DevOps

|  |  |
| --- | --- |
| Batch Code | DevOps BC=221408 |
| Topic Name | Class-2\_20Aug22-Git by Trainer-Deepak |
| Video Link | [GIT by Trainer-Deepak](file:///D:\Intellipaat\DevOps\Class-2_20Aug22-Git%20by%20Trainer-Deepak\Class-2_20Aug22-Git%20by%20Trainer-Deepak.mp4) |
| Support Study Material | [GIT pdf book](file:///D:\Intellipaat\DevOps\Class-2_20Aug22-Git%20by%20Trainer-Deepak\Module-2_Version_Control_System_GIT-PPT.pdf) |

What is version control system?

It is a system that manages the changes to the document, it helps us tracking the changes when multiple peoples working on same document.

Types of VCS

Centralized

All the developers working on a piece of code, all are keeping their code on a centralized server so that if anyone can work upon onto the same code, they will get updated one always.

E.g., Mercurial, SVN these are some vendors in market who are providing centralized VCS.

Pros and Cons

Cons:

* No local repositories
* If central server fails/crashes, all codes are inaccessible
* If no internet connectivity, failed to push/pull the code to central server

Distributed

All cons are solved in distributed VCS.

Pros and Cons

Pros:

* Host local repositories in each and every client system
* Every developer clones a copy of main repository to his local system
* Always first put the changes to local repository then it pushes to central server E.g., Perforce, Mercurial, GIT is some of Distributed VCS.

What is GIT?

It is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development.

GIT

Centralized VCS

2. Push

Code Version V1”

Code Version V1’

Code Version V1

Local Repo.

Local Repo.

Local Repo.

2. Push

2. Push

1. Push
2. Push
3. Push

Lifecycle of GIT

Following are the lifecycle stages of files in GIT.

Committing Area

Staging Area

Working Area

Working Area

* Place where project resides in local disk.
* This place may or may not be tracked by Git
* The project can be tracked by git, by using the command ***git init***
* By doing ***git init,*** it automatically creates a hidden .git folder.

Staging Area

* Once we are in working area, we have to specify which files are to be tracked by Git.
* To add files in staging area, we use command ***git add***

Git add command can be use with single file name:

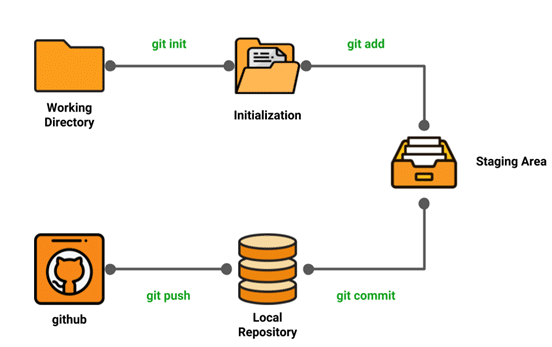
***git add <file name>***

Or can be use to add multiple files at once:

***git add .***

Committing Area

* Once the files are selected and are ready in staging area, they can now be saved in repository
* Saving a file in the repository of git is known as doing a commit
* When we commit a repository in git, the commit is identified by commit id
* The command for initializing this process is ***git commit -m “message”***



How does GIT work?



Project Folder



Commit ID: 0001

Project Folder



Commit ID: 0001



Commit ID: 0002

All these commits are bound to a branch. Any new commits made will be added to this branch.

A branch always heads to a latest commit.

The pointer to a latest commit is known as Head.

Master Branch

Commit ID: 0002

Commit ID: 0001

The default branch in a git repository is called the Master branch.

Why do we need branch?

Master Branch

Commit ID: 0003

Master Branch

Once the code is finished, tested and ready we can merge the Feature A branch, with the master branch and now the code is available on the production servers as well.

Commit ID: 0003

Feature A Branch

Commit ID: 0001

Commit ID: 0002

Commit ID: 0002

No matter how many commits are made by the developer in feature A branch, it will not affect the master branch.

Suppose for enhancing the code by adding a feature A, the task has been assigned to a developer in a separate branch A. This is done, so that master contains only code which is finalized.

Feature A Branch

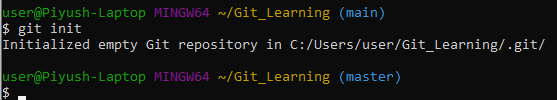
Commit ID: 0002

Commit ID: 0002

Commit ID: 0001

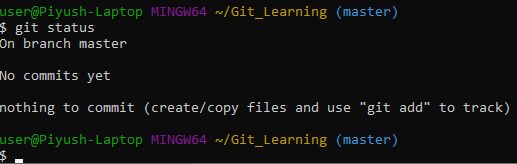
Common Git Commands:

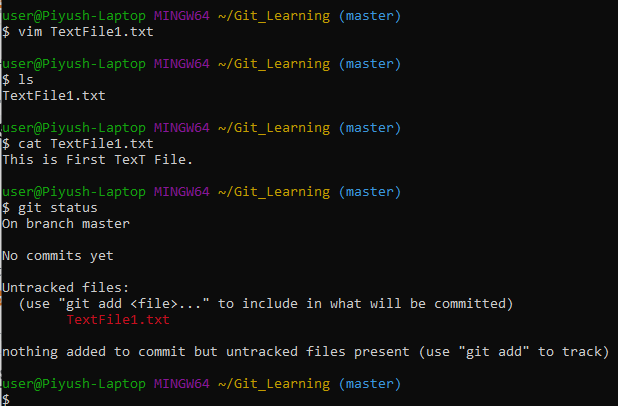
* Creating repository: command is git init



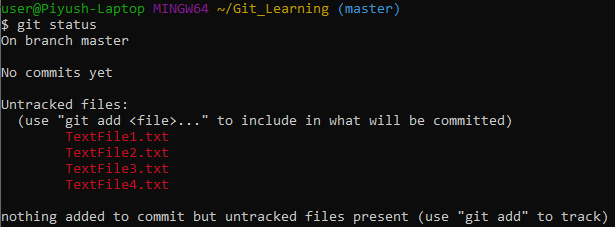
* Making changes: the command is used to check the status of files, whether they are being tracked by the Git or not.

1. command is git status

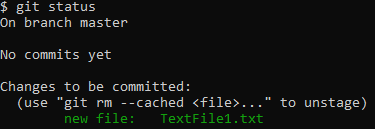




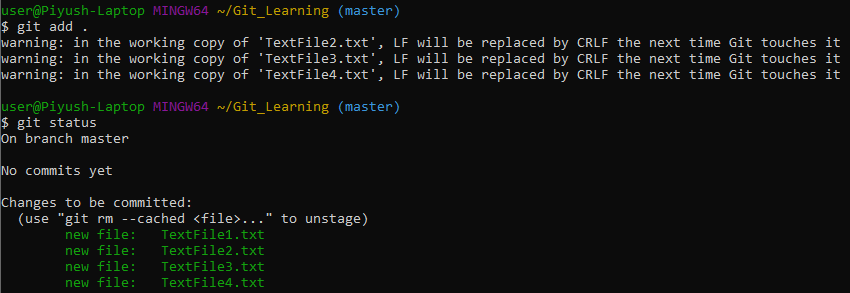
If, no files are tracked, then with the (2) command git add <file name> we stage these files.



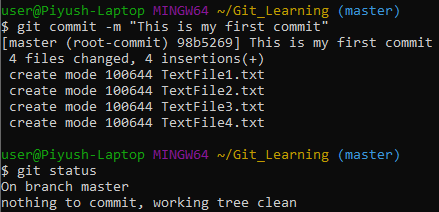




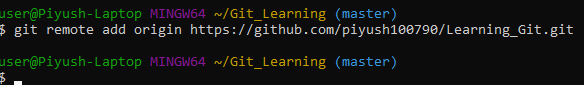
If we want to add all files at once we can use (3) command git add .



Once the files are staged, we are ready to commit them in our repository. We can commit the files using the (4) command git commit -m “custom message”



* Syncing repositories: Once everything is ready in our local repository, we can start pushing our changes to remote repository. Copy repository link and paste it in the (1) command git remote add origin “<URL to repository”



To push the changes to repository, enter the (2) command git push origin <branch name> and hit enter. This command will then prompt for user name and password, provide details and hit enter.

The local repository is now synced with remote repository on GitHub.

Similarly, if we want to download the remote repository to local system, we can use the (3) command git clone <URL>. This command will create a folder with the repository name, and download all the contents of the repository inside the folder.

The (4) command git pull <URL of link> is also used for pulling the latest changes from the repository, unlike git clone, this command can only work inside an initialized git repository. This command is used when you are already working in the cloned repository, and want to pull the latest changes, that others might have pushed to the remote repository.

* Parallel development: Imagine multiple developers are working on the same project or repository. To handle the workspace of multiple developers, branches are used. To create a branch from an existing branch, type (1) command git branch <branch name>

Similarly, to delete a branch use the (2) command git branch -D <branch name>

To switch to the new branch, type the (3) command git checkout <branch name>

Want to check the log of every commit detail in repository use the (4) command git log

If you did not wish to save your work to git repository, this command is helpful (5) git stash. If wish to stash the untracked files as well use (6) command git stash -u.

Once back and want to retrieve working, type (7) git stash pop.

This command helps to go back to previous commit, (8) git revert <commit-id>

Commit-id can be obtained by the output of command git log.

To check the differences between two versions of file use command (9)

git diff <commit-id of version x> <commit-id of version y>. Here again, commit-id can be obtained by the output of command git log.