Git

en.wikibooks.org

June 27, 2024

Contents

1 Obtaining Git

Git is available for *nix operating systems, as well as MacOS and Windows.

1.1 Binary

1.1.1 Linux

Debian-based distributions (`.deb`)

Git is available on Debian[1], and derivatives like Ubuntu[2]. It is currently packaged as `git`. More recent packages of git are available from the Ubuntu Git Maintainers' PPA[3]. You may also want to install some extended git functionality, for example `git-svn`, which allows two-way interoperability with Subversion[4], or `git-email` which provides utility functions for sending and receiving git data (primarily patches) via email.

```
$ sudo apt-get install git git-svn git-email
```

RPM-based distributions (`.rpm`)

Linux distributions using the RPM package format can use yum to get git:

```
$ sudo yum install git-core
```

1.1.2 macOS

A graphical installer can be found at Google code[5]. Alternative, if you have MacPorts[6] installed, you can do

```
$ sudo port install git-core
```

git is also included in the *com.apple.pkg.Core* package:

```
pkgutil --file-info `which git`
volume: /
path: /usr/bin/git
```

1 https://en.wikipedia.org/wiki/Debian
2 https://en.wikibooks.org/wiki/Ubuntu
3 https://launchpad.net/~git-core/+archive/ppa
4 https://en.wikibooks.org/wiki/Subversion
5 https://code.google.com/p/git-osx-installer
6 http://macports.org

```
pkgid: com.apple.pkg.Core
pkg-version: 10.13.4.1.1.1522206870
install-time: 1526062261
uid: 0
gid: 0
mode: 755
```

1.1.3 Windows

Git for Windows is available as a precompiled binary msysGit[7]. This includes the command line utilities, a GUI, and an SSH[8] client.

Additionally, those with Cygwin[9] can use its setup to obtain Git.

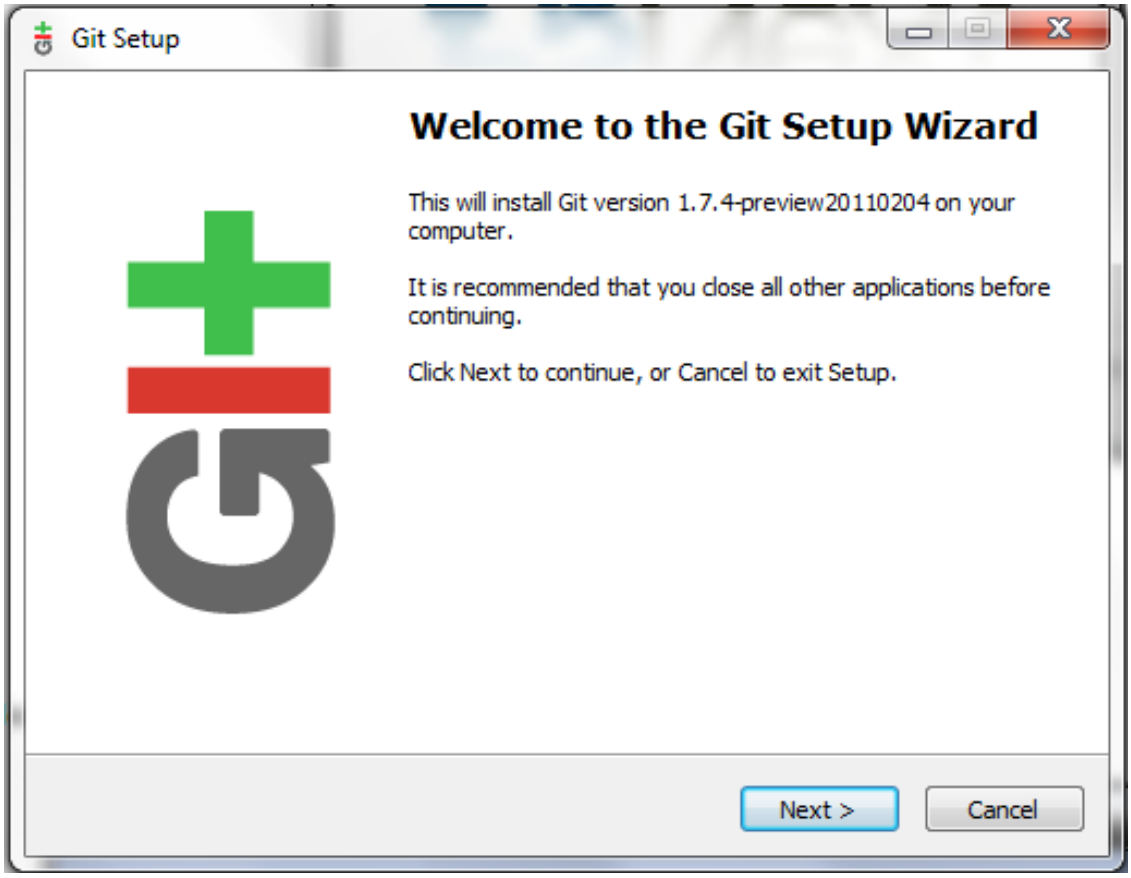

Figure 1

7 https://code.google.com/p/msysgit
8 https://en.wikibooks.org/wiki/SSH
9 https://en.wikibooks.org/wiki/Cygwin

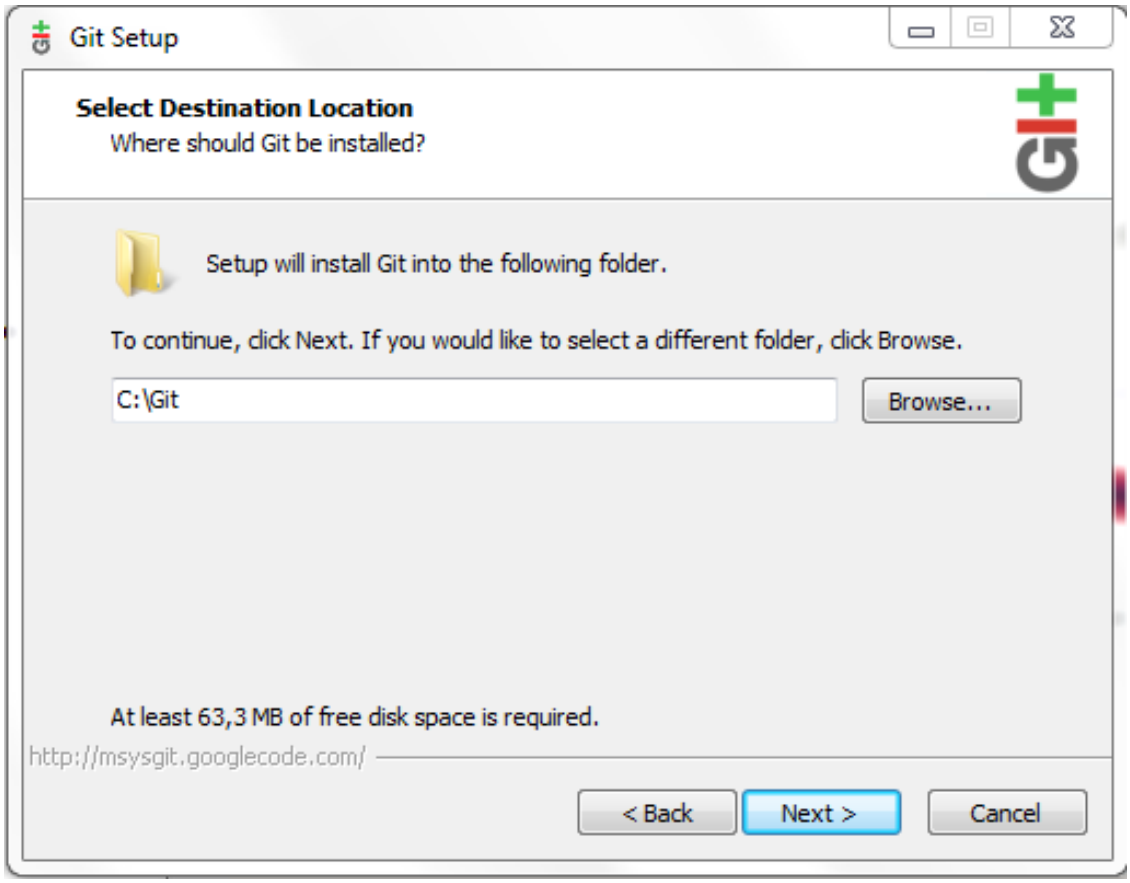
Git Setup
Select Destination Location
Where should Git be installed?
Setup will install Git into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Git
Browse...
At least 63,3 MB of free disk space is required.
http://msysgit.googlecode.com/
< Back
Next >
Cancel

Figure 2

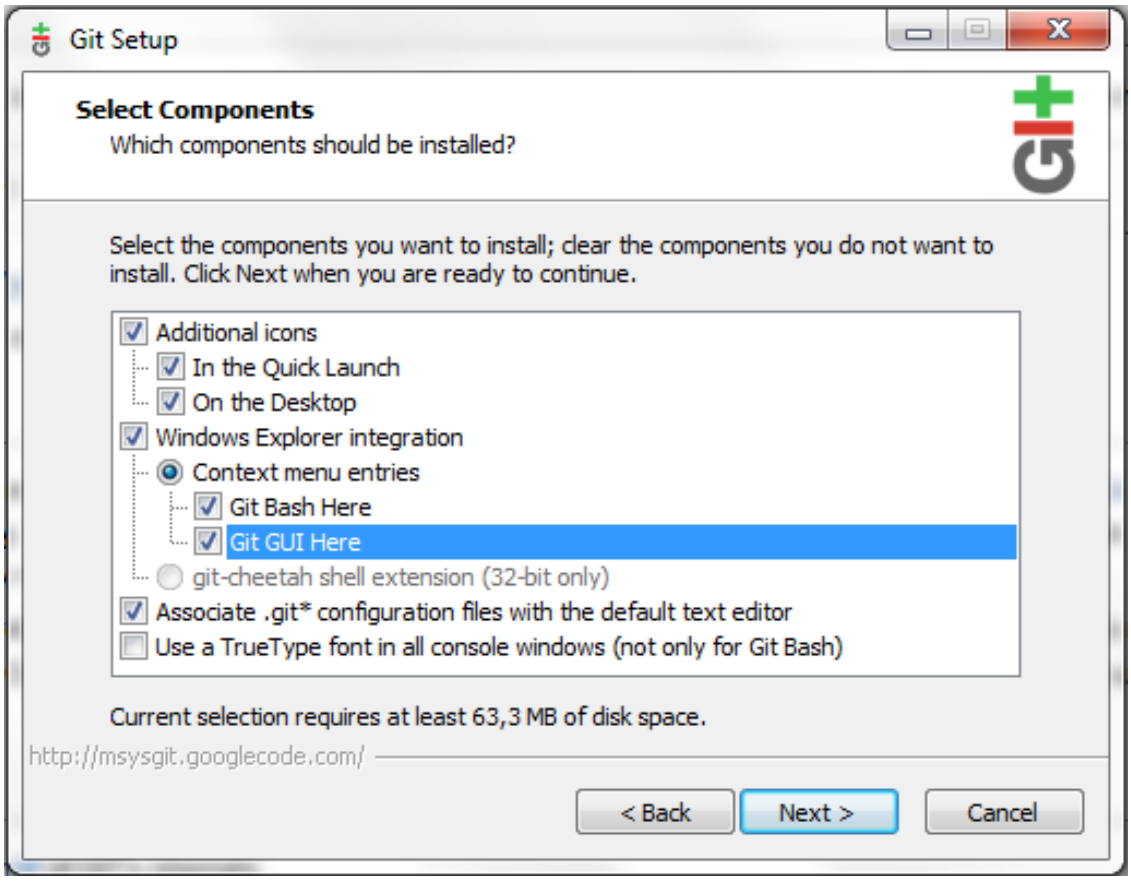
Git Setup
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Additional icons
In the Quick Launch
On the Desktop
Windows Explorer integration
Context menu entries
Git Bash Here
Git GUI Here
git-cheetah shell extension (32-bit only)
Associate .git* configuration files with the default text editor
Use a TrueType font in all console windows (not only for Git Bash)
Current selection requires at least 63,3 MB of disk space.
http://msysgit.googlecode.com/
< Back
Next >
Cancel

Figure 3

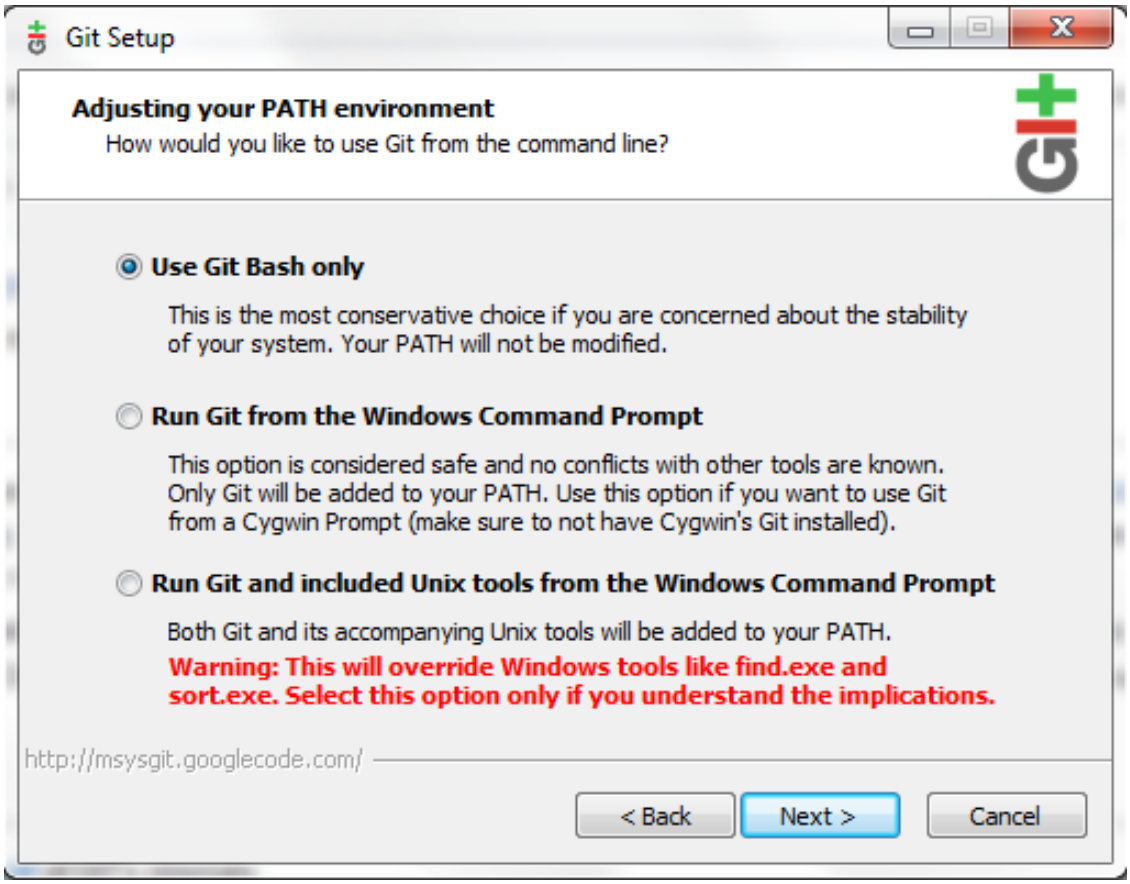
Git Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git Bash only
This is the most conservative choice if you are concerned about the stability of your system. Your PATH will not be modified.
Run Git from the Windows Command Prompt
This option is considered safe and no conflicts with other tools are known. Only Git will be added to your PATH. Use this option if you want to use Git from a Cygwin Prompt (make sure to not have Cygwin's Git installed).
Run Git and included Unix tools from the Windows Command Prompt
Both Git and its accompanying Unix tools will be added to your PATH.
Warning: This will override Windows tools like find.exe and sort.exe. Select this option only if you understand the implications.
http://msysgit.googlecode.com/
< Back
Next >
Cancel

Figure 4

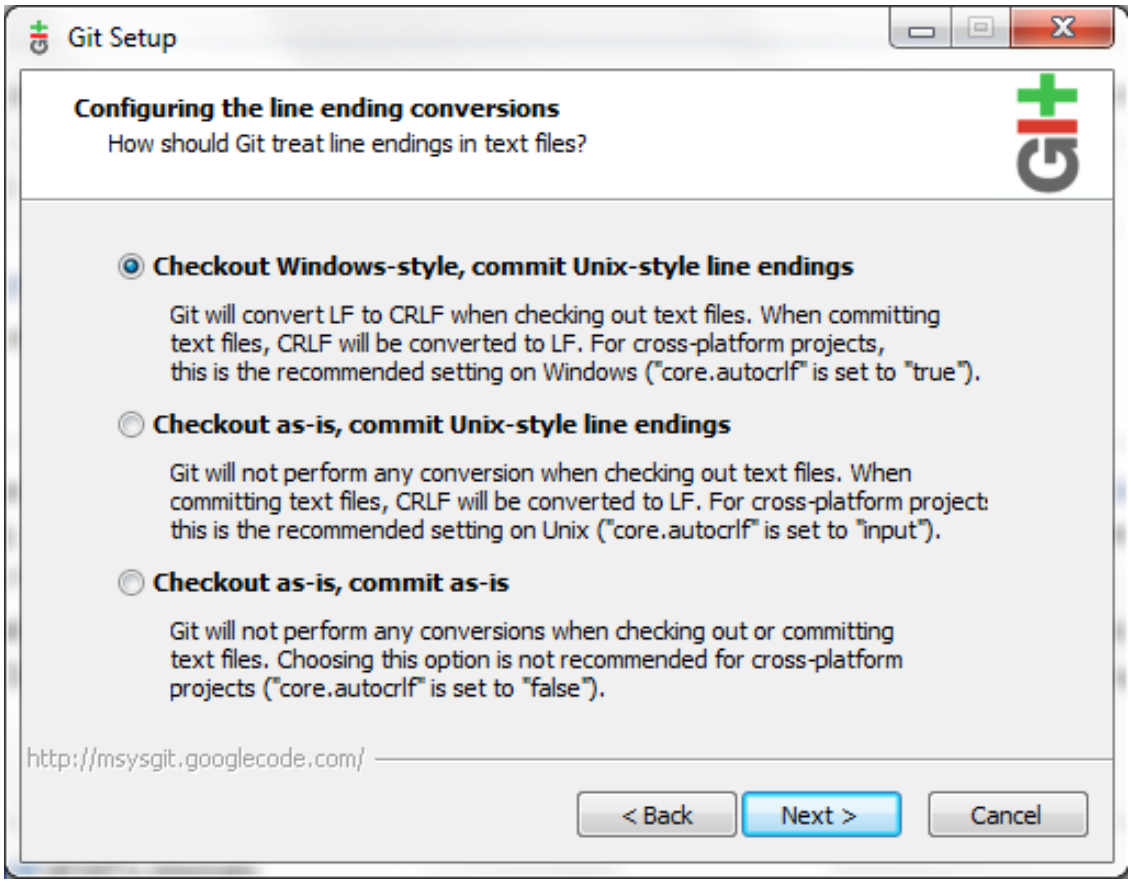
Git Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform project: this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
http://msysgit.googlecode.com/
< Back
Next >
Cancel

Figure 5

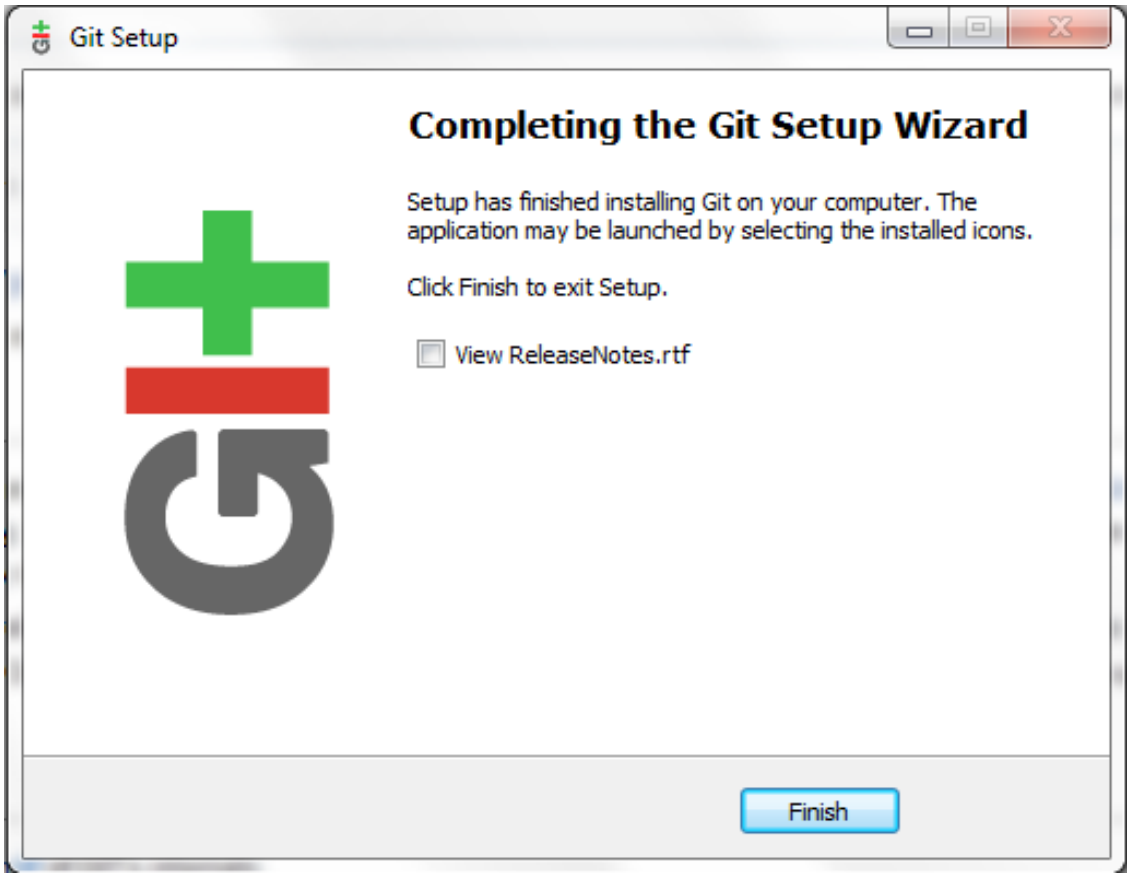

Figure 6

1.2 Source

1.2.1 Tarball

You can obtain a copy of the newest stable git from the git homepage at: git.or.cz[10]. In addition, daily snapshots[11] of git are provided by Dave Jones.

Below is an example of how to compile git from source, change "git-1.5.3.4.tar.gz" to the version you downloaded:

```
mkdir ~/src
cd ~/src
wget http://kernel.org/pub/software/scm/git/git-1.5.3.4.tar.gz
tar xzvf git-1.5.3.4.tar.gz
cd git-1.5.3.4
make configure
./configure --prefix=/usr/local
make
sudo make install
```

10 http://git.or.cz/
11 http://www.codemonkey.org.uk/projects/git-snapshots/git/

Without the added `--prefix` argument git will currently install to `~/bin`. This may or may not be what you want, in most distributions `~/bin` is not in the $PATH.[1][12] Without the –prefix, you might have to explicitly state the path to the component programs on invocation, i.e.: `~/bin/git-add foobar`. You can set `--prefix` to whatever better suits your particular setup.

1.2.2 Git

The source can additionally be acquired via git using:

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

Or in the event you have problems with the default Git port of 9418:

```
$ git clone http://www.kernel.org/pub/scm/git/git.git
```

1.3 First configuration

To avoid to reenter one's credential during each sync, register the account:

```
git config --global user.email "michael.boudran@en.wikibooks.org"
git config --global user.name "Michael Boudran"
```

To check the config:

```
 git config -l
```

To avoid the password in Linux, it's necessary to store it in plain text into:

```
vim ~/.netrc
```

With (ex: for *github.com*):

```
machine github.com
      login <user>
      password <password>
```

Please note that GitHub no longer use's Username and Password for authentication since September 2021. Instead, GitHub uses Authentication Tokens generated on [13]. A replacement for the above example would be:

```
machine github.com
      login <user>
      password <token>
```

13 `https://github.com/settings/tokens`

1.4 Other Git clients

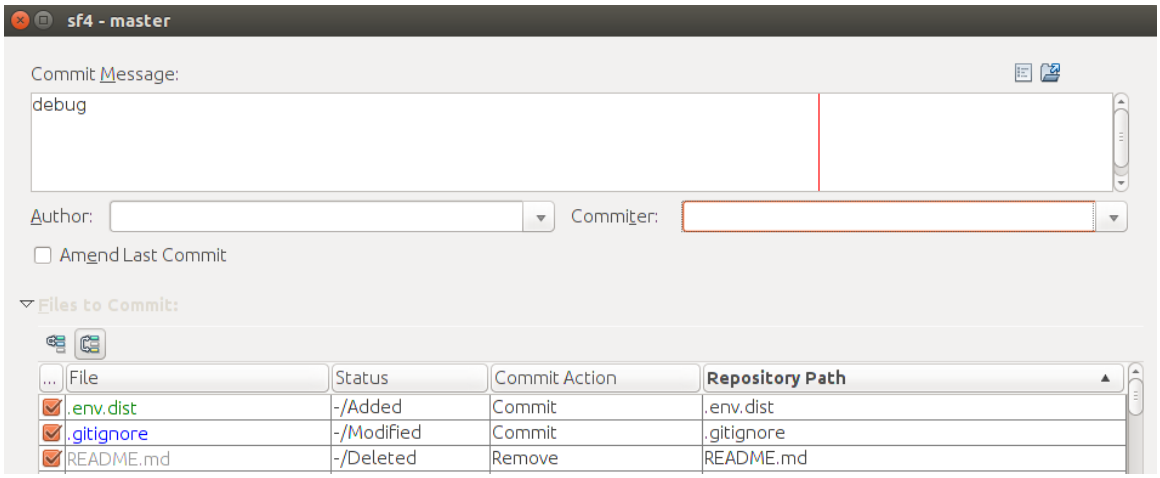

Figure 7 NetBeans commit.

Some integrated development environments like NetBeans[14] or PhpStorm[15] also provide or complete Git client.

TortoiseGit[16] allows to access to its Git commands by right-clicking on the concerned files and folders.

14 https://en.wikipedia.org/wiki/NetBeans
15 https://en.wikipedia.org/wiki/PhpStorm
16 https://en.wikipedia.org/wiki/TortoiseGit

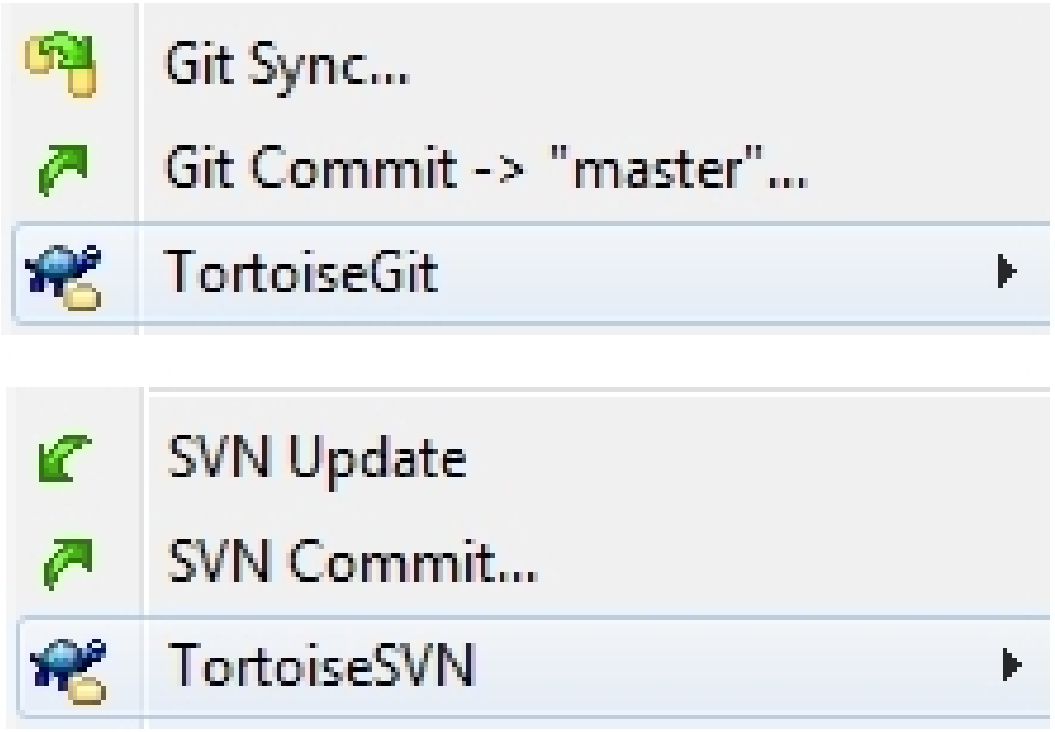

Figure 8 TortoiseGit quick options.

1.5 Footnotes

1. `^`[17] One effort to amend the lack of consistency through modern distributions and `~/bin`, has been addressed by the Ubuntu developers which seeks to patch PAM, the authentication mechanism, to set up the environmental variable `$PATH`. You can find more information out about this at `https://bugs.launchpad.net/ubuntu/+source/pam/+bug/64064`.

17 `#ref_launchpad`

2 Introduction

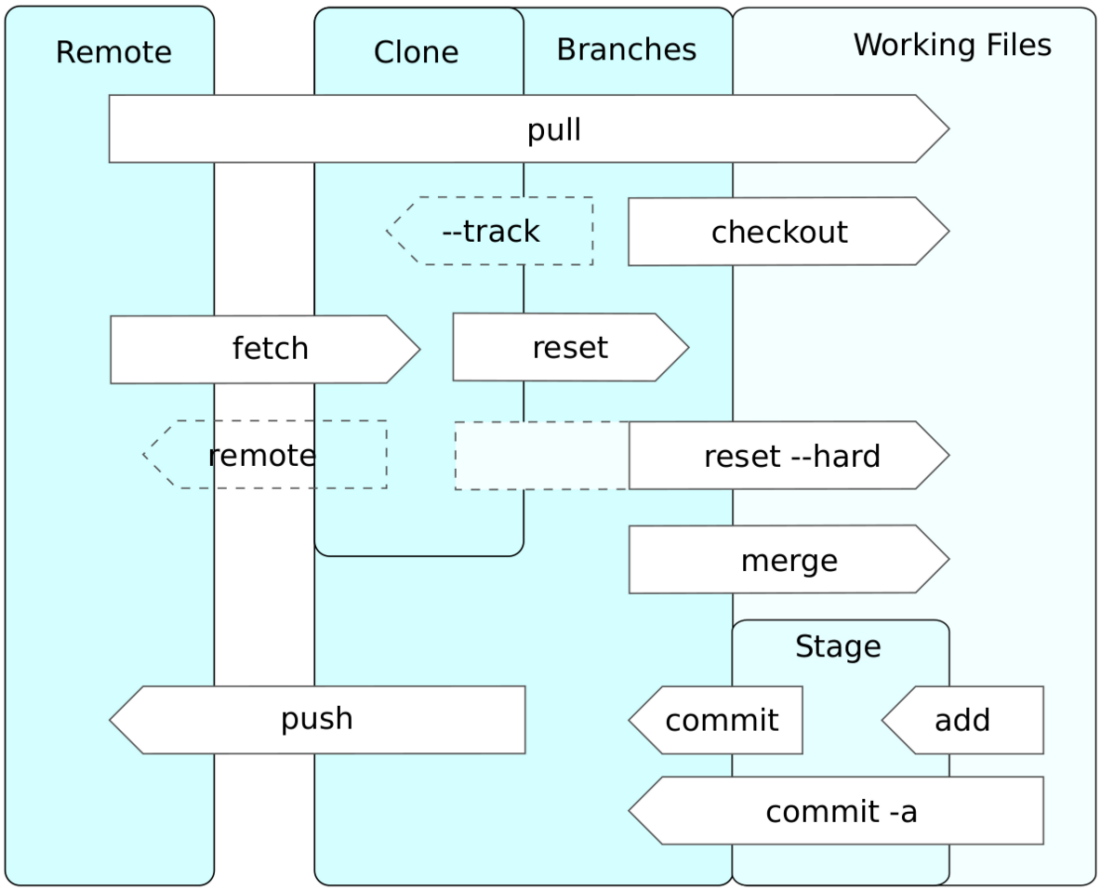

Figure 9 Git operations representation.

Here, we will introduce the simplest git commands: creating a new repository, adding and committing files, removing files, reverting bad commits, throwing away uncommitted changes, and viewing a file as of a certain commit.

2.1 Creating a git repository

Creating a new git repository is simple. There are two commands that cover this functionality: git-init(1)[1], and git-clone(1)[2]. Cloning a pre-existing repository will be covered later. For now, let's create a new repository in a new directory:

```
$ git init myrepo
```

Initialized empty Git repository in:

1. /home/username/myrepo/.git/ on Linux.
2. C:/Users/username/myrepo/.git/ on Windows.

If you already have a directory you want to turn into a git repository:

```
$ cd $my_preexisting_repo
$ git init
```

Taking the first example, let's look what happened:

```
$ cd myrepo
$ ls -A
.git
```

The totality of your repository will be contained within the `.git` directory. Conversely, some SCMs leave files all over your working directory (eg, `.svn`, `.cvs`, `.acodeic`, etc.). Git refrains, and puts all things in a subdirectory of the repository root aptly named `.git`.

Remark: to set the default directory where Git will point at each opening, under Windows right click on the shortcut, and change the path of the field called ”start in”.

2.2 Checking Your Status

To check the status of your repo, use the git-status(1)[3] command. For example, a newly-created repo with no commits in it as yet should show this:

```
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use ”git add” to track)
```

Get into the habit of frequent use of `git-status`, to be sure that you’re doing what you think you’re doing. :)

1 `http://git-scm.com/docs/git-init`
2 `http://git-scm.com/docs/git-clone`
3 `http://git-scm.com/docs/git-status`

2.3 Adding and committing files

Unlike most other VCSs, git doesn't assume you want to commit every modified file. Instead, the user adds the files they wish to commit to the *staging area* (also known as the *index* or *cache*, depending on which part of the documentation you read). Whatever is in the staging area is what gets committed. You can check what will be committed with git-status(1)[4] or `git diff --staged`.

To stage files for the next commit, use the command git-add(1)[5].

```
$ nano file.txt
hack hack hack...
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    file.txt
nothing added to commit but untracked files present (use "git add" to track)
```

This shows us that we're using the branch called "master" and that there is a file which git is not *tracking* (does not already have a commit history). Git helpfully notes that the file can be included in our next commit by doing `git add file.txt`:

```
$ git add file.txt
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm –cached <file>..." to unstage)
#
#    new file:   file.txt
#
```

After adding the file, it is shown as ready to be committed. Let's do that now:

```
$ git commit -m 'My first commit'
[master (root-commit) be8bf6d] My first commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 file.txt
```

In most cases, you will not want to use the `-m 'Commit message'` form - instead, leave it off to have `$EDITOR` opened so you can write a proper commit message. We will describe that next, but in examples, the `-m 'Commit message'` form will be used so the reader can easily see what is going on.

You can use `git add -A` to automatically stage all changed and untracked files for inclusion in the next commit. Once a file is being tracked, `git add -u` will stage it if it has changed.

4 `http://git-scm.com/docs/git-status`
5 `http://git-scm.com/docs/git-add`

2.3.1 reset

If you change your mind about staging a file, and you haven't committed yet, you can *un* stage it with the simplest form of the git-reset(1)[6] command:

```
$ git reset file.txt
```

to unstage just the one file, or

```
$ git reset
```

to remove everything in the staging area.

`git-reset` has many more functions than this, for example:

- To cancel the two latest commits without touching the files: `git reset 'HEAD~2'`.
- To cancel the two latest commits and their modifications into the files: `git reset HEAD~2 --hard`.
- To cancel the two last operations on the branch: `git reset 'HEAD@{2}'` (which uses `git reflog`). This can be used to cancel an undesired reset.

To exclude certain untracked files from being seen by `git add -A`, read on...

2.3.2 restore

`git restore` comes back to a version of the file specified in parameter[1].

2.3.3 Excluding files from Git

Often there are files in your workspace that you don't want to add to the repository. For example, `emacs` will write a backup copy of any file you edit with a tilde suffix, like `filename~`. Even though you can manually avoid adding them to the commit (which means never using `git add -A`), they clutter up the status list.

In order to tell Git to ignore certain files, you can create an ignore file, each line of which represents a specification (with wildcards) of the files to be ignored. Comments can be added to the file by starting the line with a blank or a `#` character.

For example:

```
# Ignore emacs backup files:
*~

# Ignore everything in the cache directory:
app/cache
```

Git looks for an ignore file under two names:

6 http://git-scm.com/docs/git-reset

- `.git/info/exclude` — this is specific to your own personal copy of the repository, not a public part of the repository.
- `.gitignore` — since this is outside the `.git` directory, it will normally be tracked by Git just like any other file in the repository.

What you put in either (or both) of these files depends on your needs. `.gitignore` is a good place to mention things that everybody working on copies of this repository is likely to want to be ignored, like build products. If you are doing your own personal experiments that are not likely to concern other code contributors, then you can put the relevant ignore lines into `.git/info/exclude`.

Note that ignore file entries are only relevant to the `git status` and `git add -A` (add all new and changed files) commands. Any files you explicitly add with `git add` *`filename`* will always be added to the repository, regardless of whether their names match ignore entries or not. And once they are added to the repository, changes to them will henceforth be automatically tracked by `git add -u`.

2.3.4 Good commit messages

Tim Pope[7] writes[8] about what makes a model Git commit message:

```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Write your commit message in the present tense: "Fix bug" and not "Fixed
bug."  This convention matches up with commit messages generated by
commands like git merge and git revert.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, preceded by a
  single space, with blank lines in between, but conventions vary here

- Use a hanging indent
```

Let's start with a few of the reasons why wrapping your commit messages to 72 columns is a good thing.

- Git log doesn't do any special wrapping of the commit messages. With the default pager of `less -S`, this means your paragraphs flow far off the edge of the screen, making them difficult to read. On an 80 column terminal, if we subtract 4 columns for the indent on the left and 4 more for symmetry on the right, we're left with 72 columns.

7 http://tpo.pe/

8 http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

- `git format-patch --stdout` converts a series of commits to a series of emails, using the messages for the message body. Good email netiquette[9] dictates we wrap our plain text emails such that there's room for a few levels of nested reply indicators without overflow in an 80 column terminal.

Vim users can meet this requirement by installing my vim-git runtime files, or by simply setting the following option in your git commit message file:

```
:set textwidth=72
```

For Textmate, you can adjust the "Wrap Column" option under the view menu, then use ^Q to rewrap paragraphs (be sure there's a blank line afterwards to avoid mixing in the comments). Here's a shell command to add 72 to the menu so you don't have to drag to select each time:

```
$ defaults write com.macromates.textmate OakWrapColumns '( 40, 72, 78 )'
```

More important than the mechanics of formatting the body is the practice of having a subject line. As the example indicates, you should shoot for about 50 characters (though this isn't a hard maximum) and always, always follow it with a blank line. This first line should be a concise summary of the changes introduced by the commit; if there are any technical details that cannot be expressed in these strict size constraints, put them in the body instead. The subject line is used all over Git, oftentimes in truncated form if too long of a message was used. The following are just a handful of examples of where it ends up:

- `git log --pretty=oneline` shows a terse history mapping containing the commit id and the summary
- `git rebase --interactive` provides the summary for each commit in the editor it invokes
- If the config option `merge.summary` is set, the summaries from all merged commits will make their way into the merge commit message
- `git shortlog` uses summary lines in the changelog-like output it produces
- git-format-patch(1)[10], git-send-email(1)[11], and related tools use it as the subject for emails
- git-reflog(1)[12], a local history accessible intended to help you recover from mistakes, get a copy of the summary
- `gitk`, a graphical interface which has a column for the summary
- Gitweb and other web interfaces like GitHub[13] use the summary in various places in their user interface.

The subject/body distinction may seem unimportant but it's one of many subtle factors that makes Git history so much more pleasant to work with than Subversion.

9 https://en.wikipedia.org/wiki/netiquette
10 http://git-scm.com/docs/git-format-patch
11 http://git-scm.com/docs/git-send-email
12 http://git-scm.com/docs/git-reflog
13 https://github.com

2.4 Removing files

Let's continue with some more commits, to show you how to remove files:

```
$ echo 'more stuff' >> file.txt
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   file.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Although git doesn't force the user to commit all modified files, this is a common scenario. As noted in the last line of `git status`, use `git commit -a` to commit all modified files without reading them first:

```
$ git commit -a -m 'My second commit'
[master e633787] My second commit
 1 files changed, 1 insertions(+), 0 deletions(-)
```

See the string of random characters in git's output after committing (bolded in the above example)? This is the abbreviation of the identifier git uses to track objects (in this case, a commit object). Each object is hashed using SHA-1[14], and is referred to by that string. In this case, the full string is `e6337879cbb42a2ddfc1a1602ee785b4bfbde518`, but you usually only need the first 8 characters or so to uniquely identify the object, so that's all git shows. We'll need to use these identifiers later to refer to specific commits.

To remove files, use the "rm" subcommand of git:

```
$ git rm file.txt
rm 'file.txt'
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    deleted:    file.txt
#
$ ls -a
.  ..  .git
```

Note that this deletes the file from your disk. If you only want to remove the file from the git repository but want to leave the file in your working directory, use `git rm --cached`.

```
$ git commit -m 'Removed file.txt'
[master b7deafa] Removed file.txt
 1 files changed, 0 insertions(+), 2 deletions(-)
 delete mode 100644 file.txt
```

2.4.1 Reverting a commit

To revert a commit with another, use `git revert`:

14 `https://en.wikipedia.org/wiki/SHA_hash_functions`

```
$ git revert HEAD
Finished one revert.
[master 47e3b6c] Revert "My second commit"
 1 files changed, 0 insertions(+), 1 deletions(-)
$ ls -a
. .. file.txt .git
```

You can specify any commit instead of HEAD. For example:

The commit before HEAD

```
git revert HEAD^
```

The commit five back

```
git revert HEAD~5
```

The commit identified by a given hash

```
git revert e6337879
```

2.4.2 Resetting a commit

`git reset` provides the same options as `git revert`, but instead of creating a revert commit, it just cancels the commit(s) and lets the file(s) uncommitted.

To cancel a reset, use: `git reset 'HEAD@{2}'`.

2.4.3 Throwing away local, uncommitted changes

To throw away your changes and get back to the most recently-committed state:

```
$ git reset –hard HEAD
```

As above, you can specify any other commit:

```
$ git reset –hard e6337879
```

If you only want to reset one file (where you have made some stupid mistake since the last commit), you can use

```
$ git checkout filename
```

This will delete all changes made to that file since the last commit, but leave the other files untouched.

2.4.4 Get a specific version of a file

To get a specific version of a file that was committed, you'll need the hash for that commit. You can find it with git-log(1)[15]:

```
$ git log
commit 47e3b6cb6427f8ce0818f5d3a4b2e762b72dbd89
```

15 `http://git-scm.com/docs/git-log`

```
Author: Mike.lifeguard <myemail@example.com>
Date:   Sat Mar 6 22:24:00 2010 -0400

    Revert "My second commit"

    This reverts commit e6337879cbb42a2ddfc1a1602ee785b4bfbde518.

commit e6337879cbb42a2ddfc1a1602ee785b4bfbde518
Author: Mike.lifeguard <myemail@example.com>
Date:   Sat Mar 6 22:17:20 2010 -0400

    My second commit

commit be8bf6da4db2ea32c10c74c7d6f366be114d18f0
Author: Mike.lifeguard <myemail@example.com>
Date:   Sat Mar 6 22:11:57 2010 -0400

    My first commit
```

Then, you can use `git show`:

```
$ git show e6337879cbb42a2ddfc1a1602ee785b4bfbde518:file.txt
hack hack hack...
more stuff
```

2.5 Git Checkout Is Not Subversion Checkout

If you are coming to Git after having used the Subversion[16] centralized version-control system, you may assume that the checkout operation in Git is similar to that in Subversion. It is not. While both Git and Subversion let you check out older versions of the source tree from the repository, only Subversion keeps track of which revision you have checked out. **Git does not**. git-status(1)[17] will only show you that the source tree does not correspond to the current branch HEAD; it will not check whether it corresponds to some prior commit in the history.

2.6 `diff` and `patch`: The Currency of Open-Source Collaboration

It is important to understand early on the use of the `diff(1)`[18] and `patch(1)`[19] utilities. `diff` is a tool for showing line-by-line differences between two text files. In particular, a *unified diff* shows added/deleted/changed lines next to each other, surrounded by *context* lines which are the same in both versions. Assume that the contents of `file1.txt` are this:

```
this is the first line.
this is the same line.
this is the last line.
```

while `file2.txt` contains this:

16 `http://subversion.apache.org/`
17 `http://git-scm.com/docs/git-status`
18 `https://en.wikipedia.org/wiki/diff`
19 `https://en.wikipedia.org/wiki/patch_(Unix)`

```
this is the first line.
this line has changed.
this is the last line.
```

Then a unified diff looks like this:

```
$ diff -u file1.txt file2.txt
— file1.txt   2014-04-18 11:56:35.307111991 +1200
+++ file2.txt   2014-04-18 11:56:51.611010079 +1200
@@ -1,3 +1,3 @@
 this is the first line.
-this is the same line.
+this line has changed.
 this is the last line.
$
```

Notice the extra column at the start of each line, containing a "-" for each line that is in the first file but not in the second, a "+" for each line that is in the second file but not the first, or a space for an unchanged line. There are extra lines, in a special format, identifying the files being compared and the numbers of the lines where the differences were found; all this can be understood by the `patch` utility, in order to change a copy of `file1.txt` to become exactly like `file2.txt`:

```
$ diff -u file1.txt file2.txt >patch.txt
$ patch <patch.txt
patching file file1.txt
$ diff -u file1.txt file2.txt
$
```

Notice how the second `diff` command no longer produces any output: the files are now identical!

This is how collaborative software development got started: instead of exchanging entire source files, people would distribute just their changes, as a unified `diff` or in `patch` format (same thing). Others could simply apply the patches to their copies. And provided there were no overlaps in the changes, you could even apply a patch to a file that had already been patched with another `diff` from someone else! And so this way changes from multiple sources could be merged into a common version with all the new features and bug fixes contributed by the community.

Even today, with version-control systems in regular use, such diffs/patches are still the basis for distributing changes. git-diff(1)[20] and git-format-patch(1)[21] both produce output which is compatible with `diff -u`, and can be correspondingly understood by `patch`. So even if the recipient of your patches isn't using Git, they can still accept your patches. Or you might receive a patch from someone who isn't using Git, and so didn't use `git-format-patch`, so you can't feed it to git-am(1)[22] to automatically apply it and save the commit; but that's OK, you can use git-apply(1)[23], or even `patch` itself on your source tree, and then make a commit on their behalf.

20 `http://git-scm.com/docs/git-diff`
21 `http://git-scm.com/docs/git-format-patch`
22 `http://git-scm.com/docs/git-am`
23 `http://git-scm.com/docs/git-apply`

2.7 Conclusion

You now know how to create a new repository, or turn your source tree into a git repository. You can add and commit files, and you can revert bad commits. You can remove files, view the state of a file in a certain commit, and you can throw away your uncommitted changes.

Next, we will look at the history of a git project on the command line and with some GUI tools, and learn about git's powerful branching model.

3 Branching & merging

Branching is supported in most VCSes. For example, Subversion[1] makes a virtue of "cheap copying"—namely, that creating a new branch does not mean making a copy of the whole source tree, so it is fast. Git's branching is just as fast. However, where Git really comes into its own is in *merging* between branches, in particular, reducing the pain of dealing with merge conflicts. This is what makes it so powerful in enabling collaborative software development.

3.1 Why Branch?

There are many reasons for creating multiple branches in a Git repo.

- You may have branches representing "stable" releases, which continue to get incremental bug fixes but no (major) new features. At the same time, you may have multiple "unstable" branches representing various new features being proposed for the next major release, and being worked on in parallel, perhaps by different groups. Those features which are accepted will need to be merged into the branch for the next stable release.
- You can create your own private branches for personal experiments. Later, if the code becomes sufficiently interesting to tell others about, you may make those branches public. Or you could send patches to the maintainer of the upstream public branch, and if they get accepted, you can pull them back down into your own copy of the public branch, and then you can retire or delete your private branch.

You may, in fact, want to add updates to different branches at different times. Switching between branches is easy.

3.2 Branching

3.2.1 View your branches

Use `git branch` with nothing else to see what branches your repository has:

```
$ git branch
* master
```

The branch called "master" is the default main line of development. You can rename it if you want, but it is customary to use the default. When you commit some changes, those changes are added to the branch you have checked out - in this case, master.

1 `http://subversion.apache.org/`

3.2.2 Create new branches

Let's create a new branch we can use for development - call it ”dev”:

```
$ git branch dev
$ git branch
  dev
* master
```

This only creates the new branch, it leaves your current HEAD where you remain. You can see from the * that the master branch is still what you have checked out. You can now use `git checkout dev` to switch to the new branch.

Alternatively, you can create a new branch and check it out all at once with

```
$ git checkout -b newbranch
```

3.2.3 Delete a branch

To delete the current branch, again use *git-branch*, but this time send the `-d` argument.

```
$ git branch -d <name>
```

If the branch hasn't been merged into master, then this will fail:

```
$ git branch -d foo
error: The branch 'foo' is not a strict subset of your current HEAD.
If you are sure you want to delete it, run 'git branch -D foo'.
```

Git's complaint saves you from possibly losing your work in the branch. If you are still sure you want to delete the branch, use `git branch -D <name>`instead.

Sometimes there are a lot of local branches which have been merged on the server, so have become useless. To avoid deleting them one by one, just use:

```
git branch -D `git branch --merged | grep -v \* | xargs`
```

3.2.4 Pushing a branch to a remote repository

When you create a local branch, it won't automatically be kept in sync with the server. Unlike branches obtained by pulling from the server, simply calling `git push` isn't enough to get your branch pushed to the server. Instead, you have to explicitly tell git to push the branch, and which server to push it to:

```
$ git push origin <branch_name>
```

3.2.5 Deleting a branch from the remote repository

To delete a branch that has been pushed to a remote server, use the following command:

```
$ git push origin :<branch_name>
```

This syntax isn't intuitive, but what's going on here is you're issuing a command of the form:

$ git push origin <local_branch>:<remote_branch>

and giving an empty branch in the <local_branch> position, meaning to overwrite the branch with nothing.

3.3 Merging

Branching is a core concept of a DVCS, but without good merging support, branches would be of little use.

git merge myBranch

This command merges the given branch into the current branch. If the current branch is a direct ancestor of the given branch, a *fast-forward* merge occurs, and the current branch head is redirected to point at the new branch. In other cases, a *merge commit* is recorded that has both the previous commit and the given branch tip as parents. If there are any conflicts during the merge, it will be necessary to resolve them by hand before the merge commit is recorded.

3.3.1 Handling a Merge Conflict

Sooner or later, if you're doing regular merges, you will hit a situation where the branches being merged will include conflicting changes to the same source lines. How you resolve this situation will be a matter of judgement (and some hand-editing), but Git provides tools you can use to try to get an insight into the nature of the conflict(s), and how best to resolve them.

Real-world examples of merge conflicts tend to be nontrivial. Here we will try to create a very simple, albeit artificial, example, to try to give you some flavour of what is involved.

Let us start with a repo containing a single Python source file, called `test.py`. Its initial contents are as follows:

```
#!/usr/bin/python3
#+
# This code doesn't really do anything at all.
#-

def func_common()
    pass
#end func_common

def child1()
    func_common()
#end child1

def child2()
    func_common()
#end child2

def some_other_func()
    pass
```

```
#end some_other_func
```

Commit this file to the repo, with a commit message saying something like "first version".

Now create a new branch and switch to it, using the command

git checkout -b side-branch

(This second branch is to simulate work being done on the same project by another programmer.) Edit the file `test.py`, and simply swap the definitions of the functions `child1` and `child2` around, equivalent to applying the following patch:

```
diff --git a/test.py b/test.py
index 863611b..c9375b3 100644
--- a/test.py
+++ b/test.py
@@ -7,14 +7,14 @@ def func_common()
     pass
 #end func_common

-def child1()
-    func_common()
-#end child1
-
 def child2()
     func_common()
 #end child2

+def child1()
+    func_common()
+#end child1
+
 def some_other_func()
     pass
 #end some_other_func
```

Commit the update to the branch `side-branch` with a message like "swap a pair of functions around".

Now switch back to the `master` branch:

git checkout master

This will also put you back to the previous version of `test.py`, since that was the last (in fact only) version committed to that branch.

On this branch, we now rename the function `func_common` to `common`, equivalent to the following patch:

```
diff --git a/test.py b/test.py
index 863611b..088c125 100644
--- a/test.py
+++ b/test.py
@@ -3,16 +3,16 @@
 # This code doesn't really do anything at all.
 #-

-def func_common()
+def common()
     pass
-#end func_common
+#end common
```

```
 def child1()
-    func_common()
+    common()
 #end child1

 def child2()
-    func_common()
+    common()
 #end child2

 def some_other_func()
```

Commit this change to the `master` branch, with a message like "rename func_common to common".

Now, try to merge in the change you made on `side-branch`:

git merge side-branch

This should immediately fail, with a message like

```
Auto-merging test.py
CONFLICT (content): Merge conflict in test.py
Automatic merge failed; fix conflicts and then commit the result.
```

Just to check what git-status(1)[2] reports:

```
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:      test.py

no changes added to commit (use "git add" and/or "git commit -a")
```

If we look at `test.py` now, it should look like

```
#!/usr/bin/python3
#+
# This code doesn't really do anything at all.
#-

def common()
    pass
#end common

<<<<<<< HEAD
def child1()
    common()
#end child1

=======
>>>>>>> side-branch
def child2()
    common()
#end child2
```

2 http://git-scm.com/docs/git-status

```
def child1()
    func_common()
#end child1

def some_other_func()
    pass
#end some_other_func
```

Note those sections marked "<<<<<<< HEAD" ... "=======" ... ">>>>>>> *src-branch*": the part between the first two markers comes from the HEAD branch, the one we are merging *onto* (`master`, in this case), while the part between the last two markers comes from the branch named *src-branch*, which we are merging *from* (`side-branch`, in this case).

Assuming we know exactly what the code does, we can carefully fix up all the conflicting/duplicated parts, remove the markers, and continue the merge. But perhaps this is a large project, and no single person, not even the project leader, fully understands every corner of the code. In this case, it is helpful to at least narrow down the set of commits that lead directly to the conflict, in order to get a handle on what is going on. There is a command that you can use, `git log --merge`, which is designed specifically to be used during a merge conflict, for just this purpose. In this example, I get output something like this:

```
$ git log –merge
commit 9df4b11586b45a30bd1e090706e3ff09692fcfa7
Author: Lawrence D'Oliveiro <ldo@geek-central.gen.nz>
Date:   Thu Apr 17 10:44:15 2014 +0000

    rename func_common to common

commit 4e98aa4dbd74543d7035ea781313c1cfa5517804
Author: Lawrence D'Oliveiro <ldo@geek-central.gen.nz>
Date:   Thu Apr 17 10:43:48 2014 +0000

    swap a pair of functions around
$
```

Now, as project leader, I can look further at just those two commits, and figure out that nature of the conflict is really quite simple: one branch has swapped the order of two functions, while the other has changed the name of another function being referenced within the rearranged code.

Another useful command is `git diff --merge`, which shows a 3-way diff between the state of the source file in the staging area, and the versions from the parent branches:

```
$ git diff –merge
diff –cc test.py
index c9375b3,863611b..088c125
— a/test.py
+++ b/test.py
@@@ -3,18 -3,18 +3,18 @@@
  # This code doesn't really do anything at all.
  #-

–def func_common()
++def common()
      pass
–#end func_common
-
- def child2()
-     func_common()
```

```
- #end child2
++#end common

  def child1()
–     func_common()
++     common()
  #end child1

+ def child2()
 -     func_common()
++     common()
+ #end child2
+
  def some_other_func()
      pass
  #end some_other_func
$
```

Here you see, in the first two columns of each line, "+" and "-" characters indicating lines added/removed with respect to the two branches, or a space indicating no change.

Armed with this information, I can approach the problem of fixing up the conflicted file with a bit more confidence, creating the following merged version of `test.py`:

```
#!/usr/bin/python3
#+
# This code doesn't really do anything at all.
#-

def common()
    pass
#end common

def child2()
    common()
#end child2

def child1()
    common()
#end child1

def some_other_func()
    pass
#end some_other_func
```

Just to recheck, after doing `git add test.py` on the above fixed version, but before committing, do another `git diff --merge`, which should produce output like:

```
diff --cc test.py
index c9375b3,863611b..088c125
--- a/test.py
+++ b/test.py
@@@ -3,18 -3,18 +3,18 @@@
  # This code doesn't really do anything at all.
  #-

--def func_common()
++def common()
      pass
--#end func_common
-
- def child2()
-     func_common()
- #end child2
++#end common
```

```
  def child1()
--    func_common()
++    common()
  #end child1

+ def child2()
 -    func_common()
++    common()
+ #end child2
+
  def some_other_func()
      pass
  #end some_other_func
```

And what does `git status` say?

```
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

        modified:   test.py
```

Now when you do like it says and enter `git commit`, Git automatically finishes the merge.

3.3.2 "The Stupid Content Tracker"

The git(1)[3] man page summarizes Git as "the stupid content tracker". It is important to understand what "stupid" means in this case: it means that Git does not use elaborate algorithms to try to automatically handle merge conflicts, instead it concentrates on displaying just the relevant information to help human intelligence to resolve the conflict. Linus Torvalds has famously said that he wouldn't trust his code to such elaborate merge conflict-resolution systems, which is why he deliberately designed Git to be "stupid", and therefore, reliable.

3 http://git-scm.com/docs/git

4 Beyond the basics

This tutorial covers some of the more advanced, multi-user features of git. For the single-user features, please go to the Single developer basics[1].

4.1 Checking out remote repositories

One way to check out a remote git repository is

```
$ git clone ssh://username@server.com:port/remote/path/to/repo
```

Now you have a local copy of that repository. You can use all the commands that were introduced in the Single developer basics[2]. Once you are done, you might want to check in your changes to the central repository again.

First you want to do a `git pull` in case the repository has changed in the meantime and you might have to merge your branch with the repository. After merging, you can use `git push` to send your changes to the repository:

```
$ git pull /remote/path/to/repo
```

or

```
$ cd repo
$ git pull
```

then

```
$ git push
```

4.1.1 Checking out local repositories

git clone also works for local repositories:

```
$ git clone /local/path/to/repo
```

4.1.2 Checking out remote branches

You might also want to check out remote branches, work on them and check in your local branches. First you might want to know which branches are available:

1 https://en.wikibooks.org/wiki/Git/Overview
2 https://en.wikibooks.org/wiki/Git/Overview

```
$ git branch -r
$ git remote show origin
```

Get a remote branch (pull into a local branch):

```
$ git pull origin remoteBranchName:localBranchName
```

Update a remote branch (push a local branch into a remote branch):

```
$ git push origin localBranchName:remoteBranchName
```

This assumes that you have a remote repository called ”origin”. You can check this with `git remote`.

If you want to create a local branch from a remote branch, use:

```
$ git checkout -b mylocalbranch origin/maint
```

Deleting remote branches works like this

```
$ git push origin :remoteBranchNameToDelete
```

The following command synchronizes branches $ git fetch origin

4.1.3 Tags

Git allows you to specify some tags in order to focus on some things in the history[2].

In order to add an annotated tag:

```
$ git tag -a mytag
```

or also:

```
$ git tag -a mytag my-branch
```

To add a lightweight tag:

```
$ git tag mytag
```

To force overwriting existing tag:

```
$ git tag -f mytag HEAD
```

To display previous tags:

```
$ git tag
```

Tags can be pushed to remote with

```
$ git push --tags
```

To position the repo on a tag:

```
git checkout tags/0.3.4
```

4.1.4 Tags vs Branches

Both tags and branches point to a commit, they are thus aliases for a specific hash and will save you time by not requiring to type in a hash.

The difference between tags and branches are that a branch always points to the top of a development line and will change when a new commit is pushed whereas a tag will not change. Thus tags are more useful to ”tag” a specific version and the tag will then always stay on that version and usually not be changed.

In practice, tags are used to designate the software versioning[3], and are named with numbers (ex: v1.0.2).

3 `https://en.wikipedia.org/wiki/software_versioning`

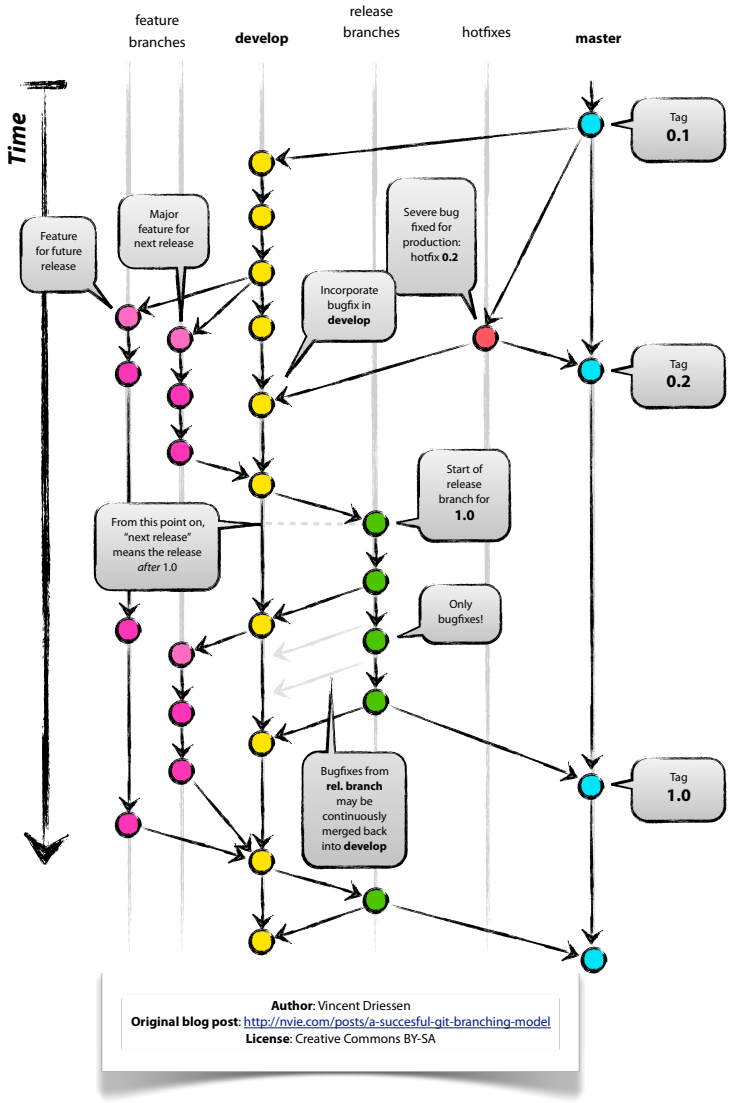

Figure 10 Example of branches with tags.

4.2 Create and Apply a Patch

In order to create a plain text patch (series) for the changes between origin and master, use

```
$ git format-patch origin/master
```

In order to apply a submitted plain text patch (series), use

```
$ git apply --stat  P1.txt  #see the stats, how much will the path change?
$ git apply --check P1.txt  #check for problems
```

```
$ git am < P1.txt          #apply the patches in the correct order
```

4.3 References

1. [4]
2. [5]

Understanding something of the internal structure of Git is crucial to understanding how Git works.

4.4 Naked Git Structure

The following is a freshly initialized git v1.9.0 repository.[2][6]

```
.
└── .git/
    ├── HEAD
    ├── branches/
    ├── config
    ├── description
    ├── hooks/
    │   ├── applypatch-msg.sample
    │   ├── commit-msg.sample
    │   ├── post-update.sample
    │   ├── pre-applypatch.sample
    │   ├── pre-commit.sample
    │   ├── prepare-commit-msg.sample
    │   ├── pre-push.sample
    │   ├── pre-rebase.sample
    │   └── update.sample
    ├── info/
    │   └── exclude
    ├── objects/
    │   ├── info/
    │   └── pack/
    └── refs/
        ├── heads/
        └── tags/
```

Additional files and folders may appear as activity happens on the repository.

4.4.1 Specific Files

COMMIT_EDITMSG

The message for a commit being made is saved here by a text editor.

4 https://git-scm.com/docs/git-restore
5 http://git-scm.com/book/en/Git-Basics-Tagging

FETCH_HEAD

Information is saved here from the last git-fetch(1)[7] operation, for use by a later git-merge(1)[8].

HEAD

HEAD indicates the currently checked out code. This will usually point to the branch you're currently working on.

You can also enter what git calls a "detached HEAD" state, where you are not on a local branch. In this state the HEAD points directly to a commit rather than a branch.

config

The configuration file for this git repository. It can contain settings for how to manage and store data in the local repository, remote repositories it knows about, information about the local user and other configuration data for git itself.

You can edit this file with a text editor, or you can manage it with the git-config(1)[9] command.

description

Used by repository browser tools - contains a description of what this project is. Not normally changed in non-shared repositories.

index

This is the staging area. It contains, in a compact form, all changes to files that have been staged for the next commit.

info/exclude

This is your own personal exclude file for your copy of the repo.

info/refs

If this file exists, it contains definitions, one to a line, of branches (both local and remote) and tags defined for the repository, in addition to ones that may be defined in individual files in `refs/heads` and `refs/tags`. This file seems to be used for large repositories with lots of branches or tags.

7 `http://git-scm.com/docs/git-fetch`
8 `http://git-scm.com/docs/git-merge`
9 `http://git-scm.com/docs/git-config`

ORIG_HEAD

Operations that change commit history on the current branch save the previous value of HEAD here, to allow recovery from mistakes.

4.4.2 Folders Containing Other Files

branches

Never seems to be used.

hooks

Contains scripts to be run when particular events happen within the git repository. Git gives you a set of initial example scripts, with `.sample` on the ends of their names (see the tree listing above); if you take off the `.sample` suffix, Git will run the script at the appropriate time.

Hooks would be used, for example, to run tests before creating each commit, filter uploaded content, and implement other such custom requirements.

logs

The reflogs are kept here.

objects

This is where all the files, directory listings, commits and such are stored.

There are both unpacked objects in numbered directories under this, and "packs" containing many compressed objects within a pack directory. The uncompressed objects will be periodically collected together into packs by automatic "git gc" runs.

refs/heads

Can contain one file defining the head commit for each local branch (but see `info/refs` above).

refs/remotes

Can contain one subdirectory for each remote repository you have defined. Within each subdirectory, there is a file defining the tip commit for each branch on that remote.

refs/tags

Can contain one file defining the commit corresponding to each tag (but see `info/refs` above).

svn

This directory will appear if you use git-svn(1)[10] to communicate with a Subversion server.

4.5 Basic Concepts

4.5.1 Object Types

A Git repository is made up of these object types:

- A *blob* holds the entire contents of a single file. It doesn't hold any information about the name of the file or any other metadata, just the contents.
- A *tree* represents the state of a directory tree. It contains the pathnames of all the component files and their modes, along with the IDs of the blobs holding their contents. Note that there is no representation for a directory on its own, so a Git repository cannot record the fact that subdirectories were created or deleted, only the files in them.
- A *commit* points to a tree representing the state of the source tree as of immediately after that commit. It also records the date/time of the commit, the author/committer information, and pointers to any parent(s) of that commit, representing the immediately-prior state of the source tree.
- A *tag* is a name pointing to a commit. These are useful, for example, to mark release milestones. Tags can optionally be digitally signed, to guarantee the authenticity of the commit.
- A *branch* is a name pointing to a commit. The difference between a branch and a tag is that, when a branch is the currently-checked-out branch, then adding a new commit will automatically update the branch pointer to point to the new commit.

Blobs, trees and commits all have IDs which are computed from SHA-1 hashes of their contents. These IDs allow different Git processes on different machines to tell whether they have identical copies of things, without having to transfer their entire contents over. Because SHA-1 is a cryptographically strong hash algorithm, it is practically impossible to make a change to the contents of any of these objects without changing its ID. Git doesn't prevent you from rewriting history, but you cannot hide the fact that you have done so.

A commit may have 0, 1 or more parents. Typically there is only one commit with no parents—a *root commit*—and that is the first commit to the repository. A commit which makes some change to one branch will have a single parent, the previous commit on that branch. A commit which is a merge from two or more branches will have two or more parent commits.

Note that a branch points to a *single* commit; the chain of commits is implicit in the parent(s) of that commit, and their parents, and so on.

10 `http://git-scm.com/docs/git-svn`

4.5.2 Topology Of Commits

The commit history in Git is arranged as a *directed acyclic graph* (DAG). To understand what this means, let's take the terms step by step.

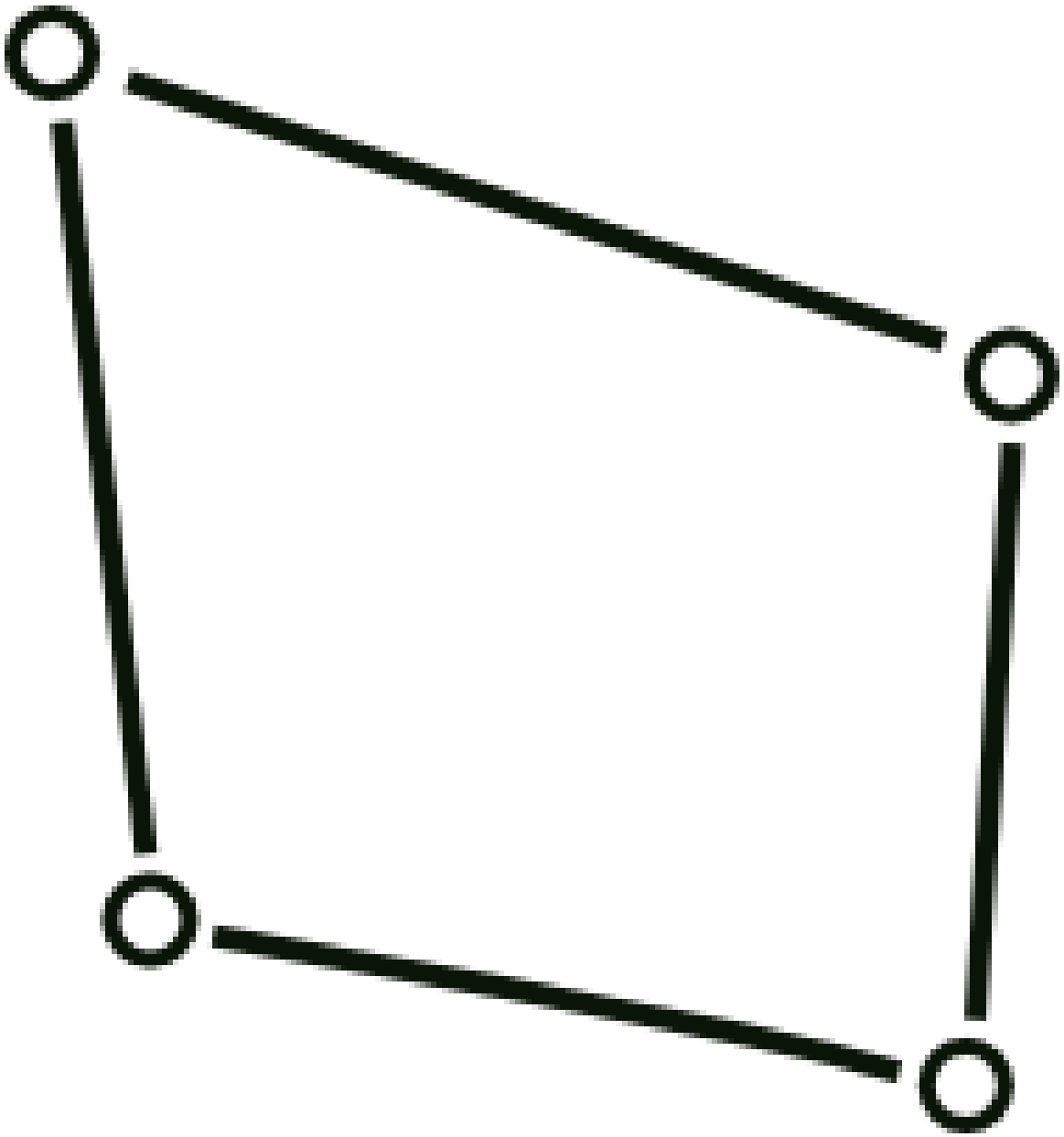

Figure 11 just a graph

- In mathematical terms, a *graph* is a bunch of points (*nodes*) connected by lines (*edges*).

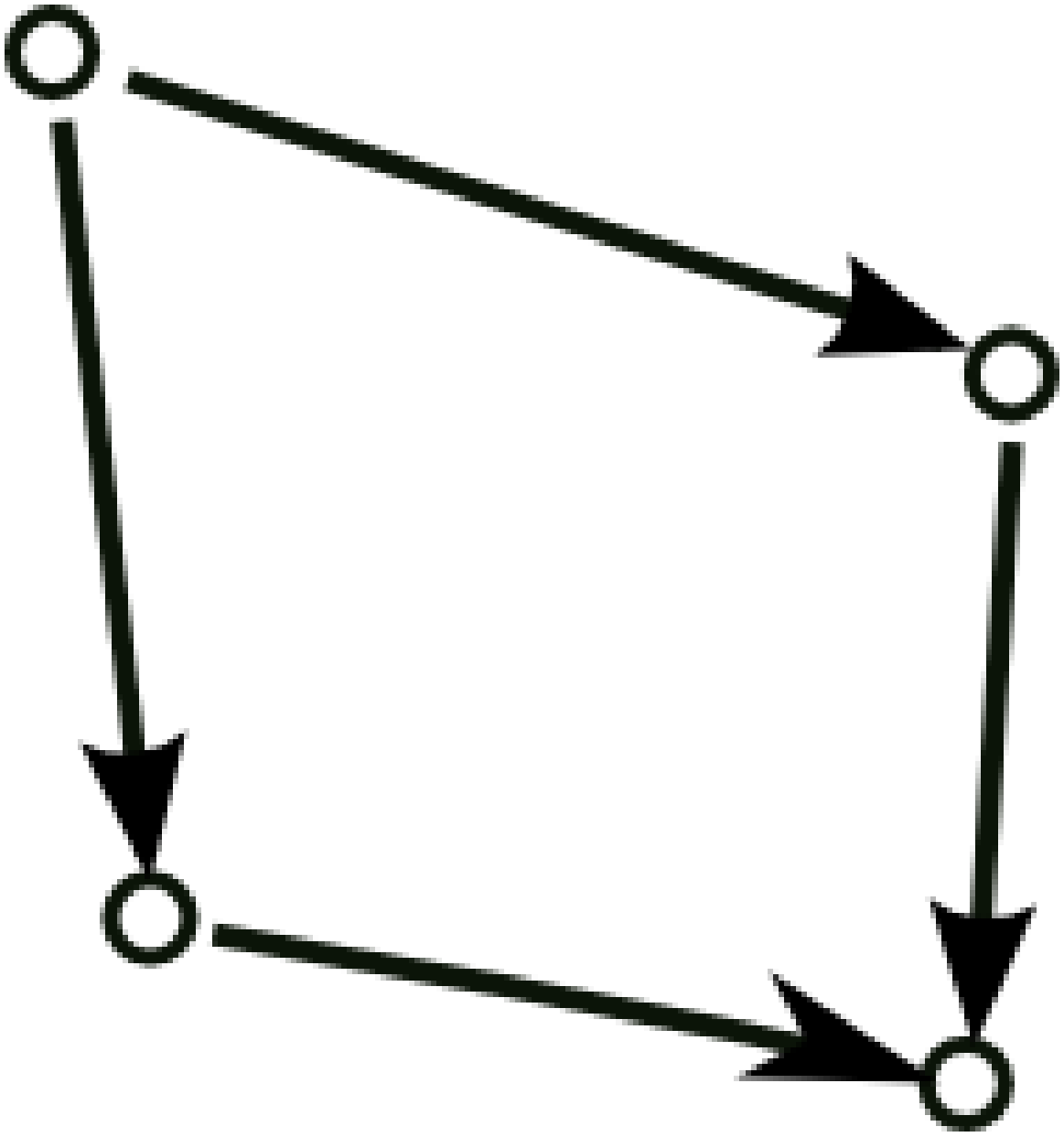

Figure 12 directed edges

- A *directed* graph is one where each edge has a direction, represented here by an arrowhead. Note that the arrow points from the child to the parent, not the other way round; it is the child that records who its parent(s) are, the parent does not record who its children are, because the set of children can change at any time, but the parent cannot change without invalidating its SHA-1 hash.

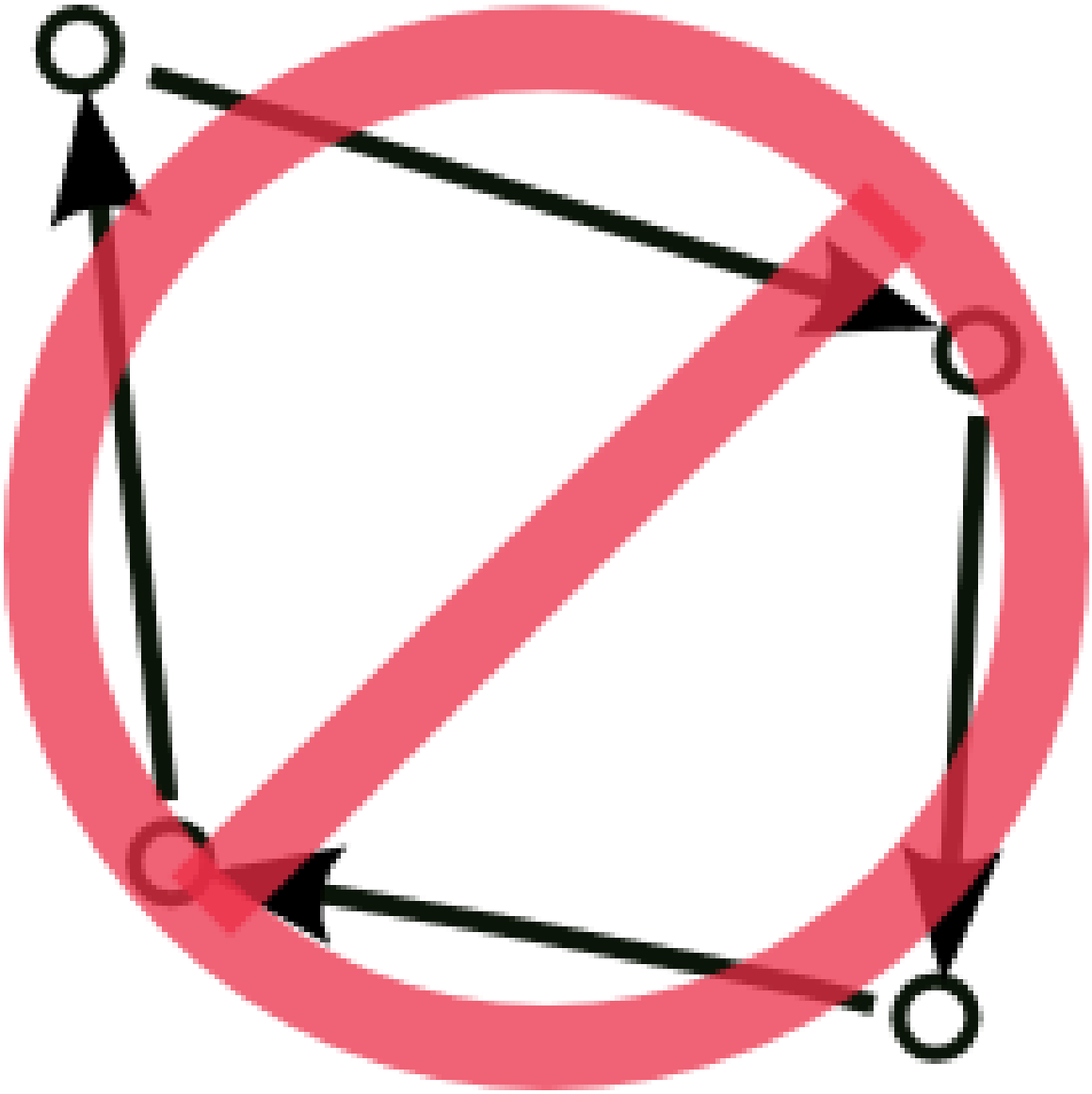

Figure 13 cycles not allowed

- *Acyclic* means that, if you start from any point and traverse edges in the direction of the arrows, you can never get back to your starting point, no matter what choice you make at any branch. No child can ever be a (direct or indirect) parent of itself!

In Git terms, each node represents a commit, and the lines and arrows represent parent-child relationships. Banning cycles simply means that a commit cannot be a (direct or indirect) parent or a child of itself!

4.5.3 The Reflog

The *reflog*s record changes that are not saved as part of the commit history—things like rebases, fast-forward merges, resets and the like. There is one reflog per branch. The reflog is not a public part of the repository, it is strictly specific to your local copy, and information

is only kept in it for a limited time (2 weeks by default). It provides a safety net, allowing you to recover from mistakes like deleting or overwriting things you didn't mean to.

4.5.4 Reachability And Garbage Collection

A commit is *reachable* if it is pointed to by a branch, tag or reflog entry, or is a parent of a commit which is reachable. A tree is correspondingly reachable if it is pointed to by a reachable commit, and a blob is reachable if it is pointed to by a reachable tree. Other commit/tree/blob objects are *unreachable*, and are not really serving any purpose beyond taking up space.

It is quite normal for your repositories to accumulate unreachable objects over time, perhaps as a result of aborted commits, deletion of unwanted branches, that kind of thing. Such objects will be deleted from the repository by a `git gc` command. This is also done automatically every now and then by some other commands, so it is rarely necessary to invoke `git gc` explicitly.

4.6 Git files out of .git folder

4.6.1 .gitkeep

Placed in a directory, it guarantees that it will be committed, even if empty.

4.6.2 .gitignore

Contains the files and folders to exclude from versioning. Ex:

```
/var/
/vendor/
/.env.*.local
```

4.6.3 .gitattributes

Contains somes attributes[1]. For example, to ignore .gitattributes and .gitignore in the "git archive" exports:

```
.gitattributes export-ignore
.gitignore export-ignore
```

4.7 Footnotes

1. ^[11] Generated with tree v1.5.1.1 using `tree -AnaF`.

11 `#ref_tree`

4.8 Principle

If you want to change the commit messages, the order or the number of commits, use:

```
$ git rebase -i HEAD~3
```

where HEAD can also be any other branch you are on and you can work on any number of commits, not just 3. You can delete commits, merge them together with ”squash” or change their order. If something goes wrong, use

```
$ git rebase -i --abort
```

Note that this will change all the commit-ids in the moved commits. These are because the commit-id also takes the commit's history in to account and the same change in a different place is considered a different commit by git. Rebasing shared changes can make combining those changes down the track difficult - you will not normally want to rebase any changes which has been incorporated in to somebody else's or a shared repository. A **superproject** is a new aspect of git which has been in development for a long while. It addresses the need for better control over numerous git repositories. The porcelain for the superproject functionality is fairly new and was only recently released with Git v1.5.3.

A Git **superproject** may consist of a set of *git repositories* and each of these ”git repositories” is called a **submodule**.You can think of a **submodule** as a **subproject,** and the **superproject** as a **supermodule.** It really doesn't make too much sense why either of these terms would have a *sub-* or *super-* prefix without their alternative; nonetheless, that's how the official Git documentation will refer to them.

The only git application specific to the submodule/superproject functionality is **git-submodule**.

4.9 Superprojects

A **Superproject**, is simply a git repository. To create a superproject, simply *git init* any directory, and *git submodule add* all of the git archives you wish to include. A quick aside, you can not currently *git submodule add* git repositories that are direct children within the same directory.[3][12]

The resulting structure will look similar to this:

```
|- superproject
  |- submodule (git archive) [a]
  |- submodule [b]
  |- submodule [c]
  |- submodule [d]
```

When someone pulls down the superproject, they will see a series of empty folders for each submodule. They can then *git submodule init* all of those that they wish to utilize.

4.10 Submodules

A *git archive* is said to become a *submodule* the second after you execute *git submodule add* in another git repository.

4.11 Work Flow

The work flow of superprojects, and submodules should generally adhere to the following:

1. Make change in submodule
2. *git commit* change in submodule
3. *git commit* change in superproject
4. *git submodule update* to push change to the individual repositories that predate the superproject.

4.12 Footnotes

1. **^13** Well that isn't true at all, Git supports this as of v1.5.3, but the official porcelain doesn't. You can *git init* a parent directory, and create your own ".gitmodules", then follow it up with a *git submodule init*. Generally speaking though, what the porcelain doesn't cover is outside of the scope of this book.

13 `#ref_lie_parent`

5 Interacting with other SCMs

This page or section is an undeveloped draft or outline.
You can help to develop the work[1], or you can ask for assistance in the project room[2].

This chapter is about inter-operation between Git and other systems.

Git is a great version control system, but it is not the only one. Many projects still use version control systems that predate git or alternate DSCMs. Thankfully, you can still use git to participate in many of these projects, thanks to programs like `git-svn`[3], `git-cvsimport`, etc.

You can also work with Git directly within many text editors, IDEs and other tools.

5.1 Customizing your command line prompt

This Bash customization script will modify your command line prompt to include some extra details whenever you're in a Git directory. It is based on Mike Stewart's work.[2]

The current branch name is shown in red (if it's got outstanding changes) or green (if there are no changes), followed by the current path. The hostname is also shown in a low-lighted colour at the beginning.

```
Color_Off="\[\033[0m\]"       # Text Reset
IBlack="\[\033[0;90m\]"       # High-intensity black
Green="\[\033[0;32m\]"        # Green
Yellow="\[\033[0;33m\]"       # Yellow
Red="\[\033[0;91m\]"          # Red
Hostname="\h"                 # Hostname (up to the first dot)
PathShort="\w"                # Current working directory (short version)
export PS1=$IBlack$Hostname$Color_Off'$(git branch &>/dev/null;\
if [ $? -eq 0 ]; then \
  echo "$(echo `git status` | grep "nothing \(added \)\?to commit" > /dev/null
 2>&1; \
  if [ "$?" -eq "0" ]; then \
    # Clean repository - nothing to commit
    echo "'$Green'"$(__git_ps1 " (%s)"); \
  else \
    # Changes to working tree
    echo "'$Red'"$(__git_ps1 " {%s}"); \
  fi) '$BYellow$PathShort$Color_Off'\$ "; \
else \
```

1 https://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger/Git&action=edit
2 https://en.wikibooks.org/wiki/Wikibooks:PROJECTS
3 https://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger/git-svn&action=edit&redlink=1

```
  # Prompt when not in GIT repo
  echo " '$Yellow$PathShort$Color_Off'\$ "; \
fi)'
```

5.2 References

1. [4]
2. Stewart, Mike, *Ultimate GIT PS1 bash prompt*[5]

5.3 Overview

Subversion is an extremely popular version control system, and there are many OSS and proprietary projects that use it. Git comes with an excellent utility, `git-svn`, that allows a user to both track a project maintained in a subversion repository, as well as participate. Users can generate local patches to send to a mailing list, or even commit changes directly back into the repository (given that they have commit access, of course).

5.4 Getting Started

To begin using git with a subversion-hosted project, you must create a local repository for your files, as well as configure `git-svn`. Typically you'll use commands much like the following:

```
mkdir project
cd project
git-svn init <url to repository root> -T/path/to/trunk
git-svn fetch -r <first rev>:HEAD
```

Usually when working with subversion repositories, you're given the full project URL. To determine the url to the repository root, you can issue the following command:

```
svn info <full project URL>
```

There will be a line in the output stating the repository root. The path to trunk is simply the rest of the URL that follows. It is possible to simply give `git-svn` the full project URL, but doing it this way gives you greater flexibility should you end up working on subversion branches down the line.

4 https://git-scm.com/docs/gitattributes
5 http://mediadoneright.com/content/ultimate-git-ps1-bash-prompt

Also note the ”first rev” argument. You could simply use ”1”, as that is guaranteed to work, but it would likely take a very long time (especially if the repository is over a slow network). Usually, for an entrenched project, the last 10-50 revs is sufficient. Again, `svn info` will tell you what the most recent revision is.

5.5 Interacting with the repository

Chances are if you're using git instead of svn to interact with a subversion repository, its because you want to make use of offline commits. This is very easy to do, if you keep a few caveats in mind. Most important is to never use ”git pull” while in a branch from which you plan to eventually run `git-svn dcommit`. Merge commits have a tendency to confuse git-svn, and you're able to do most anything you'd need without git pull.

The most common task that I perform is to combine the change(s) I'm working on with the upstream subversion changes. This is the equivalent to an `svn update`. Here's how its done:

```
git stash  # stash any changes so you have a clean tree
git-svn fetch # bring down the latest changes
git rebase trunk
git stash apply
```

The first and last steps are unnecessary if your tree is clean to begin with. That leaves ”git rebase trunk” as the primary operation. If you're unfamiliar with rebasing, you should go read the documentation for git-rebase[6]. The jist of it is that your local commits are now on top of svn HEAD.

5.6 Dealing with local changes

It often happens that there are some changes you want to have in a repository file that you do not want to propagate. Usually this happens with configuration files, but it could just as easily be some extra debugging statements or anything else. The danger of committing these changes is that you'll run ”git-svn dcommit” in the branch without weeding out your changes. On the other hand, if you leave the changes uncommitted, you lose out on git's features for those changes, and you'll have to deal with other branches clashing with the changes. A dilemma!

There are a couple of solutions to this issue. Which one works better is more a matter of taste than anything. The first approach is to keep a ”local” branch for each branch that you want to have local changes. For example, if you have local changes that you want in the branch ”foo”, you would create a branch ”foo-local” containing the commit(s) with the changes you want to keep local. You can then use rebase to keep ”foo” on top of ”foo-local”. e.g.:

```
git rebase trunk foo-local
git rebase foo-local foo
```

6 `http://www.kernel.org/pub/software/scm/git/docs/git-rebase.html`

As the example code implies, you'll still spend most of your time with ”foo” checked out, rather than ”foo-local.” If you decide on a new change that you want to keep locally, you're again faced with two choices. You can checkout ”foo-local” and make the commit, or you can make the commit on ”foo” and then cherry-pick the commit from foo-local. You would then want to use git-reset[7] to remove the commit from ”foo”.

As an alterative to the rebase-centric approach, there is a merge-based method. You still keep your local changes on a separate branch, as before. With this method, however, you don't have to keep ”foo” on top of ”foo-local” with rebase. This is an advantage, because 1) its more to type, and 2) historically, rebase has often asked you to resolve the same conflict twice if any conflicts occur during the first rebase.

So instead of using rebase, you create yet another branch. I call this the ”build” branch. You start the build branch at whatever commit you want to test. You can then ”git merge” the local branch, bringing all your changes into one tree. ”But I thought you should avoid merge?” you ask. The reason I like to call this branch the ”build” branch is to dissuade me from using ”git-svn dcommit” from it. As long as its not your intention to run dcommit from the branch, the use of merge is acceptable.

This approach can actually be taken a step further, making it unnecessary to rebase your topic branch ”foo” on top of trunk every day. If you have several topic branches, this frequent rebasing can become quite a chore. Instead:

```
git checkout build
git reset --hard trunk # Make sure you dont have any important changes
git merge foo foo-local # Octopus merges are fun
```

Now build contains the changes from trunk, foo, and foo-local! Often I'll keep several local branches. Perhaps one branch has your local configuration changes, and another has extra debugging statements. You can also use this approach to build with several topic branches in the tree at once:

```
git merge topic1 topic2 config debug...
```

Unfortunately, the octopus merge is rather dumb about resolving conflicts. If you get any conflicts, you'll have to perform the merges one at a time:

```
git merge topic1
git merge topic2
git merge local
...
```

5.7 Sending changes upstream

Eventually, you'll want your carefully crafted topic branches and patch series to be integrated upstream. If you're lucky enough to have commit access, you can run ”git-svn dcommit”. This will take each local commit in the current branch and commit it to subversion. If you had three local commits, after dcommit there would be three new commits in subversion.

7 http://www.kernel.org/pub/software/scm/git/docs/git-reset.html

For the less fortunate, your patches will probably have to be submitted to a mailing list or bug tracker. For that, you can use git-format-patch[8]. For example, keeping with the three-local-commits scenario above:

```
git format-patch HEAD~3..
```

The result will be three files in $PWD, 0001-commit-name.patch, 0002-commit-name.patch, and 0003-commit-name.patch. You're then free to mail these patches of or attach them to a bug in Bugzilla. If you'll be mailing the patches, however, git can help you out even a little further. There is the git-send-email[9] utility for just this situation:

```
git send-email *.patch
```

The program will ask you a few questions, most important where to send the patches, and then mail them off for you. Piece of cake!

Of course, this all assumes that you have your patch series in perfect working order. If this is not the case, you should read about "git rebase -i[10]".

5.8 Examples

Get Pywikipedia:

```
$ git svn init http://svn.wikimedia.org/svnroot/pywikipedia/trunk/pywikipedia/
Initialized empty Git repository in .../.git/
$ git svn fetch -r 1:HEAD
...
r370 = 318fb412e5d1f1136a92d079f3607ac23bde2c34 (refs/remotes/git-svn)
        D       treelang_all.py
        D       treelang.py
W: -empty_dir: treelang.py
W: -empty_dir: treelang_all.py
r371 = e8477f292b077f023e4cebad843e0d36d3765db8 (refs/remotes/git-svn)
        D       parsepopular.py
W: -empty_dir: parsepopular.py
r372 = 8803111b0411243af419868388fc8c7398e8ab9d (refs/remotes/git-svn)
        D       getlang.py
W: -empty_dir: getlang.py
r373 = ad935dd0472db28379809f150fcf53678630076c (refs/remotes/git-svn)
        A       splitwarning.py
...
```

Get AWB (AutoWikiBrowser):

```
$ git svn init svn://svn.code.sf.net/p/autowikibrowser/code/
Initialized empty Git repository in .../.git/
$ git svn fetch -r 1:HEAD
...
r15 = 086d4ff454a9ddfac92edb4013ec845f65e14ace (refs/remotes/git-svn)
        M       AWB/AWB/Main.cs
        M       AWB/WikiFunctions/WebControl.cs
r16 = 14f49de6b3c984bb8a87900e8be42a6576902a06 (refs/remotes/git-svn)
```

8 http://www.kernel.org/pub/software/scm/git/docs/git-format-patch.html
9 http://www.kernel.org/pub/software/scm/git/docs/git-send-email.html
10 https://en.wikibooks.org/wiki/Git/Rebasing

```
        M       AWB/AWB/ExitQuestion.Designer.cs
        M       AWB/WikiFunctions/GetLists.cs
        M       AWB/WikiFunctions/Tools.cs
r17 = 8b58f6e5b21c91f0819bea9bc9a8110c2cab540d (refs/remotes/git-svn)
        M       AWB/AWB/Main.Designer.cs
        M       AWB/AWB/Main.cs
        M       AWB/WikiFunctions/GetLists.cs
r18 = 51683925cedb8effb274fadd2417cc9b1f860e3c (refs/remotes/git-svn)
        M       AWB/AWB/specialFilter.Designer.cs
        M       AWB/AWB/specialFilter.cs
r19 = 712edb32a20d6d2ab4066acf056f14daa67a9d4b (refs/remotes/git-svn)
        M       AWB/WikiFunctions/WPEditor.cs
r20 = 3116588b52a8e27e1dc72d25b1981d181d6ba203 (refs/remotes/git-svn)
...
```

Beware, this downloading operation can take one hour.

6 Hosting

Here is where you can find a list of public repositories. Feel free to pull down a sample project and play with it.

6.1 Free, public, and open source

This is a current list of repositories that will host anything (within reason) free under the condition that it is appropriately licensed and open-source.

1. NotABug[1]
2. git.sv.gnu.org[2] Free Software Foundation's official GNU Savannah Git hosting
3. repo.or.cz[3] Git's primary free repository is maintained by Petr Baudis, who is also the maintainer of the git home page. Numerous notable projects are synced to that repository such as Postgres, and Linus's copy of the linux kernel tree. The repository and all of its projects can be inspected on the web thanks to the mod_perl gitweb interface.
4. Codeberg[4] (non-profit) provides free Git hosting in Europe, for FOSS projects only[1]
5. disroot[5]

The following sites offer free hosting and other services regardless of a project's license.

1. GitHub[6]
2. GitLab[7][2]
3. Bitbucket[8][2]

6.2 Birds of a Feather (BOF)

Please do not ask to create a project on these public repositories unless your project relates to that of the repository.

1. git.kernel.org[9] Official Kernel git repository
2. gitweb.freedesktop.org[10] X Window System specific git repository

---

1 https://notabug.org/
2 http://git.sv.gnu.org/
3 http://repo.or.cz
4 https://codeberg.org/
5 https://disroot.org/en/services/git
6 https://github.com/
7 https://gitlab.com
8 https://www.atlassian.com/software/bitbucket
9 http://git.kernel.org/
10 http://gitweb.freedesktop.org/

3. git.debian.org[11] Debian specific, hosted by Debian.

6.3 Public project specific repositories

This section is a stub.
You can help Wikibooks by expanding it[12].

6.4 Other

1. github.com[13] Offers free and paid repository management with some custom tools.
2. Assembla[14] Offers free public or private repository(500Mb) with Trac[15] tickets or integrated tickets[16]. Commercial subscriptions costs 19$/month for 2Gb of disk space.

11 http://git.debian.org/
12 https://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger/Git&action=edit
13 https://github.com/
14 http://www.assembla.com/free_git_hosting
15 http://trac.edgewall.org/
16 http://www.assembla.com/tour/tools_tickets

7 Setting up a Server

A Git server is not supposed to host any uncommitted file, so its repositories should be initialized with ”bare”:

```
git init --bare /repositories/repo1
```

Now its files are encrypted and can't be read as flat files from the server.

The distributed repositories can then be initialized with `git clone`, updated with `git pull` , and submitted to the server with `git push`.

To avoid any user to erase the server branches when pushing, the branches can be locked, forcing the users to create some pull requests[1] with their changes to validate before merging. **`git-daemon`** is a simple server for git repositories.

The daemon makes git repositories available over the `git://` protocol, which is very efficient, but insecure. Where security is a concern, pull with SSH[2] instead. The `git://` protocol listens on port 9418 by default.

The daemon *can* be configured to allow pushing, however there is **no authentication whatsoever**, so that is almost always a very bad idea. It may be appropriate in a closed LAN environment, where everyone can be trusted, however when in doubt, SSH should be used to push. **Gitosis** is a tool to secure centralized Git repositories, permitting multiple maintainers to manage the same project at once, by restricting the access to only over a secure network protocol.

7.1 Installing Gitosis

7.1.1 Checkout the Gitosis Repository

To install Gitosis, you first must have the Git client installed[3]. Once installed, checkout a copy of Gitosis from its repository:

```
git clone git://eagain.net/gitosis.git
```

Install:

```
cd gitosis
python setup.py install
```

1 https://en.wikipedia.org/wiki/pull_request
2 https://en.wikibooks.org/wiki/SSH
3 https://en.wikibooks.org/wiki/Git/Obtaining_Git

7.1.2 Create a User to Manage the Repositories

Create a user to manage the repositories:

```
sudo adduser \
  --system \
  --shell /bin/sh \
  --gecos 'git version control' \
  --group \
  --disabled-password \
  --home /home/git \
  git
```

If you don't already have a public RSA key, create one on your local computer:

```
ssh-keygen -t rsa
```

7.1.3 Copying Your Public Key to the Gitosis Server

Copy this key to the Gitosis server. Assuming you are in your home directory:

```
scp .ssh/id_rsa.pub user@example.com:/tmp
```

7.2 Setting up Gitosis

7.2.1 Initializing Gitosis

Initialize Gitosis:

```
sudo -H -u git gitosis-init < /tmp/id_rsa.pub
```

Upon success, you will see:

```
Initialized empty Git repository in ./
Initialized empty Git repository in ./
```

Ensure the Git post-update hook has the correct permissions:

```
sudo chmod 755 /home/git/repositories/gitosis-admin.git/hooks/post-update
```

7.3 Configuring Gitosis

7.3.1 Clone the Gitosis Repository

Gitosis creates its own Git repository. To configure Gitosis, you will clone this repository, set your configuration options, then push your configuration back to the Gitosis server.

Cloning the Gitosis repository:

```
git clone git@example.com:gitosis-admin.git
cd gitosis-admin
```

7.3.2 Creating a Repository

Edit `gitosis.conf`

An example of a default gitosis.conf:

```
[gitosis]


[group gitosis-admin]
writable = gitosis-admin
members = jdoe
```

7.3.3 Defining Groups, Members, Permissions, and Repositories

You can define groups of members and what permissions they will have to repositories like so:

```
[group blue_team]
members = john martin stephen
writable = tea_timer coffee_maker
```

In this example, anyone in the group `blue_team`, in this case john, martin, and stephen, will be able to write to the Git repositories `tea_timer` and `coffee_maker`

Save, commit, and push this file.

```
git commit -am "Give john, martin, and stephen access to the repositories
tea_timer and coffee_maker."
git push
```

7.3.4 Creating a Repository

Next, create one of the repositories. You'll want to change to the directory where you you want to store your local copy of the Git repository first.

Create the repository:

```
mkdir tea_timer
cd tea_timer
git init
git remote add origin git@example.com:tea_timer.git
# Add some files and commit.
git push origin master:refs/heads/master
# The previous line links your local branch master to the remote branch master
so you can automatically fetch and merge with git pull.
```

7.3.5 Adding Users to a Repository

Users are identified by their public RSA keys. Gitosis keeps these keys inside the directory `keydir` within the gitosis-admin repository. Users are linked to their Git username by the

name of the key file. For example, adding an RSA key to `keydir/john.pub` will link the user john to the machine defined by the RSA within `john.pub`. Keys *must* end in .pub!

Add a user:

```
cd gitosis-admin
cp /path/to/rsa/key/john.pub keydir/
git add keydir/*
git commit -am "Adding the john account."
git push
```

John can now clone the git repositories he has access to as defined by `gitosis.conf`. In this case, he can both read and write to the repository as he has `writable` permissions.

7.4 External Links

- Scie.nti.st[4]

Gerrit[5] is a web-based code review tool for projects using the Git VCS. It allows both a streamlined code review process, and a highly-configurable hierarchy for project members. Typically, any user may submit patches (”changesets”) to the server for review. Once someone has reviewed the changes to a sufficient degree, they are merged into the main line of development, which can then be pulled.

7.5 Overview

7.5.1 Implementation

Server

Gerrit uses a dedicated Jetty server[6], which is typically accessed via reverse proxy. Here's an example configuration for Apache[7]:

```
<VirtualHost *>
    ProxyRequests Off
    ProxyVia Off
    ProxyPreserveHost On
    <Proxy *>
        Order deny,allow{{typo help inline|reason=similar to deny,
allow|date=August 2022}}
        Allow from all
    </Proxy>

    # Reverse-proxy these requests to the Gerrit Jetty server
    RedirectMatch  ^/gerrit$               /gerrit/
    ProxyPass       /gerrit/               http://127.0.0.1:8081/gerrit/
</VirtualHost>
```

4 http://scie.nti.st/2007/11/14/hosting-git-repositories-the-easy-and-secure-way
5 https://code.google.com/p/gerrit/
6 https://en.wikipedia.org/wiki/Jetty_(Web_server)
7 https://en.wikibooks.org/wiki/Apache

JGit

Gerrit uses JGit, a Java[8] implementation of git. This has several limitations: a sacrifice in speed, and some unimplemented features[3]. For example, content-level merges are not supported in JGit - users must pull and merge changes, then re-upload them to the Gerrit server in cases where content-level merges are required.

Permissions model

Gerrit's permissions model allows a highly configurable hierarchy regarding who can submit patches, and who can review patches. This can be flattened out as desired, or ramped up, depending on the development model used for each project.

Hooks

The server allows scripts to run in response to certain events. Hook scripts live in $GIT_INSTALL_BASE/hooks, and must be set executable on Unix systems. A hook could - for instance - allow a project to install an automated gatekeeper to vote +2 'submit approved' when sufficient +1 votes 'looks good to me' have been received:

```
#!/usr/bin/perl
#
# comment-added: hook for a +2 approval from a simple quorum of +1 votes.
#
# (c) 2012 Tim Baverstock
# Licence: Public domain. All risk is yours; if it breaks, you get to keep both
 pieces.

$QUORUM = 2; # Total number of +1 votes causing a +2
$PLEBIANS = 'abs(value) < 2'; # or 'value = 1' to ignore -1 unvotes
$AUTO_SUBMIT_ON_QUORACY = '--submit'; # or '' for none
$AND_IGNORE_UPLOADER = 'and uploader_account_id != account_id'; # or '' to let
 uploaders votes count

$GERRIT_SSH_PORT = 29418;
$SSH_PRIVATE_KEY = '/home/gerrit2/.ssh/id_rsa';
$SSH_USER_IN_ADMIN_GROUP = 'devuser';

# Hopefully you shouldn't need to venture past here.

$SSH = "ssh -i $SSH_PRIVATE_KEY -p $GERRIT_SSH_PORT
 $SSH_USER_IN_ADMIN_GROUP\@localhost";

$LOG = "/home/gerrit2/hooks/log.comment-added";
open LOG, ">>$LOG" or die;

sub count_of_relevant_votes {
        # Total selected code review votes for this commit
        my $relevance = shift;
        $query = "
                select sum(value) from patch_sets, patch_set_approvals
                where patch_sets.change_id = patch_set_approvals.change_id
                and patch_sets.patch_set_id = patch_set_approvals.patch_set_id
                and revision = '$V{commit}'
                and category_id = 'CRVW'
                and $relevance
```

8 https://en.wikibooks.org/wiki/Java

```
                $AND_IGNORE_UPLOADER
                ;";
        $command = "$SSH \"gerrit gsql -c \\\"$query\\\"\"";
        #print LOG "FOR... $command\n";
        @lines = qx($command);
        chomp @lines;
        #print LOG "GOT... ", join("//", @lines), "\n";
        # 0=headers 1=separators 2=data 3=count and timing.
        return $lines[2];
}

sub response {
        my $review = shift;
        return "$SSH 'gerrit review --project=\"$V{project}\" $review
 $V{commit}'";
}

# ######################
# Parse options

$key='';
while ( $_ = shift @ARGV ) {
        if (/^--(.*)/) {
                $key = $1;
        }
        else {
                $V{$key} .= " " if exists $V{$key};
                $V{$key} .= $_;
        }
}
#print LOG join("\n", map { "$_ = '$V{$_}'" } keys %V), "\n";

# ######################
# Ignore my own comments

$GATEKEEPER="::GATEKEEPER::";
if ($V{comment} =~ /$GATEKEEPER/) {
        print LOG localtime() . "$V{commit}: Ignore $GATEKEEPER comments\n";
        exit 0;
}

# ######################
# Forbear to analyse anything already +2'd

$submittable = count_of_relevant_votes('value = 2');
if ($submittable > 0) {
        print LOG localtime() . "$V{commit} Already +2'd by someone or
 something.\n";
        exit 0;
}

# ######################
# Look for a consensus amongst qualified voters.

$plebicite = count_of_relevant_votes($PLEBIANS);

#if ($V{comment} =~ /TEST:(\d)/) {
#        $plebicite=$1;
#}

# ######################
# If there's a quorum, approve and submit.

if ( $plebicite >= $QUORUM ) {
        $and_submitting = ($AUTO_SUBMIT_ON_QUORACY ? " and submitting" : "");
        $review = " --code-review=+2 --message=\"$GATEKEEPER
```

```
 approving$and_submitting due to $plebicite total eligible votes\"
 $AUTO_SUBMIT_ON_QUORACY";
}
else {
        $review = " --code-review=0 --message=\"$GATEKEEPER ignoring $plebicite
 total eligible votes\"";
        print LOG localtime() . "$V{commit}: $review\n";
        exit 0; # Perhaps don't exit here: allow a subsequent -1 to remove the
 +2.
}

$response = response($review);

print LOG localtime() . "RUNNING: $response\n";
$output = qx( $response 2>&1   );
if ($output =~ /\S/) {
        print LOG localtime() . "$V{commit}: output from commenting: $output";
        $response = response(" --message=\"During \Q$review\E: \Q$output\E\"");
        print LOG localtime() . "WARNING: $response\n";
        $output = qx( $response 2>&1   );
        print LOG localtime() . "ERROR: $output\n";
}

exit 0;
```

7.6 Setup

7.6.1 Importing project into Gerrit

Whether or not you're permitted to do this depends on which access group(s) you're in, and where you're trying to do this. It is most useful for pushing pre-existing repos to the server, but it can in theory be used whenever you want to push changes which are not to be reviewed.

Use:

```
$ git push gerrit:project HEAD:refs/heads/master
```

since you want to directly push into the branch, rather than create code reviews. Pushing to **`refs/for/*`** creates code reviews which must be approved and then submitted. Pushing to **`refs/heads/*`** bypasses review entirely, and just enters the commits directly into the branch. The latter does not check committer identity, making it appropriate for importing past project history.

The correct permission setup can be found here: [9]. In addition, "Push Merge Commit" for "refs/*" may be needed for some repositories (for details see [10]).

9 http://stackoverflow.com/questions/8353988/how-to-upload-a-git-repo-to-gerrit
10 https://code.google.com/p/gerrit/issues/detail?id=1072

7.7 Use

7.7.1 Registering

7.7.2 Submitting changes for review

Simply push into the project's magical `refs/for/$branch` (typically master) ref using any Git client tool:

```
$ git push ssh://user@host:29418/project HEAD:refs/for/master
```

Each new commit uploaded by the git push client will be converted into a change record on the server. The remote ref `refs/for/$branch` is not actually created by Gerrit, even though the client's status messages may say otherwise. Pushing to this magical branch submits changes for review. Once you have pushed changes, they must be reviewed and submitted for merging to whatever branch they apply to. You can clone/pull from gerrit as you would any other git repo (no refs/for/branch, just use the branch name):

```
$ git clone ssh://user@host:29418/project
$ git pull ssh://user@host:29418/project master:testbranch
```

Gerrit currently has no `git-daemon`[11], so pulling is via ssh, and therefore comparatively slow (but secure). You can run `git-daemon` for the repositories to make them available via `git://`, or configure Gerrit to replicate changes to another git repository, where you can pull from.

Since you will be frequently working with the same Gerrit server, add an SSH host block in ~/.ssh/config to remember your username, hostname and port number. This permits the use of shorter URLs on the command line as shown, such as:

```
$ tail -n 4 ~/.ssh/config
Host gerrit
    Hostname host.com
    Port 29418
    User john.doe
$ git push gerrit:project HEAD:refs/for/master
```

Alternatively, you can also configure your remotes in git's configuration file by issuing:

```
$ git config remote.remote_name.fetch +refs/heads/*:refs/remotes/origin/*
$ git config remote.remote_name.url ssh://user@host:29418/project_name.git[12]
```

This should be done automatically for you if you've started your local repository off of Gerrit's project repository using

```
$ git clone ssh://user@host:29418/project_name.git[13]
```

Note that the Gerrit server has its own sshd with different host keys. Some ssh clients will complain about this bitterly.

11 https://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger/git-daemon&action=edit&redlink=1

12 ssh://user@host:29418/project_name.git

13 ssh://user@host:29418/project_name.git

Re-submitting a changeset

This is useful when there are issues with a commit you pushed. Maybe you caught them, maybe the reviewer did – either way, you want to submit changes for review, replacing the bad changes you submitted previously. This keeps code review in one place, streamlining the process. First, squash your changes into one commit using git rebase -i – you will probably want to change the commit message.

This doesn't actually replace the previous push, it just adds your updated changeset as a newer version.

You can provide a Change-Id line in your commit message: it must be in the bottom portion (last paragraph) of a commit message, and may be mixed together with the Signed-off-by, Acked-by, or other such footers. The Change-Id is available in the metadata table for your initial pushed commit. In this case, Gerrit will automatically match this changeset to the previous one.

Alternatively, you can push to a special location: the `refs/changes/*` branch. To replace changeset 12345, you push to `refs/changes/12345`. This number can be found in the URL when looking at the changeset you wish to replace: `#change,12345`. You can also find it in the "download" section for the changeset: `...refs/changes/45/12345/1` (choose the middle number: ignore `/45/` and omit the trailing `/1`). In this case, your push becomes:

```
$ git rebase -i HEAD~2 # squash the relevant commits, probably altering the
 commit message
$ git push gerrit:project
 a95cc0dcd7a8fd3e70b1243aa466a96de08ae731:refs/changes/12345
```

7.7.3 Reviewing and merging changes

7.8 See also

- Current official documentation[14]
- Download page[15]
- MW:Gerrit[16]

7.9 References

1. "The Top 10 GitHub Alternatives"[17]. 2023.
2. "Awesome GitHub Alternatives"[18].
3. Re: Failure to submit due to path conflict but no real conflict.[19] (Shawn Pearce)

14 https://gerrit-review.googlesource.com/Documentation/index.html
15 http://gerrit-releases.storage.googleapis.com/index.html
16 https://www.mediawiki.org/wiki/Gerrit
17 https://www.wearedevelopers.com/magazine/top-github-alternatives
18 https://github.com/ianchanning/awesome-github-alternatives
19 https://groups.google.com/group/repo-discuss/msg/2ee99ed99609bb4b

Wikipedia[20] has related information at ***Git (software)***[21]

Instead of having to resort to a hosting company to store your central repository, or to rely on a central server or internet connection to contribute changes to a project, it's quite possible to use removable memory to exchange and update local repositories.

The basic steps are:

1. Mount the removable memory on a pre-determined path
2. Setup a bare repository in the removable memory
3. Add the repository in the removable memory as a remote

Throughout this article, it's assumed that the repository will be mounted in /media/repositories.

7.10 Starting from scratch

Let's assume a brand new project is being started, titled Foo. The people working on project Foo will use a removable memory device as a central repository for the whole project. For this reason, let's create a fresh repository in a USB stick.

To start off, instead of relying on a path name generated automatically by the OS, the USB stick will be mounted on a custom path. Let's assume the USB pen drive is located at /dev/sdb1, and the intended mount point is /media/repositories. Another important aspect is that the USB pen drive needs to be mounted with proper permissions, so that it doesn't cause problems within the repository. One way to avoid these issues is to mount the USB drive with your own user ID and group ID, which can be achieved through the following commands:

```
$ sudo mkdir /media/repositories
$ my_uid=`id -u`
$ my_gid=`id -g`
$ mount -o "uid=$my_uid,gid=$my_gid" /dev/sdb1 /media/repositories
```

Having mounted the USB drive on the desired path, let's create a bare repository.

```
$ cd /media/repositories
$ mkdir /media/repositories/foo
$ git init --bare /media/repositories/foo
Initialized empty Git repository in /media/repositories/foo
```

With this step, a repository has been created in the USB memory drive. Now, all that's left is to mount the USB pen on any computer on a specific path, clone the repository, and work away.

```
$ git clone /media/repositories/foo
Cloning into 'foo'...
warning: You appear to have cloned an empty repository.
done.
```

20 https://en.wikipedia.org/wiki/
21 https://en.wikipedia.org/wiki/Git_(software)

Done.

In case the mount point changes to another path (in some cases, auto-mounting does that), the repository's URL can be set through the following command:

```
$ git remote set-url origin file://<path to new mount point>
```

7.11 Pushing local repository to a USB stick

Let's assume you've been working on a project, and you already have a working git repository that you've been working on your desktop. Now, you've decided that want to keep track and update several concurrent versions of that repository without having to use a network connection. One possible solution is to start a Git repository on a USB key, and use that USB key as a centralized repository where everyone can push their contributions and update their local version.

To start a Git repository on a USB stick, first let's mount the USB stick on a selected path. This is achieved through the process described in the previous section.

```
$ sudo mkdir /media/repositories
$ my_uid=`id -u`
$ my_gid=`id -g`
$ mount -o "uid=$my_uid,gid=$my_gid" /dev/sdb1 /media/repositories
```

Let's assume that the project tree is located in ~/devel/foo. To start a repository on /media/repositories/foo, run the following command:

```
git clone --bare ~/devel/foo foo
```

That's it.

Now, you can clone the Git repository stored in the USB drive and continue working on your project.

```
$ git clone /media/repositories/foo/
Cloning into 'foo'...
done.
```

If you wish to add to your local repository the newly created USB repository as a remote repository,

```
$ git remote add usb file:///media/repositories/foo
```

To establish the master branch of the USB repository as the upstream branch of your local master branch, the contents of the newly added remote branch must be fetched and the upstream branch must be specified. This step is performed by applying the following commands:

```
$ git fetch usb
From file:///media/repositories/foo
 * [new branch]      master     -> usb/master
```

```
$ git branch --set-upstream-to=usb/master
Branch master set up to track remote branch master from usb.
```

8 Git component programs

Git has many component program, the following is an unabridged list of the them as of `git version 1.5.2.5`.

git
git-add
git-add–interactive
git-am
git-annotate
git-apply
git-applymbox
git-applypatch
git-archive
git-bisect
git-blame
git-branch
git-bundle
git-cat-file
git-check-attr
git-checkout
git-checkout-index
git-check-ref-format
git-cherry
git-cherry-pick
git-clean
git-clone
git-commit
git-commit-tree
git-config
git-convert-objects
git-count-objects
git-daemon
git-describe
git-diff
git-diff-files
git-diff-index
git-diff-tree
git-fast-import
git-fetch
git-fetch-pack
git-fetch–tool
git-fmt-merge-msg
git-for-each-ref
git-format-patch
git-fsck
git-fsck-objects
git-gc
git-get-tar-commit-id
git-grep
git-hash-object
git-http-fetch
git-http-push
git-imap-send
git-index-pack
git-init
git-init-db
git-instaweb
git-local-fetch
git-log
git-lost-found
git-ls-files
git-ls-remote
git-ls-tree
git-mailinfo
git-mailsplit
git-merge
git-merge-base
git-merge-file
git-merge-index
git-merge-octopus
git-merge-one-file
git-merge-ours
git-merge-recursive
git-merge-resolve
git-merge-stupid
git-merge-subtree
git-mergetool
git-merge-tree
git-mktag
git-mktree
git-mv
git-name-rev
git-pack-objects
git-pack-redundant
git-pack-refs
git-parse-remote
git-patch-id
git-peek-remote
git-prune
git-prune-packed
git-pull
git-push
git-quiltimport
git-read-tree
git-rebase
git-receive-pack
git-reflog
git-relink
git-remote
git-repack
git-repo-config
git-request-pull
git-rerere
git-reset
git-revert
git-rev-list
git-rev-parse
git-rm
git-runstatus
git-send-pack
git-shell
git-shortlog
git-show
git-show-branch
git-show-index
git-show-ref
git-sh-setup
git-ssh-fetch
git-ssh-pull
git-ssh-push
git-ssh-upload
git-status
git-stripspace
git-symbolic-ref
git-tag
git-tar-tree
git-unpack-file
git-unpack-objects
git-update-index
git-update-ref
git-update-server-info
git-upload-archive
git-upload-pack
git-var
git-verify-pack
git-verify-tag
git-whatchanged
git-write-tree

9 Getting help

9.1 IRC

- #git on irc.freenode.net[1] (webchat[2])

9.2 Forums & mailing lists

- Mailing list: gitvger.kernel.org

9.3 Web pages

- FAQs
  - FAQ from kernel.org[3]
  - A FAQ with fast answers[4]
- Tutorials
  - Official Git documentation[5]
  - ProGit[6]
  - A very nice and concise tutorial[7] that also explains advanced features like branches, rebasing etc.
  - Git Community Book[8]
- For users of other VCS
  - Crash course for SVN users[9]
- Misc
  - Seminars and presentations[10] about Git
  - Lots of Git-related links[11]
  - The Git Parable[12]

- This page was last edited on 27 June 2024, at 11:57.

1 irc://irc.freenode.net/git
2 http://webchat.freenode.net/?channels=git
3 http://git.wiki.kernel.org/index.php/GitFaq
4 http://www.sourcemage.org/Git_Guide
5 http://www.kernel.org/pub/software/scm/git/docs/index.html
6 http://progit.org/
7 http://www-cs-students.stanford.edu/~blynn/gitmagic/
8 http://book.git-scm.com/index.html
9 http://git-scm.org/course/svn.html
10 http://git.wiki.kernel.org/index.php/GitLinks#Seminars_and_presentations
11 http://git.wiki.kernel.org/index.php/GitLinks
12 http://tom.preston-werner.com/2009/05/19/the-git-parable.html

- Text is available under the Creative Commons Attribution-ShareAlike License[13]; additional terms may apply. By using this site, you agree to the Terms of Use[14] and Privacy Policy.[15]

13 http://creativecommons.org/licenses/by-sa/4.0/
14 http://foundation.wikimedia.org/wiki/Special:MyLanguage/Policy:Terms_of_Use
15 http://foundation.wikimedia.org/wiki/Special:MyLanguage/Policy:Privacy_policy

10 Contributors

Edits	User
1	06230715alita[1]
14	Adrignola[2]
1	Andrej Shadura[3]
8	Avicennasis[4]
1	Backfromquadrangle[5]
1	Chuckhoffmann[6]
2	Covracer[7]
1	DKEdward[8]
6	DannyS712[9]
3	Dark knight[10]
1	Darklama[11]
1	DavidCary[12]
1	Dbjohn[13]
1	Devourer09[14]
2	Dirk Hünniger[15]
84	EvanCarroll[16]
2	Everrob[17]
4	Fishpi[18]
1	G Sisson[19]
3	Garrydolley[20]

1 https://en.wikibooks.org/w/index.php%3ftitle=User:06230715alita&action=edit&redlink=1
2 https://en.wikibooks.org/wiki/User:Adrignola
3 https://en.wikibooks.org/wiki/User:Andrej_Shadura
4 https://en.wikibooks.org/wiki/User:Avicennasis
5 https://en.wikibooks.org/wiki/User:Backfromquadrangle
6 https://en.wikibooks.org/wiki/User:Chuckhoffmann
7 https://en.wikibooks.org/w/index.php%3ftitle=User:Covracer&action=edit&redlink=1
8 https://en.wikibooks.org/w/index.php%3ftitle=User:DKEdward&action=edit&redlink=1
9 https://en.wikibooks.org/wiki/User:DannyS712
10 https://en.wikibooks.org/wiki/User:Dark_knight
11 https://en.wikibooks.org/wiki/User:Darklama
12 https://en.wikibooks.org/wiki/User:DavidCary
13 https://en.wikibooks.org/w/index.php%3ftitle=User:Dbjohn&action=edit&redlink=1
14 https://en.wikibooks.org/w/index.php%3ftitle=User:Devourer09&action=edit&redlink=1
15 https://en.wikibooks.org/wiki/User:Dirk_H%25C3%25BCnniger
16 https://en.wikibooks.org/wiki/User:EvanCarroll
17 https://en.wikibooks.org/w/index.php%3ftitle=User:Everrob&action=edit&redlink=1
18 https://en.wikibooks.org/w/index.php%3ftitle=User:Fishpi&action=edit&redlink=1
19 https://en.wikibooks.org/w/index.php%3ftitle=User:G_Sisson&action=edit&redlink=1
20 https://en.wikibooks.org/w/index.php%3ftitle=User:Garrydolley&action=edit&redlink=1

1 Gbd~enwikibooks[21]
1 Haaninjo[22]
22 Hannes Röst[23]
45 JackPotte[24]
2 Javornikolov[25]
1 Kittycataclysm[26]
45 Ldo[27]
1 Leaderboard[28]
2 Lucamilanesio[29]
1 MarcGarver[30]
1 Marksta~enwikibooks[31]
2 MathXplore[32]
8 Mecanismo[33]
61 Mike.lifeguard[34]
1 Naught101[35]
1 PokestarFanBot[36]
2 SHB2000[37]
3 Samwilson[38]
2 SleekWeasel~enwikibooks[39]
2 Soloturn~enwikibooks[40]
6 Soul windsurfer[41]
17 Srwalter[42]
1 Tech201805[43]
2 TheAverageDolphin[44]

21 https://en.wikibooks.org/w/index.php%3ftitle=User:Gbd~enwikibooks&action=edit&redlink=1
22 https://en.wikibooks.org/w/index.php%3ftitle=User:Haaninjo&action=edit&redlink=1
23 https://en.wikibooks.org/wiki/User:Hannes_R%25C3%25B6st
24 https://en.wikibooks.org/wiki/User:JackPotte
25 https://en.wikibooks.org/w/index.php%3ftitle=User:Javornikolov&action=edit&redlink=1
26 https://en.wikibooks.org/wiki/User:Kittycataclysm
27 https://en.wikibooks.org/wiki/User:Ldo
28 https://en.wikibooks.org/wiki/User:Leaderboard
29 https://en.wikibooks.org/w/index.php%3ftitle=User:Lucamilanesio&action=edit&redlink=1
30 https://en.wikibooks.org/wiki/User:MarcGarver
31 https://en.wikibooks.org/w/index.php%3ftitle=User:Marksta~enwikibooks&action=edit&redlink=1
32 https://en.wikibooks.org/wiki/User:MathXplore
33 https://en.wikibooks.org/wiki/User:Mecanismo
34 https://en.wikibooks.org/wiki/User:Mike.lifeguard
35 https://en.wikibooks.org/w/index.php%3ftitle=User:Naught101&action=edit&redlink=1
36 https://en.wikibooks.org/wiki/User:PokestarFanBot
37 https://en.wikibooks.org/wiki/User:SHB2000
38 https://en.wikibooks.org/wiki/User:Samwilson
39 https://en.wikibooks.org/wiki/User:SleekWeasel~enwikibooks
40 https://en.wikibooks.org/w/index.php%3ftitle=User:Soloturn~enwikibooks&action=edit&redlink=1
41 https://en.wikibooks.org/wiki/User:Soul_windsurfer
42 https://en.wikibooks.org/w/index.php%3ftitle=User:Srwalter&action=edit&redlink=1
43 https://en.wikibooks.org/wiki/User:Tech201805
44 https://en.wikibooks.org/w/index.php%3ftitle=User:TheAverageDolphin&action=edit&redlink=1

1 ThurnerRupert[45]
7 Tim Goddard[46]
5 Tricon[47]
1 Xania[48]

45 https://en.wikibooks.org/wiki/User:ThurnerRupert
46 https://en.wikibooks.org/w/index.php%3ftitle=User:Tim_Goddard&action=edit&redlink=1
47 https://en.wikibooks.org/w/index.php%3ftitle=User:Tricon&action=edit&redlink=1
48 https://en.wikibooks.org/wiki/User:Xania

List of Figures

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-4.0: Creative Commons Attribution ShareAlike 4.0 License. https://creativecommons.org/licenses/by-sa/4.0/deed.en
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-1.0: Creative Commons Attribution 1.0 License. https://creativecommons.org/licenses/by/1.0/deed.en
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- cc-by-4.0: Creative Commons Attribution 4.0 License. https://creativecommons.org/licenses/by/4.0/deed.de
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised

is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.

- LFK: Lizenz Freie Kunst. `http://artlibre.org/licence/lal/de`
- CFR: Copyright free use.
- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[49]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

49 Chapter 11 on page 79

1	Walid.bezza[50], Walid.bezza[51]	CC-BY-SA-3.0
2	Walid.bezza[52], Walid.bezza[53]	CC-BY-SA-3.0
3	Walid.bezza[54], Walid.bezza[55]	CC-BY-SA-3.0
4	Walid.bezza[56], Walid.bezza[57]	CC-BY-SA-3.0
5	Walid.bezza[58], Walid.bezza[59]	CC-BY-SA-3.0
6	Walid.bezza[60], Walid.bezza[61]	CC-BY-SA-3.0
7	JackPotte[62], JackPotte[63]	CC-BY-SA-4.0
8	EBojilova[64], EBojilova[65]	CC-BY-SA-4.0
9	Daniel Kinzler[66], Daniel Kinzler[67]	CC-BY-3.0
10	Vincent Driessen	CC-BY-SA-3.0
11	Ldo[68], Ldo[69]	CC-BY-SA-3.0
12	Ldo[70], Ldo[71]	CC-BY-SA-3.0
13	Ldo[72], Ldo[73]	CC-BY-SA-3.0

50 http://commons.wikimedia.org/w/index.php?title=User:Walid.bezza&action=edit&redlink=1
51 https:///w/index.php?title=User:Walid.bezza&action=edit&redlink=1
52 http://commons.wikimedia.org/w/index.php?title=User:Walid.bezza&action=edit&redlink=1
53 https:///w/index.php?title=User:Walid.bezza&action=edit&redlink=1
54 http://commons.wikimedia.org/w/index.php?title=User:Walid.bezza&action=edit&redlink=1
55 https:///w/index.php?title=User:Walid.bezza&action=edit&redlink=1
56 http://commons.wikimedia.org/w/index.php?title=User:Walid.bezza&action=edit&redlink=1
57 https:///w/index.php?title=User:Walid.bezza&action=edit&redlink=1
58 http://commons.wikimedia.org/w/index.php?title=User:Walid.bezza&action=edit&redlink=1
59 https:///w/index.php?title=User:Walid.bezza&action=edit&redlink=1
60 http://commons.wikimedia.org/w/index.php?title=User:Walid.bezza&action=edit&redlink=1
61 https:///w/index.php?title=User:Walid.bezza&action=edit&redlink=1
62 http://commons.wikimedia.org/wiki/User:JackPotte
63 https:///wiki/User:JackPotte
64 http://commons.wikimedia.org/w/index.php?title=User:EBojilova&action=edit&redlink=1
65 https:///w/index.php?title=User:EBojilova&action=edit&redlink=1
66 http://commons.wikimedia.org/wiki/User:Duesentrieb
67 https:///wiki/User:Duesentrieb
68 http://commons.wikimedia.org/wiki/User:Ldo
69 https:///wiki/User:Ldo
70 http://commons.wikimedia.org/wiki/User:Ldo
71 https:///wiki/User:Ldo
72 http://commons.wikimedia.org/wiki/User:Ldo
73 https:///wiki/User:Ldo

11 Licenses

11.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

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

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both

those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public 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.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

11.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

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 noncommercially. Secondarily, 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.

The "publisher" means any person or entity that distributes copies of the Document to the public.

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 under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. 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 1.3 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.

11.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

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

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public 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.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.