Ameya Kolhatkar

Introductory steps

1. Install Git on your computer: First, you'll need to install Git on your computer. You can download Git from the official website (https://git-scm.com/downloads).

2. Create a GitHub account: If you don't already have one, create a GitHub account (https://github.com/).

3. Create a new repository: Once you have a GitHub account, you can create a new repository by clicking the "New" button on the homepage or by navigating to https://github.com/new. Give your repository a name and a short description.

4. Clone the repository: To start working with your repository locally, you'll need to clone it to your computer. To do this, navigate to the repository on GitHub and click the green "Code" button. Then copy the URL of the repository. Open your command prompt/terminal on your computer and navigate to the directory where you want to clone the repository. Then run the command git clone [repository URL].

5. Make changes to the repository: Once you've cloned the repository, you can make changes to it. You can create new files, edit existing files, or delete files. You can use any text editor or IDE to work with your files.

6. Stage your changes: When you're ready to commit your changes, you'll need to stage them. To stage all changes, run the command git add . in the root directory of the repository. To stage specific files, run the command **git add [file].**

7. Commit your changes: After you've staged your changes, you'll need to commit them. To do this, run the command git commit -m "commit message", replacing "commit message" with a brief description of your changes.

8. Push your changes to GitHub: Once you've committed your changes, you can push them to GitHub by running the command **git push**.

9. Pull changes from GitHub: If there are changes to the repository on GitHub that you want to incorporate into your local repository, you can pull them by running the command **git pull.**

Fundamental Git commands

git init: Initializes a new Git repository in the current directory. This creates a new .git subdirectory that contains all the necessary files for version control.

git clone [repository URL]: Clones an existing repository from a remote location (such as GitHub) to your local machine. This creates a new directory with the same name as the repository.

git add [file]: Stages a file for commit. Git tracks changes to files, but you need to stage them before you can commit them.

git add .: Stages all files that have changed in the repository for commit.

git commit -m "commit message": Commits the staged changes to the repository. The commit message should be a brief description of the changes you made.

git push: Pushes your committed changes to the remote repository (e.g. GitHub).

git pull: Pulls changes from the remote repository to your local repository.

git status: Shows the status of the files in the repository. This command shows which files are staged for commit, which files have changed but haven't been staged, and which files are untracked (not yet added to Git).

git log: Shows a log of all the commits made to the repository. This includes the commit message, the author, and the date and time of the commit.

git branch: Shows a list of all the branches in the repository.

git checkout [branch name]: Switches to the specified branch. This command is used when you want to work on a different branch.

git merge [branch name]: Merges the specified branch into the current branch. This command is used when you want to incorporate changes from another branch into your current branch.

Branches in Git:

The **git branch** command is used to show a list of all the branches in the repository.
A branch is a separate line of development in Git. Each branch is essentially a separate version of the codebase that can be modified and merged back into the main branch (usually called the master branch) when changes are complete.

When you create a new branch in Git, you're essentially creating a copy of the current codebase that you can modify without affecting the main branch. You can use branches to work on different features or bug fixes independently of each other, and then merge those changes back into the main branch when they're ready.

The git branch command shows a list of all the branches in the repository, including the current branch. By default, Git creates a branch called master when you initialize a new repository. You can create additional branches using the git branch [branch name] command, where [branch name] is the name of the new branch.

You can switch between branches using the git checkout [branch name] command. This command changes the active branch to the specified branch, allowing you to work on that branch and make changes independently of the other branches.

Overall, branches are a powerful feature of Git that allow you to work on multiple versions of your codebase simultaneously. By creating separate branches for different features or bug fixes, you can keep your codebase organized and avoid conflicts between different changes.

File management:

To see which files are present in a Git repository, you can use the **git ls-files** command. This command lists all the files that are currently being tracked by Git in the repository, including any files that have been staged for commit.

Here's how to use the git ls-files command:

- Open a terminal or command prompt and navigate to the directory that contains the Git repository.

- Run the git ls-files command. This will display a list of all the files in the repository, one file per line.

If you want to see more information about the files in the repository, you can use the ls command instead of git ls-files. This will show all the files in the repository, whether or not they're being tracked by Git.

Keep in mind that some files may be excluded from version control using Git's .gitignore file. This file contains a list of file patterns that Git should ignore when tracking changes. If a file matches one of the patterns in .gitignore, it won't be listed by git ls-files.

Default steps to create a new repo

1. **echo "# LearningPython" >> README.md:** This command appends the text "# LearningPython" to the README.md file. The >> operator is used to append text to a file, and the echo command is used to print text to the terminal.

2. **git init:** This command initializes a new Git repository in the current directory. This creates a new .git subdirectory that contains all the necessary files for version control.

3. **git add README.md:** This command stages the README.md file for commit. This tells Git to track changes to the README.md file.

4. **git commit -m "first commit":** This command commits the changes to the README.md file to the Git repository. The commit message, "first commit", is a brief description of the changes made.

5. **git branch -M main:** This command renames the default branch from master to main. This is a more inclusive and culturally-sensitive default branch name for Git repositories.

6. **git remote add origin https://github.com/AmeyaKol/LearningPython.git:** This command sets up a remote repository for the local Git repository. The remote repository is located at https://github.com/AmeyaKol/LearningPython.git.

7. **git push -u origin main:** This command pushes the local Git repository to the remote repository on GitHub. The -u option sets the default upstream repository to origin, and the main branch is pushed to the remote repository.

Overall, this code initializes a new Git repository, adds a new file (README.md), commits the changes to the local repository, renames the default branch, sets up a remote repository on GitHub, and pushes the local repository to the remote repository. This is a common workflow for creating a new Git repository and pushing it to a remote hosting service like GitHub.