Git and GitHub Udemy Course notes

- Version control system, software that tracks and manages changes to files over time.

* Track changes across multiple files
* Compare version of a project
* ‘time travel’ back to old versions
* Revert to previous versions
* Collaborate and share changes
* Combine changes

- essence of git – create ‘checkpoints’ that you can return to in case you mess up the project.

- other systems, Subversion, CVS and Mercurial

- git vs github (not the same thing)

Git -> don’t need internet to use it, runs local on your machine

GitHub -> hosted in the cloud, makes it easier for collaboration (runs using git)

- git (2 ways of interacting with)

🡪 primarily as a command line tool, used through terminals (expected to be able to use as a software developer)

🡪 can also be accessed using GUI (graphical User Interface), like github desktop,

Easier to use (more visual), but not as fast as using a terminal

Using gitkraken for this course

- Git Bash

🡪 command line interface (Unix-based), required for windows. Windows only have command prompt as their command line interface, which is not Unix-based)

- Setting up git

🡪 Setting up name

*$ git config --global user.name "Sia Kian Zhong"*

🡪 Checking current name

*$ git config user.name*

🡪 Setting up email account

*$ git config –global user.email “kianzhongs@gmail.com*

🡪 checking current email address

*$ git config user.email*

- Basic unix commands (for Git Bash)

🡪 Navigation commands

* List contents of current directory -> <ls>
* List ALL contents of current directory, including hidden files and folders -> <ls -a>
* List contents of specific directory -> <ls (name of target directory)>
* Open file explorer of current directory -> <start .>
* Print path of working (current) directory -> <pwd>
* Change directory to specific-> <cd (path of target directory)>
* Change directory down by 1 folder -> <cd ..>
* Change directory back to main (~) folder -> <cd>

🡪 File and folder creation

* Create a new file -> <touch (path to file name OR multiple file names separated by spaces)>
* Create a new folder/directory -> <mkdir (name of folder/directory OR multiple folder names separated by spaces)>

🡪 File and folder deletion

* Delete a file permanently (not to trash can) -> <rm (name of file OR multiple file names separated by spaces)>
* Delete a folder permanently -> <rm -rf (name of folder/directory OR multiple folder names separately spaces)>

- Git Terminology

🡪 Git Repository -> workspace which tracks and manages files within a folder

- Git Commands (initialising)

* Reports the status of current git repository -> <git status>
* Instantiates/Initialises an empty git repository in current directory (i.e. enables git inside the folder, happens only once at the start of a project) -> <git init>

\*\* Big mistake that beginner Git users make

🡪 Note that all nested folders within a Git repo is tracked by Git, therefore don’t initialise ANOTHER Git repo in a nested Git folder. SEPARATE your Git Repos

🡪 BEFORE <git init> is run, run <git status> to make sure current directory is not a Git repo.

\* can resolve multiple nest git repos by deleted the hidden .git folders

- Committing

-> checkpoints in a git repository, along with a message

-> can group new saves in multiple files and folders in 1 commit, typically grouped together for a particular function, e.g. added new features for web application

-> areas in committing

* Working directory
* Staging area
* Git Repository

-> commit messages -> statement that summarises changes made in a commit

Git Commands (committing)

* Adds files onto the staging area to prepare for 1 commit -> <git add (file name)>
* Commit all files in the staging area onto git repository in 1 commit -> <git commit>
* Same as above but with a message (but recommended) -> <git commit -m “my message”>
* Check history of Git Repository -> <git log>
* Add ALL changed files in working directory into staging area -> <git add .>
* Open folder in VS code (once configured) -> <code .>