

## Git & GitHub

04/05/2020

### LESSON 1: What is Version Control?

#### → Version Control

→ Easy trick to understand this is - read the words in reverse - so "version control" would become "control version"!

↳ So, a version control system is just software that helps you control (or manage) the different versions... of something (typically source code).

#### → Difference b/w Git & GitHub

<u>git</u>	<u>GitHub</u>
→ version control tool	→ service that hosts Git Projects.
→ It is a distributed version control system.	

08/05/2020

#### → git init (initialize)

→ create brand new repositories (repos) on your computer.

#### git clone

→ copy existing repos from somewhere else to your local computer.

#### git status

→ check the status of repo



→ git init : creates the ".git" directory.  
 ↳ The directory is the brain/storage center for the repository.  
 ↳ It holds all of the configuration files & directories & is where all of the commits are stored.

→ pwd : To get the current working directory.

→ git clone :  
`git clone https://github.com/udacity/course-git-blog-project`

what if you want to use a different name instead of default one

`git clone https://github.com/udacity/course-git-blog-project (space) blog-project`

2. Git log :

→ Displays information about the existing commits.

Git show :

→ Displays info about the given commit.



Date.....

→ `git log --oneline` :

→ It will display the short SHA of the commit with message.

→ `git log --stat` :

→ displays the file(s) that have been modified.

→ displays the number of line that have been added/removed.

→ displays a summary line with the total number of modified files & lines that have been added/removed.

→ `git show` :

→ This is used to show about particular type of commit. Just we have to pass first 7 letters of "SHA".

→ It can also combine with :

• `--stat`

• `-p` or `--patch`

• `-w` ← to ignore changes to whitespace.

## 2. Committing repository

1. git add :  
→ add files from the working directory to the staging index.

2. git commit :  
→ Takes files from the staging index & save them in the repository.

3. git diff :  
→ Displays the difference b/w two versions of a file.

\* git add . → refers to all files to move into the staging index.

\* I want to bypass the code editor.

`git commit -m "initial commit"`

09/05/2020

4. git tag :

→ The `git tag` command is used to add a marker on a specific commit. The tag does not move around as new commits are added.

→ `git tag -a v1.0`

↳ add a tag to the most recent commit.

↳ "-a" flag is used to tell git to create an annotated tag.



→ If you don't provide the flag (i.e., `git tag v1.0`) then it'll create what's called a light weight tag.

→ To verify tag → `git tag`

\* Deleting a tag:

→ A git tag can be deleted with the `-d` flag (`-d` for delete)

→ `git tag -d v1.0`

it'll delete the tag `v1.0`

\* Adding a tag to a past commit

→ All you have to do is provide the SHA of the commit you want to tag.

`git tag -a v1.0 a87984`

10/05/2020

\* Git branch:

→ `git branch` command is used to interact with Git's branches:

`git branch`

→ It can be use to:

↳ List all branches names in the repository.

↳ Create new branches.

↳ delete branches.

1. Create a branch

→ `git branch sidebar`

If you want to create a branch called "sidebar", you'd run this command.

2. Git Checkout command

→ Remember that when a commit is made that it'll be added to the current branch.

→ So, even though we created the new "sidebar", no new commits will be added to it, since we haven't switched to it, yet.

→ If we made a commit right now, that commit would be added to the master branch, not the sidebar branch.

→ To switch b/w branches, we need to use Git's checkout command.

`git checkout sidebar`

3. Delete a branch

`git branch -d sidebar`

→ You can't delete a branch on which you are currently working.

→ You have to switch a branch or create a new branch



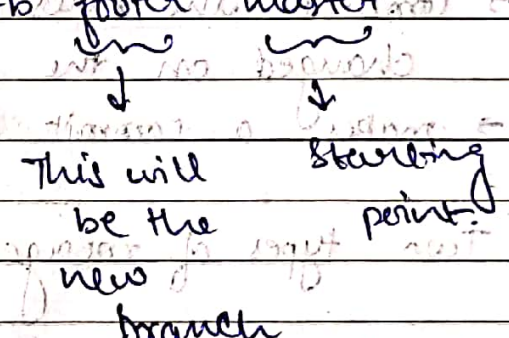
→ If you have made any commit to the branch you wanna delete now. You can't do it with the above command. You have to write:

`git branch -D sidebar.`

`-D`: used to force deletion.

What if you want to make another branch but it should start from the master branch.

`git checkout -b footer master`



This will be the new branch. Starting point.

• See all branches at once

`git log --oneline --decorate --graph --all`

• The "`--graph`" flag adds the bullets & lines to the leftmost part of the output. This shows the actual branching that's happening.

• The "`--all`" flag is what displays all of the branches in the repository.

Date.....

## Merging

Combining branches together is called merging.

`git merge <name-of-branch-to-merge-in>`

\* when a ~~branch~~ merge happens, git will:

→ look at the branch, ~~to merge~~ that's going to merge.

→ look back along the branch's history to find a single commit that both branches have in their commit history.

→ combine the lines of code that were changed on the separate branches together.

→ makes a commit to record the merge.

Two types of merge:

### 1) fast-forward merge

↳ A fast-forward merge will just move the currently checked out branch forward until it points to the same commit that the other branch.

↳ In this case, ~~the~~ branch is ahead.

git merge ~~branch~~



2) Regular merge :

→ Two divergent branches are combined.

→ A merge commit is created.

`git merge`

\* When a merge is performed & fails, this is called a merge conflict.

→ when merge conflict occurs?

→ When the exact same line(s) are changed in separate branches.

→ Changing the last commit

→ `git commit --append`

→ Reverting a commit

→ `git revert <SHA-of-commit-to-revert>`

→ This commit will undo the changes that were made by the provided commit.

→ creates a new commit to record the change.



2) Figure out the value

`rm` → this means deleting a commit.

→ It can be used to remove all

- 1) move the HEAD and current branch pointer to the referenced commit.

- 2) Erase commit with the  $-$  based flag.

- 3) Moves committed changes to the staging index with the --soft flag.

- 4) unstages committed changes --mixed flag

→ Typically, ancestry references are used to indicate previous commits. The ancestry references are named `HEAD`, `main`, or

- $\wedge \rightarrow$  indicates the parent commit.
- $\sim \rightarrow$  indicates the first parent commit.

→ git reset <reference-to-commit>  
 ↓  
 --mixed (by default, move this  
 <reference-to-commit> into working directory)  
 or

new root. append --soft (move into staging index)

-- word (~~more~~ Erase the commit)

→ Backup branch

→ git branch backup\_



## 1. First time Git configuration

→ # sets up Git with your name.

```
git config --global user.name "<Your-Full-name>"
```

→ # sets up Git with your email.

```
git config --global user.email "Your-email-address"
```

→ # make sure that Git UI is colored

```
git config --global color.ui auto
```

→ # displays the original state in a conflict

```
git config --global merge.conflictstyle diff3
```

```
git config --list
```

## 2. Git & Code Editors

vs code setup

```
git config --global core.editor "code --wait"
```