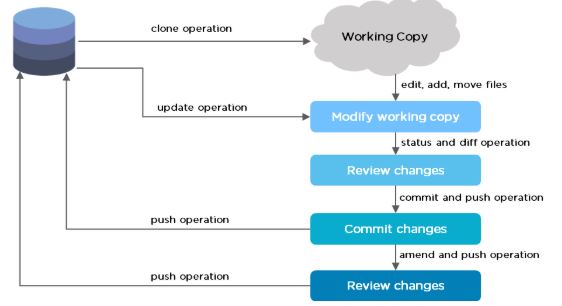
GIT and GITHUB TUTORIAL

WHAT IS GIT:

Git is a free and open source distributed **version control system used for tracking changes in computer files.**. It is a DevOps tool (**frankly just a SOFTWARE**) used for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. There are other tools like Bitbucket which are also used for this purpose, although they might not be free and open source.

Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development.

GIT Workflow:



TERMS :

Suppose you have a Git repository on your local machine, and a **working copy** of all of its files in front of you. Then,

1. Tracked and Untracked Files:

Each file in your working directory can be in one of two states: **tracked** or **untracked**.

**Tracked files** are files that were in the last snapshot, as well as any newly staged files; they can be unmodified, modified, or staged. In short, **tracked files are files that Git knows about**.

**Untracked files** are everything else — any files in your working directory that were not in your last snapshot AND are not in your staging area.

These are usually the files that are automatically generated and we don’t want to track them . ex: log files which are generated behind the scene. These are usually the files which we haven’t created or worked on.

NOTE:

* When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven’t edited anything.
* Suppose u created and added a new text file named “README”, then the Git wil show its status as “Untracked”. Untracked basically means that Git sees this file you didn’t have in the previous snapshot (commit), and which hasn’t yet been staged; Git won’t start including it in your commit snapshots until you explicitly tell it to do so. It does this so you don’t accidentally begin including generated binary files or other files that you did not mean to include.

2. Staging Area: If we say a file has been staged, basically means that we are ready with that version of file to be committed to the repository. A staging step in git allows you to continue making changes to the working directory, and when you decide you wanna interact with version control, it allows you to record changes in small commits (the next step)

Example: Changes to a file named ‘hello.html’ have been staged. This means that git knows about the change, **but it is not permanent in the repository** yet. The next commit will include these changes staged i.e the next commit would create a snapshot of this version of file .

Should you decide not to commit the change, the status command will remind you that you can use the**git reset** command to unstage these changes.

3. Commit : When we commit a staged file, we are basically saying that we are ready/comfortable with the changes to the file and hence commit the changes to the repository.

**Q:** Why do we need to Stage and Commit separately?? Or, Why cant we directly commit??

**ANSWER:** Separating staging and committing, you get the chance to easily customize what goes into a commit.

Example : Suppose you have edited three files (a.html, b.html, and c.html). After that you need to commit all the changes so that the changes to a.html and b.html were a single commit, while the changes to c.html were not logically associated with the first two files and were done in a separate commit.

In theory we should Stage and Commit the files a.html and b.html first :

git add a.html

git add b.html

git commit -m "Changes for a and b"

and then we should Stage and Commit the c.html file:

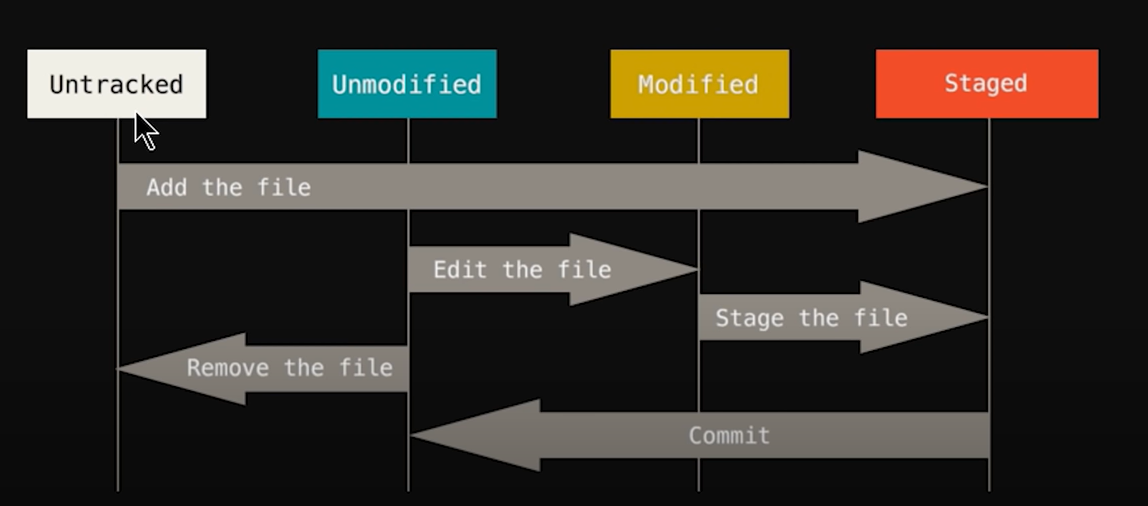
git add c.html

git commit -m "Unrelated change to c"

4. Modified Files : As you edit files, Git sees them as modified, because you’ve changed them since your last commit.

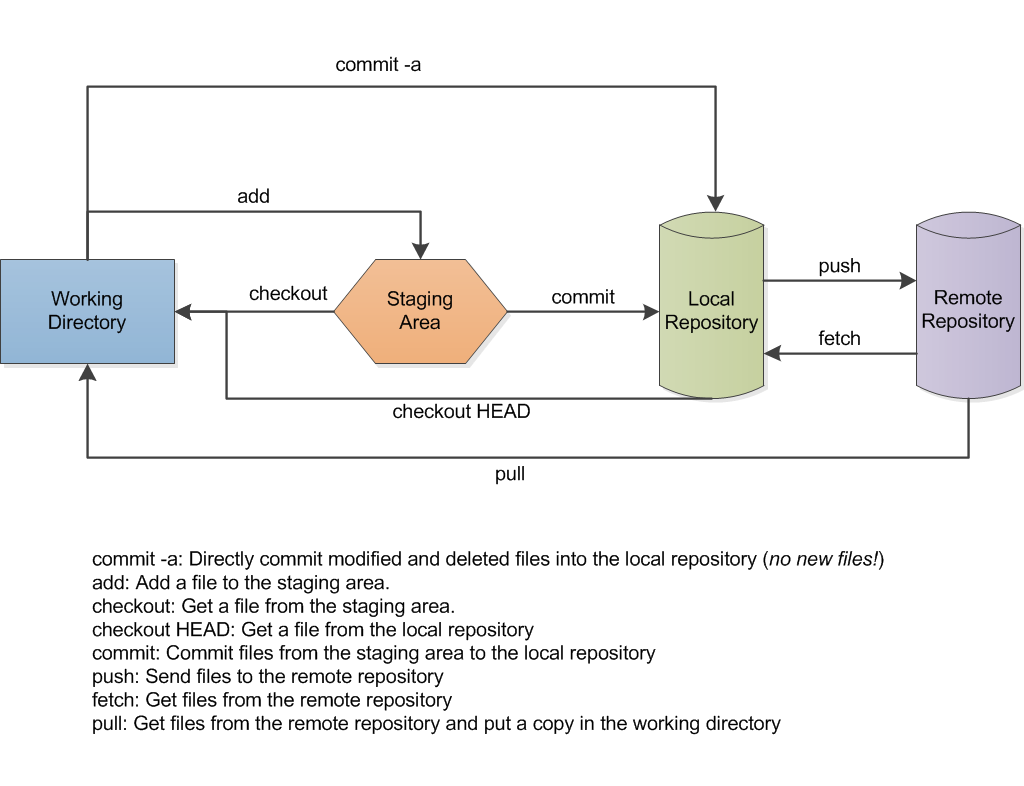
**NOTE**: If you have a Modified File, then you must Stage the file again (using “git add”) and then you either Commit the file (if you are satisfied with the changes) or you can even Un-stage the file using “**git reset”.**

5.Un-Modified Files:These are basically committed files. If we want we can edit the files or we can simply leave them as it is .



* **NOTE: Commit v/s Push**

Each developer has their own private repository to track their changes in. You make your changes locally and when you’re ready to share them with other developers, you push them back to the repository you all share. Commit saves the changes you made only to the local repository but not to the remote repository. Your commits are not automatically synchronized to the remote repository – you have to explicitly push the commits you want to share. When you use the push command, you apply your changes to the upstream repository.  
Basically, Pushing comes after committing.



**Git Commands for working using GIT Command Line:** We should learn Command line to work on wrepositories, because it gives us more control.

1. **git config**

Usage: git config –-global user.name “[name]” OR,

git config –-global user.name name

Usage: git config –-global user.email “[email address]” OR,

git config –-global user.email “[email address]”

This command sets the author name and email address respectively to be used with your commits.

Git Config Command - Git Commands - Edureka

Slly,

Usage: git config –-global user.name 🡪 This will display the user name

Usage: git config –-global user.email 🡪 This will display the email address

1. **git init**

Usage: git init [repository name] or, git init

This command is used to start a new repository. It’s the first step for making a repository from scratch.

GitInit Command - Git Commands - Edureka

1. **git clone**

Usage: git clone [url]

This command is used to obtain a repository from an existing URL. (ex: get repository from the server to ur local p.c.)



1. **ls -lart**

**Usage:** This command displays all the folder, even the hidden ones.

1. **git add**

Usage: git add [file]   or simply use: git add

This command adds a file to the staging area.

Git Add Command - Git Commands - Edureka

Usage: git add \*

This command adds one or more to the staging area.

Git Add Command - Git Commands - Edureka

Usage: git add -A

  This command adds all the Untracked files to the Staging area.

1. **git commit**

Usage: git commit -m “[ Type in the commit message]”

The command git commit records or snapshots the file permanently in the version history. The command git commit -m “[ Type in the commit message]”   also used



Usage: git commit -a

This command commits any files you’ve added with the git add command and also commits any files you’ve changed since then.

Git Commit Command - Git Commands - Edureka

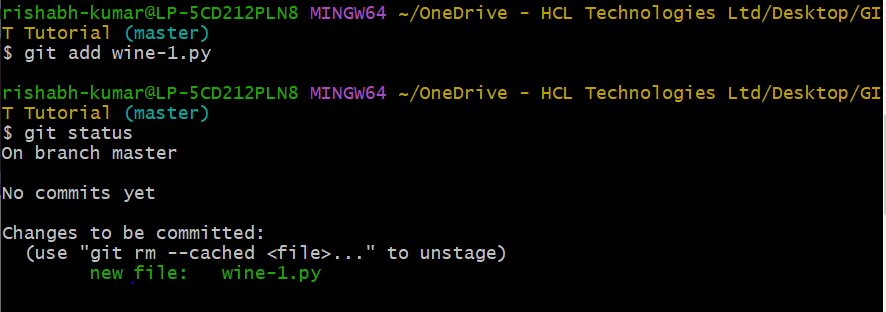
1. **git status**

Usage: git status

This command lists all the files that have to be committed. But, if they have been committed then it shows a message “nothing to commit, working tree clean” or “Untracked file”



Or,



**git diff**

Usage: git diff

This command shows the file differences which are not yet staged.



 Usage: git diff –staged

This command shows the differences between the files in the staging area and the latest version present.



Usage: git diff [first branch] [second branch]

This command shows the differences between the two branches mentioned.



**git reset**

Usage: git reset [file]

This command unstages the file, but it preserves the file contents.



Usage: git reset [commit]

This command undoes all the commits after the specified commit and preserves the changes locally.

Git Reset Command - Git Commands - Edureka

Usage: git reset –hard [commit]  This command discards all history and goes back to the specified commit.

Git Reset Command - Git Commands - Edureka

**git rm**

Usage: git rm [file]

This command deletes the file from your working directory and stages the deletion.

Git Rm Command - Git Commands - Edureka

**git log**

Usage: git log

This command is used to list the version history for the current branch.



Usage: git log –follow[file]

This command lists version history for a file, including the renaming of files also.



**git show**

Usage: git show [commit]

This command shows the metadata and content changes of the specified commit.



**git tag**

Usage: git tag [commitID]

This command is used to give tags to the specified commit.



**git branch**

Usage: git branch

This command lists all the local branches in the current repository.

Git Branch Command - Git Commands - Edureka

Usage: git branch [branch name]

This command creates a new branch.

Git Branch Command - Git Commands - Edureka

Usage: git branch -d [branch name]

This command deletes the feature branch.

Git Branch Command - Git Commands - Edureka

**git checkout**

Usage: git checkout [branch name]

This command is used to switch from one branch to another.

Git Checkout Command - Git Commands - Edureka

Usage: git checkout -b [branch name]

This command creates a new branch and also switches to it.

Git Checkout Command - Git Commands - Edureka

**git merge**

Usage: git merge [branch name]

This command merges the specified branch’s history into the current branch.

Git Merge Command - Git Commands - Edureka

**git remote**

Usage: git remote add [variable name] [Remote Server Link]

This command is used to connect your local repository to the remote server.

Git Remote Command - Git Commands - Edureka

**git push**

Usage: git push [variable name] master

This command sends the committed changes of master branch to your remote repository.



Usage: git push [variable name] [branch]

This command sends the branch commits to your remote repository.



Usage: git push –all [variable name]

This command pushes all branches to your remote repository.



Usage: git push [variable name] :[branch name]

This command deletes a branch on your remote repository.



**git pull**

Usage: git pull [Repository Link]

This command fetches and merges changes on the remote server to your working directory.



**git stash**

Usage: git stash save

This command temporarily stores all the modified tracked files.

Git Stash Command - Git Commands - Edureka

Usage: git stash pop

This command restores the most recently stashed files.



Usage: git stash list

This command lists all stashed changesets.

Git Stash Command - Git Commands - Edureka

Usage: git stash drop

This command discards the most recently stashed changeset.

Git Stash Command - Git Commands - Edureka

Want to learn more about git commands? Here is a [Git Tutorial](https://www.edureka.co/blog/git-tutorial/) to get you started. Alternatively, you can take a top-down approach and start with this [DevOps Tutorial.](https://www.edureka.co/blog/devops-tutorial)

GitCommand (Computing)

Python Terminal Commands:

1. The rm command

The rm command is used to delete files.

* **rm -i** will ask before deleting each file. Some people will have rm aliased to do this automatically (type "alias" to check). Consider using **rm -I** instead, which will only ask once and only if you are trying to delete three or more files.
* **rm -r** will recursively delete a directory and all its contents (normally rm will not delete directories, while rmdir will only delete empty directories).
* **rm -f** will forcibly delete files without asking; this is mostly useful if you have rm aliased to ``rm -i'' but want to delete lots of files without confirming each one.

NOTE: **rm -rf** is a dangerous command in linux. **if rm -rf runs with root privileges,** it force to delete all files and folders even hidden file and you must install os again. this command has not confirmation question. As a security measure, you can set rm to always get you approved for the delete operation, it uses the "-i" option whenever you want to delete a file or directory

1. List Installed Packages With Pip : pip list & pip freeze

Both  pip list  and  pip freeze  will generate a list of installed packages, just with differently formatted results. Keep in mind that  pip list  will list ALL installed packages (regardless of how they were installed). while  pip freeze  will list only everything installed by Pip.

1. List Modules In A Console Without Pip : help(module\_name)