Chaste Developers Tutorial - Version Control with Git

Martin Robinson

July 7, 2016

Why Use Version Control?

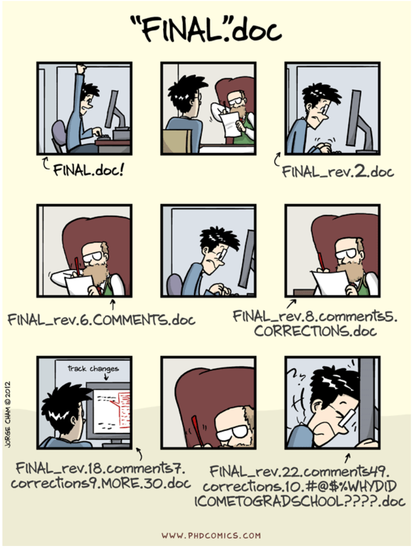

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

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, ...

Version Control

- Many different VC software packages
  - CVS (1986, 1990 in C)
  - Subversion (2000)
  - Mercurial (2005)
  - Git (2005)
  - ... many others
- What can you use it for?
  - Text files are best, can see differences between versions
  - Source code is the #1 use case
  - Can also use it for documents or presentations (e.g. latex, beamer, html, markdown)

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`

Installing Git

- Fedora Linux

```
$ yum install git
```

- Debian-based distribution (e.g. Ubuntu)

```
$ apt-get install git
```

- Graphical Mac Git installer at http://sourceforge.net/projects/git-osx-installer/
- MacPorts

```
$ sudo port install git +svn +doc +bash_completion +git
```

- Windows
  - *msysGit* at http://msysgit.github.io. Use provided msysGit shell for command line interface
  - *Cygwin* at http://www.cygwin.com/

Who are you?

Setup your git installation by telling it your name and email

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "example@maths.ox.ac.uk"
```

What is a Repository?

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

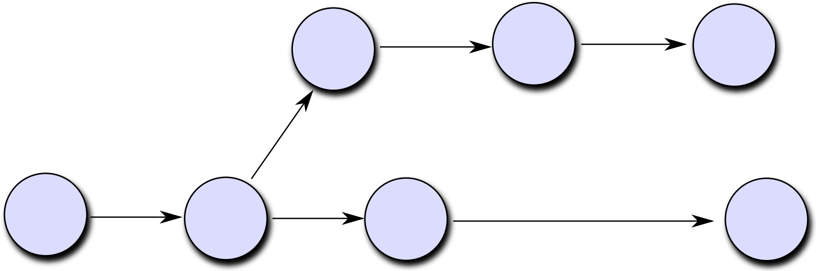


Figure 2: Series of commits

What is a Repository?

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

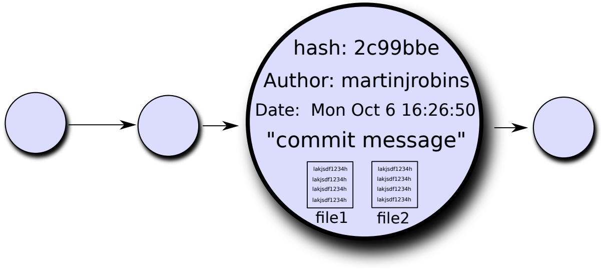
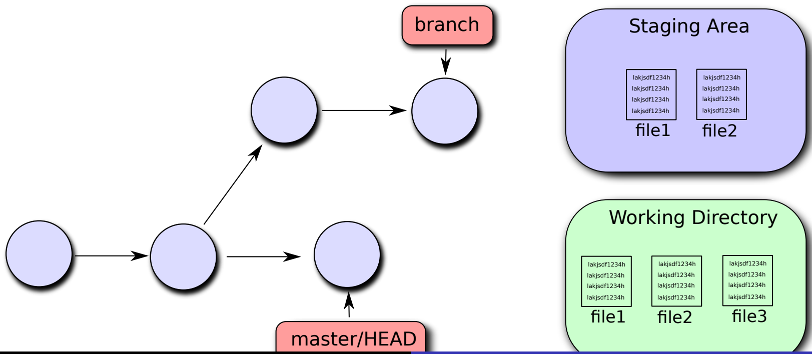


Figure 3: A commit

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

- Initiate an empty local repository. Any files already in the directory need to be added and committed before they included in the repository history.

  ```
  $ git init
  ```

- Clone a remote repository. This can be a directory for a local repository, or a URL for a remote.

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

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

Cloning a repository

for example

```
$ git clone https://server/username/my_git_repo.git
```

This uses the secure `http` protocol, might need to authenticate with a username and password.

```
$ git clone git@server:username/my_git_repo.git
```

This uses the `ssh` protocol, to use this you must have an ssh-key installed on the git server.

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 -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.
- Future pushes do not need extra arguments (even if you clone the repository again)

```
$ git push
```

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
```

- For detailed information of differences between commits use *-p* option

```
$ git log -p
```

- Use the *diff* command to see the differences between the working directory and the staging area.
  - The *–staged* option shows changes between the staging area and the last commit.
  - Use the *HEAD* pointer to see the changes between the working directory and the last commit

```
$ git diff <file>
$ git diff --staged
$ git diff HEAD
```

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
```

Undoing Changes

- Reset (not yet added to staging area) to the last committed version:

```
$ git checkout <file>
```

- You can amend the current commit (e.g. change commit message, commit new changes etc)

```
$ git commit --amend
```

- You can remove a file from the staging area (i.e. after using *git add*)

```
$git reset HEAD <file>
```

Altering your history - Reset versus Revert

- You can reset your history (soft reset) and optionally your working directory (hard reset) to a specified :

  ```
  $ git reset <commit>
  $ git reset --hard <commit>
  ```

- You can remove a specified commit from the history (a new commit is made with the necessary changes)

  ```
  $ git revert <commit>
  ```

Altering your history is best avoided unless you know what you are doing.

Example Workflows

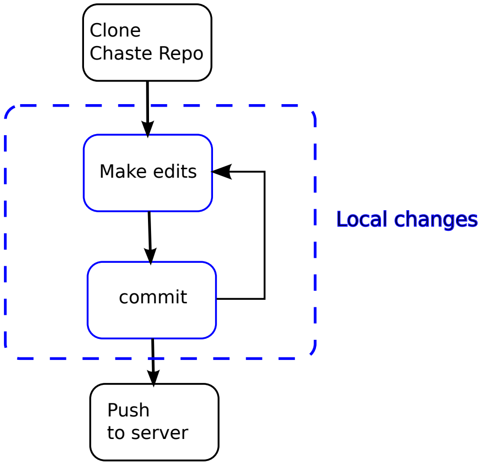


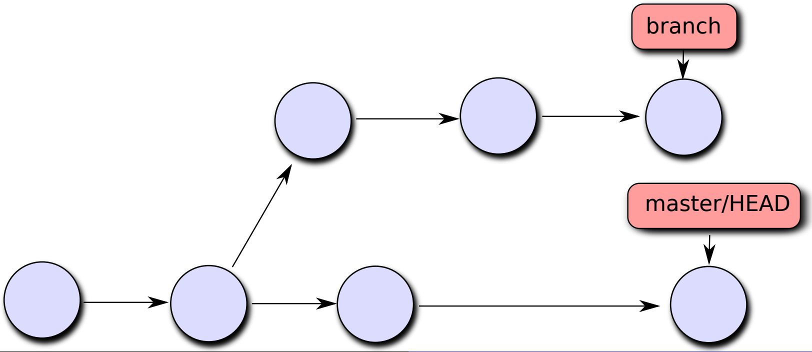
Figure 5: Minor edits

Exercise

Go to `https://github.com/martinjrobins/exercise` and follow the first exercise, “Exercise 1: Making Commits”

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

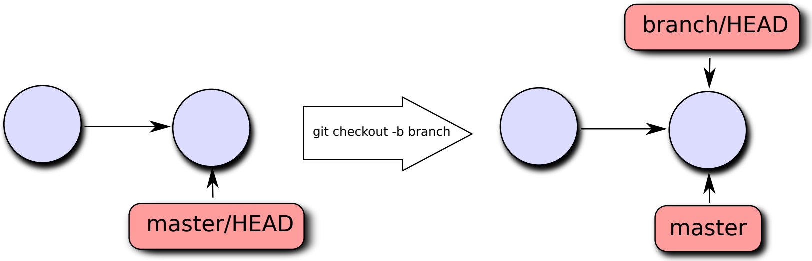


Figure 7: Creating a new branch

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

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

Branching

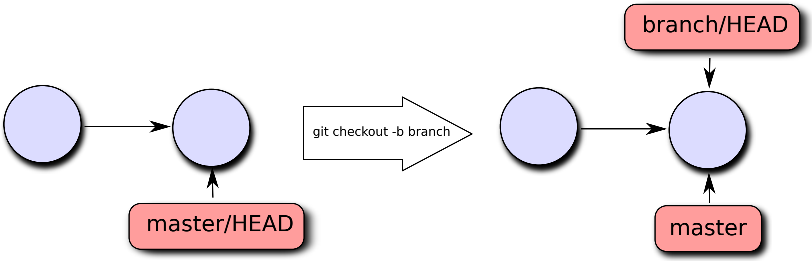


Figure 8: Creating a new branch

- 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"
```

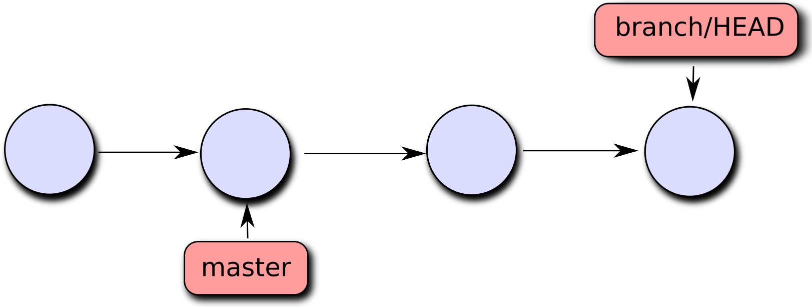
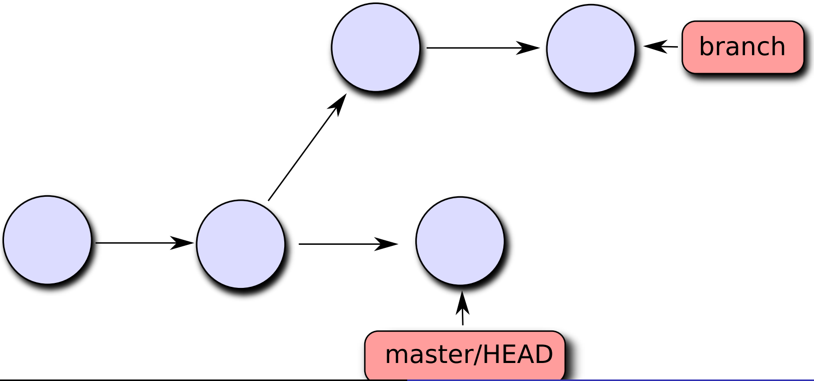


Figure 9: new edits to branch

Branching

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

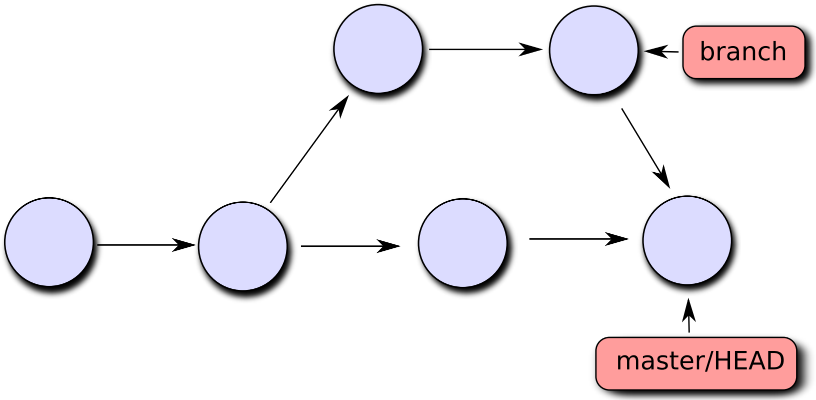
```
$ git checkout master
$ edit file1.m
$ commit -a -m "fixed a major bug"
```



Merging

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

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



Rebaseing

- You might not want to create a new commit (e.g. for many simple merges). Can also use rebasing

```
$ git rebase master
```

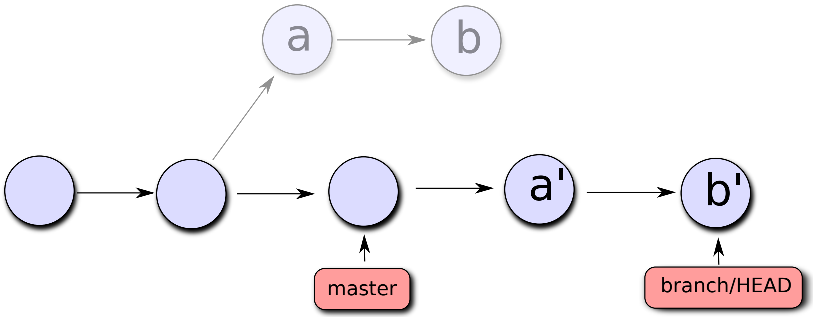


Figure 12: rebase branch onto master

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

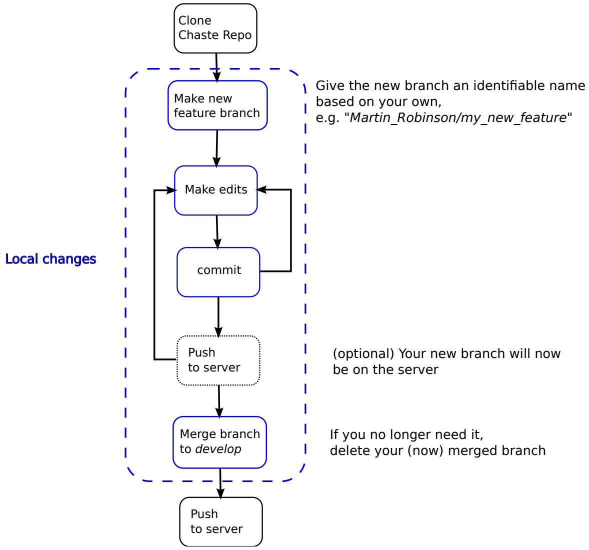
- If you open *file1.m*, you will see standard conflict-resolution markers like this:

```
<<<<<<< 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"
```

Example Workflow



Clone Chaste Repo
Make new feature branch
Make edits
commit
Push to server
Merge branch to *develop*
Push to server
Local changes
Give the new branch an identifiable name based on your own, e.g. "*Martin_Robinson/my_new_feature*"
(optional) Your new branch will now be on the server
If you no longer need it, delete your (now) merged branch

Figure 13: New features

Exercise

Go to `https://github.com/martinjrobins/exercise` and follow the second exercise, “Exercise 2: Branching and Merging”

Sharing and Collaborating

- If other people are making commits to the repository, you can pull these using

```
$ git pull
```

- This is short for `git fetch; git merge origin/master` (can also use rebasing)

Conflicts pushing to remote

- If you try to `push` new commits and they conflict with commits already in the remote repo, you will get an error. Need to then `merge` or `rebase` them:

```
$ git pull
```

or

```
$ git fetch
$ git rebase origin/master
```

Maintaining a feature branch

- If you want to avoid a busy `master` branch or develop a large feature, best to use a feature branch. e.g.

```
$ git checkout -b mrobinson/world_peace
```

- If you want to keep this up-to-date with the `master` branch you can do periodic `rebase`-ing

```
$ git pull
$ git rebase master
```

Collaborating on a feature branch

- You can push your branch to the remote for backup or to share with other developers

```
$ git checkout -b mrobinson/world_peace
$ git push origin mrobinson/world_peace
```

- Other developers can pull your branch using

```
$ git checkout --track origin/mrobinson/world_peace
```

Collaborating on a feature branch

If you and another developer(s) are busy on a branch, you might not want to be constantly merging your edits. Can setup the branch to always `fetch/rebase` instead of `fetch/merge` by setting the config parameter `branch.<name>.rebase` to true.

```
$ edit .ssh/config
```

Or you can do it once off using the `--rebase` option

```
$ git pull --rebase
```

Example Workflow

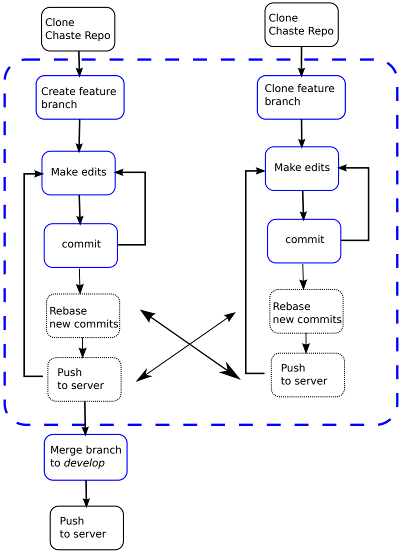


Figure 14: Branch Collaboration

Exercise

Go to `https://github.com/martinjrobins/exercise` and follow the third exercise, “Exercise 3: Collaboration”

Chaste Infrastructure

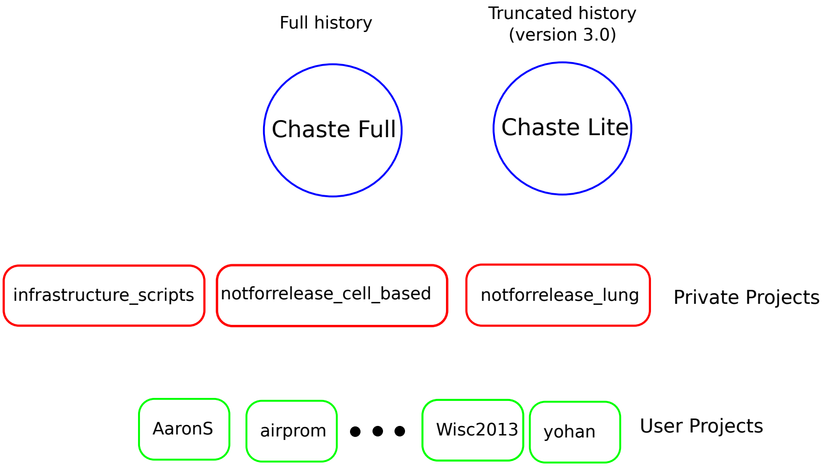


Figure 15: Chaste Git Repos

Chaste Git Repo

multiple branches

- **develop**: all development occurs here, starting point for CI testing
- **passed_continuous**: latest commit that passed `continuous` test pack
- **passed_nightly**: latest commit that passed `continuous` and `nightly` test packs
- **passed_lofty**: latest commit that passed `continuous`, `nightly` and `lofty` test pack
- **master**: latest commit that passed *all* tests
- **user_name/feature_name**: feature branches created as needed by developers (can be tested through buildbot web interface)

Practicalities

```
$ git clone -b develop https://chaste.cs.ox.ac.uk/git/chast
$ git clone https://robinsonm@chaste.cs.ox.ac.uk/git/chaste
```

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)