SKILLPILLS

Skill Pill: Introduction to Git and Version Control

Lecture 2: Git it on!

Valentin Churavy - 2016

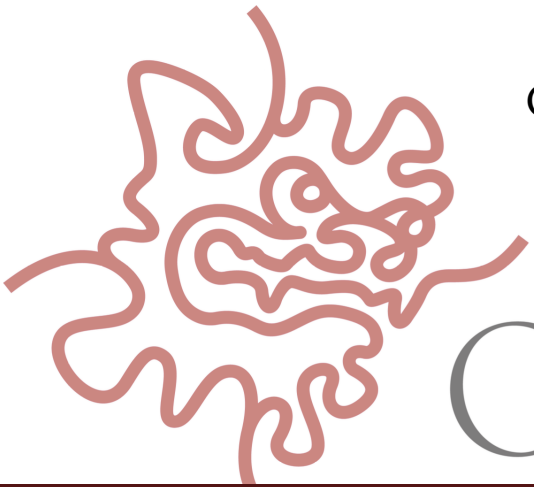
James Schloss - 2018

Christian Butcher - 2019, 2020

Okinawa Institute of Science and Technology

christian.butcher@oist.jp

November 26, 2020



OIST

Overview

SKILLPILLS

1. Recap
2. Working with Remotes
   - Remotes
   - Branches
   - Merging
3. Pull Requests on GitHub
4. More Advanced Topics
   - Rebasing and Rewriting history

Recap

Last week 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!
- **push** : Update remote *refs* and objects.

There's also a **pull** command.

- **pull** : Update from a *remote*. Technicially a combination of **fetch** and **merge** by default.

Some definitions/descriptions

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

Last week 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**.

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.

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

- Most repositories have a default branch called **master** (this is changing to "main" in some cases). 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

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

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. Gives you a chance 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 a nice way from the command line is.

g

it log --**graph** --**decorate** --**oneline** --**all**

Exercise

1. **Clone** a fork of the repository at `https://github.com/oist/skillpill-git` (you may have this available from last week)
2. **Checkout** the 'merge-main' branch
3. **Merge** the 'merge-AddNameToGreeting' branch. Optionally use "--no-ff" to force a merge commit. This will succeed without conflict.
4. Attempt to merge the 'merge-TimeOfDayGreeting' branch. This will cause a merge conflict!

Merge conflict contents

```
def main(username, timeValue):
<<<<<<< HEAD
    print("Hello " + username)

def callFctn(args):
    if len(args) > 1:
        username = args[1]
    else:
        username = "World"

    timeValue = ""
=======
    greeting = getGreeting(timeValue)
    print(greeting + " " + username)

def callFctn(args):
    username = "World"

    if len(args) > 1:
        try:
            timeValue = time.strptime(args[1], "%H:%M")
        except:
            timeValue = ""
    else:
        timeValue = ""

>>>>>>> merge-TimeOfDayGreeting
```

Exercise continued

5. Use a text editor to resolve the conflict
6. Commit the resolved file (don't forget to **add**)
7. Push your branch to your forked repository

This brings us on to "Pull Requests"...

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. In the last exercise we pushed commits to forks of the OIST repository
  2. 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

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

Secrets! II

SKILLPILLS

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
```