Version Control with Git and GitHub

What is version control?

> Your program keeps on changing. You need to keep those changes or at least track them.

- Having a version control system means you will be able to rollback to previous versions when you introduce major bug. (no need to quick fix which might generate more bugs)
- Historically, keep copies of files. But hard to keep track of what changes were made, when and by whom?

Diffing

- finding difference between two files

> `diff file1.ext file2.ext`

> wdiff can be used too, -u flag can be used with diff for different format

- Applying changes:
  - You can send a diff to someone
  - `diff -u old_file new_file > change.diff`
  - diff file is also called patch file
  - To apply the diff `patch file_old.py < change.diff`
- Why send patch and not send whole file?:
  - just changes
  - original file might have changed
- Diff command

```
~$ cat menu1.txt
Menu1:

Apples
Bananas
Oranges
Pears

~$ cat menu2.txt
Menu:

Apples
Bananas
Grapes
Strawberries
```

```
~$ diff -u menu1.txt menu2.txt
--- menu1.txt   2019-12-16 18:46:13.794879924 +0900
+++ menu2.txt   2019-12-16 18:46:42.090995670 +0900
@@ -1,6 +1,6 @@
-Menu1:
+Menu:

 Apples
 Bananas
-Oranges
-Pears
+Grapes
+Strawberries
```

- Patch command

```
~$ cat hello_world.txt
Hello World
~$ cat hello_world_long.txt
Hello World

It's a wonderful day!
~$ diff -u hello_world.txt hello_world_long.txt
--- hello_world.txt     2019-12-16 19:24:12.556102821 +0900
+++ hello_world_long.txt        2019-12-16 19:24:38.944207773 +0900
@@ -1 +1,3 @@
 Hello World
+
+It's a wonderful day!
~$ diff -u hello_world.txt hello_world_long.txt > hello_world.diff
~$ patch hello_world.txt < hello_world.diff
patching file hello_world.txt
~$ cat hello_world.txt
Hello World

It's a wonderful day!
```

What are version control systems?

- VCS keep track of all changes made.
- Collaboration easier.
- We can make edits to multiple files and treat that collection of edits as a single change, which is commonly known as a commit.
- VCS also allow you to record a message with commit.
- Repositories is like a folder for your code.
- VCS can be used for any type of file but you can easily visualize changes in text files.

Git

- Created by Linus Torvalds, open source for collaboration for Linux
- Distributed architecture
- Git can act as server program or client program

Installing git

```
git --version
```

- Linux

```
sudo apt install git
```

- MacOS

```
git --version # this also will give you option
# or install from website
```

- Windows

```
# install from website
```

Installing git on windows

```
https://gitforwindows.org #download the file and install
```

- leave installation path
- components to install can be left as is as well
- Now choose editor: Perhaps VS code
- Adjusting path environment:
    - Use pre selected
- OpenSSL library is fine
- Line ending can be left as is
- You can leave terminal emulator as is
- Extra options as is
- experimental features can be left as is

First steps with git

```
git config --global user.email "me@example.com"
git config --global user.name "My name"

git config -l # to see configs
```

```
git init # for new
git clone # for copying from cloud
```

- Staging area or index (list of what to commit)

```
git add disk_usage.py # what to add to staging area
git status # what is in staging area
```

- Commit

```
git commit -m "First commit"

# also try git commit only
```

> Working area, staging area, repository

Tracking files

- Tracked files are part of snapshot of repo

> Modified state, staged files, committed files

Basic Git workflow

- Initialize git
- git config -l for configuration
- Make changes
- Add files to staging area
- Commit with message

Commit messages and git log

```
git log # This shows list of commits with messages
```

- Summary

```
## Git Basics
- Git project consists of three sections: Git directory, working tree, and
staging area.
- `git config` command is used to set user identification for Git
repositories.
- `git init` command creates a new repository or re-initializes an existing
one.
- `ls -la` and `ls -l .git/` commands are used to check the existence and
contents of the Git directory.
- `git add` command is used to track files and add them to the staging
area.
- `git status` command provides information about the current working tree
and pending changes.
- `git commit` command saves changes from the staging area to the Git
directory.

## Writing Commit Messages
- Commit message consists of a summary and a description.
- Summary should be a header containing 50 characters or less.
- Description should be under 72 characters and provide detailed
information about the change.


[Example of a commit message](https://commit.style/)
```

Using Git locally

- Skipping the staging area

```
git commit -a -m "Description here" # You don't need to perform `git add .`
if you do this but all tracked files will be added
```

- Head alias: currently checked out snapshots of your project.
- You can move head around to see version of your project.
- Getting more info with git log:

```
git log -p  #p for patch shows changes made in commits
git log --stat #how many changes
```

- Git show

```
git show commit_id # to see description of specific commit
```

- Git diff

```
git diff # changes in repo
git diff file # changes in file

git diff --staged # shows staged changes, diff shows unstaged only
```

Working with files

```
git rm filename.py # remove from git
git mv filename.py new_name.py #to rename

# automatically
```

- .gitignore file

```
# add file names here to ignore them
```

Reverting changes

- Git checkout

```
# after you made changes
git checkout fiename # Restores to latest stored snapshot

# You can use -p flag to revert individual changes
```

- Unstage

```
git reset HEAD <file> # To unstage the file

# -p can be used as well
```

- Amending commits

```
# add missing file
git commit --ammend ## will open editor where you can change the
description

# dont use it on public commits
```

Rollback

```
git revert HEAD # Will open text editor

## That will revert the commit by adding new commit with 'inverse' changes
```