Name Inayat Yousuf.
Task: GIT.

What is Git:
Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is a popular version control system. It was created by Linus Torvalds in 2005.

It is used for:

- Tracking code changes
- Tracking who made changes
- Coding collaboration

What does Git do?

- Manage projects with **Repositories**
- **Clone** a project to work on a local copy
- Control and track changes with **Staging** and **Committing**
- **Branch** and **Merge** to allow for work on different parts and versions of a project
- **Pull** the latest version of the project to a local copy
- **Push** local updates to the main project.

Git Workflow:

A Git project can be thought of as having three parts:

1. A *Working Directory*: where you'll be doing all the work: creating, editing, deleting, and organizing files
2. A *Staging Area*: where you'll list changes you make to the working directory
3. A *Repository*: where Git permanently stores those changes as different *versions* of the project

The Git workflow consists of editing files in the working directory, adding files to the staging area, and saving changes to a Git repository. In Git, we save changes with a *commit*.

Configure Git

For the below commands we must have installed git, first of all check git is installed or not. git –version to check your git version.

git config --global user.name "yourUserName" like "inayatyousf"

git config --global user.email "yourEmailId" like "abc778@gmail.com"

Change the user name and e-mail address to your own.

Note: Use global to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global

Create a folder mkdir testGit

cd testGit [and inside this folder create a file. For testing purposes like touch test.txt] then.

git init [You just created your first Git Repository!]

Note: Git now knows that it should watch the folder you initiated it on. Git creates a hidden folder to keep track of changes.

git status: we check the Git status and see if it is a part of our repo:

```
git status

On branch master

No commits yet

Untracked files:

  (use "git add ..." to include in what will be committed)

    test.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Now Git is **aware** of the file, but has not **added** it to our repository!

Files in your Git repository folder can be in one of 2 states:

- Tracked - files that Git knows about and are added to the repository
- Untracked - files that are in your working directory, but not added to the repository

When we first add files to an empty repository, they are all untracked. To get Git to track them, we need to stage them or add them to the staging environment.

As you are working, you may be adding, editing, and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

Staged files are files that are ready to be **committed** to the repository you are working on. For now, we are done working with **test.txt** So we can add it to the Staging Environment:

Example

git add test.txt

Note: But if you have 100 files and want to add then in staged area then no need to add then one by one use **git add -A**

The above file should be **Staged**. Let's check the status::

git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached ..." to unstage)

new file: test.txt

In the above git status, the file was untracked. Now it is under the track of git.

Now the file has been added to the Staging Environment.

Git Add More than One File

You can also stage more than one file at a time.

git add --all or shorthand for this is — git add -A
Using --all instead of individual filenames will stage all changes (new, modified, and deleted) files.

Git Commit without Stage

Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The **-a** option will automatically stage every changed, already tracked file.

git commit -a -m "Updated test.txt with a new line"

Warning: Skipping the Staging Environment is not generally recommended.

Skipping the stage step can sometimes make you include unwanted changes.

Check History
To view the history of commits for a repository, you can use the log command:
git log

Git -help See Options for a Specific Command

Any time you need some help remembering the specific option for a command, you can use git *command* -help:

Syntax – **git command -help** eg **git commit -help** or **git add -help.**

we can also use **--help i**nstead of **-help** to open the relevant Git manual page.

To list all possible commands, use the help --all command: eg
git help –all

Git Branch

In Git, a **branch** is a new/separate version of the main repository. Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project. You can even switch between branches and work on different projects without them interfering with each other.

New Git Branch
Let's add some new features to our test.txt page. We are working in our local repository, and we do not want to disturb the main project.
So we create a new branch: **git branch add-some-info**

add-some-info is now our new branch in which we need to add some more information in our file without affecting the main branch or main project.

Let's confirm that we have created a new branch: **git branch**

We can see the new branch with the name "add-some-info", but the * beside master specifies that we are currently on that branch.

checkout is the command used to check out a branch. Moving us from the current branch to the one specified at the end of the command:

Eg **git checkout add-some-info or git checkout -b add-some-info**

Note: Using the -b option on checkout will create a new branch, and move to it, if it does not exist.

Now we have moved our current workspace from the master branch to the new branch

Open your editor and make some changes. After completing the changes then check status

git status

- There are changes to our file but the file is not staged for commit
- file not tracked

So we need to add both files to the Staging Environment for this branch:

git add --all

Using **--all** instead of individual filenames will Stage all changed (new, modified, and deleted) files.

Check the status of the branch:

Changes happen. So we will commit them to the branch:

git commit -m "Added some more info into this file"

Now we have a new branch, that is different from the master branch.

Now switch to the master branch:

git checkout master

In this branch, the changes are not reflected. We have made changes in the file, and we need to get those changes to the master branch. For that, we need to merge the two branches.

We have the **add-some-info branch,** and so let's merge the master and **add-some-info** branches.

First, we need to change to the master branch: **git checkout master**

Then **git merge add-some-info**

Create a Repository on GitHub

Now that you have made a GitHub account, sign in, and create a new Repo:

Copy the URL, Now paste it the following command:

git remote add origin yourURL

git remote add origin *URL* specifies that you are adding a remote repository, with the specified URL, as an origin to your local Git repo.

Now we are going to push our master branch to the origin url:

git push origin master – then it will ask you for username and password.

git rm is used to remove file from working directory as well as from the staging area.

git rm test.txt

git rm -cached filename it is used to to remove file from only staging area not from working directory

SETUP

Configuring user information used across all local repositories

git config --global user.name “[firstname lastname]”

set a name that is identifiable for credit when review version history

git config --global user.email “[valid-email]”

set an email address that will be associated with each history marker

git config --global color.ui auto

set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

git init

initialize an existing directory as a Git repository

git clone [url]

retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

git status

show modified files in working directory, staged for your next commit

git add [file]

add a file as it looks now to your next commit (stage)

git reset [file]

unstage a file while retaining the changes in working directory

git diff

diff of what is changed but not staged

git diff --staged

diff of what is staged but not yet committed

git commit -m “[descriptive message]”

commit your staged content as a new commit snapshot.

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

git branch

list your branches. a * will appear next to the currently active branch

git branch [branch-name]

create a new branch at the current commit.

git checkout

switch to another branch and check it out into your working directory

git merge [branch]

merge the specified branch's history into the current one

git log

show all commits in the current branch's history.

TRACKING PATH CHANGES

Versioning file removes and path changes

git rm [file]

delete the file from project and stage the removal for commit

git mv [existing-path] [new-path]
change an existing file path and stage the move

IGNORING PATTERNS
Preventing unintentional staging or commiting of files
logs/
*.notes
pattern*/
Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

SHARE & UPDATE
Retrieving updates from another repository and updating local repos
git remote add [alias] [url]
add a git URL as an alias
git fetch [alias]
fetch down all the branches from that Git remote

git merge [alias]/[branch]
merge a remote branch into your current branch to bring it up to date

git push [alias] [branch] eg git push origin master
Transmit local branch commits to the remote repository branch

git pull
fetch and merge any commits from the tracking remote branch