Git Cheatsheet

Meedos

Introduction

This is a short cheatsheet on how I use Git daily. I don't use the more complex commands yet as I haven't learned them. Hope this is useful to some of you. I'll try to break it up in use-cases or scenarios with examples as I believe it is easier to understand. It is highly suggested that you consult the man pages for every command that you try as it is impossible to cover all the flags in this cheatsheet.

Initializing a new git repo

On an existing codebase/project or on a empty directory type :

```
$ git init
```

This command creates a hidden folder named .git where all the git magic happens (hashes, working dir, etc...).
By default this creates the default branch "master" which you are on now, more on that later.

Most basic usage

The following commands are the most used commands by far, you'll do them a couple of times a day.

Add work to the index

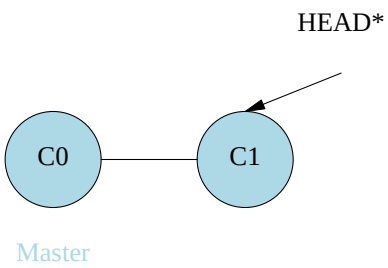
```
$ git add .
```

This command adds your files to the index waiting to be committed. The "." means all that is within the current directory ("*" works fine too). As an image, "git add" takes a snapshot of the content of the working tree, that snapshot will then be used for the next commit.

Committing To save your changes to the repository

```
$ git commit -m 'This is a commit message'
```

This creates a new commit with the current index's content.



HEAD* is where you are currently in the tree. Doing another commit will continue the tree to the right with "C2" so on and so forth.

Pushing to a remote

Let’s say you are working with a team and the code base is meant to be stored on a remote server such as gitlab.com, gitea, or even github.com. You first need to create a repository on one of those services and push you local changes to that remote. The next commands will assume you already created a repo at https://gitlab.com/meedos/test_repo.git

Adding a remote

You can add a remote with a given name and a url with.

```
$ git remote add origin https://gitlab.com/meedos/test_repo.git
```

A local repo can have multiple remotes. You can then proceed to push ("send") your changes to the repo with the adequate name :

```
$ git push origin master
```

Also you can set a default upstream for a branch that way you don’t have to type the whole command every time.

```
$ git push --set-upstream origin master
```

This way you can simply type :

```
$ git push
```

To push to origin/master.

Pulling a remote

Now, let’s say a coworker pushes work to you remote repository, or you simply want to access you work from a different computer, you’ll want to pull that work onto your local repository. To achieve this use :

```
$ git pull origin master
```

Or if you’ve setup the upstream, simply :

```
$ git pull
```

Note : this cases it is preferable to use "git fetch" instead to avoid automatic merging.

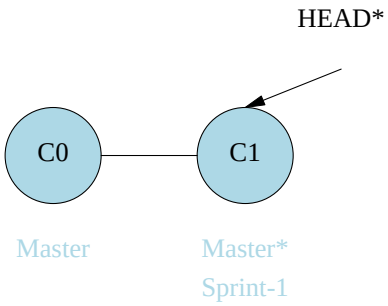
If you are extremely lucky and work alone, you could use only these commands and get away with it. But that won’t happen and there’s is so much more useful aspects to git.

Branching

Branching as it names implies allows you to create a diverging branch to your git tree. Branching should become second nature as it is a good practice to implement a feature on a separate branch without messing out your master branch. Some create a branch for each versions, for each big functionality (or User Story), or even a branch per platform (Linux, Windows) in some cases.
In any case, to create a branch type (and choose a branch name) :

```
$ git branch [BRANCH_NAME]
```

Let’s say we named our branch Sprint-1. The result can be mentally seen as :



The new branch is up to date from the branch it was created from.

Checking out

Now is a good time to talk about "git checkout". This command allows you to move around the tree. You can switch branches or go to previous commits

switch branch :

```
$ git checkout Sprint-1
```

Go to commit C0 :

```
$ git checkout C0
```

result :