

Magit User Manual

for version 1.4

The Magit Project Developers

Copyright © 2008-2015 The Magit Project Developers

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Table of Contents

1	Introduction	1
2	Acknowledgments	2
3	Sections	3
4	Status	4
5	Untracked files	5
6	Staging and Committing	6
7	History	7
8	Reflogs	8
9	Commit Buffer	9
10	Diffing	10
11	Tagging	11
12	Resetting	12
13	Stashing	13
14	Branches and Remotes	14
14.1	Branches Popup.....	14
14.2	Remotes Popup.....	14
14.3	Branches in the Branch Manager.....	14
14.4	Remotes in the Branch Manager.....	15
15	Wazzup	16
16	Merging	17

17	Rebasing	18
18	Interactive Rebasing	19
19	Rewriting	20
20	Pushing and Pulling	21
21	Submodules	22
22	Bisecting	23
23	Finding commits not merged upstream	24
24	Magit Extensions	25
24.1	Activating extensions	25
24.2	Interfacing with Subversion	25
24.3	Interfacing with Topgit	25
24.4	Interfacing with StGit	26
25	Using Git Directly	27
Appendix A	GNU Free Documentation License	28

1 Introduction

With Magit, you can inspect and modify your Git repositories with Emacs. You can review and commit the changes you have made to the tracked files, for example, and you can browse the history of past changes. There is support for cherry picking, reverting, merging, rebasing, and other common Git operations.

Magit is not a complete interface to Git; it just aims to make the most common Git operations convenient. Thus, Magit will likely not save you from learning Git itself.

This manual provides a tour of many Magit features. It isn't an introduction to version control in general, or to Git in particular.

The main entry point to Magit is `M-x magit-status`, which puts you in Magit's status buffer. You will be using it frequently, so it is probably a good idea to globally bind `magit-status` to a key of your choice.

In addition to the status buffer, Magit will also create buffers that show lists of commits, buffers with diffs, and other kinds of buffers. All these buffers are in a mode derived from `magit-mode` and have the similar key bindings. Not all commands make sense in all contexts, but a given key will do the same thing in different Magit buffers.

Naturally, Magit runs the `git` command to do most of the work. The `*magit-process*` buffer contains the transcript of the most recent command. You can switch to it with `$`.

2 Acknowledgments

Our thank goes to all current and past contributors, Marius Vollmer who started the project, and all retired and current maintainers, Phil Jackson, Peter J. Weisberg, Rmi Vanicat, Nicolas Dubebout, Yann Hodique, and Jonas Bernoulli.

For a full list of contributors, see the `AUTHORS.md` file at the top-level directory of this distribution or at [AUTHORS.md](#).

3 Sections

All Magit buffers are structured into nested 'sections'. These sections can be hidden and shown individually. When a section is hidden, only its first line is shown and all its children are completely invisible.

The most fine-grained way to control the visibility of sections is the *TAB* key. It will toggle the current section (the section that contains point) between being hidden and being shown.

Typing *S-TAB* toggles the visibility of the children of the current section. When all of them are shown, they will all be hidden. Otherwise, when some or all are hidden, they will all be shown.

The digit keys *1*, *2*, *3*, and *4* control the visibility of sections based on levels. Hitting *2*, for example, will show sections on levels one and two, and will hide sections on level 3. However, only sections that are a parent or child of the current section are affected.

For example, when the current section is on level 3 and you hit *1*, the grand-parent of the current section (which is on level one) will be shown, and the parent of the current section (level 2) will be hidden. The visibility of no other section will be changed.

This sounds a bit complicated, but you'll figure it out.

Using *M-1*, *M-2*, *M-3*, and *M-4* is similar to the unmodified digits, but now all sections on the respective level are affected, regardless of whether or not they are related to the current section.

For example, *M-1* will only show the first lines of the top-level sections and will hide everything else. Typing *M-4* on the other hand will show everything.

Because of the way the status buffer is set up, some changes to section visibility are more common than others. Files are on level 2 and diff hunks are on level 4. Thus, you can type *2* to collapse the diff of the current file, and *M-2* to collapse all files. This returns the status buffer to its default setup and is a quick way to unclutter it after drilling down into the modified files.

Because *2* and *M-2* are so common in the status buffer, they are bound to additional, more mnemonic keys: *M-h* (hide) and *M-H* (hide all). Likewise *4* and *M-4* are also available as *M-s* (show) and *M-S* (show all).

In other buffers than the status buffer, *M-h*, *M-H*, *M-s*, and *M-S* might work on different levels than on 2 and 4, but they keep their general meaning: *M-H* hides all detail, and *M-S* shows everything.

4 Status

Running `M-x magit-status` displays the main interface of Magit, the status buffer. You can have multiple status buffers active at the same time, each associated with its own Git repository.

When invoking `M-x magit-status` from within a Git repository, it will switch to the status buffer of that repository. Otherwise, it will prompt for a directory. With a prefix argument, it will always prompt.

You can set `magit-repo-dirs` to customize how `magit-status` asks for the repository to work on. When `magit-repo-dirs` is nil, `magit-status` will simply ask for a directory.

If you specify a directory that is not a Git repository, `M-x magit-status` will offer to initialize it as one.

When `magit-repo-dirs` is not nil, it is treated as a list of directory names, and `magit-status` will find all Git repositories in those directories and offer them for completion. (Magit will only look `magit-repo-dirs-depth` levels deep, however.)

With two prefix arguments, `magit-status` will always prompt for a raw directory.

Thus, you would normally set `magit-repo-dirs` to the places where you keep most of your Git repositories and switch between them with `C-u M-x magit-status`. If you want to go to a repository outside of your normal working areas, or if you want to create a new repository, you would use `C-u C-u M-x magit-status`.

You need to explicitly refresh the status buffer when you have made changes to the repository from outside of Emacs. You can type `g` in the status buffer itself, or just use `M-x magit-status` instead of `C-x b` when switching to it. You also need to refresh the status buffer in this way after saving a file in Emacs.

The header at the top of the status buffer shows a short summary of the repository state: where it is located, which branch is checked out, etc. Below the header are a number of sections that show details about the working tree and the staging area. You can hide and show them as described in the previous section.

The first section shows *Untracked files*, if there are any. See [Chapter 5 \[Untracked files\]](#), [page 5](#) for more details.

The next two sections show your local changes. They are explained fully in the next chapter, [Chapter 6 \[Staging and Committing\]](#), [page 6](#).

If the current branch is associated with a remote tracking branch, the status buffer shows the differences between the current branch and the tracking branch. See [Chapter 20 \[Pushing and Pulling\]](#), [page 21](#) for more information.

During a history rewriting session, the status buffer shows the *Pending changes* and *Pending commits* sections. See [Chapter 19 \[Rewriting\]](#), [page 20](#) for more details.

5 Untracked files

Untracked files are shown in the *Untracked files* section.

You can add an untracked file to the staging area with **s**. If point is on the *Untracked files* section title when you hit **s**, all untracked files are staged.

Typing **C-u S** anywhere will also stage all untracked files, together with all changes to the tracked files.

You can instruct Git to ignore them by typing **i**. This will add the filename to the `.gitignore` file. Typing **C-u i** will ask you for the name of the file to ignore. This is useful to ignore whole directories, for example. In this case, the minibuffer's future history (accessible with **M-n**) contains predefined values (such as wildcards) that might be of interest. If prefix argument is negative (for example after typing **C-- i**), the prompt proposes wildcard by default. The **I** command is similar to **i** but will add the file to `.git/info/exclude` instead.

To delete an untracked file forever, use **k**. If point is on the *Untracked files* section title when you hit **k**, all untracked files are deleted.

6 Staging and Committing

Committing with Git is a two step process: first you add the changes you want to commit to a 'staging area' or 'index', and then you commit them to the repository. This allows you to only commit a subset of the changes in the working tree. If you are not familiar with this concept yet, then you should change that as soon as possible using one of the fine Git tutorials. If you don't, then Git and by extension Magit will seem rather strange.

Magit shows uncommitted changes in two sections, depending on whether the changes have been staged yet. The *Staged changes* section shows the changes that will be included in the next commit, while the *Unstaged changes* section shows the changes that will be left out.

To move an unstaged hunk into the staging area, move point into the hunk and type `s`. Likewise, to unstage a hunk, move point into it and type `u`. If point is in a diff header when you type `s` or `u`, all hunks belonging to that diff are moved at the same time.

Currently it is only possible to stage from the status buffer. Staging and unstaging from diff buffers that show unstaged and staged changes is not possible yet.

If the region is active when you type `s` or `u`, only the changes in the region are staged or unstaged. (This works line by line: if the beginning of a line is in the region it is included in the changes, otherwise it is not.)

To change the size of the hunks, you can type `+` or `-` to increase and decrease, respectively. Typing `0` will reset the hunk size to the default.

Typing `C-u s` will ask you for a name of a file to be staged, for example to stage files that are hidden.

To move all hunks of all diffs into the staging area in one go, type `S`. To unstage everything, type `U`.

Typing `C-u S` will stage all untracked files in addition to the changes to tracked files.

You can discard uncommitted changes by moving point into a hunk and typing `k`. The changes to discard are selected as with `s` and `u`.

Before committing, you should write a short description of the changes.

Type `c c` to pop up a buffer where you can write your change description. Once you are happy with the description, type `C-c C-c` in that buffer to perform the commit.

If you want to write changes in a `ChangeLog` file, you can use `C-x 4 a` on a diff hunk.

Typing `c c` when the staging area is unused is a special situation. Normally, the next commit would be empty, but you can configure Magit to do something more useful by customizing the `magit-commit-all-when-nothing-staged` variable. One choice is to instruct the subsequent `C-c C-c` to commit all changes. Another choice is stage everything at the time of hitting `c c`.

Typing `M-n` or `M-p` will cycle through the `log-edit-comment-ring`, which will have your previous log messages. This is particularly useful if you have a hook that occasionally causes git to refuse your commit.

To abort a commit use `C-c C-k`. The commit message is saved and can later be retrieved in the commit message buffer using `M-n` and `M-p`.

Typing `C` will also pop up the change description buffer, but in addition, it will try to insert a `ChangeLog`-style entry for the change that point is in.

7 History

To show the repository history of your current head, type `l l`. A new buffer will be shown that displays the history in a terse form. The first paragraph of each commit message is displayed, next to a representation of the relationships between commits.

To show the repository history between two branches or between any two points of the history, type `l r l`. You will be prompted to enter references for starting point and ending point of the history range; you can use auto-completion to specify them. A typical use case for ranged history log display would be `l r l master RET new-feature RET` that will display commits on the new-feature branch that are not in master; these commits can then be inspected and cherry-picked, for example.

More thorough filtering can be done by supplying `l` with one or more suffix arguments, as displayed in its popup. `=g` ('Grep') for example, limits the output to commits of which the log message matches a specific string/regex.

Typing `l L` (or `l C-u L`) will show the log in a more verbose form.

Magit will show only `magit-log-cutoff-length` entries. `e` will show twice as many entries. `C-u e` will show all entries, and given a numeric prefix argument, `e` will add this number of entries.

You can move point to a commit and then cause various things to happen with it. (The following commands work in any list of commits, such as the one shown in the *Unpushed commits* section.)

Typing `RET` will pop up more information about the current commit and move point into the new buffer. See [Chapter 9 \[Commit Buffer\], page 9](#). Typing `SPC` and `DEL` will also show the information, but will scroll the new buffer up or down (respectively) when typed again.

Typing `a` will apply the current commit to your current branch. This is useful when you are browsing the history of some other branch and you want to 'cherry-pick' some changes from it. A typical situation is applying selected bug fixes from the development version of a program to a release branch. The cherry-picked changes will not be committed automatically; you need to do that explicitly.

Typing `A` will cherry-pick the current commit and will also commit the changes automatically when there have not been any conflicts.

Typing `v` will revert the current commit. Thus, it will apply the changes made by that commit in reverse. This is obviously useful to cleanly undo changes that turned out to be wrong. As with `a`, you need to commit the changes explicitly.

Typing `C-w` will copy the sha1 of the current commit into the kill ring.

Typing `=` will show the differences from the current commit to the *marked* commit.

You can mark the current commit by typing `..`. When the current commit is already marked, typing `.` will unmark it. To unmark the marked commit no matter where point is, use `C-u ..`

Some commands, such as `=`, will use the current commit and the marked commit as implicit arguments. Other commands will offer the marked commit as a default when prompting for their arguments.

8 Reflogs

You can use `l h` and `l H` to browse your *reflog*, the local history of changes made to your repository heads. Typing `H` will ask for a head, while `l h` will show the reflog of `HEAD`.

The resulting buffer is just like the buffer produced by `l l` and `l L` that shows the commit history.

9 Commit Buffer

When you view a commit (perhaps by selecting it in the log buffer, [Chapter 7 \[History\], page 7](#)), the “commit buffer” is displayed, showing you information about the commit and letting you interact with it.

By placing your cursor within the diff or hunk and typing `a`, you can apply the same patch to your working copy. This is useful when you want to copy a change from another branch, but don’t necessarily want to cherry-pick the whole commit.

By typing `v` you can apply the patch in reverse, removing all the lines that were added and adding all the lines that were removed. This is a convenient way to remove a change after determining that it introduced a bug.

If the commit message refers to any other commits in the repository by their unique hash, the hash will be highlighted and you will be able to visit the referenced commit either by clicking on it or by moving your cursor onto it and pressing `RET`.

The commit buffer maintains a history of the commits it has shown. After visiting a referenced commit you can type `C-c C-b` to get back to where you came from. To go forward in the history, type `C-c C-f`. There are also `[back]` and `[forward]` buttons at the bottom of the buffer.

10 Diffing

Magit typically shows diffs in the “unified” format.

In any buffer that shows a diff, you can type `e` anywhere within the diff to show the two versions of the file in Ediff. If the diff is of a file in the status buffer that needs to be merged, you will be able to use Ediff as an interactive merge tool. Otherwise, Ediff will simply show the two versions of the file.

To show the changes from your working tree to another revision, type `d`. To show the changes between two arbitrary revisions, type `D`.

You can use `a` within the diff output to apply the changes to your working tree. As usual when point is in a diff header for a file, all changes for that file are applied, and when it is in a hunk, only that hunk is. When the region is active, the applied changes are restricted to that region.

Typing `v` will apply the selected changes in reverse.

11 Tagging

Typing `t t` will make a lightweight tag. Typing `t a` will make an annotated tag. It will put you in the normal `*magit-log-edit` buffer for writing commit messages, but typing `C-c` `C-c` in it will make the tag instead. This is controlled by the `Tag` field that will be added to the `*magit-log-edit*` buffer. You can edit it, if you like.

12 Resetting

Once you have added a commit to your local repository, you can not change that commit anymore in any way. But you can reset your current head to an earlier commit and start over.

If you have published your history already, rewriting it in this way can be confusing and should be avoided. However, rewriting your local history is fine and it is often cleaner to fix mistakes this way than by reverting commits (with `v`, for example).

Typing `x` will ask for a revision and reset your current head to it. No changes will be made to your working tree and staging area. Thus, the *Staged changes* section in the status buffer will show the changes that you have removed from your commit history. You can commit the changes again as if you had just made them, thus rewriting history.

Typing `x` while point is in a line that describes a commit will offer this commit as the default revision to reset to. Thus, you can move point to one of the commits in the *Unpushed commits* section and hit `x RET` to reset your current head to it.

Type `X` to reset your working tree and staging area to the most recently committed state. This will discard your local modifications, so be careful.

You can give a prefix to `x` if you want to reset both the current head and your working tree to a given commit. This is the same as first using an unprefixed `x` to reset only the head, and then using `X`.

13 Stashing

You can create a new stash with `z z`. Your stashes will be listed in the status buffer, and you can apply them with `a` and pop them with `A`. To drop a stash, use `k`.

With a prefix argument, both `a` and `A` will attempt to reinstate the index as well as the working tree from the stash.

Typing `z -k z` will create a stash just like `z z`, but will leave the changes in your working tree and index. This makes it easier to, for example, test multiple variations of the same change.

If you just want to make quick snapshots in between edits, you can use `z s`, which automatically enters a timestamp as description, and keeps your working tree and index intact by default.

You can visit and show stashes in the usual way: Typing `SPC` and `DEL` will pop up a buffer with the description of the stash and scroll it, typing `RET` will move point into that buffer. Using `C-u RET` will move point into that buffer in other window.

14 Branches and Remotes

The current branch is indicated in the header of the status buffer. If this branch is tracking a remote branch, the latter is also indicated.

Branches and remotes can be manipulated directly with a popup menu or through the branch manager. Using the popup menu allows you to quickly make changes from any magit buffer. The branch manager is a separate buffer called `*magit-branches*`. It displays information about branches and remotes and offers a local key map for shorter key bindings. The two interaction methods are described in more details below.

14.1 Branches Popup

Typing `b` will display a popup menu to manipulate branches.

You can switch to a different branch by typing `b b`. This will immediately checkout the branch into your working copy, so you shouldn't have any local modifications when switching branches.

If you try to switch to a remote branch, Magit will offer to create a local tracking branch for it instead. This way, you can easily start working on new branches that have appeared in a remote repository.

Typing `b b` while point is at a commit description will offer that commit as the default to switch to. This will result in a detached head.

To create a new branch and switch to it immediately, type `b c`.

To delete a branch, type `b k`. If you're currently on that branch, Magit will offer to switch to the 'master' branch.

Typing `b r` will let you rename a branch. Unless a branch with the same name already exists, obviously...

Deleting a branch is only possible if it's already fully merged into HEAD or its upstream branch. Unless you type `b C-u k`, that is. Here be dragons...

Typing `b v` will launch the branch manager.

14.2 Remotes Popup

Typing `M` will display a popup menu to manipulate remotes.

To add a new remote, type `M a`.

To delete a remote type `M k`.

Typing `M r` will let you rename a remote.

14.3 Branches in the Branch Manager

In the branch manager, each branch is displayed on a separate line. The current local branch is marked by a "*" in front of the name. Remote branches are grouped by the remote they come from.

If a local branch tracks a remote branch some extra information is printed on the branch line. The format is the following: "`<branch> [<remote-branch> <remote>: ahead <a>,`

behind]”. “<remote-branch>” is omitted if it is identical to “<branch>”. “ahead” and “behind” information are only displayed if necessary.

To check out a branch, move your cursor to the desired branch and press *RET*.

Typing *c* will create a new branch.

Typing *k* will delete the branch in the current line, and *C-u k* deletes it even if it hasn't been merged into the current local branch. Deleting works for both local and remote branches.

Typing *r* on a branch will rename it.

Typing *T* on a local branch, changes which remote branch it tracks.

14.4 Remotes in the Branch Manager

In the branch manager, each remote is displayed on a separate line. The format is the following “<remote> (<url>, <push-url>)”. “<push-url>” is omitted if it is not set. The associated branches are listed under this line.

Typing *a* will add a new remote.

Typing *k* will delete the remote in the current line.

Typing *r* on a remote will rename it.

15 Wazzup

Typing `w` will show a summary of how your other branches relate to the current branch.

For each branch, you will get a section that lists the commits in that branch that are not in the current branch. The sections are initially collapsed; you need to explicitly open them with `TAB` (or similar) to show the lists of commits.

When point is on a *N unmerged commits in ...* title, the corresponding branch will be offered as the default for a merge.

Hitting `i` on a branch title will ignore this branch in the wazzup view. You can use `C-u w` to show all branches, including the ignored ones. Hitting `i` on an already ignored branch in that view will unignore it.

16 Merging

Magit offers two ways to merge branches: manual and automatic. A manual merge will apply all changes to your working tree and staging area, but will not commit them, while an automatic merge will go ahead and commit them immediately.

Type *m m* to initiate merge.

After initiating a merge, the header of the status buffer might remind you that the next commit will be a merge commit (with more than one parent). If you want to abort a manual merge, just do a hard reset to HEAD with *X*.

Merges can fail if the two branches you want to merge introduce conflicting changes. In that case, the automatic merge stops before the commit, essentially falling back to a manual merge. You need to resolve the conflicts for example with *e* and stage the resolved files, for example with *S*.

You can not stage individual hunks one by one as you resolve them, you can only stage whole files once all conflicts in them have been resolved.

17 Rebasing

Typing `R` in the status buffer will initiate a rebase or, if one is already in progress, ask you how to continue.

When a rebase is stopped in the middle because of a conflict, the header of the status buffer will indicate how far along you are in the series of commits that are being replayed. When that happens, you should resolve the conflicts and stage everything and hit `R c` to continue the rebase. Alternatively, hitting `c` or `C` while in the middle of a rebase will also ask you whether to continue the rebase.

Of course, you can initiate a rebase in any number of ways, by configuring `git pull` to rebase instead of merge, for example. Such a rebase can be finished with `R` as well.

18 Interactive Rebasing

Typing *E* in the status buffer will initiate an interactive rebase. This is equivalent to running `git rebase --interactive` at the command line. The `git-rebase-todo` file will be opened in an Emacs buffer for you to edit. This file is opened using `emacsclient`, so just edit this file as you normally would, then call the `server-edit` function (typically bound to `C-x #`) to tell Emacs you are finished editing, and the rebase will proceed as usual.

If you have loaded `rebase-mode.el` (which is included in the Magit distribution), the `git-rebase-todo` buffer will be in `rebase-mode`. This mode disables normal text editing but instead provides single-key commands (shown in the buffer) to perform all the edits that you would normally do manually, including changing the operation to be performed each commit (“pick”, “squash”, etc.), deleting (commenting out) commits from the list, and reordering commits. You can finish editing the buffer and proceed with the rebase by pressing `C-c C-c`, which is bound to `server-edit` in this mode, and you can abort the rebase with `C-c C-k`, just like when editing a commit message in Magit.

19 Rewriting

As hinted at earlier, you can rewrite your commit history. For example, you can reset the current head to an earlier commit with `x`. This leaves the working tree unchanged, and the status buffer will show all the changes that have been made since that new value of the current head. You can commit these changes again, possibly splitting them into multiple commits as you go along.

Amending your last commit is a common special case of rewriting history like this.

Another common way to rewrite history is to reset the head to an earlier commit, and then to cherry pick the previous commits in a different order. You could pick them from the reflog, for example.

Magit has several commands that can simplify the book keeping associated with rewriting. These commands all start with the `r` prefix key.

(Unless you already do so, we recommend that you don't use the functionality described here. It is semi-deprecated and will be removed once its unique features have been ported to the `git rebase --interactive` workflow. Even now the latter is almost always the better option.)

Typing `r b` will start a rewrite operation. You will be prompted for a *base* commit. This commit and all subsequent commits up until the current head are then put in a list of *Pending commits*, after which the current head will be reset to the *parent* of the base commit. This can be configured to behave like `git rebase`, i.e. exclude the selected base commit from the rewrite operation, with the `magit-rewrite-inclusive` variable.

You would then typically use `a` and `A` to cherry pick commits from the list of pending commits in the desired order, until all have been applied. Magit shows which commits have been applied by changing their marker from `*` to `..`

Using `A` will immediately commit the commit (as usual). If you want to combine multiple previous commits into a single new one, use `a` to apply them all to your working tree, and then commit them together.

Magit has no explicit support for rewriting merge commits. It will happily include merge commits in the list of pending commits, but there is no way of replaying them automatically. You have to redo the merge explicitly.

You can also use `v` to revert a commit when you have changed your mind. This will change the `.` mark back to `*`.

Once you are done with the rewrite, type `r s` to remove the book keeping information from the status buffer.

If you rather wish to start over, type `r a`. This will abort the rewriting, resetting the current head back to the value it had before the rewrite was started with `r b`.

Typing `r f` will *finish* the rewrite: it will apply all unused commits one after the other, as if you would use `A` with all of them.

You can change the `*` and `.` marks of a pending commit explicitly with `r *` and `r ..`

In addition to a list of pending commits, the status buffer will show the *Pending changes*. This section shows the diff between the original head and the current head. You can use it to review the changes that you still need to rewrite, and you can apply hunks from it, like from any other diff.

20 Pushing and Pulling

Magit will run `git push` when you type `P P`. If you give a prefix argument to `P P`, you will be prompted for the repository to push to. When no default remote repository has been configured yet for the current branch, you will be prompted as well. Typing `P P` will only push the current branch to the remote. In other words, it will run `git push <remote> <branch>`. The branch will be created in the remote if it doesn't exist already. The local branch will be configured so that it pulls from the new remote branch. If you give a double prefix argument to `P P`, you will be prompted in addition for the target branch to push to. In other words, it will run `git push <remote> <branch>:<target>`.

Typing `f f` will run `git fetch`. It will prompt for the name of the remote to update if there is no default one. Typing `f o` will always prompt for the remote. Typing `F F` will run `git pull`. When you don't have a default branch configured to be pulled into the current one, you will be asked for it.

If there is a default remote repository for the current branch, Magit will show that repository in the status buffer header.

In this case, the status buffer will also have a *Unpushed commits* section that shows the commits on your current head that are not in the branch named `<remote>/<branch>`. This section works just like the history buffer: you can see details about a commit with `RET`, compare two of them with `.` and `=`, and you can reset your current head to one of them with `x`, for example. If you want to push the changes then type `P P`.

When the remote branch has changes that are not in the current branch, Magit shows them in a section called *Unpulled changes*. Typing `F F` will fetch and merge them into the current branch.

21 Submodules

- `o u` Update the submodules, with a prefix argument it will also initialize them.
- `o i` Initialize the submodules.
- `o b` Update and initialize the submodules in one go (same as `C-u o u`).
- `o s` Synchronizes submodules' remote URL configuration setting to the value specified in `.gitmodules`.

22 Bisecting

Magit supports bisecting by showing how many revisions and steps are left to be tested in the status buffer. You can control the bisect session from both the status and from log buffers with the *B* key menu.

Typing *B s* will start a bisect session. You will be prompted for a revision that is known to be bad (defaults to *HEAD*) and for a revision that is known to be good (defaults to the revision at point if there is one). *git* will select a revision for you to test, and Magit will update its status buffer accordingly.

You can tell *git* that the current revision is good with *B g*, that it is bad with *B b* or that *git* should skip it with *B k*. You can also tell *git* to go into full automatic mode by giving it the name of a script to run for each revision to test with *B u*.

The current status can be shown as a log with *B l*. It contains the revisions that have already been tested and your decisions about their state.

The revisions left to test can be visualized in *gitk* with *B v*.

When you're finished bisecting you have to reset the session with *B r*.

23 Finding commits not merged upstream

One of the comforts of git is that it can tell you which commits have been merged upstream but not locally and vice versa. Git's sub-command for this is `cherry` (not to be confused with `cherry-pick`). Magit has support for this by invoking `magit-cherry` which is bound to `y` by default.

Magit will then ask you first for the upstream revision (which defaults to the currently tracked remote branch if any) and the head revision (which defaults to the current branch) to use in the comparison. You will then see a new buffer in which all commits are listed with a directional marker, their revision and the commit message's first line. The directional marker is either `+` indicating a commit that's present in upstream but not in head or `-` which indicates a commit present in head but not in upstream.

From this list you can use the usual key bindings for cherry-picking individual commits (`a` for cherry-picking without committing and `A` for the same plus the automatic commit). The buffer is refreshed automatically after each cherry-pick.

24 Magit Extensions

24.1 Activating extensions

Magit comes with a couple of shipped extensions that allow interaction with `git-svn`, `topgit` and `stgit`. See following sections for specific details on how to use them.

Extensions can be activated globally or on a per-repository basis. Since those extensions are implemented as minor modes, one can use for example `M-x magit-topgit-mode` to toggle the `topgit` extension, making the corresponding section and commands (un)available.

In order to do that automatically (and for every repository), one can use for example:

```
(add-hook 'magit-mode-hook 'turn-on-magit-topgit)
```

Magit also allows configuring different extensions, based on the git repository configuration.

```
(add-hook 'magit-mode-hook 'magit-load-config-extensions)
```

This will read git configuration variables and activate the relevant extensions.

For example, after running the following commands, the `topgit` extension will be loaded for every repository, while the `svn` one will be loaded only for the current one.

```
$ git config --global --add magit.extension topgit
$ git config --add magit.extension svn
```

Note the `--add` flag, which means that each extension gets its own line in the `config` file.

24.2 Interfacing with Subversion

Typing `N r` runs `git svn rebase`, typing `N c` runs `git svn dcommit` and typing `N f` runs `git svn fetch`.

`N s` will prompt you for a (numeric, Subversion) revision and then search for a corresponding Git sha1 for the commit. This is limited to the path of the remote Subversion repository. With a prefix (`C-u N s` the user will also be prompted for a branch to search in.

24.3 Interfacing with Topgit

`Topgit` is a patch queue manager that aims at being close as possible to raw Git, which makes it easy to use with Magit. In particular, it does not require to use a different set of commands for “commit”, “update”, and other operations.

`magit-topgit.el` provides basic integration with Magit, mostly by providing a “Topics” section.

Topgit branches can be created the regular way, by using a “t/” prefix by convention. So, creating a “t/foo” branch will actually populate the “Topics” section with one more branch after committing `.topdeps` and `.topmsg`.

Also, the way we pull (see [Chapter 20 \[Pushing and Pulling\]](#), page 21) such a branch is slightly different, since it requires updating the various dependencies of that branch. This should be mostly transparent, except in case of conflicts.

24.4 Interfacing with StGit

StGit is a Python application providing similar functionality to Quilt (i.e. pushing/popping patches to/from a stack) on top of Git. These operations are performed using Git commands and the patches are stored as Git commit objects, allowing easy merging of the StGit patches into other repositories using standard Git functionality.

`magit-stgit.el` provides basic integration with Magit, mostly by providing a “Series” section, whose patches can be seen as regular commits through the “visit” action.

You can change the current patch in a series with the “apply” action, as well as you can delete them using the “discard” action.

Additionally, the `magit-stgit-refresh` and `magit-stgit-rebase` commands let you perform the respective StGit operations.

25 Using Git Directly

For situations when Magit doesn't do everything you need, you can run raw Git commands using `:.` This will prompt for a Git command, run it, and refresh the status buffer. The output can be viewed by typing `$`.

Appendix A GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ‘‘GNU
Free Documentation License’’.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.