Git cheat sheet

These are based on The Git & GitHub Bootcamp on Udemy. See also this youtube playlist.

Basics

Git is a free and open source version control system developed first by Linus Torvalds in 2005.

A git repository is one with a `.git` hidden directory. Make a directory into a git repository with the command `git init`. (Before doing this, make sure you are not already in a git repository by running the `git status` command.) To convert a git repository into a normal directory, delete the `.git` directory.

The basic git workflow proceeds as follows.

1. Changes to the files are make in the "Working Directory". The command `git add` takes them to the "Staging Area". One can either specify which specific files are to be moved to the staging area by `git add <filename1> <filename2> <filename3>` or just `git add .` which adds all modified files to the staging area. Check that they are in the staging area with `git status`.

2. The command `git commit` takes the changes into the repository and creates a record of the repository at the current stage.

Note: The root of the git repository also can contain a `.gitignore` file that lists all the files (or sorts of files) that are ignored by git.

See the commits with `git log` or `git log --oneline` or `git log --graph --oneline`.

Branches

- `git branch`
- `git switch -c <new-branch-name>`
- `git switch <existing-branch-name>`
- `git branch <new-branch-name>`
- `git checkout <branch-name>`

Warning: when switching branches, sometimes git won't let you do so until you've commited (or stashed) changes. But sometimes git allows you to switch branches and tkes uncommited changes with you.

Merging

We merge branches, not specific commits. Merge is based on the current head branch. For example, so merge the branch `bug-fix` into the branch `master`, we have:

```
git switch master
git merge bug-fix
```

When merging, we produce a new commit node with two outgoing directed edges to the two previous commits.

To resolve merge conflicts:

1. Open files with merge conflicts.
2. Edit the files to remove the conflicts.
3. Remove the conflict markers.
4. Add changes and make commmit.

Diff

- `git diff` compares the staging area with the working directory.
- `git diff HEAD` shows all changes, staged or unstaged, since the last commit.
- `git diff --staged` shows changes between last commit and staged changes.
- So in a sense we have `git diff` + `git dff --staged` = `git diff HEAD`.
- One can compare branches with `git diff branch1..branch2` or commits with `git diff <commit1-hash>..<commit2-hash>`.

Reading the differences proceeds as follows. There is an `a` file marked with minus signs and a `b` file marked with plus signs. The `@@-8,7` notation means that the lines displayed from file `a` start with line 8 and go to 7 subsequent lines. Similarly for `@@+8,7` for `b`. Lines that appear only in `a` are marked with a `-`. Lines that appear only in `b` are marked with a `+`.

Git stash

This is a way to save changes without commiting. Use `git stash` and `git stash pop`. Use `git stash list` to see all the stashes. Use `git stash apply stash@{<n>}` to apply the n-th stash in the list. Use `git stash clear` to clear the stash.

Other commands

- Can go back n commits with `git checkout HEAD~<n>`.
- Use `git restore` to discard changes made since the last commit.
- Use `git reset <commit-hash>` to reset the repository to a particular commit. Use `--hard` to move changes out of the working directory.

Github

To create a repository on GitHub, there are two options:

1. If you have an existing repository on your machine, you can create a new repo on GitHub, connect your local repo using `git remote add <name-of-remote> <url>`. Push your repo to GitHub using `git push <name-of-remote> <local-branch>`. There are variations for pushing to a non-default remote branch.

2. Start the repository from scratch on GitHub by making a new repository, cloning it, working on it locally, and then pushing changes to GitHub.

Note: see all the remotes with `git remote` or `git remote -v`. Usually the GitHub remote is called `origin`.

The command `git fetch` takes all changes from the remote branch and stores them in your local repository without changing the working directory. By contrast `git pull <remote> <remote-branch>` updates (or tries to update) your working directory with the latest version of the remote repository. In a sense, `git pull` = `git fetch` + `git merge`.

The workflow for GitHub is usually:

1. Make a feature branch `feature` and work locally for a while.
2. Pull `main` or `master` from the remote.
3. Resolve conflicts.
4. Rebase if necessary.
5. Push your feature branch to remote, generate pull request.

Rebasing

This is a way to clean up your commit history. Instead of periodically merging changes to the main branch into your local feature branch, you can rebase so that the commits on your local feature branch all appear after the commits on the main remote branch.

Switch to the feature branch. The command `git rebase master` will rebase onto master, so that the commits on the feature branch will appear after the commits on the master branch. The rebasing procedure may involve resolving conflicts, and you may be prompted to use `git rebase --continue` in the process. There is also a way to abort the process.

Warning: never rebase commits that have been shared with others, e.g. on GitHub.

Interactive rebasing is a way of rewriting history even further. Example syntax is `git rebase -i HEAD~4`. One can rework commit messages, fixup by combining commits, drop commits, etc.

Tags

Tags are a way of marking certain versions of the repository. There are lightweight tags and annotated tags. Use `git tag` to list the tags and `git tag <tag-name>` to make a tag at the current commit. Note that tags are not automatically pushed to remote repos.

Reflogs

These are logs not only of the commits, but of all switches between branches, rewritten and deleted commits, etc. They are local and expire after 90 days. Example syntax: `git reflog show <branch>` or `git reflog show HEAD`.

Aliases

Local configurations for a repo are in `.git/configs`. Global user configs are in `~/.gitconfigs`. See GitAlias.

To set them up, add `[alias]` and then intented below, e.g., `logo = log --oneline`.

Useful commands: `git config alias.logo "log --oneline`, or add `--global`.

Git behind the scenes

In more abstract terms, a git repository is a directed acyclic graph with three core types of nodes:

- Blobs. These are always leaves, and correspond to the content of files.
- Trees. These have directed edges to either other trees or to blobs, and correspond to directories.
- Commits. Each of these has a single directed edge to a tree, and generally a single directed edge to another commit (though "merge commits" have directed edges to two other commits).

Each time a commit is made, the contents of each file run through a hash function. There is a dictionary where the key corresponding to a file's content hash value references an encrypted and compressed version of that content. One then makes a standardized file for each directory, which lists the name of each of its files, the hash of each file's contents, the name of each of its subdirectories, and the hash of each's subdirectory standardized file. (So this proceeds recursively from directories with no subdirectories.) The contents of this standardized file are hashed and encrypted and compressed, and we get a new key-value pair in the dictionary. Finally, we have a standardized file for the commit, which lists the commit message, the hash of the root directory, the author, the commiter, a timestap, and the previous commit(s) it is derived from. The contents of this standardized file are also hashed, encypted, and compressed.

There are also other types of nodes: branches, each of which has a single directed edge to a single commit; HEAD, which generally points to a branch node, unless the HEAD is "detached", in which case it points to a commit; and annotated refs.

If we're on a branch and make a commit, the branch moves to the new commit. The HEAD keeps pointing at the branch, so effectively also moves to the new commit.

Miscellaneous commands / information

- Configure VSCode to the the default code editor. Use `code <filename>` to open files.

- The command `open .` opens a finder window at the current working directory.