try accessing the variable.

steel /N/fs1/clwolfe/Steel $ echo $MYVAR
ImportantStuff

Notice that we didn't use the dollar sign when setting a variable, but we did when accessing the variable. MYVAR will exist until you logout, at which point it disappears.

Commands

Every time we have done anything useful, we have executed a command to do it. Here we will discuss what a command is, and what they will enable you to do.

Your First Command

We'll start off with a practical example: the cal command. Type the following at the UNIX prompt:

steel /N/fs1/clwolfe/Steel $ cal

```
   February 2000
 S  M Tu  W Th  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29
```

As you can see, the cal command simply prints out the current month's calendar.

Getting Into Arguments

Of course, what use is a calendar that only shows the current month? If only we could specify what month we wanted....

steel /N/fs1/clwolfe/Steel $ cal 6 1977

```
   June 1977
 S  M Tu  W Th  F  S
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

As you can see, we passed along extra information to the command. Each piece of extra information is called an *argument*. Commands differ widely in terms of the number and type of arguments they can accept. The same command may behave differently with a different number of argumants, as well. For example....

steel /N/fs1/clwolfe/Steel $ cal 1977

```
                                1977

         Jan                   Feb                   Mar
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
                   1           1  2  3  4  5          1  2  3  4  5
 2  3  4  5  6  7  8    6  7  8  9 10 11 12    6  7  8  9 10 11 12
 9 10 11 12 13 14 15   13 14 15 16 17 18 19   13 14 15 16 17 18 19
16 17 18 19 20 21 22   20 21 22 23 24 25 26   20 21 22 23 24 25 26
23 24 25 26 27 28 29   27 28                  27 28 29 30 31
30 31
         Apr                   May                   Jun
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
                1  2    1  2  3  4  5  6  7             1  2  3  4
 3  4  5  6  7  8  9    8  9 10 11 12 13 14    5  6  7  8  9 10 11
10 11 12 13 14 15 16   15 16 17 18 19 20 21   12 13 14 15 16 17 18
17 18 19 20 21 22 23   22 23 24 25 26 27 28   19 20 21 22 23 24 25
24 25 26 27 28 29 30   29 30 31                26 27 28 29 30

         Jul                   Aug                   Sep
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
                1  2       1  2  3  4  5  6                1  2  3
 3  4  5  6  7  8  9    7  8  9 10 11 12 13    4  5  6  7  8  9 10
10 11 12 13 14 15 16   14 15 16 17 18 19 20   11 12 13 14 15 16 17
17 18 19 20 21 22 23   21 22 23 24 25 26 27   18 19 20 21 22 23 24
24 25 26 27 28 29 30   28 29 30 31            25 26 27 28 29 30
31
         Oct                   Nov                   Dec
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
                   1           1  2  3  4  5             1  2  3
 2  3  4  5  6  7  8    6  7  8  9 10 11 12    4  5  6  7  8  9 10
 9 10 11 12 13 14 15   13 14 15 16 17 18 19   11 12 13 14 15 16 17
16 17 18 19 20 21 22   20 21 22 23 24 25 26   18 19 20 21 22 23 24
```

```
23 24 25 26 27 28 29   27 28 29 30            25 26 27 28 29 30 31
30 31
```

With only one argument, cal assumes it is a year, and prints out a 12-month calendar. This could be confusing; what if you wanted August's calendar, and tried....

steel /N/fs1/clwolfe/Steel $ cal 8

```
                                8

          Jan                   Feb                   Mar
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
 1  2  3  4  5  6  7             1  2  3  4                1  2  3
 8  9 10 11 12 13 14    5  6  7  8  9 10 11    4  5  6  7  8  9 10
15 16 17 18 19 20 21   12 13 14 15 16 17 18   11 12 13 14 15 16 17
22 23 24 25 26 27 28   19 20 21 22 23 24 25   18 19 20 21 22 23 24
29 30 31               26 27 28 29            25 26 27 28 29 30 31

          Apr                   May                   Jun
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
 1  2  3  4  5  6  7          1  2  3  4  5                   1  2
 8  9 10 11 12 13 14    6  7  8  9 10 11 12    3  4  5  6  7  8  9
15 16 17 18 19 20 21   13 14 15 16 17 18 19   10 11 12 13 14 15 16
22 23 24 25 26 27 28   20 21 22 23 24 25 26   17 18 19 20 21 22 23
29 30                  27 28 29 30 31         24 25 26 27 28 29 30

          Jul                   Aug                   Sep
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
 1  2  3  4  5  6  7             1  2  3  4                      1
 8  9 10 11 12 13 14    5  6  7  8  9 10 11    2  3  4  5  6  7  8
15 16 17 18 19 20 21   12 13 14 15 16 17 18    9 10 11 12 13 14 15
22 23 24 25 26 27 28   19 20 21 22 23 24 25   16 17 18 19 20 21 22
29 30 31               26 27 28 29 30 31      23 24 25 26 27 28 29
                                              30
          Oct                   Nov                   Dec
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
    1  2  3  4  5  6                1  2  3                      1
 7  8  9 10 11 12 13    4  5  6  7  8  9 10    2  3  4  5  6  7  8
14 15 16 17 18 19 20   11 12 13 14 15 16 17    9 10 11 12 13 14 15
21 22 23 24 25 26 27   18 19 20 21 22 23 24   16 17 18 19 20 21 22
28 29 30 31            25 26 27 28 29 30      23 24 25 26 27 28 29
                                              30 31
```

Naturally, cal printed out the calendar for the year 8, not the month. Be sure to do a sanity check on your command before pressing enter. cal is harmless, but you wouldn't want to pass the wrong argument to a more serious command, such as rm (which deletes files).

In addition to arguments, many commands allow you to specify *options*. An option is simply a modifer - it changes the way a command works. Let's try this on another non-destructive command, ls.

steel /N/fs1/clwolfe/Steel $ ls

```
Mail         dead.letter   matlabrc.m    samuel.txt
MailArchive  matlab       personal     www
```

ls prints out a list of all of the files in the current directory. We'll talk about ls in more detail later, but for now, let's try using options with ls to get more information.

steel /N/fs1/clwolfe/Steel $ ls -l

```
total 10
drwx------   2 clwolfe  staff     1024 Feb 21 14:32 Mail
drwx------   2 clwolfe  staff       96 Jan  6 10:10 MailArchive
-rw-------   1 clwolfe  staff     3148 Feb 17 14:03 dead.letter
drwx------   2 clwolfe  staff     1024 Nov 12 16:54 matlab
-rw-------   1 clwolfe  staff     4725 Aug 16  1999 matlabrc.m
drwx------   2 clwolfe  staff       96 Nov  9 15:31 personal
-rw-------   1 clwolfe  staff     3305 Dec 10 12:19 samuel.txt
drwx--x--x   8 clwolfe  staff     1024 Feb 10 14:05 www
```

Here we passed the l option (l, as in L, not 1) to the ls command. Most options begin with a -. That's how the command can tell that it is an option, not an argument. Some commands have options that start with two hyphens.

Gluing Commands Together

You may have noticed, when we were playing with the cal commands, that sometimes the output of a command will fly by too fast for you to read. This is a common problem, and it has a simple solution. You might think that there is an option that slows down the output, or something like that. In fact, there is a much more general solution - the more command. The more command is very stupid - all it knows how to do is to take whatever is put into it, and then print it to the screen, one screenful at a time. So, we need a way of gluing commands together.

We can do this using the "pipe operator", which is provided by your shell. It looks like a little vertical line (|), and is typically found near the enter key. Here it is in action:

steel /N/fs1/clwolfe/Steel $ cal 1999 | more

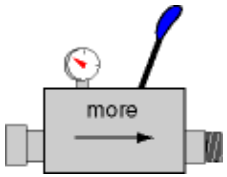
```
                                1999

          Jan                   Feb                   Mar
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
                1  2       1  2  3  4  5  6       1  2  3  4  5  6
 3  4  5  6  7  8  9    7  8  9 10 11 12 13    7  8  9 10 11 12 13
10 11 12 13 14 15 16   14 15 16 17 18 19 20   14 15 16 17 18 19 20
17 18 19 20 21 22 23   21 22 23 24 25 26 27   21 22 23 24 25 26 27
24 25 26 27 28 29 30   28                     28 29 30 31
31
          Apr                   May                   Jun
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
             1  2  3                      1          1  2  3  4  5
 4  5  6  7  8  9 10    2  3  4  5  6  7  8    6  7  8  9 10 11 12
11 12 13 14 15 16 17    9 10 11 12 13 14 15   13 14 15 16 17 18 19
18 19 20 21 22 23 24   16 17 18 19 20 21 22   20 21 22 23 24 25 26
25 26 27 28 29 30      23 24 25 26 27 28 29   27 28 29 30
                       30 31
          Jul                   Aug                   Sep
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
             1  2  3    1  2  3  4  5  6  7             1  2  3  4
 4  5  6  7  8  9 10    8  9 10 11 12 13 14    5  6  7  8  9 10 11
11 12 13 14 15 16 17   15 16 17 18 19 20 21   12 13 14 15 16 17 18
18 19 20 21 22 23 24   22 23 24 25 26 27 28   19 20 21 22 23 24 25
25 26 27 28 29 30 31   29 30 31               26 27 28 29 30
```

```
          Oct                  Nov                  Dec
  S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
                 1  2       1  2  3  4  5  6             1  2  3  4
  3  4  5  6  7  8  9    7  8  9 10 11 12 13    5  6  7  8  9 10 11
 10 11 12 13 14 15 16   14 15 16 17 18 19 20   12 13 14 15 16 17 18
 17 18 19 20 21 22 23   21 22 23 24 25 26 27   19 20 21 22 23 24 25
 24 25 26 27 28 29 30   28 29 30               26 27 28 29 30 31
 31
```

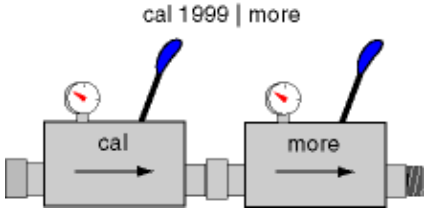
That isn't very interesting on a web page, but you should notice that in the telnet session, the output pauses when the screen is full. Press Space to see the next page.

more isn't the only command that works this way. In fact, all commands can be linked using the pipe operator.

Try thinking of a command as a box with pipes going in and out of it, like so:



In UNIX terminology, the pipe going in is called Standard Input (or STDIN), the pipe going out is called Standard Out (STDOUT), the gauge on top is called Standard Error (STDERR), and the lever on top is the command you use to start the box in motion. Not all commands use STDIN and STDERR, but every command uses STDOUT if it produces output. The pipe operator simply connects the commands together, like this:



All the pipe operator does is join the two commands together.

If you would like to direct output to a file rather than your screen (which is where STDOUT goes, by default), you can say so using another shell operator. Try this:
steel /N/fs1/clwolfe/Steel $ cal 1999 > 1999calendar.txt

The greater-than symbol (>) tells your shell that you would like STDOUT to go to a file, rather than to the terminal or to another program. Likewise, the less-than sign (<) tells your shell to feed the contents of a file to a command as STDIN. This won't work with cal, since cal doesn't use STDIN. We'll use this trick later when working with other commands.

What Happens When you Press Enter

When you press enter, after typing a command line, your shell does the following:

1. Look for any "shell meta-characters" - things like <,>, and |. Split the command line up using these as markers.
2. Look for an environment variable, called $PATH, and look in the places it lists for commands that match the names of the command you typed. We'll talk about $PATH more later.
3. If there are any >'s or <'s, open the named files for reading or writing, accordingly.
4. Connect the plumbing together, and begin executing the commands.

If something goes wrong, the command will output error messages to STDERR. STDERR by default goes to your terminal, regardless of where STDOUT is going.

Command for Commands

The following are commands that are useful for finding out more about commands.

The man command

The man command is by far the most useful command in UNIX. man is an abbreviation for "manual", as in "user's manual". If you need help with a command, you first instinct should be to type

steel /N/fs1/clwolfe/Steel $ man command

where command is the command that you need help with. For example, if you needed help with the cal command, you would type:

steel /N/fs1/clwolfe/Steel $ man cal

```
Reformatting page.  Wait... done

User Commands                                           cal(1)

NAME
     cal - display a calendar

SYNOPSIS
     cal [ [ month ] year ]

DESCRIPTION
     The cal utility writes a Gregorian calendar to standard out-
     put.   If the year operand is specified, a calendar for that
     year is written.  If no operands are specified,  a  calendar
     for the current month is written.

OPERANDS
     The following operands are supported:

     month   Specify the month to be displayed, represented as  a
             decimal  integer  from 1 (January) to 12 (December).
             The default is the current month.

     year    Specify  the  year  for  which   the   calendar   is
             displayed,  represented  as a decimal integer from 1
             to 9999.  The default is the current year.

ENVIRONMENT
     See environ(5) for descriptions of the following environment
     variables  that  affect  the  execution  of  cal:   LC_TIME,
     LC_MESSAGES, and NLSPATH.

EXIT STATUS
     The following exit values are returned:
     0         Successful completion.
     >0        An error occurred.
```

```
ATTRIBUTES
     See attributes(5) for descriptions of the  following  attri-
     butes:


     _____________________________________
    | ATTRIBUTE TYPE|  ATTRIBUTE VALUE|
    |_____________________________________
    | Availability  |  SUNWesu        |
    |_______________|_________________|

SEE ALSO
     calendar(1), attributes(5), environ(5)

NOTES
     An unusual calendar is printed for September 1752.  That  is
     the  month  11 days were skipped to make up for lack of leap
     year adjustments.  To see this calendar, type:
          cal 9 1752

SunOS 5.6            Last change: 1 Feb 1995                   1

User Commands                                              cal(1)

     The command cal 83 refers to the year 83, not 1983.

     The year is always considered to start in January.
SunOS 5.6            Last change: 1 Feb 1995                   2
```

Notice that the man page is automatically fed through more for you.

You may find that the man pages are not very helpful, or are confusingly written. You would be correct. Here is some advice about reading man pages:

- The *Synopsis* section is often the most useful part. It gives the basic form of the command, with optional options and arguments in brackets ([]).
- Many of the other sections are there for the system administrator (the person who keeps the UNIX machine at a more tolerable level of disarray). You usually won't be concerned with the Attributes or Exit Status sections.
- Read the examples, and try them out.
- Keep in mind that some of the *really* old commands (such as tar, for example) have so many modes and features that their man page is indecipherable.

Have a look at the See Also section above. Notice that the commands listed there have a number in parentheses following them. This refers to which *section* of the man system that the command appears in. Historically, man pages were divided into eight (or more) sections, as follows.

1. User Commands
2. System Calls
3. Subroutines
4. Devices
5. File Formats
6. Games
7. Miscellaneous
8. Sys. Administration

So if someone tells you to look at the hostname(1) page, they mean that you should look at the man page for the hostname user command. This is provided so that you could also look at the hostname(3) page, which is the C language system call (not, generally, what you want).

If you would like to see man pages from other sections, use this form of man:

steel /N/fs1/clwolfe/Steel $ man -s 3 hostname

which specifies to only display man pages from section 3, for example.

The apropos Command

Of course, what good is the man command if you don't know what the name of the command is? The apropos command searches through the man pages for a keyword that you specify.

For exmaple, if you wanted to find a command for working with calendars, you would use....

steel /N/fs1/clwolfe/Steel $ apropos calendar

```
cal             cal (1)         - display a calendar
calendar        calendar (1)    - reminder service
difftime        difftime (3c)   - computes the difference between two calendar times
mktime          mktime (3c)     - converts a tm structure to a calendar time
```

As you can see from the results, there are four results from the search.

The whereis Command

Often, you may know of a application or command that may or may not be installed on the machine. A good way of finding out is the whereis command. whereis looks for command files in the standard locations, based on the keywords you provide.

steel /N/fs1/clwolfe/Steel $ whereis perl

```
perl: /usr/bin/perl /usr/local/bin/perl5.00404 /usr/local/bin/perl
```

This example tells us that the popular program perl is installed in three locations.

Files

The Zen Nature of Filesystems

Most modern operating systems, such as Windows and the MacOS, protect the user from the details of the filesystem. The user is left with the impression that the notion of a *file* is not a very important one - instead, a user spends most of his or her time working with applications.

This impression is wrong (in fact, most of what you are doing is simply a series of file operations, from one point of view). Don't feel bad for being fooled - the whole point of "pretty" operating systems is to hide the details, to make your computer experience less mentally taxing.

Think of browsing the files on your current computer. You might use a series of Finder windows, or use Windows Explorer. Either way, you would see folders with icons in them. We'll be doing the exact same thing under UNIX - but there are no pretty icons. Now is the time to staunchly lay aside that mouse, and enter bravely into the world of filesystems.

Types of Files

First and foremeost, there are *plain files*. A plain file is a bunch of data that you can refer to by name. The data could be a note to your mother, a GIF image, or a computer program. The *filename* is a bunch of letters, numbers, and punctuation (e.g. "foo.txt", "stuff", "perl5.00054.tar.gz", etc).

Of course, if we just leave files laying around everywhere, things will get very disorganized very quickly. So we need some sort of organization system - a bunch of boxes to put plain files in, for example. A *directory* is exactly that - a box for files. A directory is still a file (in the general sense), but its contents are other files, rather than data. Of course, a directory can

contain other directories, allowing us to build up complicated structures of files, or very simple structures. Directories are the same thing as Folders in Windows or MacOS.

Finally, wouldn't it be neat if we could have more than one name for a file? MacOS has Aliases, and Windows has Shortcuts, which do the same thing. This kind of file (a pointer to another file) is called a *symbolic link* in UNIX. A symbolic link is just an alias for a file. We'll see how symbolic links work later on.

Trees and Paths

Let's try putting files, directories, and symbolic links together, to see what we get. In your UNIX session, try typing the following:

```
steel /N/fs1/clwolfe/Steel $ pwd
/N/fs1/clwolfe/Steel
```

You should have gotten a similar result. pwd is the UNIX command that tells you where you are. It stands for Present Working Directory, which is a mouthful. The example above tells us that I am in the directory /N/fs1/clwolfe/Steel. Wherever you go in UNIX, you'll always be in a directory - in a box, in other words.

In fact, we're in several boxes: The N directory, which contains fs1 (amongst, we presume, other things), which in turn contains clwolfe, which in turn contains a directory called Steel. The / between the names tells us when we go in one more level. The whole thing is called a *pathname* - a path through the filesystem. You've probably seen things like this before - for example, Hard Disk:System Folder:Fonts, or c:\Windows\Fonts . Notice that in UNIX, the slash is always a "foreward slash" like this: / .

Let's try making a new directory. Try this:

```
steel /N/fs1/clwolfe/Steel $ mkdir stuff
steel /N/fs1/clwolfe/Steel $ ls -F

Mail/         dead.letter   matlabrc.m     www/
MailArchive/  matlab/       personal/     stuff/
```

The first command, mkdir, makes a new directory. We'll talk about mkdir more later.

The next command, ls -F, lists the files in the currect directory. The option -F makes ls do a neat trick - it displays a slash after each directory, so that you can tell the difference between a plain file and a directory at a glance.

Notice that there is now a directory called stuff. Let's move into it.

```
steel /N/fs1/clwolfe/Steel $ cd stuff
steel /N/fs1/clwolfe/Steel/stuff $ pwd
/N/fs1/clwolfe/Steel/stuff
```

Notice that our working directory is now /N/fs1/clwolfe/Steel/stuff , as expected.

Now suppose we wanted to move back to /N/fs1/clwolfe/Steel. How would we do this? Here are two ways:

```
steel /N/fs1/clwolfe/Steel/stuff $ cd /N/fs1/clwolfe/Steel
steel /N/fs1/clwolfe/Steel/stuff $ cd ..
```

The first way uses the *full pathname* to specify what directory to change to. You can always refer to a file by its full pathname. The full pathname gives the complete path from the very first / (which, incidentally, is called the *root directory*).

You'll probably use the second method more often, though. It appears that we are cd'ing into a directory named "..". In fact, ".." is an alias for the next directory up. Try this:

```
steel /N/fs1/clwolfe/Steel $ cd stuff
steel /N/fs1/clwolfe/Steel/stuff $ ls -aF

./    ../
```

The first line shouldn't suprise you too much: we're just cd'ing back to the stuff directory. Next, we do an ls, with the options F (show /'s for directories) and a new option, a (show *all* files). UNIX "hides" files by giving them a name that starts with a "." . ls doesn't list these files by default.

Anyway, it appears that there are two subdirectories, "." and ".." . I've already told you that ".." is a reference to the next directory up in the tree. Every directory (except the root directory, which makes sense) has a ".." alias in it, so you can always move up.

The "." directory, on the other hand, refers to the current directory. You don't really need to know this, but it does let you do a "fun" trick:

steel /N/fs1/clwolfe/Steel $ cd ./././././././././././././.

I'll leave the interpretation of that one to the reader.

Also, if you recall from our discussion of what an *account* is, you should have a *home directory*. Let's learn more:

steel /N/fs1/clwolfe/Steel $ echo $HOME
/N/fs1/clwolfe/Steel

Your home directory is always available in the environment variable $HOME. My home directory is /N/fs1/clwolfe/Steel . Yours will probably be /N/fs1/username/Steel where username is your username. When you log in to a UNIX machine, you start out in your home directory. If you ever get lost, you can return to your home directory by typing:

steel /N/fs1/clwolfe/Steel $ cd ~

Just like ".." is an alias for the next directory up, "~" (called a tilde, pronounced "till-duh") is an alias for *your* home directory. You can also use it like this:

steel /N/fs1/clwolfe/Steel $ cd ~/stuff

which would change to the stuff directory in your home directory.

Permissions, Owners, and Groups

So now we know what files are and how to move around in a filesystem. If you are familiar with Windows or MacOS, you should have had little trouble with the concepts so far. However, on a personal computer (Windows or Mac), there is only one user, so there is no worry about things like "whose file is this" or "does Bob really want me looking at his files". UNIX, however, is designed to have many users on the same machine.

UNIX makes its first step in this direction by giving you a home directory. Each user of a UNIX system has their own home directory. On Steel, mine is /N/fs1/clwolfe/Steel. Yours might be /N/fs1/johndoe/Steel.

It would also be nice to be able to keep track of who owns what files, and who has permission to look at what files. UNIX provides a mechanism for controlling access to your files. Try this in your home directory:

steel /N/fs1/clwolfe/Steel $ ls -alF

```
drwxrwxrwx  10 clwolfe  students     2048 Feb 22 11:14 ./
drwxrwxrwx   8 clwolfe  students     1024 Nov  5 16:19 ../
-rw-------   1 clwolfe  students     1428 Oct 28 08:33 .Xauthority
-rw-------   1 clwolfe  students     4337 Feb 21 18:03 .bash_history
-rw-------   1 clwolfe  students     3027 Mar 10  1999 .cshrc
-rw-------   1 clwolfe  students      473 May  5  1999 .history
-rw-------   1 clwolfe  students     1808 Mar 10  1999 .login
-rw-------   1 clwolfe  students    10602 Feb 22 10:03 .pinerc
-rw-------   1 clwolfe  students     4228 May  5  1999 .profile
drwx------   2 clwolfe  students     1024 Feb 22 11:56 Mail/
drwx------   2 clwolfe  students       96 Jan  6 10:10 MailArchive/
-rw-r--r--   1 clwolfe  students       56 Feb 22 13:07 otherstuff.txt
-rw-r-----   1 clwolfe  students       53 Feb 22 13:05 studentsstuff.txt
drwx------   2 clwolfe  students       96 Feb 22 11:14 stuff/
drwx--x--x   8 clwolfe  students     1024 Feb 10 14:05 www/
```

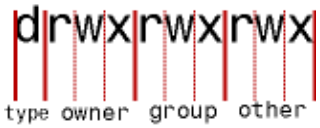
This is our old friend, ls, with three options: a (show all files, even those that begin with "."), F (show directories with a "/" at the end) and a new one, l (long format; show all details about the files). The l option is a lot like the "Details" view in Windows Explorer or MacOS Finder: it shows you all details about the files, besides just their names.

Let's take a close look at the output of ls. First of all, each file is listed, one per line. There's the directory we created, stuff/, and the two dot directories, ../ and ./ , as well as a lot plain files that begin with a dot.

Now look at the columns. The third column is interesting: each file has my username associated with it. The third column lists the *owner* of the file. Since these files are in my home directory, it seems reasonable that I would own them.

The fourth column lists the *group* that the file belongs to. Since I belong to the group students, my files are marked as being owned by someone in the students group.

What about the first column? What do those r's, w's, and x's mean?



The first column is composed of strings that are 10 letters long. The first letter tells you what type of file it is - plain files are marked with a "-", directories are marked with a "d", and links are marked with a "l" . The other nine letters state the *permissions* on the file.

Letters two through four represent the permissions that the owner of the file has given him- or her-self. If there is an "r" as the second letter, the owner can read the file (i.e., they can view the contents, copy the file, or open the file with a program). If there is a "w" as the third letter, the owner of the file can write to the file - they can change it, add to it, delete it, etc. If there is an "x" as the fouth letter, the owner of the file can execute the file - the machine will assume that the file contains computer instructions, and will try to execute them at his or her command.

Now look at the fifth through seventh letters. Again, there may be r's, w's, and x's, and they mean the same thing, but now they refer to other member of the file's group. For example, if you are a member of the group students, you would be able to read studentsstuff.txt.

As for everyone else (i.e., users who aren't me and aren't in the students group), the eigth through tenth letters determine the permissions.

Now that we know what the letters mean, let's try changing a few. The command we need is called chmod (CHange MODe).

```
steel /N/fs1/clwolfe/Steel $ chmod u-r otherstuff.txt
steel /N/fs1/clwolfe/Steel $ ls -l otherstuff.txt
--w-r--r-- 1 clwolfe students 56 Feb 22 13:07 otherstuff.txt
steel /N/fs1/clwolfe/Steel $ more otherstuff.txt
otherstuff.txt: Permission denied
```

We'll cover chmod later. Let it suffice to say that chmod u-r removes read permission from the file, for the owner. You tell this from the ls -l. We then try to read the file using more, but we can't: we don't have permission to do so. Whenever you see the error message Permission Denied you know that you are doing something that someone doesn't want you to.

Let's turn back on the read permission for the owner on that file:

```
steel /N/fs1/clwolfe/Steel $ chmod u+r otherstuff.txt
```

But what about directories? Does it make sense for a directory to have, for example, execute permission?

Well, not really. Since a directory contains other files, it doesn't make sense for a directory to be executable. Instead, the permissions work like this:

- r - A user can list the contents of the directory.
- w - a user can create or delete files in the directory
- x - a user can cd into the directory

Commands for Working with Files

Here are brief summaries of common commands used when working with files. Only basic use is covered, and there are many, many features to these commands that are not listed here. If you want more information on a command, use the man command to learn more.

The ls command

The ls command lists the contents of a directory. Its general form is:
ls [options] [filename]

If filename is given, ls only gives information about that file.

If filename is a directory, ls lists the files in that directory.

Commonly used options include:

- a include files that begin with a dot (.) .
- F mark directories with a "/", links with a "@", and executables with a "*".
- l give all information, including permissions and ownership

The pwd Command

The pwd command displays the Present Working Directory (the one you are currently in).

The cd Command

The cd command changes the current directory. It lets you move between directories.
cd [pathname]

If given, pathname should be an absolute or relative pathname. You may use "~" to return to your home directory, or you can use ".." to move one level up.

If pathname is not given, cd will move you to your home directory.

The cp Command

The cp command copies a file.
cp [options] filename1 [filename2]

filename1 is the name of the file you wish to copy. It may be the name of a file, or a full path to a file.

filename2 is the name of the new copy. If filename2 is the name of an existing directory, filename1 will be copied to that directory with its original name. If filename2 is not given, filename1 will be copied to the current directory with its original name.

Commonly used options:

- r recursive - if filename1 is a directory, cp will copy it and all of its subdirectories.
- i interactive - cp will ask for confirmation from the user before overwriting any files.

The rm Command

The rm comamnd deletes a file. Use with care - there is no "undelete"!
rm [options] filename

Commonly used options:

- f don't ask for confirmation before deleting
- i ask for confirmation before deleting (on by default)

Use a related command, rmdir to delete directories:
rmdir directory

The chmod Command

The chmod command adjusts file permissions. Use ls -l to view the permissions. You can only change permissions on a file you own.

chmod what filename

filename is the name of the file that you would like to change permissions on.

what is a three-letter code specifying how you would like to adjust the permissions. The first letter is one of "u" (user), "g", (group) or "o" (other). The second letter is one of "-" (revoke permission) or "+" (grant permission). The third letter is one of "r" (read), "w" (write), or "x" (execute).

For example, g-r revokes read permission for groups, u+x grants execute permission for the owner (the user, in other words), and o-w revokes write permission for everybody else.

The quota Command

The quota command checks your quota, and lets you know if you are using more than your fair share of disk space.

quota

Your quota for your home directory is typically 10 or 20 megabytes.

See also:

- What are the baseline disk quotas for UITS accounts, and how do I request an increase?

The cat Command

The cat command reads a file and sends it to standard output, which usually means the terminal screen. Use cat to read a file quickly.

cat filename

cat is very similar to more, except cat does not pause when the screen is full.

Directories of Interest

Now that we understand filesystems, let's take a guided tour through the typical directories of a major UNIX machine, Steel. Yes, it is perfectly acceptable to go wandering throughout a UNIX installation - the file permissions have been set up so that it is impossible for you to hurt the setup of the machine. Do keep your hands to yourself, though.

This section should be considered extremely optional.

File Storage

First of all, let's look around the home directory area.

steel /N/fs1/clwolfe/Steel $ cd ..
steel /N/fs1/clwolfe $ ls -lF

```
drwx------   3 clwolfe  students     1024 Feb 11 16:15 Cobalt/
drwx--x--x   6 clwolfe  students     2048 May  3  1998 Copper/
drwx--x--x   6 clwolfe  students     1024 Jul  9  1999 Ezinfo/
drwx------   3 clwolfe  236          1024 Oct  5 07:50 SP/
drwxrwxrwx  10 clwolfe  students     2048 Feb 22 13:06 Steel/
lrwxrwxrwx   1 clwolfe  students        9 Nov  5 16:19 WWW -> Steel/www/
drwx------   3 clwolfe  students     1024 Apr 22  1996 Zinc/
```

From your home directory, cd up one level and do an ls -lF. Notice that there are a number of directories here, and they're all owned by you. There should be a directory for each UITS-owned UNIX machine that you have an account on.

Each one of these is your home directory on each of the machines. You may be thinking "But wait - I was on Steel!!". True, you are logged in on Steel, but your home directory actually lives on another machine entirely, known as the NFS server. When you log in to a machine, your actual pwd location is on another machine entirely. This makes life easier for you, because:

- To move files from one machine to another, you need only use the cp command to copy files
- It is very easy for UITS to add more disk space to the NFS server without affecting Steel or another UITS machine.

Also notice the WWW symbolic link. You can tell that it is a symbolic link because its permission string begins with an "l". The output of ls -lF tells us that WWW actually just points to Steel/WWW on my account. This is the standard setup for Personal Home Pages at IU - if you have web pages on php.indiana.edu, the web pages are actually on the NFS server, in your Steel/WWW account. This is why you may hear someone say "I'm editing my web pages on Steel".

Let's go up another directory and look around.

steel /N/fs1/clwolfe $ cd ..
steel /N/fs1 $ ls -F

```
Lithium/      avarga/       dlauer/       hagstrom/     jsummane/     lwan/         pbsmith/      tsmedley/
M/            basu/         dmullens/     hpss/         jtownsen/     mahmed/       pbucklin/     tso/
NAME          bdf.log       doconner/     hshahran/     jtzeng/       mfedder/      peng/         tsulanke/
OLD-10FEB00/  bgygi/        dpizzo/       huber/        jvillaci/     mhankins/     pungitor/     valid/
OLD-13NOV99/  bknick/       dsoruco/      huntr/        jwelsh/       mkerr/        quinnjf/      weaver/
OLD-18MAR98/  bofferle/     dterret/      jbreeden/     jwessel/      mkirkpat/     quotas        whchuang/
OLD-25OCT99/  briggsc/      efschnei/     jburger/      jwmcfarl/     mkohli/       recollin/     wlyon/
OLD-27OCT99/  bsamuel/      ejolson/      jjjohnst/     katkins/      mkoljati/     rlambrec/     wpendlet/
OLD-7JUL99/   bwolford/     england/      jkell/        kdhunt/       morrison/     sakohler/     wpritcha/
OLD-MAR98/    carmicha/     engs/         jlstauff/     kidd/         mpanders/     shashaan/     xuyang/
Root/         cdworlan/     epourman/     jmoline/      klubell/      mperera/      shyu/         yiwzheng/
aaa/          clwolfe/      erpeters/     jnoxon/       kredding/     mufson/       sirish/       yungchen/
adonderi/     ctwardy/      fin/          joalexan/     kyukim/       nabrigg/      smin/         yzheng/
akepes/       davwils/      flury/        jomarqua/     lance/        nblair/       ssimms/
akinley/      dgetz/        ganli/        jorcrum/      lchansen/     netbug/       test
amplumme/     dholbroo/     gaparker/     josblack/     lhermesc/     niju/         tjvincen/
arcinfo/      dholian/      gdrukier/     jourbans/     llthomps/     npchrist/     tkresser/
asarkiss/     dkabilli/     geduld/       jruan/        lost+found/   nsmutzle/     tkressr/
ashlab/       dkberry/      gilbertd/     jskocili/     lwagers/      ntakebay/     trhall/
```

As you can see, there are a lot of users on the NFS server! Each one of these directories is the home directory of a user by that name. Look for yours in the list you see in your telnet window. Don't try to cd into someone else's directory (unless they ask you to) - you'll just get a Permission Denied error (which is comforting).

Let's go up another level.

steel /N/fs1 $ cd ..
steel /N $ ls -F

```
fs1/   fs12/  fs15/  fs18/  fs20/  fs23/  fs26/  fs29/  fs31/  fs34/  fs37/  fs5/   fs8/
fs10/  fs13/  fs16/  fs19/  fs21/  fs24/  fs27/  fs3/   fs32/  fs35/  fs38/  fs6/   fs9/
fs11/  fs14/  fs17/  fs2/   fs22/  fs25/  fs28/  fs30/  fs33/  fs36/  fs4/   fs7/   u/
```

Of course, fs1 is just one of the areas for home directories. The other directories listed here also have home directories in them. UITS currently has several *thousand* users - though a small fraction are logged in at any given time.

What if you have some really big files that you need to work with? Wouldn't it be nice to have a large playground, free of quotas? Let's go visit the /scr directory.

steel /N $ cd /scr

steel /scr $ cd clwolfe

/scr/username is the place to unpack large files, or play with data from tapes, or download large files from the Internet. Each user has their own directory under /scr on Steel. This directory does not have a quota, so you may place large files there temporarily. The files are deleted after not being used for 10 consecutive days, to prevent users from abusing the facility.

Where to Find Binaries

If you are using a certain command, you may need to know where the command is installed. Let's look at some of the typical places.

steel /tmp $ cd /usr/bin

In /usr/bin, you will find all of the "standard" UNIX programs. For example, you might want to look in /usr/bin for any utility that you know comes with Solaris (Steel's flavor of UNIX). Anything you find here *should* be installed on any other Solaris machine.

steel /usr/bin $ cd /usr/local/bin

However, Steel is not just any Solaris machine; it's Steel, and has a unique personality all its own. This means that there are extra utilities, UITS-written programs, and other interesting programs in /usr/local/bin. If you are running a program from /usr/local/bin, you have no gaurentee that the program will be configured the same way, be installed, or even exist on any other UNIX machine. That's why /usr/bin and /usr/local/bin are seperate.

steel /usr/local/bin $ cd /usr/local

In fact, everything under /usr/local represents additional features that the sysadmin team has installed on Steel for your benefit. Some programs might store library files here, or additional documentation in /usr/local. Directories under /usr/local are generally named after the application name. /usr/local is a lot like the "Program Files" directory on Windows.

Really large application suites are generally installed under either /opt or /opt/local.

Where to Find System Information

You might also be interested in where system configuration and logging takes place. Most of this occurs in /etc. There's not much point in poking around there, unless you know what you're looking for.

Processes

Now we know about files, commands, and shells. What more could UNIX possibly have to offer?

UNIX was designed at a time when computers were used largely for crunching huge amounts of numbers. One of the nicest features of UNIX is its ability to *multitask* - to do more than one thing at a time. A single task is called a *process*.

The ps Command

Whenever you are logged in, you are executing at least one process. Let's get a list of all the processes I am currently running:

steel /N/fs1/clwolfe/Steel $ ps

```
   PID TTY      TIME CMD
 16931 pts/38   0:00 bash
```

I'm running one process, bash. bash is my shell, so naturally it is running. The output from ps includes the following information:

- PID - the Process ID of the process. Every program that is running has a PID, and you use this number to identify them.
- TTY - the Terminal that the process is running on. This information generally is not very helpful.
- TIME - the time that the process has been running.
- CMD - The name of the command being executed.

ps has a great many options. Unfortunately, every flavor of UNIX uses a different set of options to ps. The following options are fairly standard:

- a - print all running processes, not just yours
- f - generate a "full" listing, including username and command line

Background Processes

UNIX makes a very strong distinction between processes that are running in the *foreground* and those that are running in the *background*. A foreground process can talk to your terminal, and you can talk back using the keyboard. A background process is running, but it has no way to communicate with you directly.

This is most useful when you have a large task to complete (say, calculating Pi to a million digits). We'll use the sleep command as an example. Let's try this:

steel /N/fs1/clwolfe/Steel $ sleep 3

The sleep command takes one argument, the number of seconds to sleep for. sleep then does nothing for that long.

It's pretty boring to sit and watch as sleep just sits there. We will now run sleep as a background process, which will allow us to go about our business.

steel /N/fs1/clwolfe/Steel $ sleep 30 &
[1] 22098
steel /N/fs1/clwolfe/Steel $ ps

```
  PID TTY      TIME CMD
16931 pts/38   0:00 bash
22098 pts/38   0:00 sleep
```

First of all, we use another shell meta-character, & to indicate that the command should be run in the background. You can use the & with any command.

Next, notice that the first command returns a number to us: the Process ID (PID) of the backgrounded process.

Then we use the ps command to confirm that there are now two processes running: bash and sleep.

Finally, after at least 30 seconds have gone by, the next time you execute a command or press Return, you should see somethinglike this:

steel /N/fs1/clwolfe/Steel $
[1]+ Done sleep 30

This is the shell reporting that the command completed successfully.

Killing a Process

Not all processes complete successfuly. You may find yourself stuck with a process that seems to have stopped working, or perhaps you have changed your mind about running the process.

For example, we'll start a new sleep process and kill it.

steel /N/fs1/clwolfe/Steel $ sleep 300 &
[1] 23360
steel /N/fs1/clwolfe/Steel $ ps

```
  PID TTY      TIME CMD
23360 pts/38   0:00 sleep
16931 pts/38   0:00 bash
```

steel /N/fs1/clwolfe/Steel $ kill 23360
steel /N/fs1/clwolfe/Steel $ ps

```
  PID TTY      TIME CMD
```

```
 16931 pts/38   0:00 bash
[1]+  Terminated              sleep 300
```

First, we start a new sleep process. Then we check that it is running with ps. We note the PID, then issue a kill command, passing the PID as an argument. Then we check again using ps. At the same time, the shell prints out a line indicating that our sleep process has been Terminated.

Common Tasks

Now that we have all of the basic concepts down, we take a look at doing some of the most common tasks in UNIX.

Text Editing

The most common task you will perform under UNIX will almost certainly be text editing. When you need a file that contains commands for a program, data, or practically anything else, you'll need to know how to edit a text file.

A text file is just a plain file that contains text. It does not typically contain any formatting commands, other than tabs and newlines. A text file can be easily viewed using more or cat.

There are many different text editors available. Most UNIX users develop a rabid attachment to their favorite text editor. My personal favorite is pico, a very easy-to-use text editor. vi and emacs are other very common editors.

To run pico:

steel /N/fs1/clwolfe/Steel $ pico

Your terminal should look like this:

pico is based on the pine email reader, which is used on the Shakespeare systems. If you can use pine, pico should give you no trouble.

- What is Pico?
- Emacs Quick Reference Guide
- vi Basics

Checking Mail

Electronic Mail originated with UNIX, which may explain a lot about how easy it is to use.

As a user of the Steel cluster, you may receive email there. You can check your email on steel by using pine, a mail program. pine is the same program used on the Shakespeare email servers.

- What is Pine?

Moving Files between Machines

Occasionally, you will need to move files to or from the UNIX machine.

If your files are currently on a Windows or Macintosh machine, you should use Hummingbird (Windows) or Fetch (Macintosh). Follow these instructions to move files to or from a UNIX machine:

- How do I use Fetch?
- How do I use Hummingbird FTP?

If both machines are UITS UNIX machines (such as the SP, Cobalt, or Steel), you can simply copy the files:

steel /N/fs1/clwolfe/Steel $ cp mystuff.txt ../Cobalt

This would copy the file mystuff.txt from my Steel account to my Cobalt account.

If you are wanting to move files between *any* two UNIX machines, you should use the ftp UNIX command. This is the same program as Fetch or Hummingbird, except that it is command-line based.

- What is FTP, and how do I use it to transfer files?

Writing Scripts

UNIX users are encouraged to write their own programs. The simplest kind of program is called a *shell script*. A shell script is simply a text file containing one UNIX command per line.

The first thing to do is to create a place for all of your scripts.

steel /N/fs1/clwolfe/Steel $ mkdir bin
steel /N/fs1/clwolfe/Steel $ cd bin

The standard location for user's scripts is in the bin directory, in your home directory. This pathname should already be in your $PATH environment variable by default. This means that no matter where you are, you will be able to execute your own scripts as if they were UNIX commands.

The next step is to create a text file with the commands would like to execute. Use any text editor (e.g., pico) to do this.

For example, you could write a script that prints out the words "Hello World!", by creating a text file with the following two lines:

```
#/bin/sh
echo "Hello World!"
```

Save this file in bin with the filename helloworld.

Next, make the file executable by changing its permissions:

steel /N/fs1/clwolfe/Steel/bin $ chmod u+x helloworld
steel /N/fs1/clwolfe/Steel/bin $ ls -l helloworld

```
-rwx------   1 clwolfe  iustaff        32 Feb 23 10:47 helloworld
```

As you can see, the file is now executable. Execute your script:

steel /N/fs1/clwolfe/Steel/bin $ helloworld
Hello World!

Of course, you can do a lot more than print out a simple message. Shell programming is very powerful. Try some of these examples:

- In Unix, how can I issue batches of non-interactive ftp commands?
- In Unix, how can I replace a single string in a large number of files?
- In Unix, how do I rename "*.foo" to "*.bar", or change filenames to lowercase?

Modifying Your Environment

It is often neccesary to modify your environment in UNIX. This means that you would like to make a permanent change to the configuration of your shell - for example, to add another path to the $PATH variable. Or perhaps you need to run an application that requires an environment variable to be set (GAUSS, for example).

First, let's find out where shell configuraion is done.

steel /N/fs1/clwolfe/Steel $ ls -alF

```
-rw-------   1 clwolfe  students     1428 Oct 28 08:33 .Xauthority
-rw-------   1 clwolfe  students     4324 Feb 22 19:58 .bash_history
-rw-------   1 clwolfe  students     3027 Mar 10  1999 .cshrc
-rw-------   1 clwolfe  students      473 May  5  1999 .history
-rw-------   1 clwolfe  students     1808 Mar 10  1999 .login
-rw-------   1 clwolfe  iustaff     10602 Feb 23 10:05 .pinerc
-rw-------   1 clwolfe  students     4228 May  5  1999 .profile
-rw-------   1 clwolfe  students       12 Mar 10  1999 .sh_history
-rw-r-----   1 clwolfe  students       80 May 17  1999 .signature
-rw-------   1 clwolfe  students      207 May 11  1999 .spssrc
-rw-------   1 clwolfe  students     1091 May 11  1999 .xmaplev5rc
drwx------   2 clwolfe  students     1024 Feb 22 11:56 Mail/
drwx------   2 clwolfe  iustaff        96 Feb 23 10:47 bin/
drwx------   2 clwolfe  students       96 Feb 22 11:14 stuff/
drwx--x--x   8 clwolfe  students     1024 Feb 10 14:05 www/
```

Notice all of the files that start with a "." . These are configuration files for various programs. Each file is a plain text file that you could edit with pico. The various shells use different files for configuration.

- In Unix, what "dot" files do the various shells use?

Your shell executes these files as if they were shell scripts when you log in. This sets up your environment variables and configures any shell features.

You may edit the files like any other shell script, using pico.

Submitting Batch Jobs

Batch processing is usually done on the Research SP, rather than on Steel. The Research SP is designed for batch processing.

Recall from our discussion of processes that there are foreground and background processes. A *batch job* is a background process that is run at a later time, when it is conveneient for the system. Batch jobs are usually used for any CPU-intensive task (i.e., any task that requires a large amount of number-crunching). Typically, an application package is used (such as Maple, SPSS, Matlab, SAS, etc.). Submitting a batch process is a three step process:

1. Write and debug the scripts you will use with your application. This might mean writing a syntax file for SPSS, an m-file for Matlab, etc.
2. Write a *batch script* - a script that the *batch scheduler* will use to execute your job.
3. Submit the batch script to the batch scheduler.

Step 1 is entirely dependent on what application you are using. Presumably, you know what you are doing for this step.

Step 2 is straightforward. The batch scheduler on the Research SP is LoadLeveler, another fine product from IBM. All you need to do is create a text file that looks like this:

```
#@ class=m
#@ group=standard
#@ requirements=(Feature=="maple")
#@ initialdir=/N/fs1/clwolfe/SP
#@ output=research.out
#@ error=research.err
#@ queue
maple -f -q <research
```

This is a script to run a Maple job. The first line declares that it is a job of class "m" - i.e., a math job. The third line indicates that Maple must be installed on the node that it will be run on. The fourth line sets the working directory for the job. The fifth and six lines direct STDOUT and STDERR to two files (since batch processing is non-interactive, you will lose this information if you don't save it). The seventh line tells LoadLeveler to queue the job for later processing.

Pay special attention to the last line. This is simply a command, as you would write at the command prompt, that executes Maple and feeds in the file named research, where I have (presumably) stored the Maple commands to be executed.

There are sample scripts available on the NFS server (i.e., they are available from any UITS UNIX machine, such as Steel, the Research SP, or Cobalt) in the directory /N/u/statmath/SP/scripts. You'll find a script there for every major application.

Save your batch script in your home directory, and give it a name you will remember.

The third step is to submit the batch job to the batch scheduler. If the batch script was named myjob, you would use this command:

SP % llsubmit myjob

To see a listing of the jobs that are currently in the queue, use

SP % llq

See also:

- How do I run stat/math jobs, including SAS and SPSS, under AIX loadleveler?
- The Research SP

Using Tapes

Magnetic tapes are often used to stored large amounts of rarely used data (eg, the raw data from the 1990 census). UITS has tape-handling facilities on Steel.

See the Stat/Math Document Using Tapes on Steel .

Further Reading

UNIX is a topic that is extremely broad and deep. This document is intended to be a basic introduction. Here are some resources that you may find helpful in the future:

- UNIX Workstation Support Group
- Do you have a brief synopsis of Unix history?

- In Unix, what do some obscurely named commands stand for?
- Using Math Software under UNIX

Last modified: Wednesday, 07-Jun-2000 15:26:13 EST
URL `/~statmath/support/byos/unix/gettingstarted/printable.html`