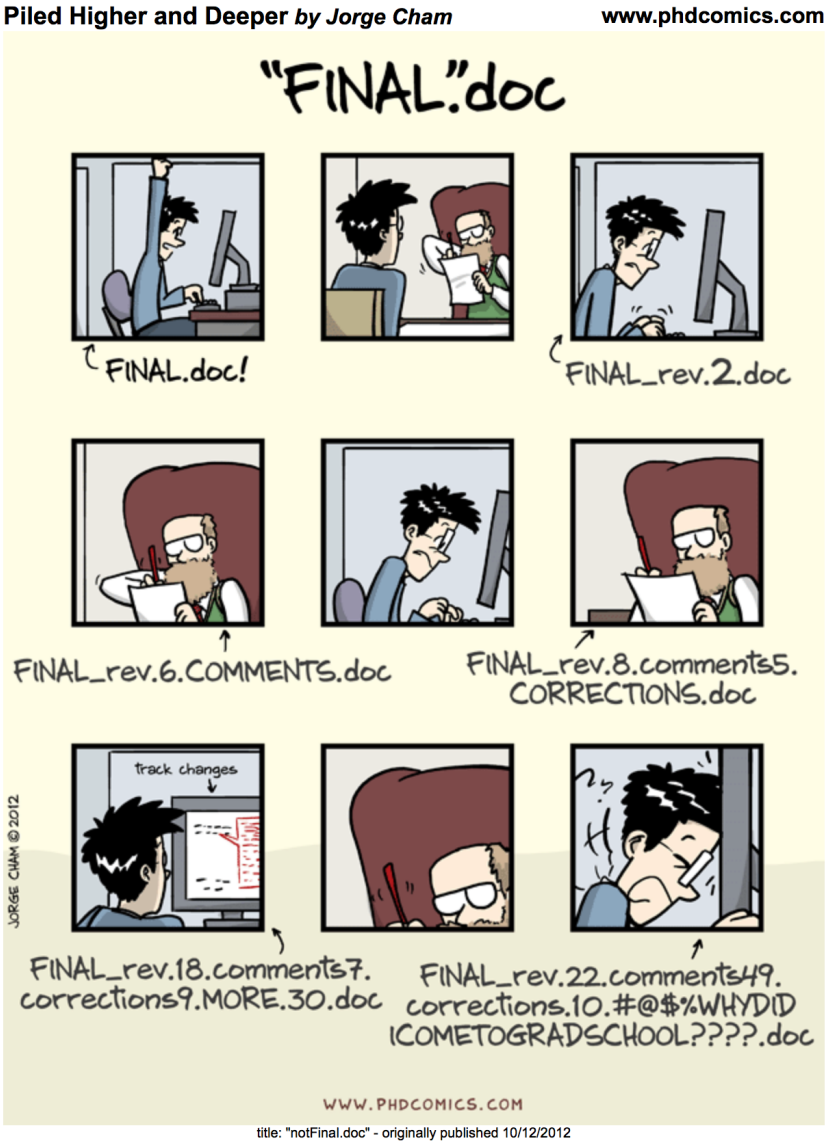
NCEAS
National Center for Ecological Analysis and Synthesis

Code versioning - Git Basics

Why do I need that again?



title: "notFinal.doc" - originally published 10/12/2012

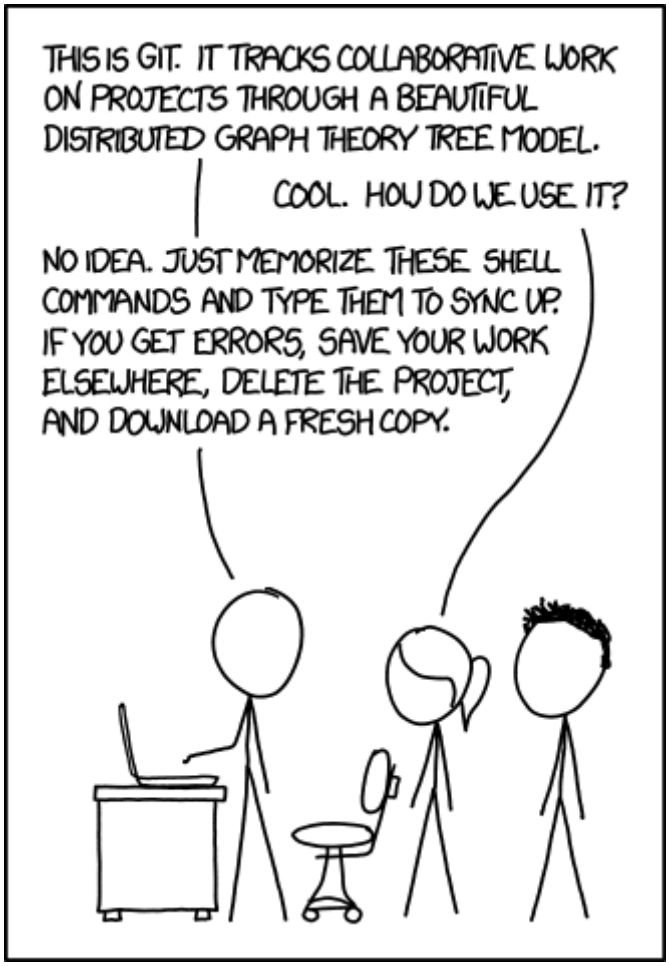
So code versioning is very useful to **keep track of changes you made to your scripts**. It allows you to choose when you have reached a stage in your code that you think is worth keeping track of, like a new function that makes your data analysis soooooo much better.

For scientists, code versioning is a useful tool to help you to track changes you make to your scripts and enable you to share your codes with your collaborators. For example, if you break your code, git can help

you to revert to an earlier working version. Want one of your collaborators to add a feature to your code to do a specific analysis? Code versioning can help you to do so in a smooth and organized manner.

Tthis training material focuses on the code verioning system calld `Git`. Note that there are others such as `Mercurial` or `svn` for example.

What is git?



Git is a *free* and *open source* distributed *version control system*. It has many functionalities and was originally geared towards software development and production environment. In fact, Git was initially designed and developed in 2005 by Linux kernel developers (including Linus Torvalds) for Linux kernel development. Here is a fun video of Linus Torvalds touting git to Google. Git can be enabled on a specific folder on your filesystem to version files (including sub-directories) within that directory. In git (and other version control systems) terms, this “tracked folder” is called a repository (which formally is a specific data structure storing versioning information).

More details:

- Git stores snapshot of your files (that have changes)
- Git is distributed, meaning that all the copies of your repository are of equal value, containing all your codes and its history. There is no central repository

- Git has integrity, meaning each file is checked (summed) to be sure there was no bit loss during any file manipulation by git. Each snapshot (also called commit) has a unique identifier.

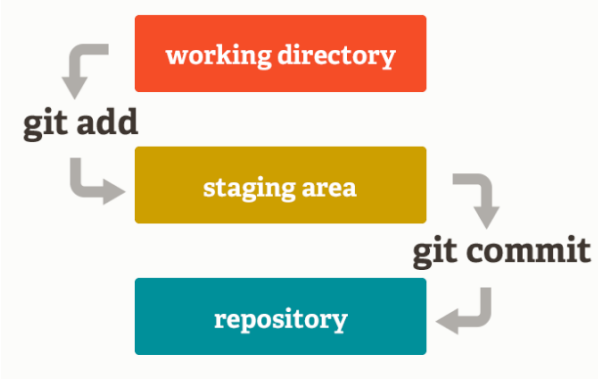
What git is not

- Git is not a backup per se
- Git is not GitHub (or more exactly: GitHub is not Git)
- it is not good at storing large data (by default)

The Git workflow

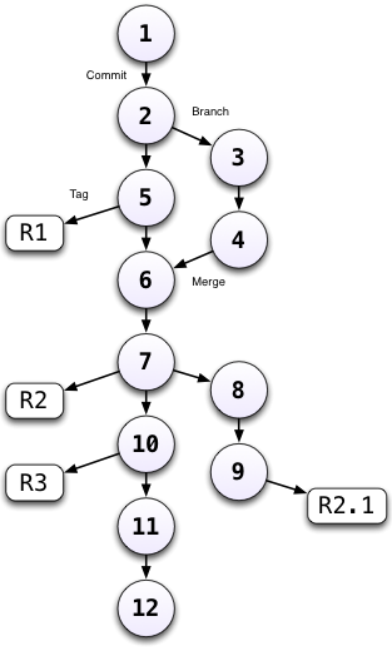
Overview

1. You modify files in your working directory and save them as usual
2. You **add** snapshots of your changes files to your staging area.
3. You do a **commit**, which takes the files as they are in the staging area and stores them as snapshots permanently to your Git directory.



And repeat!!

Everytime you create a new snapshot, you add the new version of the file to the database, but all the previous versions are kept in the database. It creates what is like a graph that you can navigate:



Getting started with Git

TRY GIT in 15min!

Before starting using git on any computer, you will have to set your identity as every snapshot of files is associated with the user who implemented the modifications to the files.

1. Setting up your identity

Setup your profile:

Your name and email:

```
git config --global user.name “yourName”
git config --global user.email “yourEmail”
```

Optional:

Check that everything is correct:

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

Modify everything at the same time:

```
git config --global --edit
```

Set your text editor:

```
git config --system core.editor vim
```

Want to know more? here

2. Starting a git repository

A **git repository** is a folder on your machine in which content is monitored for any changes by git.

`git init` is the command to start the tracking in a specific directory and transform it into a git repository:

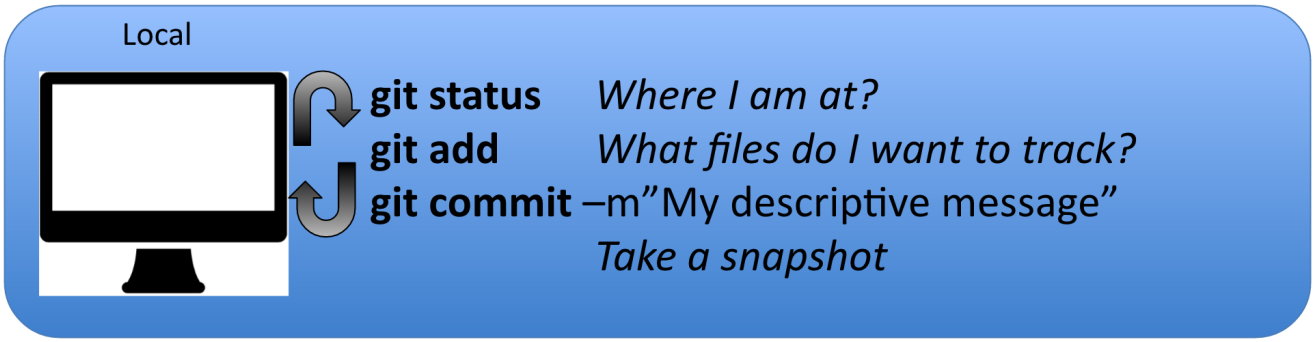
```
cd
mkdir git_nceas
cd git_nceas
mkdir snapp_workshop
git init
```

`git clone` to copy an existing repository to your machine, more precisely adding the repository in the directory you are in.

```
cd
mkdir git_nceas
cd git_nceas
git clone https://github.nceas.ucsb.edu/Training/postdoc-training.git
```

Tracking your changes

Let us have a closer look at the git workflow. It is important that a good portion of the workflow happens on your local machine (in blue), whereas a part requires interactions with a remote machine (in purple):



Example

Navigate in the `snapp_workshop` repository you just created.

1.) Let us create a csv file containing our name and favorite dessert:

```
vim favorite_dessert.csv
```

2.) Edit the new file adding headers and your info:

Note: hit the key `i` to switch to *insert mode* in vim. My file would look like this

```
My name, My desert
Julien, Ice cream
```

Exit the insert mode `esc`

Exit vim and save `:wq`

3.) Add the new file to git:

```
git status
git add favorite_dessert.csv
git status
git commit -m "Julien's favorite desert"
git status
```

4.) Add a friend to the csv:

```
vim favorite_dessert.csv

My name, My desert
Julien, Ice cream
Eliott, Crepes
```

Save and exit vim

5.) Add and commit the new version of the file

```
git status
git add favorite_dessert.csv
git status
git commit -m "Adding Eliott's favorite desert"
git status
```

6.) Check the differences between the two last commits:

```
git diff HEAD~1
```

Note: hit `q` to exit

7.) We can also look at the log of commits to look a the commit sequence

```
git log

git log -1
```

Getting information

- `git status` this command is your friend! It will tell you where you are at and what are your options. You can use at any point in your process.
- `git log` displays history of committed snapshots. It lets you list the project history, filter it, and search for specific changes.
- `git diff --cached` To be used before committing to preview the changes to be committed.
- `git diff HEAD~1 my_script.R` to inspect the changes between the last commit (HEAD) and the previous one

- `git diff HEAD~2 my_script.py` to inspect the changes between the last commit (HEAD) and 2 commits before.

Git has a lot of terms and commands, see reference at the end of this document for an extensive terminology.

Ignoring certain types of file

`.gitignore` is a specific file used to list what (type of) files you do not want git to track. This file need to be placed at the top level of the directory.

File content example from GitHub: https://gist.github.com/octocat/9257657

To create this file from the terminal/shell:

```
vim .gitignore
```

To know more: https://git-scm.com/docs/gitignore

Undoing things

Unstage a file

`git reset HEAD` lets you remove a file from the staging area

```
git reset HEAD <my_file_I_added_by_mistake.xls>
```

This will remove the file from your next commit. Can be used to undo an erronous `git add`.

Undo your last commit

`git commit --amend` let you change your last commit, like for example if you forgot a file

```
git add <missing_script.R>
git commit --amend -m "My new message"
```

More info about how to undo things here

Few terms you might have heared about git

- **HEAD**: it is the reference that points to the latest commit
- **Branches**: A branch represents an independent line of development, parallel to the master. In fact, the default branch name that is create by `git init` is ***master***.

References

- Interactive git 101: https://try.github.io/
- Very good tutorial about git: https://www.atlassian.com/git/tutorials/what-is-version-control
- Git tutorial geared towards scientits: http://nyuccl.org/pages/gittutorial/
- Short intro to git basics: https://github.com/mbjones/gitbasics
- Git documentation about the basics: http://gitref.org/basic/
- Git documentation - the basics: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics
- Git terminology: https://www.atlassian.com/git/glossary/terminology
- NCEAS wiki page on git: https://help.nceas.ucsb.edu/git?s[]=git
- In trouble, guide to know what to do: http://justinhileman.info/article/git-pretty/git-pretty.png
- Want to undo something? https://github.com/blog/2019-how-to-undo-almost-anything-with-git

License