**Git interview question and Answers.**

1. **What are the fundamental differences between DevOps & Agile?**

The differences between the two are listed down in the table below.

**DevOps** **Agile**

1)Agility in both Development & Operations 1)Agility in only Development

2)Involves processes such as CI, CD, CT, etc 2)Involves practices such as Agile Scrum,Agil Kanban,etc.

3)Timeliness & quality have equal priority 3)Timeliness is the main priority

4)Smaller release cycles with immediate feedback 4)Smaller release cycles

5)Feedback is from self (Monitoring tools) 5)Feedback is from customers

6)Agility & need for Automation 6)Agility Only

1. **What is the need for DevOps?**

This has many advantages like quick feedback from customers, better quality of software etc. which in turn leads to high customer satisfaction. To achieve this,

**companies are required to:**

--Increase deployment frequency

--Lower failure rate of new releases

--Shortened lead time between fixes

--Faster mean time to recovery in the event of new release crashing

DevOps fulfills all these requirements and helps in achieving seamless software delivery. You can give examples of companies like Etsy, Google and Amazon which have adopted DevOps to achieve levels of performance that were unthinkable even five years ago. They are doing tens, hundreds or even thousands of code deployments per daywhile delivering world class stability, reliability and security.

1. **How do all these tools work together?**

this flow may vary from organization to organization as per the requirement.

--Developers develop the code and this source code is managed by Version Control System tools like Git etc.

--Developers send this code to the Git repository and any changes made in the code is committed to this Repository.

--Jenkins pulls this code from the repository using the Git plugin and build it using tool like Ant or Maven.

--Configuration management tools like puppet deploys & provisions testing environment and then Jenkins releases this code on the test environment on which testing is done using tools like selenium.

--Once the code is tested, Jenkins send it for deployment on the production server (even production server is provisioned & maintained by tools like puppet).

--After deployment It is continuously monitored by tools like Nagios.

--Docker containers provides testing environment to test the build features.

1. **What are the advantages of DevOps?**

**Technical benefits:**

--Continuous software delivery

--Less complex problems to fix

--Faster resolution of problems

**Business benefits:**

--Faster delivery of features

--More stable operating environments

--More time available to add value (rather than fix/maintain)

**Q) What is Version control?**

With the help of GIT version control, you can track the history of a collection of files and includes the functionality to revert the collection of files to another version. Each version captures a snapshot of the file system at a certain point of time. A collection of files and their complete history are stored in a repository.

Version control systems consist of a central shared repository where teammates can commit changes to a file or set of file. Then you can mention the uses of version control.

Version control allows you to:

--Revert files back to a previous state.

--Revert the entire project back to a previous state.

--Compare changes over time.

--See who last modified something that might be causing a problem.

--Who introduced an issue and when.

**Q) What are the benefits of using version control?**

With Version Control System (VCS), all the team members are allowed to work freely on any file at any time. VCS will later allow you to merge all the changes into a common version.

--All the past versions and variants are neatly packed up inside the VCS. When you need it, you can request any version at any time and you’ll have a snapshot of the complete project right at hand.

--Every time you save a new version of your project, your VCS requires you to provide a short description of what was changed. Additionally, you can see what exactly was changed in the file’s content. This allows you to know who has made what change in the project.

--A distributed VCS like Git allows all the team members to have complete history of the project so if there is a breakdown in the central server you can use any of your teammate’s local Git repository.

**Q) What is Git ?**

**Git** is a Distributed Version Control system (DVCS). It can track changes to a file and allows you to revert back to any particular change.

--Its distributed architecture provides many advantages over other Version Control Systems (VCS) like SVN one major advantage is that it does not rely on a central server to store all the versions of a project’s files. Instead, every developer “clones” a copy of a repository with “Local repository” and has the full history of the project on his hard drive so that when there is a server outage, all you need for recovery is one of your teammate’s local Git repository here is a central cloud repository as well where developers can commit changes and share it with other teammates as you can see in the diagram where all collaborators are commiting changes “Remote repository”.

**Q) Explain some basic Git commands?**

git config --global user.name "name config the author name and email adress to be used with git config --global user.email "E-mail" your commit

git init create a new local repo

git clone/path/to/repository create a working copy a local repo

git clone username@host:/path/to/repo for a remote server,use

git add <filename> Add one or more file to staging

git add\*

git commit -m "commit message" commit changes to head

git commit -a commit ant files you have added with git add and also a commit any files you have changed

git push origin master send changes to the master branch of your remote repo

git status List the files you have changed and those you still need to add or commit

git remote add origin <server> if you have not connected your local repo to a remote server, add the server to be able to push to it

**Q) What are the advantages of using GIT?**

a) Data redundancy and replication

b) High availability

c) Only one git directory per repository

d) Superior disk utilization and network performance

e) Collaboration friendly

f) Any sort of projects can use GIT

**Q) What is ‘staging area’ or ‘index’ in Git?**

That before completing the commits, it can be formatted and reviewed in an intermediate area known as ‘Staging Area’ or ‘Index’. From the diagram it is evident that every change is first verified in the staging area I have termed it as “stage file” and then that change is committed to the repository.

**Q) What is the purpose of branching in GIT?**

The purpose of branching in GIT is that you can create your own branch and jump between those branches. It will allow you to go to your previous work keeping your recent work intact.

**Q) What is the common branching pattern in GIT?**

The common way of creating branch in GIT is to maintain one as “Main“ branch and create another branch to implement new features. This pattern is particularly useful when there are multiple developers working on a single project.

**Q) How can you bring a new feature in the main branch?**

To bring a new feature in the main branch, you can use a command “git merge” or “git pull command”.

**Q) To delete a branch what is the command that is used?**

Once your development branch is merged into the main branch, you don’t need development branch. To delete a branch use, the command

“git branch –d [head]”.

**Q) How will you know in Git if a branch has already been merged into master?**

#**git branch –merged** 🡪 lists the branches that have been merged into the current branch.

#**git branch –no-merged** 🡪lists the branches that have not been merged.

**Q) Describe branching strategies you have used.**

**Feature branching**:--A feature branch model keeps all of the changes for a particular feature inside of a branch. When the feature is fully tested and validated by automated tests, the branch is then merged into master.

**Task branching**:--In this model each task is implemented on its own branch with the task key included in the branch name. It is easy to see which code implements which task, just look for the task key in the branch name.

**Release branching:--**Once the develop branch has acquired enough features for a release, you can clone that branch to form a Release branch. Creating this branch starts the next release cycle, so no new features can be added after this point, only bug fixes, documentation generation, and other release-oriented tasks should go in this branch. Once it is ready to ship, the release gets merged into master and tagged with a version number. In addition, it should be merged back into develop branch, which may have progressed since the release was initiated.

In the end tell them that branching strategies varies from one organization to another, so I know basic branching operations like delete, merge, checking out a branch etc.

**Q) In Git how do you Revert a commit that has already been pushed and made public?**

Remove or fix the bad file in a new commit and push it to the remote repository. This is the most natural way to fix an error. Once you have made necessary changes to the file, commit it to the remote repository for that I will use

#git commit -m “commit message”

Create a new commit that undoes all changes that were made in the bad commit.to do this I will use a command

#git revert <name of bad commit>

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

Command for Git bisect is

#git bisect <subcommand> <options>

This command uses a binary search algorithm to find which commit in your project’s history introduced a bug. You use it by first telling it a “bad” commit that is known to contain the bug, and a “good” commit that is known to be before the bug was introduced. Then Git bisect picks a commit between those two endpoints and asks you whether the selected commit is “good” or “bad”. It continues narrowing down the range until it finds the exact commit that introduced the change.

**Q) What is a ‘conflict’ in git?**

A ‘conflict’ arises when the commit that has to be merged has some change in one place, and the current commit also has a change at the same place. Git will not be able to predict which change should take precedence.

**Q) How can conflict in git resolved?**

To resolve the conflict in git, edit the files to fix the conflicting changes and then add the resolved files by running “git add” after that to commit the repaired merge, run “git commit”. Git remembers that you are in the middle of a merger, so it sets the parents of the commit correctly.

**Q) What is another option for merging in git?**

“Rebasing” is an alternative to merging in git.

**Q) What is the syntax for “Rebasing” in Git?**

The syntax used for rebase is “git rebase [new-commit] “

**Q) What is Git rebase and how can it be used to resolve conflicts in a feature branch before merge?**

According to me, git rebase is a command which will merge another branch into the branch where you are currently working, and move all of the local commits that are ahead of the rebased branch to the top of the history on that branch.

Git rebase command can be used to resolve conflicts in a feature branch before merge, if a feature branch was created from master, and since then the master branch has received new commits, Git rebase can be used to move the feature branch to the tip of master.

The command effectively will replay the changes made in the feature branch at the tip of master, allowing conflicts to be resolved in the process. When done with care, this will allow the feature branch to be merged into master with relative ease and sometimes as a simple fast-forward operation.

**Q) How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?**

A sanity or smoke test determines whether it is possible and reasonable to continue testing.

This can be done with a simple script related to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you are required to enter a commit message. In this script one can run other tools, such as linters and perform sanity checks on the changes being committed into the repository.

**Q) How do you find a list of files that has changed in a particular commit?**

To get a list files that has changed in a particular commit use command

#git diff-tree -r {hash}

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names only.

You can also include the below mention point although it is totally optional but will help in impressing the interviewer.

The output will also include some extra information, which can be easily suppressed by including two flags:

#git diff-tree –no-commit-id –name-only -r {hash}

Here –no-commit-id will suppress the commit hashes from appearing in the output, and –name-only will only print the file names, instead of their paths.

**Q) How do you setup a script to run every time a repository receives new commits through push?**

There are three ways to configure a script to run every time a repository receives new commits through push, one needs to define either a pre-receive, update, or a post-receive hook depending on when exactly the script needs to be triggered.

--Pre-receive hook in the destination repository is invoked when commits are pushed to it. Any script bound to this hook will be executed before any references are updated. This is a useful hook to run scripts that help enforce development policies.

--Update hook works in a similar manner to pre-receive hook, and is also triggered before any updates are actually made. However, the update hook is called once for every commit that has been pushed to the destination repository.

-- post-receive hook in the repository is invoked after the updates have been accepted into the destination repository. This is an ideal place to configure simple deployment scripts, invoke some continuous integration systems, dispatch notification emails to repository maintainers, etc.

Hooks are local to every Git repository and are not versioned. Scripts can either be created within the hooks directory inside the “.git” directory, or they can be created elsewhere and links to those scripts can be placed within the directory.

**Q) Why GIT better than Subversion?**

GIT is an open source version control system; it will allow you to run ‘versions’ of a project, which show the changes that were made to the code overtime also it allows you keep the backtrack if necessary and undo those changes. Multiple developers can check-out, and upload changes and each change can then be attributed to a specific developer.

**Q) What is the difference between Git and SVN?**

Git SVN

1. Git is a Decentralized Version Control 1. SVN is a Centralized Version Control tool

2. It belongs to the 3rd generation of Version 2. It belongs to the 2nd generation of version

Control tools Control tools

3. Clients can clone entire repositories on their 3. Version history is stored on a server-side

local systems repository

4. Commits are possible even if offline 4. Only online commits are allowed

5. Push/pull operations are faster 5. Push/pull operations are slower

6. Works are shared automatically by commit 6. Nothing is shared automatically

**Q) What is Git ?**

Git is a Distributed Version Control system (DVCS). It can track changes to a file and allows you to revert back to any particular change.

Its distributed architecture provides many advantages over other Version Control Systems (VCS) like SVN one major advantage is that it does not rely on a central server to store all the versions of a project’s files.

Instead, every developer “clones” a copy of a repository developer can access with “Local repository” and has the full history of the project on his hard drive so when there is a server outage all you need for recovery is one of your teammate’s local Git repository.

--There is a central cloud repository as well where developers can commit changes and share it with other teammates as you can see in the diagram where all collaborators are commiting changes “Remote repository”.

**Q) What is the command to write a commit message in Git?**

Command that is used to write a commit message is

“git commit -a”

line instructs git to commit the new content of all tracked files that have been modified. Also we can use “git add <file>” before git commit -a if new files need to be committed for the first time.

**Q) What is a repository in GIT?**

A repository contains a directory named .git, where git keeps all of its metadata for the repository. The content of the .git directory are private to git.

**Q) What is ‘bare repository’ in Git?**

A “bare” repository in Git just contains the version control information and no working files (no tree) and it doesn’t contain the special .git sub-directory. Instead, it contains all the contents of the .git sub-directory directly in the main directory itself, where as working directory consist of:

--A .git subdirectory with all the Git related revision history of your repo.

--A working tree, or checked out copies of your project files.

**Q) What language is used in Git?**

Git uses ‘C’ language. GIT is fast, and ‘C’ language makes this possible by reducing the overhead of run times associated with high level languages.

**Q) What is the difference between git pull and git fetch?**

**Git pull** command pulls new changes or commits from a particular branch from your central repository and updates your target branch in your local repository.

**Git fetch** is also used for the same purpose but it works in a slightly different way. When you perform a git fetch, it pulls all new commits from the desired branch and stores it in a new branch in your local repository. If you want to reflect these changes in your target branch, git fetch must be followed with a git merge. Your target branch will only be updated after merging the target branch and fetched branch. Just to make it easy for you, remember the equation below:

Git pull = git fetch + git merge

**Q) What is Git stash?**

when you’ve been working on part of your project, things are in a messy state and you want to switch branches for sometime to work on something else. The problem is, you don’t want to do a commit of half-done work just so you can get back to this point later. The answer to this issue is Git stash. Stashing takes your working directory that is, your modified tracked files and staged changes and saves it on a stack of unfinished changes that you can reapply at any time.

**Q) what purpose we use Git ‘stash drop’ ?**

Git ‘stash drop’ command is used to remove the stashed item. It will remove the last added stash item by default, and it can also remove a specific item if you include it as an argument.

If you want to remove a particular stash item from the list of stashed items you can use the below commands:

#git stash list: It will display the list of stashed items like:

stash@{0}: WIP on master: 049d078 added the index file

stash@{1}: WIP on master: c264051 Revert “added file\_size”

stash@{2}: WIP on master: 21d80a5 added number to log

If you want to remove an item named stash@{0} use command git stash drop stash@{0}.

**Q) What is the function of ‘git stash apply’?**

When you want to continue working where you have left your work, ‘git stash apply’ command is used to bring back the saved changes onto the working directory.

Git stash commands

git stash save ----> all modification fill will moved to stash, and it will show in staging or working area

git stash save <file> --> only mention files will move to stash

git stash list --> list of stash files

git status

git stash pop ---> it will remove all stash but u will see files in staging and workig area

git stash pop <Stash{0}> --> it will remove only mention stash

git stash apply -----> it will apply latest stash

git stash apply <stash{1}> --> it will apply only mention stash

git stash show <stash> --> breaf details of stash

git stash show -p <stash> --> completed details of stash

git stash apply --> modified file will see in staging ad Working area

git stash apply stash@{2} ---> it will back only mention stash

git stash apply --index --> option is used, then tries to reinstate not only the working tree’s changes, but also the index’s ones.

git stash drop ---> it will remove all stash

git stash drop >stash> ---> it will remove stash and remove files from staging and working area

git stash branch testchanges -->

**Q) How to Creating a Branch from a Stash ?**

If you stash some work, leave it there for a while, and continue on the branch from which you stashed the work, you may have a problem reapplying the work. If the apply tries to modify a file that you’ve since modified, you’ll get a merge conflict and will have to try to resolve it. If you want an easier way to test the stashed changes again, you can run git stash branch, which creates a new branch for you, checks out the commit you were on when you stashed your work, reapplies your work there, and then drops the stash if it applies successfully:

**Q) What is the function of ‘git diff ’ in git?**

‘git diff ’ shows the changes between commits, commit and working tree etc.

**Q) What is the function of ‘GIT PUSH’ in GIT?**

‘GIT PUSH’ updates remote refs along with associated objects.

**Q) What is the difference between ‘git remote’ and ‘git clone’?**

‘git remote add’ just creates an entry in your git config that specifies a name for a particular URL. While, ‘git clone’ creates a new git repository by copying and existing one located at the URI.

**Q) What is the function of git clone?**

The git clone command creates a copy of an existing Git repository. To get the copy of a central repository, ‘cloning’ is the most common way used by programmers.

**Q) What is the function of ‘git config’?**

Git uses your username to associate commits with an identity. The git config command can be used to change your Git configuration, including your username. –example: Suppose you want to give a username and email id to associate commit with an identity so that you can know who has made a particular commit. For that I will use:

#git config –global user.name “Your Name”: This command will add username.

#git config –global user.email “Your E-mail Address”: This command will add email id.

**Q) What does commit object contains?**

Commit object contains the following components, three points present below:

--A set of files, representing the state of a project at a given point of time

--Reference to parent commit objects

--An SHAI name, a 40 character string that uniquely identifies the commit object.

**Q) How can you create a repository in Git?**

To create a repository, create a directory for the project if it does not exist, then run command “git init”. By running this command .git directory will be created in the project directory.

**Q) What is ‘head’ in git and how many heads can be created in a repository?**

A ‘head’ is simply a reference to a commit object. In every repository, there is a default head referred as “Master”. A repository can contain any number of heads.

**Q) What is Sub-Git ?**

.SubGit is a tool for SVN to Git migration. It creates a writable Git mirror of a local or remote Subversion repository and uses both Subversion and Git as long as you like.Now you can include some advantages like you can do a fast one-time import from Subversion to Git or use SubGit within Atlassian Bitbucket Server.We can use SubGit to create a bi-directional Git-SVN mirror of existing Subversion repository. You can push to Git or commit to Subversion at your convenience. Synchronization will be done by SubGit.

**Q) Mention some of the best graphical GIT client for LINUX?**

Some of the best GIT client for LINUX is

a) Git Cola

b) Git-g

c) Smart git

d) Giggle

e) Git GUI

f) qGit

**Q) What is ‘git status’ is used for?**

As ‘Git Status’ shows you the difference between the working directory and the index, it is helpful in understanding a git more comprehensively.

**Q) What is the difference between the ‘git diff ’and ‘git status’?**

‘git diff’ is similar to ‘git status’, but it shows the differences between various commits and also between the working directory and index.

**Q) What is the function of ‘git checkout’ in git?**

A ‘git checkout’ command is used to update directories or specific files in your working tree with those from another branch without merging it in the whole branch.

**Q) What is the function of ‘git rm’?**

To remove the file from the staging area and also off your disk ‘git rm’ is used.

**Q) What is the use of ‘git log’?**

To find specific commits in your project history- by author, date, content or history ‘git log’ is used.

**Q) What is ‘git add’ is used for?**

‘git add’ adds file changes in your existing directory to your index.

**Q) What is the function of ‘git reset’?**

The function of ‘Git Reset’ is to reset your index as well as the working directory to the state of your last commit.

**Q) What is git Is-tree?**

‘git Is-tree’ represents a tree object including the mode and the name of each item and the SHA-1 value of the blob or the tree.

**Q) How git instaweb is used?**

‘Git Instaweb’ automatically directs a web browser and runs webserver with an interface into your local repository.

**Q) What does ‘hooks’ consist of in git?**

This directory consists of Shell scripts which are activated after running the corresponding Git commands. For example, git will try to execute the post-commit script after you run a commit.

**Q) Explain what is commit message?**

Commit message is a feature of git which appears when you commit a change. Git provides you a text editor where you can enter the modifications made in commits.

**Q) How can you fix a broken commit?**

To fix any broken commit, you will use the command “git commit—amend”. By running this command, you can fix the broken commit message in the editor.

**Q) Why is it advisable to create an additional commit rather than amending an existing commit?**

There are couple of reason

a) The amend operation will destroy the state that was previously saved in a commit. If it’s just the commit message being changed then that’s not an issue. But if the contents are being amended then chances of eliminating something important remains more.

b) Abusing “git commit- amend” can cause a small commit to grow and acquire unrelated changes.

**Q) How do you find a list of files that has changed in a particular commit?**

For this answer instead of just telling the command, explain what exactly this command will do.

To get a list files that has changed in a particular commit use the below command:

git diff-tree -r {hash}

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names only..

The output will also include some extra information, which can be easily suppressed by including two flags:

git diff-tree –no-commit-id –name-only -r {hash}

Here –no-commit-id will suppress the commit hashes from appearing in the output, and –name-only will only print the file names, instead of their paths.

**Q) How do you squash last N commits into a single commit?**

• If you want to write the new commit message from scratch use the following command

git reset –soft HEAD~N &&

git commit

• If you want to start editing the new commit message with a concatenation of the existing commit messages then you need to extract those messages and pass them

to Git commit for that I will use

git reset –soft HEAD~N &&

git commit –edit -m”$(git log –format=%B –reverse .HEAD@{N})”

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

Git bisect is used to find the commit that introduced a bug by using binary search. Command for Git bisect is

git bisect <subcommand> <options>

Now since you have mentioned the command above explain them what this command will do.

This command uses a binary search algorithm to find which commit in your project’s history introduced a bug. You use it by first telling it a “bad” commit that is known to contain the bug, and a “good” commit that is known to be before the bug was introduced. Then Git bisect picks a commit between those two endpoints and asks you whether the selected commit is “good” or “bad”. It continues narrowing down the range until it finds the exact commit that introduced the change.

**Q) How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?**

A sanity or smoke test determines whether it is possible and reasonable to continue testing.

This can be done with a simple script related to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you

are required to enter a commit message. In this script one can run other tools, such as linters and perform sanity checks on the changes being committed into the

repository.

Finally, give an example, you can refer the below script:

#!/bin/sh

files=$(git diff –cached –name-only –diff-filter=ACM | grep ‘.go$’)

if [ -z files ]; then

exit 0

fi

unfmtd=$(gofmt -l $files)

if [ -z unfmtd ]; then

exit 0

fi

echo “Some .go files are not fmt’d”

exit 1

This script checks to see if any .go file that is about to be committed needs to be passed through the standard Go source code formatting tool gofmt. By exiting with a non-zero status, the script effectively prevents the commit from being applied to the repository.

**Q) Name a few Git repository hosting services**

• Pikacode

• Visual Studio Online

• GitHub

• GitEnterprise

• SourceForge.net

**Q) How to get back changes after Git stash ?**

After you're all done with your changes—the apply looks good and you're sure you don't need the stash any more—*then* use git stash drop to get rid of it.

I always suggest using git stash apply rather than git stash pop. The difference is that apply leaves the stash around for easy re-try of the apply, or for looking at, etc. If pop is able to extract the stash, it will immediately also drop it, and if you the suddenly realize that you wanted to extract it somewhere else (in a different branch), or with --index, or some such, that's not so easy. If you apply, *you* get to choose when to drop.

**Q) What is Difference between git revert and git reset ?**

**Reset** - On the commit-level, resetting is a way to move the tip of a branch to a different commit. This can be used to remove commits from the current branch.

**Revert** - Reverting undoes a commit by creating a new commit. This is a safe way to undo changes, as it has no chance of re-writing the commit history. Contrast this with git reset, which does alter the existing commit history. For this reason, git revert should be used to undo changes on a public branch, and git reset should be reserved for undoing changes on a private branch.

* git revert is used to undo a previous commit. In git, you can't alter or erase an earlier commit. (Actually you can, but it can cause problems.) So instead of editing the earlier commit, revert introduces a new commit that reverses an earlier one.
* git reset is used to undo changes in your working directory that haven't been comitted yet.
* git checkout is used to copy a file from some other commit to your current working tree. It doesn't automatically commit the file.
* git checkout modifies your working tree,
* git reset modifies which reference the branch you're on points to,
* git revert adds a commit undoing changes.
  1. **Diff BW Git clone and git pull ?**

git clone is how you get a local copy of an existing repository to work on. It's usually only used once for a given repository, unless you want to have multiple working copies of it around. (Or want to get a clean copy after messing up your local one...)

git pull (or git fetch + git merge) is how you update that local copy with new commits from the remote repository. If you are collaborating with others, it is a command that you will run frequently.

As your first example shows, it is possible to emulate git clone with an assortment of other git commands, but it's not really the case that git pull is doing "basically the same thing" as git clone (or vice-versa)

**Q) Diff BW Git pull and Git Featch ?**

git pull, in contrast, is used with a different goal in mind: to update your current HEAD branch with the latest changes from the remote server. This means that pull not only downloads new data; it also directly integrates it into your current working copy files. This has a couple of consequences:

* Since "git pull" tries to merge remote changes with your local ones, a so-called "merge conflict" can occur. Check out our in-depth tutorial on [How to deal with merge conflicts](https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/merge-conflicts) for more information.
* Like for many other actions, it's highly recommended to start a "git pull" only with a clean working copy. This means that you should not have any uncommitted local changes before you pull. Use Git's Stash feature to [save your local changes temporarily](https://www.git-tower.com/learn/git/ebook/en/command-line/branching-merging/stashing).

**git fetch** really only downloads new data from a remote repository - but it doesn't integrate any of this new data into your working files. Fetch is great for getting a fresh view on all the things that happened in a remote repository.  
Due to it's "harmless" nature, you can rest assured: fetch will never manipulate, destroy, or screw up anything. This means you can never fetch often enough.

**Q)What is Git Fork ?**

When you are Forking a repository you are creating a copy of repository under the GitHub Id. Any changes made to the original original repository will be reflected back to your forked repository. However, if you make any changes to your forked repository you will have to explicitly create a pull request to the original repository. When your pull request is approved by the administrator of the original repository, then your changes will be merged with the existing original code-base. Until then, your changes will be reflected only in the copy you forked.

A Clone is where you have proper duplication, and separation between, two versions of a repository. When one repository is amended, the new content must be actively copied to the other repository using a push command. And changes in the other repository are fetched.

**Q) how you can undo the last commit ? or How to undo this commit?**

If you wish to [undo/revert the last commit](https://git-scm.com/docs/git-revert) you can do the following, using the commit hash that you get from the git log command:

**git revert <commit hash>**

This command will create a new commit with the “Revert” word in the beginning of the message. After this, if you check your repository status, you’ll notice that you have the HEAD detached at the commit you tested before.

**root@debian:/home/debian/test-project# git status**  
HEAD detached at 69d885e  
(...)

[You don’t want to see this message](https://www.git-tower.com/learn/git/faq/detached-head-when-checkout-commit), so to fix this and attach back the HEAD to your working repository, you should checkout the branch you are working on:

**git checkout <current branch>**

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

First, before we bring the big guns in, let's make sure you really need them. Because in case you just want to edit your last commit, you can simply use Git's amend feature. It allows you to correct the last commit's message as well as add more changes to it. If that's what you want to do, [read more about amend](https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/undoing-things#start).

**Undoing the Last Commit**

However, of course, there a tons of situations where you really want to undo that last commit. E.g. because you'd like to restructure it extensively - or even discard it altogether!

In these cases, the "reset" command is your best friend:

**$ git reset --soft HEAD~1**

Reset will rewind your current HEAD branch to the specified revision. In our example above, we'd like to return to the one before the current revision - effectively making our last commit undone.

Note the --soft flag: this makes sure that the changes in undone revisions are preserved. After running the command, you'll find the changes as uncommitted local modifications in your working copy.

If you don't want to keep these changes, simply use the --hard flag. Be sure to only do this when you're sure you don't need these changes anymore.

**$ git reset --hard HEAD~1**

**Q) How you can unstage a file ?**

This should put you back in the state you were before staging all your changes files. Allowing you to stage changed files individually before you commit.

# Unstage an added file in Git

If you added a file by mistake, you can unstage it (but keep local changes) by saying

git reset HEAD path/to/file

This is also what git status will tell you.

**Q) What is the propose of tagging ?**

**git tag** is used to pin down a given revision as significant, often for the purpose of officially releasing code.  Once a tag is created, it's often referred to by build and deploy scripts instead of the tag's represented commit.  Tagging revisions  and deleting tags in git is easy

Tagging a Revision

**git tag {tagname**

**git tag 0.1.0**

**Pushing Tags**

You can create a tag locally but they aren't automatically pushed to the remote repository -- you must do that manually:

**git push --tags**

**Deleting Tags**

If you want to delete or remove a tag from history, you may do so using the -d flag:

git tag -d 0.1.0

git push origin :refs/tags/0.1.0

**Q) How to list the author user-name and email ?**

The git config --list command

# Q) [How to list all commits that changed a specific file?](https://stackoverflow.com/questions/3701404/how-to-list-all-commits-that-changed-a-specific-file)

git log --follow – filename

git log -p filename

**Q) What is "cherry-pick" can you explain with use case ?**

cherry-picking simply means picking a commit from a branch and applying that commit onto another branch. It is more useful for sampling out a small subset of changes from a topic branch you've decided to delete, but you still got some useful commits on it. This will introduce a new, distinct commit.

**Use Cases**

Cherry-picking becomes very useful and essential when it comes to bug fixing. This is because bugs are fixed and tested on the development branch with their respective snapshots (committed) — which might have been on a different level from your production branch.

As with other Git operations that introduce changes via the process of applying a diff, you may need to resolve conflicts to fully apply the changes from the given commit. This could be averted by doing a git cherry-pick.

This could also be useful whenever a full branch merge is not possible due to incompatible versions in the various branches.

Also, you can use git cherry-pick to pull the changes introduced to a sub-branch, without changing the branch, by your colleague working on the same project

**Why do a GIT cherry-pick**

As stated above while defining what a git cherry-pick is, you will mostly cherry-pick a snapshot instead of merging the whole branch into the current branch.

**Q) What is origin in git ?**

"origin" is the name of the remote repository where you want to publish you commits. By convention, the default remote repository is called "origin", but you can work with several remotes (with different names) as the same time

origin is an alias on your system for a particular remote repository. It's not actually a property of that repository

origin is the alias for the URL from which you cloned the repository. Note that you can change this alias.

**Q) What is fast forwed merge ?**

A **fast-forward merge** can occur when there is a linear path between [branches](https://kolosek.com/git-branches/) that we want to merge. If a master has **not diverged**, instead of creating a new commit, it will just point master to the latest commit of the hotfix branch. All commits from hotfix branch are now available in master

**Q) How to rename local branch and central repo branch ?**

1. Start by switching to the local branch which you want to rename:

git checkout <old\_name>

1. Rename the local branch by typing:

git branch -m <new\_name>

1. If you’ve already pushed the <old\_name> branch to the remote repository delete the <old\_name> remote branch:

git push origin --delete <old\_name>

1. Finally push the <new\_name> local branch and reset the upstream branch

git push origin -u <new\_name>

That’s it. At this point you have successfully renamed your local and remote Git branch.

## [Conclusion](https://linuxize.com/post/how-to-rename-local-and-remote-git-branch/#conclusion)

Renaming a local Git Branch is just a matter of one command. However you can’t directly rename a remote branch, you need to delete it and then re-push the renamed local branch.

how you can undo the last commit if commit are already pushed to central ?

### Alternative 1: Fix and commit again

No need to explain this. Simply fix and push again in a new commit.

### Alternative 2: Revert the commit

This will achieve more or less the same as fixing and committing again, but it’s done automatically and *erases* all the changes from the bad commit. If your mistakes is only a small part of a big commit, scenario 1 should be the best bet. This is useful if the commit is let’s say 5 commits behind and you wish to *erase* those changes. I say *erase* because the history will still show all commits, but the bad ones won’t affect the code anymore.

git revert acfcaf7b

**Q) Git stages and flow ?**

Stages in GIT Life Cycle

Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

* Working directory
* Staging area
* Git directory

