**Define 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.

**Define ‘bare repository’ in Git?**

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

* A working tree, or checked out copies of your project files.
* A .git subdirectory with all the Git related revision history of your repo.

**What are the different ways you can refer to a commit?**

In Git each commit is given a unique hash. These hashes can be used to identify the corresponding commits in various scenarios (such as while trying to checkout a particular state of the code using the git checkout {hash} command).

Additionally, Git also maintains a number of aliases to certain commits, known as refs. Also, every tag that you create in the repository effectively becomes a ref (and that is exactly why you can use tags instead of commit hashes in various git commands). Git also maintains a number of special aliases that change based on the state of the repository, such as HEAD, FETCH\_HEAD, MERGE\_HEAD, etc.

Git also allows commits to be referred as relative to one another. For example, HEAD~1 refers to the commit parent to HEAD, HEAD~2 refers to the grandparent of HEAD, and so on. In case of merge commits, where the commit has two parents, ^ can be used to select one of the two parents, e.g. HEAD^2 can be used to follow the second parent.

And finally, refspecs. These are used to map local and remote branches together. However, these can be used to refer to commits that reside on remote branches allowing one to control and manipulate them from a local Git environment.

**What is a conflict in git and how can it be resolved?**

A conflict arises when more than one commit that has to be merged has some change in the same place or same line of code. Git will not be able to predict which change should take precedence. This is a git conflict.

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 merge, so it sets the parents of the commit correctly.

**What does commit object contains?**

Commit object contains the following components, you should mention all the 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.

**What is SubGit?**

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.

**What is the HEAD in GIT ?**

AHEAD is a reference to the present looked at conferring.

It is a representative reference to the branch that we have looked at.

At any given time, one head is chosen as the ‘present head’ this head is otherwise called HEAD (dependably in capitalized).

**What are Git Design objectives?**

Distributed workflow (decentralised)

Easy merging (merge deemed more frequent than commit)

Integrity (protection against accidental/malicious corruptions)

Speed & scalability

**What is Version Control with Git?**

Version control is better than mailing files back and forth because:

* It’s is not impossible to coincidentally overwrite or overlook someone’s changes: whenever there’s a conflict between one person’s work and another’s, the version control system automatically notifies users.
* If people are having some questions to ask, they will maintain the records what changes they have made.
* Nothing that is committed to version control is ever lost. it’s always possible to go back in time to know exactly who wrote what on a particular day, or what version of a program was used to generate a particular set of results. This means it can be used like the undo feature in an editor, and since all previous versions of files are saved.

Git is one of many version control systems. It is more complex than some alternatives, but it is widely used, both because it’s easy to set up and because of a hosting site called GitHub, which we will get to later.

**What are the main benefits of GIT ?**

Distributed System: GIT is a Distributed Version Control System (DVCS). So you can keep your private work in adaptation control yet totally escaped others. You can work disconnected too.

⦁ Flexible Workflow:GIT enables you to make your own work process. You can utilize the procedure that is appropriate for your venture. You can go for brought together or ace slave or some other work process.

⦁ Fast: GIT is quick when contrasted with other form control frameworks.

⦁ Data Integrity: Since GIT utilizes SHA1, information isn’t less demanding to degenerate.

⦁ Free: It is free for individual utilize. Such huge numbers of beginners utilize it for their underlying activities. It likewise works exceptionally well with substantial size task.

⦁ Collaboration: GIT is anything but difficult to use for ventures in which joint effort is required. Numerous prevalent open source programming over the globe utilize GIT

11 Painful Git Interview Questions You Will Cry On

[Alex Ershov](https://dev.to/aershov24)Jul 30 *Updated on Sep 05, 2018*

[**#git**](https://dev.to/t/git) [**#interview**](https://dev.to/t/interview) [**#questions**](https://dev.to/t/questions) [**#fullstack**](https://dev.to/t/fullstack)

[](https://res.cloudinary.com/practicaldev/image/fetch/s--ZdnrNCoZ--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/images.pexels.com/photos/929382/pexels-photo-929382.jpeg%3Fauto%3Dcompress%26cs%3Dtinysrgb%26h%3D350)  
According to the latest Stack Overflow developer survey, more than 70 percent of developers use Git, making it the most-used VCS in the world. Git is commonly used for both open source and commercial software development, with significant benefits for individuals, teams and businesses.

Q1: What is Git fork? What is difference between fork, branch and clone?

Topic: **Git**  
Difficulty: ⭐⭐

* A **fork** is a remote, server-side copy of a repository, distinct from the original. A fork isn't a Git concept really, it's more a political/social idea.
* A **clone** is not a fork; a clone is a local copy of some remote repository. When you clone, you are actually copying the entire source repository, including all the history and branches.
* A **branch** is a mechanism to handle the changes within a single repository in order to eventually merge them with the rest of code. A branch is something that is within a repository. Conceptually, it represents a thread of development.

🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/3329943/git-branch-fork-fetch-merge-rebase-and-clone-what-are-the-differences/)

Q2: What's the difference between a "pull request" and a "branch"?

Topic: **Git**  
Difficulty: ⭐⭐

* A **branch** is just a separate version of the code.
* A **pull request** is when someone take the repository, makes their own branch, does some changes, then tries to merge that branch in (put their changes in the other person's code repository).

🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/19059838/whats-the-difference-between-a-pull-request-and-a-branch)

Q3: What is the difference between "git pull" and "git fetch"?

Topic: **Git**  
Difficulty: ⭐⭐

In the simplest terms, git pull does a git fetch followed by a git merge.

* When you use pull, Git tries to automatically do your work for you. **It is context sensitive**, so Git will merge any pulled commits into the branch you are currently working in. pull **automatically merges the commits without letting you review them first**. If you don’t closely manage your branches, you may run into frequent conflicts.
* When you fetch, Git gathers any commits from the target branch that do not exist in your current branch and **stores them in your local repository**. However, **it does not merge them with your current branch**. This is particularly useful if you need to keep your repository up to date, but are working on something that might break if you update your files. To integrate the commits into your master branch, you use merge.

🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/292357/what-is-the-difference-between-git-pull-and-git-fetch)

Q4: How to revert previous commit in git?

Topic: **Git**  
Difficulty: ⭐⭐⭐

Say you have this, where C is your HEAD and (F) is the state of your files.

(F)

A-B-C

↑

master

* To nuke changes in the commit:

git reset --hard HEAD~1

Now B is the HEAD. Because you used --hard, your files are reset to their state at commit B.

* To undo the commit but keep your changes:

git reset HEAD~1

Now we tell Git to move the HEAD pointer back one commit (B) and leave the files as they are and git status shows the changes you had checked into C.

* To undo your commit but leave your files and your index

git reset --soft HEAD~1

When you do git status, you'll see that the same files are in the index as before.

🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/927358/how-to-undo-the-most-recent-commits-in-git)

Q5: What is "git cherry-pick"?

Topic: **Git**  
Difficulty: ⭐⭐⭐

The command git *cherry-pick* is typically used to introduce particular commits from one branch within a repository onto a different branch. A common use is to forward- or back-port commits from a maintenance branch to a development branch.

This is in contrast with other ways such as merge and rebase which normally apply many commits onto another branch.

Consider:

git cherry-pick <commit-hash>

🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/9339429/what-does-cherry-picking-a-commit-with-git-mean)

Q6: Explain the advantages of Forking Workflow

Topic: **Git**  
Difficulty: ⭐⭐⭐

The **Forking Workflow** is fundamentally different than other popular Git workflows. Instead of using a single server-side repository to act as the “central” codebase, it gives every developer their own server-side repository. The Forking Workflow is most often seen in public open source projects.

The *main advantage* of the Forking Workflow is that contributions can be integrated without the need for everybody to push to a single central repository that leads to a clean project history. Developers push to their own server-side repositories, and only the project maintainer can push to the official repository.

When developers are ready to publish a local commit, they push the commit to their own public repository—not the official one. Then, they file a pull request with the main repository, which lets the project maintainer know that an update is ready to be integrated.

🔗**Source:** [atlassian.com](https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow)

Q7: Tell me the difference between HEAD, working tree and index, in Git?

Topic: **Git**  
Difficulty: ⭐⭐⭐

* The **working tree/working directory/workspace** is the directory tree of (source) files that you see and edit.
* The **index/staging area** is a single, large, binary file in /.git/index, which lists all files in the current branch, their sha1 checksums, time stamps and the file name - it is not another directory with a copy of files in it.
* **HEAD** is a reference to the last commit in the currently checked-out branch.

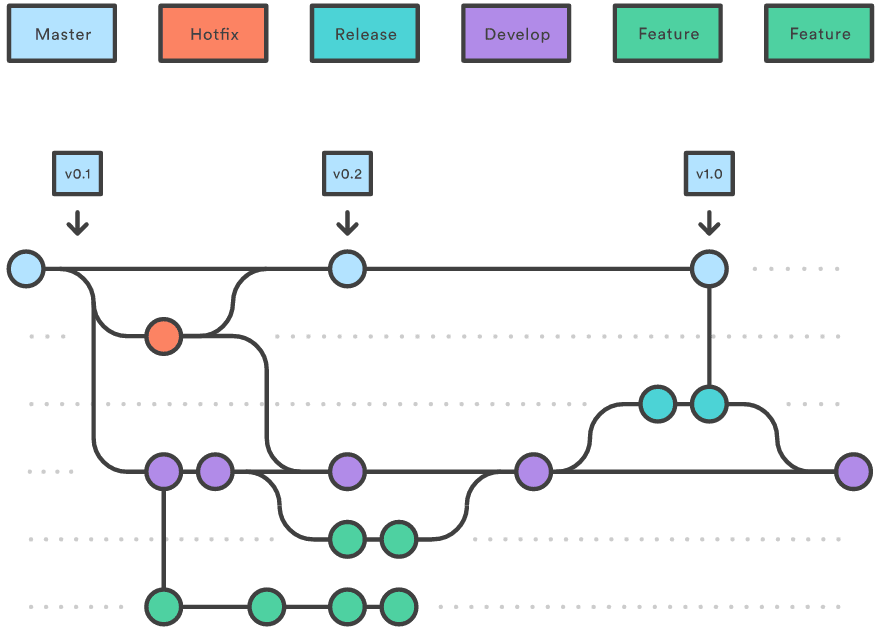
🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/3689838/whats-the-difference-between-head-working-tree-and-index-in-git)

Q8: Could you explain the Gitflow workflow?

Topic: **Git**  
Difficulty: ⭐⭐⭐

Gitflow workflow employs two parallel *long-running* branches to record the history of the project, master and develop:

* **Master** - is always ready to be released on LIVE, with everything fully tested and approved (production-ready).
  + **Hotfix** - Maintenance or “hotfix” branches are used to quickly patch production releases. Hotfix branches are a lot like release branches and feature branches except they're based on masterinstead of develop.
* **Develop** - is the branch to which all feature branches are merged and where all tests are performed. Only when everything’s been thoroughly checked and fixed it can be merged to the master.
  + **Feature** - Each new feature should reside in its own branch, which can be pushed to the develop branch as their parent one.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--pLQxGakq--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/wac-cdn.atlassian.com/dam/jcr:61ccc620-5249-4338-be66-94d563f2843c/05%2520%282%29.svg%3FcdnVersion%3Dji)

🔗**Source:** [atlassian.com](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)

Q9: When should I use "git stash"?

Topic: **Git**  
Difficulty: ⭐⭐⭐

The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy.

Consider:

$ git status

On branch master

Changes to be committed:

new file: style.css

Changes not staged for commit:

modified: index.html

$ git stash

Saved working directory and index state WIP on master: 5002d47 our new homepage

HEAD is now at 5002d47 our new homepage

$ git status

On branch master

nothing to commit, working tree clean

The one place we could use stashing is if we discover we forgot something in our last commit and have already started working on the next one in the same branch:

# Assume the latest commit was already done

# start working on the next patch, and discovered I was missing something

# stash away the current mess I made

$ git stash save

# some changes in the working dir

# and now add them to the last commit:

$ git add -u

$ git commit --ammend

# back to work!

$ git stash pop

🔗**Source:** [atlassian.com](https://www.atlassian.com/git/tutorials/saving-changes/git-stash)

Q10: How to remove a file from git without removing it from your file system?

Topic: **Git**  
Difficulty: ⭐⭐⭐⭐

If you are not careful during a git add, you may end up adding files that you didn’t want to commit. However, git rm will remove it from both your staging area (index), as well as your file system (working tree), which may not be what you want.

Instead use git reset:

git reset filename # or

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

This means that git reset <paths> is the opposite of git add <paths>.

🔗**Source:** [codementor.io](https://www.codementor.io/citizen428/git-tutorial-10-common-git-problems-and-how-to-fix-them-aajv0katd)

Q11: When do you use "git rebase" instead of "git merge"?

Topic: **Git**  
Difficulty: ⭐⭐⭐⭐⭐

Both of these commands are designed to integrate changes from one branch into another branch - they just do it in very different ways.

Consider before merge/rebase:

A <- B <- C [master]

^

\

D <- E [branch]

after git merge master:

A <- B <- C

^ ^

\ \

D <- E <- F

after git rebase master:

A <- B <- C <- D <- E

With rebase you say to use another branch as the new base for your work.

**When to use:**

1. If you have any doubt, use merge.
2. The choice for rebase or merge based on what you want your history to look like.

**More factors to consider:**

1. **Is the branch you are getting changes from shared with other developers outside your team (e.g. open source, public)?** If so, don't rebase. Rebase destroys the branch and those developers will have broken/inconsistent repositories unless they use git pull --rebase.
2. **How skilled is your development team?** Rebase is a destructive operation. That means, if you do not apply it correctly, you could lose committed work and/or break the consistency of other developer's repositories.
3. **Does the branch itself represent useful information?** Some teams use the *branch-per-feature* model where each branch represents a feature (or bugfix, or sub-feature, etc.) In this model the branch helps identify sets of related commits. In case of *branch-per-developer* model the branch itself doesn't convey any additional information (the commit already has the author). There would be no harm in rebasing.
4. **Might you want to revert the merge for any reason?** Reverting (as in undoing) a rebase is considerably difficult and/or impossible (if the rebase had conflicts) compared to reverting a merge. If you think there is a chance you will want to revert then use merge.

🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/804115/when-do-you-use-git-rebase-instead-of-git-merge)

*Thanks 🙌 for reading and good luck on your interview!*   
*Check more FullStack Interview Questions & Answers on 👉*[*www.fullstack.cafe*](https://www.fullstack.cafe/)