

GIT

Global Information Tracker



IT IS A VERSION CONTROL SYSTEM (VCS) OR SOURCE CODE MANAGEMENT (SCM).



VERSION 1.0



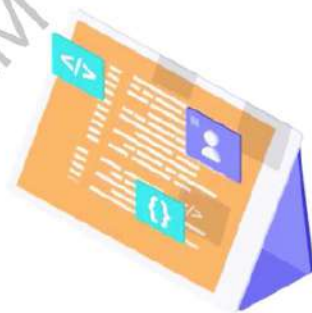
VERSION 2.0



VERSION 1.0 CODE



VERSION 2.0 CODE



VERSION 3.0 CODE

VERSION CONTROL SYSTEM (VCS)

It will maintain multiple versions of the same file.

It is platform-independent.

It is free and open-source.

They can handle larger projects efficiently.

They save time and developers can fetch and create pull requests without switching.

VCS HISTORY

SCCM:
To track only one file
[1972]



RCS:
Track multiple files
but not directories



CSV:
Track multiple files
and directories but
single user



SVN:
Track multiple files
and directories
[2000]



GIT:
Distributed Version
Control System
[2005]



WHY GIT?

If client asks you to develop application



You can develop according to client requirements



you released version-1 of the application



One year later, the client comes to you and asks to change the options in the application





But the application got failed, In this case you can rollback to specific previous version.

GIT is a Distributed Version Control System



GIT tracks changes you made, so you have a record of what has been done, and you can revert to specific versions. it makes collaboration easy, allowing changes by multiple people to all merged into one source

GIT ADD:

- Git add command is a straightforward command. It adds files to the staging area.
- We can add single or multiple files at once in the staging area.
- Every time we add or update any file in our project, it is required to forward updates to the staging area.
- The staging and committing are co-related to each other.



TYPES OF REPOSITORIES:

LOCAL REPO:

The Local Repository is everything in your .git directory. Mainly what you will see in your Local Repository are all of your checkpoints or commits. It is the area that saves everything (so don't delete it).

REMOTE REPO:

The remote repository is a Git repository that is stored on some remote computer.

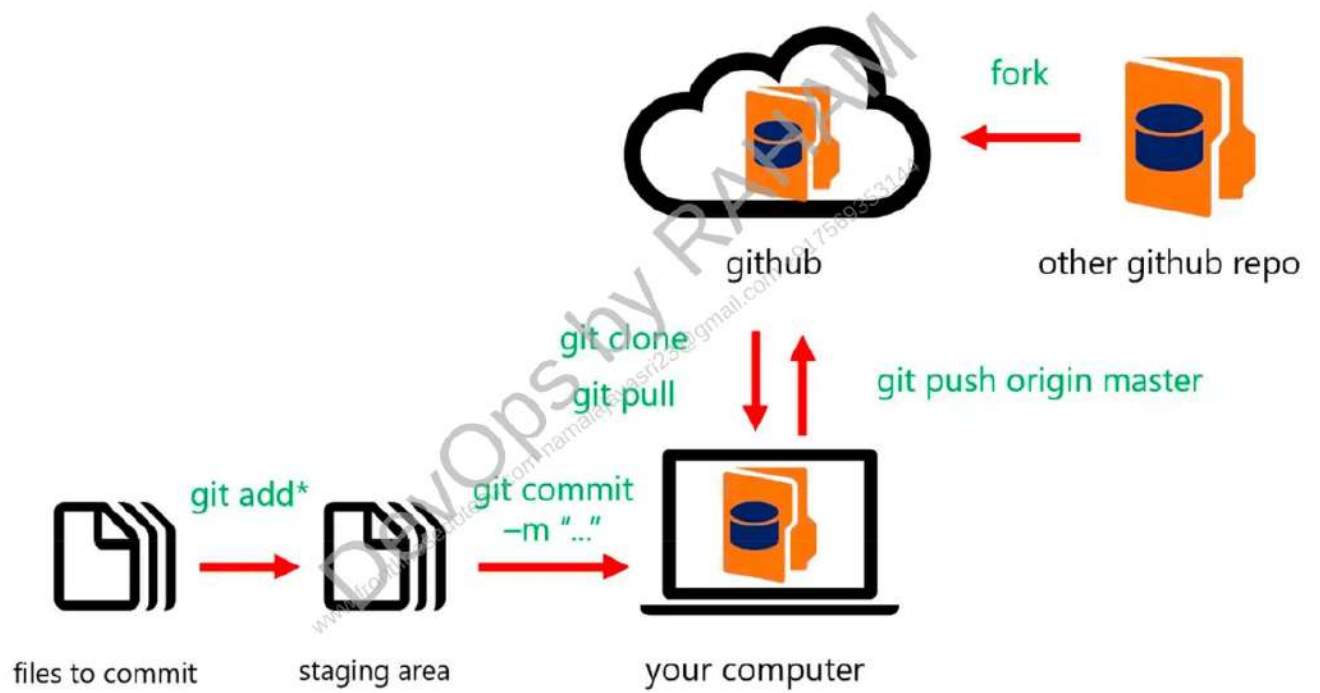
CENTRAL REPO:

This will be present in our GITHUB

GIT ALTERNATIVES:

GIT LAB, SVN, BIT BUCKET, P4, STASH, HELIX

GIT WORK FLOW:





GIT INSTALLATION:

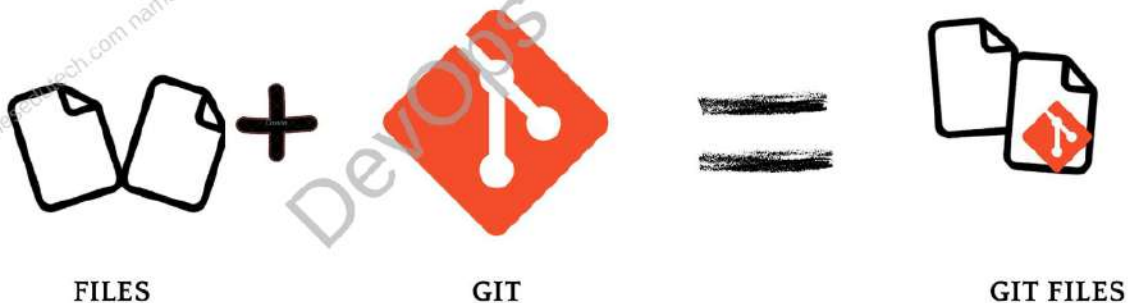
- `yum install git -y`
- `git init .`

COMMIT A FILE:

- Create a file : `touch filename`
- Now add that file : `git add .` (Dot represents current directory)
- commit the file with message : `git commit -m "commit message you want" filename`
- To see details of that file : `git log`

GIT ADD:

- Git add command is a straightforward command. It adds files to the staging area.
- We can add single or multiple files at once in the staging area.
- Every time we add or update any file in our project, it is required to forward updates to the staging area.
- The staging and committing are co-related to each other.



GIT COMMIT:

- It is used to record the changes in the repository.
- It is the next command after the git add.
- Every commit contains the index data and the commit message.

GIT STATUS:

- The git status command is used to display the state of the repository and staging area.
- It allows us to see the tracked, untracked files and changes.
- This command will not show any commit records or information.



GIT CONFIGURE:

- if you want to give your username and E-mail id to those commits then

- `git config user.name "username"`
- `git config user.email "userxyz@gmail.com"`



NOTE: now give the git log command to see changes, it won't work because after configuring we haven't done anything.

Now create a file and commit that file and give git log you will see changes as you configure.

GIT IGNORE:

- It will be useful when you don't want to track some specific files then we use a file called .gitignore
- create some text files and create a directory with "jpg" files.

- `ui .gitignore`
- `*.txt`

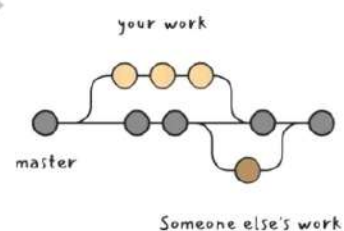
Now all the txt files will be ignore



GIT BRANCHES:

- A branch represents an independent line of development.
- The git branch command lets you create, list, rename, and delete branches.
- The default branch name in Git is master.

- | | | |
|----------------------------------|---|--|
| • To see current branch | : | <code>git branch</code> |
| • To add new branch | : | <code>git branch branch-name</code> |
| • To switch branches | : | <code>git checkout branch-name</code> |
| • To create and switch at a time | : | <code>git checkout -b branch-name</code> |
| • To rename a branch | : | <code>git branch -m old new</code> |
| • To clone a specific branch | : | <code>git clone -b branch-name repo-URL</code> |
| • To delete a branch | : | <code>git branch -d <branch></code> |

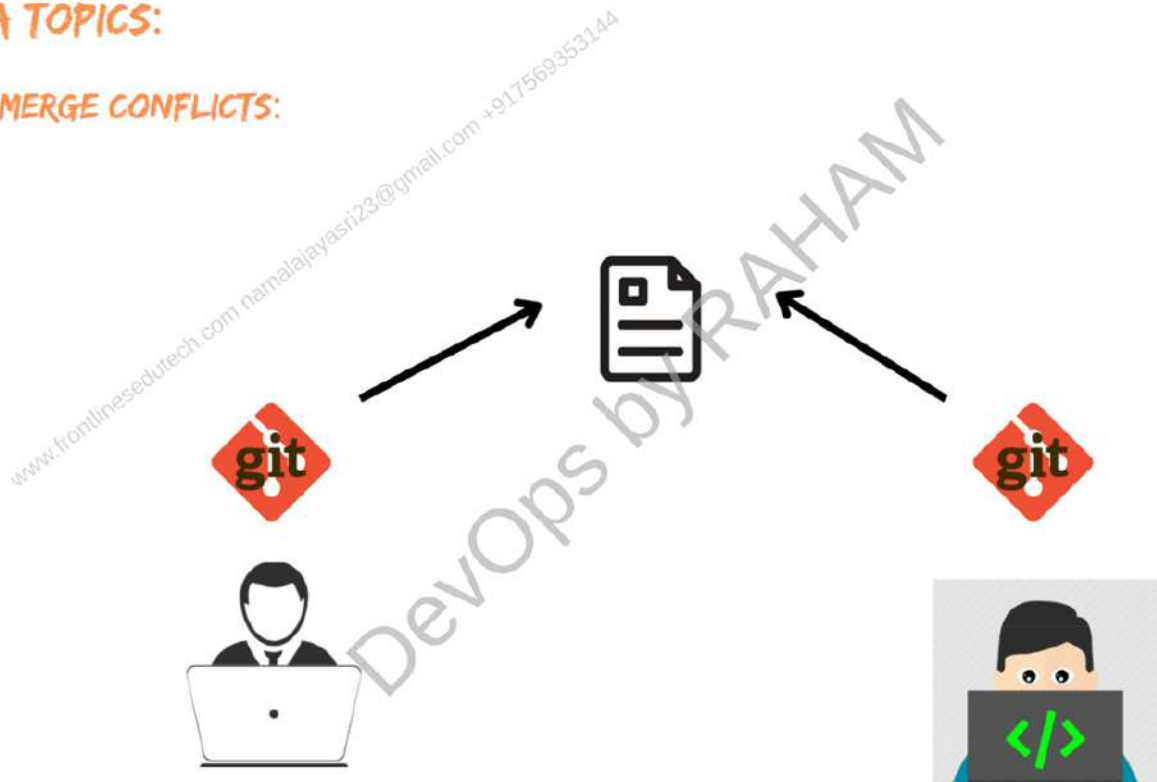


The -d option will delete the branch only if it has already been pushed and merged with the remote branch. Use -D instead if you want to force the branch to be deleted, even if it hasn't been pushed or merged yet. The branch is now deleted locally.

Now all the things you have done is on your local system.
Now we will go to GIT HUB.

EXTRA TOPICS:

GIT MERGE CONFLICTS:



Git makes merging super easy!

CONFLICTS generally arise when two people have changed the same lines in a file (or) if one developer deleted a file while another developer is working on the same file!

In this situation git cannot determine what is correct!

Lets understand in a simple way!

```
cat>file1 : hai all
add & commit
git checkout -b branch1
cat>file1 : 1234
add & commit
git checkout master
cat>>file1 : abcd
add & commit
```

```
git merge branch1 : remove it
```

IDENTIFY MERGE CONFLICTS:

see the file in master branch then you will see both the data in a single file including branch names

RESOLVE:

open file in VIM EDITOR and delete all the conflict dividers and save it!

add git to that file and commit it with the command (git commit -m "merged and resolved the conflict issue in abc.txt")

GIT REBASE:

if you have 5 commits in master branch and only 1 commit in devops branch, to get all the commits from master branch to devops branch we can use rebase in git. (command: git rebase branch_name)

GIT CHERRY-PICK:

if you have 5 commits in master branch and only 1 commit in devops branch, to get specific commit from master branch to devops branch we can use cherry pick in git. (git cherry-pick commit_id).

GIT STASH:

Using the git stash command, developers can temporarily save changes made in the working directory. It allows them to quickly switch contexts when they are not quite ready to commit changes. And it allows them to more easily switch between branches.

Generally, the stash's meaning is "store something safely in a hidden place."

- git stash
- git stash apply
- git stash list
- git stash remove
- git stash pop
- git stash clear



SOME EXTRA COMMANDS:

`git show <commit> --stat` : you'll see the commit summary along with the files that changed and details on how they changed.

`git commit --amend -m "New commit message"` : to edit the commit message

`git commit --amend --no-edit` : used to add some files in previous commit. (--no-edit means that the commit message does not change.)

`git update-ref -d HEAD` : used to delete 1st commit in the git

`git reset commit`: used to delete all the commits (upto the commit id)

`git commit --amend --author "Author Name <Author Email>"` : used to change the author of latest commit

MERGE VS REBASE:

When there are changes on the main branch that you want to incorporate into your branch, you can either merge the changes in or rebase your branch from a different point.

merge takes the changes from one branch and merges them into another branch in one merge commit.

rebase adjusts the point at which a branch actually branched off (i.e. moves the branch to a new starting point from the base branch).

Generally, you'll use rebase when there are changes are made in main/master branch that you want to include in your branch. You'll use merge when there are changes in a branch that you want to put back into main.

to merge: `git merge branch_name`

to rebase: `git rebase branch_name`



GitHub is a web-based platform used for version control. It simplifies the process of working with other people and makes it easy to collaborate on projects. Team members can work on files and easily merge their changes in with the master branch of the project.

PUSH YOUR CODE TO GIT HUB:

- `git remote add origin url`



()



- `git push -u origin branch-name`



()



GIT CLONING:



It means having same files in another folder.

To clone a git repo we need to have a repository and also check our present working directory.

- **git clone repo_url**

now we just cloned the files in repo-A to repo-B.

But before cloning we need to add and commit our files.

GIT MERGE:

- If you want to merge branch-1 with branch-2 switch to branch-1 first and give command `git merge branch-2`
- now that command had merged the content of branch-1 to branch-2.
- Whatever the content in branch-1 will be seen in branch-2 now.

`git merge branchname`

GIT FORK:

A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project

GIT PULL:

push some from local to central and add some commits in central. you can see those central commits from local by using git fetch.

git fetch is used to get the updates from central repo.

git fetch : used to get the changes from central to local

git status : to see the differences of the commits

git log origin/branch_name : to see the logs of remote repo

to get those changes from central to local we use

git pull origin branch_name






ADVANTAGES:

- Speed
- Simplicity
- Fully Distributed
- Excellent support for parallel development, support for hundreds of parallel branches.
- Integrity

DISADVANTAGES:

- Windows support issue.
- Entire download of the project history may be impractical and consume more disk space if the project has long history.

COMPARISION:

	 HELIX TEAMHUB	 GITLAB	 GITHUB	 BITBUCKET
Side-by-side view	✓	✗	✗	✗
Quick Action Buttons	✗	✓	✓	✓
Supports adding images	✓	✓	✓	✓
Supports adding other type of attachments	✓	✓	✗	✗
Search functionality for wiki pages	✓	✗	✗	✗
Support for multiple markup languages	✗	✓	✓	✓
Possibility to view the history on code level	✓	✗	✗	✓