Git And Github Tutorials

- Git and Github are not same.
- Git is the software which is used for version control.
- Github is the website or the place where the git repository is uploaded/hosted by users for version control.
- Some of the Github alternavites are Bitbucket and Gitlab.

Git

Git is a version control system.

- Use cases of Git
    - Easily recover files.
    - Who introduced an issue and when?
    - Roll back to previously working state.

History of Version Control System (VCS)

- Local VCS
  Local database to keep track of files.
    - Pros - Can track files and rollback
    - Cons - If you lose your harddisk everything is lost.

- Centralized VCS
  A centralized server to store the files.
  Files can be accessed using pull operation and stored using push operation.
    - Pros - Data is centralized and anyone can access at anytime and anywhere.
    - Cons - Cannot rollback if the server is damaged.

- Distributed VCS
  Like centralized VCS with few modifications.
  Complete full backup of project with each user working on the project.

How was git created?

➔ Created by Linux

Features of Git

- Capture spanshot and not difference.
- Almost every operation is local.
- Git has integrity.
- Git generally only adds data.

Git Installation

- Git command line tool
- Git bash (terminal program)
- Git uses linux paths and commands

Commands (Linux)

- pwd - present working directory
- ls - List content in current directory.
- cd - Change directory.
- touch filename.txt - Creates empty file with the given file extension.
- git config –global user.name “name” - configure user name.
- git config –global user.email “email” - configure user email.
- git config –list - list all the configured details.

Git - Three stage architecture

Commit means snapshot/version of the project.
Stage means sending or uploading files in next commit.

- Working directory - The current folder you are in.
- Staging area - All files and directories are ready for commit or next commit.
- Git directory (repository) - All compressed files and folders are stored in this directory(.git directory).

Git Commands

- **git init** - Initialize git repository.
- **git status** - Check if the current repository is git repository or not.
- **git add --a** - Send all files in staging area.
- **git commit -m “message”** - Commit files with a message.
- **git log** - Checking previous commits history.
- **git add filename** - Send that perticular file in staging area.
- **rm -rf .git** - Deletes git repository.[Critical command]
- **git clone github_file_url** - Clone github repository.
- **git clone github_file_url filename** - Clone github repository updating repository name to the specified filename/ repository name.
- **touch .gitignore** - Creates a new empty .gitignore file which contains all the file names that are to be ignored by git.
- **git add .** - Sends .gitignore file and all other files to staging area.
- **git diff** - Compares the working directory and staging area.
- **git diff --staged** - Compares previous commit with the staging area.
- **git commit -a -m “message”** - Direct commit without sending files into staging area.
- **git rm filename** - Remove or delete a file in git.
- **git mv filename renamed_filename** - Rename a file in git.

- **git rm --cached filename** - Untrack file from git tracking.
- **git log -p** - Shows log history of previous commits with what was removed and what was added with that commit.
- **git log -p -3** - Show log history of previous 3 commits along with the changes made. At the place of 3 we can have n values.
- **git log --stat** - Shows stats of each of the previous commits in short.
- **git log --pretty=oneline** - Show all the commit history in one line each.
- **git log --pretty=short** - Show commit history in short with author details and message.
- **git log --pretty=full** - Show commit history with full details with author and commiter details and message.
- **git log --since=2.days** - Show all the log history of 2 days. We can change the days to weeks, months, years, hours, minutes, etc.
- **git log --pretty=format:"%h –%an"** - Shows log history by formatting data by abbreviated hash code and author name.
- **git commit --amend** - Ammend current commit changes to the previous commit and is used to change the message of the previous commit if necessary.
- **git restore --staged filename** - Unstage file with the given filename.
- **git checkout -- filename** - Restore/Unmodify the file with filename to the previous commit.
- **git checkout -f** - Restores/Unmodifies all the changed or modified files to the previous commit.

File Status Lifecycle

Git repository is the repository which has .git folder. All the folders and files that are in the repository/folder that contains .git folder will be tracked.

- Untracked
    - Empty git repository with all existing untracked files or files removed from staging area.

- Unmodified
    - Files that were untracked are being tracked using git add –a command.
    - Already commited files are also are in unmodified stage.

- Modified
    - If any file content is changed or modifed that is being tracked it is unmodified as well as modified.
    - If the file is already staged and then modifed or changed it just modified file.

- Staged
    - All the modified files that are to be sent in the next commit are added in staged area using git add –a or git add filename commands.
    - After commit the files status will be changed back to unmodified.

.gitignore

- Any file names that are in are in .gitignore file are ignored by git even if they are constantly modified or changed.
- If there are multile files using same extention that are to be ignored by git just add *.file-extension in .gitignore file. [Example: For multile log files add *.log in .gitignore file]
- Filename.txt - Ignores only the file with the perticular name.
- *.txt - Ignores all file with .txt.
- /dir/ - Ignores an outer directory namd dir.
- dir/ - Ignores all the directories named dir
- You can ignore the inside directories by specifying the path.[example: static/dir]

Skipping staging area

- You can directly commit by skipping staging area but only the already tracked files will be commited.
- The new files or untracked files won't be committed.You cannot commit this files directly. You have to add these files specificly in staging area and then commit.

Working with remote repository

Most popular git hosting is Github.

Create an account on github and create a new public repository.

Pull means getting the code from remote repository to our computer.
Push means sending/ uploading our code from computer to a remote repository.

- **Setup**
  - You can setup a git repository using HTTPS or SSH.
  - HTTPS asks you to login during the setup process using a code or login credentials for remote access.
  - SSH asks you to generate an SSH key and set it in remote repository for access.

- **Creating new repository in command line**

```
echo "# Test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:LelwynVaz/Test.git
```

```
git push -u origin main
```

- **Pushing existing repository from command line**

```
git remote add origin git@github.com:LelwynVaz/Test.git
git branch -M main
git push -u origin main
```

- **Generating a new SSH key**
  Follow url:
  https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

  Follow step by step tutorial from above url.

- **Important commands while working with remotes**

  - git remote - To know or check if there is any remote repository active currently.
  - git remote add origin git@github.com:LelwynVaz/Test.git - Adds the remote git repository with a name origin.
  - git remote -v - Lists the push and pull(fetch) urls of currently active remote repository.
  - tail ~/.ssh/id_ed25519.pub - Displays the generated ssh key.
  - git remote remove origin - Removes origin.

Git alias command

Using git alias command you can make a longer or difficult command shorter by introducing your own command.

For example the default command for unstaging a file is **git restore --staged filename.** You can change it to something shorter or easy to remember using alias as shown below.

git config --global alias.unstage 'restore --staged –'

The above command changes the original unstage command to **git ustage filename**.

Both original and new custom command will perform the same operation.

To create alias command **git config --global alias.custom name 'original command'** is the syntax.

This command is used when you have to type a long command multiple times to make your task easier by writing a short command for the same.

More alias examples

- **git config --global alias.st status** - Changes **git status** command to ***git st***
- **git config --global alias.last 'log -p -1** - Changes **git log -p -1** command to ***git last***

Vim Editor

To use vim editor in git bash use the following commands. Vim is a text editor.

vi filename.txt - Opens file with filename in vim editor.

When the file is opened press **i key** on keyboard to insert any data into the file.

After writing into the file, you can save and exit vim editor by pressing **esc key** on keyboard and then press type **:wq** and press enter.

More vim commands
:x - To exit out of the editor.

cat filename.txt - Displays the text data inside the file.

Git Branches

Branches are the different versions of the main or master branch which is a stable version of the project.

A project may have multiple n number of branches which could be stable or unstable. When they are ready and tested they can be merged into the main branch.

Each branch of main can also have multiple sub branches of itself.

Each branch will have their own code and changes within it which will not harm or change the parent branch unless they are merged together.

- **Commands used in git branching**

    - **git checkout -b branchname** - Creates a new branch with branchname and take you inside it.
    - **git checkout branchname** - Switches to the specified branch if present.
    - **git branch** - Lists all the branches with green star on the branch in which you are currently in.
    - **git merge branchname** - Merges the specified branch with the branch you are currently in.

Merge conflicts

Merge conflicts appear when you changed the files base branch and the same changes are also made in the other branch which is to be merged.

You can resolve such merge conflicts by selecting which changes are to be made in the base branch - original changes, incomming changes, both changes or comparison.

<<<< HEAD (current change)
code line1
code line 2
==========
code line 1
code line 2
>>>>>>> branchname (incoming change

The above illustration is called conflict resolution markers. The show they conflicting changes in the editor (VS Code).

After selecting which changes are to introduced resolve the conflict using **git add filename** or **git add .** command.

The commit after merging branch will store a pointer to previous commit of base branch as well as to last commit of incoming branch.

Branch Management

Git stores only changed data with a pointer associated to it when a branch is created. Git doesn't stores the whole project copy separately for new branch.

- **Branch management commands**
    - **git branch -v** - Shows last commit of each brach with brancg name, commit hash, and commit message.
    - **git branch --merged** - Already merged branches.
    - **git branch --no-merged** - Not already merged branches.
    - **git branch -d branchname** - Deletes the specified branch. If the branch is not merged it gives error.
    - **git branch -D branchname** - Deletes any branch even if it is not merged without any error.
    - **git push -d origin branchname** - Deletes remote branch.

Branching Workflow in Production

There are basically two types to branches. Long Running Branches and Topic Branches.

Long Running Branches are the branches that will be running for a long time like the master branch, development branch, etc.. These brances mainly contains the stable version of the code or project.

Topic Branches are the branches that are used for a short period of time to add specific feature, fixes, or topics to the project. These branches are eventually merged back into the long running branches.