--- Configure Git ---

git config --global user.name "<First Last>"

git config --global user.email "<valid-email>"

git config --global init.defaultBranch main

--- Unix Commands ---  
mkdir <FileName>

cd <path>

ls

ls -a

List all files including hidden files to

--- Initialize Repository ---

cd d:/x  
git init (to create these files as repository)  
  
##Query Solution##  
----------------------------------------------------------  
Git will only track files that you explicitly stage using git add "filename". Untracked files remain untracked until added. Once you initialize the folder with git init, you can apply all Git functionalities to that folder and its contents.

--- Check Repo Status ---

git status

git status -s

--- Stage or Unstages a File ---

git add "<file\_name>" (to track)  
git rm --cached "<file\_name>" (to untrack a particular file)

--- Ignore Files ---

First, create a text document with extension .gitignore.  
Inside that file:  
# ignore all .txt files  
\*.txt  
  
For more details, visit:  
<https://github.com/github/gitignore>  
  
##Query Solution##  
---------------------------------------------------------  
.gitignore itself is not ignored, even though its purpose is to specify files and directories that Git should not track. If .gitignore is present in your repository, it will be treated as a regular tracked file by default.

--- Stage or UnStage All Files ---

git add --all  
or   
git add . (track all files except that ignore file)  
git rm -r --cached . (to untrack/unstage all files)

--- Commit ---

git commit -m "Descriptive commit message"

Creates a commit with a message attached

git commit -m "Descriptive commit message" --amend

Update previous commit instead of creating new one

--- Edit a Commit ---

git add --all  
git commit -m "message of already created commit" --amend

git commit

--- View Changes Before Committing ---  
git diff  
git diff -s

--- Git Environments ---

1. Working Directory: Files you're currently working on.

2. Staging Area: Files ready to be committed.

3. Repository: Files that have been committed.

---diff between git rm --cached and git reset---

git rm --cached "<file\_name>":  
Removes the file from the staging area without deleting it from the working directory.  
  
git reset "<file\_name>":  
Removes the file from the staging area and reverts it to its state in the last commit (if modified).  
  
To reset the working area too:  
git checkout -- "<file\_name>"  
git checkout -- .

--- File Reset Commands ---

git rm --cached "<file\_name>"

Removes a file from staging but keeps it in the working directory.

git reset "<file\_name>"

Removes a file from staging and reverts it to the last committed state.

git checkout -- "<file\_name>"

Resets a file in the working directory to match the last committed version.

---Git Restore ---

git restore --staged "<file\_name>":  
It unstages changes for the specified file, moving it back to the working directory. The file will no longer be included in the next commit, but its modifications will still exist in your working directory.

Use case:

Use this when you accidentally staged changes (e.g., added or deleted a file) and want to remove the file from the staging area without losing your local changes.  
  
git restore "<file\_name>":  
It discards the uncommitted changes in the working directory for the specified file, restoring it to the version from the last commit.Use case:

Use this when you want to undo all modifications made to a file and revert it to its last committed state.

##Query Solution##

---------------------------------------------------------

if I were to commit now this file would not be included because this file is currently in the working environment so let's say that maybe you're working on six files and let's say that three of them you're ready to commit you can move those three to staging and then you could commit them and the files that are current in the working files those would not be included the index.htm file is once again in my working files

--- Bypass Staging Area ---

git commit -a -m "any message"

Stage and commit changes in one step.

--- Remove and Delete File ---

git rm "<file\_name>"  
or  
Go and delete from the folder.

--- Rename File ---

git mv "<old\_file\_name>" "<new\_file\_name>"

git commit -m "Renamed file"

--- View Commit History ---

git log  
git log --oneline  
git log --all --graph

--- View Commit Changes ---

git log -p

--- Reset to a Previous Commit ---

git reset <commit\_hash>

Moves branch pointer to a specific commit but does not change the working directory.

git reset --hard <commit\_hash>

Resets both the branch pointer and working directory to the specified commit.

git restore .

Ensures the working directory matches the specified commit.

--- Branch Management ---

# Create a new branch.

git branch <branch\_name>

# List all branches.

git branch

# Switch to another branch.

git switch <branch\_name>

--- Merge Branches ---

git merge <branch\_name> -m "Merge message"

--- Delete Branches ---

git branch -d <branch\_name>

--- Merge Conflicts ---

Merge conflicts occur if you try to merge a branch into main that has changed since the branch was created. Git will prompt you to resolve the conflicts manually before merging.

**////github////**

so far we've just been working with a git repository that's hosted on your local computer but you can also host your git repository in the cloud and one of the most popular Cloud repositories is called GitHub not only do they offer a cloud repository but you could also manage your project you could set up combon boards you could track issues you could assign features or bugs to different people in a sense it's a social coding website

wondering do I really need to create a cloud repo especially if you just plan on working by yourself well if you're planning on working with a team it makes sense to set up a cloud repo so you can all work together but even if you plan on just working by yourself by storing it in the cloud if something happens to your local computer you have a backup and all of your repo files are safe

**////git workflow////**

<https://github.com/firstcontributions/first-contributions>

**////rebase////**

Avoid editing commits that have already been pushed to a shared repository. If you change commits that have already been pushed, it can cause conflicts for other collaborators. If you must do this, you'll need to force-push (git push --force), which should be done with caution.

Conflicts may arise during the rebase, and Git will prompt you to resolve them before continuing.

this panel is used if i want to change a particular commit

or if i am facing a merge conflict.

**////git branch flow////**

Typical flow:  
1. Create a branch for a feature/bug fix.  
2. Make changes.  
3. Merge the branch back into main.  
4. Delete the branch.

--- Push code on github ---

First, organize your files locally:

# Create the new folder

mkdir 2024-programming-fundamentals

# Move all files (except .git folder)

move \* 2024-programming-fundamentals/

# Move hidden files like .gitignore

move .gitignore 2024-programming-fundamentals/

1. Set Up Git & GitHub

git config --global user.name "muneebimtiaz"

git config --global user.email "muneebimtiaz6@gmail.com"

2. Create a GitHub Repository

git init

3. Push Local Code to GitHub

git remote add origin <https://github.com/yourusername/2024-Learnings.git>

# Stage all files

git add .

# Commit with a message

git commit -m "Added JavaScript basics"

# Push to GitHub

git push -u origin main

# if error happens pull it   
git pull origin main --rebase

git push -u origin main

4. Backup/Restore Workflow

Open VS Code → Press Ctrl+Shift+P → Git: Clone.

Paste your repo URL → Choose a folder to save it.

Result: A full copy of your repo is downloaded.