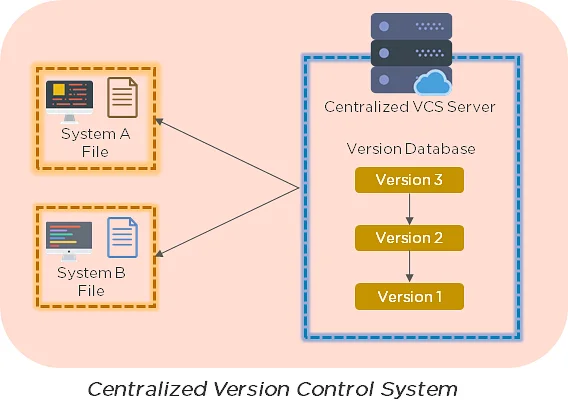
**Git Interview Question & Top Commands**

Explain the difference between a centralized and distributed version control system (VCS)

Centralized Version Control System

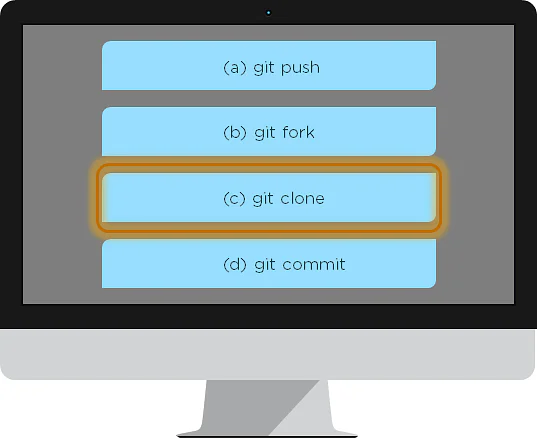
* All file versions are stored on a central server
* No developer has a copy of all files on a local system
* If the central server crashes, all data from the project will be lost



Distributed Control System

* Every developer has a copy of all versions of the code on their systems
* Enables team members to work offline and does not rely on a single location for backups
* There is no threat, even if the server crashes

What is the git command that downloads any repository from GitHub to your computer?



The git command that downloads any repository from GitHub to your computer is git clone.

Q. How do you push a file from your local system to the GitHub repository using Git?

First, connect the local repository to your remote repository:

git remote add origin [copied web address]

// Ex: git remote add origin https://github.com/Simplilearn-github/test.git

Second, push your file to the remote repository:

git push origin master

Q. How is a bare repository different from the standard way of initializing a Git repository?

Using the standard method:

git init

You create a working directory with git init

A .git subfolder is created with all the git-related revision history

Using the bare way

git init --bare

It does not contain any working or checked out a copy of source files

Bare repositories store git revision history in the root folder of your repository, instead of the .git subfolder

Q. Which of the following CLI commands can be used to rename files?

git rm

git mv

git rm -r

None of the above

The correct answer is B) git mv

Q. What is the process for reverting a commit that has already been pushed and made public?

There are two ways that you can revert a commit:

Remove or fix the bad file in a new commit and push it to the remote repository. Then commit it to the remote repository using:

git commit –m "commit message"

Create a new commit that undoes all the changes that were made in the bad commit. Use the following command:

git revert <commit id>

Example: git revert 56de0938f

Explain the difference between git fetch and git pull.

Git fetch Git pull

Git fetch only downloads new data from a remote repository Git pull updates the current HEAD branch with the latest changes from the remote server

Does not integrate any new data into your working files Downloads new data and integrate it with the current working files

Users can run a Git fetch at any time to update the remote-tracking branches Tries to merge remote changes with your local ones

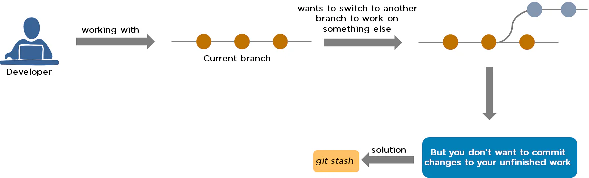
Command - git fetch origin

git fetch –-all

Command - git pull origin master

Q. What is Git stash?

A developer working with a current branch wants to switch to another branch to work on something else, but the developer doesn't want to commit changes to your unfinished work. The solution to this issue is Git stash. Git stash takes your modified tracked files and saves them on a stack of unfinished changes that you can reapply at any time.



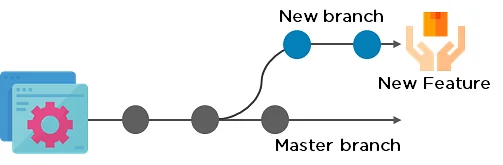
Q. Explain the concept of branching in Git.

Suppose you are working on an application, and you want to add a new feature to the app. You can create a new branch and build the new feature on that branch.

By default, you always work on the master branch

The circles on the branch represent various commits made on the branch

After you are done with all the changes, you can merge it with the master branch



Q.What is the difference between Git Merge and Git Rebase?

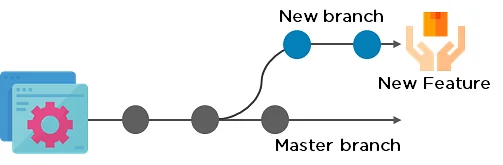
Suppose you are working on a new feature in a dedicated branch, and another team member updates the master branch with new commits. You can use these two functions:

Git Merge

To incorporate the new commits into your feature branch, use Git merge.

Creates an extra merge commit every time you need to incorporate changes

But, it pollutes your feature branch history

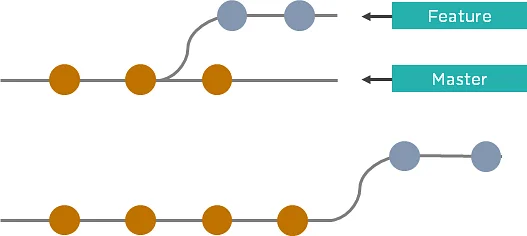


Git Rebase

As an alternative to merging, you can rebase the feature branch on to master.

Incorporates all the new commits in the master branch

It creates new commits for every commit in the original branch and rewrites project history



Q. How do you find a list of files that have been changed in a particular commit?

The command to get a list of files that have been changed in a particular commit is:

git diff-tree –r {commit hash}

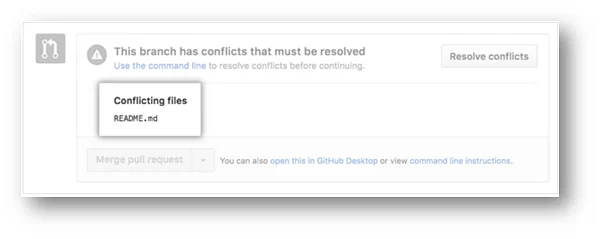
Example: git diff-tree –r 87e673f21b

-r flag instructs the command to list individual files

commit hash will list all the files that were changed or added in that commit

Q. What is a merge conflict in Git, and how can it be resolved?

A Git merge conflict happens when you have merge branches with competing for commits, and Git needs your help to decide which changes to incorporate in the final merge.



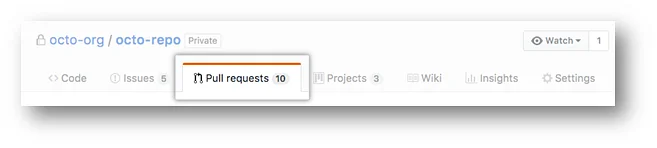
Manually edit the conflicted file to select the changes that you want to keep in the final merge.

Resolve using GitHub conflict editor

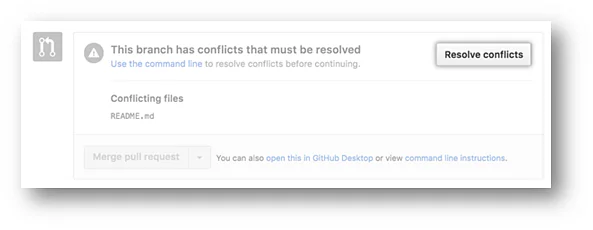
This is done when a merge conflict is caused after competing for line changes. For example, this may occur when people make different changes to the same line of the same file on different branches in your Git repository.

Resolving a merge conflict using conflict editor:

Under your repository name, click "Pull requests."

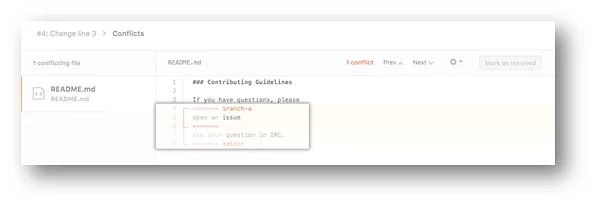


In the "Pull requests" drop-down, click the pull request with a merge conflict that you'd like to resolveNear the bottom of your pull request, click "Resolve conflicts."



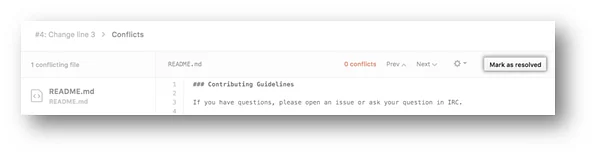
Decide if you only want to keep your branch's changes, the other branch's changes, or make a brand new change, which may incorporate changes from both branches.

Delete the conflict markers <<<<<<<, =======, >>>>>>> and make changes you want in the final merge.

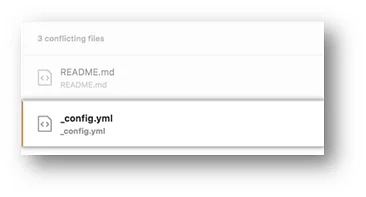


If you have more than one merge conflict in your file, scroll down to the next set of conflict markers and repeat steps four and five to resolve your merge conflict.

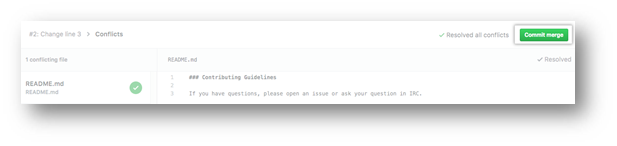
Once you have resolved all the conflicts in the file, click Mark as resolved.



If you have more than one file with a conflict, select the next file you want to edit on the left side of the page under "conflicting files" and repeat steps four to seven until you've resolved all of your pull request's merge conflicts.



Once you've resolved your merge conflicts, click Commit merge. This merges the entire base branch into your head branch.

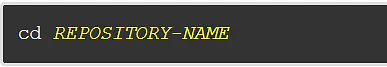


To merge your pull request, click Merge pull request.

A merge conflict is resolved using the command line.

Open Git Bash.

Navigate into the local Git repository that contains the merge conflict.



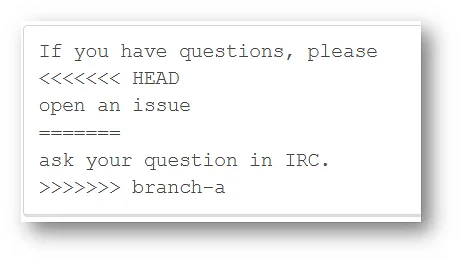
Generate a list of the files that the merge conflict affects. In this example, the file styleguide.md has a merge conflict.



Open any text editor, such as Sublime Text or Atom, and navigate to the file that has merge conflicts.

To see the beginning of the merge conflict in your file, search the file for the conflict marker "<<<<<<<. " Open it, and you'll see the changes from the base branch after the line "<<<<<<< HEAD."

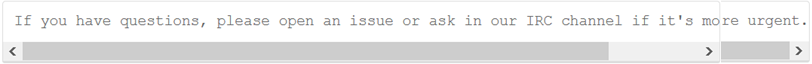
Next, you'll see "=======", which divides your changes from the changes in the other branch, followed by ">>>>>>> BRANCH-NAME".



Decide if you only want to keep your branch's changes, the other branch's changes, or make a brand new change, which may incorporate changes from both branches.

Delete the conflict markers "<<<<<<<", "=======", ">>>>>>>" and make the changes you want in the final merge.

In this example, both the changes are incorporated into the final merge:



Add or stage your changes. $ git add .

Commit your changes with a comment



Q. What is Git bisect? How can you use it to determine the source of a (regression) bug?

Git bisect is a tool that uses binary search to locate the commit that triggered a bug.

Git bisect command -

git bisect <subcommand> <options>

The git bisect command is used in finding the bug performing commit in the project by using a binary search algorithm.

The bug occurring commit is called the “bad” commit, and the commit before the bug occurring one is called the “good” commit. We convey the same to the git bisect tool, and it picks a random commit between the two endpoints and prompts whether that one is the “good” or “bad” one. The process continues uptil the range is narrowed down and the exact commit that introduced the exact change is discovered.

Q. Explain some basic Git commands.

Some of the Basic Git Commands are summarized in the below table –

|  |  |
| --- | --- |
| Command | Purpose |
| git init | Used to start a new repository. |
| git config -   * git config –global user.name “[name]” * git config –global user.email “[email address]” | This helps to set the username and email to whom the commits belong to. |
| git clone <repository path> | Used to create a local copy of an existing repository. |
| git add -   * git add <file names separated by commas> * git add . | Used to add one or more files to the staging area. |
| git commit -   * git commit -a * git commit -m “<add commit message>” | Creates a snapshot or records of the file(s) that are in the staging area. |
| git diff -   * git diff [first branch] [second branch] * git diff -staged | Used to show differences between the two mentioned branches/differences made in the files in the staging area vs current version. |
| git status | Lists out all the files that are to be committed. |
| git rm <file name(s)> | Used to delete a file(s) from the current working directory and also stages it. |
| git show <commit> | Shows the content changes and metadata of the mentioned commit. |
| git branch -   * git branch [branch name] * git branch -d [branch name] * git branch | The first one creates a brand new branch.  The second is used to delete the mentioned branch.  The last one lists out all the branches available and also highlights the branch we are in currently. |

# Top 20 Git Commands With Examples

Here are the Git commands which are being covered:

* **git config**
* **git init**
* **git clone**
* **git add**
* **git commit**
* **git diff**
* **git reset**
* **git status**
* **git rm**
* **git log**
* **git show**
* **git tag**
* **git branch**
* **git checkout**
* **git merge**
* **git remote**
* **git push**
* **git pull**
* **git stash**

### git config

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

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

### git init

Usage: git init [repository name]

### git clone

Usage: git clone [url]

### git add

Usage: git add [file]

Usage: git add \*

### git commit

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

Usage: git commit -a

### git diff

Usage: git diff

Usage: git diff –staged

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

### git reset

Usage: git reset [file]

Usage: git reset [commit]

Usage: git reset –hard [commit]

### git status

Usage: git status

### git rm

Usage: git rm [file]

### git log

Usage: git log

Usage: git log –follow[file]

### git show

Usage: git show [commit]

### git tag

Usage: git tag [commitID]

### git branch

Usage: git branch

Usage: git branch [branch name]

Usage: git branch -d [branch name]

### git checkout

Usage: git checkout [branch name]

Usage: git checkout -b [branch name]

### git merge

Usage: git merge [branch name]

### git remote

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

### git push

Usage: git push [variable name] master

Usage: git push [variable name] [branch]

Usage: git push –all [variable name]

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

### git pull

Usage: git pull [Repository Link]

### git stash

Usage: git stash save

Usage: git stash pop

Usage: git stash list

Usage: git stash drop