\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GIT & GITHUB

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

========================================

Date: Friday, August 18, 2023

Topic: Introduction to Git and Github

Facilitator: Aminu

========================================

Download Git:

# Windows: https://git-scm.com/download/win

# Mac: https://git-scm.com/download/mac

# Linux: https://git-scm.com/download/linux

References:

# Manual: https://git-scm.com/docs/git#\_git\_commands

# Book: https://git-scm.com/book/en/v2

# Videos: https://git-scm.com/videos

# Cheatsheet: https://training.github.com/

# Visual Cheatsheet: https://ndpsoftware.com/git-cheatsheet.html#loc=remote\_repo;

# Git (Lab Practice): https://gitimmersion.com/lab\_01.html

# Games/Practice:

- https://learngitbranching.js.org/

- https://ohmygit.org/

- https://gitexercises.fracz.com/

- https://www.w3schools.com/git/exercise.asp

# Atlassian: https://www.atlassian.com/git/tutorials

Areas to Learn:

- intro to git and github

- setup and config

- adding remote repos and cloning remote repos

- making our first commit

- pushing to remote repo

- creating github tokens

- creating/switching branches

- pull/fetch/merge requests

- rebasing

- resetting

- reverting

- What is Version Control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

file.txt

file1.txt

file3.txt

- Types of Version Control Systems (VCS)

- Local Version Control Systems (LVCS)

Many peopleâ€™s version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if theyâ€™re clever).

This approach is very common because it is so simple, but it is also incredibly error prone.

It is easy to forget which directory youâ€™re in and accidentally write to the wrong file or copy over files you donâ€™t mean to.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

- Centralized Version Control Systems (CVCS)

The next major issue that people encounter is that they need to collaborate with developers on other systems.

To deal with this problem, Centralized Version Control Systems (CVCSs) were developed.

These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.

For many years, this has been the standard for version control.

- Distributed Version Control Systems (DVCS)

This is where Distributed Version Control Systems (DVCSs) step in.

In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients donâ€™t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.

Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.

Every clone is really a full backup of all the data.

- What is Git?

The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data.

Conceptually, most other systems store information as a list of file-based changes. These other systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they store as a set of files and the changes made to each file over time (this is commonly described as delta-based version control).

Git doesnâ€™t think of or store its data this way.

Instead, Git thinks of its data more like a series of snapshots of a miniature filesystem.

With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.

To be efficient, if files have not changed, Git doesnâ€™t store the file again, just a link to the previous identical file it has already stored.

Git thinks about its data more like a stream of snapshots.

- Git Has Integrity

Everything in Git is checksummed before it is stored and is then referred to by that checksum.

This means itâ€™s impossible to change the contents of any file or directory without Git knowing about it.

This functionality is built into Git at the lowest levels and is integral to its philosophy.

You canâ€™t lose information in transit or get file corruption without Git being able to detect it.

The mechanism that Git uses for this checksumming is called a SHA-1 hash.

This is a 40-character string composed of hexadecimal characters (0â€“9 and aâ€“f) and calculated based on the contents of a file or directory structure in Git.

A SHA-1 hash looks something like this:

24b9da6552252987aa493b52f8696cd6d3b00373

input "Akosua" = 24b9da6552252987aa493b52f8696cd6d3b00373 diff

You will see these hash values all over the place in Git because it uses them so much. In fact, Git stores everything in its database not by file name but by the hash value of its contents.

- Git Generally Only Adds Data

When you do actions in Git, nearly all of them only add data to the Git database.

It is hard to get the system to do anything that is not undoable or to make it erase data in any way.

As with any VCS, you can lose or mess up changes you havenâ€™t committed yet, but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

- The Three States

Pay attention nowâ€‰â€”â€‰here is the main thing to remember about Git if you want the rest of your learning process to go smoothly.

Git has three main states that your files can reside in: modified, staged, and committed:

- Modified:

means that you have changed the file but have not committed it to your database yet.

- Staged:

means that you have marked a modified file in its current version to go into your next commit snapshot.

- Committed:

means that the data is safely stored in your local database.

This leads us to the three main sections of a Git project:

- the working tree, the staging area, and the Git directory.

- The working tree:

is a single checkout of one version of the project.

These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.

- The staging area:

is a file, generally contained in your Git directory, that stores information about what will go into your next commit.

Its technical name in Git parlance is the â€œindexâ€, but the phrase â€œstaging areaâ€ works just as well.

- The Git directory:

is where Git stores the metadata and object database for your project.

This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

- The basic Git workflow goes something like this:

You modify files in your working tree.

You selectively stage just those changes you want to be part of your next commit,

which adds only those changes to the staging area.

You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

If a particular version of a file is in the Git directory, itâ€™s considered committed.

If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

- 1. Uses of Git

- Git is used to tracking changes in the source code.

- Distributed version control tool used for source code management.

- Allows multiple developers to work together.

- Supports non-linear development because of its thousands of parallel branches.

- 2. Features of Git

- Free and open-source

- Tracks history

- Supports non-linear development

- Creates backup

- Scalable

- Supports collaboration

- Branching is easier

- Distributed development.

- What is Github?

- It is a cloud/web-based platform for version control and collaboration.

Other types of cloud/web-based VCS

- GitLab

- BitBucket

- AWS Codecommit

- Azure DevOps

- Subversion

- Source forge

Installing Git:

- Linux (Ubuntu):

sudo apt-get install git # OR

sudo apt install git-all

- Git Bash terminal on Windows already comes pre-installed with Git

Setting up Git:

- Types of repositories

- Local

- Remote

# Setting up Local Repo:

- Create a directory and cd into it e.g., mkdir git\_practice && cd git\_practice

- git init # initiliaze the directory as a git local repo

- git config --global user.name "Your Name" # set up user credentials

- git config --global user.email "your\_email@whatever.com"

- ls -a # .git directory confirms directory is now a git repo

# Your Local repo has three (3) areas

i. Working area

If you create a file locally,

it stays in your working area untracked until staged.

To stage a file, run:

git add . # stages all files in the current directory

git add -A # stages all files in the current directory

git add <file\_name> # stages a specific file

git rm --cached <file> # remove staged file

ii. Staging area

After running git add command,

the file is now tracked and in your staging area.

To commit your file, run:

git commit -m "<message>"

file1.txt, file2.txt, file3.txt, file4.txt

git add file1.txt

git add file2.txt

git commit -m "commit\_message"

git add file3.txt

git add file4.txt

git commit -m "commit\_message"

cd -

# go back to previous directory

iii. Commit area

Your commit area hosts all committed files.

To see your commits, run:

git log # this shows the commit hash (SHA-1), author, date, etc.

git log --oneline # shortened version of the log output

To undo commits, use:

git revert <SHA\_of\_file>

iv. Add remote repo

- git remote add <name> <url> # attach a local directory to a Github repo

- git clone <url> # make a copy of a remote repo locally

- Your default branch name

By default Git will create a branch called master when you create a new repository with git init.

From Git version 2.28 onwards, you can set a different name for the initial branch.

To set main as the default branch name do:

git config --global init.defaultBranch main

- Checking Your Settings

If you want to check your configuration settings, you can use the

git config --list command to list all the settings Git can find at that point:

git config --list

- Getting Help

If you ever need help while using Git,

there are three equivalent ways to get the comprehensive manual page (manpage) help for any of the Git commands:

git help <command>

git <command> --help

man git-<command>

In addition, if you donâ€™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 â€œhelpâ€ output with the -h option, as in:

Syntax: git <command> -h

git add -h

Working in Git:

- To know the remote repo you are connected to, use:

git remote

- To push your local commits to remote, run:

git push <remote\_repo> <local\_branch\_name>

- Creating and using Git branches

A Git branch represents an independent line of development within which changes can be made.

The most common reason to work with multiple branches simultaneously is so that you can try out some experimental ideas without impacting other parts of your project.

git branch <branch\_name> # creates a branch

git branch # shows list of branches on the repo

git checkout <branch\_name> # switches to the specified branch

git checkout -b <branch\_name> # creates a branch and switces to it

git branch -a # list all local and remote branches

- Cloning and Forking remote repos

To clone a repo is to copy it from the remote to your local

git clone <url>

- Forking a repo makes it your own in Github

NB: You can only fork public repos

- Git Fetch, Merge and Pull

- Git Fetch

This downloads changes in the remote repo to your local machine

git remote -v

git fetch <remote\_repo>

# After fetching, compare the target branch with your local

- Git Merge

Merging means combining remote with local

git merge <origin/branch>

# This dowloads remote branch to your local

Learn more here: https://www.atlassian.com/git/tutorials/merging-vs-rebasing

- Git Pull

Combines both fetch and merge as one operation

git pull origin <main/branch>

Learn more here: https://www.atlassian.com/git/tutorials/syncing

- Git Rebase

When merging two or more branches that have diverged significantly,

we may want to replay some of our earlier work onto the current branch.

Instead of creating a merge commit, Git allows us to rebase instead!

Learn more here: https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase

- Git Reset and Revert

Learn more here: https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting

- Git Stashing

git stash

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Some Basic Git commands:

- How to check your Git configuration:

The command below returns a list of information about your git configuration including user name and email:

git config -l

- How to cache your login credentials in Git:

You can store login credentials in the cache so you don't have to type them in each time.

Just use this command:

git config --global credential.helper cache

- How to add a file to the staging area in Git:

The command below will add a file to the staging area.

Just replace filename\_here with the name of the file you want to add to the staging area.

git add <filename\_here>

- How to add all files in the staging area in Git

If you want to add all files in your project to the staging area,

you can use a wildcard . and every file will be added for you.

git add .

- How to add only certain files to the staging area in Git

With the asterisk in the command below,

you can add all files starting with 'fil' in the staging area.

git add fil\*

- How to check a repository's status in Git:

This command will show the status of the current repository including staged, unstaged, and untracked files.

git status

- How to commit changes in the editor in Git:

This command will open a text editor in the terminal where you can write a full commit message.

A commit message is made up of a short summary of changes,

an empty line, and a full description of the changes after it.

git commit

- How to commit changes with a message in Git:

You can add a commit message without opening the editor.

This command lets you only specify a short summary for your commit message.

git commit -m "your commit message here"

- How to commit changes (and skip the staging area) in Git:

You can add and commit tracked files with a single command by using the -a and -m options.

git commit -a -m "your commit message here"

- How to see your commit history in Git:

This command shows the commit history for the current repository:

git log

- How to see your commit history including changes in Git:

This command shows the commit's history including all files and their changes:

git log -p

- How to see a specific commit in Git:

This command shows a specific commit.

Replace commit-id with the id of the commit that you find in the commit log after the word commit.

git show <commit-id>

- How to see log stats in Git:

This command will cause the Git log to show some statistics about the changes in each commit,

including line(s) changed and file names.

git log --stat

- How to see changes made before committing them using "diff" in Git:

You can pass a file as a parameter to only see changes on a specific file.

git diff shows only unstaged changes by default.

We can call diff with the --staged flag to see any staged changes.

git diff

git diff <file\_name>

git diff --staged

- How to see changes using "git add -p":

This command opens a prompt and asks if you want to stage changes or not,

and includes other options.

git add -p

- How to remove tracked files from the current working tree in Git:

This command expects a commit message to explain why the file was deleted.

git rm <file\_name>

- How to rename files in Git:

This command stages the changes, then it expects a commit message.

git mv <oldfile> <newfile>

- How to revert unstaged changes in Git:

git checkout <file\_name>

- How to revert staged changes in Git:

You can use the -p option flag to specify the changes you want to reset.

git reset HEAD <filename>

git reset HEAD -p

- How to rollback the last commit in Git:

git revert will create a new commit that is the opposite of everything in the given commit.

We can revert the latest commit by using the head alias like this:

git revert HEAD

- How to rollback an old commit in Git:

You can revert an old commit using its commit id.

This opens the editor so you can add a commit message.

git revert <commit\_id\_here>

- How to create a new branch in Git:

By default, you have one branch, the main branch.

With this command, you can create a new branch.

Git won't switch to it automatically â€“ you will need to do it manually with the next command.

git branch <branch\_name>

- How to switch to a newly created branch in Git:

When you want to use a different or a newly created branch you can use this command:

git checkout <branch\_name>

- How to list branches in Git:

You can view all created branches using the git branch command.

It will show a list of all branches and

mark the current branch with an asterisk and highlight it in green.

git branch

- How to create a branch in Git and switch to it immediately:

In a single command, you can create and switch to a new branch right away.

git checkout -b <branch\_name>

- How to delete a branch in Git:

When you are done working with a branch and have merged it,

you can delete it using the command below:

git branch -d <branch\_name>

- How to merge two branches in Git:

To merge the history of the branch you are currently in with the branch\_name,

you will need to use the command below:

git merge <branch\_name>

- How to show the commit log as a graph in Git:

We can use --graph to get the commit log to show as a graph.

Also, --oneline will limit commit messages to a single line.

git log --graph --oneline

- How to show the commit log as a graph of all branches in Git:

Does the same as the command above, but for all branches.

git log --graph --oneline --all

- How to abort a conflicting merge in Git:

If you want to throw a merge away and start over,

you can run the following command:

git merge --abort

- How to add a remote repository in Git

This command adds a remote repository to your local repository

(just replace https://repo\_here with your remote repo URL).

git add remote https://repo\_here

- How to see remote URLs in Git:

You can see all remote repositories for your local repository with this command:

git remote -v

- How to get more info about a remote repo in Git:

Just replace origin with the name of the remote obtained by

running the git remote -v command.

git remote show <origin>

- How to push changes to a remote repo in Git:

When all your work is ready to be saved on a remote repository,

you can push all changes using the command below:

git push

- How to pull changes from a remote repo in Git:

If other team members are working on your repository,

you can retrieve the latest changes made to the remote repository with the command below:

git pull

- How to check remote branches that Git is tracking:

This command shows the name of all remote branches that Git is tracking for the current repository:

git branch -r

- How to fetch remote repo changes in Git:

This command will download the changes from a remote repo

but will not perform a merge on your local branch (as git pull does that instead).

git fetch

- How to check the current commits log of a remote repo in Git

Commit after commit, Git builds up a log.

You can find out the remote repository log by using this command:

git log origin/main

- How to merge a remote repo with your local repo in Git:

If the remote repository has changes you want to merge with your local, then this command will do that for you:

git merge origin/main

- How to get the contents of remote branches in Git without automatically merging:

This lets you update the remote without merging any content into the local branches.

You can call git merge or git checkout to do the merge.

git remote update

- How to push a new branch to a remote repo in Git:

If you want to push a branch to a remote repository you can use the command below.

Just remember to add -u to create the branch upstream:

git push -u origin <branch\_name>

- How to remove a remote branch in Git:

If you no longer need a remote branch you can remove it using the command below:

git push --delete origin branch\_name\_here

- How to use Git rebase:

You can transfer completed work from one branch to another using git rebase.

git rebase <branch\_name\_here>

Git Rebase can get really messy if you don't do it properly.

Read more on rebase here: https://git-scm.com/book/it/v2/Git-Branching-Rebasing

- How to run rebase interactively in Git:

You can run git rebase interactively using the -i flag.

It will open the editor and present a set of commands you can use.

git rebase -i master

# p, pick = use commit

# r, reword = use commit, but edit the commit message

# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

# f, fixup = like "squash", but discard this commit's log message

# x, exec = run command (the rest of the line) using shell

# d, drop = remove commit

- How to force a push request in Git:

This command will force a push request.

This is usually fine for pull request branches because nobody else should have cloned them.

But this isn't something that you want to do with public repos.

git push -f

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Install Oh My Zsh (zsh shell):

Use any of the following links in your console

1. sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"

2. sh -c "$(wget https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh -O -)"

3. You can equally use the below steps:

- sudo apt update && sudo apt upgrade -y

- sudo apt install zsh -y # installs oh-my-zsh shell

- zsh --version # checks the zsh shell version installed

- zsh # starts the zsh shell

- echo $SHELL # checks the default shell

- chsh -s $(which zsh) # changes default shell to zsh

Extra Settings on zsh:

Reference: https://phoenixnap.com/kb/install-zsh-ubuntu

- Enable Auto Suggestions:

1. Add the auto-suggestion plugin repository first.

Run the following command to clone the repository:

- git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH\_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions

2. Open the .zshrc configuration file using a text editor:

- vi ~/.zshrc

3. Scroll down to the plugins section of the file and activate the auto-suggestion plugin by adding it to the plugins, as shown below:

- plugins=(git zsh-autosuggestions)

4. Save the changes and exit the text editor.

5. Restart the terminal and start Zsh to enable the auto-suggestions plugin.

When you start typing a command, use the right arrow key to accept the suggestion and automatically complete the command

- Enable Syntax Highlighting:

Syntax highlighting provides numerous benefits, including improved code readability, error detection, language recognition, productivity, and aesthetics.

Follow the steps below to add syntax highlighting to Zsh:

1. Clone the plugin repository by running the following command:

- git clone https://github.com/zsh-users/zsh-syntax-highlighting.git ${ZSH\_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-syntax-highlighting

2. Open the .zshrc file:

- vi ~/.zshrc

3. Scroll down to the plugins section and add zsh-syntax-highlighting to the list of plugins:

- plugins=(git zsh-autosuggestions zsh-syntax-highlighting)

4. Save the changes and exit the file.

5. Restart the terminal and start Zsh to enable syntax highlighting.

For example, typing a command with a typo makes it red,

while typing a command correctly highlights it green.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Since Git might read the same configuration variable value from more than one file, itâ€™s possible that you have an unexpected value for one of these values and you donâ€™t know why.

In cases like that, you can query Git as to the origin for that value, and it will tell you which configuration file had the final say in setting that value:

git config --show-origin rerere.autoUpdate