

**What is Continuous Integration and Continuous Deployment(CI/CD)?**

Continuous Integration (CI) is a practice in which developers can able to merge their code ,create a build of their code and test their code.

Continuous Deployment (CD) is a practice in which we can release the changes from the repository to different environment ( non-prod , stage , Production).

**What is the difference between Git and Github?**

Git is a version control system used to track changes that generally occur in code and documents.

Github is the web/host service to git repository and different tools for collaboration.

**what are other version control system?**

SVN , Beanstalk , GitLab

**How does GIT work?**

Git specifically works by taking “snapshots” of files. In Git repository ll commits, files, tags, and filesystem tree are different types of objects. Which are stored in form of key-value pair.

**What is snapshot in git?**

While developing something, we make change in our code. When we use git commit to git repository, it save a snapshot of all the files in the repository.

So when we check out a particular commit " git checkout commit hash " git reconstructs a complete snapshot of the code at the time.

The each snapshot detail present in ".git/object"

**How to creates a new Git repository?**

The "git init" command creates a new Git repository.

```
Ex → cd StudyDevops/
      pwd
      git init # create local repository
      ls -la
```

**How to Connects to the remote repository?**

```
git remote add origin the_remote_repository_URL
```

**How to Verifies the new remote URL?**

```
git remote -v
```

**What is Git Merge?**

Git merge is used to merge two repositories, without losing the data.

```
git merge <branch_which_we_want_to_merge> <branch_where_we_want_to_merge>
```

```
git merge <a_branch_name> -----> the branch will merge to current branch
```

**How we can rollback to any previous version in git?**

```
git checkout <hash-code>
```

**What is the difference between Non-bare Repositories , Bare Repositories?**

Non-bare Repositories are default git repository has a .git folder. We can perform all the operations in it.

All bare repositories have .git extension. No commits can be made in a bare repository.

Possible operations on the Bare Repository is Push and Clone. So it act as a central repository.

That means any new contributor can clone the repository into a local one without getting any unsaved changes or conflicting work of others.

**What is .gitignore file?**

.gitignore is an auto-generated file inside the project folder that ignores/prevents files which are listed in it to get committed to the local and remote repositories.

**What is the difference between fork and clone?**

Fork is use to create a copy of a repository in github.

Cloning makes a local copy of an existing repository.

**How to Clone an existing repository?**

`git clone git_repo_url`

EX → `mkdir AllDevopsTopics`

`cd AllDevopsTopics/`

`git clone https://github.com/sksumanta/studyDevops.git # clone the remote repository`

**what is the difference between git remote and git clone?**

git remote adds, a reference to a remote repository for further tracking.

`git remote add origin "url-of-repository"`

git clone is used to copy or clone a different repository.

`git clone "url-of-repository"`

**How to check what are the changed files in your working directory in git?**

`git status`

**What is staging area?**

After "git add ." and before "git commit -m "message" " the code store in a temporary area that is known as staging area.

**How to Add all current changes to staging area in git?**

`git add`

**How to Add some changes interactively to staging area?**

`git add -p`

**Difference between "git add -A" and "git add" ?**

"git add" ---- places the modified version of the file from working directory to the staging area.

"git add -A" ---- stages all the changes.

**What is "git add -u" ?**

" git add -u " only adds currently tracked files (which have been modified) to the staging area and also checks if they have been deleted. It will operate on the entire tree.

**What is "git add ."?**

only adds what is present there, not what has been deleted (if tracked).

**How to Commit all local changes?**

`git commit -a`

How to Commit previously staged changes?

```
git commit -m "message"
```

How to overwriting commit message and adding another change?

```
git commit --amend -m "New commit message"
```

```
ex-->git add change_file
```

```
git commit --amend -m "New commit message"
```

How to see Show all commits?

```
git log
```

The Git log ----- shows Commit Hash , author name , Commit Date , Commit Message

How to view Commit History by Commit ID?

```
git log <commit hash>
```

How to view git log /Commit History of a File?

```
git log <filename>
```

How to view Commits History between two commit hash / for a range?

```
git log <older commitId>..<newer commitId>
```

How to view a limited number of Git Commits History?

```
git log -n number -oneline
```

```
git log -n 3 -oneline (issue last 3 git commits) it will show last 4 commit logs
```

what is git blame?

The git blame command is used to examine the contents of a file line by line and see when each line

was last modified and who was modified.

```
git blame <filename> -----> it will show you what you need to know for the file .
```

```
git blame -L <starting line number>,<ending line number> <filename>
```

what is git branch?

Git branches are effectively a pointer to a snapshot of your changes.

why git branch?

When developers want to add new feature or fix for a bug, it is not recommended to do the activity in the master branch. if the change committed directly in the master branch, then the code may become unstable. So developer need to create new branch to encapsulate their changes. Once the work is done, they can merge the branch to the main branch.

What Branching strategy you follow?

```
master Branch -----> dev Branch
```

```
|----- feature1 Branch |___ merge to Dev Branch --- Unit test , code coverage ,security or compliance issues
```

```
|----- feature2 Branch | |___ if success dev code merge to staging branch same test conducted
```

```
|___ if success code merge to master branch
```

**How to create new branch?**

`git branch <branch-name> -----> Create a new branch`

**How to list local branches?**

`git branch`

**How to list remote branches?**

`git branch -r`

**How to list all local and remote branches?**

`git branch -a`

**How to create a new branch and checkout to that branch?**

`git checkout -b branch-name`

**How to restoring a Revision/version in a New Local Branch**

`git checkout -b new-project-state commit_hash`

**How to Switch to a Branch in Local Repo?**

`git checkout branch-name`

**How to switch to a Branch that Came from a Remote Repo?**

`git checkout --track origin/branch-name`

**How to push to a branch if branch is not in the remote repo?**

`git push -u origin branch-name`

or

`git push -u origin HEAD`

**How to push the branch if local branch already exists on the remote?**

`git push`

**How to delete a remote branch?**

`git push origin --delete branch-name`

**How to delete a local branch?**

`git branch -d my-branch-name --> The -d option only deletes the branch if it has already been merged`

`git branch -D my-branch-name --> The -D option is use to delete the branch irrespective of its merged status.`

**Delete a branch on the remote?**

`git branch -dr my-branch-name`

or

`git push origin --delete branch_name`

or

`git push origin :branch_name`

**How to rename the current branch\_name to new branch\_name?**

`git branch -m <branch_name>`

How to Rename the differnt branch\_name to new branch\_name?

```
git branch -m <old_branch_name> <new_branch_name>
```

How to create remote branch?

```
git remote add new-remote_branch https://github.com/XXXXXX/XXXXX.git
```

How to pushes the local branch to a newly created remote branch?

```
git push new-remote_branch local_repo_branch_name
```

What is difference between tag and branch in Git?

Branches allow you to code features or fix bugs without impacting the main code branch.

Tags are essential for marking a point in time in your code, such as a new release of your application.

How to create a tag?

```
git tag tag_name
```

How to Mark the current commit with a tag? or

How to ReTagging/Replacing old tags?

```
git tag -a -f tag_name commit_hash ( Ans for both qus )
```

```
git tag -a -f v1.4 15027957951b64cf874c3557a0f3547bd83b3ff6
```

How to push a tag / Publish your tag?

```
git push origin tag_name
```

How to Checking out tags?

```
git checkout tag_name
```

How to checkout particular commit/snapshot/ Revision into a new branch?

```
git checkout -b temporay-test-branch commit_hash
```

```
ex --> git checkout -b temp-test-branch 56a4e5c08
```

so it will create a temporary branch and cpoy the commit into that branch, then you

do your

work.once your work is complet you can delete the branch.

what is detached HEAD?

when a specific commit hash is checked out instead of a branch that is know as detached head. In this case head pointer move to the specific commit so when we do any changes that changes not belongs to any branch. This means they can easily get lost once you check out a different revision or branch as the changes not recorded in the context of a branch.

To avoid this you can do the below

```
git checkout -b temporay-test-branch commit_SHA1_hash_reference
```

```
ex --> git checkout -b temp-test-branch 56a4e5c08
```

so it will create a temporary branch and cpoy the commit into that branch, then you

do your

work.once your work is complet you can delete the branch.

In case of "rebase" git temporary detached HEAD.

### How to Merging Changes in git?

Everytime it is not possible to integrate the individual commit to the master. Instead, you tell Git which branch you want to integrate and Git will figure out which commits you don't have in your current working branch.

Only these commits will be integrated.

`git checkout master` # moving to the master or current working branch

`git merge branch_name` # the branch will be merged with the current working branch (that is master branch)

### Why merge conflict?

if two people changed the same file, or if one person decided to delete the file while the other person decided to modify the same file, that time Git simply cannot know what is correct.

Then git mark the file as conflict.

At the time of integration git will create merge conflict.

### How to Handle Merge Conflicts?

Step 1: The easiest way to resolve a conflicted file is to open it and make necessary changes.

Step 2: After editing the file, use the git add a command to stage the new merged content.

Step 3: The final step is to create a new commit with the help of the git commit command.

### Merge Strategies in Git?

Before making a merge option make sure the receiving branch and the merging branch are up-to-date with the latest remote changes.

Git provides various methods to merge different commits into a base commit.

Fast Forward

Recursive

Ours

Octopus

Resolve

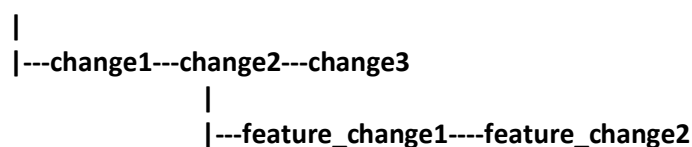
Subtree

The most commonly used strategies are Fast Forward Merge and Recursive Merge.

### What is Fast Forward Merge Strategies in Git?

When you create a branch, make some commits in that branch, the time you're ready to merge, there is no new merge on the master. That way master's pointer is just moved straight forward and history is one straight line.

Master



In this case we need to use ----> `git rebase feature_branch`

example--->

`git branch`

o/p --> \* master

multi

sub

`git branch div` ( create a new branch)

`git checkout div`

`touch dividing.txt`

```

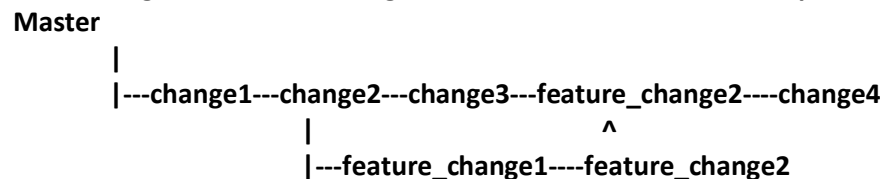
git add dividing.txt
git commit -m "dividing" dividing.txt

git checkout master
ls
o/p--> addition.txt , multiplay.txt
git rebase div ( while present in master branch )
ls
o/p --> addition.txt , multiplay.txt , dividing.txt ( get in master branch)

```

### What is Recursive merge strategies?

In Recursive merge, after you branch and make some commits, there are some new original commits on the 'master'. So, when it's time to merge, git recurses over the branch and creates a new merge commit. The merge commit continues to have two parents.



In this case we need to use ----> `git merge --no-ff feature_branch`

### What is Ours merge strategies?

While resolving merge conflicts, you could instruct git to pick all the changes made to one branch over another.

In this case we have two merge strategy options: -Xtheirs and -Xours

If you want to override the changes in the feature branch with your master branch, you can run the following command after checking out to master:

```
git merge -Xours feature
```

checkout to	Command	with Strategy	Outcome
master	git merge feature	-Xtheirs	Keep changes from feature branch
master	git merge feature	-Xours	keep changes from master branch
feature	git rebase master	-Xtheirs	Keep changes from feature branch
feature	git rebase master	-Xours	keep changes from master branch

There also another option for Ours merge that is " `git merge -s ours branch_name` "

```

git checkout sub
ls
    add.txt , sub.txt,div
    touch div/div1.txt
git commit

git checkout master
ls
    add.txt , multi.txt, sub.txt,div
git merge -s ours sub
ls
    add.txt , multi.txt, sub.txt,div ( here is no change in master )

```

**What is Octopus merge strategies?**

octopus merges involve merging 2 or more branches onto the current branch using a single commit.

```
git merge -s octopus feature1 feature2 feature3
```

```
git checkout master
ls ----- add.txt , multi.txt, sub.txt,div
git merge -s octopus algebra physics math
ls ----- add.txt , Algebra1.txt , integration.pdf, multi.txt,
phy_wave.txt, sub.txt,div ( in master branch)
```

**What is Resolve merge strategies?**

The resolve merge strategies is a safe and fast merge technique to solve only two HEADs using a 3-way merge algorithm.

```
git merge -s resolve feature1
```

```
git checkout master
ls
add.txt , multi.txt, sub.txt,div
git merge -s resolve div
ls
add.txt , multi.txt, sub.txt,div
```

**What is Subtree merge strategies?**

The subtree merge strategy is useful when merging a branch (or as we'll see in this recipe another project) into a subdirectory of a Git repository instead of the root directory. When using the subtree merge strategy, the history of the subproject is joined with the history of the super project, while the subproject's history can be kept clean except for commits indented to go upstream.

```
git merge -s subtree branch_name
```

```
git checkout div
ls
add.txt , multi.txt , sub.txt
git subtree add --prefix=sub_merge https://bitask.org/vi_mirror/vi_su.git
o/p --> added dir sub_merge

git checkout master
ls
git merge -s subtree div
ls
add.txt , multi.txt , sub.txt , sub_merge
```

**Fast-Forward merge vs Recursive merge?**

**Fast Forward** No new commits on the master

Linear History

No merge commits

git rebase

**Recursive** New commits on the master

Commit 2 parents

Merge commit is created

git merge--no-ff



**How to tracked Changes to files in git?**

**git diff**

**How to Showing Differences for a Specific File or Directory?**

**git diff file\_or\_directory\_name**

**git diff multi.txt**

**git diff machinefolder**

**check the changes in a particular commit?**

**git diff commit\_id**

**how to check difference between two separate commits?**

**git diff commit-id-1 commit-id-2**

**how to check difference between two separate commits for a file?**

**git diff commit-id-1 commit-id-2 file\_name.**

**git diff 6eeda c87e1e physical.txt**

**How to check difference from a commit id to the latest commit on the branch for the directory?**

**git diff commitId branch\_name directory\_name**

**git diff c87e1e master multi**

**How to check the changes between the previous commit and the current commit.**

**git diff HEAD^ HEAD**

**How to check changes in a particular branch?**

**git diff branch\_name**

**How to check difference between the two branches?**

**git diff branch1\_name branch2\_name**

**How to check the summary of changes for a commit?**

**git diff -stat commit\_id**

**How to check the summary of changes for a branch?**

**git diff --stat branch**

**How to check files name in a folder that changed after a commit?**

**git diff --name-only commit-id folder-path**

**How to check the file name which are different than a branch in the current branch?**

**git diff --name-only other\_branch\_name**

**How to check the file name which are different than a commit in the current commit?**

**git diff --name-only other\_commit-id**

**What is the Difference Between Git Fetch and Git Pull?**

git fetch only downloads new data from a remote repository - but it doesn't integrate any of this new data into your working files.

`git fetch <remote>`

git pull downloads new data integrates it into your current working files.

Since "git pull" tries to merge remote changes with your local ones, so it may create "merge conflict".

`git pull<remote><branch>`

**What is git head and how it work?**

Git head is the pointer to current revision/branch. When we execute "git checkout" command git determine which revision of your project you want to work on and automatically moves the HEAD pointer . Then git places all of that revision's files in your working copy folder. Normaly we can provide "git checkout" command as below

`git checkout branch_name`

or

`git checkout commit_SHA1_hash_reference` # commit\_SHA1\_hash\_reference is the first 7 char of commit

**what is detached HEAD?**

when a specific commit hash is checked out instead of a branch that is know as detached head. In this case head pointer move to the specific commit so when we do any changes that changes not belongs to any branch. This means they can easily get lost once you check out a different revision or branch as the changes not recorded in the context of a branch.

To avoid this you can do the below

`git checkout -b temporay-test-branch commit_SHA1_hash_reference`

ex --> `git checkout -b temp-test-branch 56a4e5c08`

so it will create a temporary branch and copy the commit into that branch, then you do your work. once your work is complete you can delete the branch.

In case of "rebase" git temporary detached HEAD.

**How to checkout particular commit/snapshot into a new branch?**

`git checkout -b temporay-test-branch commit_SHA1_hash_reference`

ex --> `git checkout -b temp-test-branch 56a4e5c08`

so it will create a temporary branch and copy the commit into that branch, then you do your

work. once your work is complete you can delete the branch.

**What is the difference between git reset and git revert?**

git revert creates a new commit which undoes the changes from last commit.

git reset depends on the usage, can modify the index or change the commit which the branch head is currently pointing at.

**How can I undo the last commit?**

`git reset --soft HEAD~1` # when use the soft parameter you can undo the last commit and the and you'll find the changes as uncommitted local modifications in your working copy.

`git reset --hard HEAD~1` # when use the hard parameter you can undo the last commit and delete the present commit.

`git reset --hard commit_hash` # when use the hard parameter with commit\_hash you can undo the commit till the commit\_hash and delete all other commit after the commit\_hash.

How can revert back / redo an undo commit?

`git revert commit_hash`

How to revert the last n number of commits?

`git revert HEAD~n...HEAD`

`git revert HEAD~3...HEAD` ---- revert the last three commits

How can I ignore files that have already been committed to the repo?

when you create a new repository, you should also create a ".gitignore" file with all the file name you want to ignore.

`git reset filename` # or

`echo filename >> .gitignore` # add it to .gitignore to avoid re-adding it

how to save the changes Temporarily or what is git stash?

suppose you're in the middle of some changes for a feature and some important bug has reported.

In that case you dont want to commit your half finished work but you want to save it in your local repository. so that once you fix the bug you can start half finished work.

In this case we can use "git stash"

To get the list of current Stashes we can use "git stash list"

When to use Stash?

before checking out a different branch.

before pulling remote changes.

before merging or rebasing a branch.

how to restore back to the saved stash?

`git stash pop` - Restore back to the saved state, but it deletes the stash from the temporary storage.

`git stash apply stash_name` - Restore back to the saved state and leaves the stash list for possible later reuse.

`git stash drop stash_name` - it delete the stash

What is "git cherry-pick"?

The "git cherry-pick" is use to push particular commits from one branch within a repository to a different branch.

`git cherry-pick <commit-hash>`

a - b - c - d Main

\

e - f - g Feature

`git checkout main`

`git cherry-pick f` -----> the commit (f) will come to master branch

a - b - c - d - f Main

\

e - f - g Feature

ex ----

```
git checkout div
git logs -3 --format=format:"%H" { only show last 3 commit hash
    e1fd3ca3a ,    b1aefec53 , c53efec53
git checkout master
git cherry-pick b1aefec53 ----- this commit go to master branch
```

How to delete github repository?

On GitHub, navigate to the main page of the repository.  
Under your repository name, click Settings.  
Under Danger Zone, click Delete this repository.

How to Discard all local changes in your working directory?

```
git reset -hard HEAD
```

How to discard local changes in a specific file?

```
git checkout HEAD
```

How do you discard local file changes? (before commit)

```
git checkout -- <file_name>
```

How to Reset your HEAD pointer to a previous commit and discard all changes since then?

```
git reset --hard
```

How to squash last two commits? ( "Squash" means combining multiple commits into one.)

```
git reset --soft HEAD~2      or
git rebase -i HEAD~2
```

In what situations are you using git rebase?

Suppose a team is working on a `feature` branch that is coming from the `main` branch of the repo. At a point, where the feature development is done, and finally we wish to merge the feature branch into the main branch without keeping the history of the commits made in the feature branch, a `git rebase` will be helpful.

how to rebase the branch?

inside the your branch

```
A---B---C (master)
```

```
\
```

```
D---E (my-feature)
```

```
git rebase master
```

```
A---B---C (master)
```

```
\
```

```
D'---E' (my-feature rebased) ----> now feature branch updated
```

inside the master branch

```
git rebase beanch_name
```

How to Cleaning Ignored Files?

```
git rm -r --cached .
```

```
git add .
```

```
git commit -m "Clean up ignored files"
```

**How to remove tracked files from the Git index?**

**git rm**

The git rm command operates on the current branch only.

**How to remove untracked file from git?**

**git clean -ndx** -----list out the files which could remove

**git clean -fxd** ----- remove the files { *-d* allow to delete directory , *-x* delete files