Advanced Programming (IT)

Dr. Simon Rogers (simon.rogers@glasgow.ac.uk, @sdrogers)

January 2018

Contents

Motivation

- It's important to keep track of changes to code
- Especially when working in a team
- *Version Control* systems allow us to do this
- Most popular: **Subversion** and **git**

Introduction to git

- *Warning* Git has a steep learning curve
  - ..but it's worth it
- This is based upon the excellent tutorials at

What does git do?

- Git keeps track of files in a directory and its subdirectories
- Keep track?
  - remembers all changes
  - allows you to rewind changes
  - allows you to see what other people have done
  - allows you to create independent *branches*

Try it!

- Navigate to a directory you wish to version control with git
- issue `git init`
- Folder is now a git repository!

Basic operation

- Change something (change file, add file, remove file)
- Add changed file to *Staging Area*
- *Commit* changes

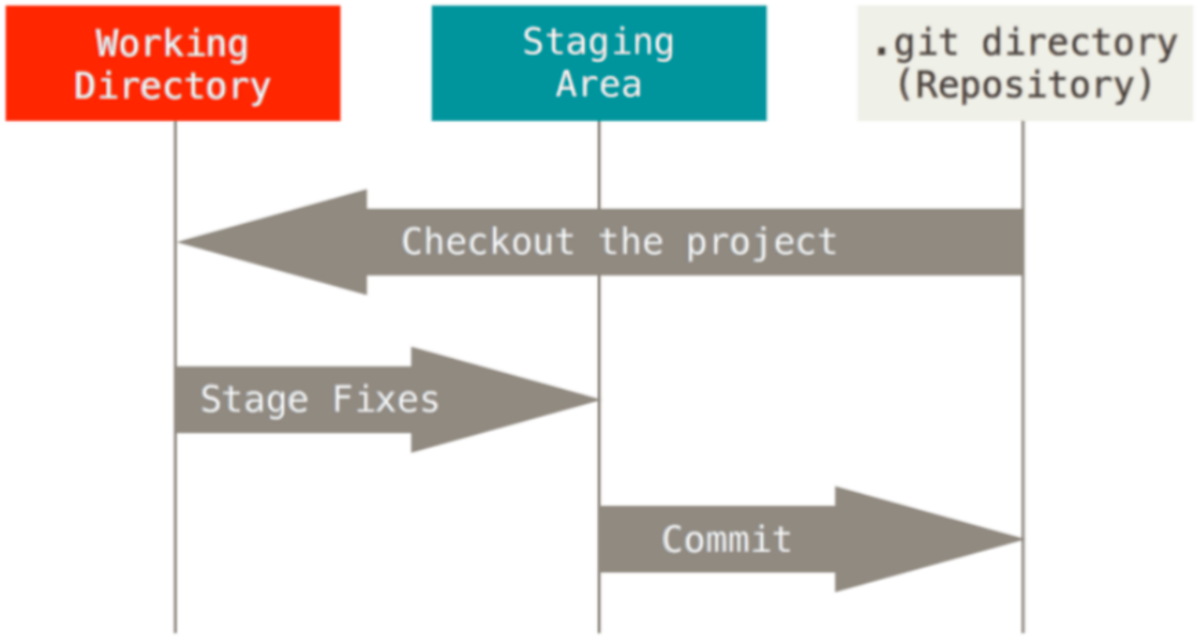

Figure 1: The three components of Git (figure from git-scm.com)

Adding files

- When a file is created, you *have* to tell git about it:
  - `git add myfile.java`
- To *Commit* the changes
  - `git commit -m "added some files"`
- If you get stuck, `git status` is your friend!

Standard Commit command

- Normally you will be commiting changes to existing files.
- They still need to be added to the Staging Area
- Shortcut, use `-a`
  - `git commit -a -m "changed some files"`

- This adds any changed files **that git is aware of** to the staging area and then commits

Jumping to a previous commit

- Each commit has a unique ID
- You can jump to it with:
    - `git checkout <ID>`
- Try this and be amazed at how the contents of the repository immediately change!

Branches

- Git's true power lies in *branches*
- Branches allow you to switch between different versions of your repository
- When you create a repo, you have one branch (called **Master**)
    - The name **Master** is *just a convention* – it's not special
- You can switch between brances at any time with:
    - `git checkout <branchname>`
- And create new branches with:
    - `git checkout -b <newbranchname>`
- You can also merge a branch (e.g. newfeature) into the current branch with:
    - `git merge newfeature`

Branches are one of the reasons for git's popularity. There's a great tutorial here. Note that in that example the commits go from left to right. Having a main branch and calling it **Master** is just a convention. No branch in git has any intrinsic priority over another. A standard use case for branching would be:

```
1. Identify an issue or new feature for your code
2. Create a new branch (from e.g. `Master`) (`git checkout -b mybranch`)
3. Work to solve the problem on that branch
4. Test etc
5. Commit your changes (`git commit -a -m "fixed the issue...yay"`)
6. When ready merge your new branch back into `master`
7. `git checkout master`
8. `git merge mybranch`
9. Repeat...
```

Git v github

- Common error: git and github are the same thing. Not true!
- **Git**: version control system
- **Github**: website that will host repositories for you
- All notes and code for this course are available on github: http://github.com/sdrogers/APIT
- Github also has useful features like issue tracking

Github etc just host a copy of your repository. There's nothing special about the repository they store, although by convention it's normally considered to be the *main* copy.

Synchronising with a remote repository

- Easiest way: clone an already existing repo.
    - e.g. `git clone https://github.com/sdrogers/APIT.git`
- Can also setup a `remote` for a pre-existing repo (see online docs)
- To merge a branch (`master`) from the remote (`origin`) to the current local branch:
    - `git pull origin master`

- This grabs the `master` branch from the server and merges it with whichever branch is checked out locally.
- To send your changes to the server:
  - `git push origin master`
- This sends the current local branch to the server and merges it into the `master` branch there

You can also get the changes from the server without merging into the current branch using `fetch` instead of `pull`.

Conflicts

- You **will** get merge conflicts (where the same file has changed in both branches)
- Don't panic!
- Do `git status` and follow the instructions

Git tips

- Play around with an unimportant repository
- Create something on github and play with other people (you'll quickly learn about conflicts)
- Applications like *SourceTree* are handy to look over commit histories (unless you have hardcore command line skills)
- Big binary files (e.g. .pdf, .class, .sqlite) are often problematic
- `git status`