SKILLPILLS

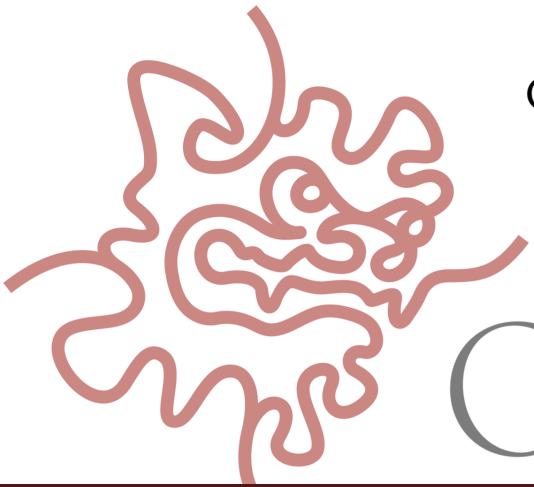
Skill Pill: Introduction to Git and Version Control

Lecture 2: Git it on!

Valentin Churavy - 2016
James Schloss - 2018
Christian Butcher - 2019

Okinawa Institute of Science and Technology
christian.butcher@oist.jp

August 14, 2019



OIST

Overview

SKILLPILLS

Recap

Yesterday we covered (don't forget to prefix with **git**):

- **clone** : Cloning a repository into a new directory
- **add** : Add file contents to the *index*. This makes git track the file.
- **commit** : Record changes. Store the staged files as a new part of the history!
- **pull** : Updating from a *remote*. Technicially a combination of **fetch** and **merge** by default.
- **push** : Update remote *refs* and objects.

remote : Another git repository. We used GitHub to provide this.
index : A single, large, binary file listing all files in the current branch with some extra information. Reflects the "proposed next commit".
refs : Short for references. Can point to almost anything in git.

Tips so far...

- You can use `git help <command>` or `git <command> --help` to get information about a command, like `clone`.
- `git add -p` uses *patch mode* to interactively add parts of a file. **-i** is interactive without patch mode.
- `git rm` can be used to remove files from the index (and optionally working directory), whilst `git mv` can help you move files within the repository.
- `git commit --amend` opens an editor to alter the previous commit's message. Don't do this if you already **push**ed the commit!

Reset and Checkout

We also considered **reset** and **revert**.

- Reset is a fairly complicated tool, which modifies the three 'trees' we have briefly mentioned/considered - `HEAD` (your last commit), `index` (the staging area) and the `working directory`.
- If you're interested to know more about this tool, there is a long and informative guide at `https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified`.
- This content is really beyond the scope of our Skill Pill. :(

Remotes

Yesterday we introduced **GitHub**. GitHub is a service that offers you a solution to remotely store your repositories.

- Git is *Distributed* Version Control System (DVCS). Every copy of your repository, may it be remote or local, is independent of each other. There is no central master repository.
- In order to synchronize these distributed copies we introduce the concept of a remote.

git **remote**

- There can be as many remotes as you want each with different names. When you clone a repository there will be one default remote called **origin**.

Exercise 1a

1. Tell Jeremie your GitHub user name so you can be added to the allowed committers for the repository...
2. Clone this repository from GitHub:
   `https://github.com/oist/skillpill-git-group1`
3. Add your favourite color to the colors file.
4. Commit your change with an appropriate message, but don't **push**!

1b: Alternative Exercise if I can't get access to the repository...

1. Create an empty directory for your repositories (note plural)
2. In this directory, `A`, create a new directory, called 'myremote' (or change as needed below).
3. Change into this new directory, and run `git init --bare`. This creates a 'bare' repository, in which nothing is checked out.
4. Move back into directory A. Create a new directory, and then run `git init` (not bare), add a file, and commit it.
5. Run the command `git remote add origin ../myremote/`, which adds a remote repository called 'origin', pointing to the directory 'myremote'.

1b: Alternative Exercise Continued...

6. Push your commit with the command `git push -u origin master`, which sets your branch to track the new branch 'master' on 'origin'.
7. Go back to A. Clone your 'myremote' repository with the command `git clone myremote secondcopy` (a new directory named 'secondcopy' will be created).
8. Make changes to your file in both your first repository, and 'secondcopy' (not 'myremote'). Commit both. Push from only the first repository (not 'secondcopy').

Merging - An Introduction

SKILLPILLS

- We perform *merges* to "join two or more development histories together".
- It is most commonly performed invisibly by `git pull` and performs by default a "fast-forward" merge.
- We usually see this first when we try to pull some changes and we cannot perform a fast-forward merge.
- In that case, we have to resolve the *merge conflict*.

Merging

SKILLPILLS

Merging is the act of joining two branches together or to join two different branches. You will always merge *from* a branch/remote into a branch.

- git **fetch** Gets remote changes
- git **merge** Merge changes (ff by default)
- git **add** Resolve merge-conflict

Options for merge:

–no-commit Performs the merge, but doesn't commit yet. Giving you the change to edit the merge commit.

–ff-only Aborts when we can't perform a fast-forward merge.

–abort Aborts current conflict-resolution and reset to previous state.

You can visualize your history in many different ways, but the best way on the command line is.

git log **–graph –decorate –oneline**

Exercise - Assuming we did Exercise 1a

1. Try to push your changes to the 'colors' file with **git push**
2. You'll receive a message indicating that there are remote changes, and instructing you to use **git pull**
3. Use **git pull** to trigger a merge conflict
4. Use a text editor to resolve the conflict
   - There will be lines like `<<<<<<< HEAD`, `=======` and `>>>>>>> 1a2bde2189`
   - Your changes are marked by the section between `HEAD` and the `=======` line
   - Update the file to reflect the new version you expect
5. Commit the resolved file (don't forget to **add**)
6. Try to push your branch again. If someone else already did, you'll be back at the beginning!
7. Obviously this can be annoying - we'll look at Pull Requests (a GitHub feature) later as a different way to handle this in bigger repositories

Exercise - Assuming we did Exercise 1b

1. Go into your 'secondcopy' directory and try to **push** the changes
2. You'll receive a message indicating that there are remote changes, and instructing you to use **git pull**
3. Use **git pull** to trigger a merge conflict
4. Use a text editor to resolve the conflict
   - There will be lines like `<<<<<<< HEAD`, `=======` and `>>>>>>> 1a2bde2189`
   - Your changes are marked by the section between `HEAD` and the `=======` line
   - Update the file to reflect the new version you expect
5. Commit the resolved file (don't forget to **add**)
6. Push again. You should now manage to update the 'remote' repository
7. Go into your first directory and use **git pull** to get the changes you just made
8. Both of your repository copies are now up to date with `myremote`

Branches

Since there git is decentralized there is no one state of the repository that is correct. To manage this complexity git has the notion of a branch.

- Branches are parallel timelines and are lightweight, so branch often and branch early.

- git **branch** Manages branches.
- git **checkout** Switch between branches.

- Most repositories have a default branch called **master**. Branches are just names for points in the history.
- Once we start working with branches we have to ask ourselves how are we going to join them back up? We can do this by performing a merge.
- You can also associate a local branch with a remote branch by setting it as upstream. git push -**u**.

Exercise

1. Create a new branch, based of master
2. Add a few commits to your branch
3. Change back onto master
4. Check the contents of the file(s) you changed on your other branch whilst you're on the master branch

Pull Requests

SKILLPILLS

- Pull Requests are a GitHub-specific feature (also implemented on other platforms, but not a git feature) used to allow contributing code to a repository.
- They are typically used when you don't have write access to a repository
- They can also be used to allow review of your code, perhaps by a coworker, even if you could directly push your changes
- Without using extensions, you must use the website to use them

Demo + Exercise

- Demonstration...
- Practice:
  1. Clone our repository from GitHub:
     `https://github.com/oist/skillpill-git-group1`
  2. Fork it (use the website)
  3. Add your fork of the repository as a remote to your local repository
  4. Push a change to your fork
  5. Open a pull request on GitHub against the original repository

Rewriting History

Rebases are a way to create fast-forward merges, by altering *history*. Each branch has a root commit from which it diverged from the original commit. By rebasing we change this root. This has a couple of side effects.

- Linear commit history.
- No merge commits within a branch.
- commit-ids change.

- git **pull –ff-only** Don't merge if there are conflict with the remote
- git **rebase** Perform a rebase
- git **rebase -i** Perform a interactive rebase
- git **push -f** Force push your changes
- git **pull –rebase** Perform a pull with a rebase

Exercise

1. create a branch, with some commits
2. go back to master and do some additional work
3. rebase your branch onto master
4. merge your branch onto master

Autosquash

- git config rebase.autosquash true
- git commit –**squash**=*some-hash*
- git commit –**fixup**=*some-hash*

Autosquash will reorder the commits appropriatly before you perform a git **rebase -i**.

Blame

There is no such thing as *good* code. If you are using git with people, chances are that something will break at some time and you need someone to blame. That's what git blame is for:

```
git blame -L 1,3 file
```

Stash

When you are moving between branches you sometines want to keep your non-commited changes associated with the branch you where doing them one.

- git **stash**
- git **stash pop**

- git commit –**amend** Amend the last commit.
- git add -**i** Interactive add
- git add -**p** Interactive add in patch mode.
- git **rm** Removes file.
- git **mv** Move file within repository