SOFTWARE ACADEMY
GIT MANUAL

CHAPTER 1

Introduction

Operation System

An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.

Types of Operating System

There are many types of Operating System but we are listing out the most popular operating system here

- Windows OS
- Linus OS
- Apple macOS

Windows OS

Created by Microsoft, Microsoft Windows is one of the most popular proprietary operating systems for computers in the world. Most personal computers come preloaded with a version of Microsoft Windows.

Apple macOS

Developed by Apple, this proprietary operating system runs on the manufacturer's personal computers and desktops.

Linux

Linux Operating System is an Operating system that runs on a Linux Machine created by the Finnish programmer Linus Torvalds, Linux is today developed by programmer collaborators across the world who submit tweaks to the central kernel software.

An operating can have a Graphical User Interface (GUI) and Command Line Interface (CLI)

GUI

The graphical user interface is a form of user interface that allows users to interact with electronic devices through graphical icons eg when you drag and drop items on your computer system, when you right click an icon etc.

CLI

A command-line interface (CLI) is a text-based user interface (UI) used to run programs, manage computer files and interact with the computer. We will be using command line to interact with the computer throughout this training.

Server Operating System (Server OS)

A server operating system (OS) is a type of operating system that is designed to be installed and used on a server computer.

Some common examples of server OSs include:

- Red Hat Enterprise Linux
- Windows Server
- Mac OS X Server

Client OS

The Client Operating System is the system that works within computer desktops and various portable devices, smartphones and small computer devices are able to support client operating systems.

Chapter 2

Linux Commands

The command line or command-line interface, is a text-based application for viewing, handling, and manipulating files on your computer. It's much like Windows Explorer or Finder on the Mac, but without the graphical interface. Other names for the command line are: CMD, CLI, prompt, console or terminal.

Open the command-line interface

To start some experiments we need to open our command-line interface first. Depending on your version of Windows and your keyboard, one of the following should open a command window. Just type cmd on the search bar the command window opens.

Difference between OS X prompt, Linux and Windows Prompt

Prompt

You now should see a white or black window that is waiting for your commands.

Prompt: OS X and Linux

```
$
```

Prompt: Windows

```
>
```

Basics

Each operating system has a slightly different set of commands for the command line, so make sure to follow instructions for your operating system. In this training we will be using the Linux commands.

Current directory

The command below shows your current directory

```
$ pwd
/Users/olasitarska
```

Note: 'pwd' stands for 'print working directory'.

Linux command for listing Directories/files in a folder

```
$ ls
```

Change current directory

```
$ cd Desktop
```

Check if changed: OS X and Linux

```
$ pwd
/Users/olasitarska/Desktop
```

Create directory

How about creating a practice directory on your desktop? You can do it this way: This little command will create a folder with the name practice on your desktop. You can check if it's there by looking on your Desktop or by running a **ls**

```
$ mkdir
```

Creating multiple Directory

To create multiple directory use the command below

```
$ mkdir css js
```

Creating a file

To create a file use the command below

```
$ touch index.html
```

Creating multiple files

To create multiple files use the command below

```
$ touch index.html about.html
```

Renaming a file

To rename a file use the command below

```
$ rename about.html about-us.html
```

You can check if the changes occur by running the command below

```
$ ls
```

Moving out of a directory

We don't want to leave a mess, so let's remove everything we did until that point.

```
$ cd ..
```

How to copy a folder

Use the command below to copy a folder to another folder

We are coping **js** folder into **assets** folder

```
$ cp js -r assets
```

How to copy a folder out of a folder

You have to enter the folder and run the command below

```
$ cp js -r ../
```

Copying multiple files

To copy multiple files or folder use the command below

```
$ cp -r js css assets
```

Moving/Cutting a file into a folder

To a move a file to a folder run the command below

```
$ mv index.html pages
```

Moving a file out of a folder

```
$ mv index.html ../
```

Deleting a folder

To delete a folder run the command below

```
$ rm –r css
```

Deleting multiple folders

Run the command below to delete multiple folders

```
$ rm –r css js
```

Deleting a file

To delete a file run the command below

```
$ rm style.css
```

Deleting multiple files

To delete multiple files run the command below

```
$ rm style.css script.js
```

Deleting multiple files of the same type

```
$ rm *.js
```

Adding content to a file

```
$ echo ‘Hello World’ > index.html
```

To view content in a file

```
$ cat index.html
```

Clear

The clear command is use to clear command prompt history

```
$ clear
```

Exit

That's it for now! You can safely close the command line now. Let's do it the hacker way, alright?

```
$ exit
```

Chapter 3
Git and Github

Git

Git is a virtual control system that used to track changes made to our computer files, it allows team members irrespective of their geographical location to work on the same codebase. Git was designed and developed by Linus Torvalds for Linux kernel development.

When your codebase is Git enabled it becomes a repository

Github

Github is a webite where we host our Git Repository you will need to create an account in Github for you to host your repository on Github.

Why a Version Control System like Git is needed

Real life projects generally have multiple developers working in parallel. So a version control system like Git is needed to ensure there are no code conflicts between the developers.

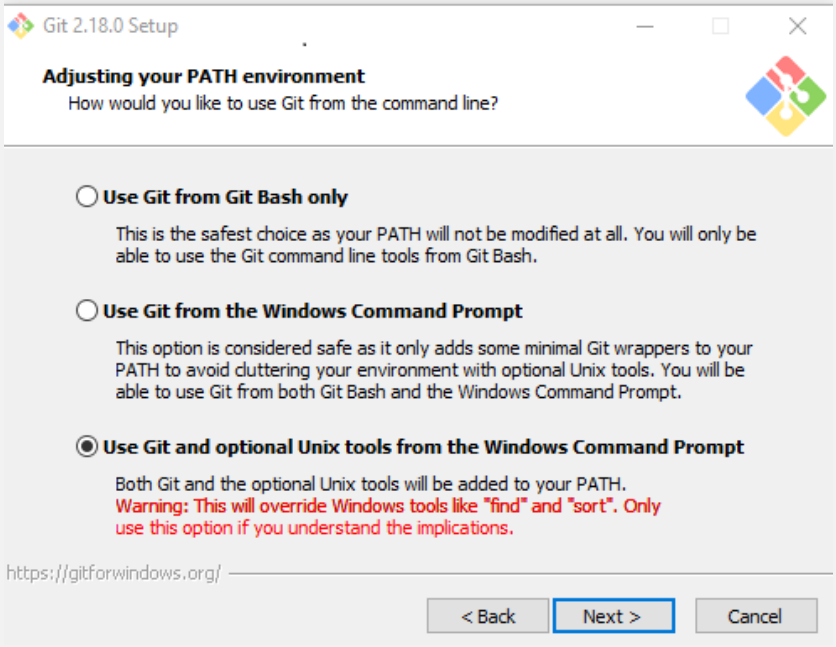
Additionally, the requirements in such projects change often. So a version control system allows developers to revert and go back to an older version of the code.

Finally, sometimes several projects which are being run in parallel involve the same codebase. In such a case, the concept of branching in Git is very important.

Installing Git

Installing Git: Windows

You can download Git from git-scm.com. You can hit "next" on all steps except for one; in the step entitled "Adjusting your PATH environment", choose "Use Git and optional Unix tools from the Windows Command Prompt" (the bottom option).



This option will allow you to run linux command on windows OS Other than that, the defaults are fine. Checkout Windows style, commit Unix-style line endings is good.

Do not forget to restart the command prompt or powershell after the installation finished successfully

Configure Git

After you install Git, configure it for first time use using git config, a built-in tool that obtains and sets configuration variables.

These configuration variables are located in three different places on a GNU/Linux system:

- /etc/gitconfig - stores the configuration information for all system users and their respective repositories.
- ~/.gitconfig - stores user-specific configuration files on the system.
- .git/config - this is the configuration file of your current working repository.

For a Windows system, the .gitconfig file is located in the $HOME directory of the user's profile. The full path is C:\Document and Settings\$USER or C:\Users\$USER
After installing Git make sure your username and email address are set correctly. To verify, use the command:

```
$ git config  --list
```

If your name and email are not listed in the output, use the following commands to set them manually, replacing examplename and user@example.com:

```
$ git config --global user.name examplename
```

```
$ git config --global user.email user@example.com
```

Set your default text editor, replacing editor-name with your desired editor:

```
$ git config --global core.editor editor-name
```

The output of git config --list should show echo the information you inputted:

```
MacBook-Pro:~ user$ git config --list

user.name=exampleuser

user.email=user@email.com

core.editor=editor-name
```

Getting Help

If you ever need help while using Git, there are two equivalent ways to get the comprehensive manual page (manpage) help for any of the Git commands:

```
$ git help <verb>
```

These commands are nice because you can access them anywhere, even offline. If the manpages and this book aren't enough and you need in-person help, you can try the #git or #github channel on the Freenode IRC server (irc.freenode.net).
In addition, if you don't need the full-blown manpage help, but just need a quick refresher on the available options for a Git command, you can ask for the more concise "help" output with the -h or --help options, as in:

```
$ git add -h
```

Chapter 5

Getting a Git Repository

You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can clone an existing Git repository from elsewhere.

In either case, you end up with a Git repository on your local machine, ready for work.

Initializing a Repository in an Existing Directory

If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to go to that project's directory. If you've never done this, it looks a little different depending on which system you're running:

for Linux:

```
$ cd /home/user/my_project
```

for Mac:

```
$  cd /Users/user/my_project
```

for Windows:

```
$ cd /c/user/my_project
```

and type:

```
$ git init
```

This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet.

If you want to start version-controlling existing files (as opposed to an empty directory), you should probably begin tracking those files and do an initial commit. You can accomplish that with a few git add commands that specify the files you want to track, followed by a git commit:

To track a single file run the command below

```
$ git add index.html
```

To track files of the same type run the command below

```
$ git add *.html
```

Or

```
$ git add *.php
```

To view status of your repository run the command below

```
$ git status
```

This gives information concerning the state of your repository The files that are tracked will be colored green while the files not tracked will be colored red

To save changes to your repository or commit to your repository run the command below

```
$ git commit -m "Created html files"
```

To remove files from Git you need to remove it from your tracked files

```
$ git rm PROJECTS.md
```

Chapter 5

Git Branch

In Git, a branch is a new/separate version of the main repository. Let's say you have a large project, and you need to add login functionality to the project and you do not want this login functionality mess up the original project, you can use a git command to create a separate version of the project and then work on the login functionality in this project if everything works fine the way you want it you can now merge this version to the original.

By default when you create a repository you are set to the master branch the master branch is a default branch that serves as base for creating other branches.

To create a branch you can run the command below

```
$ git branch new_branch
```

Note that the new_branch can be called anything eg "testing", "development"

To list branch in your local repo

```
$ git branch
```

To move to a new branch

```
$ git checkout new_branch
```

Shortcut for creating a new branch and moving into it

```
$ git checkout -b another_branch
```

Deleting a branch

To delete this new branch "another_branch" I need to checkout to another branch eg "master" or "new_branch" and then delete the branch eg

```
$ git checkout new_branch
```

```
$ git branch -d another_branch
```

Git Merge

Git merge allows us to merge a branch into another branch eg to merge "new_branch" into the master you have to checkout to the master branch and then merge "new_branch"

```
$ git checkout master
```

```
$  git merge new_branch
```

To rename a branch

The format for renaming a branch is below git branch -m old_branch new_branch In our case we do this below

```
$ git branch -m new_branch another_branch
```

You can run git branch to verify

```
$ git branch
```

Merge Conflict

Merge conflict occurs when a line of code in a particular file on a branch is different from the same line of code on the same file in another branch eg
In a **script.js** file in the **master** branch I have the code below take note where you have **c = a + b;**

```
a = 10;
b = 20;
c =  a + b;
```

And in **script.js** file in **another_branch** I have the code below take note where you have **d = a + b;**

```
a = 10;
b = 20;
d = a + b;
```

A merge conflict will occur because the codes are different are different on the same file on different branches. Below is how the merge conflict will look like

```
a = 10;
b = 20;
<<<<<<< HEAD
c =  a + b;
=======
d = a + b;
>>>>>>> another_branch
```

Starting from the code below is the branch you are on and <<<<<< HEAD down to ====== is the start and stop of where the conflict is occurring on the file in the **master** branch

```
<<<<<<< HEAD
c =  a + b;
=======
```

While ===== down to >>>>>> another_branch is the start and stop of where the conflict is occurring in **another_branch**

```
=======
d = a + b;
>>>>>>> another_branch
```

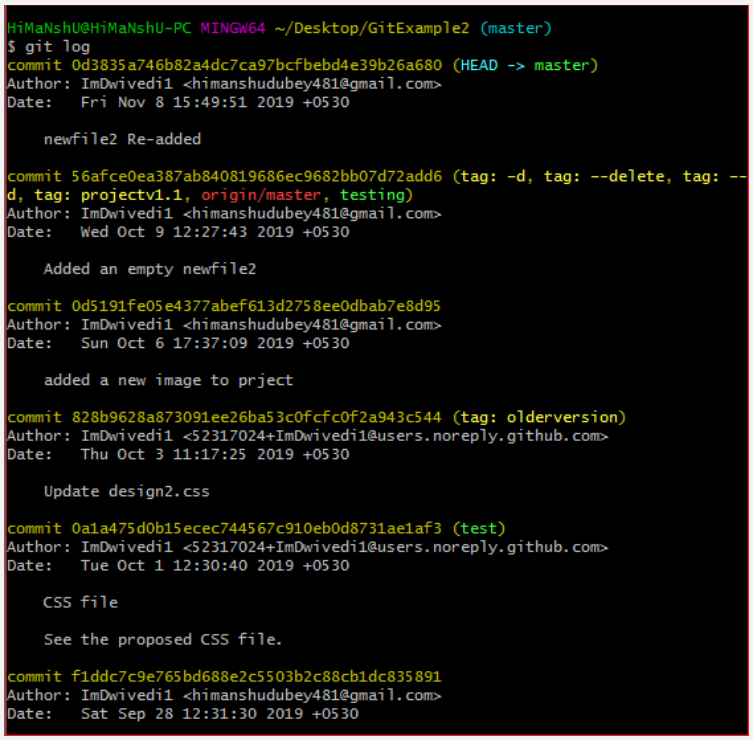
Git log

The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message, and other commit metadata.

Git Reset

Let's say in the process of your application development you had issues with your project you can revert back to the last time your project was working normal this technique is called "reset to Head" and it is quite a powerful tool for developers to use.
To reset the head to a point the project was working normal, you have to run the git log to get the commit id, you can choose the first-four or first-six character of the commit id

```
$ git log
```




```
$ git reset --hard e08845
```

This will take you to the time when the project was working fine

Connecting local repository to remote repository

1. In the upper right corner, next to your avatar or identicon, click and then select New repository.
2. Name your repository e.g sample_repo.
3. Write a short description.
4. Click Create repository.

Connect your local repo to remote

```
$ git remote add origin https://github.com/your_username/sample_repo.git
```

Push your local repo to remote

To push your local repo to the remote repo for the first time run the command below

```
$ git push -u origin master
```

For subsequent pushing to the master branch run the command below

```
$ git push
```

To push your local repo to the remote repo run the command below

```
$ git push origin another_branch
```

Git Pull

Run this command to get your changes from the remote repo to your local repo

```
$ git pull
```

To pull a remote branch locally run the command below

```
$ git pull origin remote_branch_name
```

Git Fetch

Git fetch is a git commad you run to get updates from the remote repo this command does not merge the changes from the remote repo to local repo rather it gets or refreshes the remote repo for updates you can now run **git pul**l to merge the updates.

```
$ git fetch
```

Git Clone

The git clone command is used to download a repository from Github

```
$ git clone https://github.com/Bussyboo98/eaglesbrand.git
```