SKILLPILLS

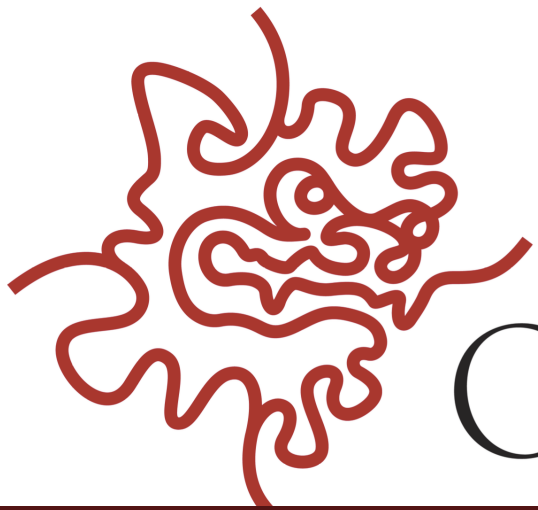
Skill Pill: Introduction to Git and Version Control

Lecture 2: Git it on!

James Schloss and **Valentin Churavy**

Okinawa Institute of Science and Technology
valentin.churavy@oist.jp

July 11, 2018



OIST

Overview

1. Working collaborative
   - Remotes
   - Branches
   - Merging
   - Rebasing and Rewriting history
2. Workflow
   - Documentation and other helpful material

In case of fire

1. git commit -am "untested due to fire"

2. git push -f

3. leave building

- Everything we will work on today will be helpful if you work solo.
- These things will become really useful once you work with multiple people.

In case of fire

1. `git commit -am "untested due to fire"`

2. `git push -f`

3. leave building

- Everything we will work on today will be helpful if you work solo.
- These things will become really useful once you work with multiple people.
- The most useful tip I can give you is: Consistency & Discipline.
- We are going to talk about Workflows at the end of the day.

Interlude: .gitconfig

The file *.gitconfig* can be used to set default options per user or per project. The user files is in *~/.gitconfig*. Each option can also be set with git **config**.

```
[user]
  email = v.churavy@gmail.com
  name  = Valentin Churavy
[github]
  user = vchuravy
[push]
  default = simple
[rerere]
  enabled = true
```

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.

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**

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

1. Clone our repository from Github: `https://github.com/oist/skillpill-git`
2. Fork it
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

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.
- git **cherry-pick** Moving commits 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. Cherry-pick **one** of the commits onto master.

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. 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

1. Merge your branch onto master
2. Create a file with some content, branch and change the contents both in master and the new branch
3. Merge the two branches and correct the merge conflicts

Rewriting History

SKILLPILLS

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
```

Secrets! II

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

Storytelling from the battlefields.

Documentation

- The Git book: `https://git-scm.com/book`
- The Git help/man pages: git **help** or git *command* –help
- Caching your password: `https://help.github.com/articles/caching-your-github-password-in-git/`
- SSH-keys: `https://help.github.com/categories/ssh/`
- Workflow: `https://www.atlassian.com/git/tutorials/comparing-workflows/centralized-workflow`
- Understand .git `https://medium.freecodecamp.com/understanding-git-for-real-by-exploring-the-git-directory`