Notes for Git Workshop presentation

Janus Valberg-Madsen

2018-10-19

These are my presenter notes, made for printing.

Overview

1. First, I will go through what version control is, and why it's a good idea to use it for any type of project
2. Then, in very general terms, I will talk about the central concepts in Git
3. Then, I will briefly cover the features provided by GitHub
4. Then, we will do a live demonstration, going through each step involved in a collaborative workflow
5. Then, **you** will set up Git together with a graphical interface on your computers
6. Then, we will walk you through, step by step, how to set up a repository to use for your P1 projects
7. If there's any time left after that, we have some exercises for you

Why VC/Git?

As the name implies, a VCS gives you control over the version of your project. This means:

- You have a history tree of changes made over the course of the project

  While working, you mark certain points in time as snapshots of your project, so to speak. You give these snapshots apt descriptions, so that at any point, you can easily see what changes have been made when, and even go back to an earlier state.

- With an online repository (e.g. on GitHub), working on the same project is both easy and structured

  A VCS makes sure no changes go unnoticed; On a cloud service like Dropbox, if the same file is saved by two different people, whoever saved the latest version "wins". With a VCS, conflicting changes are made visible, so you can select which parts should "win" (may be both).

- You have backups, so no need to panic if computer dies

So why Git specifically? There are other systems out there (e.g. SVN), but:

- Git is distributed (as opposed to centralised)

  I won't go into technical details about this, but put shortly this type of system makes many commands faster as well as working offline.

- Git is popular

  Problems are inevitable, but no matter what problem you run into, it has been had before, and a detailed explanation on how to fix it exists on the internet. Google "git " and look for StackOverflow links.

- GitHub

  GitHub is an online platform for collaboration with Git. More details about this later.

Git concepts

When you activate Git in a folder (`git init`), it creates a hidden subfolder (`.git/`) for keeping track of stuff. That "stuff" can be represented as shown on the figure.

- **Working Tree:** your files and folders as they appear in your file explorer
- **Staging Area:** a virtual space for preparing a commit
- **Commit History:** a local history of snapshots of the project; arrows represent relationships between two commits (they point to parent snapshots)

The Git workflow revolves around creating snapshots of changes called commits. This is done in two steps.

Add

The first step in creating a commit is adding changes to the Staging Area.

Here, you tell Git which of the changes you've made in your Working Tree that you want to be part of the next commit/snapshot. You don't have to Add all the changes, and generally, you shouldn't.

Commits should contain logical units of change, so add changes that are related to each other, e.g. changes to a chapter in a `.tex` file + an image that is used in this chapter.

Commit

When you have added all the changes you want to group together as one single snapshot, then you commit it. This appends the commit to the history tree, making it accessible as a snapshot.

During committing, you write a brief description (and optionally, a longer, more detailed description), which can really help give you an overview of what happened where, when inspecting the log.

Pushing

Using version control locally is useful in its own right, but it's when you want to collaborate on complex projects that it really shines. One way of sharing work with each other is by setting up a remote repository, which you will all push to and pull from.

On this figure, the left tree is the commit history on our computer, and the right tree is the history on the remote. The local commit `C3` does not show up on the remote until we push it.

`(next slide: combined tree view)`

Here is a more compact way of representing the same concept:

1. Before pushing,
    - our local repository (on `master` branch) has the commits `C1 ← C2 ← C3`
    - the remote (on `origin/master` branch) only has `C1 ← C2`
2. After pushing,
    - Our changes (`C3`) are merged into `origin/master`

Pulling

The exact opposite to pushing is called fetching, but most of the time, you want to pull instead. The difference between the two lies in what happens to your working tree (local files).

Take a look at the figure. The scenario is opposite from before; now `origin/master` is ahead of `master` by one commit. In addition to the commit history, I'm also showing you the working tree, i.e. the local files.

Let's say the commit `C4` changes `file` from version `v3` to `v4`. Before pulling, we don't have `C4` in our commit tree, and `file` is at version `v3`.

When we pull from the remote, Git actually does two things:

1. It fetches new commits from the remote
2. It merges them into our local tree and updates the working tree

On the figure, this is reflected in `file`. Pulling from `origin` brings in `C4` and the changes in it are applied to our working tree (`file v3 → v4`).

NB: pulling is only possible, if the working tree is clean (i.e. doesn't have uncommitted changes).

GitHub

A remote repository is usually hosted on a webserver, and for this purpose, it doesn't get much easier than it is with GitHub.

GitHub is an online service for hosting a Git repository. In addition to that, it also offers several, very useful features:

- Navigate and view code + changes in diff view
- Comment on commits
- Project boards a la Trello

Time for the live demo

1. **Janus:** Go to GitHub and create a new repo
2. **Janus:** Init with README, TeX.gitignore, set to Private
3. **Janus:** Clone to desktop
4. **Janus:** Download project template to local repo
5. **Janus:** Add   Commit   Push
6. **Mads:** pull to computer
7. **Mads:** make a change and push it
8. **Janus:** pull and see the change
9. **Janus:** make and commit a change
10. **Mads:** make a push a change
11. **Janus:** pull and see the conflict
12. **Janus:** explain conflict resolution
13. **Janus:** commit and push merged conflicts