SKILLPILLS

Skill Pill: Introduction to Git and Version Control
Lecture 2: Git it on!

Valentin Churavy

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

June 2, 2016

OIST

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:

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.

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
```

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.

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**

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**.

Since there git is decentralized there is no one state of the repository that is correct. No manage this complexity git has the notion of a branch. Branches are parallel timelines and they 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**.

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** Get 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 Abort the 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 –pretty –oneline**

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

Secrets! I

Autosquash

- git config rebase.autosquash true
- git commit **--squash=**_some-hash_
- git commit **--fixup=**_some-hash_

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

Bisect

You have a new test case that produces wrong results in your software. Question! When did the error first appear? Git bisect is here to help, by performing a binary search through history to find the last good commit (or the first bad commit).
This is the reason, why every commit should be functional!.

SKILLPILLS

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**

Storytelling from the battlefields.

- 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/`