Version Control with Git

Martin Robinson

2019

Why Use Version Control?

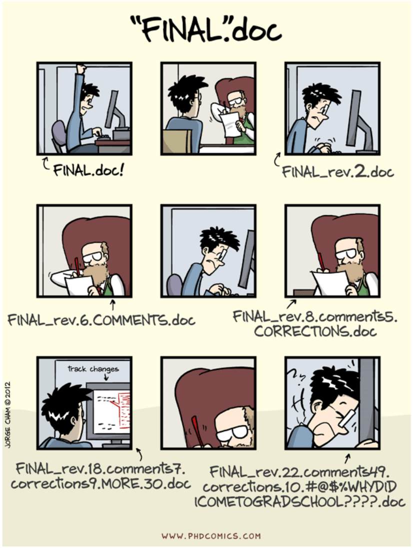

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Figure 1: "Piled Higher and Deeper" by Jorge Cham, http://www.phdcomics.com

Why Use Version Control?

Version control is better than mailing files back and forth:

- Nothing that is committed to version control is ever lost.
- As we have this record of who made what changes when, we know who to ask if we have questions later on, and, if needed it, revert to a previous version
- the version control system automatically notifies users whenever there's a conflict between one person's work and another's.

Teams are not the only ones to benefit from version control, Version control is the lab notebook of the digital world.

Not just for software: books, papers, small data sets, . . .

What I use Git for

- code (personal and collaborations)
- papers (latex or markdown)
- presentations (markdown -> beamer)
- grant applications (larger ones)
- teaching (course notes, exercises, admin etc.)

Everything on GitHub (https://github.com/martinjrobins) in public and private repositories

Git

- Developed in 2005 by the Linux development community for the Linux kernel project
- Features:
  - Branching and merging
  - Fast
  - Distributed
  - Flexible staging area
  - Free and open source
- http://git-scm.com/
- http://git-scm.com/book/en/Getting-Started-Installing-Git

What is a Repository?

- A git repository is a collection of *commits* arranged in a sequential or branching network

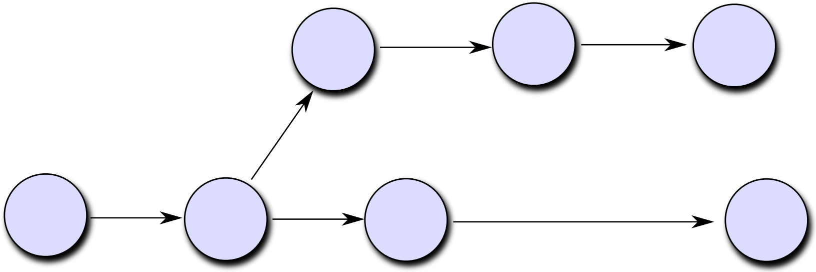


Figure 2: Series of commits

What is a Repository?

- Each commit contains snapshots of the files that are added, along with timing, author etc. information

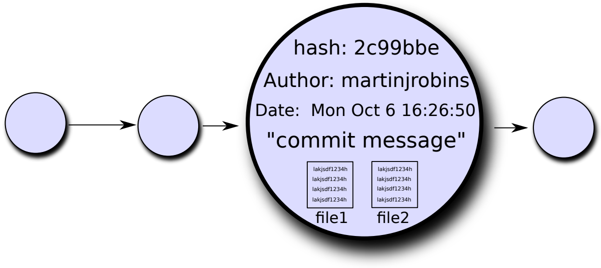
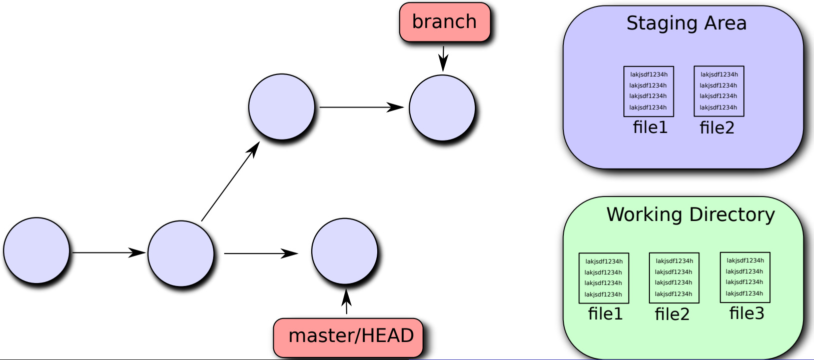


Figure 3: A commit

What is a Repository?

- As well as the commits, branch pointers show the progress of different branches
- The *working directory* is simply the current set of files in the user's local directory
- The *staging area* is where new edits are added in preparation for creating a new commit



Creating a Repository

Normally you would work with a remote repository (e.g. one on https://github.com or https://bitbucket.org). You can import this into your current directory using the `clone` command

```
$ git clone <repo>
```

for example,

```
$ git clone https://github.com/martinjrobins/exercise.git
```

This will download the repository and create a copy on your computer. This is a *separate* local git repository that is linked to the remote

Adding Content

- Use ‘git add’ to add a change in the *working directory* to the *staging area*. All changes or new files need to be added to the staging area before they can be committed (or use the ‘-a’ commit option, see below)

```
$ git add <file>
$ git add <directory>
```

- Commit all changes in the staging area to the project history

```
$ git commit
$ git commit -m "your commit message"
```

- Commit all changes in the working directory

```
$ git commit -a
```

Sending Commits to Remote

- When you want to send your new commits to the remote, use the *push* command.

  ```
  $ git push
  ```

- If the branch you are pushing doesn't yet exist on the remote, you can use this command to push a new branch

```
$ git push -u <repo> <branch>
```

- This command pushes the current branch to the remote *repo/branch* branch.
- The *-u* option sets up the current branch to track the *repo/branch* branch.

Examining the history

- View the repository commit history using the *git log* command

```
  $ git log
commit 0d5ef743e3c0e58ea92154016ed301c80ab03428
Author: martinjrobins <martinjrobins@gmail.com>
Date:   Mon Oct 6 16:47:06 2014 +0100

    this is the third commit

commit a0687f67bc59aadde572ca1395bae2dc1ea462b2
Author: martinjrobins <martinjrobins@gmail.com>
Date:   Mon Oct 6 16:46:48 2014 +0100

    this is the second commit

commit c7245bbb67c23eec849e9bb2097b45e9c4986149
Author: martinjrobins <martinjrobins@gmail.com>
Date:   Mon Oct 6 16:46:33 2014 +0100
```

Examining the history

- For a brief summary use *–oneline*

```
$ git log --oneline
```

- You can go back to any of your commits in the past using the `checkout` command

  ```
  $ git checkout <commit>
  ```

- Make sure to go back to the branch you are working on!

e.g.

```
$ git checkout master
```

Examining the staging area

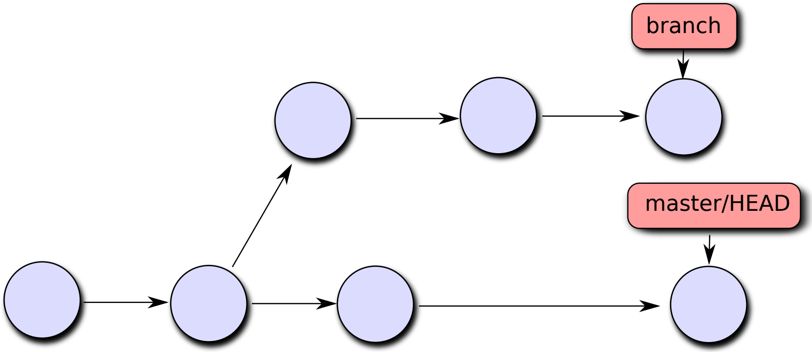
- Use the *git status* command to get details of the staging area and working directory

```
$ git status

# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be commit
#   (use "git checkout -- <file>..." to discard changes in
#
# modified:   file1.m
#
no changes added to commit (use "git add" and/or "git comm
```

Branching

- Branching can be used to:
  - create/try out a new feature
  - provide conflict-free parallel editing for multiple team members
    - separate editing into "stable" and "in development" branches
- A branch is simply a pointer to a commit. The current branch is pointed to by *HEAD*



Branching

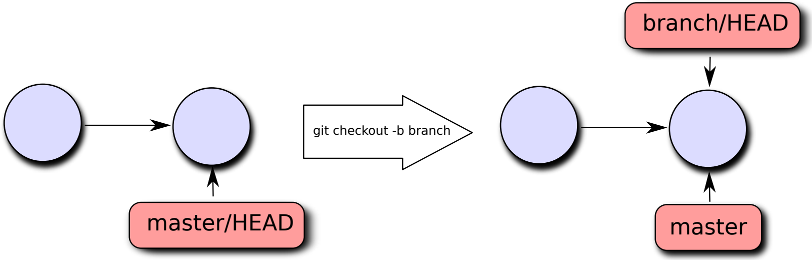


Figure 6: Creating a new branch

- You can create a new branch and switch to it using the *checkout* command

```
$ git checkout -b <branch>
```

Branching

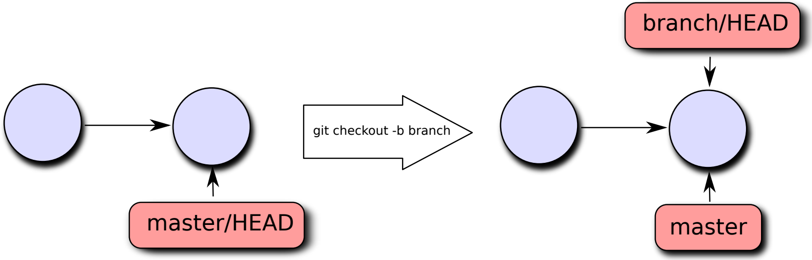


Figure 7: Creating a new branch

- This is shorthand for

```
$ git branch <branch>
$ git checkout <branch>
```

Branching

- Make a few new edits to your new branch

```
$ edit file1.m
$ commit -a -m "added wow new feature"
$ edit file1.m
$ commit -a -m "fixed bugs in new feature"
```

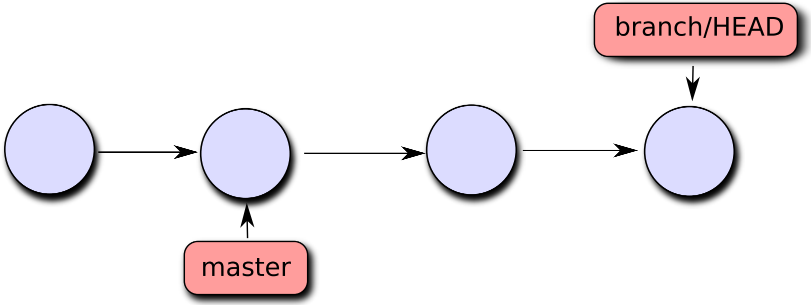
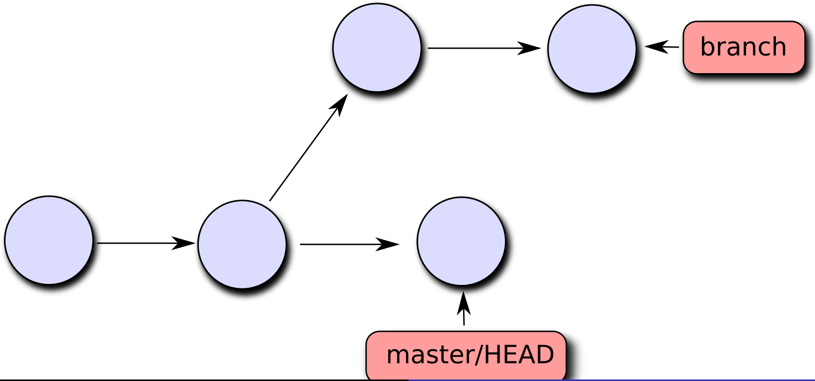


Figure 8: new edits to branch

Branching

- You might need to got back to the *master* branch to make some corrections

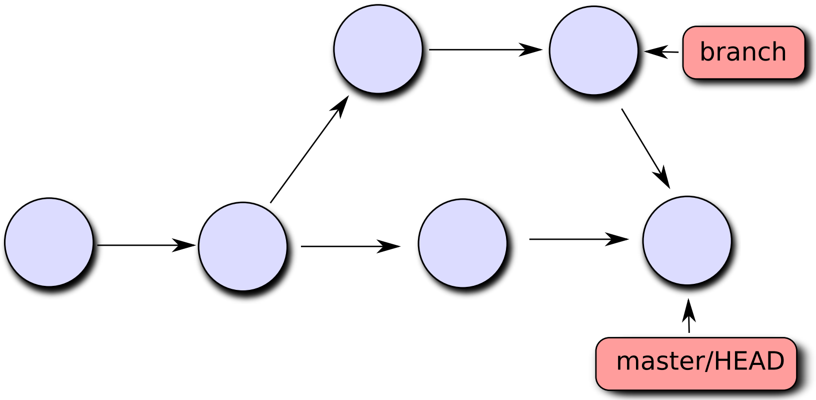
```
$ git checkout master
$ edit file1.m
$ commit -a -m "fixed a major bug"
```



Merging

- Now we have two separate branches, but what if we want to merge them together?

```
$ git checkout master
$ git merge branch
```



Merge Conflict

If there are conflicting edits to *file1.m* in *master* and *branch*, you might get an error message like so:

```
$ git merge branch
Auto-merging file1.m
CONFLICT (content): Merge conflict in file1.m
Automatic merge failed; fix conflicts and then commit the r
```

Resolving Merge Conflict

- If you open *file1.m*, you will see standard conflict-resolution markers like this:

```
<<<<<<< HEAD
This is the new line in master
=======
This is the new line in branch
>>>>>>> branch
```

- If you want to see which files are still unmerged at any point, you can use *git status* to see the current state of the merge
- Finally, commit the results of the merge

```
$ git commit -a -m "merged branch into master"
```

More info

- If there is any command you are unclear about, you can use *git -h* to get more information. Or simply google it. . .
- Further tutorial can be found online at:
  - Git documentation and book (http://git-scm.com/doc)
  - Atlassian tutorials (https://www.atlassian.com/git/tutorials)
  - Software Carpentry Foundation (http://software-carpentry.org/)
- Acknowledgements: material for this lecture modified from links above.
  - Git book: Creative Commons Attribution-NonCommercial-ShareAlike 3.0
  - Atlassian tutorials: Creative Commons Attribution 2.5 Australia License.
  - Software Carpentry: Creative Commons Attribution Licence (v4.0)

Exercise

Go to https://github.com/martinjrobins/exercise and follow instructions for a guided git exercise. . . .