git: version control and collaboration

A. J. Roberts

February 26, 2016

The software package git empowers you to easily maintain a history of a project, so you can recover old information for example, and additionally to collaborate with others on the project, so you can all work in parallel for example. These two aspects are introduced separately: Section 2 introduces how to maintain a local history;[1] and Section 3 introduces collaboration over the internet. In this two column format, the left column is for general information and the command line version, whereas the right column is for graphical user interfaces. Get more information via Scott Chacon's surprisingly comprehensible *Pro Git* [`http://git-scm.com/book/`].

Contents

[1] I base this introduction on *Git for the lazy*, `http://www.spheredev.org/wiki/Git_for_the_lazy` [Nov 2011].

1 Installation and initialisation

Download Perhaps download git from `http://git-scm.com/download` To install on a Mac: download the `dmg` file; double click to unpack; look in the Git 'disk symbol' on the desktop; double click on the `pkg` file, and follow instructions. This installs git software accessible as commands from any terminal window.[2]

Introduce yourself to git Set some useful global parameters for git. In a terminal window type the following commands[3]

```
git config --global user.name "Your Name"
git config --global user.email "youremail@place"
git config --global color.ui auto
git config --global color.interactive auto
git config --global core.editor "nano -w"
```

Review these settings, at any time, by executing in a terminal window the command

[2] If the command `git` does not execute, then add the following line to your home `.profile` or `.bash_profile` file: `export PATH=/usr/local/git/bin:$PATH`

[3] The first two commands establish your identity; the third and fourth invoke some pretty colouring for the command interaction; and the last changes the default editor for command interaction to something that is generally easier to use (nano is a free version of pico).

Perhaps use a graphical user interface Many people want a graphical user interface to git. The web has many suggestions, and differing opinions.

I tentatively recommend using *GitHub for the Mac (Windows)* (requires OS 10.6 or later), but use another GUI if you prefer. [2] Download GitHub from `http://mac.github.com/` (`http://windows.github.com/`), and drag into your application folder, and perhaps your Dock. [3]

If not already done, you may introduce yourself to git by running GitHub and entering your name and email in the initialisation information box. But other configuration commands have to be entered from a terminal window.

In this document, remember to distinguish between the github GUI software (most of the time), and the free but public hosting service provided at `github.com` (sometimes).

Other GUI software is SourceTree from the public git hosting service `https://bitbucket.org/`

[2] Another GUI is git gui which comes with git; type the command `git gui` in a terminal window. Or perhaps use `gitx`.

[3] For some unknown reason GitHub once suddenly stopped working for me: it would not start-up. The solution was to upgrade to MacOSX 10.6.8. I really like the software github—in any case the command line always works.

```
git config --global --list
```

2 Manage your project locally

With git there are *three* places for storage of your files:

- your working area, directory/folder, where you work, edit and refine files in your project as usual;
- a local store of the history of the project and all its stages that is managed by git (hidden in `.git/`); and
- possibly an online repository, also managed by git, where you and collaborators merge independent progress on the project.

This second section only addresses the first two aspects of you working alone, with git, in your local working and storage area. Section 3 discusses collaboration over the internet.

Git also use the term "branch". Git branches appear very powerful. But for simplicity I ignore branches herein, and recommend you do not do anything 'branch' related until you are experienced with git.

2.1 Start your project

Make your working directory for the project as usual, say called `myproject`. Then execute the following commands in a terminal window.

`cd myproject` First change to the directory in which you are going to work.

`git init` Tell git to start managing the history of selected files in this directory.[4]

`git add .` Tell git to manage all the current files in the directory; or instead of the dot, specify a list of specific files.

`git commit` Stores the information that this (first) version is an identifiable stage, a 'milepost', in the project. git will request you type a message in the editor: make the first line a one line overall summary, such as "Initial version", and optionally provide additional information in subsequent lines.

Equivalent GitHub GUI

- Initialise an existing directory for git by dragging the folder icon onto the GitHub application or its window. Click `Yes`.
- In `Repositories` view, double click on the entry for `myproject`.
- Click on `Settings` in the left-hand tabs: then type lines `.DS_Store` into the `Ignored files` window and click `Save Changes` on bottom-right.[4]
- To add files to be managed, click on the `Changes` tab on the left: by default GitHub adds all files to management, change if you wish; files managed but unchanged are not shown; the contents of shown files are displayed in the right-hand pane.
- To commit: enter a message such as "Initial version" in the single line in the top-left, and enter more detail if you wish just below; and finally click `Commit Changes` button near top-left.

2.2 Work in small stages

It is best to work in small stages: remember, if you cannot summarise the work of the last stage in a one line sentence, then you have gone too long without committing. Typically work according to the following cycle, repeat as much as you like but ensure you end with the `add` and `commit`.

Working Work, edit and refine files in your project as usual.

`git status` Optional, checks which files you have changed.

`git diff` Optional, check what the actual changes were.

Equivalent GitHub GUI

- Work, edit and refine files in your project as usual.
- Start GitHub and double-click the project directory from the list in the `Repositories` window.
- Managed files you have edited will be listed; unmanaged files are also listed; managed files that have not changed are not listed.
- Optional, click on the file name in the left-pane to see,

[4]git creates an invisible sub-directory called `.git` in which is stored the history of the stages in your project. Do not meddle with this sub-directory.

[4]You may also want to ignore other files including `*.aux`, `*.log`, `*.out`, `*.synctex.gz`, `*.blg`, `*.toc`, `*.trc`, `*.xref`, `*.stc*`, `*.mtc*`, `*.maf`, and `*~`. The asterisk matches all files with that extension.

`git add file1 file2` Essential, nominates the files (and perhaps new files and new directories) whose updates are to be saved as the new version at this stage.

`git commit` Essential, commits the current version of your nominated files as an identifiable stage in your project; enter and save your commit message (of at least a one line summary).

in the right pane, the `diff`erences between the current version and the last commit.

- By default, all new files (even in a new directory) and all changed files will get committed; change if you wish.
- To commit: enter a message such as "Initial version" in the single line in the top-left, and more detail if you wish just below; and finally click `Commit Changes` button near top-left.

When you have two or more Git projects, change between them in GitHub by choosing `Repositories` from either the `View` menu or top-left of the bar, and then double clicking on the one for action.

2.3 Review your work

`git log` To overview history so far of the project.

Github: just click on the `History` tab on the top-left.

`git log --pretty=oneline` Lists the one line summaries.

`git commit --amend` Changes the message of the last commit.

`git reset --hard` If you have not committed, but realised that you have messed up your local files, then use this to recover the files as at the last commit.

Github: In GitHub's history view, click `Rollback to this commit` button.

`git checkout filename` Just recovers the named file as at the last commit.

`git mv oldname newname` Version control requires a little discipline. One is that files under version control must only be renamed/moved via git using this command.

Github: allows one to be undisciplined in moving and deleting files.

`git rm filename` Similarly delete/remove files under version control only via git using this command.

`git reset --hard HEAD^^` If you have mistakenly pushed/synced a couple of commits to a repository and need to revert to an earlier version on the repository, then do this command—for the specific case of removing the last two commits, use more or less carets `^` for other cases. Then to permanently remove from the repository you also have to execute `git push --force` (and presumably get everyone to also reset their versions).

2.4 Example: a memory stick coordinates work on two computers

Suppose you have a home computer and a work computer, and that you want to work on a project on both computers. One solution is to use a memory stick to transfer information, and git to manage the merging of developments and changes.

Suppose you have the current version of the project in directory/folder `myproject` on one computer. Do the following setup via terminal windows.

Memory stick

`cd /Volumes/YourStick` to go to the memory stick in Mac OSX, or do similar on other systems.

`mkdir myproject.git`

`cd myproject.git` to go inside the directory.

`git init --bare` to initialise it as an empty git repository.

Computer with current version

`cd myproject` or whatever you need to get to the directory with your project.

`git init` if not already under git management

```
git add *
```

```
git commit -m "initialise"
```

```
git remote add origin /Volumes/YourStick/myproject.git
```

```
git push -u origin master
```

The memory stick now has a git copy of your project, the current version, and you have linked the local copy to the one on the memory stick.

The other computer Move the existing version of your work somewhere else as a precaution.

`cd Documents` or to wherever you want the `myproject` directory to reside.

```
git clone /Volumes/YourStick/myproject.git
```

This creates a new local copy of the current version of the project information on the memory stick, in a directory called `myproject`, and links the local copy to that on the stick.

Work as normal After this initialissation, work and commit as normal as described in previous subsections. The extra ingredient is to `git push`, `git pull`, or GitHub `Sync` from your computers to the memory stick as appropriate in order to synchronise information on each of your computers and the memory stick.

All except the fourth git command here could be done with GitHub. The fourth has to be done with a terminal window.

2.5 Example: compare changes to earlier versions

Download `git-latexdiff` from `https://gitlab.com/git-latexdiff` and execute `make install` (or otherwise place a copy of `git-latexdiff` into `/usr/bin` with all the other git executables).

Then in a terminal window in any git repository you can execute something like

```
git latexdiff --main file.tex HEAD~1
```

to generate a difference file of the source that has been typeset into pdf with all the differences marked: old material in red; new material in blue. (Although it does fail for some.)

Use `HEAD~n` to compare changes since the `n`th previous commit.

3 Collaborate over the internet

To share and collaborate we need an information store on the internet. Although git is completely egalitarian in that no repository has any distinguished status, nonetheless, most collaborative projects invoke one repository to be the 'main' repository. I describe two alternatives: one is via an open service on the web by github; and the other is to use our Maths web server.

Create a account with GitHub.com The company GitHub provides some free storage (as well as a commercial service). Go to `https://github.com/plans` and click on `Create a free account`. Follow the instructions which involves: registering; generating an ssh key to give to GitHub; saving the API-token from GitHub into your git preference; and setting git config. GitHub's instructions lead you through the process.

The advantage of GitHub is that you can collaborate with anybody, and the security of their storage. The disadvantage

Ensure ssh access to server Ensure you have an account on our server[5]. Then configure, using public key cryptography, so that you can `ssh` to the server without entering your password. Section describes one way to setup ssh to do so.

3.1 Clone a collaborative existing project

Using our Maths server Assume that a collaborator, say `username`, has established a repository, say `ourproject.git` (they must have configured it for sharing).

- Perhaps the simplest is to issue the command

  ```
  git clone a1234567@www.maths.adelaide.edu.au:\
  /home/username/ourproject.git
  ```

  where `a1234567` is your username on the server.

  - Within your current local directory/folder, this command creates a new folder called `ourproject` (the `.git` gets dropped) which is a ‘copy’ of the repository.
  - Further, in the local git configuration it stores the location information about the remote repository. The remote repository then is the default.

- Alternatively, manage via a terminal window.
  - `git pull` gets any updates your collaborators have uploaded to the repository—conflicts will need to be resolved.
  - Work and commit as in Sections 2.2–2.3.

of GitHub is that the world can read your project (in minute detail).

Using GitHub.com server Assume a collaborator, `username`, established a repository, say `ourproject`: they must give you collaboration rights by web browsing to the repository, clicking `Admin`, clicking `Collaborators` on the left menu, then entering your GitHub name and `Adding`.

- Web browse to GitHub `https://github.com`, and login (top-right).
- Find the project (somehow): for example, just go to `https://github.com/username/ourproject`
- On the left-side beneath “username/ourproject” and beneath “Code”, you should see and click on `Clone in Mac`.
- Click `OK` and then choose a location on your computer for the repository folder to be created and material copied.
- On your computer, drag that new folder onto GitHub for GitHub to manage the local repository.
- Thereafter, work and commit as in Sections 2.2–2.3, but additionally occasionally *Click the `Sync` button on the `Changes` window.*
- Checking and downloading, ‘pulling’, any changes by your collaborators is one additional step you should do before starting work each session: from menu `Repository` select `Pull`; alternatively, from the very right of the top menu bar, click on `Branch in Sync`.

- To use the GitHub application to manage the local and remote storage, just drag your clone of `ourproject` onto GitHub. Thereafter, work and commit as in Sections 2.2–2.3, but additionally occasionally *Click the `Sync` button on the `Changes` window.* Before starting work each session get any changes by your collaborators: from menu `Repository` select `Pull`; alternatively, from the very right of the top menu bar, click on `Branch in Sync`.

[5]Currently `www.maths.adelaide.edu.au`

  - `git push` puts your commits onto the remote repository for others to get.

- In any event, one can inspect what is in the maths server repository by the following.[6]
  - `ssh -X www.maths.adelaide.edu.au` to login to the server with X-windows enabled.
  - Navigate to the git directory.
  - `git instaweb --httpd=webrick` will start up a browser (firefox) view of the contents and history of the repository (and any others that it finds).

3.2 Using ssh without passwords is best

The aim is to be empower ssh access to our server without having to enter in your password every time. This means GitHub, for example, will be able to invoke git protocols without the password.

On your computer Execute the following in a terminal window.

- Make an ssh directory. First check: type `cd ~`, then `ls -la|grep ssh` and if it lists a directory `.ssh` then skip the rest of this step. Execute `mkdir .ssh`
- Ensure suitable permissions. Again execute `ls -la|grep ssh` and it should only be `rwx` for you alone. If others have any permissions, then get rid of them by `chmod go-rwx .ssh`
- Check for keys. Execute `cd .ssh` and then `ls`: if you see two files `id_rsa` and `id_rsa.pub`, then skip the next step.
- Create keys. Execute `ssh-keygen -t rsa -C "email"` and just type Return/Enter when it asks for a file and for passphrases (although it is more secure with a passphrase). Two files `id_rsa` and `id_rsa.pub` should appear.

  Although `id_rsa.pub` is intended for others, *you must keep the contents of* `id_rsa` *secret to yourself only.*

On the server Open another terminal window and ssh to the server (entering your password for the last time!).

- Make an ssh directory. First check: type `cd ~`, then `ls -la|grep ssh` and if it lists a directory `.ssh` then skip the rest of this step. Execute `mkdir .ssh`
- Copy your key. Execute `cd .ssh`. Then copy the contents of the file `id_rsa.pub` into the file `authorized_keys` with perhaps `cat >> authorized_keys`.

  Use `ls -l` to check `authorized_keys` has permissions `rw-r--r--`

On your computer Execute ssh to the server and you should be able to do it without a password. If so, success.

[6]Currently `git` is installed in my home directory so you will have to execute the command `ln -s /home/a1184615/bin` in your home directory, logout, and then login again with ssh.

3.3 Create a new external repository

Given a local git repository, you want to *also* place the repository somewhere on the internet, perhaps for backup, but mainly for collaboration.

Establish a repository on the Maths server One creates an empty repository, and then fills it with information.[7]

- Login to the maths server.
- `mkdir ourproject.git`
- `cd ourproject.git`
- `git init --bare --shared` for access and modification only by those in your unix Group. Alternatively, `git init --bare --shared=0666` for access and modification by all who can login to the server.
- Then check to remove group and other write permissions from your home directory `/home/a1234567` (so that ssh will permit public key access).

Back on your own computer:

- clone the empty repository as in Section 3.1;
- then into the folder that is created, drag the content you want to be managed by git, then add, commit, push/pull as needed.

Using the free service of GitHub.com Ensure you have, and with a web browser login to, a GitHub account as described above.

- Click on either `Create a repository` or `New repository` in the web interface to `github.com`.
- Follow the route flagged `Existing Git Repository` and connect the web storage to your existing local git folder.
- Thereafter work as described previously.

Already locally exists However, if the content is already in a folder locally managed by git, then do the following. From within the local git folder,

```
git remote add a1234567@www.maths.adelaide.\
edu.au:/home/username/ourproject.git
```

Then push and pull as needed.

Github GUI Go to the repository managing in GitHub, click on `Settings`, enter into the `Primary remote repository` box the remote address

```
a1234567@www.maths.adelaide.edu.au:\
/home/username/ourproject.git
```

Then push/pull/sync.

3.4 Resolving conflicts with others

Mostly git will successfully merge edits by multiple people on the one file. However, if the edits are in the same region, then git is likely to flag the edits as conflicting and require you to resolve the conflict.

As yet the details are unclear.

[7]Currently `git` is installed in my home directory so you will have to execute the command `ln -s /home/a1184615/bin` in your home directory, logout, and then login again with ssh.