Getting Started with Git & GitHub

What is Git?

Git - Git is an open-source distributed version control system

It is used for:

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

Why Git?

- ✓ Developers can work together from anywhere in the world.
- ✓ Developers can see the full history of the project.
- ✓ Developers can revert to earlier versions of a project.

What is GitHub?

- ✓ Git is not the same as GitHub.
- ✓ GitHub makes tools that use Git.
- ✓ GitHub is the largest host of source code in the world.

Git Install

You can download Git for free from the following website: https://www.git-scm.com/

Using Git with Command Line

The first thing we need to do, is to check if Git is properly installed:

```
git –version
```

After installing Git, create a GitHub account and configure the Git as follows.

Configure Git

This is important for version control systems, as each Git commit uses this information:

```
git config --global user.name "username"
git config --global user.email "email"
git config --global init.defaultBranch main
```

Creating Git Folder

create a new folder for our project:

```
mkdir myproject #mkdir makes a new directory.
cd myproject #cd changes the current working directory.
```

Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

```
git init
```

Git Status

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

To check the status of our repository. we can use

git status

or use the --short option to see the changes in a more compact way:

git status --short

Short status flags are:

- ✓ ?? - Untracked files
- ✓ A - Files added to stage
- ✓ M - Modified files
- ✓ D - Deleted files

Git Staging

Staged files are files that are ready to be committed to the repository you are working on.

To make a file to staged, use

git add "file-name"

To add more than one, use

git add --all or ***git add -A*** or ***git add .***

Git Commit

Since we have finished our work, we are ready move from stage to commit for our repo.

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.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

git commit -m "message"

The commit command performs a commit, and the -m "message" adds a message.

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 "message" or ***git commit -am "message"***

Git Commit Log

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

git log

To get the history of each commit in a line without complete details, use:

git log --oneline

See All Possible Commands

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

git help -all

Working with Git Branches

In Git, a branch is a new/separate version of the main repository.

New Git Branch

To create a new branch

git branch "branch-name"

To check whether the branch is created or not, we use

git branch (* denotes the current working directory)

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:

git checkout "branch-name"

The following command, automatically creates a new branch, if that branch doesn't exist and then checkout to it.

git checkout -b "branch-name"

To delete a branch, use

git branch -d "branch-name"

Once you've created branches, you can switch between them to work on different features or fixes independently. Here are some more commands for managing branches:

Merge Branches

To integrate changes from one branch into another, you can merge them:

git checkout "branch-name" (to which branch you want to merge)
git merge "branch-name" (from which branch you want to merge)

Resolve Merge Conflicts

Sometimes, Git can't automatically merge branches due to conflicting changes. In such cases, you need to resolve conflicts manually:

- ✓ Git marks the conflicted areas in your files.
- ✓ You manually edit the files to resolve conflicts.
- ✓ After resolving conflicts, stage the changes and commit them.

Remote Branches

Branches can exist not only in your local repository but also in remote repositories like GitHub. To work with remote branches:

- ✓ List Remote Branches:
  git branch -r or ***git remote***
- ✓ Add a remote repository to your local Git repository:
  git remote add origin <url>
- ✓ Fetch Remote Branches:
  git fetch origin
- ✓ Create a Local Branch from a Remote Branch:
  git checkout -b local-branch-name origin/remote-branch-name
- ✓ Push a Local Branch to a Remote Repository:
  git push origin local-branch-name

Summary

Git is a version control system that allows developers to track changes, collaborate, and maintain a complete history of their work. GitHub complements Git by providing a platform for hosting repositories and collaborating with others. By following the steps outlined in this guide, you can effectively use Git to manage your projects and work efficiently with version control.