**What is Git?**

Git was created by Linus Torvalds in 2005 for Linux kernel development. It’s a free, open-source tool that allows developers to:

* Track changes to files over time.
* Collaborate with others without overwriting each other’s work.
* Revert to previous versions of code.
* Work on multiple features or experiments simultaneously using branches.
* Unlike centralized version control systems (e.g., SVN)

**Why use Git?**

* Collaboration: Multiple developers can work on the same project simultaneously
* Versioning: Track and revert changes easily
* Branching: Experiment with new features without affecting the main code base
* Speed: Fast operations due to local repository management
* Reliability: Data integrity is ensured with cryptography hashing
* Flexibility: Supports various workflows (e.g., Git flow, trunk-based development)

**Key Features of Git?**

* Distributed: Each developer has a full local copy of the repository, enabling offline work and faster operations.
* Version Control: Tracks changes to files, allowing you to revert, compare, or merge versions.
* Branching and Merging: Supports lightweight branches for experimentation and seamless merging.
* Collaboration: Facilitates teamwork through remote repositories (e.g., GitHub, GitLab, Bitbucket).
* Open-Source: Free and widely used across industries.

What is GitHub?

* Git is not the same as GitHub
* GitHub makes tools that use Git.
* GitHub is the largest Host of source in the World and that has been owned by Microsoft Since. 2018.

**Configuring Git for the first time.**

$ git config --global user.name “<Enter your username here>”

$ git config --global user.email “<Enter your email here>”

**General Git Features.**

Initializing Git.

*$ git init*

Git now knows that it should watch the folder you initiated it

on. Git creates a hidden folder to keep track of changes.

Staging files/Adding files to Git repo :-

* Staged files are files that are ready to be committed to the
* repository you are working on.
* When you first add files to an empty repository, they are all
* UN tracked. To get Git to track them, you need to stage them, or add
* them to the staging environment.

$ git add <filename with extension>

Staging all files in a folder :-

$ git add --all

$ git add -A

**Making a Commit :-**

**Adding commits keep track of our progress and changes as we**

**work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.**

**When we commit, we should always include a message.**

**$ git commit -m “<Enter your message here>”**

**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.**

**$ git commit -a -m “<Enter your message here>”**

**Status of files and log :-**

**$ git status**

**File status in a more compact way :-**

**$ git status --short**

**Log of a file :-**

**Log is used to view the history of commits for a repo.**

**$ git log**

**$ git log –oneline**

**Git Help :-**

**If you are having trouble remembering commands or options**

**for commands, you can use Git help.**

**See all the available options for the specific command –**

**$ git <command> -help**

**See all possible commands –**

**$ git help –all**

**If you find yourself stuck in the list view, SHIFT + G to jump the**

**end of the list, then q to exit the view.**

**Git Branching :-**

**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.**

**We can even switch between branches and work on different**

**project without them interfering with each other.**

**Making a new Git Branch :-**

**$ git branch <name of branch>**

**$ git branch**

**Switching to other Branches :-**

**$ git checkout <branch name>**

**Making a new branch :-**

**$ git checkout -b <branch name>**

**Deleting a Branch :-**

**$ git branch -d <branch name>**

**Merging two Branches :-**

**$ git merge <branch name>**

**Working with git hub**  To create a Git repository, you need to first set up a directory for your project. Then, initialize the directory as a Git repository using the git Init command. Finally, you can add files to the repository, stage them, and commit them.

Here's a step-by-step guide:

* 1. Create a project directory:

Create a new folder to house your project files.

* 2. Initialize Git:

Open a terminal or command prompt and navigate to the project directory.

Run the command git init. This creates a hidden .git folder, which is the heart of your Git repository.

* 3. Add files:

Add your project files to the repository directory.

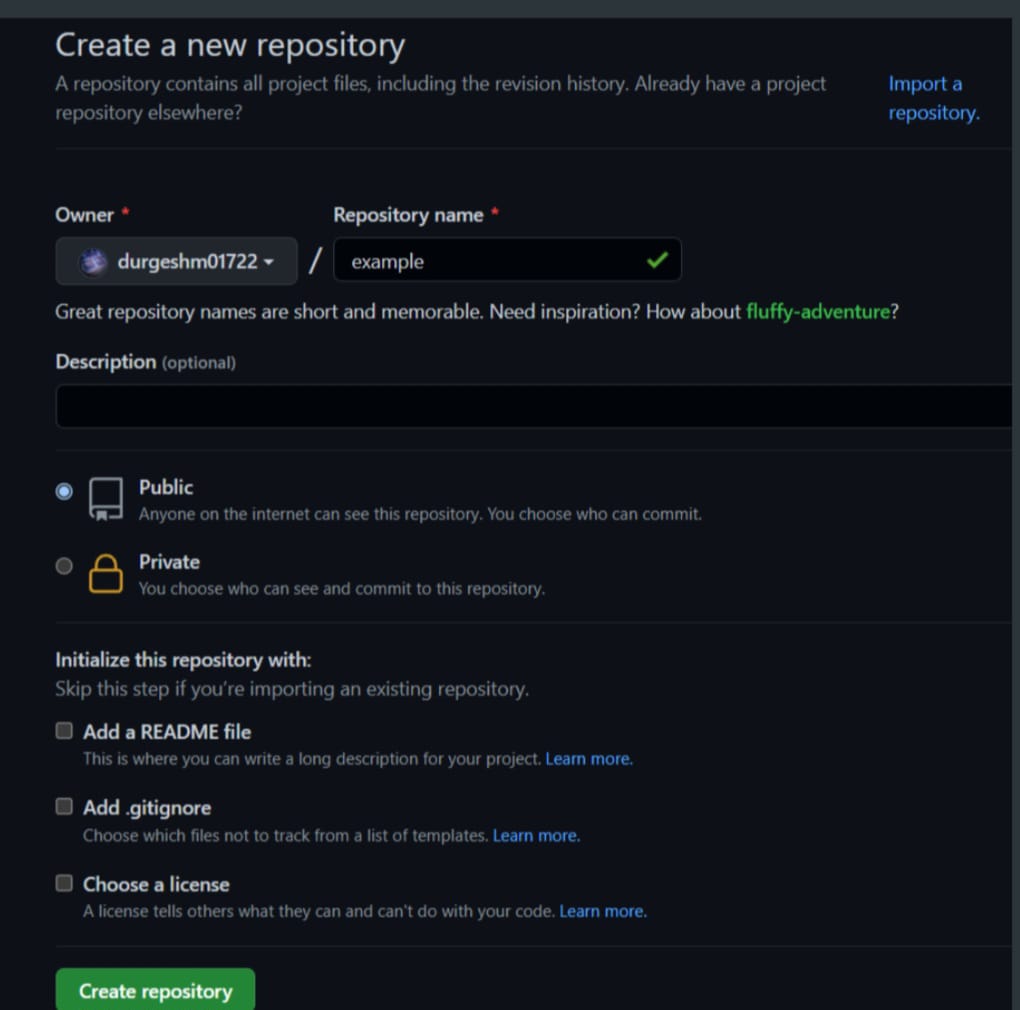
* 4. Stage and commit changes:

Stage the files you want to include in the first commit: git add . (or git add <file\_name>)

Create the first commit: git commit -m "Initial commit" (replace "Initial commit" with a descriptive message).

* 5. (Optional) Push to a remote repository:

If you want to share your repository, you can link it to a remote repository (like GitHub, Git Lab, or Bit bucket) by using git remote add origin <remote\_URL> and then git push -u origin main.



**Git Undo :-**

**Git Revert :-**

**‘revert’ is the command we use when we want to take a previous commit and add it as a new commit, keeping the log intact. First thing, we need to find the point we want to return to. To do that, we need to go**

**through the log. To avoid the very long log list, use the --one line option which gives just one line per commit showing –**

**i. The first seven characters of the commit hash**

**ii. The commit message**

**Git Revert HEAD :-**

**We revert the latest commit using ‘git revert HEAD’ (revert the latest**

**change, and then commit). By adding the option --no-edit, we can skip the**

**commit message editor (getting the default revert message).**

**$ git revert HEAD --no-edit**

**Git Revert to any commit :-**

**To revert to earlier commits, use ‘git revert HEAD~x’ (x being a**

**number. 1 going back one more, 2 going back two more, etc.)**

**Git Reset :-**

**‘reset’ is the command we use when we want to move the repository**

**back to a previous commit, discarding any changes made after that commit.**

**First, get the seven characters of the commit hash from the log for the**

**commit that you want to go back for. Then we reset our repository back to**

**that specific commit using ‘git reset commit hash’ (commit hash being the first**

**7 characters of the commit hash we found in the log).$ git reset <commithash>**