# Interview Questions and Answers You Will Cry On

Here are some tough, sample DevOps interview questions and answers for experienced software professionals hoping to get an advantage over fellow job applicants and candidates. Looking to get that DevOps job and you need some help to prep? Or are you a DevOps manager interviewing potential applicants? These popular DevOps interview questions and answers for experienced software pros will test the mettle of even the most advanced CI/CD experts.

<https://www.theserverside.com/video/10-tough-Jenkins-interview-questions-for-DevOps-engineers>

<https://www.mytectra.com/interview-question/?s=Puppet>

<https://www.mytectra.com/interview-question/aws-interview-questions/>

<https://www.zeolearn.com/interview-questions/ansible>

<https://www.mytectra.com/interview-question/tag/automation-testing-with-python/>

<https://www.whizlabs.com/blog/ansible-interview-questions/>

<https://devinpractice.com/2018/07/22/ansible-20-interview-questions/>

<https://www.gangboard.com/blog/ansible-interview-questions-and-answers/>

<http://aptronnoida.in/iqa/best-ansible-interview-questions-answers/>

<https://www.stechies.com/ansible-interview-questions/>

<https://www.knowledgehut.com/interview-questions/ansible>

<https://www.algrim.co/142-ansible-interview-questions>

<https://www.onlineinterviewquestions.com/ansible-interview-questions/>

<https://codingcompiler.com/ansible-interview-questions/>

<https://codingcompiler.com/linux-administrator-interview-questions-answers/>

<https://www.codecompiled.com/python-tutorial-for-beginners/python-interview-questions-and-answers/>

<https://mindmajix.com/github-interview-questions>

<https://intellipaat.com/blog/interview-questions/>

<https://intellipaat.com/blog/interview-question/git-interview-questions-answers/>

<https://career.guru99.com/top-40-interview-questions-on-git/>

<https://www.janbasktraining.com/blog/chef-interview-questions/>

<https://www.unixmen.com/install-oracle-java-jdk-8-centos-76-56-4/>

<https://www.knowledgehut.com/interview-questions>

### DevOps interview questions and answers

More and more organizations expect to see a reference to DevOps on the resumes of applicants applying for technical positions. These sample DevOps interview questions are the ones candidates get asked to test their mettle. And if you really want to impress on that DevOps engineering interview, tackle these 10 tough [Jenkins interview questions and answers](https://www.theserverside.com/video/10-tough-Jenkins-interview-questions-for-DevOps-engineers) when you've mastered all of the questions here.

### Sample DevOps interview questions

The following is the unanswered list of the most commonly asked DevOps interview questions:

1. **Name five important DevOps tools that organizations should consider adopting when undergoing a DevOps transition.**
2. **What is continuous integration (CI), and what is its purpose?**
3. **What does it mean to**[***shift left in DevOps***](https://devopsagenda.techtarget.com/opinion/Experts-Shift-left-toward-DevOps-automated-testing)**?**
4. **The acronym CAMS is often used to describe the core tenets of DevOps. What does it stand for?**
5. **Is a culture change a requirement or a result of doing DevOps?**
6. **Name three important DevOps key performance indicators (KPIs).**
7. **Describe three DevOps anti-patterns.**
8. **What are benefits of implementing DevOps automation?**
9. **Name two popular Java development frameworks for creating microservices.**
10. **What is continuous delivery (CD)?**

### Basic DevOps interview questions answered

Regardless of whether you agree or disagree with my answers to these sample DevOps interview questions, be prepared to answer them in your own way and in your own voice.

**1. Name five important DevOps tools that organizations should consider adopting when undergoing a DevOps transition.**

I would answer the first of these sample DevOps interview questions with an assertion that there are five key arenas in which tools can assist in a DevOps transition:

* configuration management
* source code management
* CI
* containerization
* collaboration

Chef, Puppet and Ansible are all highly capable configuration management tools.

### Jenkins and GitHub  DevOps interview questions

[Git](https://www.theserverside.com/tutorial/Five-basic-Git-commands-every-beginner-needs-to-know) and a cloud-based platform, such as [GitHub or GitLab](https://www.theserverside.com/tutorial/Attain-Jenkins-Git-integration-with-a-GitHub-pull-request), make the management of distributed source code easier.

Jenkins is perhaps the most popular [CI server](https://www.theserverside.com/tutorial/Create-a-Jenkins-build-job-A-CI-tutorial-for-beginners) on the market, although there are some highly competent competitors, such as Concourse CI and Atlassian's Bamboo.

For containerization, Docker tends to get the most attention, but Rkt and LXD have a significant following as well.

In terms of collaboration, Jira is an excellent tool for Agile team collaboration.

**2. What is CI, and what is its purpose?**

The key to answering the second of these sample DevOps interview questions is concentrating on CI and not falling into a description of CD or continuous deployment.

CI is the process of compiling the entire code base every time a member of the software development team checks code into the shared source code repository.

If a team member checks in code that causes compilation to fail, then the developer has broken the build. If this is the case, other developers won't be able to synchronize with the shared source code repository without introducing compilation errors into their own local workspace. This means collaborative and shared software development cannot go forward. As such, when a [CI build](https://www.theserverside.com/video/Modify-CI-jobs-with-this-Jenkins-parameterized-build-example) breaks, it is imperative that the problem is corrected immediately.

A CI process often includes a suite of unit, integration and regression tests that run every time compilation succeeds. If any of these tests fail, the build is considered unstable but not broken. An unstable build isn't uncommon, especially during an Agile sprint, where development is ongoing.

**3. What does it mean to shift left in DevOps?**

If you were to graph the traditional software development lifecycle on a piece of paper, the left side of the graph would likely include tasks such as design and development, while the right side would likely include user acceptance, stress testing and production staging. To shift left in DevOps implies a desire to take many of those tasks that often happen toward the end of the application development process and move them into earlier stages.

In some cases, this might mean to incorporate [static code analysis](https://www.theserverside.com/video/Use-Maven-Checkstyle-Plugin-to-enforce-Java-quality-rules) routines in every build. Another way to perform a DevOps shift left is to create production-ready artifacts at the end of every Agile sprint so that users and stakeholders can get incremental updates on how development is progressing. Proper DevOps means shifting left as much as possible.

**4. The acronym CAMS is often used to describe the core tenets of DevOps. What does it stand for?**

The CAMS acronym stands for culture, automation, measurement and sharing.

**5. Is a culture change a requirement or a result of doing DevOps?**

The term culture describes the processes and practices people use. And culture can only change when people change the processes and practices they routinely employ. The introduction of [distributed version control system tools](https://www.theserverside.com/video/How-to-use-the-git-reset-hard-command-to-change-a-commit-history), such as Git, deployments with Docker instead of the use of monolithic application servers, and the use of CI tools, such as [Jenkins or Maven](https://www.theserverside.com/opinion/Jenkins-vs-Maven-Compare-these-build-and-integration-tools), in the software development lifecycle will result in a culture change. But these changes are the result of the introduction of new DevOps tools, not a prerequisite.

Any [cultural change](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/The-DevOps-encompasses-culture-and-collaboration-myth-destroyed) that DevOps precipitates is an output of using new tools and adopting new processes. It is not an input.

It should be noted that many [DevOps evangelists](https://devopsagenda.techtarget.com/opinion/Be-wary-of-DevOps-evangelists-preaching-a-culture-change) would take serious umbrage with this sample DevOps interview question and answer. Be prepared to argue and defend both sides of the culture argument.

**6. Name three important DevOps KPIs.**

There are many KPIs that can be used to measure DevOps success. Three of the most common DevOps KPIs are:

* mean time to failure recovery
* deployment frequency
* percentage of failed deployments

**7. Describe three DevOps anti-patterns**

One DevOps anti-pattern is to reduce the silo-based isolation of development and operations with a new DevOps team that silos itself from other parts of the organization.

The first DevOps anti-pattern mentioned leads to the second one, which is failing to include all aspects of the organization in an ongoing DevOps transition. Organizations embarking upon a DevOps transition should have some type of DevOps center of excellence that makes sure all tiers of the organization -- from IT executives to project managers to members of the security and hardware provisioning teams -- embrace and employ DevOps practices.

Another common anti-pattern is not defining KPIs at the start of a DevOps transition. Without identifying important DevOps KPIs and creating a baseline at the start of the journey, it is impossible to evaluate success and identify failures.

**8. What are benefits of implementing DevOps automation?**

The core benefit to DevOps automation is that it takes the possibility for human error out of the CD equation.

Furthermore, when automation replaces manual tasks, those tasks become repeatable and predictable. As such, when something goes wrong in an automated process, it is easy to identify and correct the problem. These types of refinements are said to harden the DevOps process, leading to more reliable and robust systems.

Another benefit to DevOps automation is that it can remove bottlenecks from the CI pipeline. This increases deployment frequency and reduces the number of failed deployments, both of which are important DevOps KPIs.

**9. Name two popular Java development frameworks for creating microservices.**

There are a number of Java frameworks for microservices development on the market today; two of the most popular are [Spring Boot](https://www.theserverside.com/tutorial/Spring-MVC-tutorial-How-Spring-Boot-aids-Java-web-development) and Eclipse MicroProfile. Don't fall for the [Java microservice myth](https://www.theserverside.com/video/Lets-dispel-three-common-Java-microservices-myths) that Spring Boot and the Eclipse MicroProfile are the only players in the game though. There are many competitors.

**10. What is CD?**

CD expands upon the ideas of CI. With CI, if you can compile a build, it's in its optimum state. In contrast to CI, CD asserts that there should be a main branch of ongoing source code integration that you can package, deploy and release into production at any given point.

Author Martin Fowler [contends that](https://martinfowler.com/bliki/ContinuousDelivery.html) you are doing CD if:

* Your software is deployable throughout its lifecycle.
* Your team prioritizes keeping the software deployable over working on new features.
* Anyone can get fast, automated feedback on the production readiness of their systems whenever somebody makes a change to them.
* You can perform push-button deployments of any version of the software to any environment on demand.

CI is a part of CD, but so is the creation of deployable [container-packaged executables](https://www.theserverside.com/tutorial/How-to-embed-Tomcat-and-Java-web-apps-in-an-executable-JAR). So, too, is the use of a comprehensive suite of tests that evaluates software quality, performance under load and user acceptance.

### Sample DevOps interview questions and answers summary

This certainly isn't an exhaustive list of every question you might encounter, but many of these sample DevOps interview questions and answers are likely to be encountered in a technical interview -- especially if DevOps is part of the job requirement. If you understand the answers provided here and you can respond in your own words, you are well on your way to landing that new job as a DevOps engineer.

**GIT**

1. **Difference between Git vs SVN**

The main point in Git vs SVN debate boils down to this: Git is a distributed version control system (DVCS), whereas SVN is a centralized version control system.

2. **What is Git fork? What is difference between fork and branch? How to create tag?**

Create the target repository in your personal space.

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

A fork is really a Github (not Git) construct to store a clone of the repo in your user account. As a clone, it will contain all the branches in the main repo at the time you made the fork.

*Create Tag:*

* Click the releases link on our repository page.
* Click on Create a new release or Draft a new release.
* Fill out the form fields, then click Publish release at the bottom.
* After you create your tag on GitHub, you might want to fetch it into your local repository too: git fetch.

**3. What is git cherry-pick?**

Cherry picking in git means to choose a commit from one branch and apply it onto another.

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

Make sure you are on the branch you want apply the commit to. git checkout master Execute the following:

*git cherry-pick <commit-hash>*

**4. How to revert previous commit in git?**

To revert to a previous commit, ignoring any changes:

*git reset — hard HEAD*

where HEAD is the last commit in your current branch

**5. How to rebase master in git? Difference between rebase and merge. How to squash or fixup commits?**

Rebasing is the process of moving a branch to a new base commit.

The golden rule of git rebase is to never use it on public branches. … The only way to synchronize the two master branches is to merge them back together, resulting in an extra merge commit and two sets of commits that contain the same changes.

**6. Explain git stash, pop**

Use git stash when you want to record the current state of the working directory and the index, but want to go back to a clean working directory.

git stash pop applies the top stashed element and removes it from the stack. git stash apply does the same, but leaves it in the stash stack.

**7. Branching strategies Pros and Cons — Feature, Release branching, Git flow, Lean Git flow**

Pro: By keeping latest deployed version in trunk, small fixes can be rolled out quickly without extensive testing of the latest development version.

Pro: Developers can work more freely in tighter iterations without stepping on eachother’s feet.

Pro: if you have many branches you’ll be pushed to adopt a modern DVCS (my experience is with Mercurial but I hear git or Bazaar are also good) rather than stay with a traditional centralized system (like, say, svn).

Pro: Branches can be used to facilitate ‘what-if’ scenario’s in trying out new code. At the end a decision can be made to merge the new feature or to abandon it.

Con: Having too many branches in the air at the same time and you start forgetting where things where commited, where changes have been made etc.

Con: someone has to manage the branch(es) and keep on top of things. In most teams this falls by the way-side.

**8. What is GIT?**

GIT is a distributed version control system and source code management (SCM) system.

**9. 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.

**10. What are Git repository hosting services you used?**

Github, Bitbucket, Gitlab or you can specify if you have used something else here.

**11. What is the function of ‘git reset’?**

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

**12. What is the syntax for “Rebasing” in Git?**

Syntax: “git rebase [new-commit] “

**13. 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.

**14. What language is used in Git?**

Git used “C programing language”.

**15. 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

Hope this post will help you to crack your interview. As always credit for this post goes to multiple blogger and QnA websites.

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 <baseOfRepo>/.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.

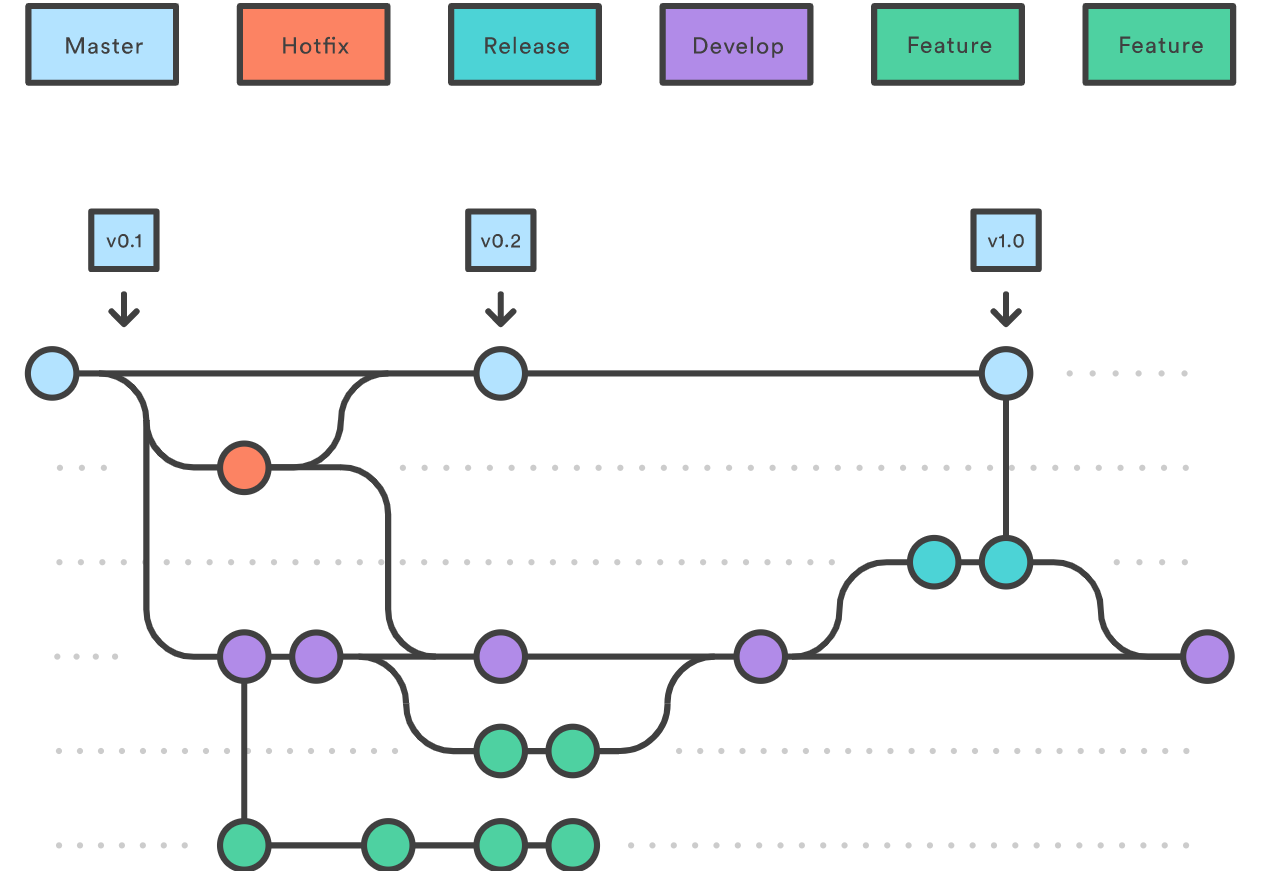
🔗**Source:** [stackoverflow.com](https://stackoverflow.com/questions/3689838/whats-the-difference-between-head-working-tree-and-index-in-git)

### 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 master instead 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.



🔗**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****<br/>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.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

new to learn git cherry-pick command  
git pull = git fetch + merge

git commit --ammend  
This should be --amend

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

here .gitingore needs a relook

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

As organizations successfully implement their DevOps roadmaps, distributed version control systems, such as Git and GitHub, play a bigger role in everyday operations. If you are interested in landing a DevOps job, you'll want to be ready for the tough GitHub interview questions and answers that gauge a candidate's version control competency.

### Tough GitHub interview questions

Before we start, note that these GitHub interview questions and answers focus solely on the features that the cloud-based GitHub service provides and not the Git tool independently or Git-and-GitHub integration.

Before you jump into the answers, read the individual questions, and see how well you can formulate your own answers.

1. What is the difference between Git and GitHub?
2. Name two Git commands that are not supported by GitHub Desktop.
3. Which four states can be applied to a commit using the GitHub status API?
4. Name three popular DevOps tools with which GitHub commonly integrates.
5. What does the GitHub unicorn indicate?
6. What is the key motivation for a paid GitHub subscription?
7. Name two insights GitHub provides about repositories.
8. How can you stop an addled developer from accidentally pushing a bad commit to the master branch?
9. What is a GitHub interaction limit?
10. Where in a Git project can you use a GitHub emoji? Provide an example of three GitHub emoji.

### GitHub interview questions and answers

As always, it's not good enough to simply memorize the answers and regurgitate what is written here. If you want to land that [DevOps job](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/The-DevOps-encompasses-culture-and-collaboration-myth-destroyed), make sure you can articulate the answers to these interview questions on GitHub in your own words. If you can do that, you'll be one step closer to that new job.

**1. What is the difference between Git and GitHub?**

Git is a distributed version control tool that developers install on local machines and use to track commit histories, merge files and share code with fellow developers. In comparison, GitHub is a cloud-based source code repository built around the Git tool. Along with providing a central location from which Git users can push and pull code, GitHub also adds a variety of services and features that are not native to Git, such as forking, user management, online editing and branch protection.

Git is maintained by The Linux Foundation, while GitHub was acquired by Microsoft. Git competes with other version control tools, such as Subversion and Mercurial. GitHub competes with GitLab, Atlassian Bitbucket and other SaaS-based services.

Candidates being asked GitHub interview questions will often be told to differentiate between the online service and the Git tool itself, so be prepared when the opportunity arises.

**2. Name two Git commands that are not supported by GitHub Desktop.**

The GitHub Desktop app is designed primarily to simplify the most [commonly used functions](https://www.theserverside.com/video/Test-Git-command-basics-on-the-GitHub-Desktop-app). It [supports basic Git commands](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Step-by-step-guide-to-install-Git-on-Windows-desktop-computers), such as init, add, commit, push, pull and fetch. It does not currently have support for more advanced Git capabilities, including the revert and [cherry-pick](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Need-to-git-cherry-pick-a-commit-Heres-an-example-how) commands.

It should be noted that the GitHub Desktop app enables a developer to open a terminal window and lets the developer manually issue Git commands, such as reset and cherry-pick.

**3. What four states can be applied to a commit using the GitHub status API?**

Just because a Git commit succeeds without conflict doesn't mean the code in question compiles, nor does it guarantee that any unit or integration tests pass. As such, the GitHub status API enables you to mark commits with one of four statuses: error, failure, pending or success.

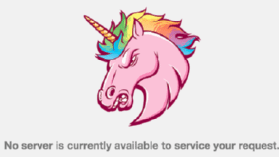
The GitHub status API is often used by [CI tools](https://www.theserverside.com/video/10-tough-Jenkins-interview-questions-for-DevOps-engineers) that merge topic branches into feature branches and then push the newly generated commit on the feature branch back to GitHub. It is also possible to configure branch protection rules to inspect the incoming commit status and reject the push if it is flagged as anything but a success.

**4. Name three popular DevOps tools with which GitHub commonly integrates.**

As a source code repository, GitHub tends to be the foundation for any organization that builds a DevOps-based infrastructure. As such, GitHub commonly integrates with CD software, such as Jenkins; [Maven repositories](https://www.theserverside.com/news/1364121/Setting-Up-a-Maven-Repository), such as Artifactory; bug tracking software, such as Manuscript; and Agile workflow tools, such as Jira.

If you answer a GitHub interview question about peripheral tools, it's a good idea to mention any tools you have experience with and have integrated.

**5. What does the GitHub unicorn indicate?**

The angry pink unicorn with a rainbow mane indicates site maintenance.

Nobody will ever believe you if you say you saw a unicorn. You will, however, see the GitHub unicorn if there's site maintenance or a temporary outage. Don't expect it to happen often, though.

**6. What is the key motivation for a paid subscription?**

Prior to a [change in the subscription model](https://www.theserverside.com/feature/Want-a-private-GitHub-repository-It-comes-with-a-catch) that occurred in January 2019, the primary motivation for a paid subscription was the ability to make repositories private. When on the free tier, all repositories were publicly viewable. However, to bring themselves in line with competitors like Atlassian Bitbucket and GitLab, the basic account is allowed to mark a repository as private. However, there are limitations. Private GitHub repositories created with a basic account:

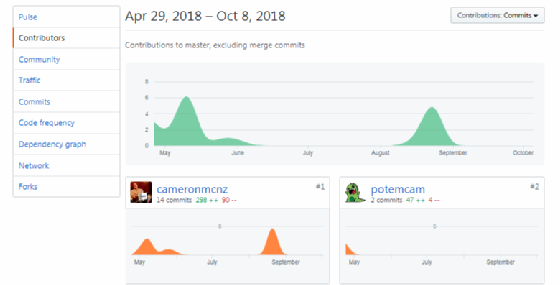
* cannot implement branch protection rules;
* are not given access to a project wiki;
* cannot take advantage of the GitHub Pages feature; and
* are limited to three contributors.

Developers interested in these features within the scope of a private GitHub repository will need to upgrade to a GitHub Pro paid subscription.

**7. Name two insights GitHub provides about repositories.**

Two interesting DevOps metrics GitHub provides through its insights tab are the repository's pulse and the project's traffic, although you should be ready to mention more than just two when being quizzed with GitHub interview questions.

The pulse provides interesting information, such as the number of merged pull requests, proposed pull requests, closed issues and new issues.

Expect a GitHub interview question about project contributors and insights.

The GitHub traffic insight tool will graph repository views over time and also indicate the number of times the repository has been cloned, how many Git users have cloned the repository and which files are most popular.

Other GitHub insight pages include:

* contributors
* community
* commits
* code frequency
* dependency graph
* network
* forks

**8. How can you stop an addle-minded developer from accidentally pushing a bad commit to the master branch?**

A treasured GitHub feature is the application of branch protection rules. These rules enable the GitHub administrator to specify [important branch](https://www.theserverside.com/tutorial/How-to-git-revert-a-commit-A-simple-undo-changes-example) names that can neither be deleted nor be the subject of a forced push without a user having administrative privileges.

To apply even more granular GitHub branch protections:

* require pull request reviews before merging;
* require a GitHub API status check to pass before merging;
* require signed commits; and
* include administrators in the branch protection rules.

Because Git branch protection is not native to the Git tool, it is a common interview question.

**9. What is a GitHub interaction limit?**

Toward the end of a development sprint, with time running out and developers anxious about getting Agile points on the Scrum board, temperatures in the office can rise. But if developers need to cool off for a while, organizations can place interaction limits on a given repository. This will restrict who can comment, create open issues and create pull requests for a 24-hour period or until the GitHub interaction limits are rescinded, whichever comes first.

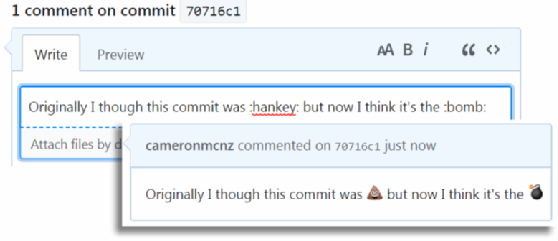
The interaction limits can be configured in three different ways:

* place GitHub interaction limits only on new users;
* place a GitHub interaction limit on people who have never made a commit; or
* place a GitHub interaction limit on all users who do not have push access to the repository.

**10. Where in a Git project can you use a GitHub emoji? Provide an example of three GitHub emoji.**

GitHub emoji were [introduced in February 2018](https://blog.github.com/2018-02-22-label-improvements-emoji-descriptions-and-more/) and can be applied to labels to describe issues and pull requests.

Three popular GitHub emoji include :bug:, :hankey: and :boom:.

Don't be surprised by a GitHub interview question about emoji.

### Interview questions on GitHub

These sample GitHub interview questions and answers are by no means exhaustive, but they should give you a good idea of what types of DVCS topics you need to be ready for when you apply for a DevOps job.

GitHub is popular because it provides a wide array of services and features around the singularly focused Git tool. But if you're familiar with some of the most important GitHub features, such as branch protection, repository insights and even GitHub Desktop, you should have no problem responding to interview questions on GitHub.

#### DevOps job preparation

Prepare yourself for that next DevOps job with the following key resources:

* Learn the [five basic Git commands](https://www.theserverside.com/tutorial/Five-basic-Git-commands-every-beginner-needs-to-know) every DevOps pro needs to master.
* Understand the build lifecycle of [Apache Maven](https://www.theserverside.com/tutorial/How-to-install-Maven-and-build-apps-with-the-mvn-command-line).
* Learn to [automate build jobs](https://www.theserverside.com/tutorial/Create-a-Jenkins-build-job-A-CI-tutorial-for-beginners) with [Jenkins CI.](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Continuous-integration-benefits-Why-adopting-a-CI-CD-tool-like-Jenkins-makes-sense)
* Test yourself with this [Git quiz](https://www.theserverside.com/quiz/Prove-your-DVCS-IQ-with-our-Git-quiz-for-beginners).
* Learn how to i[ntegrate Jenkins and Artifactory](https://www.theserverside.com/video/Install-the-Jenkins-Artifactory-plugin-in-5-easy-steps)
* Prepare with these sample [DevOps interview questions](https://www.theserverside.com/video/Tackle-these-10-sample-DevOps-interview-questions-and-answers).

With a strong knowledge of Git, DevOps, Jenkins and Maven, you will be able to walk into your GitHub interview with confidence and, we hope, walk out with a job.

### Jenkins interview questions strategies

A good strategy to use to apply to this set of tough Jenkins interview questions and answers for DevOps professionals is to first read through each question and formulate your own response. Then, read our answers. If you disagree, that is OK; just make sure you are ready to articulate your own ideas on the topic.

### Jenkins interview questions for DevOps

Here are the 10 sample Jenkins interview questions for DevOps engineers. Before delving into the set of answers, read through this list, and see how comfortable you are coming up with your own responses to these tough Jenkins interview questions.

1. What are the software prerequisites that must be met before Jenkins is installed?
2. What is the syntax Jenkins uses to schedule items such as build jobs and SVN polling?
3. Name a Jenkins environment variable you have used in a shell script or batch file.
4. Name three security mechanisms Jenkins uses to authenticate users.
5. Describe the standard process to configure and use third-party tools within Jenkins.
6. Name two ways a Jenkins node agent can be configured to communicate back with the Jenkins master.
7. How do you take a backup of your Jenkins build jobs in order to prepare for disaster recovery?
8. Name three steps or stages a typical Jenkins pipeline might include.
9. How can you temporarily turn off Jenkins security if the administrative users have locked themselves out of the admin console?
10. Polling a [Git repository](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-git-cherry-pick-from-another-branch-to-your-own) for new commits is considered a Jenkins anti-pattern. What is a sound alternative to SVN polling?

# How to 'git cherry-pick' from another branch to your own

In a previous tutorial, we took a look at how to [cherry-pick a commit](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Need-to-git-cherry-pick-a-commit-Heres-an-example-how) on the current branch, but one of the ancillary questions that commonly arises is how to perform a git cherry-pick from another branch. The topic of how to git cherry-pick from another branch, along with what the result of such an operation would be, is the focus of this tutorial.

As with all git tutorials, this one will start off with a clean repository and an empty working directory, which means the first step is to create a new folder, which I will name git cherry-pic example. The next step is to issue a git init call from within that folder.

/c/ git cherry-pick example (master)

$ git init

Initialized empty Git repository in C:/\_git-cherry-pick-example/.git/

### Preparing a branch for a git cherry-pick

With the repository initialized, the next step is to create three new files, adding a commit after each individual file is created. Since the repo was just initialized, all of this will occur on the master branch.

/c/ git cherry-pick example **(master)**

$ echo 'abba' > abba.html

$ git add . | git commit -m '1st commit: 1 file'

$ echo 'bowie' > bowie.html

$ git add . | git commit -m '2nd commit: 2 files'

$ echo 'chilliwack' > chilliwack.html

$ git add . | git commit -m '3rd commit: 3 files'

We are about to git cherry-pick from another branch, and specifically, we will be pulling in the second commit, but before we do we will delete all of these files and perform a commit to put the master branch back into an empty state.

/c/ git cherry-pick example (master)  
$ rm \*.html  
$ git add . | git commit -m ‘4th commit: 0 files’

[master d6a8ce2] 4th commit: 0 files

3 files changed, 3 deletions(-)

delete mode 100644 abba.html

delete mode 100644 bowie.html

### Inspecting the commit history

Issuing a git reflog command will show the rich commit history of the master branch. Note the hexadecimal id of the second commit, 63162ea, as this is the one we will use when we git cherry-pick from another branch.

/c/ git cherry-pick example (master)

$ git reflog

d6a8ce2 (HEAD -> master) HEAD@{0}: commit: 4th commit: 0 files

bc0f7d1 HEAD@{1}: commit: 3rd commit: 3 files

63162ea HEAD@{2}: commit: 2nd commit: 2 files

6adc6ff HEAD@{3}: commit (initial): 1st commit: 1 file

### Switching to a feature branch

We will now create and move development onto a new branch named feature.

/c/ git cherry-pick example **(master)**

$ git branch feature

$ git checkout feature

Switched to branch 'feature'

/c/ git cherry-pick example **(feature)**

We will then create one file named zip.html and commit this file in order to create a small history of development on the feature branch.

/c/ git cherry-pick example **(feature)**

$ echo 'zip' > zip.html

$ git add . | git commit -m '1st feature branch commit: 1 file'

The next step is to git cherry pick from another branch to this new one, but before we do, think about what the expected result is. We will cherry-pick the 2nd commit from the master branch, namely the commit where the file named bowie.html was created. In the other branch, the bowie.html file sits alongside the abba.html file, which was created prior. What will the cherry-pick bring back? Will it bring back the abba.html and bowie.html files? Will it resurrect just the bowie.html file? Or will the command fail as we try to git cherry-pick across branches? Let’s see what happens.

### How to git cherry-pick across branches

The id of the bowie.html commit was 63162ea, so the command to git cherry-pick is:

/c/ git cherry-pick example (feature)

**$ git cherry-pick 63162ea**

[feature d1c9693] 2nd commit: 2 files

Date: Thu May 17 17:02:12 2018 -0400

1 file changed, 1 insertion(+)

create mode 100644 bowie.html

$ ls

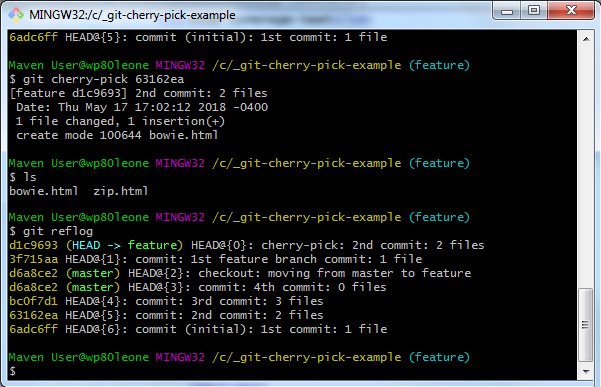
**bowie.html** zip.html

The output of the command to git cherry-pick from another branch is a single file being added to the current working tree, namely the bowie.html file. The directory listing command issued above shows two files, the zip.html file and the bowie.html file, indicating that the only change to the working tree was the addition of the second file.

### How git cherry-pick works

As you can see from this example, when you cherry-pick, what is returned is not the entire state of the branch at the time the commit happened, but instead, only the delta between the commit that happened and the state of the git repository prior to the cherry-picked commit.

It should also be noted that any time you git cherry-pick from another branch, a new commit gets registered in the branch history, as is evidenced by the following reflog:

[](https://cdn.ttgtmedia.com/ITKE/uploads/blogs.dir/319/files/2018/05/Reflog-after-a-git-cherry-pick-from-another-branch-1.jpg)

Needing to git cherry-pick from another branch is a common occurrence during software development cycles. As you can see from this example, so long as the hexadecimal id of the commit is known, performing a git cherry-pick from another branch is a safe and rather simple function to perform, especially if the branch doing the cherry-pick can merge the change without any clashes or conflicts occurring.

### Further improve your DevOps tools knowledge

Mastered Git? The next step in your DevOps journey is mastering Jenkins CI. Here are some great Jenkins tutorials that will take you from beginner to expert.

Step 1 — How to [download, configure and install Jenkins tutorial](https://www.theserverside.com/tutorial/Need-a-CI-tool-Heres-a-Jenkins-tutorial-for-beginners)

Step 2 — Create your first [Jenkins freestyle build job example](https://www.theserverside.com/tutorial/Create-a-Jenkins-build-job-A-CI-tutorial-for-beginners)

Step 3 — Pull from the [Jenkins environment variables list](https://www.theserverside.com/tutorial/Jenkins-environment-variables-list-for-shell-script-build-jobs) in your shell scripts

Step 4 — Fix common [Jenkins plugin installation errors](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/SunCertPathBuilderException-fix-for-Jenkins-plugin-download)

Step 5 — Add [String and Boolean Jenkins parameters](https://www.theserverside.com/video/Modify-CI-jobs-with-this-Jenkins-parameterized-build-example) to your builds

Step 6 — Start [pulling from GitHub with the Jenkins Git plugin](https://www.theserverside.com/tutorial/Attain-Jenkins-Git-integration-with-a-GitHub-pull-request)

Step 7 — What happens when you [git reset hard and push](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-a-git-reset-and-push-to-remote-works-on-previous-local-commits)?

Step 8 — Get a handle on the  [Jenkins vs. Maven debate](https://www.theserverside.com/opinion/Jenkins-vs-Maven-Compare-these-build-and-integration-tools)

Step 9 — Learn how to do a [hard git reset on a commit](https://www.theserverside.com/video/How-to-use-the-git-reset-hard-command-to-change-a-commit-history)

### Tough Jenkins interview questions answered

Here are the aforementioned answers to the advanced Jenkins interview.

**1. What are the software prerequisites that must be met before Jenkins is installed?**

Since version 2.54, Jenkins requires an installation of the Java Development Kit (JDK). [JAVA\_HOME](https://www.theserverside.com/tutorial/How-to-install-the-JDK-on-Windows-and-setup-JAVA_HOME) should also be configured prior to installation. Version 1.8 of the JDK is the minimum that Jenkins will support.

Jenkins also requires the services of a Jakarta Enterprise Edition (EE) web profile-compliant servlet engine in order to run, although Jenkins comes with an [embedded Jetty runtime](https://www.theserverside.com/tutorial/How-to-embed-Tomcat-and-Java-web-apps-in-an-executable-JAR) that can be used if an existing Tomcat, WildFly or WebSphere application server is not available.

**2. What is the syntax Jenkins uses to schedule items such as build jobs and SVN polling?**

Jenkins uses the cron syntax to schedule various items within the tool.

The cron syntax is represented by five asterisks, with each one separated by a space. The first asterisk represents minutes, the second represents hours, the third the day of the month, the fourth the month itself and the fifth the day of the week. For example, to schedule a build job to pull from GitHub every Friday at 5:30 p.m., the syntax would be: 30 17 \* \* 4.

**3. Name a Jenkins environment variable you have used in a shell script or batch file.**

There are a number of [environment variables](https://www.theserverside.com/tutorial/Jenkins-environment-variables-list-for-shell-script-build-jobs) that are available by default in any Jenkins build job. A few commonly used ones include:

* $JOB\_NAME
* $NODE\_NAME
* $WORKSPACE
* $BUILD\_URL
* $JOB\_URL

Note that, as new Jenkins plug-ins are configured, more environment variables become available. For example, when the Jenkins Git plug-in is configured, new [Jenkins Git environment variables](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Complete-Jenkins-Git-environment-variables-list-for-batch-jobs-and-shell-script-builds), such as $GIT\_COMMIT and $GIT\_URL, become available to be used in scripts.

This isn't one of the most overly advanced Jenkins interview questions and answers, but it does demonstrate that the interviewee has indeed created a scripted Jenkins build job before.

**4. Name three security mechanisms Jenkins uses to authenticate users.**

Jenkins can authenticate users in one of three ways:

1. Jenkins can use an internal database to store user data and credentials. (This is the default.)
2. Jenkins can be configured to authenticate against a Lightweight Directory Access Protocol server.
3. Jenkins can be configured to employ the authentication mechanism used by the application server upon which it is deployed.

**5. Describe the standard process to configure and use third-party tools within Jenkins?**

The process to use a third-party tool, such as [Artifactory](https://www.theserverside.com/video/How-to-set-up-and-use-a-JFrog-Artifactory-Maven-repository), Node, [SonarQube](https://www.theserverside.com/video/How-to-use-the-SonarQube-Maven-Plugin-for-continuous-inspection)or [Git](https://www.theserverside.com/tutorial/Five-basic-Git-commands-every-beginner-needs-to-know) typically follows a four-step process.

1. The third-party software must be installed.
2. A Jenkins plug-in that supports the third-party tool must be installed through the Jenkins admin console.
3. The third-party tool must be configured in the Tools tab of the Manage Jenkins section of the admin console.
4. Finally, the plug-in can be used from within a Jenkins build job. The plug-in will then facilitate communication between the Jenkins build job and the third-party tool.

This is a tough Jenkins interview question for DevOps professionals because not every third-party tool is configured in exactly the same way. For example, Jenkins can be configured to [install Maven](https://www.theserverside.com/tutorial/How-to-install-Maven-and-build-apps-with-the-mvn-command-line) itself, rather than requiring a pre-existing installation. Similarly, third-party tools, like [Checkstyle](https://www.theserverside.com/video/Use-Maven-Checkstyle-Plugin-to-enforce-Java-quality-rules) or JaCoCo, can be downloaded at build time by Maven. So these four steps are not always adhered to strictly, but at a high level, these are the typical steps required to install and configure a third-party Jenkins tool.

**6. Name two ways a Jenkins node agent can be configured to communicate back with the Jenkins master.**

The tool provides two mechanisms for starting a Jenkins node agent:

1. Launch a Jenkins node agent from a browser window.
2. Launch a Jenkins node agent from the command line.

When a Jenkins node agent is launched from a browser, a [JNLP file](https://whatis.techtarget.com/fileformat/JNLP-Java-Web-Start-Sun-Microsystems-Inc) is downloaded. When it runs, the JNLP file launches a new process on the client machine that runs Jenkins jobs.

To launch from the command line, the agent.jar file is required on the client. This [executable JAR](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Embed-the-WebSphere-Liberty-profile-in-an-executable-JAR-with-Maven) file is run from the command line, along with a reference to the slave agent's JNLP file that is hosted on the server. Like the JNLP file downloaded through a web browser, running this command launches a process on the client that can communicate with the Jenkins master and run Jenkins build jobs when it has idle clock cycles.

**7. How do you take a backup of your Jenkins build jobs in order to prepare for disaster recovery?**

Each Jenkins build stores its configuration as XML in a subdirectory of the JENKINS\_HOME\jobs folder. By copying this folder to a secondary location, the configuration of all of the build jobs managed by the Jenkins master will be backed up as a result. Checking this folder into a [source code management tool like Git](https://www.theserverside.com/tutorial/How-to-git-revert-a-commit-A-simple-undo-changes-example) isn't a bad idea either. Knowledge of ways to perform [Jenkins Git integration](https://www.theserverside.com/tutorial/Attain-Jenkins-Git-integration-with-a-GitHub-pull-request) is always looked upon fondly in a DevOps interview.

By simply copying the contents of this folder to a new Jenkins server instance, all of the build jobs described in this folder will be restored the next time the Jenkins server is started.

It's also worth noting that a history of all build jobs that have been run is written to this folder as well. To really impress the interviewer asking this advanced Jenkins interview question for DevOps, mention that using a .gitignore file and ignoring job-related files will ensure that any Git repository that stores Jenkins project configuration data will make sure only configuration data is stored. Build job history should be archived elsewhere and not stored in a source code repository.

**8. Name three steps or stages a typical Jenkins pipeline might include.**

A full-blown Jenkins pipeline will build a project from source code, put it through a variety of unit, integration, performance and user acceptance tests, and then, finally, if every test succeeds, deploy a packaged application to an application server, Nexus repository or Docker container. So, three fundamental stages would be:

1. Build
2. Test
3. Deploy

Of course, the best way to implement a Jenkins pipeline as code is to employ many modular steps. To really impress the interviewer asking one of these tough Jenkins interview questions for DevOps engineers, expand on these three stages, and describe a more complete Jenkins pipeline as code example.

#### Advanced Jenkins interview questions and answers

A more comprehensive Jenkins pipeline might be broken down even further into something like this:

* Pull from GitHub using the Jenkins Git plug-in.
* Compile the application using the Jenkins Maven plug-in.
* Ensure developers are complying with coding standards by using the Jenkins Checkstyle Plugin.
* Calculate McCabe cyclomatic complexity, and using that data, calculate unit test coverage. The JaCoCo plug-in is often used to perform code coverage calculations.
* Check for bugs and potential security flaws with static code analysis tools, such as SonarQube, PMD or FindBugs.
* Obtain manual sign-off by the user acceptance team by including a Groovy script in the Jenkins pipeline.
* Run stress tests, and measure the application's performance under load.
* Package the application in a format suitable for deployment. This is often a WAR file for web apps, an EAR file for enterprise Java apps or an executable JAR file for Docker-based microservices.
* Deploy the application to a Maven repository, such as Nexus or Artifactory.
* Archive all of the generated reports for future reference.

**9. How can you temporarily turn off Jenkins security if the administrative users have locked themselves out of the admin console?**

The JENKINS\_HOME folder contains a file named config.xml. When security is enabled, this file contains an XML element named useSecurity that will be set to true. By changing this setting to false, security will be disabled the next time Jenkins is restarted.

<useSecurity>false</useSecurity>

When asked one of these advanced Jenkins interview questions on DevOps security, be sure to emphasize that disabling security should always be both a last resort and a temporary measure. Once any authentication issues are resolved, be sure to re-enable Jenkins security and reboot the CI server.

**10. Polling a Git repository for new commits is considered a Jenkins anti-pattern. What is a sound alternative to SVN polling?**

Constantly polling a source code management tool like Git or Subversion to check if a new commit has been issued is a waste of clock cycles and should be avoided.

A better approach is to reverse this process and have the source code tool trigger a Jenkins build when new commits happen. With GitHub or GitLab, it is relatively easy to configure a post-commit hook that runs every time a commit is successful. When provided with the URL of the Jenkins build, the post-commit hook can easily trigger a Jenkins build, eliminating the need to have Jenkins constantly poll the [source code repository](https://www.theserverside.com/video/How-to-use-the-git-reset-hard-command-to-change-a-commit-history).

### Impress on that Jenkins interview

These are indeed a set of tough Jenkins interview questions and answers for DevOps engineers and aspiring cloud-native practitioners [to tackle](https://github.com/spikenode/DevOps-Interview-Questions). But if you are comfortable with the responses provided here and can comfortably put these responses into your own words, you'll have no problem impressing your future employer with your ability to respond to a variety of advanced Jenkins interview questions that cover a wide and disparate range of DevOps topics.

**Python**

**What is the maximum possible length of an identifier?**

a) 31 characters

b) 63 characters

c) 79 characters

d) none of the mentioned

**Answer: d**

**Explanation:** Identifiers can be of any length.

 ———————————————————–

**Why are local variable names beginning with an underscore discouraged?**

a) they are used to indicate a private variable of a class

b) they confuse the interpreter

c) they are used to indicate global variables

d) they slow down execution

**Answer: a**

**Explanation:** As Python has no concept of private variables, leading underscores are used to indicate variables that must not be accessed from outside the class.

 ——————————————————————–

**What is the output of print 0.1 + 0.2 == 0.3?**

a) True

b) False

c) Machine dependent

d) Error

**Answer: b**

**Explanation:** Neither of 0.1, 0.2 and 0.3 can be represented accurately in binary. The round off errors from 0.1 and 0.2 accumulate and hence there is a difference of 5.5511e-17 between (0.1 + 0.2) and 0.3

 ———————————————————————–

**What is the result of round(0.5) – round(-0.5)?**

a) 1.0

b) 2.0

c) 0.0

d) None of the mentioned

**Answer: b**

**Explanation**: Python rounds off numbers away from 0 when the number to be rounded off is exactly halfway through. round(0.5) is 1 and round(-0.5) is -1.

—————————————————————————-

**What is the output of the following expression if x=456?**

 print(“%-06d”%x)

a) 000456

b) 456000

c) 456

d) error

**Answer: c**

**Explanation:** The expression shown above results in the output 456.

 ————————————————————————————-

**What is the output of the following?**

 i = 0

while i < 3:

    print(i)

    i += 1

else:

    print(0)

 a) 0 1 2 3 0

b) 0 1 2 0

c) 0 1 2

c) error

**Answer: b**

**Explanation:** The else part is executed when the condition in the while statement is false

———————————————————————————————-

**What is the output of the following?**

 x = “abcdef”

i = “a”

while i in x[:-1]:

    print(i, end = ” “)

 a) a a a a a

b) a a a a a a

c) a a a a a a …

d) a

**Explanation:** String x is not being altered and i is in x[:-1].

——————————————————————————————————

**What is the output of the following?**

 print(‘ab12’.isalnum())

a) True

b) False

c) None

d) Error

**Answer: a**

**Explanation:** The string has only letters and digits.

 ———————————————————————————————————–

**Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after list example.pop()?**

a) [3, 4, 5, 20, 5, 25, 1].

b) [1, 3, 3, 4, 5, 5, 20, 25].

c) [3, 5, 20, 5, 25, 1, 3].

d) [1, 3, 4, 5, 20, 5, 25].

**Answer: a**

**Explanation:** pop() by default will remove the last element.

 ————————————————————————————————————–

**What is the output of the code shown below?**

 l1=[1,2,3]

l2=[4,5,6]

[x\*y for x in l1 for y in l2]

a) [4, 8, 12, 5, 10, 15, 6, 12, 18]

b) [4, 10, 18]

c) [4, 5, 6, 8, 10, 12, 12, 15, 18]

d) [18, 12, 6, 15, 10, 5, 12, 8, 4]

**Answer: c**

**Explanation:** The code is shown above returns x\*y, where x belongs to the list l1 and y belongs to the list l2. Therefore, the output is: [4, 5, 6, 8, 10, 12, 12, 15, 18].

 ———————————————————————————————————

**Which of the following statements create a dictionary?**

a) d = {}

b) d = {“john”:40, “peter”:45}

c) d = {40:”john”, 45:”peter”}

d) All of the mentioned

**Answer: d**

**Explanation:** Dictionaries are created by specifying keys and values.

 —————————————————————————————————-

**What is the output of the below program?**

 lamb = lambda x: x \*\* 3

print(lamb(5))

 a) 15

b) 555

c) 125

d) None of the mentioned

**Answer: c**

 —————————————————————————————————

**What is the value returned by Math.floor(3.4)?**

a) 3

b) 4

c) 4.0

d) 3.0

**Answer: a**

**Explanation:** The floor function returns the biggest number that is smaller than or equal to the number itself.

 ———————————————————————————————————

**What is the output of the code shown below?**

 l=[2, 3, [4, 5]]

l2=l.copy()

l2[0]=88

l

l2

 a) [88, 2, 3, [4, 5]]

[88, 2, 3, [4, 5]]

 b) [2, 3, [4, 5]]

[88, 2, 3, [4, 5]]

 c) [88, 2, 3, [4, 5]]

[2, 3, [4, 5]]

 d) [2, 3, [4, 5]]

[2, 3, [4, 5]]

**Answer: b**

**Explanation:** The code shown above depicts a deep copy. In a deep copy, the base address of the objects is not copied. Hence the modification done on one list does not affect the other list.

<https://www.linuxhelp.com/how-to-add-local-folder-to-a-gitlab-project>

## How to add local folder to a GitLab project

GitLab is an open source application used to code, test and deploy code together. It provides the Git repository management with access controls, code reviews, issue tracking and activity feeds. This tutorial will explain about the process of adding local folder to a GitLab project.

### Configuration procedure

To start the configuration procedure, first move to the local folder of the Git using the cd command.

[user1@localhost Desktop]$ cd git/

Run the Git init command to initialize Git repository in the target system.

[user1@localhost git]$ git init

Initialized empty Git repository in /home/user1/Desktop/git/.git/

Add the project path in which the local folder is to be copied and execute the command.

[user1@localhost git]$ git remote add origin git@192.168.7.212:user1/sample-project.git

Now add the files to the list which has to be tracked. To add the whole files give “ .”

[user1@localhost git]$ git add .

Run the Git commit command to list the files that has to be pushed.

[user1@localhost git]$ git commit -m ' 1st commit'

est-svgs/tiger.svg

Now execute the following command to push the files to the project.

[user1@localhost git]$ git push -u origin master

Counting objects: 213, done.

Compressing objects: 100% (197/197), done.

Writing objects: 100% (213/213), 717.83 KiB, done.

Total 213 (delta 19), reused 0 (delta 0)

remote: Resolving deltas: 100% (19/19), done.

To git@192.168.7.212:user1/sample-project.git

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

Wasn' t that an easy configuration procedure? Now adding local folder to a GitLab project is easy.

o add a different folders on Gitlab using different branches.  
  
For example ,I do have branches named test1 test2  
  
First convert the branch from master branch to test1 branch  
'git checkout test1'  
From test1 branch, I am pushing the data to the origin  
  
git add .  
  
git commit -m "test1 Commit"  
  
git push origin test1  
  
Now again change the branch from test1 to test2 using the git checkout and follow the same procedures where while pushing the origin,change the tes1 as test2

**how to Add the new gitlab remote to your existing repository and push ?**

**A**

To Add the new gitlab remote to your existing repository and push use the following command #git remote add gitlab url-to-gitlab-repo  
#git push gitlab master

**Q**

**How to remove a directory from git repository?**

**A**

To remove a directory from git repository run the following command  
git rm -r one-of-the-directories  
git commit -m "Remove duplicated directory"  
git push origin (typically 'master', but not always)

**Q**

**How to start the git from terminal ?**

**A**

Run the Git init command to initialize Git repository in the target system.  
  
git init

**Q**

**What is the command to commit a git?**

**A**

Run the Git commit command to list the files that has to be pushed.  
  
git commit -m ' 1st commit'

**Q**

**how to push the Files in git?**

**A**

To push the Files in git use the following command  
  
git push -u origin master

## Problem

You accidentally deleted a branch in your Git repository.

## Solution

Make sure to perform all of this locally, and confirm your repo is in the state you desire before pushing to Bitbucket Cloud. It may also be a good idea to clone your current repo, and test these solutions out first.

1. If you just deleted the branch, you'll see something like this in your terminal:

Deleted branch <your-branch> (was <sha>)

1. To restore the branch, use:

git checkout -b <branch> <sha>

**If you don't know the 'sha' off the top of your head, you can:**

1. Find the 'sha' for the commit at the tip of your deleted branch using:

git reflog

1. To restore the branch, use:

git checkout -b <branch> <sha>

**If your commits are not in your reflog:**

1. You can try recovering a branch by reseting your branch to the sha of the commit found using a command like:

git fsck --full --no-reflogs --unreachable --lost-found | grep commit | cut -d\ -f3 | xargs -n 1 git log -n 1 --pretty=oneline > .git/lost-found.txt

1. You can then display each commit using one of these:
2. git log -p <commit>

git cat-file -p <commit>

# Why am I not prompted for password when pushing or pulling to my repositories in Bitbucket Cloud via HTTPS?

## Problem

When you push or pull to a private repository over HTTPS, you are not prompted for a password.

## Cause

The most likely reason for this is that Git has been configured to use a credential helper.  
The configuration could have been made a) for all users in your system b) only for your user c) for a specific repository.  
You can check which one of the 3 is the case by running these commands respectively:

git config --system credential.helper

git config --global credential.helper

git config --local credential.helper

In case a credential helper is configured, you should see it listed in the output like below:

$ git config --system credential.helper

osxkeychain

## Resolution

If you wish to be prompted for a password every time you push or pull to your remote repo, you can remove the credential helper from the configuration.  
Depending on where the credential helper is configured, you can unset it using the respective command:

git config --system --unset credential.helper

git config --global --unset credential.helper

git config --local --unset credential.helper

# Http status code 412 when using https

## Problem

When attempting to perform git activities over https with Bitbucket Cloud, you may encounter this error:

error: The requested URL returned error: 412 while accessing https://{username}@bitbucket.org/{username}/{repo name}.git/info/refs

fatal: HTTP request failed

## Cause

This error is due to not having a password set on your account. This can happen if you've created your account using the "Log in with Google|Twitter|Facebook|GitHub" and opted to not create a Bitbucket Cloud password during the sign-up process, as shown below. The actual error message you are seeing is a bug, which is being tracked [here](https://bitbucket.org/site/master/issue/6018/http-412-error-when-connecting-to.).

From time to time, you may encounter a situation where someone has accidentally pushed their changes to Bitbucket Cloud without first pulling the changes from the server. When this happens, Bitbucket Cloud will show that all the commits have been 'stripped'. Unfortunately, this means you now have a limited amount of time to fix this situation. The good news is that you can almost always fix this yourself locally, without any intervention from Bitbucket Cloud support. Here's the basics:

# How to restore orphaned or deleted commits

## Your commits aren't gone (yet)

When you use --force, you are adding new commits to Git and updating the branch reference of your branch. You have only orphaned the existing previous line of work as it is no longer referenced by anything. Git will gc (garbage collect) orphaned data on its own. It checks to see if it should gc every time you interact with Git. This means the moment it happens, you need to start work to recover it. There may be other ways to do this, but this is by far the easiest if you have a copy of the repo with the orphaned commits:

1. Make a backup of the repo locally.
2. Find the last commit that your 'master' (or other) branch should have been on before the push. You can use the 'stripped' hashes to find this or find another user who has a clone of the repo in the state you expect it to be in.
3. Create a new branch in your local repository with that commit as the head of the branch:

$ git branch master-recovery fg378o78

1. Verify that your data is in a state you expect it to be in
2. Now, push that branch into the 'master' branch

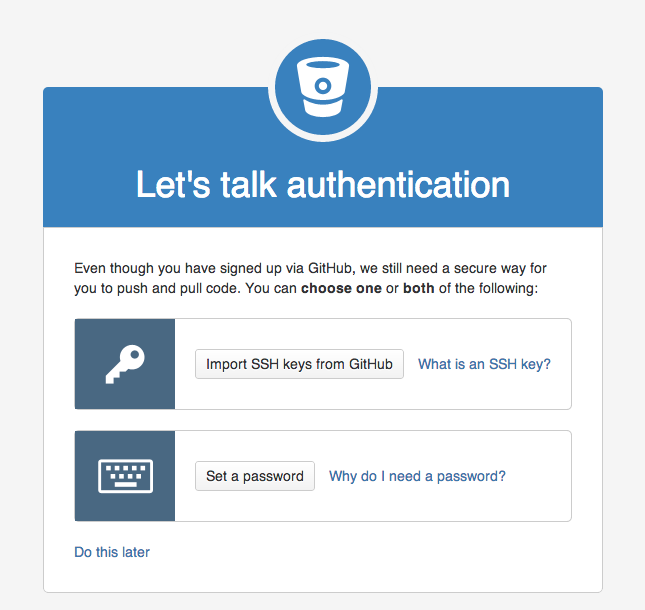
git push origin master-recovery:/refs/heads/master --force

1. You will be telling the remote Git that you want the current commit of master-recovery to be the current commit of master.
2. Now, pull master again from Bitbucket Cloud:

git pull origin master

1. You should now have your data locally and on Bitbucket Cloud back in the state it was in before the accidental push.

To prevent this in the future, you can [disable non-fast-forward pushes](https://confluence.atlassian.com/bitbucket/git-fast-forwards-and-branch-management-329977726.html) from any repository's admin page.



## Resolution

If you choose to use https to operate with Git, you'll need to set a password on your account. This can be done by going to the Manage account menu item in the upper right corner of any page. It can also be found directly at https://bitbucket.org/account/password/change/{YOUR USERNAME}/. If you do not wish to set a password on Bitbucket Cloud, you can still authenticate your actions using SSH. Please review [our guide to setting up SSH.](https://confluence.atlassian.com/display/BITBUCKET/Set+up+SSH+for+Git)

# fatal: 'origin' does not appear to be a git repository

## Problem

You receive the following error from Git

fatal: 'origin' does not appear to be a git repository

fatal: The remote end hung up unexpectedly

## Cause

This is typically because you have not set the **origin** alias on your Git repository.

## Resolution

To resolve, do the following:

git remote add origin URL\_TO\_YOUR\_REPO

This will add an alias in your .git/config file for the remote clone/push/pull site URL. This URL can be found on your repository Overview page.

If this doesn't resolve your issue, please read over our [Bitbucket Cloud 101](http://confluence.atlassian.com/) guide for detailed setup and configuration guides for Git.

# How to get started with GIT and work with GIT Remote Repo

## 

### 1.  Introduction

GIT is a Version Control System (VCS) (aka Revision Control System (RCS), Source Code Manager (SCM)). A VCS serves as a Repository (or repo) of program codes, including all the historical revisions. It records changes to files at so-called commits in a log so that you can recall any file at any commit point.

Why VCS?

1. The Repository serves as the backup (in case of code changes or disk crash).
2. It is a living archive of all historical revisions. It lets you revert back to a specific version, if the need arises.
3. It facilitates collaboration between team members, and serves as a project management tool.
4. more...

Git was initially designed and developed by Linus Torvalds, in 2005, to support the development of the Linux kernel.

GIT is a Distributed Version Control System (DVCS). Other popular VCSes include:

1. The standalone and legacy Unix's RCS (Revision Control System).
2. Centralized Client-Server Version Control System (CVCS): CVS (Concurrent Version System), SVN (Subversion) and Perforce.
3. Distributed VCS (DVCS): GIT, Merurial, Bazaar, Darcs.

The mother site for Git is [http://git-scm.com.](http://git-scm.com/)

### 2.  Setting Up Git

You need to setup Git on your local machine, as follows:

1. Download & Install:
   * For Windows and Mac, download the installer from <http://git-scm.com/downloads> and run the downloaded installer.
   * For Ubuntu, issue command "sudo apt-get install git".

For Windows, use the "Git Bash" command shell bundled with Git Installer to issue commands. For Mac/Ubuntu, use the "Terminal".

1. Customize Git:  
   Issue "git config" command (for Windows, run "Git Bash" from the Git installed directory. For Ubuntu/Mac, launch a "Terminal"):
2. // Set up your username and email (to be used in labeling your commits)
3. $ **git config --global user.name "your-name"**

$ **git config --global user.email "your-email@youremail.com"**

The settings are kept in "<GIT\_HOME>/etc/gitconfig" (of the GIT installed directory) and "<USER\_HOME>/.gitconfig" (of the user's home directory.  
You can issue "git config --list" to list the settings:

$ **git config --list**

user.email=xxxxxx@xxxxxx.com

user.name=xxxxxx

### 3.  Git Basics

##### Git Commands

Git provides a set of simple, distinct, standalone commands developed according to the "Unix toolkit" philosophy - build small, interoperable tools.

To issue a command, start a "Terminal" (for Ubuntu/Mac) or "Git Bash" (for Windows):

$ **git <command> <arguments>**

The commonly-used commands are:

1. **init**, **clone**, **config**: for starting a Git-managed project.
2. **add**, **mv**, **rm**: for staging file changes.
3. **commit**, **rebase**, **reset**, **tag**:
4. **status**, **log**, **diff**, **grep**, **show**: show status
5. **checkout**, **branch**, **merge**, **push**, **fetch**, **pull**

##### Help and Manual

The best way to get help these days is certainly googling.

To get help on Git commands:

$ **git help <command>**

// or

$ **git <command> --help**

The GIT manual is bundled with the software (under the "doc" directory), and also available online @ <http://git-scm.com/docs>.

#### 3.1  Getting Started with Local Repo

There are 2 ways to start a Git-managed project:

1. Starting your own project;
2. Cloning an existing project from a GIT host.

We shall begin with "Starting your own project" and cover "Cloning" later @ "[Clone a Project from a Remote Repo](https://www.ntu.edu.sg/home/ehchua/programming/howto/Git_HowTo.html#CloneProject)".

##### Setup the Working Directory for a New Project

Let's start a programming project under the working directory called "hello-git", with one source file "Hello.java" (or "Hello.cpp", or "Hello.c") as follows:

// Hello.java

public class Hello {

public static void main(String[] args) {

System.out.println("Hello, world from GIT!");

}

}

Compile the "Hello.java" into "Hello.class" (or "Hello.cpp" or "Hello.c" into "Hello.exe").

It is also highly recommended to provide a "README.md" file (a text file in a so-called "Markdown" syntax such as "[GitHub Flavored Markdown](https://help.github.com/articles/github-flavored-markdown)") to describe your project:

// README.md

This is the README file for the Hello-world project.

Now, we have 3 files in the working tree: "Hello.java", "Hello.class" and "README.md". We do not wish to track the ".class" as they can be reproduced from ".java".

##### Initialize a new Git Repo (git init)

To manage a project under Git, run "git init" at the project root directory (i.e., "hello-git") (via "Git Bash" for Windows, or "Terminal" for Ubuntu/Mac):

// Change directory to the project directory

$ **cd /path-to/hello-git**

// Initialize Git repo for this project

$ **git init**

Initialized empty Git repository in /path-to/hello-git/.git/

$ **ls -al**

drwxr-xr-x 1 xxxxx xxxxx 4096 Sep 14 14:58 .git

-rw-r--r-- 1 xxxxx xxxxx 426 Sep 14 14:40 Hello.class

-rw-r--r-- 1 xxxxx xxxxx 142 Sep 14 14:32 Hello.java

-rw-r--r-- 1 xxxxx xxxxx 66 Sep 14 14:33 README.md

A hidden sub-directory called ".git" will be created under your project root directory (as shown in the above "ls -a" listing), which contains ALL Git related data.

Take note that EACH Git repo is associated with a project directory (and its sub-directories). The Git repo is completely contain within the project directory. Hence, it is safe to copy, move or rename the project directory. If your project uses more than one directories, you may create one Git repo for EACH directory, or use symlinks to link up the directories, or ... (?!).

##### Git Storage Model

The local repo after "git init" is empty. You need to explicitly deposit files into the repo.

Before we proceed, it is important to stress that Git manages changes to files between so-called commits. In other words, it is a version control system that allows you to keep track of the file changes at the commits.

##### Staging File Changes for Tracking (git add <file>...)

Issue a "git status" command to show the status of the files:

$ **git status**

On branch master

Initial commit

Untracked files:

(use "git add <file>..." to include in what will be committed)

Hello.class

Hello.java

README.md

nothing added to commit but untracked files present (use "git add" to track)

By default, we start on a branch called "master". We will discuss "branch" later.

In Git, the files in the working tree are either untracked or tracked. Currently, all 3 files are untracked. To stage a new file for tracking, use "git add <file>..." command.

// Add README.md file

$ **git add README.md**

$ **git status**

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: README.md

Untracked files:

(use "git add <file>..." to include in what will be committed)

Hello.class

Hello.java

// You can use wildcard \* in the filename

// Add all Java source files into Git repo

$ **git add \*.java**

// You can also include multiple files in the "git add"

// E.g.,

// git add Hello.java README.md

$ **git status**

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: Hello.java

new file: README.md

Untracked files:

(use "git add <file>..." to include in what will be committed)

Hello.class

The command "git add <file>..." takes one or more filenames or pathnames with possibly wildcards pattern. You can also use "git add ." to add all the files in the current directory (and all sub-directories). But this will include "Hello.class", which we do not wish to be tracked.

When a new file is added, it is staged (or indexed, or cached) in the staging area (as shown in the GIT storage model), but NOT yet committed.

Git uses two stages to commit file changes:

1. "git add <file>" to stage file changes into the staging area, and
2. "git commit" to commit ALL the file changes in the staging area to the local repo.

The staging area allows you to group related file changes and commit them together.

##### Committing File Changes (git commit)

The "git commit" command commits ALL the file changes in the staging area. Use a -m option to provide a message for the commit.

$ **git commit -m "First commit"** // -m to specify the commit message

[master (root-commit) 858f3e7] first commit

2 files changed, 8 insertions(+)

create mode 100644 Hello.java

create mode 100644 README.md

// Check the status

$ **git status**

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

Hello.class

nothing added to commit but untracked files present (use "git add" to track)

##### Viewing the Commit Data (git log)

Git records several pieces of metadata for every commit, which includes a log message, timestamp, the author's username and email (set during customization).

You can use "git log" to list the commit data; or "git log --stat" to view the file statistics:

$ **git log**

commit 858f3e71b95271ea320d45b69f44dc55cf1ff794

Author: username <email>

Date: Thu Nov 29 13:31:32 2012 +0800

First commit

$ **git log --stat**

commit 858f3e71b95271ea320d45b69f44dc55cf1ff794

Author: username <email>

Date: Thu Nov 29 13:31:32 2012 +0800

First commit

Hello.java | 6 ++++++

README.md | 2 ++

2 files changed, 8 insertions(+)

Each commit is identified by a 40-hex-digit SHA-1 hash code. But we typcially use the first 7 hex digits to reference a commit, as highlighted.

To view the commit details, use "git log -p", which lists all the patches (or changes).

$ **git log -p**

commit 858f3e71b95271ea320d45b69f44dc55cf1ff794

Author: username <email>

Date: Thu Nov 29 13:31:32 2012 +0800

First commit

diff --git a/Hello.java b/Hello.java

new file mode 100644

index 0000000..dc8d4cf

--- /dev/null

+++ b/Hello.java

@@ -0,0 +1,6 @@

+// Hello.java

+public class Hello {

+ public static void main(String[] args) {

+ System.out.println("Hello, world from GIT!");

+ }

+}

diff --git a/README.md b/README.md

new file mode 100644

index 0000000..9565113

--- /dev/null

+++ b/README.md

@@ -0,0 +1,2 @@

+// README.md

+This is the README file for the Hello-world project.

Below are more options of using "git log":

$ **git log --oneline**

// Display EACH commit in one line.

$ **git log --author="<author-name-pattern>"**

// Display commits by author

$ **git log <file-pattern>**

// Display commits for particular file(s)

// EXAMPLES

$ **git log --author="Tan Ah Teck" -p Hello.java**

// Display commits for file "Hello.java" by a particular author

##### File Status (git status)

A file could be untracked or tracked.

As mentioned, Git tracks file changes at commits. In Git, changes for a tracked file could be:

1. unstaged (in Working Tree) - called unstaged changes,
2. staged (in Staging Area or Index or Cache) - called staged changes, or
3. committed (in local repo object database).

The files in "working tree" or "staging area" could have status of unmodified, added, modified, deleted, renamed, copied, as reported by "git status".

The "git status" output is divided into 3 sections: "Changes not staged for commit" for the unstaged changes in "working tree", "Changes to be committed" for the staged changes in the "staging area", and "Untracked files". In each section, It lists all the files that have been changed, i,e., files having status other than unmodified.

When a new file is created in the working tree, it is marked as new in working tree and shown as an untracked file. When the file change is staged, it is marked as new (added) in the staging area, and unmodified in working tree. When the file change is committed, it is marked as unmodified in both the working tree and staging area.

When a committed file is modified, it is marked as modified in the working tree and unmodified in the staging area. When the file change is staged, it is marked as modified in the staging area and unmodified in the working tree. When the file change is committed, it is marked as unmodified in both the working tree and staging area.

For example, made some changes to the file "Hello.java", and check the status again:

// Hello.java

public class Hello {

public static void main(String[] args) {

System.out.println("Hello, world from GIT!");

System.out.println("Changes after First commit!");

}

}

$ **git status**

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: Hello.java

Untracked files:

(use "git add <file>..." to include in what will be committed)

Hello.class

no changes added to commit (use "git add" and/or "git commit -a")

The "Hello.java" is marked as modified in the working tree (under "Changes not staged for commit"), but unmodified in the staging area (not shown in "Changes to be committed").

You can inspect all the unstaged changes using "git diff" command (or "git diff <file>" for the specified file). It shows the file changes in the working tree since the last commit:

$ **git diff**

diff --git a/Hello.java b/Hello.java

index dc8d4cf..f4a4393 100644

--- a/Hello.java

+++ b/Hello.java

@@ -2,5 +2,6 @@

public class Hello {

public static void main(String[] args) {

System.out.println("Hello, world from GIT!");

+ System.out.println("Changes after First commit!");

}

}

The older version (as of last commit) is marked as --- and new one as +++. Each chunk of changes is delimited by "@@ -<old-line-number>,<number-of-lines> +<new-line-number>,<number-of-lines> @@". Added lines are marked as + and deleted as -. In the above output, older version (as of last commit) from line 2 for 5 lines and the modified version from line 2 for 6 lines are compared. One line (marked as +) is added.

Stage the changes of "Hello.java" by issuing the "git add <file>...":

$ **git add Hello.java**

$ **git status**

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: Hello.java

Untracked files:

(use "git add <file>..." to include in what will be committed)

Hello.class

Now, it is marked as modified in the staging area ("Changes to be committed"), but unmodified in the working tree (not shown in "Changes not staged for commit").

Now, the changes have been staged. Issuing an "git diff" to show the unstaged changes results in empty output.

You can inspect the staged change (in the staging area) via "git diff --staged" command:

// List all "unstaged" changes for all files (in the working tree)

$ **git diff**

// empty output - no unstaged change

// List all "staged" changes for all files (in the staging area)

$ **git diff --staged**

diff --git a/Hello.java b/Hello.java

index dc8d4cf..f4a4393 100644

--- a/Hello.java

+++ b/Hello.java

@@ -2,5 +2,6 @@

public class Hello {

public static void main(String[] args) {

System.out.println("Hello, world from GIT!");

+ System.out.println("Changes after First commit!");

}

}

// The "unstaged" changes are now "staged".

Commit ALL staged file changes via "git commit":

$ **git commit -m "Second commit"**

[master 96efc96] Second commit

1 file changed, 1 insertion(+)

$ **git status**

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

Hello.class

nothing added to commit but untracked files present (use "git add" to track)

Once the file changes are committed, it is marked as unmodified in the staging area (not shown in "Changes to be committed").

Both "git diff" and "git diff --staged" return empty output, signalling there is no "unstaged" and "staged" changes.

The stage changes are cleared when the changes are committed; while the unstaged changes are cleared when the changes are staged.

Issue "git log" to list all the commits:

$ **git log**

commit 96efc96f0856846bc495aca2e4ea9f06b38317d1

Author: username <email>

Date: Thu Nov 29 14:09:46 2012 +0800

Second commit

commit 858f3e71b95271ea320d45b69f44dc55cf1ff794

Author: username <email>

Date: Thu Nov 29 13:31:32 2012 +0800

First commit

Check the patches for the latest commit via "git log -p -1", with option -n to limit to the last n commit:

$ **git log -p -1**

commit 96efc96f0856846bc495aca2e4ea9f06b38317d1

Author: username <email>

Date: Thu Nov 29 14:09:46 2012 +0800

Second commit

diff --git a/Hello.java b/Hello.java

index dc8d4cf..ede8979 100644

--- a/Hello.java

+++ b/Hello.java

@@ -2,5 +2,6 @@

public class Hello {

public static void main(String[] args) {

System.out.println("Hello, world from GIT!");

+ System.out.println("Changes after First commit!");

}

}

I shall stress again Git tracks the "file changes" at each commit over the previous commit.

##### The .gitignore File

All the files in the Git directory are either tracked or untracked. To ignore files (such as .class, .o, .exe which could be reproduced from source) from being tracked and remove them from the untracked file list, create a ".gitignore" file in your project directory, which list the files to be ignored, as follows:

# .gitignore

# Java class files

\*.class

# Executable files

\*.exe

# Object and archive files

# Can use regular expression, e.g., [oa] matches either o or a

\*.[oa]

# temp sub-directory (ended with a directory separator)

temp/

There should NOT be any trailing comments for filename. You can use regexe for matching the filename/pathname patterns, e.g. [oa] denotes either o or a. You can override the rules by using the inverted pattern (!), e.g., Adding !hello.exe includes the hello.exe although \*.exe are excluded.

Now, issue a "git status" command to check the untracked files.

$ **git status**

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

nothing added to commit but untracked files present (use "git add" to track)

Now, "Hello.class" is not shown in "Untracked files".

Typically, we also track and commit the .gitignore file.

$ **git add .gitignore**

$ **git status**

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: .gitignore

$ **git commit -m "Added .gitignore"**

[master 711ef4f] Added .gitignore

1 file changed, 14 insertions(+)

create mode 100644 .gitignore

$ **git status**

On branch master

nothing to commit, working directory clean

#### 3.2  Setting up Remote Repo

1. Sign up for a GIT host, such as Github <https://github.com/signup/free> (Unlimited for public projects; fee for private projects); or BitBucket @ <https://bitbucket.org/> (Unlimited users for public projects; 5 free users for private projects; Unlimited for Academic Plan); among others.
2. Login to the GIT host. Create a new remote repo called "test".
3. On your local repo (let's continue to work on our "hello-git" project), set up the remote repo's name and URL via "git remote add <remote-name> <remote-url>" command.  
   By convention, we shall name our remote repo as "origin". You can find the URL of a remote repo from the Git host. The URL may take the form of HTTPS or SSH. Use HTTPS for simplicity.
4. // Change directory to your local repo's working directory
5. $ **cd /path-to/hello-git**
7. // Add a remote repo called "origin" via "git remote add <remote-name> <remote-url>"
8. // For examples,
9. $ **git remote add origin https://github.com/your-username/test.git** // for GitHub

$ **git remote add origin https://username@bitbucket.org/your-username/test.git** // for Bitbucket

You can list all the remote names and their corresponding URLs via "git remote -v", for example,

// List all remote names and their corresonding URLs

$ **git remote -v**

origin https://github.com/your-username/test.git (fetch)

origin https://github.com/your-username/test.git (push)

Now, you can manage the remote connection, using a simple name instead of the complex URL.

1. Push the commits from the local repo to the remote repo via "git push -u <remote-name> <local-branch-name>".  
   By convention, the main branch of our local repo is called "master" (as seen from the earlier "git status" output). We shall discuss "branch" later.
2. // Push all commits of the branch "master" to remote repo "origin"
3. $ **git push origin master**
4. Username for 'https://github.com': \*\*\*\*\*\*
5. Password for 'https://your-username@github.com': \*\*\*\*\*\*\*
6. Counting objects: 10, done.
7. Delta compression using up to 8 threads.
8. Compressing objects: 100% (10/10), done.
9. Writing objects: 100% (10/10), 1.13 KiB | 0 bytes/s, done.
10. Total 10 (delta 1), reused 0 (delta 0)
11. To https://github.com/your-username/test.git
12. \* [new branch] master -> master

Branch master set up to track remote branch master from origin.

1. Login to the GIT host and select the remote repo "test", you shall find all the committed files.
2. On your local system, make some change (e.g., on "Hello.java"); stage and commit the changes on the local repo; and push it to the remote. This is known as the "Edit/Stage/Commit/Push" cycle.
3. // Hello.java
4. public class Hello {
5. public static void main(String[] args) {
6. System.out.println("Hello, world from GIT!");
7. System.out.println("Changes after First commit!");
8. System.out.println("Changes after Pushing to remote!");
9. }

}

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working dire

modified: Hello.java

no changes added to commit (use "git add" and/or "git commit -a")

// Stage file changes

$ **git add \*.java**

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: Hello.java

// Commit all staged file changes

$ **git commit -m "Third commit"**

[master 744307e] Third commit

1 file changed, 1 insertion(+)

// Push the commits on local master branch to remote

$ **git push origin master**

Username for 'https://github.com': \*\*\*\*\*\*

Password for 'https://username@github.com': \*\*\*\*\*\*

Counting objects: 5, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 377 bytes | 0 bytes/s, done.

Total 3 (delta 1), reused 0 (delta 0)

To https://github.com/your-username/test.git

711ef4f..744307e master -> master

Again, login to the remote to check the committed files.

#### 3.3  Cloning a Project from a Remote Repo (git clone <remote-url>)

As mentioned earlier, you can start a local GIT repo either running "git init" on your own project, or "git clone <remote-url>" to copy from an existing project.

Anyone having read access to your remote repo can clone your project. You can also clone any project in any public remote repo.

The "git clone <remote-url>" initializes a local repo and copies all files into the working tree. You can find the URL of a remote repo from the Git host.

// SYNTAX

// ======

$ **git clone <remote-url>**

// <url>: can be https (recommended), ssh or file.

// Clone the project UNDER the current directory

// The name of the "working directory" is the same as the remote project name

$ **git clone <remote-url> <working-directory-name>**

// Clone UNDER current directory, use the given "working directory" name

// EXAMPLES

// ========

// Change directory (cd) to the "parent" directory of the project directory

$ **cd path-to-parent-of-the-working-directory**

// Clone our remote repo "test" into a new working directory called "hello-git-cloned"

$ **git clone https://github.com/your-username/test.git hello-git-cloned**

Cloning into 'hello-git-cloned'...

remote: Counting objects: 13, done.

remote: Compressing objects: 100% (11/11), done.

remote: Total 13 (delta 2), reused 13 (delta 2)

Unpacking objects: 100% (13/13), done.

Checking connectivity... done.

// Verify

$ **cd hello-git-cloned**

$ **ls -a**

. .. .git .gitignore Hello.java README.md

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

The "git clone" automatically creates a remote name called "origin" mapped to the cloned remote-URL. You can check via "git remote -v":

// List all the remote names

$ **git remote -v**

origin https://github.com/your-username/test.git (fetch)

origin https://github.com/your-username/test.git (push)

#### 3.4  Summary of Basic "Edit/Stage/Commit/Push" Cycle

// Edit (Create, Modified, Rename, Delete) files,

// which produces "unstaged" file changes.

// Stage file changes, which produces "Staged" file changes

$ **git add <file>** // for new and modified files

$ **git rm <file>** // for deleted files

$ **git mv <old-file-name> <new-file-name>** // for renamed file

// Commit (ALL staged file changes)

$ **git commit -m "message"**

// Push

$ **git push <remote-name> <local-branch-name>**

OR,

// Stage ALL files with changes

$ **git add -A** // OR, '**git add --all**'

$ **git commit -m "message"**

$ **git push**

OR,

// Add All and Commit in one command

$ **git commit -a -m "message"**

$ **git push**

#### 3.5  More on Staged and Unstaged Changes

If you modify a file, stage the changes and modify the file again, there will be staged changes and unstaged changes for that file.

For example, let's continue the "hello-git" project. Add one more line to "README.md" and stage the changes:

// README.md

This is the README file for the Hello-world project.

Make some changes and staged.

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:

modified: README.md

$ **git add README.md**

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:

modified: README.md

Before the changes are committed, suppose we modify the file again:

// README.md

This is the README file for the Hello-world project.

Make some changes and staged.

Make more changes before the previous changes are committed.

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:

modified: README.md

Changes not staged for commit:

modified: README.md

// Now, "README.md" has both unstaged and staged changes.

// Show the staged changes

$ **git diff --staged**

diff --git a/README.md b/README.md

index 9565113..b2e9afb 100644

--- a/README.md

+++ b/README.md

@@ -1,2 +1,3 @@

// README.md

This is the README file for the Hello-world project.

+Make some changes and staged.

// Show the unstaged changes

$ **git diff**

diff --git a/README.md b/README.md

index b2e9afb..ca6622a 100644

--- a/README.md

+++ b/README.md

@@ -1,3 +1,4 @@

// README.md

This is the README file for the Hello-world project.

Make some changes and staged.

+Make more changes before the previous changes are committed.

// Stage the changes

$ **git add README.md**

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:

modified: README.md

// Show staged changes

$ **git diff --staged**

diff --git a/README.md b/README.md

index 9565113..ca6622a 100644

--- a/README.md

+++ b/README.md

@@ -1,2 +1,4 @@

// README.md

This is the README file for the Hello-world project.

+Make some changes and staged.

+Make more changes before the previous changes are committed.

// Commit the staged changes

$ **git commit -m "Unstaged vs. Staged Changes"**

[master a44199b] Unstaged vs. Staged Changes

1 file changed, 2 insertions(+), 0 deletion(-)

Take note that the stage changes are cleared when the changes are committed; while the unstaged changes are cleared when the changes are staged.

For convenience, you can also use the "git-gui" tool to view the unstaged and staged changes.

#### 3.6  Git GUI Tools

##### Git-GUI (Windows)

For convenience, Git provides a GUI tool, called git-gui, which can be used to perform all tasks and view the commit log graphically.

Install "Git-Gui".

To run the git-gui, you can right-click on the project folder and choose "Git Gui"; or launch the Git-bash shell and run "git gui" command.

To view the log, choose "Repository" ⇒ "Visualize master's history", which launches the "gitk". You can view the details of each commit.

You can also view each of the file via "Repository" ⇒ "Browse master's Files" ⇒ Select a file.

Git-gui is bundled with Git. To launch git-gui, right click on the working directory and choose "git gui", or run "git gui" command on the Git-Bash shell.

[TODO]

##### EGit Plugin for Eclipse

[TODO]

### 4.  Tagging

Tag (or label) can be used to tag a specific commit as being important, for example, to mark a particular release. The release is often marked in this format: *version-number*.release-no.modificaton-no (e.g., v1.1.5) or or *version-number*.release-no.upgrade-no\_modificaton-no (e.g., v1.7.0\_26).

I recommend that you commit your code and push it to the remote repo as often as needed (e.g., daily), to BACKUP your code. When you code reaches a stable point (in turn of functionality), create a tag to mark the commit, which can then be used for CHECKOUT, if you need to show your code to others.

##### Listing Tags (git tag)

To list the existing tags, use "git tag" command.

##### Types of Tags - Lightweight Tags and Annotated Tags

There are two kinds of tags: lightweight tag and annotated tag. Lightweight tag is simply a pointer to a commit. Annotated tag contains annotations (meta-data) and can be digitally signed and verified.

##### Creating an Annotated Tag (git tag -a <tag-name> -m <message>)

To create an annotated tag at the latest commit, use "git tag -a <tag-name> -m <message>", where -a option specifies annotation tag having meta-data. For example,

$ **git tag -a v1.0.0 -m "First production system"**

// List all tags

$ **git tag**

v1.0.0

// Show tag details

$ **git show v1.0.0**

// Show the commit point and working tree

To create a tag for an earlier commit, you need to find out the commit's name (first seven character hash code) (via "git log"), and issue "git tag -a <tag-name> -m <message> <commit-name>". For example,

$ **git log**

......

commit 7e7cb40a9340691e2b16a041f7185cee5f7ba92e

......

Commit 3

$ **git tag -a "v0.9.0" -m "Last pre-production release" 7e7cb40**

// List all tags

$ **git tag**

v0.9.0

v1.0.0

// Show details of a tag

$ **git show v0.9.0**

......

[TODO] Diagram

##### Creating Lightweight Tags (git tag <tag-name>)

To create a lightweight tag (without meta-data), use "git tag <tag-name>" without the -a option. The lightweight tag stores only the commit hash code.

##### Signed Tags

You can signed your tags with your private key, with -s option instead of -a.

To verify a signed tag, use -v option and provide the signer's public key.

[TODO] Example

##### Pushing to Remote Repo

By default, Git does not push tags (and branches) to remote repo. You need to push them explicitly, via "git push origin <tag-name>" for a particular tag or "git push origin --tags" for all the tags.

### 5.  Branching/Merging

#### 5.1  Git's Data Structures

Git has two primary data structures:

1. an immutable, append-only object database (or local repo) that stores all the commits and file contents;
2. a mutable staging area (or index, or cache) that caches the staged information.

The staging area serves as the connection between object database and working tree (as shown in the storage model diagram). It serves to avoid volatility, and allows you to stage ALL the file changes before issuing a commit, instead of committing individual file change. Changes to files that have been explicitly added to the index (staging area) via "git add <file>" are called staged changes. Changes that have not been added are called unstaged changes. Staged and unstaged changes can co-exist. Performing a commit copies the statged changes into object database (local repo) and clears the index. The unstaged changes remain in working tree.

The object database contains these objects:

* Each version of a file is represented by a blob (binary large object - a file that can contain any data: binaries or characters). A blob holds the file data only, without any metadata - not even the filename.
* A snapshot of the working tree is represented by a tree object, which links the blobs and sub-trees for sub-directories.
* A commit object points to a tree object, i.e., the snapshot of the working tree at the point the commit was created. It holds metadata such as timestamp, log message, author's and committer's username and email. It also references its parent commit(s), except the root commit which has no parent. A normal commit has one parent; a merge commit could have multiple parents. A commit, where new branch is created, has more than one children. By referencing through the chain of parent commit(s), you can discover the history of the project.

Each object is identified (or named) by a 160-bit (or 40 hex-digit) SHA-1 hash value of its contents (i.e., a content-addressable name). Any tiny change to the contents produces a different hash value, resulted in a different object. Typically, we use the first 7 hex-digit prefix to refer to an object, as long as there is no ambiguity.

There are two ways to refer to a particular commit: via a branch or a tag.

* A branch is a mobile reference of commit. It moves forward whenever commit is made on that branch.
* A tag (like a label) marks a particular commit. Tag is often used for marking the releases.

#### 5.2  Branching

Branching allows you and your team members to work on different aspects of the software concurrently (on so-called feature branches), and merge into the master branch as and when they completes. Branching is the most important feature in a concurrent version control system.

A branch in Git is a lightweight movable pointer to one of the commits. For the initial commit, Git assigns the default branch name called master and sets the master branch pointer at the initial commit. As you make further commits on the master branch, the master branch pointer move forward accordingly. Git also uses a special pointer called HEAD to keep track of the branch that you are currently working on. The HEAD always refers to the latest commit on the current branch. Whenever you switch branch, the HEAD also switches to the latest commit on the branch switched.

##### Example

For example, let's create a Git-managed project called git\_branch\_test with only the a single-line README.md file:

This is the README. My email is xxx@somewhere.com

$ **git init**

$ **git add README.md**

$ **git commit -m "Commit 1"**

// Append a line in README.md: This line is added after Commit 1

$ **git status**

$ **git add README.md**

$ **git commit -m "Commit 2"**

// Append a line in README.md: This line is added after Commit 2

$ **git status**

$ **git add README.md**

$ **git commit -m "Commit 3"**

// Show all the commits (oneline each)

$ **git log --oneline**

44fdf4c Commit 3

51f6827 Commit 2

fbed70e Commit 1

##### Creating a new Branch (git branch <branch-name>)

You can create a new branch via "git branch <branch-name>" command. When you create a new branch (says devel, or development), Git creates a new branch pointer for the branch devel, pointing initially at the latest commit on the current branch master.

$ **git branch devel**

Take note that when you create a new branch, the HEAD pointer is still pointing at the current branch.

##### Branch Names Convention

* master branch: the production branch with tags for the various releases.
* development (or next or devel) branch: developmental branch, to be merged into master if and when completes.
* topics branch: a short-live branch for a specific topics, such as introducing a feature (for the devel branch) or fixing a bug (for the master branch).

##### Switching to a Branch (git checkout <branch-name>)

Git uses a special pointer called HEAD to keep track of the branch that you are working on. The "git branch <branch-name>" command simply create a branch, but does not switch to the new branch. To switch to a branch, use "git checkout <branch-name>" command. The HEAD pointer will be pointing at the switched branch (e.g., devel).

$ **git checkout devel**

Switched to branch 'devel'

Alternatively, you can use "git checkout -b <branch-name>" to create a new branch and switch into the new branch.

If you switch to a branch and make changes and commit. The HEAD pointer moves forward in that branch.

// Append a line in README.md: This line is added on devel branch after Commit 3

$ **git status** // NOTE "On branch devel"

$ **git add README.md**

$ **git commit -m "Commit 4"**

[devel c9b88d9] Commit 4

You can switch back to the master branch via "git checkout master". The HEAD pointer moves back to the last commit of the master branch, and the working directory is rewinded back to the latest commit on the master branch.

$ **git checkout master**

Switched to branch 'master'

// Check the content of the README.md, which is reminded back to Commit 3

If you continue to work on the master branch and commit, the HEAD pointer moves forward on the master branch. The two branches now diverge.

// Append a line in README.md: This line is added on master branch after Commit 4

$ **git status** // NOTE "On branch master"

$ **git add README.md**

$ **git commit -m "Commit 5"**

[master 6464eb8] Commit 5

If you check out the devel branch, the file contents will be rewinded back to Commit-4.

$ **git checkout devel**

// Check file contents

#### 5.3  Merging Two Branches (git merge <branch-name>)

To merge two branches, says master and devel, check out the first branch, e,g, master, (via "git checkout <branch-name>") and merge with another branch, e.g., devel, via command "git merge <branch-name>".

##### Fast-Forward Linear Merge

If the branch to be merged is a direct descendant, Git performs fast forward by moving the HEAD pointer forward. For example, suppose that you are currently working on the devel branch at commit-4, and the master branch's latest commit is at commit-3:

$ **git checkout master**

// Let discard the Commit-5 totally and rewind to commit-3 on master branch

// This is solely for illustration!!! Do this with great care!!!

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

HEAD is now at 7e7cb40 Commit 3

// HEAD~1 moves the HEAD pointer back by one commit (-1)

// --hard also resets the working tree

// Check the file contents

$ **git merge devel**

Updating 7e7cb40..4848c7b

Fast-forward

README.md | 1 +

1 file changed, 1 insertion(+)

// Check the file contents

Take note that no new commit is created.

##### 3-Way Merge

If the two branches are diverged, git automatically searches for the common ancestor commit and performs a 3-way merge. If there is no conflict, a new commit will be created.

If git detects a conflict, it will pause the merge and issue a merge conflict and ask you to resolve the conflict manually. The file is marked as unmerged. You can issue "git status" to check the unmerged files, study the details of the conflict, and decide which way to resolve the conflict. Once the conflict is resolve, stage the file (via "git add <file>"). Finally, run a "git commit" to finalize the 3-way merge (the same Edit/Stage/Commit cycle).

$ **git checkout master**

// undo the Commit-4, back to Commit-3

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

HEAD is now at 7e7cb40 Commit 3

// Change the email to abc@abc.com

$ **git add README.md**

$ **git commit -m "Commit 5"**

$ **git checkout devel**

// undo the Commit-4, back to Commit-3

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

// Change the email to xyz@xyz.com to trigger conflict

$ **git add README.md**

$ **git commit -m "Commit 4"**

// Let's do a 3-way merge with conflict

$ **git checkout master**

$ **git merge devel**

Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Automatic merge failed; fix conflicts and then commit the result.

$ **git status**

# On branch master

# You have unmerged paths.

# (fix conflicts and run "git commit")

#

# Unmerged paths:

# (use "git add <file>..." to mark resolution)

# both modified: README.md

no changes added to commit (use "git add" and/or "git commit -a")

The conflict file is marked as follows (in "git status"):

<<<<<<< HEAD

This is the README. My email is abc@abc.com

=======

This is the README. My email is xyz@xyz.com

>>>>>>> devel

This line is added after Commit 1

This line is added after Commit 2

You need to manually decide which way to take, or you could discard both by setting the email to zzz@nowhere.com.

$ **git add README.md**

$ **git commit -m "Commit 6"**

Take note that In a 3-way merge, a new commit will be created in the process (unlike fast-forward merge).

##### Deleting a Merged Branch (git branch -d <branch-name>)

The merged branch (e.g., devel) is no longer needed. You can delete it via "git branch -d <branch-name>".

$ **git branch -d devel**

Deleted branch devel (was a20f002).

// Create the development branch again at the latest commit

$ **git branch devel**

#### 5.4  Rebasing Branch (git rebase)

The primary purpose for rebasing is to maintain a linear project history. For example, if you checkout a devel branch and work on commit-5 and commit-6, instead of doing a 3-way merge into the master branch and subsequently remove the devel branch, you can rebase the commit-5 and commit-6, on commit-4, and perform a linear forward merge to maintain all the project history. New commits (7 and 8) will be created for the rebased commit (5 and 6).

The syntax is:

// SYNTAX

$ **git rebase <base-name>**

// <base-name> could be any kind of commit reference

// (such as an commit-name, a branch name, a tag,

// or a relative reference to HEAD).

Examples:

// Start a new feature branch from the current master

$ **git checkout -b feature master**

// Edit/Stage/Commit changes to feature branch

// Need to work on a fix on the master

$ **git checkout -b hotfix master**

// Edit/Stage/Commit changes to hotfix branch

// Merge hotfix into master

$ **git checkout master**

$ **git merge hotfix**

// Delete hotfix branch

$ **git branch -d hotfix**

// Rebase feature branch on master branch

// to maintain a linear history

$ **git checkout feature**

$ **git rebase master**

// Now, linear merge

$ **git checkout master**

$ **git merge feature**

#### 5.5  Amend the Last Commit (git commit --amend)

If you make a commit but want to change the commit message or adding more changes, you may amend the recent commit (instead of creating new commit) via command "git commit --amend"):

$ **git commit --amend -m "message"**

For example,

// Do a commit

$ **git commit -m "added login menu"**

// Realize that you have not staged some files.

// Amend the commit

$ **git add morefile**

$ **git commit --amend**

// You could modify the commit message here

#### 5.6  More on "git checkout" and Detached HEAD

"git checkout" can be used to checkout a branch, a commit, or files. The syntaxes are:

$ **git checkout <branch-name>**

$ **git checkout <commit-name>**

$ **git checkout <commit-name> <filename>**

When you checkout a commit, Git switches into so-called "Detached HEAD" state, i.e., the HEAD detached from the tip of a branch. Suppose that you continue to work on the detached HEAD on commit-5, and wish to merge the commit-5 back to master. You checkout the master branch, but there is no branch name for your to reference the commit-5!!!

In Summary, you can use "git checkout <commit-name>" to inspect a commit. BUT you should always work on a branch, NOT on a detached HEAD.

#### 5.7  More on "git reset" and "git reset --hard"

[TODO] examples and diagram

$ **git reset <file>**

// Unstage the changes of <file> from staging area,

// not affecting the working tree.

$ **git reset**

// Reset the staging area

// Remove all changes (of all files) from staging area,

// not affecting the working tree.

$ **git reset --hard**

// Reset the staging area and working tree to match the

// recent commit (i.e., discard all changes since the

// last commit).

$ **git reset <commit-name>**

// Move the HEAD of current branch to the given commit,

// not affecting the working tree.

$ **git reset --hard <commit-name>**

// Reset both staging area and working tree to the given

// commit, i.e., discard all changes after that commit.

[TODO] Diagram

[TODO] --soft option

#### 5.8  git revert <commit-name>

The "git revert" undoes a commit. But, instead of removing the commit from the project history, it undos the changes introduced by the commit and appends a new commit with the resulting content. This prevents Git from losing history. "git revert" is a safer way comparing with "git reset".

// SYNTAX

$ **git revert <commit-name>**

// EXAMPLE

[TODO] example and diagram

#### 5.9  Summary of Work Flows

##### Setting up GIT and "Edit/Stage/Commit/Push" Cycle

**Step 1**: Install GIT.

* For Windows and Mac, download the installer from <http://git-scm.com/downloads> and run the downloaded installer.
* For Ubuntu, issue command "sudo apt-get install git".

For Windows, use "git-bash" command shell provided by Windows installer to issue command. For Mac/Ubuntu, use "Terminal".

**Step 2**: Configuring GIT:

// Setup your username and email to be used in labeling commits

$ **git config --global user.email "your-email@yourmail.com"**

$ **git config --global user.name "your-name"**

**Step 3**: Set up GIT repo for a project. For example, we have a project called "olas1.1" located at "/usr/local/olas/olas1.1".

$ **cd /usr/local/olas/olas1.1**

// Initialize the GIT repo

$ **git init**

$ **ls -al**

// Check for ".git" directory

Create a "README.md" (or "README.textile" if you are using Eclipse's WikiText in "textile" markup) under your project directory to describe the project.

**Step 4**: Start "Edit/Stage/Commit/Push" cycles.

Create/Modify files. Stage files into the staging area via "git add <file>".

// Check the status

$ **git status**

......

// Add files into repo

$ **git add README.md**

$ **git add www**

......

// Check the status

$ **git status**

......

**Step 5**: Create a ".gitignore" (in the project base directory) to exclude folders/files from being tracked by GIT. Check your "git status" output to decide which folders/files to be ignored.

For example,

# ignore files and directories beginning with dot

.\*

# ignore directories beginning with dot (a directory ends with a slash)

.\*/

# ignore these files and directories

www/test/

www/.\*

www/.\*/

The trailing slash indicate directory (and its sub-directories and files).

If you want the ".gitignore" to be tracked (which is in the ignore list):

$ **git add -f .gitignore**

// -f to override the .gitignore

**Step 6**: Commit.

$ **git status**

......

// Commit with a message

$ **git commit -m "Initial Commit"**

......

$ **git status**

......

**Step 7**: Push to the Remote Repo (for backup, version control, and collaboration).

You need to first create a repo (says olas) in a remote GIT host, such as GitHub or BitBucket. Take note of the remote repo URL, e.g., https://username@hostname.org/username/olas.git.

$ **cd /path-to/local-repo**

// Add a remote repo name called "origin" mapped to the remote URL

$ **git remote add origin https://hostname/username/olas.git**

// Push the "master" branch to the remote "origin"

// "master" is the default branch name of your local repo after init.

$ **git push origin master**

Check the remote repo for the files committed.

**Step 8**: Work on the source files, make changes, commit and push to remote repo.

// Check the files modified

$ **git status**

......

// Stage for commit the modified files

$ **git add ....**

......

// Commit (with a message)

$ **git commit -m "commit-message"**

// Push to remote repo

$ **git push origin master**

**Step 9**: Create a "tag" (for version number).

// Tag a version number to the current commit

$ **git tag -a v1.1 -m "Version 1.1"**

// -a to create an annotated tag, -m to provide a message

// Display all tags

$ **git tag**

......

// Push the tags to remote repo

// ("git push -u origin master" does not push the tags)

$ **git push origin --tags**

##### Branch and Merge Workflow

It is a good practice to freeze the "master" branch for production; and work on a development branch (says "devel") instead. You may often spawn a branch to fix a bug in the production.

// Create a branch called "devel" and checkout.

// The "devel" is initially synchronized with the "master" branch.

$ **git checkout -b devel**

// same as:

// $ **git branch devel**

// $ **git checkout**

// Edit/Stage/Commit

$ **git add <file>**

$ **git commit -m "commit-message"**

// To merge the "devel" into the production "master" branch

$ **git checkout master**

$ **git merge devel**

// Push both branches to remote repo

$ **git push origin master devel**

// Checkout the "devel" branch and continue...

$ **git checkout devel**

// Edit/Stage/Commit/Push

// Need to fix a bug in production (in "master" branch)

$ **git checkout master**

// Spawn a "fix" branch to fix the bug, and merge with the "master" branch

// To remove the "devel" branch (if the branch is out-of-sync)

$ **git branch -d devel**

// To re-create the "devel" branch

$ **git checkout -b devel**

#### 5.10  Viewing the Commit Graph (gitk)

You can use the "git-gui" "gitk" tool to view the commit graph.

To run the git-gui, you can right-click on the project folder and choose "Git Gui"; or launch the Git-bash shell and run "git gui" command.

To view the commit graph, choose "Repository" ⇒ "Visualize master's history", which launches the "gitk". You can view the details of each commit.

### 6.  Collaboration

Reference: <https://www.atlassian.com/git/tutorials/making-a-pull-request/how-it-works>.

#### 6.1  Synchronizing Remote and Local: Fetch/Merge, Pull and Push

##### Setup up a remote repo (revision)

As described earlier, you can use "git remote" command to set up a "remote name", mapped to the URL of a remote repo.

// Add a new "remote name" maps to the URL of a remote repo

$ **git remote add <remote-name> <remote-url>**

// For example,

$ **git remote add origin https://hostname/username/project-name.git**

// Define a new remote name "origin" mapping to the given URL

// List all the remote names

$ **git remote -v**

// Delete a remote name

$ **git remote rm <remote-name>**

// Rename a remote name

$ **git remote rename <old-remote-name> <new-remote-name>**

##### Cloning a Remote Repo (revision)

$ **git clone <remote-url>**

// Init a GIT local repo and copy all objects from the remote repo

$ **git clone <remote-url> <working-directory-name>**

// Use the working-directory-name instead of default to project name

Whenever you clone a remote repo using command "git clone <remote-url>", a remote name called "origin" is automatically added and mapped to <remote-url>.

[TODO] Diagram

##### Fetch/Merge Changes from remote (git fetch/merge)

The "git fetch" command imports commits from a remote repo to your local repo, without updating your local working tree. This gives you a chance to review changes before updating (merging into) your working tree. The fetched objects are stored in remote branches, that are differentiated from the local branches.

$ **cd /path-to/working-directory**

$ **git fetch <remote-name>**

// Fetch ALL branches from the remote repo to your local repo

$ **git fetch <remote-name> <branch-name>**

// Fetch the specific branch from the remote repo to your local repo

// List the local branches

$ **git branch**

\* master

devel

// \* indicates current branch

// List the remote branches

$ **git branch -r**

origin/master

origin/devel

// You can checkout a remote branch to inspect the files/commits.

// But this put you into "Detached HEAD" state, which prevent you

// from updating the remote branch.

// You can merge the fetched changes into local repo

$ **git checkout master**

// Switch to "master" branch of local repo

$ **git merge origin/master**

// Merge the fetched changes from stored remote branch to local

[TODO] Diagram

##### git pull

As a short hand, "git pull" combines "git fetch" and "git merge" into one command, for convenience.

$ **git pull <remote-name>**

// Fetch the remote's copy of the current branch and merge it

// into the local repo immediately, i.e., update the working tree

// Same as

$ **git fetch <remote-name> <current-branch-name>**

$ **git merge <remote-name> <current-branch-name>**

$ **git pull --rebase <remote-name>**

// linearize local changes after the remote branch.

The "git pull" is an easy way to synchronize your local repo with origin's (or upstream) changes (for a specific branch).

[TODO] Diagram

##### Pushing to Remote Repo (revision)

The "git push <remote-name> <branch-name>" is the counterpart of "git fetch", which exports commits from local repo to remote repo.

$ **git push <remote-name> <branch-name>**

// Push the specific branch of the local repo

$ **git push <remote-name> --all**

// Push all branches of the local repo

$ **git push <remote-name> --tag**

// Push all tags

// "git push" does not push tags

$ **git push -u <remote-name> <branch-name>**

// Save the remote-name and branch-name as the

// reference (or current) remote-name and branch-name.

// Subsequent "git push" without argument will use these references.

[TODO] Diagram

#### 6.2  "Fork" and "Pull Request"

"Fork" and "Pull Request" are features provided by GIT hosts (such as GitHub and BitBucket):

* Pushing "Fork" button to copy a project from an account (e.g., project maintainer) to your own personal account. [TODO] diagram
* Pushing "Pull Request" button to notify other developers (e.g., project maintainer or the entire project team) to review your changes. If accepted, the project maintainer can pull and apply the changes. A pull request shall provide the source's repo name, source's branch name, destination's repo name and destination's branch name.

#### 6.3  Feature-Branch Workflow for Shared Repo

Feature-Branch workflow is more prevalent with small teams on private projects. Everyone in the team is granted push access to a single shared remote repository and feature (or topic) branches are used to isolate changes made by the team members.

The project maintainer starts the "master" branch on the shared remote repo. All developers clone the "master" branch into their local repos. Each developer starts a feature branch (e.g., "user1-featureX") to work on a feature. Once completed (or even work-in-progress), he files a "pull request" to initiate a review for his feature. All developers can provide comments and suggestions. Once accepted, the project maintainer can then merge the feature branch into the "master" branch.

The steps are:

1. Mark, the project maintainer, starts the project by pushing to the shared remote repo's "master" branch.
2. Carol, a contributor, clones the project into her local repo, via:
3. // Carol:
4. $ **cd parent-directory-of-the-working-directory**
5. $ **git clone https://hostname/path-to/project-name.git**
6. // Create a remote-name "origin" (default), branch "master"

// on her local repo

1. Carol starts a feature branch (says "carol-feature") under the "master" branch to work on a new feature, via:
2. // Carol:
3. $ **git checkout -b carol-feature master**
4. // Create a new branch "carol-feature" under "master" branch
5. // and switch to the new branch
6. // Edit/Stage/Commit/Push cycles on carol-feature branch
7. $ **git status**
8. $ **git add <file>**
9. $ **git commit -m <message>**
10. $ **git push origin carol-feature**

// Repeat until done

1. Carol completes the new feature. She files a "pull request" (by pushing the "pull request" button on the Git host) to notify the rest of the team members.
2. Mark, the project maintainer, or anyone in the team, can comment on Carol's feature. Carol can re-work on the feature, if necessary, and pushes all subsequent commits under her feature branch.
3. Once the feature is accepted, Mark, or anyone in the team (including Carol), performs a merge to apply the feature branch into the "master" branch:
4. // Mark, or Anyone:
5. $ **git checkout master**
6. // Switch to the "master" branch of the local repo
7. $ **git pull origin master**
8. // Fetch and merge the latest changes on local's "master" branch,
9. // if any (i.e., synchronize)
10. $ **git pull origin carol-feature**
11. // Fetch and merge carol-feature branch on local's "master" branch
12. $ **git push origin master**

// Update the shared remote repo

1. Everyone can update their local repo, via:
2. // Everyone:
3. $ **git checkout master**
4. // Switch to the "master" branch of the local repo
5. $ **git pull origin master**

// Fetch and merge the latest changes on local "master" branch

[TODO] Diagram

#### 6.4  Forking Workflow

In Forking workflow, Instead of using a common shared remote repo, each developer forks the project to his own personal account on the remote host. He then works on his feature (preferably in a feature branch). Once completed, he files a "pull request" to notify the maintainer to review his changes, and if accepted, merge the changes.

Forking workflow is applicable to developers working in small teams and to a third-party developer contributing to an open source project.

The steps are:

1. Mark, the project maintainer, pushes the project from his local repo ("master" branch) to a remote Git host. He permits "read" access by contributors.
2. Carol, a contributor, gotos Mark's repo, forks the project (by pushing the fork button). "Forking" copies the project to Carol's own personal account on the same Git host.
3. Carol then clones the project from her forked repo into her local repo, via:
4. // Carol:
5. $ **cd parent-directory-of-the-working-directory**
6. $ **git clone https://hostname/carol/project-name.git**
7. // Create a remote name "origin" automatically

// Copy the "master" branch

1. When a fork is cloned, Git creates a remote-name called origin that points to the fork, not the original repo it was forked from. To keep track of the original repo, Carol creates a remote name called "upstream" and pulls (fetches and merges) all new changes:
2. // Carol:
3. $ **cd carol-local-repo-of-the-fork**
4. $ **git remote add upstream https://hostname/mark/project-name.git**
5. // Create a remote-name "upstream" pointing to the original remote repo
7. $ **git remote -v**
8. // List all the remote names and URLs
9. // origin: mapped to Carol's forked remote repo
10. // upstream: mapped to Mark's original remote repo
12. $ **git pull upstream master**
13. // Fetch and merge all changes from the original remote repo to local repo

// for the "master" branch

1. Now, Carol can make changes on her local repo (on a new branch), stage and commit the changes, and pushes them to her forked remote repo (so called Edit/Stage/Commit/Push cycles):
2. // Carol:
3. $ **git checkout -b carol-feature master**
4. // Create a new branch called "carol-feature" under "master"
5. // and switch to the new branch
7. // Edit/Stage/Commit/Push cycles on "carol-feature" branch
8. $ **git status**
9. $ **git add <file>**
10. $ **git commit -m <message>**
11. $ **git push origin carol-feature**

// Repeat until done

1. Carol files a pull request to Mark (the project maintainer) by pushing the pull-request button. She needs to specify her forked remote repo-name, her branch name (carol-feature), Mark's remote repo-name, and Mark's branch name (master).
2. Mark opens the pull request (in pull request tab), reviews the change, and decides whether to accept the changes. Mark can ask Carol to re-work on the feature. Carol repeats the Edit/Stage/Commit/Push cycles.  
   If Mark decides to accept the changes, he pushes the "Merge" button to merge Carol's contribution to his master branch on the remote repo.  
   [If there is no "Merge" button] Mark needs to do the following:
3. // Mark:
4. $ **git checkout master**
5. // Checkout the "master" branch of the local repo
7. $ **git remote add carol git://hostname/carol/project-name.git**
8. // Add a new remote pointing to the Carol's forked remote repo
10. $ **git pull carol carol-feature**
11. // Fetch and merge the changes into local repo's master branch
13. $ **git push origin master**

// Push the update to the Mark's original remote repo

1. All contributors (including Mark and Carol) shall regularly synchronize their local repo by fetch/merge with Mark's master branch.
2. // Carol (and everyone):
3. $ **git checkout master**
4. // Switch to the "master" branch of the local repo
5. $ **git pull upstream master**
6. // Fetch and merge the latest changes on "master" branch from

// original remote repo to his local repo

#### 6.5  Other Workflows

There are other workflows such as "Centralized Workflow" and "GitFlow Workflow". Read "<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>".

### 7.  Miscellaneous and How-To

##### Stage and Commit (git commit -a -m <message>)

You can skip the staging (i.e., the "git add <file>...") and commit all changes in the working tree via "git commit -a -m <message>" with -a (or --all) option.

##### Stage all changes (git add -A)

You can use "git add -A" to stages all changes in the working tree to the staging area.

##### Unstage a Staged file (git rm --cached <file> / git reset head <file>)

Recall that you can use "git add <file>" to stage new files or modified files into the staging area.

To unstage a staged new file, use "git rm --cached <file>".

To unstage a staged modified file, use "git reset head <file>".

##### Unmodified a modified file (git checkout -- <file>)

After a commit, you may have modified some files. You can discard the changes by checking out the last commit via "git checkout -- <file>".

##### How to Amend the Last Commit (git commit --amend)

If you make a commit but want to change the commit message:

$ **git commit --amend -m "message"**

If you make a commit but realize that you have not staged some file changes, you can also do it with --amend:

$ **git add morefile**

$ **git commit --amend**

You can also make some changes to working tree, stage, and amend the last commit

// Edit morefile (make changes)

$ **git add morefile**

$ **git commit --amend**

##### How to Undo the Previous Commit(s) (git reset)

To undo previous commit(s):

// Reset the HEAD to the previous commit

// --soft to keep the working tree and index

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

$ **git reset --soft HEAD^** // Unix

// Make changes

......

// Stage

$ **git add ......**

// Commit

$ **git commit -c ORIG\_HEAD**

The "git reset --hard HEAD~1" moves the HEAD to the previous commit, restore the working tree and discard the index (i.e., discard all change after the previous commit). Instead of HEAD~n, you can also specify the commit hash code.

The "git rest HEAD~1" with default --mixed moves the HEAD to the previous commit, keep the working tree and discard the index

The "git reset --soft HEAD~1" moves the HEAD to the previous commit, keep the working tree and the index (i.e., keep all changes after the previous commit).

[TODO] Examples, diagrams and "git status" outputs.

For a public repo, you probably need to make another commit and push the commit to the public repo, or ...

##### Relative Commit Names

A commit is uniquely and absolutely named using a 160-bit (40-hex-digit) SHA-1 hash code of its contents. You can always refer to a commit via its hash value or abbreviated hash value (such as the first 7 hex-digit) if there is no ambiguity.

You can also refer to a commit relatively, e.g., master~1 (Windows), master^ (may not work in Windows), master^1 refers to the previous (parent) commit on the master branch; master~2, master^^ refers to the previous of the previous (grandparent) commit, and etc. If a commit has multiple parents (e.g., due to merging of branches), ^1 refers to the first parent, ^2 refers to the second parent, and so on.

**REFERENCES & RESOURCES**

1. GIT mother site @ [http://git-scm.com](http://git-scm.com/) and GIT Documentation @ <http://git-scm.com/doc>.
2. Git User's Manual @ <http://www.kernel.org/pub/software/scm/git/docs/user-manual.html>.
3. Git Hosts: GitHub @ [https://github.com](https://github.com/), Bitbucket @ [https://bitbucket.org](https://bitbucket.org/).
4. Git Tutorials @ <https://www.atlassian.com/git/tutorials>.
5. Bitbucket Documentation Home @ <https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+Documentation+Home>.
6. Bitbucket 101 @ <https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+101>.
7. Jon Loeliger and Matthew McCullough, "Version Control with Git", 2nd Eds, O'reilly, 2012.
8. Scott Chacon, "Pro Git", Apress, 2009.

**Other Centos 7 Setup for Apps**

<https://www.vultr.com/docs/category/centos/?page=1>

<https://www.vultr.com/docs/installing-docker-on-centos-7>

**GIT INTERVIEW**

GIT is one of the most popular version control systems for enterprise application and big data solution. It is also a must-have piece of technology for all android, software or iOS developers. Many companies use the GIT framework in their software development architectures. This technology plays a vital role in a lot of organizations.

Git is an open-source distributed version control system specially designed to manage everything right from small to very large projects with speed and performance. For an individual, it is easy to learn and has a small footprint with fast performance. Git basically outclasses various tools like CVS, ClearCase, Perforce, and Subversion with features like convenient staging areas, cheap local branching.

According to the latest Stack Overflow developer survey, more than 70 percent of developers use Git, making it the most-used Version Control Systems (VCS) in the world. It is commonly used for both open source and commercial software development, with significant benefits for individuals, teams and businesses.

Learning Git will fetch you a huge job opportunity because 99% of the reputed companies are using Git and GitHub. Learning Git will make you more hirable and help you differentiate yourself from others. People with Git knowledge are offered handsome annual packages. The average pay for a Front end developer with Git skill is [$67,078](https://www.payscale.com/research/US/Skill=Git/Salary) per year. Git is a great field for those who look for advancement in their career.

Every candidate faces jitters when it comes to Git interview. If you are planning to build a career as a developer yet facing troubles in cracking the Git interview,  so here are the best samples of advanced Git interview questions. These interview questions on Git will aid you to crack your Git interview and help you in achieving your dream job. Git is a great field for those who seek advancement in their career and will help to build your concepts around Git and ace the GIT interview smoothly.

These were a few top git interview questions about Git that you need to prepare. Though Git is much deeper and vast, these interview questions will help you a lot to get through.

Crack your git interview easily today. All the best!

**BASIC**

[**What is a git commit?**](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-653)

A commit is the group of -

1. Saved changes to the Git repository
2. This impacts history
3. Uniquely identified by a SHA-1 hash

#### [What is a git branch?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-654)

Branches can be thought of as a timeline with commits. By default, the master branch is the main, primary branch that we usually work with. HEAD is a pointer to the last commit on the current branch. Remote is simply a refine, or a pointer, to a related repository somewhere that's not local and that could be within local network or enterprise network or somewhere out on the internet. Some examples of places that might be hosting remote repositories would include GitHub, Gitlab, Bitbucket etc.

#### [How to check if git is installed and how to get help on various git commands?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-655)

On the terminal/cmd using 'git version' command; it will show the installed git's version. Using command 'git help'; we can display the list of commands supported by the git. Using 'git help <command-Name>' we can get the detailed information about specific git command; on windows it will be displayed in the web-browser but on Mac/Linux it will be displayed on the terminal itself. Command 'git help -a' gives a list of subcommands.

**[How to get the current status of local repository?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-656)**

Using 'git status' command in the repository working directory, when this command is used it will display-

* The current branch name
* If it is an initial commit or not
* List of the unstage/untracked files(which are newly added or modified but not added to staging/index) List of the files pending to commit(these files are previously added to stating/index)

If there is no file/folder to add or commit then it will just display name of the branch and message "nothing to commit, working directory clean"

#### [What are the different ways to add/stage files/changes to the git repository?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-657)

There are two ways to add or stage the files or the changes to the git repository.

Using 'git add' command, this will add the file to staging

Using 'git commit -am <your custom message>', the '-a' option with commit command will add the files to staging(not to 'newly added files' but only that have been modified and deleted) and commit too. basically 2 in 1, it adds then commit the files. '-m' options for providing the custom commit message.

[**What are the steps to unstage files and discard the changes in working directory?**](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-658)

 If the git status command displays 'myfile.txt' as file pending to commit then

1. To unstage a file(which is not yet committed but in staging stage); command 'git reset <file-name>' is used. After using 'git reset HEAD myfile.txt' command; 'git status' command will not show myfile.txt in the list of files pending to commit, now it will be displayed under the list of the unstage/untracked files.
2. Now to discard the changes('myfile.txt') in working directory; command 'git checkout -- <file-name> is used. After using 'git checkout -- myfile.txt' command; 'git status' command will not show myfile.txt in the list of unstage/untracked files as this file 'myfile.txt' will be replaced with the last committed version from the repository.

#### [What is git stash?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-659)

When you are working in your working directory and done few changes so few items in staging, and now if you have to provide any hot fix based on the last commit on the remote repository then to do it the best way is save the current state of the working directory and the index/staging (something like on a stack), for that we use command 'git stash' or 'git stash save'. After using this command, the current working directory will be having the same match as on the remote last commit means locally no changes exist to stage or commit. Now you can do your stuff on the clean copy of the remote; git stash will not cover git ignored files and the newly created files but not 'staged' yet in current working directory.

[**Intermediate**](https://www.knowledgehut.com/interview-questions/.intermediate)

**[What is the difference between "git pull" and "git fetch"?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2887)**

Pull

$ git pull origin master

git pull says "**bring the changes in the remote repository to where I keep my own code.**"

Normally git pull does this by doing a git fetch to bring the local copy of the remote repository up to date, and then merging the changes into your own code repository and possibly your working copy. If you don’t closely manage your branches, you may run into frequent conflicts.

Fetch

$ git fetch origin

git fetch is the command that says "**bring my local copy of the remote repository up to date.**"

When you command fetch, Git extract all commits from the target branch that does not exist in the current branch and keeps them in your local repo. But, it does not merge them with your current branch. This is mostly useful if you need to keep your repo up to date, but are working on some solution that might break if you update your files. To integrate the commits into your master branch, we use merge.

#### Git data transport commands

**[How to revert the previous commit in git?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2888)**

git reset --hard HEAD~N- To undo last commit and changes.

N stands for how far you would like to undo your changes. For undo last changes N=1

Let's talk about below use case where C is current HEAD while F is a state of files.

  (F)

A-B-C

   ↑

 master

If you like to undo commit C. You do this:

git reset --hard HEAD~1

The result is:

(F)

A-B

 ↑

master

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

Suppose we would like to undothe commit but keep our changes before we do good commit. Let's start again with the above example, with C as our HEAD:

  (F)

A-B-C

   ↑

 master

We can easily achieve this by, leaving off the --hard:

git reset HEAD~1

In this case, the result will be below :

  (F)

A-B-C

 ↑

master

In both scenarios, the HEAD was just a pointer to the last commit. When we run a git reset HEAD~1, we ask Git to move the HEAD pointer one commit back. But (unless we use --hard) we leave our files as they were. So now git status will show the changes you had checked into C. Your files will remain as it is but only GIT HEAD has moved one commit back.

**[What is the difference between HEAD, working tree and index, in Git?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2889)**

A single git repository can track any number of branches, but your working tree is always referring just one of them (the "current" or "checked out" branch), and HEAD points to that branch.

HEAD is the commit at the top of the current branch. If you've just checked out the branch, i.e. have no modified files, then its content matches the working tree. As soon as you modify anything, it no longer matches.

A working tree is the dir and sub-directory within it that contain the source files. It can be anywhere, but normally it is the same dir in which the hidden .git dir is located.

The index/staging area is a single, large, binary file in <baseOfRepo>/.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.

**[How to remove a file from git without removing it from your file system?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2890)**

Sometimes we may end up adding files in our commit which we never intended to commit. Nothing to worry if you come across a scenario. The command 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.

We can use git rm --cached on the file if you want to remove from the version control cache but do not want to remove/delete from your filesystem. So if you wanted to remove foo.txt from version control like this just run this command:

git rm --cached foo.txt

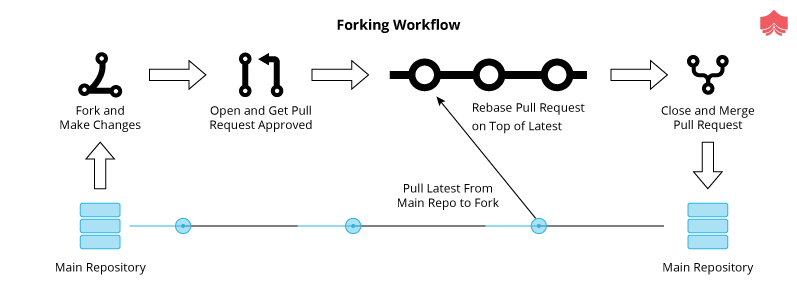
We can also use below git commands to achieve the same result:

git reset filename

The above command is used to undo the local changes. Git reset operates on "**The Three Trees of Git**". These trees are the Commit History ( HEAD ), the Staging Index, and the Working Directory

**[Explain the advantages of Forking Workflow.](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2891)**

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 forking workflow can be summarized as below:

* Fork a GitHub repository.
* Clone the forked repository to your local system.
* Add a Git remote for the original repository.
* Create a feature branch in which to place your changes.
* Make your changes to the new branch.
* Commit the changes to the branch.
* Push the branch to GitHub.
* Open a pull request from the new branch to the original repo.
* Clean up after your pull request is merged.

**[What is "git cherry-pick"?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2892)**

Cherry picking in Git shifts a commit from one branch and apply it onto another one. This is indifferent to other ways like merge and re-bases which moves many commits onto another branch. Make sure we are on the same branch we would like to apply the commit to.

1. We will first move to master branch  
   git checkout master
2. Once we are on the same branch we run the following command:  
   git cherry-pick <commit-hash>

**Note**:commit-hash: Unique identifier for  commit.

If we cherry-pick from a public repo branch, we should consider below command :

git cherry-pick -x <commit-hash>

This will always generate a standardized commit message. This way, our team members can still keep track of the origin of the commit and may avoid merge conflicts in the future. If we have notes attached to the commit that does not follow the cherry-pick. To bring them over as well, we can use:

git notes copy <from> <to>

A common use case for cherry-picking is to forward- or back-port commits from a maintenance branch to a development branch.

**[What is Git fork? What is the difference between fork, branch and clone?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2893)**

Clone: By cloning a repository, you download the whole repository with dotfiles (like .git, .gitignore etc.) to your local computer. A clone is a localized copy of some remote repository. When we clone, we are actually bringing the entire source repository, including all the history and branches locally.

Fork: By forking a repository, you just virtually create a “copy” of the main repository to your account. Forks are usually useful in creating Pull Request(s) to the main repository. A fork is a remote, server-side copy of a repository, distinct from the original.

A branch is a way to handle the changes within a single repo to eventually integrate them with the rest of the code. A branch exists within a repository only. Conceptually, it represents a thread of development which is needed to manage development/maintenance.

The owner of the main repository gets a notification on their newsfeed when you fork their repository, however, it’s impossible to track if you clone / download their repository. However, note that you won’t be able to submit Pull Request(s) with a download as there are no dotfiles

**[What language is used in GIT?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2894)**

GIT is fast, and ‘C’ language makes this possible by reducing the overhead of runtimes associated with higher languages.

Git’s Source code is hosted on Github here: [git/git](https://github.com/git/git)

So, From that these are the languages used in that repository:

* C - 45%
* Shell - 35%
* Perl - 8%
* Tcl - 5%
* Python - 2%
* C ++ - 2%

Git's design was inspired by BitKeeper and Monotone. Git was originally designed as a low-level version-control system engine, on top of which others could write front ends, such as Cogito or StGIT. The core Git project has since become a complete version-control system that is usable directly. While strongly influenced by BitKeeper, Torvalds deliberately avoided conventional approaches, leading to a unique design

The below are characteristics of GIT which makes it very popular in SCM category:

1. Strong support for non-linear development.
2. Compatibility with existing systems and protocols
3. Efficient handling of large projects
4. Cryptographic authentication of history
5. Toolkit-based design.

**[What is the function of ‘GIT PUSH’ in GIT?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2895)**

‘GIT PUSH’ updates remote refs along with associated objects. The "**push**" command is widely used to publish any new local commits on a remote server. The source (i.e. which branch the data should be uploaded from) is always the currently checked out HEAD branch. The target (i.e. which branch the data should be uploaded to) can be specified in the command's option. Sometimes we use below command :

**git push origin master**

**push** = push your changes to the remote server

**origin** = remote Server origin

**master** = Master branch

If you have other remote branches you have something like "git push origin test" then you push your changes to the test remote branch.

Just specifying

git push origin

will push every local branch that has a matching remote branch to that branch per default. Not just the current branch. This is the same as using git push origin:



**[What is the difference between git checkout and git clone?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2896)**

git clone is to fetch your repositories from the remote git server.

we clone a remote  repository with below command

**git clone [url]**

For example, Let’s say  if we want to clone the Open Framework Git library called open\_framework, we can easily do so like this:

**$ git clone git://github.com/SU-SWS/open\_framework.git**

That will create a directory named open\_framework (at our local file system location), and initializes a .git directory inside it, extract all the data for that repository, and checks out a working copy of the latest version. If we go into the newly created open\_framework directory, we can see the project files in there, ready to be worked on or used.

Cloning a Repository Into a Specific Local Folder

If we want to clone the repo into a particular directory named something other than open\_framework, we can specify that as the next command-line option:

**$ git clone git:github.com/SU-SWS/open\_framework.git mynewtheme**

The above command does the same thing as the previous one, but the clone gets created in the target directory called mynewtheme.

git checkout is to check out the desired status of your repository (like branches or particular files).

E.g., you are currently on the master branch and you want to switch into the develop branch.

git checkout develop\_branch

**[What is the difference between git and svn?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2898)**

GIT is a distributed version control system where each developer gets its own local copy of the remote repository. While SVN comes under centralized version control. Centralized implies a client-server relationship. There is one main repository (repo) on a server, and every working copy clones and commits to this server. Decentralized is the opposite: there is no “**central**” repo.

You can not commit in SVN if you are not connected to the central repo so network connectivity is mandatory for SVN for any commit. With Git, you do not have this issue. Your local copy is a repository, and you can commit to it and get all the benefits of source control. When you regain connectivity to the main repository, you can commit against it.

| **Git** | **SVN** |
| --- | --- |
| 1. Decentralized Version Control tool | 1. Centralized Version Control tool |
| 2. It comes under the third generation of version control tools | 2. It comes under the 2nd generation of Version Control tools. |
| 3. We can clone the entire repo on our local systems | 3. Version history is stored on a server-side repository |
| 4. The commits are possible even if remote repo not available | 4. Only online commits are allowed |
| 5. The basic /pull operations are faster | 5. The Push/pull operations are slower as compared to GIT. |
| 6. Works are shared automatically by commit | 6. Nothing is shared automatically, you need manual intervention |

**[What is ‘head’ in git and how many heads can be created in a repository?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2899)**

In every repository, there is a default head referred to as “Master”. When you switch branches with git checkout, the HEAD revision changes to point to the tip of the new branch. A repository can contain any number of heads. While HEAD may regularly be portrayed as indicating the latest submit or indicating what branch you're on, neither one of the descriptions is very right.

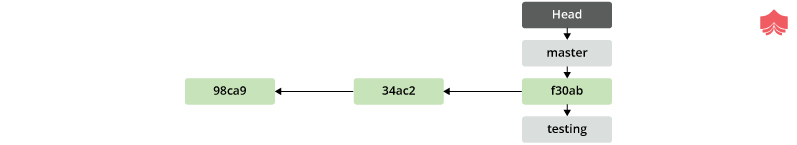
It focuses on what you've at present looked at, pretty much, which may not be what's the most recent. HEAD is a common content record in .git/HEAD, so you can investigate. So at this moment, my HEAD contains:

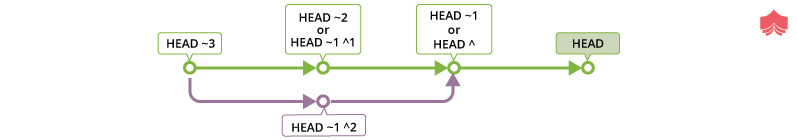
ref: refs/heads/test

This implies I've looked at the 'test 'branch, which is indicated by the ref 'refs/heads/test'. That this is (usually\*) put away in the record .git/refs/heads/test, and contains:

7fc4bd269202ead49b9e5d4fa5f8e7db1525ea5b

which is the commit ID for that branch.





~(Tilde) and ^(caret), pointing to the relative position of a commit. The most common is to add ~(Tilde) after the HEAD to point to the generation of the parent; ^(caret) can point to the previous generation of generations.

**[What Git repository hosting services is widely used?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2900)**

A source-code storehouse is a chronicling and web facilitating office where source code, for programming or for site pages, is kept, either openly or secretly. They are frequently utilized by open-source programming ventures and other multi-engineer activities to deal with different forms.

Kindly find beneath probably the most well known facilitating administrations accessible :

**Alternative 1: GitHub**

It's as yet the biggest network site for programming advancement, despite everything it has probably the best apparatuses for issue following, code audit, persistent reconciliation, and general code the board. the universally adored open source appropriated rendition control framework.

**Choice 2: GitLab**

It's completely open source. You can have your code directly on GitLab's site much like you would on GitHub, yet you can likewise decide to self-have your very own GitLab case individually server and have full power over who approaches everything there and how things are overseen. GitLab practically has highlighted equality with GitHub, and a few people may even say its ceaseless incorporation and testing devices are prevalent. In spite of the fact that the network of engineers on GitLab is unquestionably littler than the one on GitHub,

**Choice 3: Bitbucket**

Bitbucket has been around for a long time. Bitbucket was procured by a bigger partnership (Atlassian) eight years back It's as yet a business stage like GitHub, however, it's a long way from being a startup, and it's on really stable balance, hierarchically. Bitbucket shares the vast majority of the highlights accessible on GitHub and GitLab, in addition to a couple of novel highlights of its own, similar to local help for Mercurial storehouses.

**Alternative 4: SourceForge**

The granddaddy of open source code store locales is SourceForge. It used to be that on the off chance that you had an open source venture, SourceForge was the spot to have your code and offer your discharges. It took a short time to move to Git for adaptation control, and it had its very own rash of the business obtaining and re-securing occasions, combined with a couple of tragic packaging choices for a couple of open source ventures. So, SourceForge appears to have recuperated from that point forward, and the site is as yet a spot where many open source extends live.

<https://www.git-tower.com/blog/git-hosting-services-compared/>

**[What are the Advantages Of Using Git?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2901)**

GIT is a distributed version control system where each developer gets its own local copy of the remote repository. Having a full nearby history makes Git quick since it implies you needn't bother with a system transfer speed to make submits, examine past renditions of a record, or perform diffs between submits. Conveyed advancement likewise makes it simpler to scale your group. If somebody breaks the creation branch in SVN, different designers can't check in their progressions until it's fixed. With Git, this sort of circumstance doesn't exist. Everyone can proceed with their advancement in their very own neighbourhood storehouses.

Some of the benefits of GIT are listed below :

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

**[What is git rebase and how can it be used to resolve conflicts in a feature branch before the merge?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2908)**

In simple words, git rebase allows one to move the first commit of a branch to a new starting location. For example, if a feature branch created from the master branch, and in between if the master branch has received additional commits, git rebase can be used to move the feature branch to the tip of master. The command will integrate the changes made in the feature branch at the tip of the master, allowing conflicts to be resolved amicably. When it is done with care, this will always allow the feature branch to be merged into master seamlessly and sometimes as a simple fast-forward operation.

Rebase compresses all the changes into a single “patch.” Then it integrates the patch onto the target branch. Unlike merging, rebasing flattens the history because it transfers the completed work from one branch to another.you should never rebase commits once they've been pushed to a public repository. The rebase would replace the old commits with new ones. We can have a scenario where if we started doing some development and then another developer made an unrelated change. You probably want to pull and then rebase to base your changes from the current version from the repo.

**[When should we use GIT stash?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2909)**

This command saves our local modifications and reverts the working directory to match the HEAD commit of the repository. The changes stashed away by above command can be listed with git stash list, and checked with git stash show, and restored with git stash apply. We can understand this scenario with the below examples. In the below code we have modified one file (index.html) and added new file stsyle.css in the repository.

$ git status

On branch master

Changes to be committed:

new file: style.css

Changes are 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

There are lots of scenarios where we would be needing a clean working copy as recommended or even required: This situation arises when merging branches, when pulling from a remote, or simply when checking out a different branch.

The "git stash" command always help us to (temporarily but safely) and store our uncommitted local changes - and leave us with a clean working copy.

Continuing Where You Left Off

As already mentioned, Git's Stash is meant as temporary storage. When you're ready to continue where you left off, you can restore the saved state easily:

$ git stash pop

**[What is the function of ‘git config’?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2910)**

Git has an in-built command called git config that gives you a chance to get and set setup factors that control all parts of what Git looks like and works in your repository. The 'git config' order is an advantageous method to set design alternatives for your Git installation. The behaviour of a store, client data, inclinations and so forth can be characterized through this direction. The primary thing you ought to do when you introduce Git is to set your client name and email address. This is significant in light of the fact that each Git submit utilizes this data, and it's changelessly prepared into the submits you begin making:

Setting your Git username for every repository on your computer

1. Open Git Bash.
2. Set a Git username:

    $ git config --global user.name "XYZ"

1. Confirm that you have set the Git username correctly:
2. $ git config --global user.name  > XYZ

**Checking Your Settings**

If you want to check your configuration settings, you can use the git config --list command to list all the settings Git can find at that point:

$ git config --list

user.name=XYZ

user.email=abc@example.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

**[What is the function of ‘git diff ’ in git?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2911)**

'git diff ' dependably show the progressions between various submits, submit and working tree etc. Git has an Inuit Index between local repo and our working directory. So the vast majority of Git directions really allude to list or the local repo. When we state HEAD in our Git direction, it really alludes the local repo.

//contrast the working directory and local repo.

**git diff HEAD [filename]**

//contrast the working directory and index.

**git diff [filename]**

//contrast the index and local repo.

**git diff - cached[filename]**

We can likewise look at records from two distinctive submits. Each submit in Git has a hash submit id which we can get when we type git log. At that point, we can utilize the submit id with diff order this way.

**git diff 8ez2..e13 922...a3f86**

We can look at a solitary file, yet the entirety of our progressions without a moment's delay. On the off chance that we have made changes in numerous files, simply don't make reference to any file name in the diff direction which will diff all the changed records.

//contrasts working directory and index,

//for example, demonstrates the progressions that are not staged yet.

**git diff**

//contrasts working catalog and local repo

//demonstrates the rundown of changes after your last submit.

**git diff HEAD**

//contrasts index and local repo

//demonstrates the diff between your last submit and changes to be submitted straightaway.

**git diff - cached**

**[What is ‘git status’ is used for?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2912)**

A 'Git Status' is the most generally utilized command on GIT platform. It records down the distinction between the working catalog and the index, it is useful in understanding a git all the more completely.

To Check the status of the local repo

Utilize the git status command, to check the present condition of the local repo.

$ git status

On branch master

Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

Being "fully informed regarding 'origin/master' is the best message for git status. It really implies there is nothing to push. "working registry clean" implies that every one of the documents in the present index is being overseen by git (or are as a rule deliberately overlooked by means of .gitignore) and the latest adaptation of the record has been submitted and nothing pending.

In the event that git status has referenced "Unmanaged records:", we may need to include at least one unmanaged file.

$ git status

On branch master

Your branch is ahead of 'origin/master' by 1 commit.

  (use "git push" to publish your local commits)

Untracked files:

  (use "git add <file>..." to include in what will be committed)

xyz.txt

nothing added to commit but untracked files present (use "git add" to track).

We will use the git status, to keep monitoring the states of both the working directory and the repository.

**[How to resolve conflicts in GIT?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2913)**

Suppose you are going to pull most recent from storehouse and run over clash with neighborhood duplicate. We should attempt to comprehend this situation with underneath precedents :

git pull origin master ( Trying to get the latest from remote repo)

From giturl/projectname

\* branch            master -> FETCH\_HEAD

Updating b044a4a..gg36324

error: Entry abc' not uptodate. Cannot merge.

A plainly specific file(abc) isn't up to date. Now you will attempt to set your nearby duplicates right by running beneath commands:

git add abc

git commit -m "added files "

git pull origin master

From  giturl/projectname

\* branch            master -> FETCH\_HEAD

Auto-merging abc

CONFLICT (content): Merge conflict in abc

Automatic merge failed; fix conflicts and then commit the result.

So you decide to take a look at the changes by running mergetool :

git mergetool

Then you realize that you would like to keep your copies rather than a remote copy as the remote copy is not up to date :

git checkout --ours abc

git checkout --theirs abc

git add abc

git commit -m "using theirs"

And then we try a final time

git pull origin master

From  giturl/projectname

\* branch            master -> FETCH\_HEAD

Already up-to-date.

**[List some useful commands in GIT.](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2916)**

There are some helpful GIT commands which are generally utilized amid advancement. A portion of the essential GIT commands recorded down underneath classified into various categories:

git init Initialize a nearby Git store

git clone remoteurl/[username]/[repository-name].git Create a nearby duplicate of a remote store

git branch List branches (the reference mark indicates the present branch)

git branch - a List all branches (nearby and remote)

git branch [branch name] Create another branch

git branch - d [branch name] Delete a branch

git status Check status

git add [file-name.txt] Add a document to the arranging zone

git add- A Add all new and changed records to the organizing region

git commit- m "[commit message]" Commit changes

git rm - r [file-name.txt] Remove a document (or organizer)

git push origin- delete[branch name] Delete a remote branch

git checkout - b [branch name] Create another branch and change to it

git checkout - b [branch name] origin/[branch name] Clone a remote branch and change to it

git checkout [branch name] Switch to a branch

git checkout - Switch to the branch last looked at

git checkout - [file-name.txt] Discard changes to a record

git merge[branch name] Merge a branch into the dynamic branch

git  merge[source branch] [target branch] Merge a branch into an objective branch

git stash Stash changes in a filthy working catalog

git stash clear Remove all reserved sections

git push origin [branch name] Push a branch to your remote store

git push - u origin[branch name] Push changes to remote storehouse (and recall the branch)

git push Push changes to remote storehouse (recollected branch)

git push origin- delete[branch name] Delete a remote branch

git pull Update nearby archive to the most up to date submit

git pull cause [branch name] Pull changes from remote archive

git remote add origin remoteurl/[username]/[repository-name].git Add a remote storehouse

git remote set-url origin remoteurl/[username]/[repository-name].git Set a repo birthplace branch to SSH

git log View changes

git log - summary View changes (nitty gritty)

git diff [source branch] [target branch] Preview changes before blending

**Advanced**

[**What are the steps of git local workflow?**](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-660)

1. For starting local, we initialize our current working project directory using 'git init' or 'git init <your-project-name>'(this will create a new directory with provided your-project-name as working directory) command and on the GitHub site set up your GitHub repository.
2. Now in the local working directory, we do changes (either adding/editing files etc)
3. Then we're going to 'stage' our changes using the "git add" command
4. Then commit our changes that are in the staging area using the 'git commit -m <your custom message here>'
5. Setup a remote repository (origin) using command 'git remote add origin git@github.com:User/UserRepo.git' (you can change it later using command 'git remote set-url origin git@github.com:User/UserRepo.git')
6. Once we're ready to collaborate with others on the main repository, we'll push our changes up to our remote repository on GitHub(or the hosted remote repository) using command 'git push -u origin master'; here -u is upstream(use it only for first push command just once), origin is remote name and master is the branch's name.
7. Once everyone else had pushed their changes to the remote repository,then we'll do a pull from the remote repository to our local git repository using 'git pull' or 'git pull <remote-name> <branch-name>' command
8. Later on whenever doing new working the local repo; do 'git pull' and then start new work.

[**What are the steps of git remote workflow?**](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-661)

For example using GitHub as a remote repository

1. On the GitHub set up your remote GitHub repository
2. Then do a git's clone using 'git clone' command to create a new repository on our local system.
3. Now in the local working directory we do changes(either adding/editing files etc)
4. Then we're going to 'stage' our changes using the "git add" command
5. Then commit your changes that are in the staging area using the 'git commit -m <your custom message here>'
6. Setup a remote repository(origin) using command 'git remote add origin git@github.com:User/UserRepo.git' (you can change it later using command 'git remote set-url origin git@github.com:User/UserRepo.git')
7. Once we are ready to collaborate with others on the main repository, we'll push our changes up to our remote repository on GitHub(or the hosted remote repository) using command 'git push -u origin master'; here -u is upstream(use it only for first push command just once), origin is remote name and master is the branch's name.
8. One everyone else had pushed their changes to the remote repository,then we'll do a pull from remote repository to our local git repository using 'git pull' or 'git pull <remote-name> <branch-name>' command
9. Later on whenever doing new working the local repo; do 'git pull' and then start new work.

#### [How to do git configuration to get or set various user options?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-662)

When reading/getting, the values are read from the system, global and repository local configuration files by default, and options --system, --global, --local and --file <filename> can be used to tell the command to read from only that specific location. When writing, the new value is written to the repository local configuration file by default(--local), and options --system, --global, --file <filename> can be used to tell the command to write to that location.

Set a user's email-id (to be recorded in any newly created commits) : git commit --global user.email "sc.2017.india@gmail.com"

Set a user's full name (to be recorded in any newly created commits) : git commit --global user.name "Tushar Tandulkar"

Set signed key for a signed tag or commit : git config --global user.signingkey <your-signed-key-here> Command 'git config --global --list' will show all of the current global settings/options.

Configuration saved using '--global' is stored in a file '.gitconfig' in the current user's home directory. User's name and email-id should be set before using git further.

#### [What are the steps to permanently remove a committed file from the repository?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-664)

To remove a file for example 'myfile.txt'; which is already committed and a part of repository, the first step is to 'staging' this file deletion using 'git rm myfile.txt' command. After this command if we use 'git status' it will show message e.g. 'deleted: myfile.txt' . Now as second step to make it permanently we have to commit this staging using 'git commit -m <your message>' command.

Another way is if we remove the file 'myfile.txt' from the working directory manually for example using terminal rm command or using menu as we delete file normally using operating system's GUI. post manual removal the 'git status' command will display the message 'Changes not staged for commit:' and file information like 'deleted: myfile.txt'. To stage these manual changes(we can say same behaviour as working directory changes of add/remove/update files) we have to use -u (recursively update) option with add e.g. 'git add -u'(pre git 2.0 version) or 'git add .'(git 2.0 and above version) command. Then use git commit command to make them permanent.

#### [How do you restructure the working directory files in git repository?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-665)

For example if we have a file called 'myfile.txt' to restructure/move in the sub directory 'mydir' under git's working directory; rather than cut and paste(manual mv command), we can use 'git mv <file-name> <target-dir-name>' e.g. 'git mv myfile.txt mydir'. This will move the file to provided directory and marks it to 'staging'. If we use 'git status' command now, it will display changes available to commit e.g. 'renamed: myfile.txt -> mydir/myfile.txt'. Now using git commit command make this restructuring permanently in the repository.

If we move file/folder out of the working directory (basically like permanently removing a committed file from the repository) then post movement we have to use 'git add -u' command to mark it staging and then commit command to make it permanently.

#### [How to configure git to not to add and commit non-source files like IDE configuration or debug log files?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-666)

In the working directory, create a new file '.gitignore'; basically this file's each line will contain a entry(file or folder name or file extension with filter) to be ignored by git to add or commit in the repository. for example if we want ignore all of the file with '.debug' extension then add a entry '\*.debug' in the file. Stage this file '.gitignore' using 'git add .' and then commit it to repository.

**[How do you setup and use SSH authentication to connect with github/remote repository?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-667)**

1. To do SSH setup for git; we need to create a '.ssh' directory in the user's home directory.
2. Then in inside (using 'cd' command) this (e.g. '/Users/SandeepChoudhary/.ssh') newly directory, use command 'ssh-keygen -t rsa -C "<your-email-id>"'. Here -t is used to specify type of key to create and -C(in upper case only) to provide the comment in this case the email-id.
3. Then during execution of this command, provide inputs like name of file to save the key(just press enter so by default it will be saved in 'id\_rsa' file, provide passphrase. Now a file 'id\_rsa.pub' will be created in '.ssh. Directory.
4. Now on the github under 'Account Settings' > 'SSH keys' > 'Add SSH key', provide custom title as this ssh key will be tightly coupled with your specific machine where ssh key is generated. Under key option, copy paste the content of the 'id\_rsa.pub' file from your machine to here and click 'add key'.
5. Now on your machine using command 'ssh -T git@github.com' you will establish a ssh connection/tunnel with github.

#### [How do you manage set of tracked/remote repositories?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-668)

We add a remote repository(origin) using command 'git remote add origin git@github.com:User/UserRepo.git' (you can change it later using command 'git remote set-url origin git@github.com:User/UserRepo.git').

We can list the remote repositories using 'git remote -v' command and this will display the name and the url of the remote repositories.

We push our changes up to our remote repository using command 'git push -u origin master'; here -u is upstream(use -u only for first push command just once), origin is remote name and master is the branch's name.

Then we'll do a pull from remote repository to our local git repository using 'git pull' or 'git pull <remote-name> <branch-name>' command to receive all your remote changes (commits) from the remote repository.

#### [How to know what are the different saved states/stash of the current working directory?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-669)

Using the command 'git stash list'. The output of this command is a list of the saves states starting from zero the latest to the oldest one. Each entry has the stash index starting from zero e.g. stash@{n} where n is index, followed by the branch name and the summarized description.

#### [What is git stash drop?](https://www.knowledgehut.com/interview-questions/git#collapse-beginner-670)

If at any point of time we want to remove a single stashed state from the stash list, we use the command 'git stash drop'(it will remove the latest from list i.e. stash@{0}) or 'git stash drop stash@{n}'; here stash@{n} must be a valid stash log reference as displayed in stash list command's output.

**[Explain the steps to work on any production issue in GIT](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2897)**

Let's go through the steps to understand flow in case we are working on any production issue :

**Step 1**: We will create a local branch for developing any feature /bug which will be a child branch inherited from the production branch. The below command will create headerissuebranch locally on which developer would work and fix production issues.

git checkout -b headerissue production branch

We can check the branches existing locally by running below command.

**step 2**: git branch -List all branches existing locally

You can progress the changes by running below command

**step 3**:git status- To see any changes in the branch

**step 4**:Once you are done with changes committing changes locally not remotely.  Once unit testing and Release testing gets completed, you can move to merge the changes in the production branch.

**step 5**:  Now you can move to the production branch

    $ git checkout productionbranch

Switched to branch productionbranch and merge the changes once

   $ git merge --no-ff headerissue (merging with issue branch)

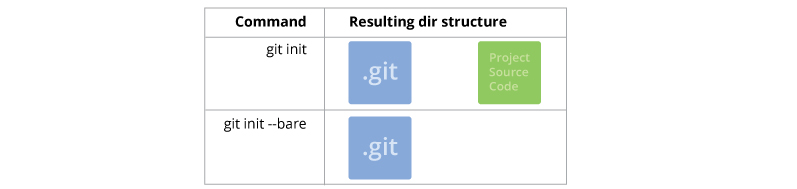
   $ git branch -d headerissue(delete local branch)

   $ git push origin productionbranch (pushing issue fix in production branch)

**[What do understand by ‘bare repository’ in Git?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2902)**

A git bare repository is helpful when you would like to collaborate with other users but don’t want to use a hosting service like GitHub. In such cases, you can set up a remote server that is only accessible by your internal team, create a bare repository over there, which acts as the central git server from which each of you can push and pull your code from your local boxes.

A bare repository made with git init -- bare is for sharing. On the off chance that you are working together with a group of designers, and need a spot to share changes to a repo, at that point you will need to make a bare repo in a centralized spot where all clients can push their changes (frequently the simple decision is GitHub). Since GIT is an appropriated rendition control framework, nobody will legitimately alter documents in the common brought together storehouse. Rather designers will clone the mutual bare repo, make changes locally in their working duplicates of the repo, at that point push back to the commonly exposed repo to roll out their improvements accessible to different clients.



Since nobody ever makes alters straightforwardly to records in the mutual bare repo, a working tree isn't required. Actually, the working tree would simply get in the way and cause clashes as clients to push code to the store. This is the reason uncovered storehouses exist and have no working tree.

How do I make existing non-bare repository bare?

After making sure that there are no uncommitted changes, etc.:

$ mv repo/.git repo.git

   $ git --git-dir=repo.git config core.bare true

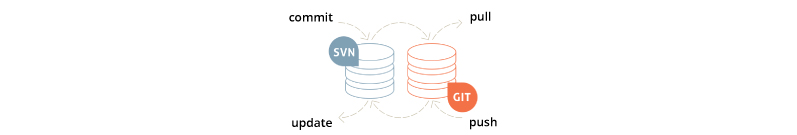
   $ rm -rf repo

**[What is a Sub Git?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2903)**

SubGit is a device for empowering a consistent and calm SVN to Git relocation. It makes a writable Git reflection of a neighbourhood or remote Subversion archive and utilizes both Subversion and Git as long as you can imagine. It creates a bi-directional mirror that can be used for pushing to Git as well as committing to Subversion. SubGit also takes care of synchronization between Git and Subversion. It requires access to your GitLab server as it interacts with the Git repositories directly in a filesystem level.

Some of the benefits of using SubGit tool are as follows:

1. Single, concentrated mirror area and set-up
2. No uncommon directions - utilize the full intensity of Git on the customer side
3. Move clients and instruments to Git at agreeable pace with zero vacation and no medium-term switches
4. Handle standard, custom or single-catalogue SVN ventures



**[What is git hooks?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2904)**

Sometimes there is a need to enforce policies, ensure consistency, and control your environment, and even handle deployment tasks. Git hooks are a basic idea that was actualized to address a need. When creating programming on a common undertaking, keeping up style direct norms, or while conveying programming (all are circumstances that git is regularly included with), there are frequently dull errands that you will need to do each time an activity is taken. Git snares are occasion based. When you run certain git directions, the product will check the hooks catalogue inside the git store to check whether there is related content to run. Git hooks are run locally.

Some example hook scripts include:

* **pre-commit**: Check the commit message for spelling errors.
* **pre-receive**: Enforce project coding standards.
* **post-commit**: Email/SMS team members of a new commit.
* **post-receive**: Push the code to production.

Every Git repository has a .git/hooks folder with a script for each hook you can bind to. You're free to change or update these scripts as necessary, and Git will execute them when those events occur.

Git hooks are divided into two categories:

* **Client-Side Hooks:**Hooks that are called and executed on the committer's computer.
* **Server-Side Hooks:** These hooks are executed on servers that are used to receive pushes.

**[Why do we use git bisect?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2905)**

GIT cut up completes a paired inquiry to locate a specific relapse which is a guilty party of submitting some wrong code in archive. It is preposterous to expect to assess every single resolve to discover the relapse as it is very time-consuming. Suppose you have beneath improvement history:

... - 0 - 1 - 2 - 3 - 4\* - 5 - current

You come to realize that your program isn't working accurately at the present correction, and it was working at the modification 0. So the relapse was likely presented in one of the submits 1, 2, 3, 4, 5, current.

You can attempt to check each submits, manufacture it, check if the relapse is available or not. On the off chance that there is countless, this may take quite a while. This is a straight pursuit. We can improve by completing a double pursuit. This is the thing that the git divide order does. At each progression, it attempts to decrease the number of amendments that are conceivably awful significantly.

You'll utilize the order this way:

$ git stash save

$ git bisect start

$ git bisect bad

$ git bisect good 0

Bisecting: 2 revisions left to test after this (roughly 2 steps)

[< ... sha ... >] 3

After this direction, git will check out a submit. For our situation, it'll be submitting  3. You have to fabricate your program and check whether the relapse is available. You'll likewise need to tell git the status of this modification with either git bisect awful if the relapse is available, or git bisects great on the off chance that it isn't.

How about we guess that the relapse was presented in submit 4. At that point, the relapse is absent in this update, and we tell it to git.

$ make

$ make test

... ... ...

$ git bisect good

Bisecting: 0 revisions left to test after this (roughly 1 step)

[< ... sha ... >] 5

It will then check out another submit. Either 4 or 5 (as there are just two submits). We should assume it picked 5. After manufacture, we test the program and see that the relapse is available. We at that point tell it to git:

$ make

$ make test

... ... ...

$ git bisect bad

Bisecting: 0 revisions left to test after this (roughly 0 steps)

[< ... sha ... >] 4

We test the last modification, 4. Furthermore, since the one presented the relapse, we tell it to git::

$ make

$ make test

... ... ...

$ git bisect bad

< ... sha ... > is the first bad commit

< ... commit message ... >

In this basic circumstance, we just needed to test 3 forms (3, 4, 5) rather than 4 (1, 2, 3, 4). This is a little win, however, this is on the grounds that our history is so little. On the off chance that the pursuit run is of N submits, we ought to hope to test 1 + log2 N submits with git bisect rather than generally N/2 submits with a direct hunt.

When you've discovered the submit that presented the relapse, you can ponder it to discover the issue. When this is done, you use git bisect reset to return everything on the first state before utilizing git bisect direction.

**[what are branching strategy in GIT?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2906)**

it is important to understand the basics of Git version control. There are a parcel of individuals that utilizes

IDE's and visual instruments to do the essential tasks like commit and push changes, make and union

branches, think about history and return changes, without figuring out how Git really works. The

expanding technique relies upon the span of your group, the number of groups taking a shot at the

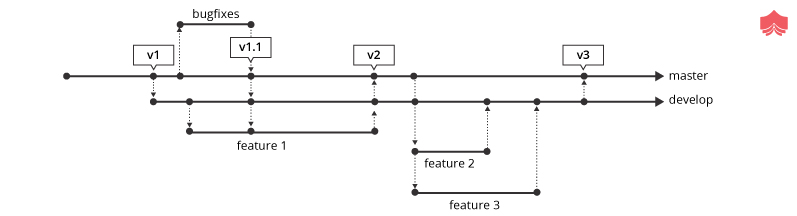
venture, if the undertaking has more than one element being created in the meantime, the recurrence

you discharge code to generation… the more unpredictable your situation, the more you depend on an

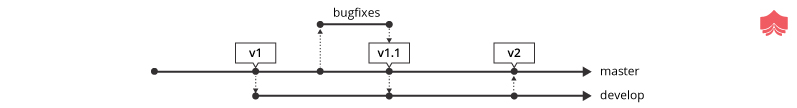
appropriate branch methodology to help it

There are three popular branching models in GIT.

* A component branch model keeps the majority of the progression for a specific element within a branch. At the point when the element is completely tried and approved via computerized tests, the branch is then converged into an ace.



* In this model, each undertaking is actualized without anyone else branch with the assignment key incorporated into the branch name. It is anything but difficult to see which code actualizes which task, simply search for the undertaking key in the branch name.



* Once the create branch has gained enough highlights for a discharge, you can clone that branch to shape a Release branch. Making this branch begins the following discharge cycle, so no new highlights can be included after this point, just bug fixes, documentation, and other discharge situated assignments ought to go in this branch. When it is prepared to dispatch, the discharge gets converged into an ace and labelled with a variant number. Moreover, it ought to be converted once again into creating a branch, which may have advanced since the discharge was started.

**[One of your teammates accidentally deleted a branch and has already pushed the changes to the central git repo. There are no other git repos, and none of your other teammates had a local copy. How would you recover this branch?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2907)**

Check out the latest commit to this branch in the reflog, and then check it out as a new branch. Reflog is a mechanism to record when the tip of branches are updated. This command is to manage the information recorded in it. Basically every action you perform inside of Git where data is stored, you can find it inside of the reflog.

git reflog won't navigate HEAD's progressive system by any stretch of the imagination. The reflog speak to the requested rundown of the submits that HEAD has indicated: it's fixed history for our repo. The reflog isn't a piece of the repo itself (it's put away independently to the submits themselves) and is excluded in pushes, gets or clones; it's absolutely neighbourhood. you can't generally lose information from your repo once it's been dedicated. On the off chance that you incidentally reset to a more established submit, or rebase wrongly, or whatever other tasks that outwardly "evacuate" submit, you can utilize the reflog to see where you were previously and git reset - hard back to that ref to reestablish your past state

**[How do you find the list of files changed in particular commit?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2914)**

Sometimes you require to inspect files which were part of a particular commit. Each commit is tagged with hash. You can use below command to see the list of files changed during particular commit.

**git diff-tree -r {hash}**



hash- commit Id

Given the submit hash, this will list down every one of the records that were changed or included that submit. The - r flag makes the order list singular records, instead of falling them into root index names as it were. The yield will likewise incorporate some additional data, which can be effectively stifled by including two or three flags::

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

https://d2o2utebsixu4k.cloudfront.net/media/images/1560418981401-image4.jpg

Here - no-commit- id will stifle the commit hashes from showing up in the output, and - name-only will just print the record names, rather than their ways.

**[In how many ways we can refer commits in GIT?](https://www.knowledgehut.com/interview-questions/git" \l "collapse-beginner-2915)**

In Git, each commit is extraordinarily recognized by a novel hash number. These hashes can be utilized to distinguish the related commit much of the time, (for example, when we attempt to check out a specific condition of the code utilizing the git checkout {hash} order).

Aside from this, Git dependably keep data on various nom de plumes to guide to a specific commit, known as refs. Likewise, every single label that we make in the repo turns into a ref (and that is actually why we can utilize labels rather than submit hashes in different git directions). Git dependably keeps up various extraordinary false names that change depending on the condition of the archive, for example, HEAD, FETCH\_HEAD, MERGE\_HEAD, and so forth.

Git dependably permits resolves to be alluded as in respect to each other. For instance, HEAD~1 alludes to the commit parent to HEAD, HEAD~2 alludes to the grandparent of HEAD, etc. In the event of consolidation commits, where the commit has two guardians, ^ can be utilized to choose one of the two guardians, for example, HEAD^2 can be utilized to pursue the second parent.

Lastly, refspecs. These are constantly used to outline and remote branches together. In any case, these can be utilized to allude to commits that live on remote branches enabling one to control and control them from a neighborhood Git condition.

**ANSIBLE INTERVIEW**

## DESCRIPTION

Ansible is a configuration management tool which is mainly used for application deployment, task automation, and server configuration which greatly enhances business agility by reducing operational costs, response times, and providing greater benefits to organisations. It is a very popular tool which is fairly easy to use and helps in IT orchestration and can handle complex tasks.  
  
We have come up with a few Ansible interview questions which will surely help you to be more confident in your interview. The following interview questions on Ansible have been designed by experts and intends to give you a glimpse of the interview questions which you might face while you appear for your Ansible interview:

#### [What is Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-806)

Ansible is an open source automation platform that is used for configuration management, application deployment task automation etc.

It can also do I-T orchestration to run tasks in sequence and create a chain of events which must happen on several different servers or devices. It is developed in Python language. It is useful while deploying any application using ssh without any downtime.

In simple terms Ansible enables us to define our infrastructure as code(IAC) in a simple declarative manner.

[**What are the advantages of using Ansible?**](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-807)

* We can easily configure Ansible to run and manage complex processes and functions.
* It is a resource sensitive platform. This means that it optimally utilizes available resources without compromising on the quality of the automated task.
* When Ansible is not managing remote nodes it does not run any background processes. This saves up system memory and does not overwhelm the CPA.
* Most importantly Ansible is powerful reliable and secure as it executes tasks in order and limits the transmission of potentially sensitive data between machines.
* It is easy to set up and configure which is a real advantage when it comes to handling complex processes and large scale deployments with its streamlined efficiency.
* It has proven to be a boon for I.T. administrators as it automates to mundane tasks of automation and system upkeep.
* Ansible has a huge engaging online presence in the form of a support community with thousands of experienced members sharing their knowledge and expertise.
* Ansible does not require any dedicated agents to work on it. It only needs open SSH or python on the remote servers. means the ability to manage remote systems without installing additional agents.
* The core idea in Ansible is that we only run and spools automation as needed and we can repeat it without any side effects.
* Ansible is useful in environments where different users have different levels of access. For example, developers, administrators, and QA engineers have managed access to their respective isolated environments.
* There is no accidental risk of a developer pushing content to production.

[**Compare Ansible vs Puppet**](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-808)

* Ansible is easier to setup than Puppet
* Network Management more easy to using Ansible
* In Terms of scalability, both are highly scalable
* For Ansible 'YAML + Python' and for Puppet 'DSL + PuppetDSL' is used as a configuration language
* Costing of Ansible in production is cheaper than Puppet.
* Ansible is agentless but Puppet needs a customs agent on the remote nodes.

[**Compare Ansible vs Chef**](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-809)

* Ansible is easier to setup than Chef
* Network Management more easy to using Ansible
* In Terms of scalability, both are highly scalable
* For 'Ansible YAML' + Python and for 'DSL + Ruby' is used as a configuration language
* Costing of Ansible in production is cheaper than Puppet.
* Ansible is agentless but Chef needs a customs agent on the remote nodes.

[**How to setup Ansible?**](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-810)

There are two ways to install it.

1. Using Linux based installation procedure (e.g. using apt or yum)

* For this first add the repository to the APT using command on terminal 'sudo apt-add-repository -y ppa:ansible/ansible'
* Update the repository using command 'sudo apt-get update'
* Now install Ansible using command 'sudo apt-get install -y ansible'

1. Using python installer

* For this first install python on the local machine
* Then using 'sudo pip install ansible' command on terminal install Ansible
* Later on, it can be updated using 'sudo pip install --upgrade ansible' command

Once Ansible is installed, then verify it with command 'ansible --version' on the terminal.

[**What is Ansible Role?**](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-811)

* Roles are a way to group multiple tasks together into one container to do the automation in a very effective manner with a proper directory structure, they are built on the idea of include files and combine them to form clean, reusable abstractions.
* Roles provide a framework for fully independent or interdependent, collections of variables, tasks, files, templates, and modules.
* Each role is typically limited to a particular theme or desired end result, with all the necessary steps to reach that result either within the role itself or in other roles listed as dependencies.
* Roles themselves are not playbooks. There is no way to directly execute a role. Roles have no setting for which host the role will apply to. Top-level playbooks are the glue that binds the hosts from your inventory to roles that should be applied to those hosts.

#### [What is a Playbook?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-812)

Playbook is a way to send commands to remote computers in a scripted way and has a collection of YAML based files. Instead of using Ansible commands individually to remotely configure computers from the command line, we can configure entire complex environments by passing a script to one or more systems. For example, install and configure something. Basically, a series of resources (e.g. package, copy, file etc) that will achieve our end result in the correct order.

**[You have to ensure quality of the final deliverables. You have heard about Quality Assurance and Quality Control which are used for managing quality. What is the difference between QA and QC?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-2781)**

**Knowledge Area: Project Quality Management**

Managing the quality of the project deliverables is extremely important. Quality of the deliverables is ensured by doing both Quality Assurance and Quality Control.

Quality Assurance comes from the laid down processes and standards to do the project work. Adherence to the processes and standards assures that the final deliverables will meet the expected standards of quality. This is done proactively through the execution and development.

Quality Control is the process of monitoring the final deliverables and results created by the team after adhering to the chosen processes and standards. Quality Control uses inspection to check if the final deliverable is correct in all respect and meets all the expected specification or not.

For more detailed understanding on the above topic, refer: [Qualty assurance and quality control](https://www.knowledgehut.com/tutorials/project-management/quality-assurance-and-quality-control)

#### INTERMEDIATE

**[What is a configuration management tool?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-3572)**

Configuration management tool helps to maintain the desired state of a system. It helps to reduce deployment time and helps to keep consistent configuration across similar systems. It greatly reduces the effort required to do repetitive tasks.  It is one of the building blocks of the CI/CD pipeline. Top configuration management tools in the market are Ansible, Chef, Puppet & Salt. Configuration management tool plays a vital role in Infrastructure as code also. Let's take a real-life scenario to understand it more, if 100 web servers need to be built/configured then without Automation or configuration management tool, it can be a very tedious, lengthy and error-prone task. Also, it will require dedicated resource/efforts and there could be chances to have in-consistency in the configuration. A configuration management tool comes here as a savior and can build/configure 100 web servers very quickly with consistent configuration across all servers. In short, Configuration management tool is very helpful in configuring of systems to a particular state with consistent configuration and fewer efforts.

**[What are Ansible and its uses?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-3574)**

Ansible is a configuration management tool. It helps to automate repetitive tasks and to achieve the desired state of systems. Redhat is an official vendor for Ansible. It is very simple and easy to use and because of this many organizations adopting it rapidly. There is no need to learn any new language to work on Ansible. It uses YAML to write tasks, plays or playbooks. Ansible can be used in two ways, Ansible command line or Ansible Tower. Ansible can complement CI/CD pipeline greatly by making require configuration automatically with consistency. Thus, it helps to speed up the roll-out new changes or products to production. Ansible can be integrated with Jenkins to enhance CI/CD pipeline and to execute configuration tasks automatically from Jenkins. Ansible can also help in defining Infrastructure as a code, so you can deploy as many servers with consistent configuration.

**[Explain the term “Idempotency”.](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-3575)**

For any configuration management tool, Idempotency is a key feature. Idempotency means that you can execute one or more tasks on a server any number of times and it will not make any change if it is already in the desired state.  Idempotency is a key feature of Ansible too. With this, Ansible makes sure that it doesn’t make unnecessary changes on managed hosts. Think of automating a task using scripts and automating the same task using Ansible. If you run a script to perform a task, it will perform required steps on each and every execution of the script, on the other hand, Ansible will make changes only once, even though you execute it a thousand times. Why idempotency is required? So, let's understand it with an example: suppose you need to append a line “this is a test” to a file and you created a script for this. Now, whenever you run this script it will append the same line again and again ( until you are smart enough to include an existence test in the script ). With Ansible, you just write to append line in a file and then Ansible automatically checks, if it is already there or not.

**[What are Ansible roles and their purpose?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-3576)**

Role is a group of variables, tasks, files, and handlers that store in the standardized file structure. Roles facilitate reuse and modularization of configuration. For exp: 2 roles have been written, 1 for Apache HTTP and the second one for Apache Tomcat. Now if you need to configure HTTP and Tomcat both on a server than you can just include both roles in the playbook or if you need only one from it then include only 1 role in the playbook. In this way, there is no need for major changes in playbook other than modification of role name, thus provide reusability and modularity.  So, the role is in simple words are small blocks, which can be used as per requirement anywhere to build a big or small wall.

**[What is Ansible Galaxy?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-3577)**

Ansible Galaxy is a hub or community for finding and sharing Ansible content. Roles for provisioning infrastructure, deploying the application and for many other days today, tasks can be found and downloaded from Galaxy. Content on galaxy can be browsed over [https://galaxy.ansible.com](https://galaxy.ansible.com/). Ansible-galaxy command can be used to install, import, list, delete or search roles from/to Galaxy server. One can find playbooks to configure Apache HTTP, Tomcat, Oracle DB or many more as per need and requirement. Instead of writing long playbooks every time, Galaxy can be checked for already created playbooks/roles and if require that playbooks/roles can be modified as per need locally.

**[Difference between play and playbook?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-3578)**

Ansible play is a set of tasks that are run on one or more managed hosts. A play may include one or more tasks, and the most common way to execute play is to use a playbook. Play always starts with the top directory, which means it will start from the “- hosts” line always.

Ansible playbook contains one or more plays to have the feasibility of running multiple tasks on a different set of hosts.

Below is the example of a play:

**-** hosts**:** webservers

  tasks**:**

**-** name**:** ensure apache is at the latest version

    yum**:**

      name**:** httpd

      state**:** latest

Below is the example of a playbook:

**---**

**-** hosts**:** webservers

  tasks**:**

**-** name**:** ensure apache is at the latest version

    yum**:**

      name**:** httpd

      state**:** latest

**-** hosts**:** databases

  tasks**:**

**-** name**:** ensure postgresql is at the latest version

    yum**:**

      name**:** postgresql

      state**:** latest

#### [How Ansible Works?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-815)

On the local machine user uses playbook to configure and manage the execution YAML based scripts. Main building blocks of Ansible are playbook, plugins, modules, host inventories, group\_vars, roles, handlers, and tags. Using these components we define our configuration and scripts and that is executed on the remote nodes using SSH tunneling. Commands/steps in scripts are executed in order but execution happens in parallel on multiple remote nodes managed by Ansible. Ansible pushes out small program termed as 'modules'; which are responsible for performing all heavy liftings.

Ansible carries out these modules upon the SSH by default and eliminates them from the remote node machine when finished.

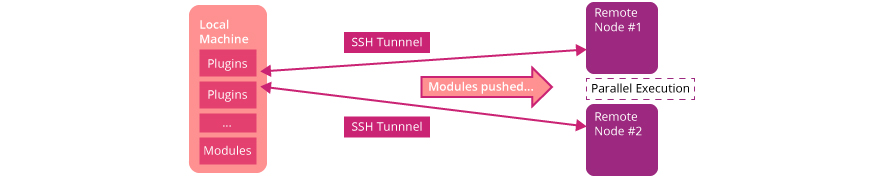
[**What is the use of Ansible**](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-816)

* It is an open source IT configuration management, deployment, and orchestration tool.
* It is unique from other management tools in many respects, aiming to provide large productivity gains to a wide variety of automation challenges as a more productive drop-in replacement for many core capabilities in other automation solutions.
* To solve major unsolved IT challenges such as clear orchestration of complex multi-tier workflows and cleanly unifying OS configuration and application
* Software deployment under a single solution.
* It is designed to be minimal in nature, consistent, secure, and highly reliable, with an extremely low learning curve for administrators, developers, and IT managers.
* To keep descriptions of IT easy to build, and easy to understand - such that new users can be quickly brought into new IT projects, and longstanding automation content is easily understood even after months of being away from a project.
* To make things powerful for expert users, but equally accessible for all skill levels, ensuring a quicker time to market for IT projects and faster, less-error prone turnaround on IT configuration change.

**[Ansible workflow?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-817)**

Workflows are only available with Enterprise-level licenses. Workflows allow configuring a sequence of disparate job templates that may or may not share inventory, playbooks, or permissions. However, workflows have ‘admin’ and ‘execute’ permissions, similar to job templates. A workflow accomplishes the task of tracking the full set of jobs that were part of the release process as a single unit.

A workflow job can have the following states (no Failed



* Waiting
* Running
* Success (finished)
* Cancel
* Error

[**What is Ansible Tower?**](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-818)

Ansible Tower is an enterprise framework for controlling, securing and managing Ansible automation with a UI and restful API.

**It provides the following features**

* Real-time Playbook Output and Exploration
* 'Push Button' Automation( trigger execution with minimum clicks)
* Enhanced and Simplified Role-Based Access Control and Auditing
* Cloud & Autoscaling Flexibility (allows nodes to request configuration on demand)
* The Ideal RESTful API for a systems management application
* Backup and Restore, replicate Ansible tower instance as needed
* Ansible Galaxy Integration ('Don’t Repeat Yourself', using centralized copies of Ansible roles, such as in Ansible Galaxy and uses of requirements.yml file)
* Inventory Support for OpenStack(This allows to easily target any of the virtual machines or images, which are running in the OpenStack cloud)
* Remote Command Execution 12. Integrated Notifications(allows to easily keep track of the status of your automation using e.g. Slack, E-mail, SMS etc)
* Red Hat Satellite 6 and Red Hat CloudForms Integration
* Run-time Job Customization (Bringing the flexibility of the command line to Tower)

#### [Describe reusability in Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-819)

Ansible playbook files can be reused in multiple ways. Imports and Include are helpful to create/break up multiple small files of a massive playbook. Roles are primarily used to manage various tasks in a package including handlers, variables, plug-in and other modules. They can also be uploaded and shared by Ansible Galaxy. Ansible Tower automatically fetches the roles that playbook needs from Galaxy, GitHub, or your local source control.

#### [What is a 'module' in Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-820)

Ansible works by connecting to the remote nodes over SSH and pushing out small programs, called 'modules' to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules and removes them when finished.

Ansible ships with a number of readymade modules (called the 'module library') that can be executed directly on remote hosts or through Playbooks. Users can also write their own modules. These modules can control system resources, like services, packages, or files (anything really), or handle executing system commands.

#### [What mean by 'facts' in Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-821)

Whenever we run an Ansible playbook, Ansible first gathers information ('facts' an exhaustive list of all the environment details) about each host in the play. Facts can be extremely helpful when we’re running playbooks; we can use gathered information like host IP addresses, CPU type, disk space, operating system information, and network interface information to change when certain tasks are run or to change certain information used in configuration files.

To get a list of every gathered fact available, we can use 'ansible munin -m setup' command with the setup module.

#### [What is a handler in Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-822)

* Handlers are just like regular tasks in a playbook, but they will only be run if a task notifies the handler. If a task that would’ve notified the handlers is skipped due to a when condition or something of the like, the handler will not be run.
* Handlers are automatically loaded from roles/<role\_name>/handlers/main.yaml. These handlers can be referenced by any task within the role, or by any tasks within any other role that lists this role as a dependency.
* Handlers will run once, and only once, after all of the tasks complete in a particular play.
* If the play fails on a particular host (or all hosts) before handlers are notified, the handlers will never be run(but we can use the command line flag --force-handlers when running the playbook to run the handler always)

#### [What is a task in Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-823)

* At a basic level, a task is a way to call an Ansible module. In playbook goal of a play is to map a group of hosts to some well-defined roles, represented by things Ansible calls tasks.
* The task file is the main part of a role. If roles/<role\_name>/tasks/main.yaml exists, all the tasks therein and any other files it includes will be embedded in the play and executed.

#### [Describe variables in Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-824)

There are two types of variables that can be defined in a role:

1. Role variables: loaded from roles/<role\_name>/vars/main.yaml
2. Role defaults: which are loaded from roles/<role\_name>/defaults/main.yaml

* The difference between 'variables' and 'defaults' is the precedence order.
* Role defaults are the lowest order variables. Any other definition of a variable will take precedence over a role default.
* Role defaults can be thought of as placeholders for actual data, a reference of what variables a developer may be
* interested in defining with site-specific values.
* Role variables, on the other hand, have a higher order of precedence. Role variables can be overridden, but generally, they are used when the same data set is referenced more than once within a role. If the data set is to be redefined with site-local values, then the variable should be listed in the role defaults rather than the role variables.

#### [What is meant by tags in Ansible?](https://www.knowledgehut.com/interview-questions/ansible#collapse-beginner-825)

* If we have a large playbook, it may become useful to be able to run only a specific part of it rather than running everything in the whole Ansible playbook, Ansible supports a 'tags:' attribute for this.
* When we execute a playbook, we can filter tasks based on 'tags' in two ways:
* On the command line, with the --tags or --skip-tags options
* In Ansible configuration settings, with the TAGS\_RUN and TAGS\_SKIP options
* We can apply the same tag to more than one task and do task inheritance as well.

**[Explain Ansible tags?](https://www.knowledgehut.com/interview-questions/ansible" \l "collapse-beginner-3579)**

Ansible tags are very useful when the playbook is large. It may become useful to be able to run only a specific part of it rather than running everything in the playbook. For example, a playbook creates 3 users but in some cases, you need only 1 user from it, so here tag is a big savior.

During playbook execution, tasks can be filtered out in two ways:

* On the command line, with –tags or –skip-tags options.
* In Ansible configuration settings, with the TAGS\_RUN and TAGS\_SKIP options.

Skip tags can be used to skip one or more tag, it can be defined in the command line as –skip-tags   ‘tag1,tag2’. On the other hand, --tags helps to execute one or tags associated with the mentioned tag.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q1. What Is Ansible ?**  
Ansible is a software tool to deploy application using ssh without sny downtime.It is also used to manage and configure software applications. Ansible is developed by Python language.

**Q2. What is Ansible galaxy?**  
Galaxy refers to bothe website and CLI tool used to the interact with the website where you can download and share roles with other members of the ansible communty.

**Q3. What is Forks in Ansible**  
Forks is a way to improve your ansible performance defining how many ansible processes will be created to communicate with remote hosts.

**Q4. Briefly Explain Pipelining in Ansible.**  
Pipelining allows Ansible to use stream commands over a single connection instead of opening connection for each ansible command.

**Q5. How can we use controlpersist to speed up ansible deployment?**  
This allows us to create a single master connection that can be reused subsequently for a given amount of time.

**Q6. Explain “fire and forget” concept in ansible.**  
This allows us to  run a task without waiting for completion. You simply run the task Async and set poll=0.  Later in the playbook, use async\_status to check the status of the job.

**Q7. How do we make a variable available to a host or group without including it in the inventory file?**  
You can create a variable file under group\_vars. For example, lets say we want to make a variable available too the **webserver** host group, you simple create **group\_vars/webservers** and define the variable inside the file.

**Q8. What’s The Use Of Ansible ?**  
Ansible can be used in IT infrastructure to manage and deploy software applications to remote nodes. For example, let’s say you need to deploy a single software or multiple software to 100’s of nodes by a single command, here ansible comes into picture, with the help of Ansible you can deploy as many as applications to many nodes with one single command, but you must have a little programming knowledge for understanding the ansible scripts.  
We’ve compiled a series on Ansible, title ‘Preparation for the Deployment of your IT Infrastructure with Ansible IT Automation Tool‘, through parts 1-4 and covers the following topics.

**Q9. Why would you want to disable ansible facts.**  
You can disable facts if it its not being used to save on memory used for storing the variables created during facts/

**Q10. What are ansible strategies?**  
Ansible strategies are plugins that modifies the way ansible works. For example, the linear strategy executes task on the host in series waiting for all hosts to complete before moving to next task while free moves on to next task once it finish execution on a host. For debug strategy, it executes linear but triggers debugger on failure.

**Q11. How can I set the PATH or any other environment variable for a task or entire playbook?**  
Setting environment variables can be done with the environment keyword. It can be used at the task or the play level:

environment:

PATH: "{{ ansible\_env.PATH }}:/thingy/bin"

SOME: value

Note : starting in 2.0.1 the setup task from gather\_facts also inherits the environment directive from the play, you might need to use the |default filter to avoid errors if setting this at play level.

**Q12. How Do I Submit A Change To The Documentation ?**  
Documentation for Ansible is kept in the main project git repository, and complete instructions for contributing can be found in the docs.

**Q13. When Should I Use {{ }} ? Also, How To Interpolate Variables Or Dynamic Variable Names ?**  
A steadfast rule is ‘always use {{ }} except when when:‘. Conditionals are always run through Jinja2 as to resolve the expression, so when: failed\_when: and changed\_when: are always templated and you should avoid adding {{}}.  
In most other cases you should always use the brackets, even if previouslly you could use variables without specifying (like with\_ clauses), as this made it hard to distinguish between an undefined variable and a string.  
Another rule is ‘moustaches don’t stack’. We often see this:

{{ somevar\_{{other\_var}} }}

The above DOES NOT WORK, if you need to use a dynamic variable use the hostvars or vars dictionary as appropriate:

{{ hostvars[inventory\_hostname][‘somevar\_’ + other\_var] }}

**Q14. How do I handle different machines needing different user accounts or ports to log in with?**  
Setting inventory variables in the inventory file is the easiest way.  
Ansible 2.0 has deprecated the “ssh” from ansible\_ssh\_user,  ansible\_ssh\_host, and ansible\_ssh\_port to become ansible\_user, ansible\_host, and ansible\_port.

If you are using a version of Ansible prior to 2.0, you should continue using the older style variables (ansible\_ssh\_\*). These shorter variables are ignored, without warning, in older versions of Ansible.

For instance, suppose these hosts have different usernames and ports:

[webservers]

[asdf.example.com](http://asdf.example.com/) ansible\_port=5000 ansible\_user=alice

[jkl.example.com](http://jkl.example.com/) ansible\_port=5001 ansible\_user=bob

You can also dictate the connection type to be used, if you want:

[testcluster]

localhost ansible\_connection=local

/path/to/chroot1 ansible\_connection=chroot

[foo.example.com](http://foo.example.com/) ansible\_connection=paramiko

You may also wish to keep these in group variables instead, or file them in a group\_vars/<groupname> file. See the rest of the documentation for more information about how to organize variables

**Q15. How To Install Ansible ?**  
The best way to get Ansible for Ubuntu is to add the project’s PPA (personal package archive) to your system.

To do this effectively, we need to install the software-properties-common package, which will give us the ability to work with PPAs easily. (This package was called python-software-properties on older versions of Ubuntu.)

* sudo apt-get update
* sudo apt-get install software-properties-common

**Once the package is installed, we can add the Ansible PPA by typing the following command**

sudo apt-add-repository ppa:ansible/ansible

Press ENTER to accept the PPA addition.

**Next, we need to refresh our system’s package index so that it is aware of the packages available in the PPA. Afterwards, we can install the software:**

* sudo apt-get update
* sudo apt-get install ansible
* We now have all of the software required to administer our servers through Ansible.

**Q16. How Do I Generate Crypted Passwords For The User Module ?**  
mkpasswd –method=sha-512

If this utility is not installed on your system (e.g. you are using OS X) then you can still easily generate these passwords using Python. First, ensure that the Passlib password hashing library is installed.

pip install passlib

Once the library is ready, SHA512 password values can then be generated as follows:  
python -c “from passlib.hash import sha512\_crypt; import getpass; print sha512\_crypt.encrypt(getpass.getpass())”

Use the integrated Hashing filters to generate a hashed version of a password. You shouldn’t put plaintext passwords in your playbook or host\_vars; instead, use Vault to encrypt sensitive data.

**DOCKER INTERVIEW**

**BASIC**

Docker is an open source software development platform which runs software packages called “containers”. A container is a standard unit of software that packages up all the dependencies of an application so that the application runs quickly and reliably from one computing environment to another.

Today there are more than thousands of vacancies available for the Docker developers, only thing is you must be thorough with all the component of Docker technologies.  It is very difficult for a person to grab a job in today’s date. Companies look forward to hiring a candidate who has a deep knowledge of the subject. The average pay for a Software Engineer with Docker skills is [$90,805](https://www.payscale.com/research/US/Job=Software_Engineer/Salary/ddd2cd6f/Docker) per year. This often confuses the candidate about which type of docker interview questions to prepare for.

With research we have bought you a few Docker interview questions that you might encounter in your upcoming interview. here will help you let out find all the solutions that are frequently asked in your upcoming Docker interview. All these docker interview questions and answers for experienced and freshers will alone help you to crack the docker interview and make you the best among all your competitors. So, in order to succeed in the interview you need to read and re-read these docker interview questions and their solutions. You can also enhance your skills through training on [Docker](https://www.knowledgehut.com/search?q=docker+).

Docker interview questions and answers here will also help you on your way to master your skills and will take you to the giant world where worldwide and local businesses, huge or medium, are picking up the best and quality Docker professionals. Once you are prepared with these docker interview questions and answers you will be able to pass the interview easily.

Keep learning and practicing these top industry-selected interview questions on docker that will help you prepare for your docker interview. Have a pretty good idea of what to expect in your docker job interview.

Do keep visiting these docker interview questions which will enable you to crack your upcoming interviews easily.

All the best for your upcoming Docker interview.

**[How is Docker different from standard virtualization using VMs?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2873)**

Virtual Machines (VMs) virtualize the underlying hardware. They run on physical hardware via an intermediation layer known as a hypervisor. They require additional resources are required to scale-up VMs.

They are more suitable for monolithic applications. Whereas, Docker is operating system level virtualization. Docker containers userspaceace on top the of host kernel, making them lightweight and fast. Up-scaling is simpler, just need to create another container from an image.

Generally, Docker is more suitable for Microservices based cloud applications.

**[What are the major components of Docker Architecture?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2874)**

The four major components of Docker are daemon, Client, Host, and Registry

**Docker daemon:**It is also referred to as ‘dockerd’ and it accepts Docker API requests and manages Docker objects such as images, containers, networks, and volumes. It can also communicate with other daemons to manage Docker services.

**Docker Client:** It is the predominant way that enables Docker users to interact with Docker. It sends the docker commands to docked, which actually executes them using Docker API. The Docker client can communicate with more than one daemon.

**Docker Registry:** It hosts the Docker images and is used to pull and push the docker images from the configured registry. Docker Hub is the public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. However, it is always recommended for organizations to use own private registry.

**Docker Host:** It is the physical host (VM) on which Docker Daemon is running and docker images and containers are created.

**[What is a Volume in docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2875)**

A data volume is a specially-designated directory that is located outside of the root filesystem of a container (i.e. created on the host), designed to persist data, independent of the container’s life cycle. This allows sharing data within containers by importing volume directory in other containers.

Data volumes provide several useful features:

* Data volumes persist even if the container itself is deleted.
* Data volumes can be shared and reused among containers.
* Changes to a data volume can be made directly.
* Volumes can be initialized when a container is created.

**[When is .dockerignore file used?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2876)**

A typical Dockerfile contains one or more COPY commands to copy files and/or folders from the developer machine to the docker image, which eventually become part of the container. While copying folders to a docker image, it is quite possible that some unwanted files are also copied to the image. This may create a bulky image and hence cause performance issues in the container.

In order to avoid this, we can create a file named **.**dockerignore along with Dockerfile in the same directory. This file is used to list all the files and directories that need to be excluded while copying folders onto the image. It contains a pattern and none of the files matching it is added to the image. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images.

**[What is docker-compose?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2877)**

Compose is a tool provided by Docker for defining and running multi-container applications together in an isolated environment. Either a YAML or JSON file can be used to configure all the required services like Database, Messaging Queue along with the application server. Then, with a single command, we can create and start all the services from the configuration file.

It comes handy to reproduce the entire application along with its services in various environments like development, testing, staging and most importantly in CI as well.

Typically the configuration file is named as docker-compose.yml. Below is a sample file:

version: '3'

services:

  app:

    image: appName:latest

    build: .

    ports:

    - "8080"

    depends\_on:

      - oracledb

    restart: on-failure:10

  oracledb:

    image: db:latest

    volumes:

      - /opt/oracle/oradata

    ports:

      - "1521"

**[What is Docker Hub?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2878)**

Docker Hub is a service provided by Docker for finding and sharing container images. The default version of Hub is the cloud-based registry that hosts all the public docker images like Ubuntu, Linux, etc.

We need to create repositories to push and pull the docker images, allowing us to share container images within our team, organization, customers. In the case of public repositories, we can share the images with the entire Docker community.

Docker images are pushed to Docker Hub through the ‘docker push’ command. A single Docker Hub repository can hold many Docker images.

It also allows you to link repositories with GitHub in order to automate building, testing and deploying of our application images. It provides a centralized resource for container image discovery, distribution and change management, collaboration and workflow automation throughout the development pipeline.

We can also use third-party Repository tools like Nexus and JFrog Artifactory to store and manage docker images.

**[Explain basic Docker usage workflow.](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2879)**

Everything starts with the Dockerfile. The Dockerfile is the source code of the Image.  
Once the Dockerfile is created, you build it to create the image of the container. The image is just the "compiled version" of the "source code" which is the Dockerfile.

Once you have the image of the container, you should redistribute it using the registry. The registry is like a git repository -- you can push and pull images.

Next, you can use the image to run containers. A running container is very similar, in many aspects, to a virtual machine (but without the hypervisor).

**[What are the steps involved while using Docker for application development?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2880)**

All the steps below are based on the prerequisite that Docker is already installed on the machine:

a)**Create Dockerfile:**

The initial step is to create a Dockerfile file using a suitable base image along with all the required steps/commands, like setting environment variables, adding application jar, etc. This creates several layers on the existing base image.

b) **Build image:**

Once the Dockerfile is ready, we can either use docker command or via a Gradle task to generate a docker image. This image contains all the application dependencies required to run the application in a container.

docker build -t-test/security tool.

c) **Run the image:**

Once the docker image is built, we can create and start the container using command.

docker run --name rest\_tool test/security tool.

d) **Start Containers using Compose:**

In case we have multiple containers constituting an application like database, messaging queue, etc.; then it is advisable to use docker-compose to run multiple containers simultaneously. It is also useful in the CI pipeline for running the application and performing tests.

docker-compose up

e) **Test the Application:**

After the containers are up and running, the application is ready for Integration or Acceptance tests to be performed. Ideally, it is integrated into the CI pipeline for determining if the new code changes are affecting the existing flows.

f) **Push image:**

Typically in a container-based development environment, the deliverable artifact is a docker image. This image needs to be published to an internal Registry like Artifactory so that it can be propagated to next levels like Continuous Delivery and Deployment pipelines.

docker push test/securityTool

g) **Production orchestration:**

Ideally, organizations need to use orchestration tools like Kubernetes to run the containers in a Pod to perform load balancing, service discovery, etc in a production environment. It also helps in providing scalability and high availability of the application.

**INTERMEDIATE**

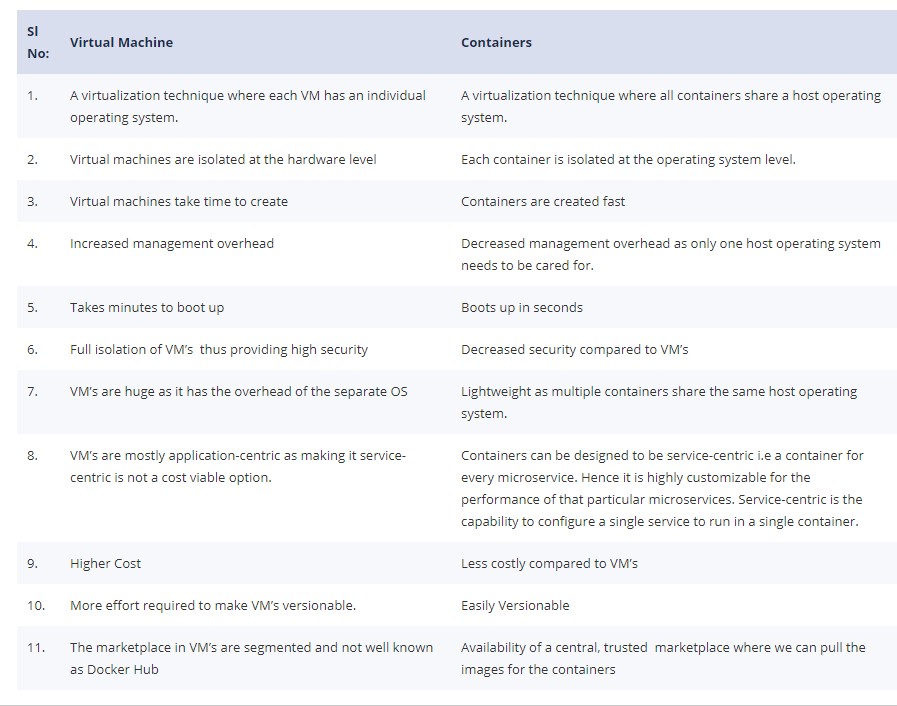
**[What are containers?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2858)**

A container is a standalone unit of software that packages together applications with all its dependencies and configurations. Containers help in abstracting applications from the computing environment in which they run.  Containers help in running the applications correctly in various computing environments. This helps to resolve the biggest problem “ It doesn’t work in my system. Multiple containers in an isolated manner can co-exist in a host system. All containers share the common host operating system. Thus making the containers lightweight and more preferable than virtual machines.

Detaching applications from its environment helps container-based applications to be deployed faster and on any on-premise systems or cloud. Developers with the help of containers can now concentrate on how to develop an application rather than how to run an application on different environments.IT operations teams focus on deployment and management rather than with application details such as specific software versions and configurations. Main competitors for containers are the virtual machines. Virtual machines are not lightweight compared to containers.

Containerization as a strategy is used as a practice across digital transformation due to its capability to bring a DevOps culture. The Dockerfile which defines the configuration of the image also acts as Infra as code up to some extent. Thus enabling teams to abstract their infrastructure in the form of code.

**[Tell me the differences between virtual machines and containers?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2860)**



**[What is Docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2861)**

Docker is an open source container platform, from Docker, Inc launched in 2013. Docker enables operating system level virtualization. Docker is available as free and enterprise version. Docker containers, unlike Virtual Machines, share the same host operating system. It has less overhead compared to VM’s. Docker created the industry guidelines for containers, making it more portable and secure. Docker is having the best isolation capability that is beneficial when more containers run on the same host system. Docker could be run on any system on-premises or cloud. Using docker we can create, start, stop, move, pause, unpause or delete a container.

A Docker container can connect to one or more networks, attach storage to it, and even create a new image depending on the current state of the container. Docker also helps in controlling how the containers are isolated from other containers or host machine. Dockerfile in docker platform helps to achieve infrastructure as code thus enabling consistent and reliable computing environment. This reduces a lot of configuration and infrastructure related defects that would improve the quality of applications.

Docker is supported on all the latest technologies like Google Cloud Platform and by Google Kubernetes Engine. Docker was primarily developed for Linux systems but now also supports Windows and Mac Os.

**[What are the container platforms available other than docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2862)**

According to 2018 reports Docker is the most popular container platform constituting 83% of container space. It’s preferred because of the ease with which the user can work with. Docker also provides greater isolation between the containers on the same host operating system. Few of the other popular ones are

CoreOS rkt

12 percent of production containers were rkt (pronounced “Rocket”) from CoreOS. rkt supports two types of images: Docker and appc. The main advantage of rkt is its usage with Kubernetes.In Kubernetes, an rkt container runtime can be declared as

“ kubelet --container-runtime=rkt”.

Compared with Docker, rkt has fewer third-party integrations. Red Hat recently acquired CoreOS. Application Container Image or appc is defined by the App Container spec. An ACI contains all necessary files to execute an application and an image.

Mesos Containerizer

4 percent of production containers were Mesos from Apache. Mesos supports both Docker and appc image types. It works better with the frameworks of big data applications.  Mesos is not a standalone container and requires the Mesos framework to make it run.

LXC Linux Containers

1 percent of containers were LXC Linux Containers. LXC also has a great active community around it. The main disadvantage of LXC is its incompatibility with Kubernetes.

**[Why Docker is chosen compared to other container platforms?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2863)**

1. Docker has a huge user base in comparison with its competitors. According to a recent report, by an infrastructure monitoring tool company, it claims 25 % of companies are already using containerization technology. Out of which 83 % of containers are docker containers. This user base includes big banks, manufacturing companies, large product firms, etc.  Hence Docker is the most popular option to choose from the existing container technologies.
2. Docker being an open source technology enjoys huge community support and hence it has been the most preferred containerization tool in the industry. Docker was primarily developed for Linux but now supports Windows and Mac OS environments.
3. There is no container limitation on running Docker as the underlying could be anything like a laptop/cloud system. It only depends on the host system’s OS resources.
4. Docker HUB forms the repository of docker images. Docker Hub users can upload and download the docker images. A lot of applications are released as docker images which enable reusability and interoperability.
5. Docker is extensively documented technology available in the containerization space. That makes it easy for even the first time users to work with it.
6. Docker works very well with  Google Cloud Platform and by Google Kubernetes Engine and across different cloud platform providers.  Docker is also well supported by other configuration management platforms like Chef, Ansible, Puppet, etc

**[What are the basic actions performed on the docker container?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2864)**

Basic actions on Docker containers are:

* **Create a docker container**

Following command creates the docker container with the required images.

docker create --name <container-name> <image-name>

* **Run docker container**

Following command helps  to execute the container

docker run -it -d --name <container-name> <image-name> bash

Main steps involved in the run command is

1. Pulls the image from Docker Hub if it’s not present in the running environment
2. Creates the container.
3. The file system is allocated and is mounted on a read/write layer.
4. A network interface is allocated that allows docker to talk to the host.
5. Finds an available IP address from pool.
6. Runs our application in our case “bash” shell.
7. Captures application outputs

* **Pause container**

Processes running inside the container is paused. Following command helps us to achieve this.

docker pause <container-id/name>

Container can’t be removed if in a paused state.

* **Unpause container**

Unpause moves the container back to run the state. Below command helps us to do this.

docker unpause <container-id/name>

* **Start container**

If container is in a stopped state, container is started.

docker start <container-id/name>

* **Stop container**

Container with all its processes is stopped with below command.

docker stop <container-id/name>

To stop all the running Docker containers use the  below command

docker stop $(docker ps -a -q)

* **Restart container**

Container along with its processes are restarted

docker restart <container-id/name>

* **Kill container**

A container can be killed with below command

docker kill <container-id/name>

* **Destroy container**

The entire container is discarded. It is preferred to do this when the container is in a stopped state rather than do it forcefully.

docker rm <container-id/name>

**[What are the disadvantages of Docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2865)**

Like any other technologies, Docker has its own advantages and disadvantages. Docker is not a silver bullet and needs care in architecting and orchestrating docker containers keeping below points in mind.

* Docker is not so fast compared to bare metal servers. Even though containers are lightweight and so easy to boot up it is subjected to decreased network performance. Performance of the containers are affected as a single operating system caters to multiple containers.
* **Integrability:** Even though Docker is open source some of the container products don’t work with all. This may be due to the competition prevailing in the market. For eg:  OpenShift which is a container-as-a-service platform from Red Hat, only works with the Kubernetes.
* **Data loss in containers:** When the container exits the data will be lost. Data could be saved through volume mounting like Docker Data Volumes but more hard work needs to be done in this space.
* **Poor or no GUI:** Containers were used mainly for deploying server application that doesn't require GUI. There are some methods to run GUI app inside containers but it’s somewhat clumsy these days.
* **More suitable for applications that use microservices:** Generally the docker’s benefit is to ease the application delivery by providing a packaging mechanism but the true benefit comes when we use microservices.
* There are other virtualization techniques like unikernels which is based on library operating systems. Unikernels provide improved security, more optimisation and boots faster.

**[What is docker image?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2866)**

A Docker image is an executable file, that creates a Docker container. An image is built from the executable version of an application together with its dependencies and configurations. Running instance of an image is a container.

Docker image includes system libraries, tools, and other files and dependencies for the application. An image is made up of multiple layers. Layered structure helps the developers to reuse already available static image layers from the Docker Hub, for different projects. This saves the developers time. Each image has a base layer which could be already available in Docker Hub or built from scratch. Then a readable/writable layer over the static layer is created that helps to customise the container. Each layer of the docker image could be verified in /var/lib/docker/aufs/diff, or via the Docker history command.

Storage drivers are used to managing the image layers. A writable layer created while creating the container is called the container layer. Multiple containers can share the same base layer and have their own writable layer. Few commands that can be used with images are:

docker history shows the history of an image and its layers.

docker update helps to update the container configurations.

docker tag creates a tag for the container and organizes container images.

docker search looks for the image in Docker Hub

docker save allows saving of images.

docker rmi removes one or multiple images.

**[What is the difference between ADD and COPY in Dockerfile?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2881)**

The ADD instruction is used to copy files from build context to the image. Apart from regular files, it also allows URL and archive (tar, gzip, etc) files as the <source> parameter.

When a URL is provided, a file is downloaded from the URL and copied to the <destination>.

It automatically unpacks compressed files, if the <source> argument is a local file in a recognized compression format (tar, gzip, bzip2, etc).

Whereas COPY does a straight-forward, as-is copy of files and folders from the build context into the container. It doesn't support URLs or gives any special treatment to archives.

Anything that you want to COPY into the container must be present in the local build context.

The recommendation from the Docker team is to use COPY in almost all cases.

**[What is the difference between ENTRYPOINT and CMD in Dockerfile?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2882)**

ENTRYPOINT is a definition in Dockerfile that specifies a command that will always be executed when the container starts. It allows us to configure a command along with its parameters that will run as an executable in a container. Even if we do not specify any ENTRYPOINT, we may inherit it from the base image specified using the FROM keyword in your Dockerfile. Most of the official Docker base images have an ENTRYPOINT of /bin/sh or /bin/bash.

To override the ENTRYPOINT at runtime, we can use --entrypoint.

Whereas, the main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable or for executing an ad-hoc command in a container. It can omit the executable as well, in which case we must specify an ENTRYPOINT instruction as well specified with the JSON array format.

The thumb rule is that every Dockerfile should specify at least one of CMD or ENTRYPOINT commands.

**[How is Kubernetes related to Docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2883)**

Though Docker is a great tool for quick development and environment setup, it cannot be used in production directly. It needs a layer of orchestration for container management.

That’s where Kubernetes comes into the picture.

Kubernetes is an open source orchestration system for Docker containers. It manages containerized applications across multiple hosts and provides basic mechanisms for deployment, maintenance, and scaling of applications.

In a Kubernetes environment, one or more docker containers are run inside a ‘pod’. A pod is a basic unit that Kubernetes deals with and it represents one or more containers that should be controlled as a single "application".

**[What is Docker Swarm?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2884)**

Swarm is the native way provided by docker for implementing clusters for Docker containers. Basically, it allows a group of machines that are running Docker to join as a single cluster. Docker commands executed on a cluster are implemented on all the hosts by the swarm manager. Machines in a swarm can either be physical or virtual. After joining a swarm, they are referred to as nodes.

It also provides built-in load balancing and Service Discovery for the applications running as containers.

However, a more popular way of implementing clustering in production is achieved using Kubernetes. It is technically superior to Docker Swarm as it provides better health checking provisions at the pod level, better container scheduling and scaling.

**[Can you compare Chef with Docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2885)**

A Chef is a Configuration Management tool which is primarily used by admins and DevOps teams to manage the application environments, web-server configuration, databases, and load balancers. Instructions are created in the form of recipes, that defines the components in the infrastructure and how those components can be deployed, configured and managed. In a nutshell, it makes it automates the rolling out of changes to various environments or machines.

Whereas Docker is a way to package code into consistent units of work known as containers. These units of work can then be deployed to development, QA and production environments with far greater ease and consistency.

**[How does Docker play a role in Continuous Integration?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2886)**

Continuous Integration is a mechanism used to enable DevOps methodology in development process i.e. helps to obtain immediate feedback on the changes made in the code by executing unit and acceptance tests, as soon as the code changes are checked-in to the repository.

A typical process involves developers merging their changes to the branch as often as possible and they are validated by building, running and executing automated tests against the application.

Docker plays a key role in enabling CI, as it can seamlessly integrate with any popular CI tools like Jenkins or TeamCity. Also, several Gradle plugins like Palantir, trans mode are available for automating the docker build and publish images by creating respective Gradle tasks.

We can then define stages in Jenkins to perform tasks like building Docker images, starting containers using compose and executing Acceptance tests to validate the application changes.

#### Advanced

**[What is Docker architecture?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2867)**

Docker is having a  client-server architecture. The Docker client talks to the Docker daemon. Docker daemon helps in building, running, and distributing your Docker containers. The Docker client and daemon can co-exist on the same system or on different systems. This client-daemon connection is done via REST API.

**The Docker daemon**

The Docker daemon is also known as docker engine manages docker objects such as images, containers, networks, and volumes. Communication with other docker daemons also happens through this. Docker daemon is started by the command “dockerd”.Docker daemon talks with the kernel manages system calls for creation, running, destroying, etc of containers.

**The Docker client**

The Docker client is the utility we use when we use commands such as docker run. The Docker client sends these requests to Docker daemon, which does the rest. One Docker client can connect with more than one Docker daemon.

**Docker registries**

A Docker registry is where the Docker images are stored. Docker Hub is a public registry and docker searches for images on Docker Hub by default. Other registries are Docker Datacenter (DDC), which also includes the Docker Trusted Registry (DTR). Whenever docker pull or docker run commands are executed, the required images are pulled from the registry. We can also push the newly created docker images to Docker registry.

**[What is Docker Swarm?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2868)**

From Docker Engine version 1.12 Docker swarm mode was introduced that creates a cluster of one or more Docker Engines. A swarm can have more than one node which could be either a  physical or virtual machines that run docker engine.

Main Features are

* **Cluster management:**Docker Swarm could be created using Docker CLI  where application services could be deployed.No need of any other clustering tools.
* **Distributed design:** At deployment time, the Docker Engine handles any specialization at runtime. Both manager and worker nodes use the Docker Engine.
* **Declarative method:** A declarative method is used for defining the desired state of various services. For example, an application with a web front end service could be combined with message queueing services and a database backend.
* **Scaling:** Whenever we need to scale the swarm, one of the manager nodes automatically adapts by adding or removing tasks.
* **The desired state attained:**The swarm manager node checks for the actual state and your expressed desired state to bring the cluster back to the desired state. For example, if a service needs to run 20 replicas of a container, and each worker machine hosting 10 of them, the manager always creates new ones to replace the replicas that crashed.
* **Multi-host networking and service discovery:** The swarm manager assigns Ip addresses to the containers from the pool when it initializes or updates the application. Each service in the swarm has a unique DNS name and load balancer for the containers. Swarm helps the user to define how to distribute load balancing within service containers.

**[What is developer workflow of docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2869)**

The workflow starts with:

* **Attaining the docker image**

If the docker image is present in Docker Hub or that kind of Docker registries the images are selected from them. Developers can also create a great number of images if need be. Images have a layered architecture. Mostly a combination of a base layer and then a customised layer is used to create a docker image. There are several types of files that help to build images and they are kept in respective repositories. Some of the important files to consider to create docker images are docker files, docker-compose.yml files, etc. Multiple containers could be run together using docker-compose.yml files. Some of the types of repositories where the images are stored are

1. **Version Control** - This is used mainly used to store text-based files such as Dockerfiles, docker-compose.yml, and configuration files. Some of the version control systems are git, svn, Team Foundation Server, VSTS, and Clear Case.
2. **Repository Manager** - Large binary files such as that of Maven/Java, npm, NuGet, and RubyGems are stored in repository managers.Whereas smaller binary files could be saved to version control. Examples include Nexus, Artifactory, and Archiva.
3. **Package Repository** - Packaged applications for example like an operating system such as CentOS, Ubuntu, and Windows Server are stored in the package repository. Examples are yum, apt-get, and package management. They can also run several containers together described in a docker-compose.yml file and test the application.

* **Deploy on UCP(Docker Universal control plane)**

A CI/CD system runs unit tests, builds the applications, and create a  Docker image on the Docker Universal Control Plane (UCP). If the image passes all tests they are signed using Docker Content Trust and shipped to Docker Trusted Registry (DTR). The developer could also do testing on an integration environment on UCP in cases where the developer’s machine does not have access to all the resources to run the entire application.

* **Push Images**

Once the image is ready newer images created are pushed into Docker Trusted Registry. The developer can use the command

docker push < image name>

The developer account in DTR is a must have to do this action

* **Commit**

Once the application is tested on UCP files used to create the application, its images, and its configuration is committed to version control. The commit triggers could be set up to trigger the CI/CD workflow. The images could now be used to create containers.

**[What are Dockerfiles?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2870)**

Dockerfile is a text file that has instructions to build a Docker image. All commands in dockerfile could also be used from the command line to build images.

docker build command creates an image from the dockerfile and its contexts. Contexts are a set of files at a PATH/URL. URL could be any repository location of any version control system. PATH could also be a location in your local system. The following command shows building docker image from the contexts in the current directory and send the context to the docker daemon

$ docker build .

Sending build context to Docker daemon  6.51 MB

...

Root directory or path is not preferred as context path because the entire contents of your hard drive will be transferred to the Docker daemon.

Docker daemon performs a validation of the dockerfile before running instructions in dockerfile. Docker daemon takes on the instructions one by one creating an updated image each time.

Sample Dockerfile :

FROM ubuntu:16.04

COPY . /app

RUN make /app

CMD python /app/app.py

Each instruction in a dockerfile  creates one read-only layer:

* FROM creates a layer from the ubuntu:16.04 Docker image as the base image.
* COPY adds files or contexts from your Docker client’s current directory.
* RUN builds your application. Here we use make.
* CMD specifies the command to be run inside the container.

**[What are docker-compose.yml files?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2871)**

Docker compose is a tool that helps to run multi-container docker applications. Compose uses docker-compose.yml which is a YAML file to configure application’s services. Compose workflow is a three-step process. First is to create a dockerfile to build the image, secondly define the services to be run on a container in the docker-compose.yml file and thirdly docker-compose up command starts up your entire applications. dockerfiles and docker-compose.yml files are similar but the main difference is that docker compose file manages multi-container architecture rather than a single container.

Sample docker-compose.yml file looks like:

version: "7"

services:

  testservice:

    # replace username/repo:tag with your name and image details

    image: <image name>

    deploy:

      replicas: 10

      resources:

        limits:

          cpus: "0.2"

          memory: 70M

      restart\_policy:

        condition: on-failure

    ports:

      - "9080:80"

    networks:

      - webnet

This docker-compose.yml is reflected upon below:

* Pull the image from Docker registry .
* Run 10 instances of the pulled image as a service called testservice, with setting limits on each one as, 20% of a single core of CPU time and 70 MB of RAM.
* Immediately restart containers on failure.
* Map port 9080 on the host system to test services port 80.
* test services containers share port 80 through a load-balanced network called webnet.

**[What is Docker volume mounting?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-2872)**

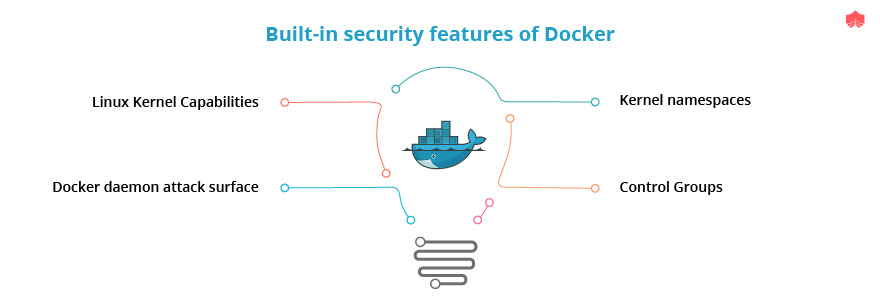
Data is not persistent inside a container. One of the most used methods is Volume mounting.  Another method for persisting data is to include data in the writable layer but that layer is tightly coupled with the host machine where the container is running, thus the data couldn’t be managed easily. Also writing into the writable layer is complex and needs to use a driver. To retain data, the best-preferred way is to use a Docker volume mount where it mounts another directory into your container. This directory is on the host which could be seen within the container. The main advantage of volume mounting is

* Easy to manage using Docker CLI commands.
* We have support for both Windows and Linux systems
* Multiple containers can share the volumes.
* Volumes are easier to back up
* Volume drivers are present to encrypt volume contents and to store on cloud systems etc
* The container can pre-populate the new volumes with its contents
* Volumes don't increase the size of the containers.

Following commands could be used to manage volumes:

* docker volume create <name> to create the volume
* docker volume ls to list the volumes available to you
* docker volume inspect <name> to get more details about the volume like path, drivers etc.
* docker volume rm <name> to remove the container.
* docker volume prune to remove unused volumes

**[What are the basic built-in security features of Docker?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3105)**

****

* **Kernel namespaces**

Namespaces define the context in which names are defined whether it be variable names or function names. In other words, namespace defines the scope of the names. For eg: namespaces are like surnames. Suppose we have two people with the same name “John “ their surname differentiates them.

Each container in docker creates a set of namespaces specific to the container. Hence is the first and a great method of security between containers.

* **Control Groups**

Control groups facilitate resource accounting and limiting. Control Groups doesn’t allow a container to exhaust the host system’s CPU, memory, disk I/O, etc. It also doesn’t allow data and processes of container to be accessed by another container.

* **Docker daemon attack surface**

When a “docker run “ command is performed docker client speaks to docker daemon who manages the images and containers. Docker daemon needs root privileges. Extra precaution must be taken to give access only to trusted users to control docker daemon. A  container could even be started from the root directory on your host and the container can alter your host filesystem without any restriction.

* **Linux Kernel Capabilities**

Containers could be started with a reduced set of capabilities. This would mean that “root” within a container has fewer privileges than the real “root”. This, in turn, reduces the damage by an intruder with root privileges.

**[What is the swarm mode Public Key Infrastructure?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3106)**

Built-in Public Key Infrastructure System helps to secure the deployment of container orchestration systems. Transport Layer Security (TLS) is used in Public Key Infrastructure to communicate with other nodes in a swarm.

When a swarm is initialized with “docker swarm init” command in a docker host, root Certificate Authority (CA) with a key pair is created. This is for securing nodes that join the particular swarm.

“--external-ca”  flag is used with docker swarm init command to use external root CA.

Manager node generates worker token and manager token. Each token has the digest of the root CA certificate and a randomly generated secret. When a new node joins the docker swarm with the worker token the node uses the digesting part to verify the root CA from the manager node. While the leader node uses the secret to approve the new joining node. Manager node issues a certificate to the joining node with a randomly generated node ID

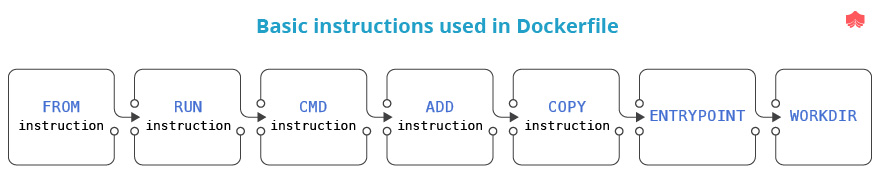
By default, swarm performs the renewal of the certificate every three months but it can be modified with the command

“docker swarm update --cert-expiry <TIME PERIOD> “

In case if the leader-manager node is down we can rotate the root CA within the swarm so that no nodes trust the certificate signed by old root CA. This can be done by the command “docker swarm ca --rotate”.

This command thus creates a cross signed certificate telling the nodes that still trusted  old CA to start verification against new root CA

**[What are the basic instructions used in Dockerfile and how they differ from each other?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3107)**

****

* **FROM instruction**

Usage :

FROM <image> [AS <name>]

Or

FROM <image>[:<tag>] [AS <name>]

Or

FROM <image>[@<digest>] [AS <name>]

FROM helps to set the base image for the following instructions in the Dockerfile. The base image could be one pulled from the remote public repository or a local image built by the user. FROM can appear multiple times in Dockerfile and each time it appears it clears any stage created from previous instructions. Hence usually an AS <NAME> is added to the FROM instruction to identify the last image created just before the FROM instruction.

* **RUN instruction**

Usage:

* RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on Linux or cmd /S /C on Windows)
* RUN ["executable", "param1", "param2"] (exec form)

RUN instruction creates a new layer on the current image which is used for the next step in Dockerfile.RUN cache is valid for the next build provided “docker build” command is not run with “--no-cache” flag.

RUN in the executable form doesn’t do shell processing like variable substitution.

* **CMD instruction**

Usage:

* CMD ["executable","param1","param2"] (exec form, this is the preferred form)
* CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
* CMD command param1 param2 (shell form)

CMD provides defaults for executing container. There could only be one CMD instruction but if there are many, only the last one has the effect. If CMD is providing default arguments for the ENTRYPOINT instruction, then the CMD and ENTRYPOINT instructions should be specified with the JSON array format.

* **ADD instruction**

Usage:

* ADD [--chown=<user>:<group>] <src>... <dest>
* ADD [--chown=<user>:<group>] ["<src>",... "<dest>"] (this form is required for paths containing whitespace)

The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

* **COPY instruction**

Usage:

* COPY [--chown=<user>:<group>] <src>... <dest>
* COPY [--chown=<user>:<group>] ["<src>",... "<dest>"] (this form is required for paths containing whitespace)

The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.Main difference between COPY and ADD is that ADD supports the source to be a URL or tar file.

* **ENTRYPOINT**

Usage:

* ENTRYPOINT ["executable", "param1", "param2"] (exec form, preferred)
* ENTRYPOINT command param1 param2 (shell form)

An ENTRYPOINT allows you to configure a container that will run as an executable. As a best practice ENTRYPOINT should be defined when using the container as an executable and CMD should be used as a way of defining default arguments for an ENTRYPOINT command.CMD could be overridden while running a container with arguments.

* **WORKDIR**

Usage:

WORKDIR /path/to/workdir

The WORKDIR instruction sets the working directory for RUN, CMD, ENTRYPOINT, COPY and ADD instructions in the Dockerfile.

**[What is Docker context?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3108)**

Docker builds command builds the images from Dockerfiles and “context”.A build’s context is the set of files located in the specified PATH or URL. The PATH is a directory on your local host filesystem while the URL could be a Git repository location, pre-packaged tarball contexts, and plain text files. COPY instruction could be used to reference a file in the context.

In the case of Git repositories, first the repository is pulled into a temporary directory on your local host then after it’s successful, the directory is sent to the Docker daemon as the context.

For eg:

docker build https://github.com/dockersample/test.git#container:docker

uses a directory called docker in the branch container as the context.

In the tarball context.For eg:

docker build http://server/context.tar.gz

sends the URL itself to docker daemon.Tar file is downloaded on the host the Docker daemon is running on, which need not be the same host from which the build command is being issued.

In text files context a single Dockerfile could be passed in the URL or pipe the file in via STDIN. For eg: To pipe a Dockerfile from STDIN we use the instruction :

docker build - < Dockerfile

**[How to work with docker build cache?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3109)**

While building a docker image Docker steps through the instructions in the Dockerfile adding layers to the base image. Every time Docker encounters a new instruction it checks for an existing image in its cache which it can reuse locally. Thus helping to build the final docker image quickly. If you don't want to use cache “ --no-cache=true” option with docker build command is used. Basic rules about cache are as follows:

* After the parent image is in the cache, the next instruction is compared against all child images derived from that parent image to see if anyone of them was built using the next instruction. If not, the Docker builds the image at this step once again and the cache is invalidated.
* For the ADD and COPY instructions, the contents of the file(s) in the image are examined and a checksum is calculated for each file. Last-modified and last-accessed times of the file(s) are not considered in these checksums. These checksums are compared with checksums of the existing images. If there are changes cache is invalidated.
* Unlike the ADD and COPY commands, in the case of a RUN  command, the files updated in the container are not examined to determine if a cache hit exists. In that case, just the command string itself is used to find a match.

Once the cache is invalidated, all the following Dockerfile commands generate new images.

**[What are the networks used in Docker Swarm?](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3110)**

When a Docker Swarm is created or a docker host joins the swarm two networks are created.

* an overlay network called ingress, which manages control and data traffic related to swarm services. This is the default network unless the user specifies other user-defined overlay networks. User-defined overlay networks could be created by using the command

“docker network create”

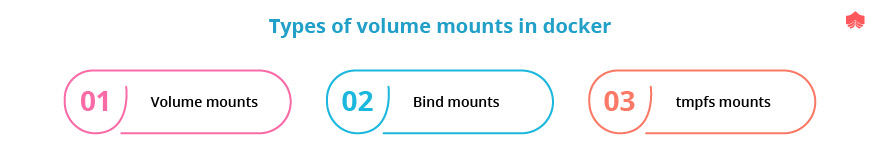
ingress network facilitates load balancing among a service’s nodes.

To encrypt the application data traffic on a given overlay network, use the --opt encrypted flag on docker network create.

To attach a service to an existing overlay network, use the --network flag to docker service create, or the --network-add flag to docker service update command. Service containers connected to an overlay network can communicate with each other through it.

* a bridge network called docker\_gwbridge, which connects the Docker daemon to the other docker daemons in a swarm. docker\_gwbridge connects the overlay networks (including the ingress network) to an individual Docker daemon’s physical network. It is a virtual bridge that exists in the kernel of the docker host. For customising the bridge network we have to do that before docker host joins the swarm or by temporarily removing the host from the swarm.

**[Explain different types of volume mounts in docker.](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3111)**



There are three mount types available in Docker

* Volume mounts are the best way to persist data in Docker. Data are stored in a part of the host filesystem which is managed by Docker containers. (/var/lib/docker/volumes/ on Linux).Main advantages of using volume mounts are
  + Easy to use and backup
  + Docker CLI commands or the Docker API can be used to manage volume mounts.
  + Linux and Windows containers support volume mounts.
  + Volumes could be shared securely among multiple containers.
  + New volumes can use pre-populated content by a container.

Starting with Docker 17.06, -v or --volume flag and --mount flag could be used for docker swarm services and standalone containers. To create a docker volume. For eg:

“docker volume create my-vol” creates new volume  “my-vol”.

We can inspect a volume with the command “docker volume inspect”

For eg:

“docker volume inspect my-vol “ gives the output

[

{

    "Driver": "local",

    "Labels": {},

    "Mountpoint": "/var/lib/docker/volumes/my-vol/\_data",

    "Name": "my-vol",

    "Options": {},

    "Scope": "local"

}

]

If we need to start a container with “my-vol”

* With -v flag

“docker run -d  --name devtest -v my-vol:/app nginx:latest”.

Here nginx images with the latest tag are executed with using volume mount “my-vol”

* With --mount flag

“docker run -d --name devtest --mount \ source=my-vol,target=/app nginx:latest”

* Bind mounts may be stored anywhere on the host system. A file or directory on the host machine is mounted into a container unlike volume mounts where a new directory is created within Docker’s storage directory on the host machine, and Docker manages that directory’s contents. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
* tmpfs mounts are stored in the host system’s memory only and are never written to the host system’s file system. When the container stops, the tmpfs mount is removed, and files won’t persist.

**[Explain the purpose of the .dockerignore file.](https://www.knowledgehut.com/interview-questions/docker" \l "collapse-beginner-3112)**

Whenever a “docker builds” command is executed we see a line that tells uploading context. This refers to the creation of a .tar file by including all files in the directory where the Dockerfile is present and uploading them to docker daemon. Consider if we are putting Dockerfile in home directory entire files in your home and in all subdirectories would be included in the creation of a .tar file. Thus before updating the context docker daemon checks for the .dockerignore file. All files that match the data in the .dockerignore file would be neglected. Hence sensitive information is not sent to the Docker daemon.

sample .dockerignore file looks like this:

# comment

\*/temp\*

\*/\*/temp\*

Temp?

This .dockerignore file ignores all files and directories whose names start with temp in any immediate subdirectory of the root or from any subdirectory that is two levels below the root. It also excludes files and directories in the root directory whose names are a one-character extension of temp. Everything that starts with # is ignored.

**JENKINS INTERVIEW**

**BASIC**

## DESCRIPTION

Jenkins, a very important tool in DevOps that automates the rest of the process with the help of plugins once the changes are made in a source code repository (SCR). If you are applying for any DevOps role or any Cloud-native practitioner role, you can encounter many Jenkins interview questions.   
  
Following are the most frequently asked interview questions and answers on Jenkins. These interview questions on Jenkins can be useful for freshers and experienced to clear the interview.   
When any organization seeking to fill the vacancies for the DevOps roles, they mostly look for the following set of skills in an individual and these are:  
  
An individual should be fluent in web languages like Ruby, Python, PHP or Java.  
An individual should be aware of the various infrastructure automation tools like Chef, Puppet, Ansible, SaltStack or Windows PowerShell DSC.  
  
Interpersonal skills that help you communicate and collaborate across teams and roles.  
This skills-set will make you eligible for the interview. Jenkins basic interview questions and answers mentioned here will help you to prepare for the interview. Here is a list of the top Jenkins interview questions and answers.

#### [How to fix a broken build for your project in Jenkins and how to make sure project build doesn't break in Jenkins at all?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-546)

The user needs to open the console output for the build and will try to see if any file changes that were missed during building the project. If there are no issues on that then a better approach would be clean and update his local workspace to replicate the problem on their local machine and will try to solve it.

To make sure Jenkins build is not broken at all we need to make sure that we perform a successful clean install on the local machine with all unit tests. Then make sure that all code changes are checked in without any issues. Then synchronize with a repository to make sure that all required config and changes and any differences are checked into the repository.

#### [What are the different ways in which build can be scheduled in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-547)

The below are the ways of scheduling build in Jenkins

* Builds can be triggered by source code management commits.
* Builds can be triggered sequentially after completion of other builds.
* Can be scheduled to run at a specified time using the CRON jobs
* Manual Build Requests.

#### [Elaborate on how to move or copy Jenkins from one server to another?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-548)

Please follow the below steps,

* Slide a job from one installation of [Jenkins](https://www.mytectra.com/continuous-integration-with-jenkins-training.html) to another by copying the related job directory
* Make a copy of an already existing job by making a clone of a job directory by a different name
* Renaming an existing job by renaming a directory.

**[What is the use of the Role Based strategy plugin?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-551)**

The role-based strategy plugin allows us to create three different types of roles as describes below,

* **Global Roles:** Some of the roles such as admin, job creator, anonymous can be created while selecting this option. The user can allow setting Overall, slave, job, and View and SCM permissions on a global basis.
* **Project roles:** This job is basically to allow the creation of Job and Run permissions on a project basis.
* **Slave roles:** This job is only to set node-related permissions.

#### [What do you mean by build pipeline in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-555)

Creating a chain of jobs in Jenkins is the process of automatically starting the sequential job after one job is executed successfully. This approach lets the user build multi-step build pipelines or trigger the rebuild of a project if one of the Project dependencies is updated.

#### [What do you mean by a Jenkins File and what are its advantages?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-556)

A Jenkins file is a text file that contains the definition of a Jenkins Pipeline and it is generally checked into source control.

* Audit trail for the Pipeline can be monitored
* It serves as a single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

#### [What the Source Code Management tools does Jenkins support?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-560)

Jenkins supports a wide range of Source Code Management tools and few of them are mentioned below,

* AccuRev
* CVS
* Subversion
* Git
* Mercurial
* Perforce
* Clearcase
* RTC

#### [What is the use of setting environment directive in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-561)

The environment directive specifies a sequence of key-value pairs which will be defined as environment variables for the all steps, or stage-specific steps, depending on where the environment directive is located within the Pipeline. This directive supports a special helper method **credentials()** which can be used to access pre-defined Credentials by their identifier in the Jenkins environment.

#### [What are the Stages in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-565)

A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline. Stages contain a sequence of one or more stage directives, the stages section is where the bulk of the "work" described by a Pipeline will be located. Minimum, it is recommended that stages contain at least one stage directive for each discrete part of the continuous delivery process, such as Build, Test, and Deploy.

**[What is the programming language that Jenkins builds on?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-567)**

Jenkins is an open source automation server written in Java. As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.

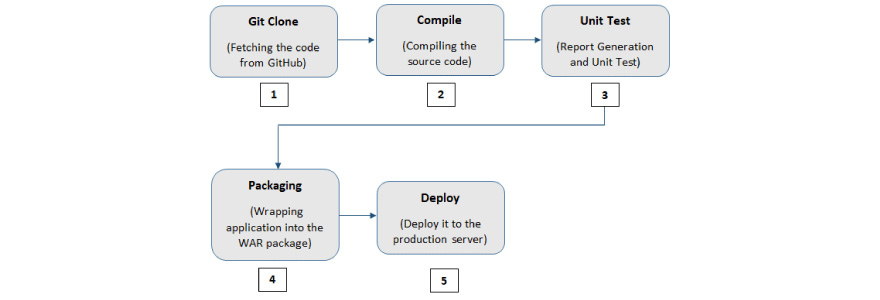
**Impress Employers with your Jenkins knowledge**

So, this is all about the**best interview questions and answers on Jenkins.** I hope these Jenkins interview questions help you crack your interview.

Prepare yourself with the **best interview questions and answers on Jenkins** to land your dream job!

**[How can you define continuous delivery workflow?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3036)**

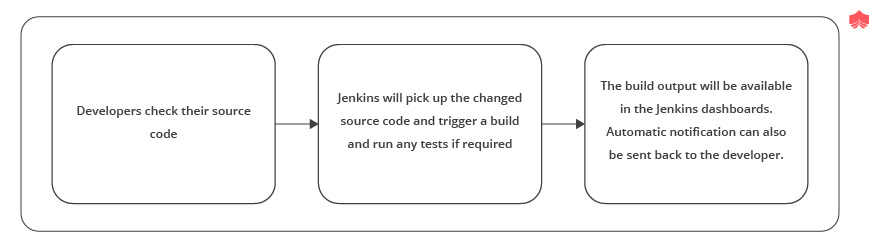
Continuous delivery (CD ) is a product design methodology in which team software in short cycles, guaranteeing that the product can be dependably released whenever and, when releasing the product, doing as such manually. It goes for building, testing, and releasing programming with more prominent speed and recurrence. The methodology lessens the cost, time, and risk of conveying changes by considering progressively gradual updates to applications underway. A direct and repeatable arrangement procedure is significant for continuous delivery. CD stands out from continuous deployment, a comparable methodology where software is likewise delivered in short cycles yet through automatic arrangements as opposed to manual ones. The flowchart beneath demonstrates the Continuous Delivery Workflow. Hope it will be much easier to understand with visuals.



**INTEMEDIATE**

**[Why do we use Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3022)**

Jenkins is an open source tool written in Java with modules worked for Continuous Integration reason. Jenkins is utilized to construct and test your product extends consistently making it simpler for engineers to coordinate changes to the undertaking, and making it simpler for clients to acquire a new form. It additionally enables you to persistently convey your product by coordinating with an enormous number of testing and arrangement innovations. With Jenkins, associations can quicken the product improvement process through computerization. Jenkins incorporates improvement life-cycle procedures of assorted types, including manufacture, record, test, bundle, organize, convey, static examination and considerably more. Jenkins facilitate Continuous Integration with the help of various plug-ins. Plug-in allows the combination of Various DevOps stages. If we need a specific use case, we have to introduce the plug-in for those tools. For example Git, Maven 2, Amazon EC2, HTML distributer and so on.

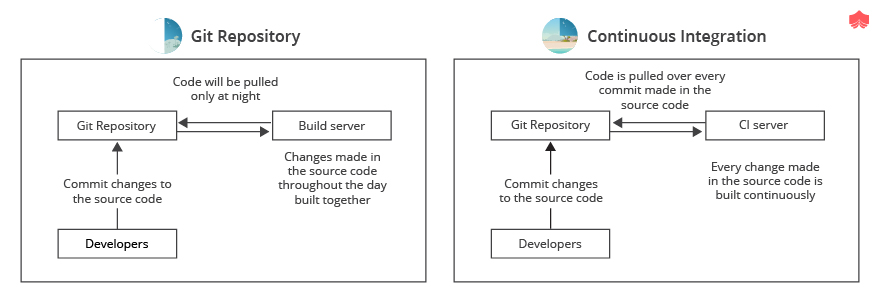
**The simple workflow of Jenkins**

**Points of interest of Jenkins include:**

* It is an open source instrument with incredible network support.
* It is easy to install and configure
* It has 1000+ modules to facilitate your work. On the off chance that a module does not exist, you can code it and offer with the network.
* It is free of cost.
* It is worked with Java and henceforth, it is convenient to all the real stage

**[What is meant by continuous integration in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3023)**

Consistent integration is a procedure where all development work is incorporated as ahead of schedule as could reasonably be expected. The subsequent artifacts are consequently made and tried. This procedure permits to recognize mistakes as ahead of schedule as could reasonably be expected. Jenkins is a well known open source device to perform constant integration and build. Prior to CI, Nightly build was famous where code was getting pulled distinctly around evening time however CI guaranteed the early discovery of deformities.



**Continuous Integration has numerous advantages:**

* Bid a fond farewell to long and tense combinations
* Increment perceivability empowering more noteworthy correspondence
* Catch issues early and halt them from developing in any way
* Invest less energy investigating and additional time including highlights
* Fabricate a strong establishment
* Quit holding on to see whether your code's getting down to business
* Diminish incorporation issues enabling you to convey programming all the more quickly

**[Name some popular  CI Tools.](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3024)**

**Here is a rundown of the main 8 Continuous Integration tools:**

* **Jenkins**: Jenkins is an open-source CI tool written in Java.
* **TeamCity**: TeamCity is the full-grown CI server, originating from the labs of the JetBrains organization. Incredible arrangement by and large, however because of its intricacy and value, more qualified for big business needs.
* **Travis CI**: A Mature arrangement that offers both facilitated and On-premises variations, cherished and utilized by numerous groups, very much reported.
* **Go CD**: Go is the most up to date Cruise Control manifestation from the ThoughtWorks organization. Barring the business bolster that ThoughtWorks offers, Go is for nothing out of pocket. It is accessible for Windows, Mac, and different Linux circulations.
* **Bamboo:**It is Atlassian offerings.Great On-premises CI instrument that initially offered Cloud arrangement as well. Bitbucket Pipelines supplanted the cloud arrangement. A Pipeline is a cutting edge and quick cloud CI instrument incorporated into Bitbucket. Has a free preliminary for 30 days, and paid plans after that.
* **GitLab CI:**GitLab CI is a fundamental piece of the open-source Rails venture GitLab, which was exposed by the organization GitLab inc
* **CircleCI**: Another cloud elective that originates from the organization with a similar name. CircleCI as of now just backings GitHub and the rundown of upheld dialects incorporates Java, Ruby/Rails, Python, Node.js, PHP, Haskell, and Scala.
* **Codeship:**Codeship comes in two unique variants: Basic and Pro. The fundamental form offers out-of-the-container Continuous Integration administration yet doesn't have docker backing and its principal intention is to construct applications with normal work processes through the UI. Star form offers greater adaptability and docker support.

**[Which SCM tools Jenkins supports?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3025)**

SCM stands for Source Code Management is an integral part of any development project in the current IT world. It is very critical to manage source code in an efficient way. There are several SCM tools which are available. Some advantages of using SCM:

1. **Backups:** Always accept that your PC is going to be sucked into a monstrosity dark opening at any second and work to limit the misfortune from that. Source control enables you to effortlessly push finished work to a remote host so little work is lost with a solitary PC.
2. **Record of work:**On numerous ventures, there's intermittently when you need to return the code to a past state to perceive how something was done, if a bug was available, or work out why the present code is broken. Source control makes this simple.
3. **Arrangement:** Having source control enables you to robotize fabricates and organizations. No one ought to ftp records around any longer as that is inclined to human blunder.
4. **Forming:** It's anything but difficult to swap between renditions of your code, giving you a chance to switch between fixing a little bug underway and thoroughly reworking your key usefulness.
5. **Simple venture setup:** As a task's source code can likewise incorporate arrangement contents, introduce records, and so forth… at that point setting up an undertaking can be as simple as cloning the storehouse and running a few contents to set up a domain that is indistinguishable from every single other designer.

Jenkins underpins AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase and RTC. For every one of them exists a module and as you likely definitely know, Jenkins isn't constrained to just that rundown, anybody can make an SCM module for different choices on the off chance that they need to.

**Here is the connection to the modules:**

* **ClearCase**: <https://wiki.jenkins-ci.org/display/JENKINS/ClearCase+Plugin>
* **RTC:** <https://wiki.jenkins-ci.org/display/JENKINS/Team+Concert+Plugin>
* **Git:** <https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin>
* **Mercurial:** <https://wiki.jenkins-ci.org/display/JENKINS/Mercurial+Plugin>
* **Perforce:** <https://wiki.jenkins-ci.org/display/JENKINS/Perforce+Plugin>
* **AccuRev:** <https://wiki.jenkins-ci.org/display/JENKINS/AccuRev+Plugin>
* **CVS:** <https://wiki.jenkins-ci.org/display/JENKINS/CVS+Plugin>
* **Subversion:** <https://wiki.jenkins-ci.org/display/JENKINS/Subversion+Plugin>

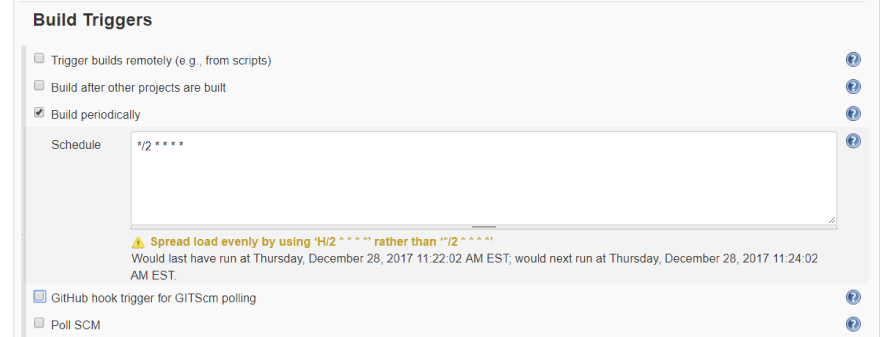
**[How do you schedule a build in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3026)**

In Jenkins, under the job configuration arrangement, we can characterize different build triggers. Basic discover the 'Build Triggers' segment and check the ' Build Periodically' checkbox. With the occasionally build you can plan the construct definition by the date or day of the week and an opportunity to execute the assembly. The configuration of the 'Schedule' textbox is as per the following:

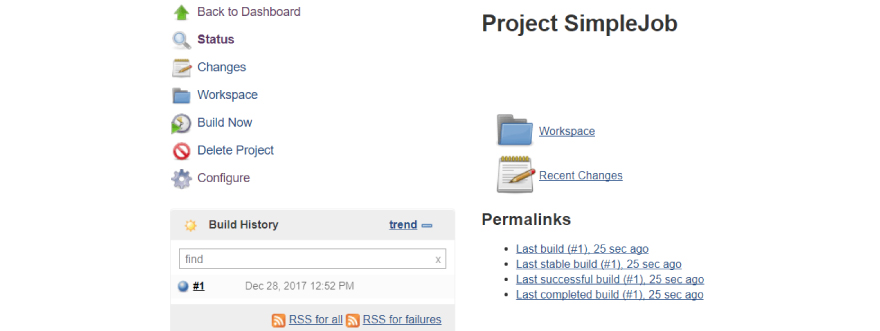
MINUTE (0-59), HOUR (0-23), DAY (1-31), MONTH (1-12), DAY OF THE WEEK (0-7)

In the activity design page, we should look down directly to the Build Triggers area. Since we expect to make a direct activity, we should choose the checkbox stamped Build intermittently. When we select this checkbox, a Text Box is shown with the Schedule name. We need to offer some benefit in a Cron-consistent arrangement. There's broad data accessible on the page in the event that we click the question mark next to the textbox.

Let’s type \*/2 \* \* \* \* here, which represents an interval of two minutes:l



After selecting out of the content box, we can see data directly underneath the container. It enlightens us regarding when will the activity keep running straightaway. How about we spare the activity – in around two minutes, we should see the status of the principal execution of the activity:



Since we’ve configured the job to run every two minutes, we should see multiple build numbers when we go back to the job dashboard after waiting for some time.

**[What Are The Advantages Of Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3027)**

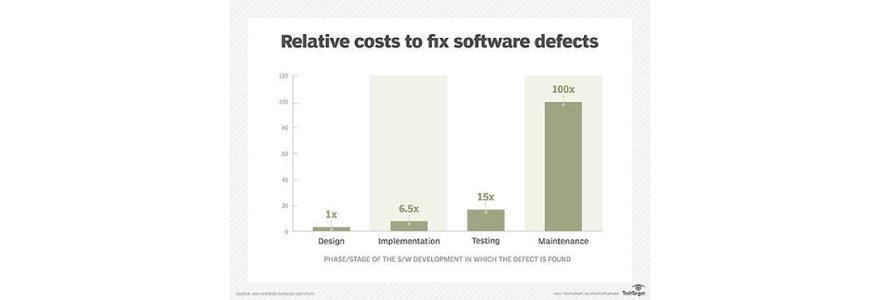
In the present DevOps world, continuous delivery and deployment are basic to conveying fantastic programming item quicker than any time in recent memory. Jenkins is an open-source persistent incorporation server written in Java. It is by a wide margin the most generally utilized instrument for overseeing constant reconciliation builds and delivery pipelines. It helps engineers in structure and testing programming continuously. It expands the size of mechanization and is rapidly picking up ubiquity in DevOps circles. One of the key points of interest of Jenkins is that it requires little upkeep and has worked in a GUI apparatus for simple updates. Jenkins additionally gives tweaked arrangement as there are more than 400 modules to help to build and to test essentially any venture. By executing the correct setup for you, you get practically prompt input. You will dependably know whether the manufacturer broke. You will become more acquainted with what the explanation behind occupation come up short was and you can likewise become acquainted with how you can return it.

**Advantages of Jenkins include:**

1. Bugs tracking are simple at a beginning period being developed condition.
2. Gives an enormous number of module support.
3. Iterative improvement to the code.
4. Build failures are stored at incorporation arrange.
5. For each code commit changes a programmed assemble report warning produces.
6. To advise engineers about build success /failures, it is coordinated with the LDAP mail server.
7. Accomplishes continuous integration agile development and test driven improvement.
8. With straightforward advances, the maven release project is automated.

**[What does it mean to shift left in DevOps?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3028)**

If you somehow managed to diagram the conventional programming development life cycle on a bit of paper, the left half of the chart would almost certainly incorporate tasks, for example, design and development, while the right side would probably incorporate client feedback, stress testing, and production staging. To move left in DevOps infers a longing to take a significant number of those undertakings that regularly occur close to the finish of the application advancement procedure and move them into prior stages. Now and again, this may intend to join static code investigation schedules in each build. Another approach to play out a DevOps move left is to make production-ready artifacts toward the finish of each Agile sprint with the goal that clients and partners can get gradual reports on how improvement is advancing. Proper DevOps means moving left however much as could be expected.



Shifting left testing in software production allows developers to catch and fix issues earlier.

**[What are the software prerequisites that must be met before Jenkins is installed?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3029)**

**Prerequisites:**Before you continue to introduce Jenkins in your windows framework, there are a few requirements for Jenkins to introduce Jenkins in your PC.

**Hardware requirements:**

* You need at least 256 MB of RAM in your PC or workstation to introduce Jenkins. You need in any event 1 GB of space in your hard drive for Jenkins.

**Software Requirements:**

* Since Jenkins keeps running on Java, you need either Java Development Kit (JDK) or Java Runtime Environment (JRE).

**Release Types:**Jenkins releases two sorts of adaptations dependent on enterprise needs. Long-term support release, Weekly release

**Long-term support release (LTS):**Long-term support releases are accessible at regular intervals. They are steady and are generally tried. This release is expected for end clients.

**Weekly release:**Weekly release are made accessible consistently by fixing bugs in its prior rendition. These releases are expected towards module engineers. We will utilize the LTS release however the procedure continues as before for Weekly release.

**[How can you move or copy Jenkins from one server to another?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3030)**

Move a job starting with one installation of Jenkins then onto the next by just duplicating the relating work directory. Make a duplicate of a current job by making a clone of a job directory by an alternate name. Rename a current job by renaming a directory. Note that on the off chance that you change a job name you should change whatever other activity that endeavors to call the renamed job. Those tasks should be possible notwithstanding when Jenkins is running. For changes like these to produce results, you need to click "reload config" to compel Jenkins to reload design from the disk.

**Follow these steps to move or copy Jenkins from one server to another:**

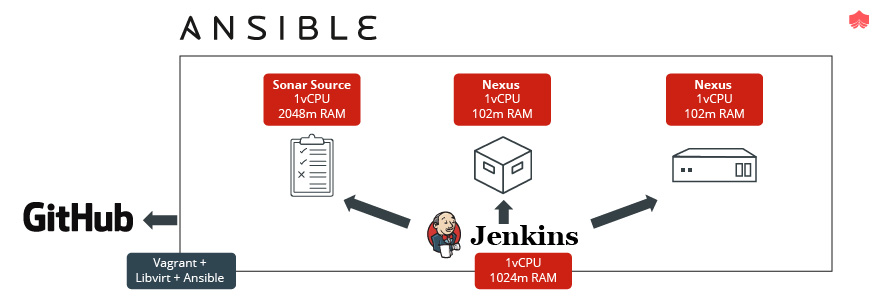
* Copy the related job directory and move a job from one installation of Jenkins to another.
* Make a copy of an already existing job by making a clone of a job directory by a different name.
* Renaming an existing job by renaming a directory.

We can also try one of the plug-ins as well as having Job export options like Job Importer Plug-in. Jenkins CLI can be also used if we have less number of jobs but usually at the enterprise level, we have a large number of jobs so not widely used.

**[What is the use of Ansible in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3031)**

Ansible is an incredible asset for IT automation and can be utilized in a CI/CD procedure to arrangement the target environment and to an application on it. The most effective method to utilize Ansible for environment provisioning and application deployment in a Continuous Integration/Continuous Delivery (CI/CD) process utilizing a Jenkins Pipeline. Jenkins is a notable tool for automation CI/CD. Shell scripts are regularly utilized for provisioning environments or to deploy applications amid the pipeline stream. Despite the fact that this could work, it is bulky to keep up and reuse contents over the long haul.

The motivation behind utilizing Ansible in the pipeline stream is to reuse jobs and Playbooks for provisioning, leaving Jenkins just as a procedure orchestrator rather than a shell script agent.



**The above  represents the accompanying engineering components:**

Github is the place where our project is facilitated and where Jenkins will survey for changes to begin the pipeline stream.

SonarSource is our source code analysis server. On the off chance that anything turns out badly amid the examination (for example insufficient unit tests), the stream has interfered. This progression is imperative to ensure the source code quality record.

Nexus is the artifact archive. After an effective accumulation, unit tests, and quality examinations, the binaries are transferred into it. Later those binaries will be downloaded by Ansible amid the application deployment.

The Ansible Playbook, which is a YAML record coordinated in the application source code, conveys the Spring Boot App onto a CentOS machine.

Jenkins is our CI/CD process orchestrator. It is dependable to assemble every one of the pieces, bringing about the application effectively deployed in the target machine.

**[Name four important KPI in Dev Ops.](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3032)**

Individuals utilizing DevOps practices depend on a few key performance indicators(KPIs) to pass judgment on the achievement of their DevOps endeavors:

**Deployment frequency**: The capacity to make code changes rapidly and effectively is a key upper hand for any organization that necessities to convey new highlights rapidly to clients, and react to their feedback.

**Speed of deployment**: Frequent code deployment depends in enormous part on having the option to move rapidly from committed code to that code running effectively in the production condition.

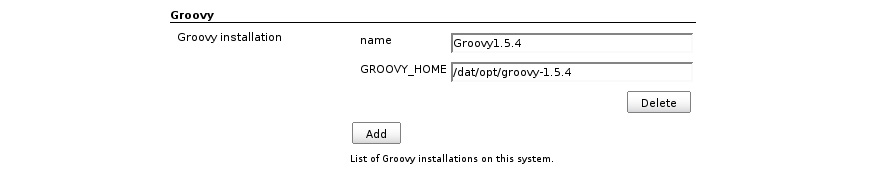
**Failure Rate**: It is extraordinary to send all the more regularly and rapidly, however on the off chance that changes bomb similarly as every now and again, you've picked up nothing. Failed deployments can bring services down, bringing about lost income and baffled clients. DevOps practices can have a major effect on the failure rate.

**Time to Recovery**: At the point when service goes out, the capacity to recuperate rapidly can have an immense effect on business results. It's not astonishing, at that point, that enormous web organizations like Google, Etsy, Netflix, and Amazon push the envelope in their endeavors to improve time to recuperation, normally breaking their applications and foundation to find -   arrangement against - anything that can turn out badly.

**[What is Groovy in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3033)**

Groovy is the default scripting language that is being utilized being developed from JMeter Version 3.1. Presently Apache Groovy is the dynamic object-oriented programming language that is utilized as a scripting language for the Java stage. Apache Groovy accompanies some valuable highlights, for example, Java Compatibility and Development Support. The groovy module adds the capacity to straightforwardly execute Groovy code.

To arrange accessible Groovy installation on your framework, go to Jenkins setup page, discover area 'Groovy' and fill the structure as appeared.



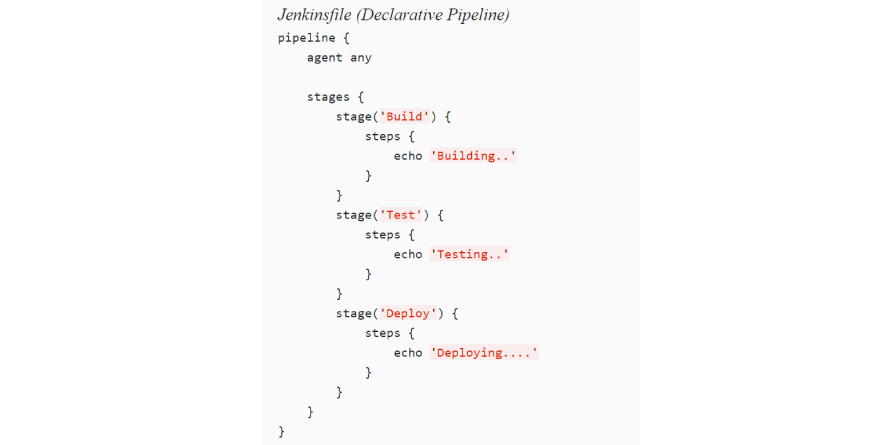
On the off chance that you don't arrange any Groovy establishment and select (Default) choice in a job, the module will fall back into considering only the groove order, accepting you have awesome paired on the default way on a given machine.

**[What is Jenkins file?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3034)**

For defining a Pipeline in SCM, a Jenkins file is a text file that contains the meaning of a Jenkins Pipeline and is registered with source control. Consider the accompanying Pipeline which executes an essential three-organize continuous delivery pipeline. Making a Jenkins file, which is registered with source control, gives various prompt advantages:

1. Code review/emphasis on the Pipeline
2. Audit trail for the Pipeline
3. Single wellspring of truth for the Pipeline, which can be seen and altered by numerous individuals from the undertaking.

Please find below Pipeline which implements a basic three-stage continuous delivery pipeline:

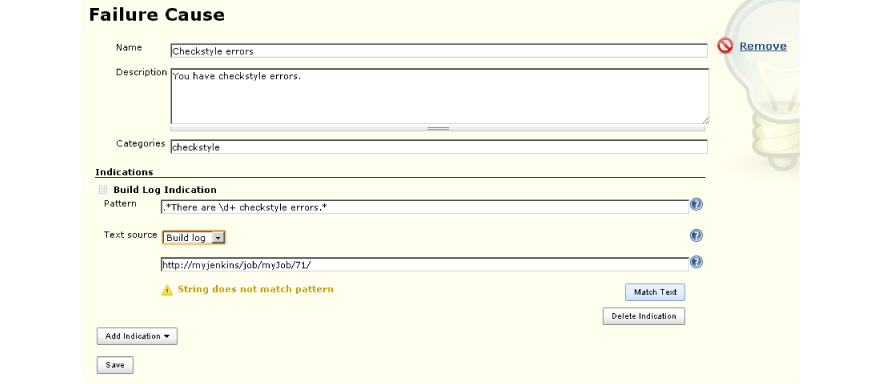


**[What do you do to make sure that your project build doesn't break in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3035)**

We ensure that we perform effective clean introduce on our local machine with all unit tests. At that point, we ensure that we check in all code changes. We do complete a Synchronize with the archive to ensure that all required config and POM changes and any differences are registered with the repository. You have to pursue the underneath referenced strides to ensure that the Project build does not break:

* Perfect and successful installation of Jenkins on your local machine with all unit tests.
* All code changes are reflected effectively.
* Checking for repository synchronization to ensure that every one of the distinctions and changes identified with config and different settings is spared in the repository.

There is a build failure analyzer plug-in which tells us a reason for build failure in case the build gets failed even after following the above steps. The module accompanies an empty information base of failure causes. Populating this learning base is finished by utilizing the connection "Failure Cause Management". The connection has appeared if the consent UpdateCauses is set for the current user. Press "Create New" and include a name and a depiction for the Failure Cause. The depiction ought to contain the motivation behind why this build flopped just as potential answers for the build failure.



**[Is there any relationship between Hudson and Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3037)**

The basic difference is one is corporate offering while other is community offering. If we consider all aspect then this would be very exciting to know that Hudson is the conventional name for Jenkins. Jenkins is the latest offering by the core engineers of Hudson. To know why you have to know the historical backdrop of the project. It was initially open source and bolstered by Sun System. Like quite a bit of what Sun did, it was genuinely open, yet there was a touch of amiable disregard. The source, trackers, site, and so forth were facilitated by Sun on their closed java.net environment. At that point Oracle purchased Sun. For different reasons Oracle has not been modest about utilizing what it sees as its assets. Those incorporate some power over the strategic platform of Hudson, and especially command over the product name. Numerous clients and developers weren't happy with that and chosen to leave. So it comes down to what Hudson versus Jenkins offers. Both Oracle's Hudson and Jenkins have the code. Hudson has Oracle and  Sonatype corporate help and the brand. Jenkins has the vast majority of the core engineers, the network, and (up until this point) considerably more genuine work.

**[Define the triggers in Jenkins.](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3038)**

There are various ways to re-trigger the pipeline in an automated way, triggers are characterized in the Jenkins. A few of pipelines are cooperating with sources like GitHub, BitBucket, or different triggers first then they are actualized to play out a particular activity.

A build trigger might be utilized for different purposes relying upon the context of the project.

* **For instance:**In the event that an enterprise might want to have a CI/CD pipeline setup utilizing Jenkins. In this case, an organization can set up build triggers to trigger downstream system build, for example,
  + Integration tests
  + Code health check up
  + Load Tests
  + Start to finish Tests
  + Deployment.
* The above steps will be anchored to the parent work and can be activated one by one or in parallel contingent upon the stage (this is the place where build trigger is utilized, trigger the downstream system build if the parent build is a success).
* Build occasionally can be utilized to keep running on standard jobs(in the event that we have a team set up which will be deploying master every evening). At that point, we can set up the jobs to build occasionally late night times at a fixed time (additionally the job can be activated dependent on progress as clarified in (1)
* Polling SCM keeps checking for any new code addition by checking commit history and trigger build thereafter.

Pooling your archive and construct dependent on that

**[Polling a Git repository for new commits is considered a Jenkins anti-pattern. What is a sound alternative to SVN polling?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3040)**

A continuously checking /monitoring SCM tools like GIT or subversion for identifying any new commits are considered a waste of clock cycles and not considered best practices. We should avoid this process. There is a better approach available to us which is a reversal of the above approach. The other approach is industry standard one and more popular. The other approach encourages build getting triggered from source code tool whenever new code is committed or existing code undergoes some change. This is very easy to configure with GitHub or GItLab using a post-commit hook that runs every time code commit is successful. This setup eliminates the need for constantly monitoring source code as a post-commit hook will trigger the build whenever any code gets committed in the source code.

**[How can you move or copy Jenkins from one server to another?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3041)**

Move a job starting with one installation of Jenkins then onto the next by just duplicating the relating work directory. Make a duplicate of a current job by making a clone of a job directory by an alternate name. Rename a current job by renaming a directory. Note that on the off chance that you change a job name you should change whatever other activity that endeavors to call the renamed job. Those tasks should be possible notwithstanding when Jenkins is running. For changes like these to produce results, you need to click "reload config" to compel Jenkins to reload design from the disk.

We can do the steps below to copy Jenkins from the primary server to other servers:

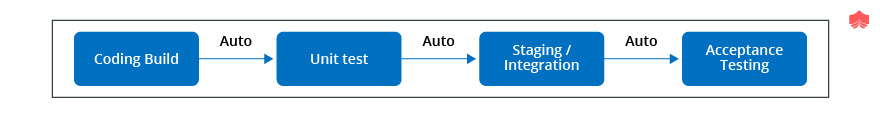
1. As part of the first step, we can start copying and moving Job directory & job name from the primary installation of Jenkins to another one.
2. We can also start making another copy of the already configured job by cloning job directory by renaming it to a different name.
3. Renaming already configured job by giving it some other name.

We can also try one of the plug-ins as well as having Job export options like Job Importer Plugin. Jenkins CLI can be also used if we have less number of jobs but usually at the enterprise level, we have a large number of jobs so not widely used.

**[What is the difference between Continuous Integration, Continuous Delivery, and Continuous Deployment?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3042)**

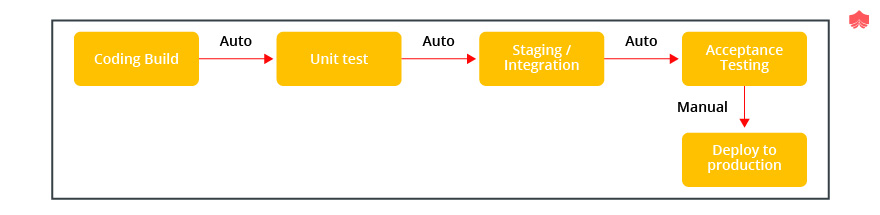
**Continuous integration**

CI is DevOps practices where developers regularly integrate code with at regular time interval  The integration is accompanied by build and tests in an automatic way. The automated testing is optional in  CI but it is implied. The biggest advantage of CI is frequent integration which leads to early detection of errors and early remediation. We can work on changes which lead to specific build error or code issue. Since integration was frequent so fixing becomes easier.



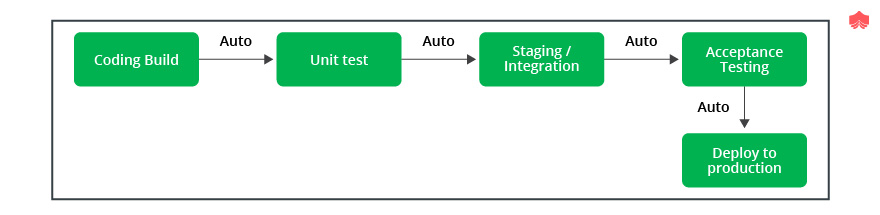
**Continuous delivery**

CD is the next step of CI so it can be considered as an extension of continuous integration which facilitates releasing new changes in the production environment in a controlled way. The code base has to always be in a deployable state to facilitate CD pipeline in DevOps.The continuous delivery ensures seamless delivery of any changes whether it be code /configuration in production. This is very useful for the complex system where several developers working on the same code base making it very challenging to ensuring code base in a deployable state. The advantages of  CD is early detection of production issue and quick fix for same.



**Continuous deployment**

This practice is one more step ahead of continuous delivery. The continuous delivery releases changes in production in a controlled way but continuous deployment facilitate the release of changes to production automatically once all changes pass through production pipeline stages successfully. It ensures no human intervention. The feedback loop is much faster if we enable continuous deployment for our project. It supports the theme fast to market. It is similar to continuous delivery but release happens seamlessly.



**[What is the difference between Maven, Ant, and Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3043)**

Maven and ANT are classified as build tool but Maven has one additional advantage that it supports project management, dependency management, and standard project layout. Jenkins is a continuous integration tool which is much more superior than a build tool. ANT is the oldest one in a  lot and widely used build tool. A build tool is useful in creating binary artifacts like JAR file or WAR file. We can easily set up continuous delivery pipeline using Jenkins /Hudson which ensures triggering automatic build, test and deploys code base to production. The gang of four ANT,  Maven, Jenkins, and Hudson are tools for building, uni testing, continuous integration and facilitate project management. The Jenkins and Hudson are siblings from the same family while one is enterprise offerings and other is open source supported by dev community. Maven offerings are not mere of any other build tool offerings. We need to set up everything in ANT  like souce, build and target directory while Maven has a predefined structure for same. It is said that convention rule dominates Maven while ANT follows configuration rule. The continuous integration tool helps us in triggering build whenever any new changes committed in the source code which ensures if build gets compiled successfully or not. At the same time when compilation is successful, run the unit test and deploy build at a scheduled time ANT vs Maven



**[Why is Jenkins called continuous delivery tool?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3044)**

Continuous Delivery is the capacity to get changes of assorted types—including new highlights, design changes, bug fixes and analyzes—into generation, or under the control of clients, securely and rapidly in a reasonable manner. We accomplish this by guaranteeing our code is always in a deployable state, even despite groups of thousands of engineers making changes once a day. We along these lines totally kill the integration, testing and hardening stages that generally pursued "dev complete", just as code freezes.

Developers take a shot at their local environment for making changes in the source code and push it into the code repository. At the point when a change is distinguished, Jenkins plays out a few tests and code standards to check whether the changes are great to deploy or not. Upon a successful build, it is being seen by the developers. At that point, the change is conveyed manually on a staging environment where the customer can examine it. At the point when every one of the changes gets approved by the developers, testers, and customers, the final result is saved manually on the production server to be utilized by the end clients of the product. Along these lines, Jenkins pursues a Continuous Delivery approach and is known as the Continuous delivery Tool. Although there are manual steps

**ADVANCED**

#### [Explain how you can deploy a custom build of a core plugin?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-550)

Below are the steps to deploy a custom build of a core plugin:

* Stop Jenkins.
* Copy the custom HPI to $Jenkins\_Home/plugins.
* Delete the previously expanded plugin directory.
* Make an empty file called <plugin>.hpi.pinned.
* Start Jenkins.

#### [What is the Jenkins job DSL plugin?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-557)

The Jenkins Job DSL Plugin is basically the Domain Specific Language (DSL) plugin that allows users to describe jobs using a scripting language named Groovy, the plugin can manage the scripts and updating of the Jenkins jobs which are created and maintain them as a result.

#### [What is CI and CD process?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-559)

**CI (Continuous integration)**

Continuous Integration is a process that helps developers to integrate code into a shared repository several times. Each check-in is verified by an automated build process, allowing teams to detect problems early.

**CD (Continuous delivery)**

Continuous delivery is an extension of a continuous integration process which helps to make sure that new changes can be released to the customers quickly in a sustainable way.

#### [What is meant by the input directives in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-563)

With the help of input directives, the user could prompt the inputs with the help of input steps. The stage is paused after different options have been applied and before we can enter to the stage agent to evaluate its condition. Once the input is approved, we could continue with the stage ahead. The parameters that are given as the part of input submission will be available in the environment for the rest of the stage.

#### [What is meant by scripted pipeline in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-564)

The scripted pipeline is very much similar to the declarative pipeline that's built on the top of the underlying pipeline sub-system. The scripted pipeline is based on a general-purpose language based on Groovy. A list of features or benefits that are available in Groovy can be used along scripted pipeline too. In brief, this is a highly flexible tool that can be used in multiple continuous delivery pipelines.

#### [What is the blue ocean in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins#collapse-beginner-566)

Blue Ocean is a project that rethinks the user experience of Jenkins, modeling and presenting the process of software delivery by surfacing information that’s important to development teams with as few clicks as possible, while still staying true to the extensibility that is core to Jenkins. While this project is in the alpha stage of development, the intent is that Jenkins users can install Blue Ocean side-by-side with the Jenkins Classic UI via a plugin.



**[What is build pipeline in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3045)**

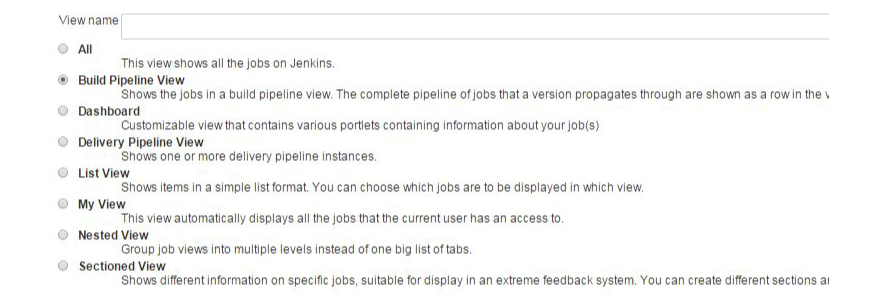
A pipeline is a collection of jobs that brings the product from version control under the control of the end clients by utilizing computerization apparatuses. It is an element used to consolidate consistent conveyance in our product advancement work process. Throughout the years, there have been various Jenkins pipeline releases including, Jenkins Build stream, Jenkins Build Pipeline module, Jenkins Workflow, and so forth. What are the key highlights of these modules? They speak to different Jenkins jobs as one entire work process as a pipeline. These pipelines are an accumulation of Jenkins employments which trigger each other in a predefined arrangement.

Let me clarify this with a model. Assume I'm building up a little application on Jenkins and I need to fabricate, test and convey it. To do this, I will distribute 3 employment to play out each procedure. Along these lines, job1 would be for fabricating, job2 would perform tests and job3 for an organization. I can utilize the Jenkins assemble pipeline module to play out this assignment. In the wake of making three employments and binding them in a grouping, the fabricate module will run these occupations as a pipeline.

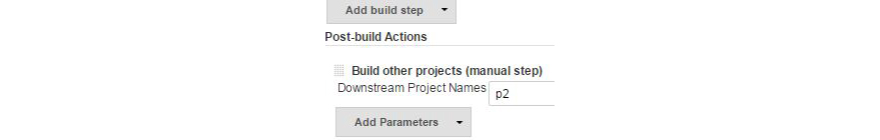
**Step 1:**  For creating a build pipeline, First create a set of jobs.



**Step 2:**  Now create a build pipeline view and in the configure section add the first job that you want to run in the pipeline



**Step 3:** Now for every job, add a post-build action, which can be modified many ways, for example, manual trigger or automatic trigger on the success of the build.



**step 4**: Once all the downstream jobs are configured, just build the p1 job. and go to the new pipeline view that you have created.



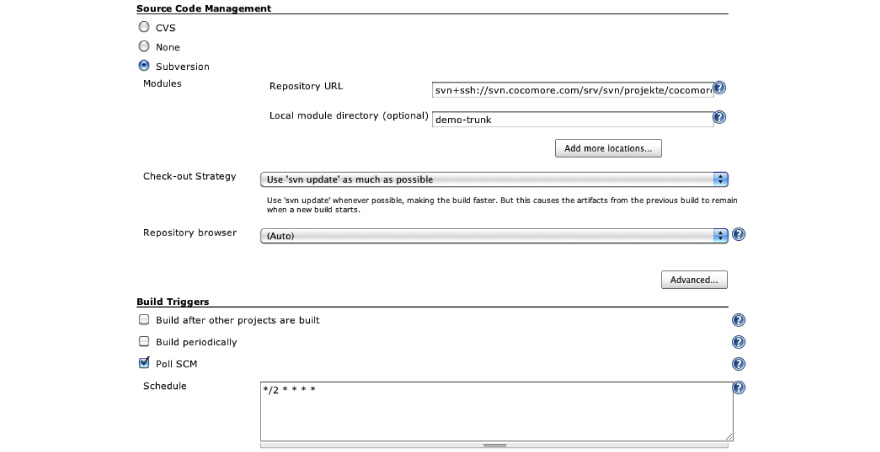
**Note:** You won't see a pipeline until the first job is started. In this example, after the p1 job is started you will start seeing a pipeline.

**[How do you configure automatic builds in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3046)**

Works in Jenkins can be activated intermittently (on a timetable, indicated in the arrangement), or when source changes in the task have been distinguished, or they can be naturally activated by mentioning the URL:

http://YOURHOST/jenkins/work/PROJECTNAME/construct

In the undertaking arrangement, there ought to be a Build Triggers area. This controls how frequently Jenkins polls your SCM for code changes. When utilizing SVN, you can stand to check as often as possible in light of the fact that the checking isn't costly. So you can advise Jenkins to check each moment or something like that. Set this to Poll SCM and set the calendar to something like \*/n \* \* \* \* (supplant n with your poll interval in minutes).



**[How to create a backup and copy files in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3047)**

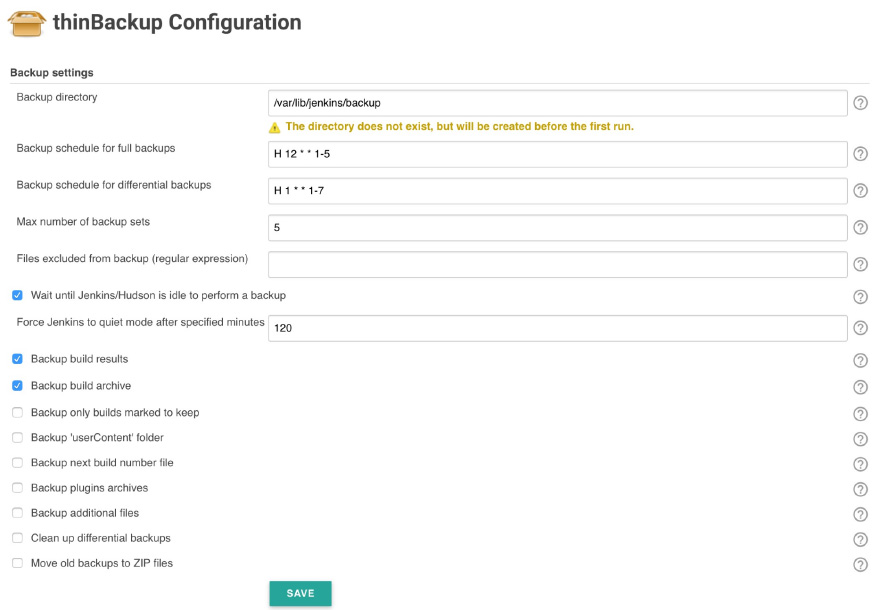
It is critical to have Jenkins reinforcement with its information and setups. It incorporates, work configs, manufactured logs, modules, module design, etc. Jenkins Thin Backup is a   module for sponsorship up Jenkins. It backs up every one of the information dependent on your timetable and it handles the reinforcement maintenance too.

**To begin, first, introduce the module.**

1. Go to Manage Jenkins – > Manage Plugins
2. Snap the Available tab and look for “Thin backup”
3. Introduce the module and restart Jenkins.

**Once introduced, pursue the means given beneath for designing reinforcement settings.**

1. Go to Manage Jenkins — > ThinBackup
2. Snap settings alternative.
3. Enter the reinforcement choices as appeared and spare it. Every one of the alternatives is clear as crystal. The reinforcement index you determine ought to be writable by the client which is running the Jenkins administration. All the Jenkins reinforcement will be spared to the reinforcement registry you indicate.



It's anything but a smart thought to keep the Jenkins back in Jenkins itself. It is an absolute necessity to move slim reinforcements to distributed storage or some other reinforcement area. So that, regardless of whether Jenkins server crashes you will have every one of the information. In the event that you are on AWS, Azure or Google CLoud, you can transfer the reinforcements separate stockpiling arrangements.

There is an alternative way of backup of the Jenkins Home folder. This contains all of your build jobs configurations, your slave node configurations, and your build history. To create a backup of your Jenkins setup, just copy this directory.

**You can see where is your Jenkins home with:**

echo $JENKINS\_HOME

And for example, if you only want to back up the jobs you can go to:

cd $JENKINS\_HOME/jobs

And make a backup for that folder.

All that configuration will be a bunch of XML files.

If you are using the [official Jenkins docker image](https://hub.docker.com/_/jenkins/), the home will be on: /var/jenkins\_home

**[Which Commands Can Be Used To Start Jenkins Manually?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3048)**

There are various ways to start and stop Jenkins. We have commands/ plug-ins to achieve the same.

Firstly, we can use any one of the below commands to start Jenkins manually using Jenkins URL:

(Jenkins\_url)/restart:       Forces a restart without waiting for builds to complete.

(Jenkin\_url)/safeRestart:   Allows all running builds to complete.

We should always do safeRestart as it waits for running build to complete before restarting Jenkin server. There are other ways as well to start/stop. We can go to Open Console/Command line --> Go to our Jenkins installation directory. The below commands also produce the same output:

**to stop:**

jenkins.exe stop

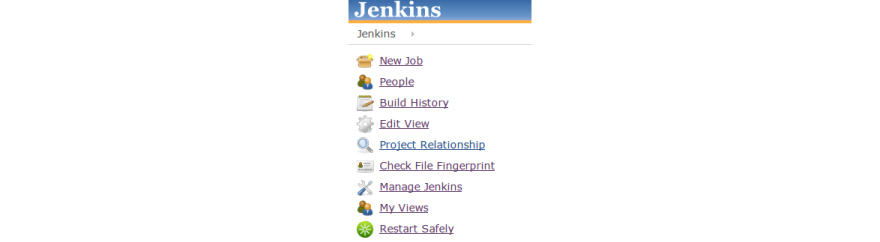
**to start:**

jenkins.exe start

**to restart:**

jenkins.exe restart

The [SafeRestart Plugin](https://wiki.jenkins-ci.org/display/JENKINS/SafeRestart+Plugin) can also be used. It is pretty useful (Restart Safely). Once you install the plug-in, It adds a link to be able to restart from the main menu only:



In Red Hat Linux you can use below commands as well :

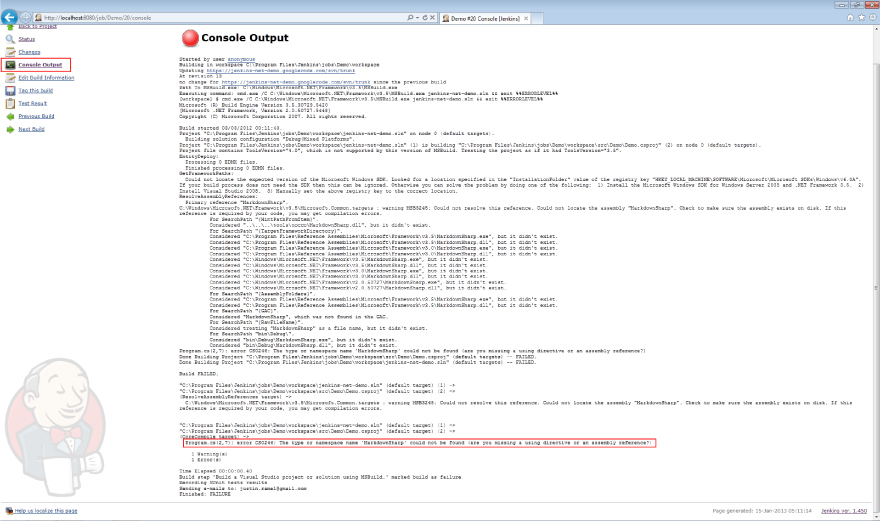
* **To know the status of Jenkins:** sudo service Jenkins status
* **To start the Jenkins:**sudo service Jenkins start
* **To stop the Jenkins:** sudo service Jenkins stop
* **To restart the Jenkins:** sudo service Jenkins restart

**[What You Do When You See A Broken Build For Your Project In Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3049)**

We can check the console output in Jenkins to investigate the output of the build. This can help us in identifying if any files change missed during commit. If the console output is of no help we can use a local copy of workspace to replicate the issue.

Last time we had a failing build with Jenkins showing a red ball to prove it. The first place to look when we have a failing build is the console output. You can get to the console output via the main menu on the left of your project page.

### Console output



**[What is the syntax Jenkins uses to schedule items such as build jobs and SVN polling?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3050)**

Jenkins employs the cron syntax to schedule jobs within the tool.

Five asterisks are the core of cron language structure, with every one isolated by a space. The principal asterisk mark speaks to minutes, the second speaks to hours, the third day of the month, the fourth the month itself and the fifth day of the week. For instance, to plan a build job to pull from GitHub each Friday at 6:30 p.m., the language structure would be: 30 18 \* 4.

A CRON expression is a string containing five or six fields isolated by space that speaks to a lot of times, regularly as a schedule to execute some everyday practice. In certain employment of the CRON group, there is likewise a seconds field toward the start of the example. In that case, the CRON expression is a string involving 6 or 7 fields. A CRON syntax also supports special  Backing for every special character relies upon explicit appropriations and versions of cron.

Asterisk( \*): The reference mark shows that the cron expression matches for all values of the field. E.g., utilizing an asterisk in the fourth field (month) shows each month.

Slash(/) :Slash portray additions of reaches. For instance, 4-59/15 in the first field (minutes) demonstrates the 4 minute of the hour and every 15 minutes from that point. The structure "\*/… " is comparable to the structure "first-last/… ", that is, an addition over the largest possible range of the field.

The comma (,): Commas are utilized to isolate things of a rundown. For instance, utilizing "MON, WED, FRI" in the fifth field (day of the week) implies Mondays, Wednesdays, and Fridays.

Hyphen ( – ) :Hyphens characterize ranges. For instance, 2000-2010 shows each year somewhere in the range of 2000 and 2010 AD, comprehensive.

Percent ( %): Percent-signs (%) in the direction, except if got away with a backslash (\), are changed into newline characters, and all information after the first % is sent to the order as standard info.



**[Describe the standard process to configure and use third-party tools within Jenkins.](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3051)**

The procedure to utilize a third party tool, for example, Artifactory, Node, SonarQube or Git normally pursues a four-advance procedure.

1. The third-party tool must be installed.
2. A Jenkins module that supports the third party tool must be introduced through the Jenkins administrator console.
3. The third-party tool must be arranged in the Tools tab of the Manage Jenkins area of the administrator console.
4. At last, the plug-ins can be utilized from inside a Jenkins build job. The module will at that point encourage correspondence between the Jenkins build job and the third party.

**[How can you temporarily turn off Jenkins security if the administrative users have locked themselves out of the admin console?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3052)**

The JENKINS\_HOME organizer contains a document named config.xml. At the point when security is empowered, this record contains an XML component named useSecurity that will be set to true. By changing this setting to false, security will be handicapped whenever Jenkins is restarted. **<useSecurity>false</useSecurity>**

The incapacitating security ought to dependably be both a final retreat and a brief measure. When any conformation issues are settled, make certain to re-empower Jenkins security and reboot the CI server.

Please find the steps below :

* Go to $JENKINS\_HOME in the file system and discover the config.xml document.
* Open this document in the editorial manager.
* Search for the <useSecurity>true</useSecurity> component in this document.
* Supplant true with false
* Expel the components authorization strategy and security realm
* Start Jenkins

At the point when Jenkins returns, it will be in an unsecured mode where everybody gets full access to the framework. In the event that this is as yet not working, taking a stab at renaming or erasing config.xml.

**[What is the difference between Agent and Node?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3053)**

**Agent:**The straightforward answer is, Node is for scripted pipelines & Agent is for declarative pipelines. In declarative pipelines, the agent directive is utilized for determining which agent /slave the job/task is to be executed on. This mandate just enables you to indicate where the undertaking is to be executed, which agent, slave, mark or docker image. In scripted pipelines, the node step can be utilized for executing a script/advance on a particular agent, mark, slave. The node step alternatively takes the operator or name and afterward a conclusion with code that will be executed on that node.

The declarative pipelines is another augmentation of the pipeline DSL (it is fundamentally a pipeline content with just one stage, a pipeline venture with contentions (called directives), these directives ought to pursue a particular language structure. The purpose of this new arrangement is that it is progressively exacting and accordingly ought to be simpler for those new to pipelines, take into account graphical altering and considerably more. scripted pipelines are the fallback for cutting edge prerequisites.

**[How do you secure Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3054)**

In the default setup of Jenkins 1.x, Jenkins does not play out any security checks. This implies the capacity of Jenkins to launch procedures and access local files are accessible to any individual who can get to Jenkins web UI and some more.

Securing Jenkins has two viewpoints to it.

1. Access control, which guarantees clients are verified when getting to Jenkins and their activities are approved.
2. Securing Jenkins against outer dangers

You should secure the entrance to Jenkins UI with the goal that clients are validated and suitable arrangement of authorizations are given to them. This setting is controlled for the most part in two ways:

* Security Realm, which decides clients and their passwords, just as what groups the clients have a place with.
* Approval Strategy, which figures out who approaches what.

You may utilize outside LDAP or Active Directory as the security domain, and you may pick "everybody full access once signed in" mode for approval methodology. Or then again you may let Jenkins run its very own client database, and perform access control dependent on the authorization/client grid.

Some important security considerations:

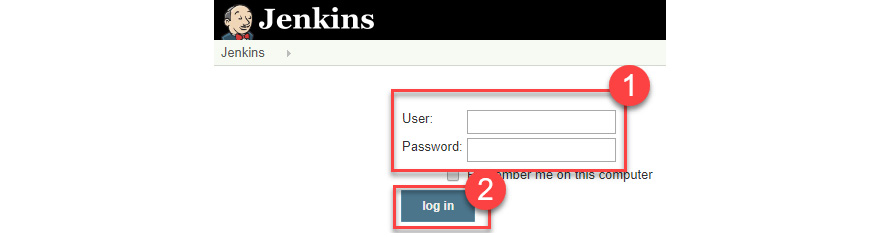
* Global security ought to be empowered.
* Jenkins ought to be incorporated with suitable modules.
* Automate the way toward setting rights and benefits.
* Limit the physical access to organizers.
* Intermittently run security reviews.

**[How do you create a Job in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3055)**

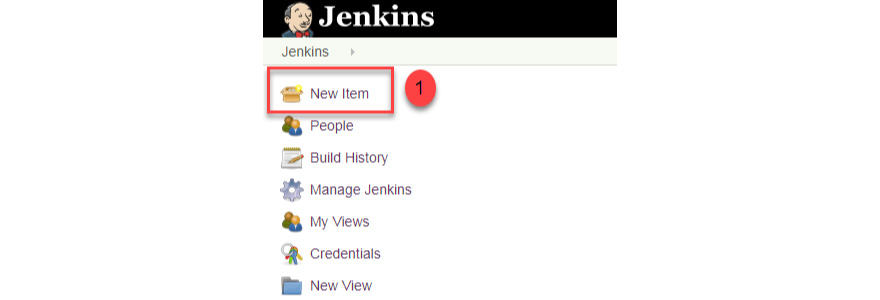
As a part of the first step we need to visit the Jenkins URL, once we are on Jenkin page we need to click "New Job", then we need to click on  "Build a free-style software project". The above activity consists of several components: like SCM, for instance, CVS or Subversion where our source code abides. optional triggers to control when builds will be performed by Jenkins.some sort of script that plays out the build(ant, maven, shell script and so on.) where the veritable work happens optional steps to accumulate information out of the build, for instance, archiving the artifacts along with recording Javadoc and test results.

Please find the below steps for creating jobs in Jenkins :

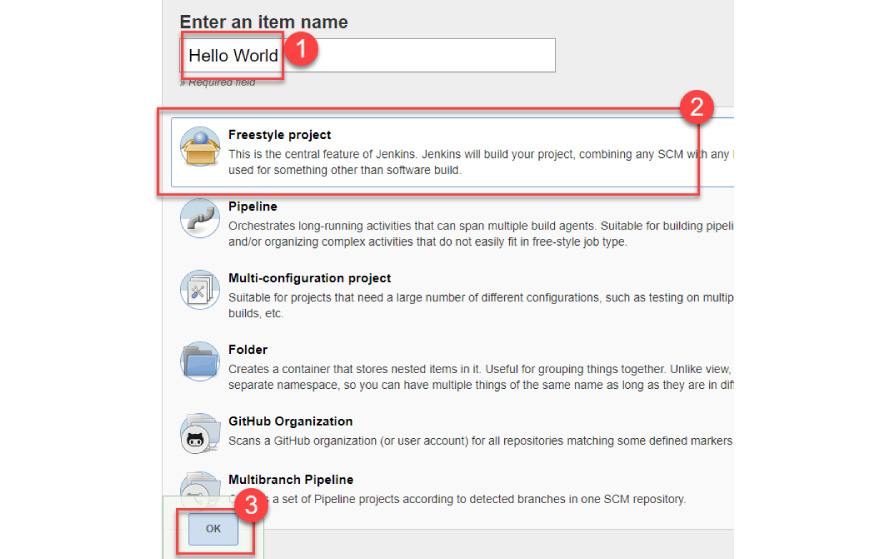
**Stage 1:**First of all we need to sign in to Jenkins Dashboard for creating a freestyle job. Usually, Jenkins hosted locally at only if we have configured Jenkins in some other way, use the hosted URL to get to our Jenkins dashboard.



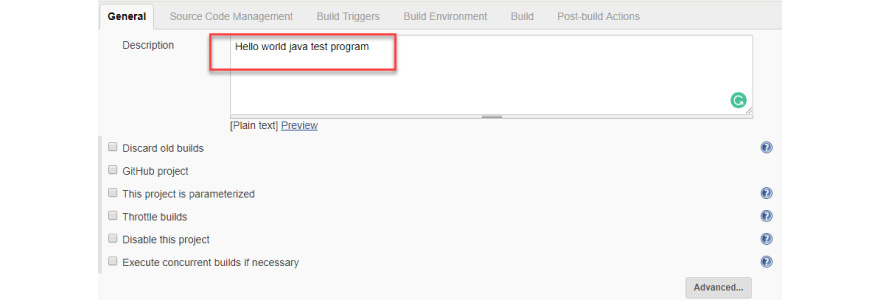
**Stage 2:**In the upper left-hand side of our dashboard click the "New Item".



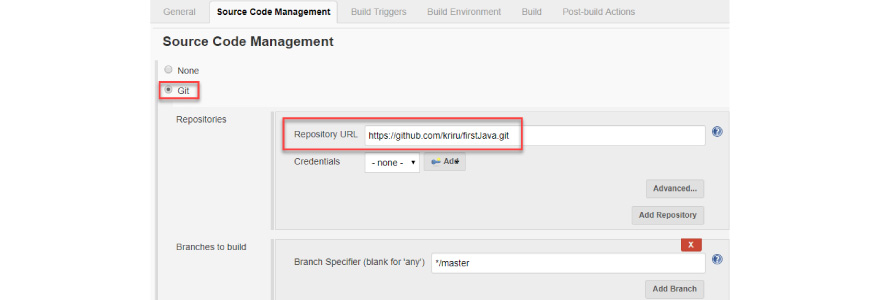
**Stage 3:** In the following screen, we need to enter the name of the item which we would like to create. We will create the "Hello world" for this exercise: Choose Freestyle project and click OK



**stage 4:** Let's enter some project details like below screen :



**Step 5:** Now we can click on the Source Code Management section and enter our repository URL. We have a test repository hosted at [https://github.com/\*/\*.git](https://github.com/*/*.git)



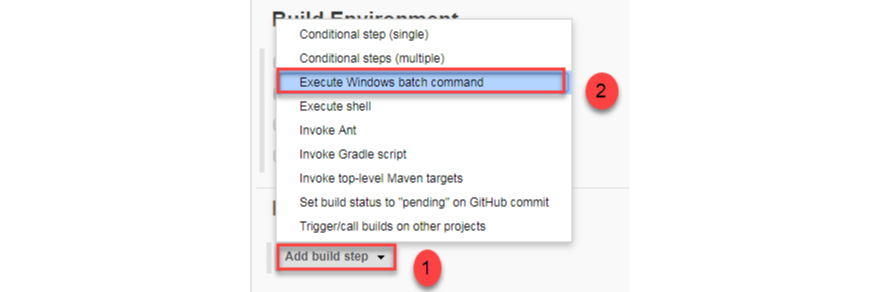
We can also leverage local repo as well in case if we have.

On the off chance that if our GitHub repo is private, Jenkins will initially approve our login accreditation with GitHub and after validation pulls the source code from our GitHub repo.

**Stage 6:** Once all configuration settings completed, it's an ideal opportunity to build the code. We can change the settings under the build segment to build the code at the time whenever we need. Build can be scheduled as well to trigger it at the specified time.

In the build section we need to do below two things :

1. We need to first click on "Add build step"
2. We need to click on  "Execute Windows batch command" to execute the commands we wanted to trigger during the build process.

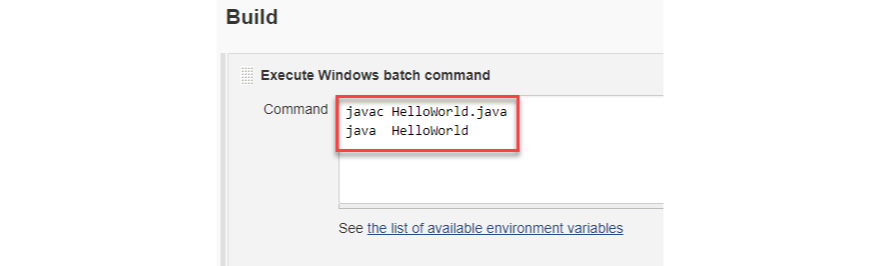


Let's add some sample java commands to test the workflow :

We can add below commands for our testing :

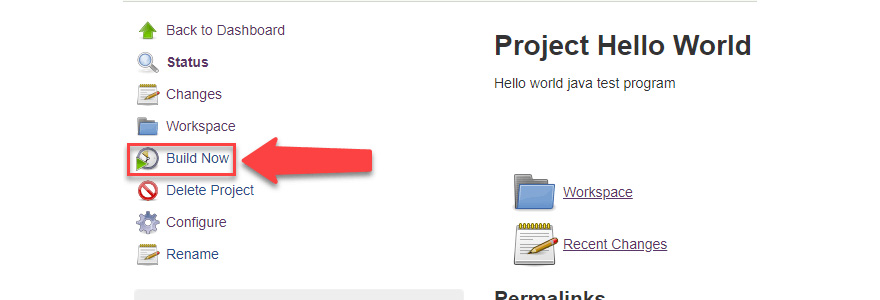
javac HelloWorld.java

java HelloWorld

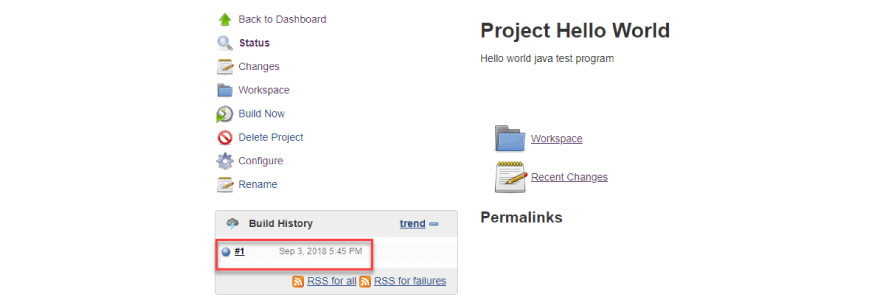


**Stage 7:** Once we are done with all the changes, we can click apply and then save the project.

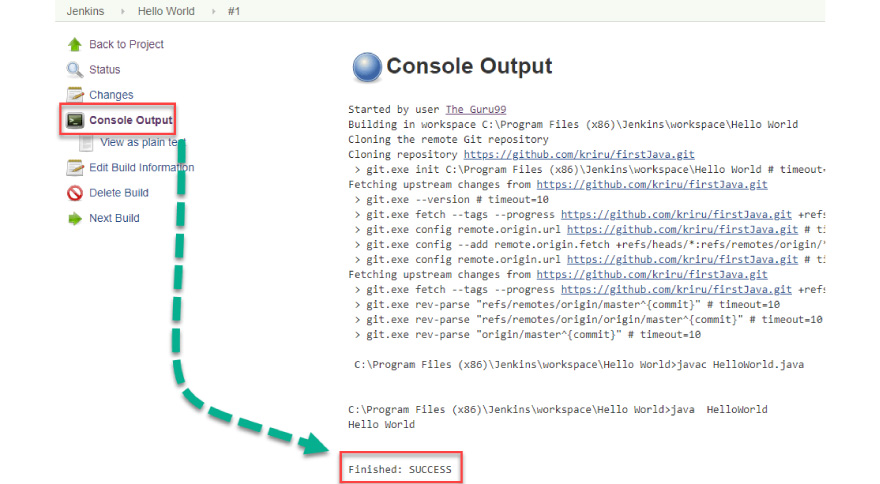
**Stage 8:**We should click the Build Now button which is on the left-hand side to trigger the build.



**Stage 9:** Under build history section we can check the status of every build triggered.



**Step 10:**Click on the build number and then Click on console output to see the status of the build you run. It should show you a success message, provided you have followed the setup properly.



**[What Are The Most Useful Plugins In Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3056)**

**Some most useful plugins in Jenkins:**

### Amazon ECS Container Service

The objective of  ‘Amazon ECS Container Service’ plugin to manage Jenkins agent hosted on the cloud and at the same time helping in deploying Docker-based applications on a cloud. Each Jenkin build is carried out in separate docker container which gets cleaned up after.

### Dashboard View Plugin

This dashboard gives a bird view of the status of tasks configured in Jenkins. This is used for monitoring purposes. It also helps us in tracking the time taken in building jobs which are configured and entire time duration for all the jobs.

### View Job Filters Plugin

It creates views for Jenkins jobs. We can have a view of build status and different triggers.

### Build Pipeline Plugin

This gives us clear sight of jobs making our build pipeline also we can have a better look of upstream and downstream. Additionally, it helps us in defining manual triggers for some tasks which need some customization before executing. It is one of the critical plugins as it has in-built support of scripts which helps in building complex DevOps pipeline.

### Git Plugin

This plug-is needed in case If we need to access the GitHub but at the same time it works as a repository browser for other SCM providers.

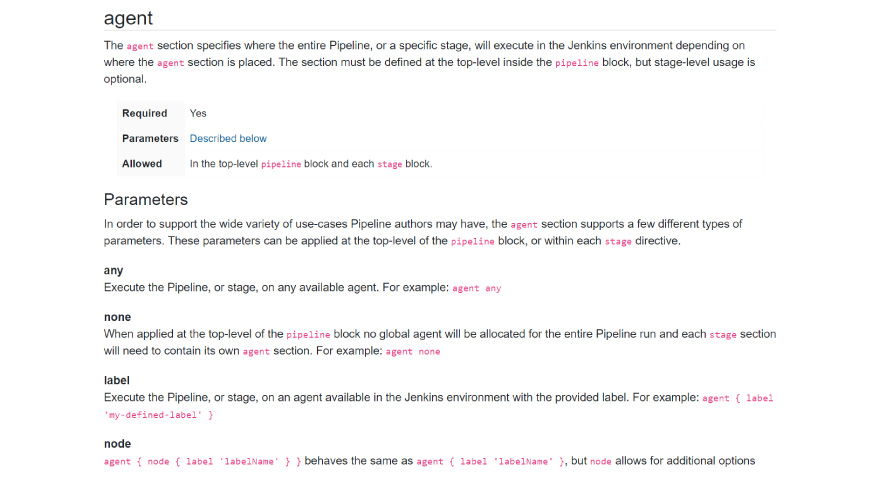
### GitHub Integration Plugin

This is one of the basic plug-ins which we need to get the source code from code repository hosted with GIT. It helps us in scheduling build, pulling code base on regular interval from GIT to Jenkin. The build gets triggered automatically once scheduled.

**[How will you define the Jenkins agent?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3057)**

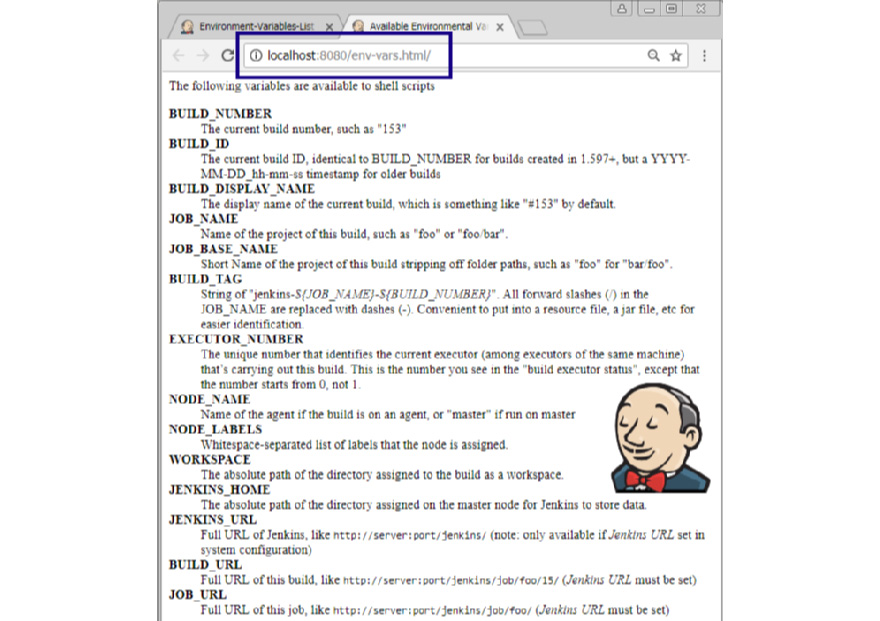
An agent represents the build pipeline or highlighting specific steps where execution will be getting performed or agent location. Inside the pipeline block,  an agent is available at a higher level but it is optional to use stage-level.

1. The agent section determines whether an entire pipeline or particular stage will be part of Jenkins environment driven by where exactly agent section is located. The agent section must be at the top-level and within pipeline block only. You can visit the below URL to get some more details on agent syntax in Jenkin pipeline. .(<https://jenkins.io/doc/book/pipeline/syntax/#agent>
2. There are many ways to create Agent/Node but we can follow the below tutorial to know about steps to create agent/Node.<https://devopscube.com/setup-slaves-on-jenkins-2/>
3. The different parameters are supported by Agent to support various use   These parameters actually work at the top level of pipeline block or it can work under stage directive as well.



**[Name a Jenkins environment variable you have used in a shell script or batch file.](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3058)**

The Jenkins environment variables frequently prove to be useful when you have to keep in touch with some advanced shell contents. Moreover, in the event that you realize how to infuse environment variables into the Jenkins build process, it can open up a totally different universe of specialized chances, as it'll give you access to a portion of the product's internals.

A simple method to get the Jenkins environment variables list from your local installation is to annex env-vars.html to the server's URL. For a privately facilitated Jenkins server, the URL would be **http://localhost:8080/env- vars.html.  **

The least demanding approach to perceive how these Jenkins environment variables work is to make a freestyle job, reverberation every section in the rundown and see the value Jenkins relegates to every property. By default, few environment variables are always available in Build Job.  Some of them are available only when a relevant plug-in is configured in Jenkin set up some GIT related variables are available when GIT plugin gets configured.

**[Name three steps or stages a typical Jenkins pipeline might include.](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3059)**

Jenkins pipeline is configured to build a project by extracting it from source code and then ensure that the build goes through different stages like unit, performance, and user acceptance testing. Once every stage is successful, it also facilitates deployment to an application server. So overall if we talk about different stages any project goes through can be classified into three broad categories

* **Build -**This stage ensures code extracted from code repository for build purpose and in case of any failure, developers come to know the reason for build failure. This is a very critical stage in build pipeline and subsequent stages will be triggered only when the project exits this stage successfully.
* **Test -**This stage ensures the build is unit /performance/user tested so the issue can be caught at an early stage only.
* **Deploy -** This stage took care of deployment request once testing is successful It is the last stage in the pipeline.

**The above stage can be further divided into smaller sections which help us in understanding the importance of three primary stages of Jenkins pipeline:**

* Pull the code from source repository using the proper plug-in. for GIT source code we can utilize GIT plugin and so on.
* Once the code extracted ensure to compile the code using compatible compiler library like Maven Plugin for Java code.
* The conformation with coding standard is also well supported by some of the plugins available in Jenkins. We can use the Checkstyle plugin for the same.
* The Code health check-up is also well supported by different tools available like SonarQube, PMD or FindBugs.
* Incorporating groovy syntax we can get manual sign off from Business users easily.
* We can run different tests to measure application load performance.
* It also helps is packaging application in a form we called ready to deploy stage. For example, the WAR format supported for JAVA web application project, etc.
* It also facilitates deployment of binary to artifacts repository like Nexus.
* It also helps in storing most of the reports for future reference.

**[Name two ways a Jenkins node agent can be configured to communicate back with the Jenkins master.](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3060)**

**There are two ways we can start Jenkins Node agent :**

1. We can start a Node agent from a browser window itself.
2. There is another way of starting an agent from the command line as well.

The JNLP file gets downloaded when we start an agent using a browser window. When this file runs it triggers a separate process to launch Jenkins jobs. There is a different process which runs in case of agent get launched from the command line. There is one JAR file which is required on the client machine and this file gets launched from the command line but still, you need to refer slave agent JNLP file available on Jenkin server. The command line triggers a process on the client machine which actually communicates with Jenkin’s mater and triggers build jobs in Jenkin when it identifies idle clock cycles

**[Name five important DevOps tools that organizations should consider adopting when undergoing a DevOps transition?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3062)**

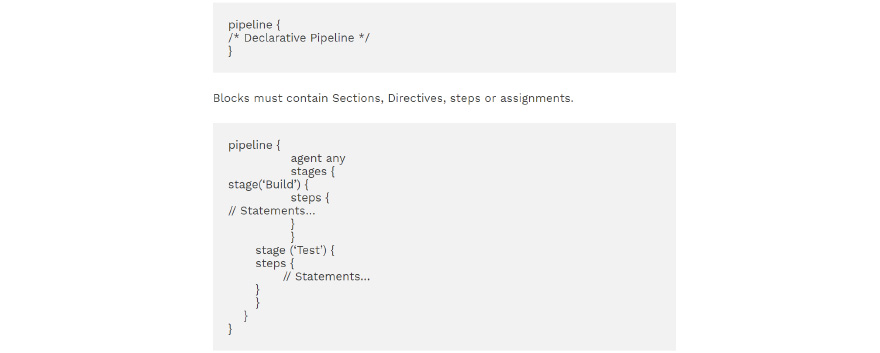
There are five key areas in which tools can assist in a DevOps transition:

* **configuration management**: There are various configuration management tools. For example Chef, Puppet and Ansible are considered to be best configuration management tools.
* **source code management**: This helps any enterprise in managing source code in such a way that a distributed team can work efficiently. There are several popular source code tool available like Git, GitHub or GitLab, are few of them to name.
* **CI**: Jenkins is leading one in CI tools and most popular one if industry wise trends are considered. There are other CI tools as well like Concourse CI and Atlassian's Bamboo.
* **containerization**: docker is a market leader in this section but there are others as well like Rkt and LXD.
* **collaboration**: JIRA from Atlassian is an incredible Project Management programming and in the meantime can likewise be an exceptionally solid Collaboration device that can be utilized in an Organization. It is a product device structured particularly to catch, dole out and organize tasks for the advancement of the Project execution. It has one of the basic and natural interfaces that assists anybody with no learning to pick up a grasp over it right away.

**[What are declarative pipelines in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3063)**

Jenkins has two ways of creating pipelines. One is declarative and another one is scripted. The scripted pipelines are also called as traditional pipelines, it is dependent on groovy syntax.  The declarative one is an easy one as syntax is more simplified. Declarative pipeline got introduced after Jenkin version 2.5. Declarative Pipelines are the latest offering from Jenkins that streamline the traditional groovy syntax of Jenkins pipelines (top-level pipeline) with specific use cases, for example, No semicolon can be used as a statement separator. The top-level pipeline ought to be encased inside block viz;

The regular syntax  structure for a declarative pipeline is below :



The declarative pipeline has three major sections explained below :

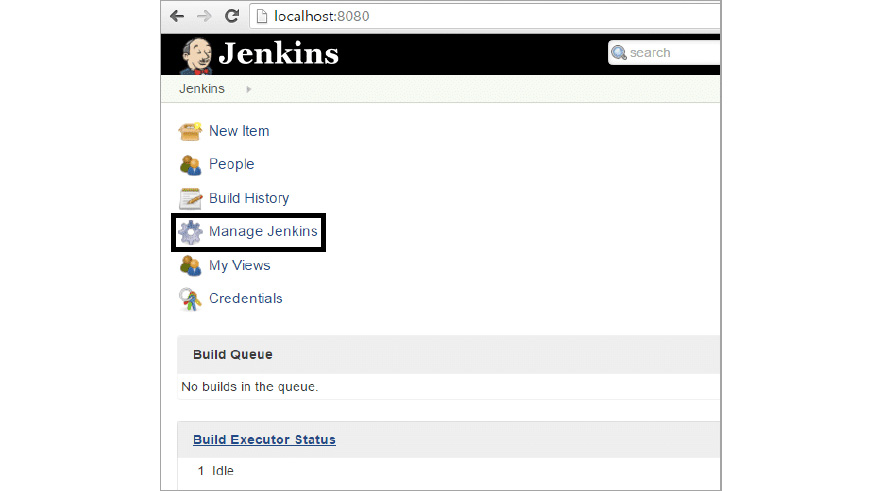
**Pipeline**: The section of the script where we write declarative pipelines. It is an umbrella under which all other sections will reside.

**Agent**: This is the starting point of a pipeline from where it starts executing.

**Stage**: The stage is nothing but steps enclosing all pipelines sections.

**[What is backup plugin in Jenkins?](https://www.knowledgehut.com/interview-questions/jenkins" \l "collapse-beginner-3064)**

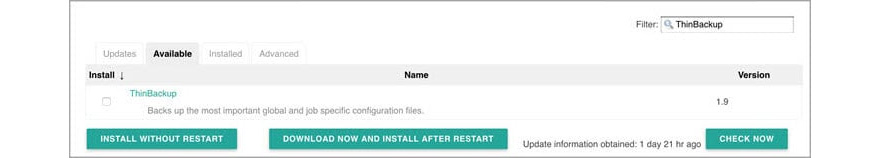
Jenkins Backup Plugin is used to take a back up of the configurations and settings so as to use them later on if there is any failure. We can follow the following steps for backing up our settings by utilizing the Backup Plugin.  
  
**Stage 1:**Log in to Jenkin server and then click on Manage Jenkins section.



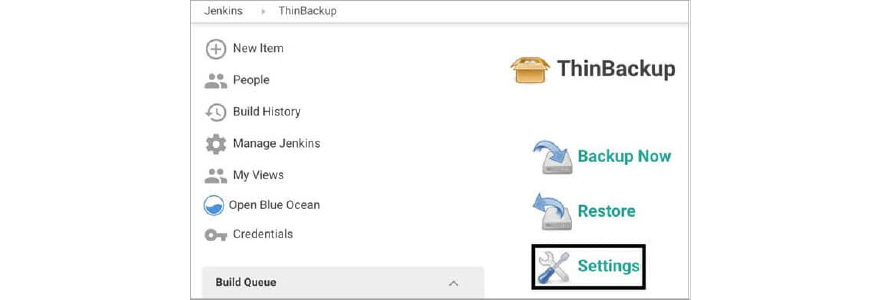
**Stage 2:** Once we click on Manage Plugins it opens up the below page.



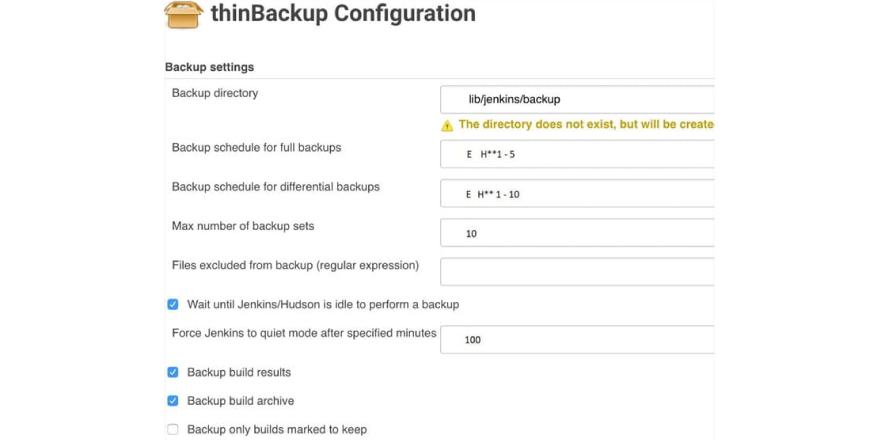
**Stage 3:** Click on Available section in the below page and search for ThinBackup in the filter section. Once we click on the selected it will start installing the plug-in the backend.



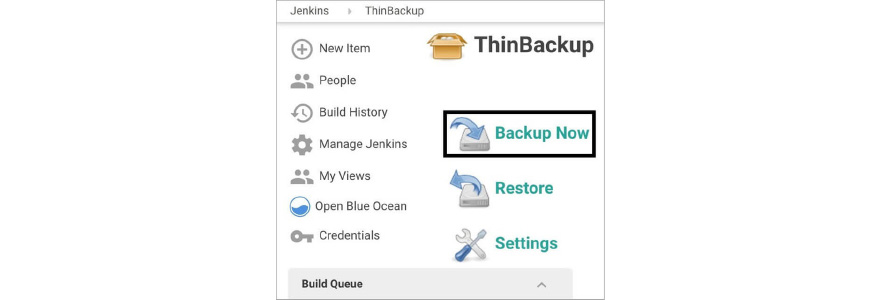
**Stage 4:** After successful installation, the below screen will pop up. Click the settings sections highlighted below.



**Stage 5:**Here we have to fill the basic details like backup directory as shown on the below screen and save the settings. The back-up will be stored to the Backup Directory.



**Stage 6:** We can test if a backup is working on not by clicking on Backup Now as shown in the below picture.



**Stage 7:**We can navigate to the back-up directory defined in  ThinBackup Settings to check if a backup exists or not.

**Kubernetes INTERVIEW**

**INTERMEDIATE**

## DESCRIPTION

Prepare better for your Application developer interview with the top Kubernetes interview questions curated by our experts. These Kubernetes Interview Questions & Answers will help convert your Application developer/DevOps engineer interview into a top job offer. The following list of interview questions on Kubernetes covers the conceptual questions for freshers and experts and helps you answer different questions like the difference between config map and secret, ways to monitor that a Pod is always running, ways to test a manifest without actually executing it. Get well prepared with these interview questions and answers for Kubernetes.

**[What is the difference between config map and secret? (Differentiate the answers as with examples)](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2405)**

Config maps ideally stores application configuration in a plain text format whereas Secrets store sensitive data like password in an encrypted format. Both config maps and secrets can be used as volume and mounted inside a pod through a pod definition file.

**Config map:**

                 kubectl create configmap myconfigmap

 --from-literal=env=dev

**Secret:**

echo -n ‘admin’ > ./username.txt

echo -n ‘abcd1234’ ./password.txt

kubectl create secret generic mysecret --from-file=./username.txt --from-file=./password.txt

**[If a node is tainted, is there a way to still schedule the pods to that node?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2406)**

When a node is tainted, the pods don't get scheduled by default, however, if we have to still schedule a pod to a tainted node we can start applying tolerations to the pod spec.

**Apply a taint to a node:**

kubectl taint nodes node1 key=value:NoSchedule

**Apply toleration to a pod:**

spec:

tolerations:

- key: "key"

operator: "Equal"

value: "value"

effect: "NoSchedule"

**[Can we use many claims out of a persistent volume? Explain?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2407)**

The mapping between persistentVolume and persistentVolumeClaim is always one to one. Even When you delete the claim, PersistentVolume still remains as we set persistentVolumeReclaimPolicy is set to Retain and It will not be reused by any other claims. Below is the spec to create the Persistent Volume.

**apiVersion:** **v1**

**kind:** **PersistentVolume**

**metadata:**

**name:** **mypv**

**spec:**

**capacity:**

**storage:** **5Gi**

**volumeMode:** **Filesystem**

**accessModes:**

**-** **ReadWriteOnce**

**persistentVolumeReclaimPolicy:** **Retain**

**[What kind of object do you create, when your dashboard like application, queries the Kubernetes API to get some data?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2408)**

You should be creating serviceAccount. A service account creates a token and tokens are stored inside a secret object. By default Kubernetes automatically mounts the default service account. However, we can disable this property by setting automountServiceAccountToken: false in our spec. Also, note each namespace will have a service account

**apiVersion:** **v1**

**kind:** **ServiceAccount**

**metadata:**

**name:** **my-sa**

**automountServiceAccountToken:** **false**

**[What is the difference between a Pod and a Job? Differentiate the answers as with examples)](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2409)**

A Pod always ensure that a container is running whereas the Job ensures that the pods run to its completion. Job is to do a finite task.

**Examples:**

kubectl run mypod1 --image=nginx --restart=Never

kubectl run mypod2 --image=nginx --restart=onFailure

○ → kubectl get pods

NAME           READY STATUS   RESTARTS AGE

mypod1         1/1 Running   0 59s

○ → kubectl get job

NAME     DESIRED SUCCESSFUL   AGE

mypod1   1 0            19s

**[How do you deploy a feature with zero downtime in Kubernetes?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2410)**

By default Deployment in Kubernetes using RollingUpdate as a strategy. Let's say we have an example that creates a deployment in Kubernetes

kubectl run nginx --image=nginx **# creates a deployment**

○ → kubectl get deploy

NAME    DESIRED  CURRENT UP-TO-DATE   AVAILABLE AGE

nginx   1  1 1            0 7s

Now let’s assume we are going to update the nginx image

kubectl set image deployment nginx nginx=nginx:1.15 **# updates the image**

Now when we check the replica sets

kubectl get replicasets **# get replica sets**

NAME               DESIRED CURRENT READY   AGE

nginx-65899c769f   0 0 0       7m

nginx-6c9655f5bb   1 1 1       13s

From the above, we can notice that one more replica set was added and then the other replica set was brought down

kubectl rollout status deployment nginx

**# check the status of a deployment** rollout

kubectl rollout history deployment nginx

**# check the revisions in a deployment**

○ → kubectl rollout history deployment nginx

deployment.extensions/nginx

REVISION  CHANGE-CAUSE

1         <none>

2         <none>

**[How to monitor that a Pod is always running?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2411)**

We can introduce probes. A liveness probe with a Pod is ideal in this scenario.

A liveness probe always checks if an application in a pod is running,  if this check fails the container gets restarted. This is ideal in many scenarios where the container is running but somehow the application inside a container crashes.

**spec:**

**containers:**

**-** **name:** **liveness**

**image:** **k8s.gcr.io/liveness**

**args:**

**-** **/server**

**livenessProbe:**

**httpGet:**

**path: /healthz**

**[What are the types of multi-container pod patterns? (Explain each type with examples)](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2412)**

* **sidecar:**

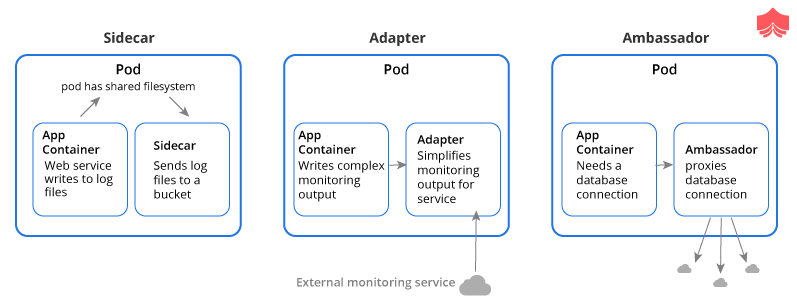
A pod spec which runs the main container and a helper container that does some utility work, but that is not necessarily needed for the main container to work.

* **adapter:**

The adapter container will inspect the contents of the app's file, does some kind of restructuring and reformat it, and write the correctly formatted output to the location.

* **ambassador:**

It connects containers with the outside world. It is a proxy that allows other containers to connect to a port on localhost.



reference:https://matthewpalmer.net/Kubernetes-app-developer/articles/multi-container-pod-design-patterns.html

**[What is the difference between replication controllers and replica sets?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2413)**

The only difference between replication controllers and replica sets is the selectors. Replication controllers don't have selectors in their spec and also note that replication controllers are obsolete now in the latest version of Kubernetes.

**apiVersion:** **apps/v1**

**kind:** **ReplicaSet**

**metadata:**

**name:** **frontend**

**labels:**

**app:** **guestbook**

**tier:** **frontend**

**spec:**

**# modify replicas according to your case**

**replicas:** **3**

**selector:**

**matchLabels:**

**tier: frontend**

**template:**

**metadata:**

**labels:**

**tier:** **frontend**

**spec:**

**containers:**

**-** **name:** **php-redis**

**image:** **gcr.io/google\_samples/gb-frontend:v3**

Reference: https://Kubernetes.io/docs/concepts/workloads/controllers/replicaset/

**[How do you tie service to a pod or to a set of pods?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2414)**

By declaring pods with the label(s) and by having a selector in the service which acts as a glue to stick the service to the pods.

**kind:** **Service**

**apiVersion:** **v1**

**metadata:**

**name:** **my-service**

**spec:**

**selector:**

**app: MyApp**

**ports:**

**-** **protocol:** **TCP**

**port:** **80**

Let's say if we have a set of **Pods** that carry a label **"app=MyApp"** the service will start routing to those pods.

**[Having a Pod with two containers, can I ping each other? like using the container name?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3506)**

Containers on same pod act as if they are on the same machine. You can ping them using localhost:port itself. Every container in a pod shares the same IP. You can `ping localhost` inside a pod. Two containers in the same pod share an IP and a network namespace and They are both localhost to each other. Discovery works like this: Component A's pods -> Service Of Component B -> Component B's pods and Services have domain names servicename.namespace.svc.cluster.local, the dns search path of pods by default includes that stuff, so a pod in namespace Foo can find a Service bar in same namespace Foo by connecting to `bar`

**[Does the rolling update with state full set replicas =1 makes sense?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3507)**

No, because there is only 1 replica, any changes to state full set would result in an outage. So rolling update of a StatefulSet would need to tear down one (or more) old pods before replacing them. In case 2 replicas, a rolling update will create the second pod, which it will not be succeeded, the PD is locked by first (old) running pod, the rolling update is not deleting the first pod in time to release the lock on the PDisk in time for the second pod to use it. If there's only one that rolling update goes 1 -> 0 -> 1.f the app can run with multiple identical instances concurrently, use a Deployment and roll 1 -> 2 -> 1 instead.

**[Different Ways to provide API-Security on Kubernetes?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3508)**

Use the correct auth mode with API server authorization-mode=Node,RBAC

Ensure all traffic is protected by TLS

Use API authentication (smaller cluster may use certificates but larger multi-tenants may want an AD or some OIDC authentication).

Make kubeless protect its API via authorization-mode=Webhook

Make sure the kube-dashboard uses a restrictive RBAC role policy

Monitor RBAC failures

Remove default ServiceAccount permissions

Filter egress to Cloud API metadata APIs

Filter out all traffic coming into kube-system namespace except DNS

**[what does kube-proxy do?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3509)**

kube-proxy does 2 things

* for every Service, open a random port on the node and proxy that port to the Service.
* install and maintain iptables rules which capture accesses to a virtual ip:port and redirect those to the port in (1)

The kube-proxy is a component that manages host sub-netting and makes services available to other components.Kubeproxy handles network communication and shutting down master does not stop a node from serving the traffic and kubeproxy works, in the same way, using a service. The iptables will route the connection to kubeproxy, which will then proxy to one of the pods in the service.kube-proxy translate the destination address to whatever is in the endpoints.

**[What runs inside the kubernetes worker nodes?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3510)**

Container Runtime

* Kubelet
* kube-proxy

Kubernetes Worker node is a machine where workloads get deployed. The workloads are in the form of containerised applications and because of that, every node in the cluster must run the container run time such as docker in order to run those workloads. You can have multiple masters mapped to multiple worker nodes or a single master having a single worker node. Also, the worker nodes are not gossiping or doing leader election or anything that would lead to odd-quantities. The role of the container run time is to start and managed containers. The kubelet is responsible for running the state of each node and it receives commands and works to do from the master. It also does the health check of the nodes and make sure they are healthy. Kubelet is also responsible for metric collectins of pods as well. The kube-proxy is a component that manages host subnetting and makes services available to other components.

A default deny policy on all inbound on all namespaces is good practice. You explicitly allow per deployment.

Use a podsecurity policy to have container restrictions and protect the Node

Keep kube at the latest version.

**[Is there a way to make a pod to automatically come up when the host restarts?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3511)**

Yes using replication controller but it may reschedule to another host if you have multiple nodes in the cluster

A replication controller is a supervisor for long-running pods. An RC will launch a specified number of pods called replicas and makes sure that they keep running. Replication Controller only supports the simple map-style `label: value` selectors. Also, Replication Controller and ReplicaSet aren't very different. You could think of ReplicaSet as Replication Controller. The only thing that is different today is the selector format. If pods are managed by a replication controller or replication set you can kill the pods and they'll be restarted automatically. The yaml definition is as given below:

apiVersion: v1

kind: ReplicationController

metadata:

name: test

spec:

replicas: 3

selector:

app: test

template:

metadata:

name: test

labels:

app: test

spec:

containers:

name: test

image: image/test

ports:

containerPort: 80

**[Is there any other way to update configmap for deployment without pod restarts?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3512)**

well you need to have some way of triggering the reload. ether do a check every minute or have a reload endpoint for an api or project the configmap as a volume, could use inotify to be aware of the change. Depends on how the configmap is consumed by the container. If env vars, then no. If a volumeMount, then the file is updated in the container ready to be consumed by the service but it needs to reload the file

The container does not restart. if the configmap is mounted as a volume it is updated dynamically. if it is an environment variable it stays as the old value until the container is restarted.volume mount the configmap into the pod, the projected file is updated periodically. NOT realtime. then have the app recognise if the config on disk has changed and reload

**[Do rolling updates declared with a deployment take effect if I manually delete pods of the replica set with kubectl delete pods or with the dashboard? Will the minimum required a number of pods be maintained?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3513)**

Yes, the scheduler will make sure (as long as you have the correct resources) that the number of desired pods are met. If you delete a pod, it will recreate it. Also deleting a service won't delete the Replica set. if you remove Service or deployment you want to remove all resources which Service created. Also having a single replica for a deployment is usually not recommended because you cannot scale out and are treating in a specific way

Any app should be `Ingress` -> `Service` -> `Deployment` -> (volume mount or 3rd-party cloud storage)

You can skip ingress and just have `LoadBalancer (service)` -> `Deployment` (or Pod but they don't auto restart, deployments do)

**[what is the difference between externalIP and loadBalancerIP ?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3514)**

loadBalancerIP is not a core Kubernetes concept, you need to have a cloud provider or controller like metallb set up the loadbalancer IP. When MetalLB sees a Service of type=LoadBalancer with a ClusterIP created, MetalLB allocates an IP from its pool and assigns it as that Service's External LoadBalanced IP.the externalIP, on the other hand, is set up by kubelet so that any traffic that is sent to any node with that externalIP as the final destination will get routed.`ExternalIP` assumes you already have control over said IP and that you have correctly arranged for traffic to that IP to eventually land at one or more of your cluster nodes and its is a tool for implementing your own load-balancing. Also you shouldn't use it on cloud platforms like GKE, you want to set `spec.loadBalancerIP` to the IP you preallocated. When you try to create the service using .`loadBalancerIP` instead of `externalIP`, it doesn't create the ephemeral port and the external IP address goes to `<pending>` and never updates.

**[In  Kubernetes - A Pod is running 2 containers, when One container stops - another Container is still running, on this event, I want to terminate this Pod?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3515)**

You need to add a liveness and readiness probe to query each container,  if the probe fails, the entire pod will be restarted .add liveness object that calls any api that returns 200 to you from another container and both liveness and readiness probes run in infinite loops for example, If X depended to Y So add liveness  in X that check the health of Y.Both readiness/liveness probes always have to run after the container has been started .kubelet component performs the liveness/readiness checks and set initialDelaySeconds and it can be anything from a few seconds to a few minutes depending on app start time. Below is the configuration spec

livenessProbe spec:

livenessProbe:

httpGet:

path: /path/test/

port: 10000

initialDelaySeconds: 30

timeoutSeconds: 5

readinessProbe spec:

readinessProbe:

httpGet:

path: /path/test/

port: 10000

initialDelaySeconds: 30

timeoutSeconds: 5

**[what is the ingress, is it something that runs as a pod or on a pod?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3516)**

An ingress is an object that holds a set of rules for an ingress controller, which is essentially a reverse proxy and is used to (in the case of nginx-ingress for example) render a configuration file. It allows access to your Kubernetes services from outside the Kubernetes cluster. It holds a set of rules. An Ingress Controller is a controller. Typically deployed as a Kubernetes Deployment. That deployment runs a reverse proxy, the ingress part, and a reconciler, the controller part. the reconciler configures the reverse proxy according to the rules in the ingress object. Ingress controllers watch the k8s api and update their config on changes. The rules help to pass to a controller that is listening for them. You can deploy a bunch of ingress rules, but nothing will happen unless you have a controller that can process them.

LoadBalancer service -> Ingress controller pods -> App service (via ingress) -> App pods

**[What happens if  daemonset can be set to listen on a specific interface since the Anycast IP will be assigned to a network interface alias](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3517)**

Yes, hostnetwork for the daemonset gets you to the host, so an interface with an Anycast IP should work. You'll have to proxy the data through the daemonset.Daemonset allows you to run the pod on the host network, so anycast is possible.Daemonset allows us to run the pod on the host network At the risk of being pedantic, any pod can be specified to run on the host network.  The only thing special about DaemonSet is you get one pod per host. Most of the issues with respect to IP space is solved by daemonsets. As kube-proxy is run as daemonset, the node has to be Ready for the kube-proxy daemonset to be up.

**[How to forward port `8080 (container) -> 8080 (service) -> 8080 (ingress) -> 80 (browser)` how is it done?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3518)**

The ingress is exposing port 80 externally for the browser to access, and connecting to a service that listens on 8080. The ingress will listen on port 80 by default. An "ingress controller" is a pod that receives external traffic and handles the ingress  and is configured by an ingress resource For this you need to configure ingress selector and if no 'ingress controller selector' is specified then no ingress controller will control the ingress.

simple ingress Config will look like

host: abc.org

http:

paths:

backend:

serviceName: abc-service

servicePort: 8080

Then the service will look like

kind: Service

apiVersion: v1

metadata:

name: abc-service

spec:

ports:

protocol: TCP

port: 8080 # this is the port the service listens on

targetPort: 8080

**[Are deployments with more than one replica automatically doing rolling updates when a new deployment config is applied?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3519)**

The Deployment updates Pods in a rolling update fashion when .spec.strategy.type==RollingUpdate .You can specify maxUnavailable and maxSurge to control the rolling update process. Rolling update is the default deployment strategy.kubectl rolling-update updates Pods and ReplicationControllers in a similar fashion. But, Deployments are recommended, since they are declarative, and have additional features, such as rolling back to any previous revision even after the rolling update is done.So for rolling updates to work as one may expect, a readiness probe is essential. Redeploying deployments is easy but rolling updates will do it nicely for me without any downtime. The way to make a  rolling update of a Deployment and kubctl apply on it is as below

spec:

minReadySeconds: 180

replicas: 9

revisionHistoryLimit: 20

selector:

matchLabels:

deployment: standard

name: standard-pod

strategy:

rollingUpdate:

maxSurge: 1

maxUnavailable: 1

type: RollingUpdate

**[If you have multiple containers in a Deployment file, does use the HorizontalPodAutoscaler scale all of the containers?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3520)**

Yes, it would scale all of them, internally the deployment creates a replica set (which does the scaling), and then a set number of pods are made by that replica set. the pod is what actually holds both of those containers. and if you want to scale them independently they should be separate pods (and therefore replica sets, deployments, etc).so for hpa to work You need to specify min and max replicas  and the threshold what percentage of cpu and memory you want your pods to autoscale..without having the manually run kubectl autoscale deployment ,you can use the below yaml file to do the same

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

annotations:

name: app

spec:

maxReplicas: 15

minReplicas: 10

scaleTargetRef:

apiVersion: autoscaling/v1

kind: Deployment

name: app targetCPUUtilizationPercentage: 70

**[Suppose you have to use database with your application but well, if you make a database container-based deployment. how would the data persist?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3521)**

Deployments are for stateless services, you want to use a StatefulSet or just define 3+ pods without a replication controller at all. If you care about stable pod names and volumes, you should go for StatefulSet.Using statefulsets you can maintain which pod is attached to which disk.StatefulSets make vanilla k8s capable of keeping Pod state (things like IPs, etc) which makes it easy to run clustered databases. A stateful set is a controller that orchestrates pods for the desired state. StatefulSets formerly known as PetSets will help for the database if hosting your own. Essentially StatefulSet is for dealing with applications that inherently don't care about what node they run on, but need unique storage/state.

**[If a pod exceeds its memory "limit" what signal is sent to the process?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3522)**

SIGKILL as immediately terminates the container and spawns a new one with OOM error. The OS, if using a cgroup based containerisation (docker, rkt, etc), will do the OOM killing. Kubernetes simply sets the cgroup limits but is not ultimately responsible for killing the processes.`SIGTERM` is sent to PID 1 and k8s waits for (default of 30 seconds) `terminationGracePeriodSeconds` before sending the `SIGKILL` or you can change that time with terminationGracePeriodSeconds in the pod. As long as your container will eventually exit, it should be fine to have a long grace period. If you want a graceful restart it would have to do it inside the pod. If you don't want it killed, then you shouldn't set a memory `limit` on the pod and there's not a way to disable it for the whole node. Also, when the liveness probe fails, the container will SIGTERM and SIGKILL after some grace period.

**ADVANCED**

**[Let’s say a Kubernetes job should finish in 40 seconds, however on a rare occasion it takes 5 minutes, How can I make sure to stop the application if it exceeds more than 40 seconds?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2415)**

When we create a job spec, we can give --activeDeadlineSeconds flag to the command, this flag relates to the duration of the job, once the job reaches the threshold specified by the flag, the job will be terminated.

**kind: CronJob**

**apiVersion: batch/v1beta1**

**metadata:**

**name: mycronjob**

**spec:**

**schedule: "\*/1 \* \* \* \*"**

**activeDeadlineSeconds: 200**

**jobTemplate:**

**metadata:**

**name: google-check-job**

**spec:**

**template:**

**metadata:**

**name: mypod**

**spec:**

**restartPolicy: OnFailure**

**containers:**

**- name: mycontainer**

             image: alpine

             command: ["/bin/sh"]

             args: ["-c", "ping -w 1 google.com"]

**[How do you test a manifest without actually executing it?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2416)**

use --dry-run flag to test the manifest. This is really useful not only to ensure if the yaml syntax is right for a particular Kubernetes object but also to ensure that a spec has required key-value pairs.

kubectl create -f <test.yaml> --dry-run

Let us now look at an example Pod spec that will launch an nginx pod

**○ → cat example\_pod.yaml**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: my-nginx**

**namespace: mynamespace**

**spec:**

**containers:**

**- name: my-nginx**

**image: nginx**

**○ → kubectl create -f example\_pod.yaml --dry-run**

**pod/my-nginx created (dry run)**

**[How do you initiate a rollback for an application?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2417)**

Rollback and rolling updates are a feature of Deployment object in the Kubernetes. We do the Rollback to an earlier Deployment revision if the current state of the Deployment is not stable due to the application code or the configuration. Each rollback updates the revision of the Deployment

**○ → kubectl get deploy**

**NAME    DESIRED  CURRENT UP-TO-DATE   AVAILABLE AGE**

**nginx   1  1 1            1 15h**

**○ → kubectl rollout history deploy nginx**

**deployment.extensions/nginx**

**REVISION  CHANGE-CAUSE**

**1         <none>**

**2         <none>**

**kubectl undo deploy <deploymentname>**

**○ → kubectl rollout undo deploy nginx**

**deployment.extensions/nginx**

**○ → kubectl rollout history deploy nginx**

**deployment.extensions/nginx**

**REVISION  CHANGE-CAUSE**

**2         <none>**

**3         <none>**

We can also check the history of the changes by the below command

**kubectl rollout history deploy <deploymentname>**

**[How do you package Kubernetes applications?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2418)**

Helm is a package manager which allows users to package, configure, and deploy applications and services to the Kubernetes cluster.

helm init  # when you execute this command client is going to create a deployment in the cluster and that deployment will install the tiller, the server side of Helm

The packages we install through client are called charts. They are bundles of templatized manifests. All the templating work is done by the Tiller

helm search redis **# searches for a specific application**

helm install stable/redis **# installs the application**

helm ls **# list the applications**

**[What are init containers?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2419)**

Generally, in Kubenetes, a pod can have many containers. Init container gets executed before any other containers run in the pod.

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: myapp-pod**

**labels:**

**app: myapp**

**annotations:**

**pod.beta.Kubernetes.io/init-containers: '[**

**{**

**"name": "init-myservice",**

**"image": "busybox",**

**"command": ["sh", "-c", "until nslookup myservice; do echo waiting for myservice; sleep 2; done;"]**

**}**

**]'**

**spec:**

**containers:**

**- name: myapp-container**

**image: busybox**

**command: ['sh', '-c', 'echo The app is running! && sleep 3600']**

**[What is node affinity and pod affinity?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2420)**

1. Node Affinity ensures that pods are hosted on particular nodes.

Pod Affinity ensures two pods to be co-located in a single node.

**Node Affinity**

**apiVersion:** **v1**

**kind:** **Pod**

**metadata:**

**name:** **with-node-affinity**

**spec:**

**affinity:**

**nodeAffinity:**

**requiredDuringSchedulingIgnoredDuringExecution:**

**nodeSelectorTerms:**

**- matchExpressions:**

**- key: Kubernetes.io/e2e-az-name**

**operator: In**

**values:**

**- e2e-az1**

**Pod Affinity**

**apiVersion:** **v1**

**kind:** **Pod**

**metadata:**

**name:** **with-pod-affinity**

**spec:**

**affinity:**

**podAffinity:**

**requiredDuringSchedulingIgnoredDuringExecution:**

**- labelSelector:**

**matchExpressions:**

**- key: security**

**operator: In**

**values:**

**- S1**

The pod affinity rule says that the pod can be scheduled to a node only if that node is in the same zone as at least one already-running pod that has a label with key “security” and value “S1”

**Reference:** https://Kubernetes.io/docs/concepts/configuration/assign-pod-node/

**[How do you drain the traffic from a Pod during maintenance?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2421)**

When we take the node for maintenance, pods inside the nodes also take a hit. However, we can avoid it by using the below command

kubectl drain <nodename>

When we run the above command it marks the node unschedulable for newer pods then the existing pods are evicted if the API Server supports eviction else it deletes the pods

Once the node is up and running and you want to add it in rotation we can run the below command

kubectl uncordon <nodename>

Note: If you prefer not to use kubectl drain (such as to avoid calling to an external command, or to get finer control over the pod eviction process), you can also programmatically cause evictions using the eviction API.

More info: https://Kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/

**[I have one POD and inside 2 containers are running one is Nginx and another one is  wordpress So, how can access these 2 containers from the Browser with IP address?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3497)**

Just do port forward

kubectl port-forward [nginx-pod-name] 80:80

kubectl port-forward [wordpress-pod-name] drupal-port:wordpress-port

To make it permanent, you need to expose those through nodeports whenever you do kubectl port forward it adds a rule to the firewall to allow that traffic across nodes but by default that isn’t allowed since flannel or firewall probably blocks it.proxy tries to connect over the network of the apiserver host as you correctly found, port-forward on the other hand is a mechanism that the node kubelet exposes over its own API

**[If I have multiple containers running inside a pod, and I want to wait for a specific container to start before starting another one.](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3498)**

One way is  Init Containers are for one-shot tasks that start, run, end; all before the next init container or the main container start, but  if a client in one container wants to consume some resources exposed by some server provided by another container or If the server  ever crashes or is restarted, the client will need to retry connections. So the client can retry always, even if the server isn't up yet. The best way is sidecar pattern\_ are where one container is the Main one, and other containers expose metrics or logs or encrypted tunnel or somesuch. In these cases, the other containers can be killed when the Main one is done/crashed/evicted.

**[What is the impact of upgrading kubelet if we leave the pods on the worker node - will it break running pods? why?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3499)**

Restarting kubelet, which has to happen for an upgrade will cause all the Pods on the node to stop and be started again. It’s generally better to drain a node because that way Pods can be gracefully migrated, and things like Disruption Budgets can be honored. The problem is that `kubectl` keeps up with the state of all running pods, so when it goes away the containers don’t necessarily die, but as soon as it comes back up, they are all killed so `kubectl` can create a clean slate. As kubelet communicates with the apiserver, so if something happens in between of upgrade process, rescheduling of pods may take place and health checks may fail in between the process. During the restart, the kubelet will stop querying the API, so it won’t start/stop containers, and Heapster won’t be able to fetch system metrics from cAdvisor. Just make sure it’s not down for too long or the node will be removed from the cluster!

**[How service that selects apps based on the label and has an externalIP?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3500)**

The service selects apps based on labels, so if no pods have appropriate labels, the service has nothing to route and labels can be anything you like. Since all pod names should be unique, you can just set the labels as the pod name. Since statesets create the same pods multiple times, they won't be configured with distinct labels you could use to point disparate services to the correct pod. If you gave the pods their own labels manually it will work. Also, service selects pods based on selector as well their location label as well Below .yaml file of Grafana dashboard service shows the same

apiVersion: v1

kind: Service

metadata:

name: grafanaportforward

namespace: kubeflow

labels:

run: grafana-test

spec:

ports:

- port: 3000

protocol: TCP

name: grafana

externalIPs:

- x.y.x.q

selector:

app: grafana-test

**[Does the container restart When applying/updating the secret object (kubectl apply -f mysecret.yml)?  If not, how is the new password applied to the database?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3501)**

If you are mounting the secret as a volume into your pod, when the secret is updated the content will be updated in your pod, without the pod restarting. It's up to your application to detect that change and reload, or to write your own logic that rolls the pods if the secret changes .volumeMount controls what part of the secret volume is mounted into a particular container (defaults to the root, containing all those files, but can point to a specific file using `subPath`), and where in the container it should be mounted with `mountPath`.Example spec is below

volumeMounts:

- readOnly: true

mountPath: /certs/server

name: my-new-server-cert

volumes:

- name: server-cert

secret:

secretName: mysecret

Also, it depends on how the secret is consumed by a container. If env vars, then no. If a volumeMount, then the file is updated in the container ready to be consumed by the service but it needs to reload the file. The container does not restart. if the secret is mounted as a volume it is updated dynamically. if it is an environment variable it stays as the old value until the container is restarted

**[How should you connect an app pod with a database pod?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3502)**

By using a service object. reason being, if the database pod goes away, it's going to come up with a different name and IP address.  Which means the connection string would need to be updated every time, managing that is difficult. The service proxies traffic to pods and it also helps in load balancing of traffic if you have multiple pods to talk to. It has its own IP and as long as service exists pod referencing this service in upstream will work and if the pods behind the service are not running, a pod will not see that and will try to forward the traffic but it will return a 502 bad gateway.So just defined the Service and then bring up your Pods with the proper label so the Service will pick them up.

**[How to configure a default ImagePullSecret for any deployment?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3503)**

You can attach an image pull secret to a service account. Any pod using that service account (including default) can take advantage of the secret.you can bind the pullSecret to your pod, but you’re still left with having to create the secret every time you make a namespace.

imagePullSecrets:

name: test

Also, you can  Create the rc/deployment manually and either specify the imagepullsecret or a service account that has the secret or add the imagepullsecret to the default service account, in which case you'd be able to use `kubectl run` and not have to make any manual changes to the manifest. Depending on your environment and how secret this imagepullsecret is, will change how you approach it.

**[I have a configmap for 3 files that are going to be mounted in supposing "fluentd/etc/" and the respective files would be fluent.conf,  kubernetes.conf, systemd.conf, config map in deployment.yaml is like this  
  
volumeMounts:  
  
name: fluentd  
  
mountPath: /fluentd/etc  
  
name: varlog  
  
mountPath: /var/log  
  
name: container1  
  
mountPath: /var/lib/docker/containers  
  
readOnly: true  
  
securityContext:  
  
privileged: true  
  
terminationGracePeriodSeconds: 30  
  
volumes:  
  
name: varlog  
  
hostPath:  
  
path: /var/log  
  
name: container1  
  
hostPath:  
  
path: /var/lib/docker/containers  
  
name:  fluentd  
  
configMap:  
  
name: fluentd-config  
  
When deploying you will get an error of mounting as read-only, which is effecting to fluent to read some of the mentioned sources in the configmap.how can we avoid this read-only error?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3504)**

configmaps are always mounted read-only. if you need to modify a configmap in a pod, you should copy it from the configmap mount to a regular file in the pod and then modify it. To solve this issue we should use an init container to mount the configmap, copy the configmap into an `emptyDir` volume and share the volume with the main container.

configmaps are mounted read-only so that you can't touch the files. when the master configmap changes the mounted file also changes. so if you were to modify the local mounted file, it would be overwritten anyways.

**[If you have a pod that is using a ConfigMap which you updated, and you want the container to be updated with those changes, what should you do?](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-3505)**

if the config map is mounted into the pod as a volume, it will automatically update not instantly and the files will change inside the container. If it is an environment variable it stays as the old value until the container is restarted

For example: create a new config.yaml with your custom values

apiVersion: v1

kind: ConfigMap

metadata:

name: testconfig

namespace: default

data:

config.yaml: |

namespaces:

default

labels:

"app"

"owner"

Then create a pod definition, referencing the ConfigMap

apiVersion: v1

kind: Pod

metadata:

name: testobject

spec:

serviceAccountName: testobject

containers:

name: testobject

image: test/appv1

volumeMounts:

name: config-volume

mountPath: /app/config.yaml

subPath: config.yaml

volumes:

name: config-volume

configMap:

name: testconfig

restartPolicy: Never

**CHEF INTERVIEW**

Chef, a company and the creator of a configuration management tool, is an automation tool that presents a way to explain infrastructure as code. Infrastructure as code (IAC) indicates managing infrastructure by writing code (Automating infrastructure) rather than using manual processes. It can also be termed as programmable infrastructure. Chef uses a pure-Ruby, domain-specific language (DSL) for writing system configurations. is written in Ruby and Erlang. It utilises a pure-Ruby, domain-specific language (DSL) for writing system configuration "recipes". Chef is proficient in streamlining the task of configuring and maintaining a company's servers and integrating with cloud-based platforms such as Internap, Amazon EC2, Google Cloud Platform, Oracle Cloud, OpenStack, SoftLayer, Microsoft Azure, and Rackspace to automatically provision and configure new machines. Chef’s versatile and affordable features make it a favourite in providing solutions for both small and large scale systems.

Chef’s business is expanding exponentially with annual recurring revenue increase expediting as the company scales. More than half of the Fortune 50 has Chef as one of their prime applications and 80 per cent of Chef’s revenue comes from enterprise businesses.

Top companies like HP, Amazon Web Services, Hewlett-Packard, Facebook, IBM, Microsoft, Oracle and Cisco use Chef as one of the software in their workflow and are in pursuit of Chef Developer and Full Stack engineers.

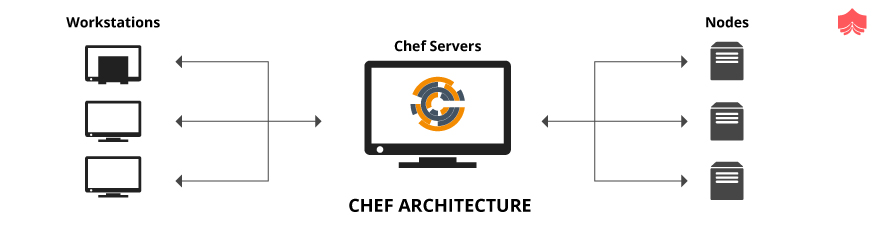
None of us can consider Interviews as a piece of cake. However, it becomes easy to handle your interviews if you are prepared with these Chef interview questions for experienced and freshers to help you in pursuing your dream career. Responding quickly and effectively to questions asked by the employers is a must in an interview. As these interview questions on Chef are very obvious, your prospective recruiters will anticipate you to be able to answer the same. These Chef interview questions and answers will boost your spirit required to ace the interview.

These Chef programming interview questions will help you to get your dream job either in Data Science, Machine Learning or just Chef coding. These Chef interview questions and answers for experienced professionals will undoubtedly raise your confidence to handle any relevant interview and will prepare you to answer the most difficult questions in the most ideal way. Suggested by experts, these Chef developer interview questions have proven to be of great value.

Both the job aspirants and even the recruiters may refer to these Chef basic interview questions in order to know the appropriate questions they need to assess a candidate.

Make a mark in your career with the next Chef interview. Create your own destiny and all the best.

**Explain the Architecture of Chef.**



Chef architecture has three main components, Chef server, Chef node, and Chef workstation. At the workstation, code for provisioning and managing the infrastructure is created and tested. This code is implemented as recipes and a collection of recipes is known as a cookbook. Once the cookbook is ready, it is pushed to the Chef server from the workstation. We use the “knife” command-line tool or Berkshelf” to upload the cookbook. Chef Server typically acts as a hub for the infrastructure code. Chef node could be any system that is managed by Chef. It could be an actual or virtual or cloud machine.

Chef nodes communicate with the Chef server using an application called “chef-client”.Chef architecture works on pull configuration that means we don’t have to perform any actions on the Chef server to push the configurations to Chef nodes.“The chef-client” pulls down all the data needed for the Chef node to meet the desired criteria. This is also referred to as Chef run. During Chef run, a tool called OHAI collects the current status of the attributes of the node.

**How Chef differs from Ansible & Puppet?**

**Chef:**

* Has a client and server architecture. Apart from a Chef Client and Server, there is one another important component called a Chef Workstation.
* The Chef workstation and Chef client could be windows or Linux/Unix machines, but the Chef server supports only Linux/Unix.
* The Chef configuration management tool is in the market for a long time hence is stable, mature and has good community support
* Chef uses the pull configuration
* Chef uses an imperative language written using the ruby DSL (Domain Specific Language)
* Chef uses a command-line tool called knife for communication.
* Ideal for developers with advanced programming skills

**Puppet:**

* Has an agent/master architecture.
* Puppet agent supports both Windows and Linux/Unix whereas puppet master support only Linux/Unix
* Puppet is also in the market for a long time hence is stable, mature and has good community support
* Puppet also uses the pull configuration
* Puppet uses a declarative language written using the puppet DSL (Domain Specific Language)
* Puppet uses SSL (Secure sockets layer) for communication
* Ideal for developers with less programming skills

**Ansible:**

* Ansible is agentless where no software needs to be  installed on client systems.
* Ansible server supports only Linux/Unix
* Ansible is a new player in the market but shows a promising growth trend
* Ansible uses the push configuration
* Ansible uses YAML (Ain’t Mark-up Language)
* Ansible uses SSH(Secure socket shell) for communication
* Ideal for developers with basic programming skills

**What is Test Kitchen? Explain its importance?**

Test Kitchen is an in-built Chef tool to test your recipes in a sandbox environment before moving it to production. We know that Chef does the provisioning and managing of infrastructure through code. A defect in the code can even change the production environment so we need an isolated environment where we can test our infrastructure code without worrying about the after-effects and the Test kitchen provides this. The test kitchen also helps us to test our infrastructure code on any platform by creating an instance with the respective OS. System tests using InSpec is often performed using Test kitchen as it helps us to test all actions in a sandbox environment. For eg: installation of the “Apache “ server on a Windows system is tested in Test kitchen by actually installing it on a provisioned Windows system.

.kitchen.yml created while generating a cookbook is the configuration file for the Test Kitchen. The test kitchen creates a sandbox environment using vagrant or docker.  We can create multiple kitchen instances using a single .kitchen.yml file. The Test Kitchen is now often used in the CI pipeline of the applications to run the tests faster.

**What is Chef Supermarket?**

Chef Supermarket is a repository for cookbooks shared publicly. It’s a repository similar to GitHub. We can have two types of supermarket, private and other public. Public Chef Supermarket is hosted by Chef Software. We can find many general cookbooks like for eg: Nginx in public Chef supermarket. If any of the publicly shared cookbooks meet our requirements we can use them from the supermarket. We often come across situations where we need to modify the behavior of community cookbooks to achieve our requirements. This could be done by creating wrappers. So Chef supermarket is a central repository where one can share, download or collaborate on cookbooks.

Many companies prefer to create a private supermarket to share their company-specific cookbooks internally. Organisations whose cookbooks have security-related data prefer to only use private Chef Supermarket. They install a private supermarket by using “supermarket-omnibus-cookbook” available in Chef public supermarket. Retrieval of a cookbook in a private supermarket is faster compared to the public supermarket as we have fewer cookbooks to search from and due to its proximity.

**What Happens in a Chef Client Runs?**

Chef Client Run (CCR) is performed on each node by following several steps to ensure that the Chef node reaches its desired state. The different steps include

1. **Get node data** – Here the "client.rb" file present in Chef node is read and retrieves the Chef node data through the OHAI processes. The most important data acquired here is the node name.
2. **Authentication** – Authenticates the Chef node with the Chef server using the RSA private key.
3. **Get and rebuild the node object** - Once authenticated the node object is pulled down from the Chef Server and then rebuilt. If it is the first client run no node object will be present on Chef Server.
4. **The run list expanded** - Run list of the rebuilt node object is expanded to get a complete list of roles and recipes that will be applied to Chef node.
5. **Sync cookbooks** - The Chef node also checks with the server for all the cookbooks required to perform the run list. If the cookbooks don’t match with the ones already cached in the node through previous client runs, then the changed files or new files are downloaded.
6. **Reset node object** - All attributes in the node object is set according to attribute precedence.
7. **Load cookbooks and collect resources** – The cookbooks are then loaded to the memory, analyzed closely and all the resources required to perform the operations are collected.
8. **Converge** – This is where the actual execution of the resources collected takes place on the Chef node.
9. **Update node object** - Node object in the Chef server is updated with the node object built during the Chef Client run.
10. **Chef run wait** - Chef client run waits for the next time it’s run.

**Explain Kitchen Test. Kitchen converge and kitchen verify commands.**

The “kitchen test” command is used to fully test the cookbooks in a CI/CD pipeline and is an elaborate process consisting of creating a new instance, converging it, verifying and then destroying the instance. Since this is an elaborate process it is not suited to be used in the development phase of a cookbook. "kitchen converge" and "kitchen verify" commands are more useful for the cookbook development phase.

“Kitchen converge” applies the cookbook to the existing sandbox environment. It will install the chef-client, load the cookbooks and other necessary files and goes ahead with the first Chef client run using kitchen.yml attributes. “Kitchen converge” has two types of return codes. A zero denotes success and a non zero value denotes failure at any step. If “kitchen converge” fails, try to delete the kitchen instance with “kitchen destroy” command and then run “kitchen converge” once again.

“Kitchen verify” helps in validating the state of the kitchen instance against test suites specified in .kitchen.yml file. Often "kitchen converge" and "kitchen verify" are frequently used to develop infrastructure code using Test Driven Development.

**Explain the purpose of Metadata.rb file?**

“metadata.rb” file is at the top level of every cookbook directory and gets created as soon as a cookbook is created. (For JAVA programmers the concept is similar to a constructor getting created when an object is created!) It specifies all the dependencies required to correctly deploy a cookbook to its nodes. The Chef server compiles this metadata.rb file into metadata.json file and stores it. We can edit the metadata in the metadata.rb file and should be recompiled every time by the Chef server for the changes to be implemented. Metadata.rb file is compiled when

1. a cookbook is uploaded to the Chef server
2. when the cookbook metadata knife subcommand is run and
3. when "knife cookbook create" subcommand is run, then the knife creates metadata.rb file automatically.

Metadata.rb files have entries under ‘depends’ keyword which specifies the different dependencies required to successfully deploy the cookbook in a specific node. If these dependencies are not given properly we may often face errors while deploying cookbooks and Chef nodes would not be configured as expected. Apart from "depends", we have other details given in the metadata.rb file like license, version, gem, etc. which can be parametrized using different operators (=,>=, >, etc)

**What is the use of chefignore file?**

From Chef 0.10, a ‘chefignore’ file is used to ignore unwanted temporary files like swap files, version control files, build data files, etc while uploading to the Chef server. This file is could be available at any of the subfolders of your cookbook repository. This file helps us to stop uploading unwanted files or confidential files to Chef Server.

For eg: while using Vim editor, it leaves behind swap files. These are of no use in reusing the uploaded cookbooks. These will get uploaded into the Chef Server unless specified in the chefignore file. The following entry in the chefignore file stops all swap files created by Vim editor being uploaded. This helps us to stop accidentally uploading these swap files which is not related to the cookbook and not required for the correct functioning of the cookbook.

# vim

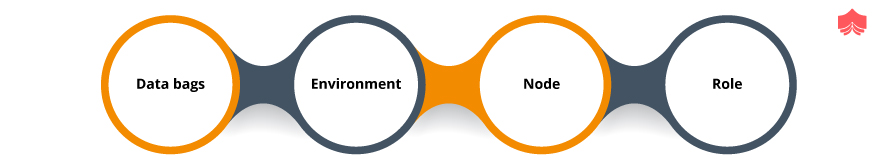
\*.sw[a-z]

Chefignore file can use \*,\*\* and? wildcards to specify the ignore patterns in chefignore file. The pattern matching in the file is concerning

* root of the cookbook
* Relative directory names

**Explain different search indexes in Chef to search node data.**

The different search indexes for node data in Chef Server are as follows:-



1. **Data bags:** Data bags are like a container where global data is stored and shared across the Chef nodes and these data are accessible by Chef Server. Data bags are used to securely store information like admin credentials etc in the form of JSON files. These are indexed and are loaded in a Chef search.
2. **Environment:** We at times want our nodes to converge to a testing environment or maybe to a production environment so that admins can know in which stage of the code development, the node is included. Environment details are also updated in Chef Server and could be searched for. “\_default” is the default environment for any Chef node. Multiple environments could be added and worked upon.’
3. **Node:** Node is a system that we want to be configured and maintained using Chef Infra Client. The node could be a physical server or a virtual machine or a cloud machine. Details of the node are also updated in Chef Server and searched upon.
4. **Role:** Role is used for defining certain patterns or processes across all Chef nodes. Multiple configurations of the nodes at a different time could be achieved in Chef using the “roles” feature.

**Differentiate Between Recipe & Cookbook.**

Recipes are the basic configuration element in any organizations. It’s a programmatic representation of the desired state of the nodes. Usually written in ruby with Chef Domain Specific Language (DSL), these specify all the standards (policies) or updates that the nodes should follow. Recipes are often a collection of resources that are the building blocks of the infrastructure of any organizations.

Cookbooks are a collection of recipes. Apart from recipes, cookbooks contain other elements like metadata, attributes, libraries, resources, templates, and tests which helps the nodes to achieve their desired state. Cookbooks can be created by Chef commands or by the Chef command-line tool called Knife. The cookbooks are the fundamental units of chefs and are uploaded to the Chef Servers and then read and deployed by the Chef Clients at the nodes. Cookbooks are standalone entities that could be transferred easily between the Chef server and node. It typically implements a scenario of infrastructure and has everything required for the scenario. Cookbooks could be version controlled and hence help the development team to collaborate and share cookbooks.

**Explain the difference between “Knife Upload” and “berkshelf upload” commands.**

The “knife upload” command is used to upload the cookbooks to the Chef server. It does not help in managing the cookbook dependencies. So knife command is not chosen for very complicated and large cookbooks with dependencies that in turn might be dependent on others and so on. For eg: Assume that a cookbook called “my-app” depends on community cookbook “nginx” which in turn depends on another community cookbook “ohai”. When we try to upload “my-app” to Chef Server using the “knife upload” command, we have to make sure all other dependent cookbooks are uploaded manually to Chef Server. This problem is solved using “berkshelf upload”

Berkshelf command is similar to knife command and helps to upload cookbooks to Chef Server. But it also helps in the management of cookbooks outside of Chef repository. This is quite useful in case of large and complex cookbooks with lots of dependencies. “berkshelf upload” will upload all dependent cookbooks to the Chef server automatically. All the dependent cookbooks are fetched using the “berks install” command of the Berkshelf tool.

Write & Explain a simple Inspec test.

InSpec is an in-built testing framework for testing and auditing infrastructure in Chef. A sample InSpec test to verify whether “Nginx” server is installed and running in a system could be written as follows:

*```control 'Nginx-install-1.0' do*

*title 'Check if nginx is installed'*

*describe package 'nginx' do*

*it { should be\_installed }*

*end*

*end*

*control 'Nginx should be running' do*

*describe service 'nginx' do*

*it { should be\_running }*

*end*

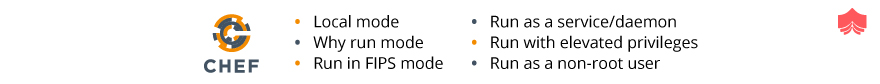
*end```*

Main components of InSpec tests are:

* **Control:** A requirement is defined inside a control block
* **Title:** Title defines the metadata of the control block describing the importance of the requirement.
* **Resource:** Resources are the basic building blocks of infrastructure. A control block may have multiple resources. Resources could be anything like file, package, service, etc.
* **Describe:** Tests for a particular resource is grouped into a describe block. A control block should have a minimum one describe block. A describe block is defined within a “do end” block
* **It or its:** Each line starting with "it" or "its" is a test to access resource-specific matcher. “should be\_installed” and “should be\_running” are in-built methods in DSL to verify whether the package is installed and whether the service is running.

Explain modes of the chef-client Run.

Chef Client run could be performed in the following modes:



1. **Local mode:** In this mode, we can run the Chef Client against the local Chef repository mimicking the behaviour as if it was running against a Chef Server. This is achieved through ‘Chef zero’ (previously a standalone tool)which acts as a lightweight Chef Server. Running in local mode is the easiest way to quickly test and verify the infrastructure code, though this has some disadvantages of not testing the authentication or security checks which are needed when we connect to the actual Chef Server
2. **Why run mode:** Why run mode gives you a preview of what will happen on your node when you perform Chef Client run. In why run mode no actual modifications are done on your node. This mode follows the principle of no-op(no operation). All actions that happen in normal Chef Client run is performed until the actual configuration of the node.
3. **Run in FIPS (Federal Information Processing Standards) mode:** Chef Client run can be performed in FIPS mode to follow the FIPS standards. Any violation of these standards will throw a run exception.
4. **Run as a service/daemon:** In this mode, we can make the Chef Client run happen in our desired time intervals. **Chef Infra Client** cookbook from GitHub repository is used to perform Chef Client run.
5. **Run with elevated privileges:** When we are trying to configure nodes with different OS like Windows or Linux we often run into some errors while performing Chef Client run, this is mostly due to the access privileges on these systems. The client run solves this issue by providing the ability to run the Chef Client with root access rather than by a local system user
6. **Run as a non-root user:** This mode is helpful when multiple teams (devs, administration, etc) are involved in your infrastructure development and not everyone needs the same amount of accesses. You can limit the access and make the Chef Client run as a non-root user.

**What are the various ways in which you can update a Chef run list for a Chef node?**

We can update the Chef run lists with the below two options:

**A) The Chef run list can be updated using the knife commands:**Knife is a command-line utility tool that securely communicates to the Chef Server. We have different commands for adding, removing and setting recipes in the run list.

For eg:

*```knife node run\_list add <NODE\_NAME> <RUN\_LIST\_ITEM> (options)```*commands could be used to add a recipe or a role in the run list of a node.

Similarly

*```knife node run\_list remove <NODE\_NAME> <RUN\_LIST\_ITEM> (options)```* command could be used to remove from the run list.

We can also specify the run list for a node while bootstrapping the node using the “knife bootstrap” command.

**B) We can also update the run list of a node through the Chef management console.** The Chef Management Console is a GUI interface through which we can manipulate the Chef run list. But it should be noted that as per the latest updates on Chef this interface is soon to be depreciated, and replaced with Chef Automate.

**[Differentiate unit tests and integration tests in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4101)**

Unit tests are easy to run simpler tests used to test a single unit or component of the infrastructure. Chef uses ChefSpec, an in-built testing framework to write unit tests for the infrastructure. In the infrastructure point of view unit tests are verified on the resource collection created during the Chef Client run and not by actually configuring the node.

For eg: If we want to write a unit test to verify if the “Apache” package is installed, no actual installation of the Apache server is tested but just verifies if the package “Apache” is added to the resource collection list. Resource collection will have a list of all actions that will be performed in the actual Chef Client run.

Integration tests in Chef's perspective are system tests where tests are performed by provisioning a new system and configuring the system to our requirements. This takes a lot of time as we have to perform the actual configuration of the provisioned system. System tests are also a costly affair compared to unit tests as we need to provision a new system exclusively for testing purposes.

**[Explain what is “file” resource in Chef and how it differs from “cookbook\_file” resource  and “template” resource in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4215)**

“file” resources in Chef are used to manage files in a Chef node during a chef-client run . A sample file resource is as follows:

```file ‘/tmp/index.html’ do

     content 'Hello World'

   end```

“file” resource has a name field that defines the resource block. In the above example ‘/tmp/index.html’ defines the name of the file which is managed by Chef with the path. Various actions performed by file resource is as follows:

* create : creation of a file
* delete : Delete a file
* touch : Changes the access time and modified time of the file
* nothing : Do no action unless notified by other resource
* create\_if\_missing : Create a file only if the file is not present in Chef node

“cookbook\_file” resource is used to copy files from the “files” folder of the Chef cookbook to a path in Chef node during a chef-client run. Sample “cookbook\_file” is specified as below:

```cookbook\_file '/var/index.php' do

source 'index.php'

action :create

  end```

Here file “index.php” from “files” folder in the cookbook is copied to “index.php” file in “/var” path in Chef node. If file not present it’s created at the path.If the file is present and if the checksum of the “index.php” file matches the “index.php” file in the cookbook of Chef Server, no file transfer is performed.

“template” resources are used to dynamically generate static files from an embedded Ruby template placed in the “templates” folder of a cookbook. For eg:

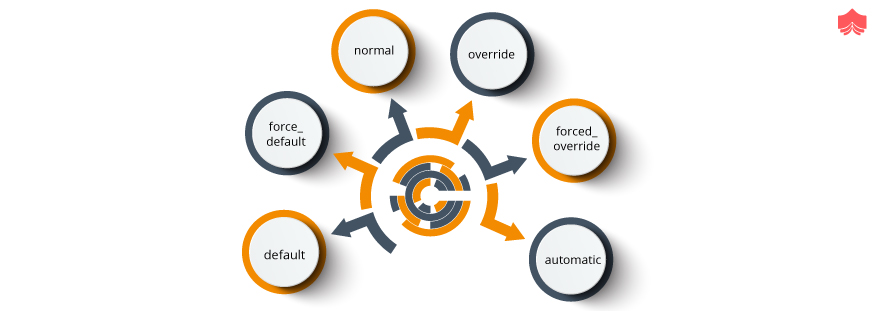
```template '/etc/motd' do

     source 'motd.erb'

 end```

“/etc/motd” is the location at which a new file is created on Chef node during a chef-client run  based on the template file “motd.erb” placed in the “templates” folder of the cookbook.A template file may contain Ruby statements and expressions unlike in “file” resource or “cookbook\_file” resource.

**[Explain different types of attributes in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4102)**



Attributes define a node! In other words, it’s a characteristic of your node when it runs in a specific role under a specific environment against a specific cookbook or recipe. Hence attributes can be found in all these Chef components such as roles, environment, cookbooks and even as a separate set called “attribute files”. The different types of attributes available in the chef are:-

1. **default:** This is the most generic type of attribute and also the one with the lowest precedence. These are largely used in cookbooks so that nodes remain the same way as they are and function further. At the start of every Chef Client run, the default attributes are reset automatically
2. **force\_default:** As the name suggests with the use of these attributes, the default attributes are forced to take up the new values assigned to them. The use of force\_default attributes comes when we want our nodes to run in a different role or environment.force\_default attributes defined in a cookbook have higher precedence than default attributes defined in a role or environment.
3. **normal:** Unlike default attributes, these are not reset during a new Chef Client run. They just persist in the node object.
4. **override:** Generally found in recipes and are used to override any default, force\_default or normal attributes. These are also reset before a  Chef Client run.
5. **forced\_override:** These attributes make sure that the override attributes defined in a cookbook are forced to take up the new values assigned to them.
6. **automatic:** These attributes are defined by Ohai to mainly understand the state of the node at the beginning of a Chef Client run. These have the highest precedence

**[Explain “knife ssl fetch” and “knife ssl check” commands.](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4103)**

In the Chef architecture, we have multiple Chef nodes managed by a Chef Server. Hence care must be taken to authenticate a Chef node with Chef Server. No stranger nodes should be able to send requests to Chef Server. So we use the “knife ssl fetch “ command to copy the self-signed SSL certificate of Chef Server to the trusted certificates directory in a respective node or a Chef Workstation. Only nodes or workstations with a valid SSL certificate could communicate with Chef Server without any errors. The URL for Chef Server is often specified in “client.rb” file in Chef node and “knife.rb” file in Chef Workstation. We can override this setting by supplying the URL of any Chef Server when the “knife ssl check” and “knife ssl fetch” command is run

“Knife ssl check” command is used to verify if the SSL certificate has a valid X.509 certificate property. “knife ssl check” and “knife ssl fetch” could be used to troubleshoot communication errors with Chef Server. This process is not valid for verifying communication with Hosted Chef Servers.

**[What is "execute" resources in Chef? Explain with an example.](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4104)**

execute resources in Chef is used to execute a single command. All commands executed via execute resources are not idempotent and are specific to the environment in which it’s run. We need to use guards like “not\_if”, “only\_if” to make the execution of the resource idempotent. During a Chef Client run, guards property verifies the state of the Chef node. "execute" resources could be run alone or in combination with other in-built Chef resources like templates, etc. "execute" resources could be used when we need to reuse existing shell scripts for infrastructure configurations. We need to specify the command to run the shell script in "execute" resource block.

For eg:

```execute 'apache\_test' do

     command '/usr/apachectl configtest'

end```

“apache\_test” is the name provided for the "execute" resource block. The actual command run is “/usr/apachectl configtest”.

Execute resource could have only two actions, nothing and run. If the action specified is "run", the command provided is executed. If the action specified is nothing, the command provided in the "execute" block is prohibited from running. script resource is often confused with execute resource. Script resource is used to execute a script using an interpreter provided like bash etc

**[What are the roles in Chef? Why do we use them?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4105)**

Chef nodes in your infrastructure may perform different roles at different times. For eg: Few of the Chef nodes would be web servers and one would be a load balancer. Grouping of the nodes into web servers and load balancer could be achieved in Chef using “roles” feature. The roles are defined through the role attributes. You can assign roles to your identical servers and all of them can go ahead and run the same run list mapped to the corresponding role. This helps in avoiding the process of running run lists manually on each node each time when you have many nodes performing the same function.

For configuring a new role for our node, we can make use of Knife commands. As a prerequisite, any previous roles or recipes on the run list can be removed using the “knife node run-list remove” command and then we can go ahead and add our newly added role using the “knife node run-list add role” command. The role attributes could only be defined as either ‘default’ or ‘override’ attributes. When a Chef Client run happens, the role attributes are matched against the attributes present on the node and if it takes precedence over default attributes, new settings are applied.

**[What are the various ways to bootstrap a node?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4106)**

Bootstrap is the process by which the Chef Client is installed on the Chef nodes. Chef nodes could be physical, virtual or cloud machine. The bootstrap process can be done in two ways:

1. **Using the “knife bootstrap” sub-command**

The “knife bootstrap” command is issued from the Chef workstation. The command first creates an SSH connection with the node then installs the Chef Client along with all its dependencies on the node. This command also generates keys and register the node with Chef Server. In earlier versions of Chef Client, "ORGANIZATION-validator.pem" file was used to authenticate the Chef node with the Chef Server during the first Chef Client run. Starting from Chef Client 12.1, "USER.pem" file could be used to authenticate the new Chef node. This process is also known as “validatorless bootstrap”.

1. **Manual install in the node directly**

We need to login to the Chef node and manually install the chef-client. Once it’s installed we need to manually set up the connection with the Chef Server.

Unattended Installs are often done on Chef nodes created in AWS Auto Scaling, AWS CloudFormation, Rackspace Auto Scale, and PXE. Chef Client is installed without SSH connection on the Chef nodes.

**[What are the data bags in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4107)**

Security is an important feature in infrastructure management, in Chef this could be achieved through data bags. While preparing recipes for setting the different nodes we might need to provide some sensitive data as well for example like authorization credentials. It is not a good practice to hard code them as this may lead to many security risks. It is here where we can make use of data bags. Data bags are places where we can save these important data, the data gets stored in JSON format (Key-value pairs).

Data bags are indexed hence can be called whenever required by the recipes or cookbooks through a search. The data bags are encrypted and hence the contents of the data bags can be accessed only by having a decrypting key, this is how data bags give security to the chef infrastructure. Other than encryption strategies, a Chef Data Bag can be secured using a ‘Chef Vault’. Data bags can be created through the commands in the knife utility tool or manually. Data bags could be considered as a method to store global data shared across nodes.

**[Why are the environments used in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4108)**

Setting up environments is an important part of infrastructure management. We know that for the successful development of any product we need different environments like a development environment, testing environment, production environment, etc. The concept of environments in Chef is the extrapolation of the same. We at times want our Chef nodes to converge to a testing environment or a production environment configurations. Hence we can have a particular configuration for a testing environment and different configuration for the production environment. This switching of node behaviour is achieved through Chef environment settings. Environments help administrators to easily identify the stage where the particular node is a part of.

If we don’t declare any environments the Chef assumes the nodes to be in “\_default” environment. Minimum one environment exists in an organisation. Environment attributes have higher precedence than default attributes of a node. Environment data could be stored in Ruby format or JSON format. An environment could be created as follows:

* Using Chef Management console at Chef Server
* Using the knife tool
* Creating a ruby file in an environments subfolder of chef-repo
* Creating a JSON file in chef-repo
* Using Chef Infra Server REST API

**[What is a wrapper cookbook?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4109)**

Chef Software hosts a repository for general-purpose community cookbooks called Chef Supermarket. Hence Chef users can use these designed and tested cookbooks wherever possible which in turn will reduce the infrastructure development time. But often we come across situations where we need to tweak the community cookbooks for our requirement i.e where wrapper cookbooks are created. Wrapper cookbooks are similar to normal cookbooks but they use recipes from other cookbooks also.One another scenario where wrapper cookbooks are created is where you have a large number of baseline cookbooks and we need to organise them for maintainability.

We create wrapper cookbooks and put in external recipes from other cookbooks for our use. This could be achieved by modifying

1. metadata.rb file to include details about all dependent cookbooks. All dependent cookbooks are specified in “depends” directive
2. Using ‘include\_recipe‘ directive in a recipe to include external recipes from other cookbooks

Wrapper classes can be further customized to your needs through some simple attribute changes. Attribute definitions and run lists of the nodes could also be version controlled using wrapper cookbooks.

**[Explain the directory structure of a chef-repo in your Chef  Workstation.](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4110)**

chef-repo in your Chef Workstation is a directory that holds all details about cookbooks, environments, roles, and data bags.chef-repo is like source code and need to be synchronized with version control system like Git.chef-repo has following directories and files

* .chef folder is a hidden folder that stores all validation files required for authentication between Chef node, Chef Server and Chef Workstation.
* cookbooks folder to store all cookbooks used by Chef Client to configure Chef nodes. While using "knife upload" cookbooks from this folder are updated to the Chef Server
* data\_bags folder to store all data bags created for an organisation. A single data bag creates a subfolder with a JSON file.
* environments folder is used to store all details about various environments available in the Chef Server.
* roles folder contain all definitions of various roles available in the Chef Server.
* Chefignore file is used to ignore unwanted files like swap files, version control data, build output data so that these details don't get uploaded to Chef Server. Wild cards like ?\*,\*\* could be used to specify ignore patterns in chefignore files. This file could be located on any sub-folders in chef-repo.

**[Differentiate between chef-apply and chef-client?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4216)**

Chef-apply is an in-built utility in Chef development kit. It helps to run a recipe from the command line. The basic syntax for chef-apply is as follows:  
*chef-apply test.rb*, where test.rb is the recipe which is run.  
chef-apply with “-e” option could be used to execute a resource. For eg:  
*chef-apply -e "package 'nginx'"* installs ‘nginx’ package.chef-apply could be run in a why-run mode where everything of a chef-client run is performed except modification of the system.  
chef-client is an agent that is run on every node to modify the node so that it attains the desired state. chef-client applies cookbook on to any node managed by Chef Server. During a Chef Client run the following activities take place

* Registering a node with Chef Server
* Synchronising cookbooks to be run on Chef node
* Rebuilding node objects retrieved from Chef Server. In the case of the first Chef Client run, there won't be any node object from Chef Server.
* Compiling resource collection
* Performing all actions on the node to configure the node to the desired state.

**[Why is the Starter Kit used in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4217)**

During Chef Server creation, you are asked to download Starter kit to Chef Workstation. After downloading, unzip the starter kit and move it to your desired path. Every Time the Starter Kit is downloaded, a new key pair is created and resets the older one. Starter Kit has all the configuration details for communicating with Chef Server. It helps us to create a folder structure for developing cookbooks. Main directories present in Starter Kit would be as follows:

* chef-repo: All details like cookbooks, roles, environments, data bags are all stored in chef-repo. Whenever Chef client is run on the local mode, it looks for chef-repo path and all paths are relative to chef-repo.chef-repo is like a source code which is usually in sync with a version control system such as Git. Hence all modifications done in the chef-repo could be tracked down.
* .chef folder : .chef folder is a hidden folder found in chef-repo directory . It has a knife configuration file and .pem files used as secret authentication key files. All configurations for knife commands are present in knife configuration files.

**[What are the various ways in which we can apply an updated cookbook on the Chef node?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4218)**

Whenever a cookbook is modified, the new version details of the cookbook is updated in “metadata.rb” file and then uploaded to Chef Server using knife upload or berkshelf upload. There are 3 ways in which the new cookbook would be applied to the desired Chef nodes.

* We can run chef-client with latest cookbooks from Chef Server on all nodes or specific nodes using “knife ssh” command. For eg:  
  *knife ssh '\*:\*' 'sudo chef-client'*
* SSH into the respective node where you want to run the latest cookbook and run the chef-client .chef-client picks up the latest cookbook in the Chef Server unless otherwise specified as a cookbook restriction says for eg: in an environment.
* chef-client could be run as a daemon or service at a specified time. Chef Infra Client cookbook available in GitHub repository could be used for this purpose.chef-client with “USR1” option could be used to trigger chef-client and do the node convergence. Apart from making a chef-client run, a service we can utilise Cronjob to run it often.

**[What are handlers in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4219)**

Handlers are used to solve different situations that arise during a Chef Client run. There are 3 types of handlers in Chef.

* Exception: Exception handler is invoked whenever a situation arises where the Chef Client fails. A recipe with “chef\_handler” resource is added to the run list of a node which gets loaded first on every Chef Client run.
* Report: Report handlers are used when Chef Client run succeeds and then wants to report back more information about the successful run. These are loaded first on every Chef Client run by adding a recipe with “chef\_handler” resource on the nodes run list.
* Start: Start handlers are used to trigger events at the start of a chef-client run. These could be loaded first on every Chef Client run by adding the “start\_handlers” in the “client.rb” file. If we are using chef-client cookbook start\_handlers are installed using the “chef\_gem” resource. Hence, “start\_handlers” will be present on the node before a chef-client run.

Exception and report handlers provide a wide range of information about the Chef Client run and these data could be used for analysis across the organizations.

**[Why is Ohai used in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4220)**

Ohai in Chef is used to detect Chef node attributes and these are provided to chef-client during the Chef Client run. Ohai collects details about various platforms, network data, memory data, CPU data, kernel data and other configuration details. All these attributes collected are automatic attributes that are not changed even after Chef Client run. Automatic attributes are having higher precedence overall default, override, force-default and normal attributes. Ohai gets installed with the Chef Client installation.

The ohai resource is available in Chef to reload the configuration of a node. These are often used to refer back system attributes that are changed by a recipe like a recipe that creates a new user during a Chef Client run. The syntax is as follows :

ohai 'name' do

plugin   String

action   Symbol # defaults to reload if not specified

end

where:

* ohai is the resource executed.
* name is the resource block name.
* action is performed to bring the node into the desired state.
* a plugin is the property of the resource.

**[Why Foodcritic is used in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4221)**

Foodcritic is an open-source tool used to detect common problems in our cookbooks like style, syntaxes, best practices, correctness etc. Foodcritic validates the Ruby code written in cookbook against several rules and creates a list of violations without running a Chef Client run. List of rules is available in Foodcritic website.

A foodcritic is often run from chef-repo with the name of the cookbook whose correctness we are going to verify. The correctness of the intention of the recipes in the cookbook is not checked instead the structure and syntax of the cookbook are checked. Whenever a Foodcritic detects violation it doesn't typically mean a change of code. For eg: Assume that rule number “FC003” defines a scenario where recipe uses a search method to get relevant data from the Chef Server. The rule suggests that an error may occur in the cookbook when the Chef Server is unavailable. We only bother to adopt this rule only if chef-solo is a part of our project workflow as chef-solo doesn’t work with Chef Server.

Food critic could also be used with Continuous Integration servers like Jenkins and Travis thus enabling automated checking within the delivery pipeline of our project.

**[Explain with an example of why ChefSpec is used in Chef.](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4222)**

ChefSpec is an in-built testing framework for testing resources and recipes. It’s a part of Chef Development Kit. Usually, unit tests are written in ChefSpec and is an extension of Behavior Driven Development framework called RSpec for Ruby.

A sample unit test to check if “Nginx” package is installed or not is written as follows:

*describe 'nginx::default' do*

*context 'When all attributes are default, on Ubuntu 16.04' do*

*let(:chef\_run) do*

*runner = ChefSpec::ServerRunner.new(platform: 'ubuntu',*

*version: '16.04')*

*runner.converge(described\_recipe)*

*end*

*it 'install a package' do*

*expect(chef\_run).to install\_package('nginx')*

*end*

*end*

where

* describe block denotes a scenario
* context block defines test within tests and each context block is executed individually.
* ChefSpec::ServerRunner is used to simulate Chef Client run without actually configuring anything. In our example without actually installing “Nginx” server
* “It” is where we actually write the test using the Ruby DSL.

**[How to create a custom resource in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4223)**

Custom resources in Chef are add on resources transported as a pat of cookbooks and used by Chef Client. These are reusable in the same way as pre-defined resources in Chef. Custom resources are kept as a separate Ruby file in the resources folder of a cookbook. For eg: Assume that we need to create a custom resource called “website” that make use of in-built resources of Chef like file, package and service resources.  
“website.rb “ file looks as follows:

property :home, String, default: '<h1>Welcome!</h1>'

action: create do

  package 'nginx'

  service 'nginx' do

action [:enable, :start]

  end

  file '/var/www/html/index.html' do

content new\_res.homepage

  end

end

action :delete do

  package 'nginx' do

action :delete

  end

end

Where

* “Home” is the property that sets the default value “*Welcome!”* for index.html file.
* The new resource has two actions, to create and delete. Create is the default action.

The website resource is named after the cookbook name and the file name in the resources folder. Hence if the name of our cookbook is “sample” our custom resource is defined as “sample\_website” and could be used in the cookbook as follows:

sample\_website 'nginx' do

  home '<h1>Welcome to the Sample

 website!</h1>'

end

**[Differentiate between “knife supermarket download “ and “knife supermarket install” command](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4224)**

“Knife supermarket download” command is used to download the cookbook available in Chef Supermarket be it private or public supermarket.

The cookbooks downloaded are in the form of tar.gz files and are downloaded to the current working directory. If the specified cookbook is deprecated in the Chef Supermarket the user is notified the same and provides the details of the most recent version of the non-deprecated cookbook. For eg :  
*knife supermarket download httpd*  
downloads “httpd “cookbook from Chef Supermarket.

“knife supermarket install “ command is used to communicate with cookbooks in public or private Chef Supermarket. The downloaded cookbooks are installed in the local git repository unlike in the case of “knife supermarket download” command.“knife supermarket install “ command does the following steps

* A new vendor branch is created for the cookbook
* The cookbook is downloaded in tar.gz format and then untarred and committed to the vendor branch creating a tag
* The new branch is automatically merged with the master branch. This helps the Chef practitioner to modify the community cookbook in the master branch but still can pull down newer versions of the community cookbook available in Chef Supermarket

Hence this command allows updated upstream versions to be used without losing the local modifications done to the cookbook.

**[How requests send to Chef Server during first Chef Client run is authenticated?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4225)**

Chef Node is any physical, virtual or cloud machine configured by Chef Client and managed by Chef Server. A “chef-client run” is used to describe a series of steps the chef-client performs when it is configuring a node. During a chef-client run, the specified node is registered with Chef Server. This is done to avoid unwanted requests to be sent to Chef Server from chef-clients. Every request initiated by chef-client is authenticated using Chef Server API and a private key stored in “/etc/chef/client.pem” file in Chef Node.

During the first chef-client run on any node, the node won’t have the private key hence private key specified in the chef-validator placed in “/etc/chef/validation.pem” file is used. If chef-validator is unable to send an authenticated request to Chef Server the chef-client run will fail. After the node gets registered with the Chef Server chef-client attains a “client.pem” private key which will be used for all future authentication requests to the Chef server.chef-validator is not used after the initial chef-client run and hence could be deleted.

**[What are the various debugging techniques used to debug chef-client run failures?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4226)**

Best practices in developing a cookbook are to use

* Usage of platform resources instead of custom ones
* Usage of community resources without creating a new one

These techniques help us to reduce the debugging effort greatly.

Main debugging techniques used to detect defects in recipes and chef-client runs are

* Use Empty run-list

Run Chef Client with an empty run list. This helps us to make sure that the chef-client run failure is not due to the recipes added in the run list. We are also now sure that the failure is due to the configuration settings of Chef Client. chef-client run failure could also occur due to permission issues of the user at Chef Server and also on the node where Chef Client is run.

* Verbose logging with knife

A built-in verbose logging of knife with “-v” option could be used to log messages

* Chef Client verbose logging

Chef Client could be run with verbose logging by using the “-l” option. “-L” option could also be used to specify the location of the log file.

* Log resource

log resource could be used to log the debug messages while running the Chef Client. This resource is also built into resource collection and run during the convergence phase. If we want to add log entries that are not added to the resource collection, use Chef:: Log. A sample log resource looks as below:

log 'mtr' do

  message 'A info message added.'

  level :info

end

Where  
“log” is the resource used  
“mtr” is the name of the resource block  
“message” and “level” are properties of the log resource defining the message logged and it’s level.

**[Explain TDD using Test Kitchen in Chef](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4227)**

Test-Driven Development is a DevOps practise where code is developed just to make a failing test pass. It follows a red, green, refactors cycle. In the case of Infrastructure point of view, we first write a test and see it fail. Then write the code to make the test pass and then refactor if required. TDD helps to create high quality, simpler code with high modularity.

In Chef, Test Kitchen helps us to perform TDD. We first write the InSpec test and run “kitchen verify” command to perform the test. If the test fails, enough code is written to make the test pass. We converge the node to the desired state using “kitchen converge” command. Then again “kitchen verify” command is run to verify if the test has passed. Then the code is refactored if required when all tests have passed. In places where CI/CD is used, we can use “kitchen test “ command to invoke creation of kitchen instance, converging and verifying and at last destroyed the kitchen instance. Hence helping us to detect the defects very easily and in the early stages of project development.

**[Explain semantic versioning of cookbook.](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4228)**

A cookbook is versioned whenever there is a third party component change, a bug fix, the addition of new features or improvements to existing cookbook. This helps us to easily organise the cookbook and let us know which version of the cookbook has what functionalities. This also helps us to specifically update a version of the cookbook to Chef Server and use it for chef-client runs.

A cookbook version is in the format “a.b.c” where a, b, c are decimal numbers. “a”, “b”, “c”  represents major, minor and patch versions respectively. For eg: 1.1.2 version represents major and minor release version “1” with patch version “2”.Versions like “1.2a.3” or “1.2.3.4” or “3” are not allowed.

A version restriction could be attained by combining a cookbook version with an operator. Following operators are used

= : Equals  
>: Greater than  
< : Less than  
< =: Less than and equal to  
> =: Greater than and equal to  
~>: approximately greater than. For eg: ~> 3.2.5 denotes cookbooks greater than and equal to 3.2.5 but less than 3.3.0

 Cookbook versions are specified in the “metadata.rb” file located at the top of our cookbook.

**[What is the purpose of “include\_recipe” directive in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4229)**

include\_recipe method is used to include one or more recipes from cookbooks. It might be recipes from the same cookbook or dependent cookbooks. The syntax for the include\_recipe method is as follows:

include\_recipe <Recipe name with the cookbook>  
For eg :    
include\_recipe 'apache2::mod\_ssl'

includes all resources from the “mod\_ssl.rb” file in “apache2” cookbook to our current recipe where the “include\_recipe” directive is specified. If same recipes are included multiple times using “include\_recipe” directive method, only the first is included and rest is neglected.”include\_recipe” directive is considered better practice than specifying the recipe in the run list especially when one cookbook depends on another cookbook. This also lessens the burdens of running the dependent cookbooks first by specifying it first in the run list of every Chef node.

In the case of wrapper cookbooks “include\_recipe” directive is used to include functionalities from the dependent cookbooks. If it’s not specified but the dependent cookbooks are specified in the depends section of “metadata.rb” file, an error occurs.

**[What are libraries in Chef? When we use them?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4230)**

Libraries store arbitrary Ruby code that could be reused across the recipes within any cookbook that depends on the library. This is often located in the library folder of our cookbooks. First and the foremost usage of libraries are to provide helper methods that reduce the code duplication and also provide a mechanism to hide the implementation logic in the recipes. Libraries get loaded first when the Chef loads our cookbooks.It make use of  “ load\_libraries\_from\_cookbook” method from the “Chef::RunContext::CookbookCompiler” class. Typical use cases of libraries are

* For creating modules containing methods used in our recipes. Functions are defined inside a module which is in-turn included in our recipes as a mixin. This is done by including module namespace in our recipe using “include” keyword and hence making all methods defined in the module available to recipes as if declared locally.
* Libraries extend Chef’s core classes and even helps in overriding their methods. For eg: if we need to modify a method in Chef:: Node class, you can define the method in a library.
* Facilitates the creation and implementation of heavyweight resources and providers.

**[What are the Policyfiles? Why are they used?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4231)**

Policyfiles are used to combine the advantages of roles and environment with Berkshelf.In usual Chef Workflow, the versions and the locations of cookbooks being used by a Chef Client are updated in “metadata.rb” file and “Berksfile”. Often this is a tedious task and a small error in these files make Chef Client apply the wrong cookbook and converged node would not acquire the required state. These problems could be solved by using a single document called Policyfiles to get the specific cookbook revisions and recipes that Chef Infra Client would apply. Policyfiles are applied on a group of nodes, cookbooks or settings. The syntax for a Policyfile.rb file is as follows:

name "name"

run\_list "ITEM", "ITEM", ...

default\_source :SOURCE\_TYPE, \*args

cookbook "NAME" [, "VERSION\_CONSTRAINT"] [, SOURCE\_OPTIONS]

where

* “name” is the name of the policy
* “run\_list “ is the list of recipes applied by Chef Client on all nodes associated with the policy
* “default\_source” specifies the location of the cookbook
* “cookbook” defines the name of the cookbook and version constraints. An alternate source path for the cookbook could also be specified here.

**[What is Chef shell and why it’s used?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4232)**

Chef shell is an interactive tool to work with resources. It helps to do REPL with the resources.chef-shell provides an easy method to test resources interactively rather than uploading our cookbook to Chef Server and applying it on Chef nodes. Breakpoints could be added on to the recipe execution using chef-shell.chef-shell could be used in three modes

* Standalone mode runs Chef with nothing loaded but interactively. This is the default behaviour of chef-shell if nothing specified.
* Solo mode runs chef-shell as the chef-solo client. This mode is enabled by using “-s” or “--solo” command-line flag. This mode loads all cookbooks as in the case of chef-solo and uses any JSON file provided with “-j” command-line option.
* Client mode makes chef-shell to run as in the case of chef-client on the host. This mode is invoked by “-z” or “--client” command-line flag.chef-shell uses local client configuration to connect to Chef Server and download the relevant run\_lists, attributes or cookbooks. We can override the server URL in the client configuration by providing “-s” command-line option. “-c” command-line option could also be used to use a different configuration file for chef-shell.

**[Explain Berkshelf workflow in Chef.](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4234)**

Berkshelf is the dependency manager that comes with the Chef Development Kit. Whenever you create a cookbook say for eg with “chef generate cookbook” command a file called Berksfile is created on top of the cookbook directory. This file contains the source path for all cookbooks. Assume that you created a new cookbook called “my-app” which depends on another community cookbook “nginx” from public Chef Supermarket. The typical steps we follow are:

* The details about the dependent cookbooks are updated in the “metadata.rb” file.
* The default recipe in the “my-app” cookbook is having an entry for including recipes from the community cookbook using “include\_recipe” directive
* Run “berks install “command to install all the dependent cookbooks in ~/.berkshelf/cookbooks default path. Another file called Berksfile.lock is created that has the exact version of all cookbooks installed by Berkshelf. This avoids unwanted mixing up of different versions of dependent cookbooks while sharing the “my-app” cookbook.
* The “my-app” cookbook along with its dependency cookbooks are uploaded into Chef Server using “berks upload” command
* Chef-client applies the relevant cookbooks on different Chef Nodes.

**[Differentiate chef\_gem and gem\_package resources?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4236)**

“chef\_gem” and  “gem\_package” resources are used to install Ruby gems. Every system where Chef Client is installed has two versions of Ruby running. One is a system-wide instance of Ruby and the other is the one available only with Chef Client. “chef\_gem” resource is used to install Ruby into Chef Client instance of Ruby and “gem\_package” resource to install Ruby into a system-wide instance.

gem\_package resource has a property called “gem\_binary” property that is used by Chef Client to detect the environment settings and then install the gems.          "chef\_gem" resource is having all properties and options as that of “gem\_package” resource except the “gem\_binary” property.”chef\_gem” resource always uses “CurrentGemEnvironment” where the Chef Client is running.

“chef\_gem” resource has additional functions of

* Running its actions immediately before the convergence of the node. This allows the gem to be used in the recipe just after it’s installation.
* Gems are made aware of the changes so that they could be used immediately by running “Gem.clear\_paths”.

**[Differentiate chef\_gem and gem\_package resources?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4236)**

#### [20.](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4238)

**[What is blacklisting and whitelisting attributes?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4238)**

The attributes that should not be saved by the node is called Blacklisted attributes. These are defined in the client.rb file. Attributes are blacklisted based on attribute types. “automatic\_attribute\_blacklist “ defines a hash that blacklists automatic attributes. Similarly “default\_attribute\_blacklist”, “normal\_attribute\_blacklist” and “override\_attribute\_blacklist” are hashes that blacklist default attributes, normal attributes and override attributes respectively. The best practice is to use “automatic\_attribute\_blacklist “ as automatic attributes generate most data after an Ohai run. For eg:

automatic\_attribute\_blacklist ['filesystem']

could be used to blacklist only ‘filesystem’ attributes in the below sample attribute data

{

  "filesystem" => {

    "/dev/disk0s2" => {

      "size" => "10mb"

    }

}

}

All other automatic attributes are saved by the node.

The attributes that are needed to be saved by a node are called Whitelist attributes.

These are also defined in the client.rb file. Similar to blacklist attributes, attributes are also whitelisted using attribute type.”automatic\_attribute\_whitelist”, “default\_attribute\_whitelist”, “normal\_attribute\_whitelist”, “ override\_attribute\_whitelist” are hashes used to whitelist automatic, default, normal and override attributes respectively. For eg:

automatic\_attribute\_whitelist ['network/interfaces/']

could be used to whitelist only network attributes among the automatic attributes generated like below:

{

"network" => {

    "interfaces" => {

      "eth0" => {...},

      "eth1" => {...},

    }

  }

}

**[Explain chef-solo?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4240)**

chef-solo is a command where Chef Client is run in a way that it doesn’t require Chef Server to get the required cookbooks.chef-solo make use of the local mode of the chef-client run. It doesn’t require authorisation as in the case of Chef Servers. It also doesn’t provide a centralised distribution of cookbooks or centralised API to interact with infrastructure components. Cookbooks could be loaded from two different locations i.e from a local directory or a URL where the tar.gz archive is present. In chef-solo, the node objects are stored in the form of JSON files in a local disk rather than as node objects in Chef Server. For eg:

*chef-solo -c ~/solo.rb -j ~/node.json -r* [*http://www.sample.com/chef-solo.tar.gz*](http://www.sample.com/chef-solo.tar.gz)

runs chef-solo with solo.rb configuration file using node.json file and retrieves cookbook from the<http://www.sample.com/chef-solo.tar.gz> URL.

The basic workflow for chef-solo is as follows:

* Chef Workstation should have “solo.rb” and “node.json” file inside the chef-repo directory
* Add “solo.rb” and “node.json” to the git repository
* Create or download the cookbooks from Chef Supermarket
* Commit and push the files to git repo
* If you want to run chef-solo in a Chef Node,
  + Login to the chef node where the chef-solo is going to be run
  + Clone the chef-repo from git with the relevant cookbooks that need to be applied on the Chef node
  + Run the “chef-solo” command by specifying “solo.rb” file and “node.json” file to be used.

**[How can you collect extra details about Chef node apart from the ones collected via generic Ohai Plugins?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4241)**

Ohai framework provides a  mechanism to create our own customised plugins to get more details about a Chef Node. For eg: if we want to find out if a Chef node is a virtual machine or not, no Ohai plugin gathers this information so we need to create a plugin for the same. Ohai makes uses of Ruby-based DSL to create plugins. Ohai plugins are usually kept in “lib/ohai/plugins” directory in the repository. A sample Ohai plugin will look like below:

Ohai.plugin(:Sample) do

  provides "level"

  collect\_data do

    level 100

  end

end

where

* name of the plugin “Sample” is passed to Ohai.plugin method
* “provides” method has a list of attributes our plugin provides. Here “level” is the attribute provided.
* “collect\_data” block is executed when a plugin is run by Ohai. In our example, the “level” attribute is set as 100.

Ohai plugins are tested in IRB Ruby shell. This helps us to run plugins without performing actual Chef Client runs or configuring nodes. After testing is complete and we need to run the plugins we have to specify the path of the new plugin by specifying Ohai::Config[:plugin\_path] << */location/of/plugins* line in client.rb or solo.rb file. This helps Ohai to load the plugins  correctly

**[What is .kitchen.yml file? Why is it used?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4243)**

kitchen.yml file is the configuration file for the Test Kitchen. All details needed to create a sandbox environment for testing our infrastructure using Chef is included in .kitchen.yml file

Syntax of a .kitchen.yml file is as follows:

The basic structure of a .kitchen.yml file is as follows:

*driver:*

*name: driver\_name*

*provisioner:*

*name: provisioner\_name*

*verifier:*

*name: verifier\_name*

*transport:*

*name: transport\_name*

*platforms:*

*- name: platform-version*

*driver:*

*name: driver\_name*

*suites:*

*- name: suite\_name*

*run\_list:*

*- recipe[cookbook\_name::recipe\_name]*

where

* driver\_name is the name of the driver used to create kitchen instances. For eg: kitchen-Dokken is used to create instances using Docker.
* provisioner\_name is used to denote how Chef Client is simulated in testing either as chef-solo or chef-zero.
* verifier\_name denotes the tool used to run the tests. Usually, InSpec is used.
* transport\_name specifies the transport used to execute commands on the kitchen instance. Normally ssh is used.
* platform section in .kitchen.yml specifies the type and version of the kitchen instance created.
* suites\_name represents the test suite run on the kitchen instance and it contains a run\_list applied which is the list of recipes applied.

**[What is Cookstyle in Chef?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4245)**

Cookstyle is an in-built  linting tool that checks for

* Best practices and style conventions
* Source code uniformity
* Metrics like in “line length” and functional size

Cookstyle is based on RuboCop Ruby linting tool.

Cookstyle is run from the command line against a cookbook and verifies all the Ruby files in the cookbook. Cookstyle is run as

*cookstyle <cookbook path>*and the output is provided via standard output on the terminal.

The output gives us information about the number of files present and verified, defects detected and place of defects. The result of the verification of the files is shown below:

* . - No issues found in the file
* C - Convention issue in the file
* W - Warning issue in the file
* F - Fatal error in the file
* E - Error in the file

Autocorrection facility is available for files with cookstyle warnings but care must be taken not to alter the functionality of the code after using auto-correction. Default settings in the cookstyle could be overridden by .rubocop.yml file in a cookbook.

**[What is frozen cookbooks? How are they created?](https://www.knowledgehut.com/interview-questions/chef" \l "collapse-beginner-4246)**

Frozen cookbooks are those cookbooks which cannot be re-imported or modified. Hence the frozen cookbooks would not even be accidentally modified. For eg:  This method helps us to protect our production environments from being modified while testing changes made to the development infrastructure. We can freeze a cookbook as follows:

* Use “--freeze” option while uploading the cookbook to Chef Server using “knife upload” command. Once a cookbook is frozen it could be updated only by using “--force” option with “knife upload” command and if the “--force” is not specified an error message would be returned.
* “berkshelf upload” command automatically makes the uploaded cookbook frozen. To stop freezing of the cookbook we could use the option “--no-freeze”. Similar to a knife, “bookshelf upload” has an option “--force” to update the frozen cookbooks.   Another option called “--halt-on-frozen” exits the “berkshelf upload” command with a non-zero exit code if the same version of the cookbook is present in Chef Server.

**LINUX INTERVIEW**

**ADVANCED**

Preparing for a Linux interview in a short time is not a challenge anymore. Here are some popular Linux interview questions to get you through. Bridge your knowledge gap with the top linux interview questions and answers for experienced and freshers listed here that discusses topics like the difference between Unix and Linux, usage of different commands in Linux, etc. and get hired as a Linux administrator, cloud administrator and similar profiles.Once you are prepared with these tricky interview questions, you will be able to pass the toughest of Linux interviews easily.

#### [Explain with examples in details about the different commands to find memory usage in Linux?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1121)

From the command shell, use the command for memory usage information :

**cat /proc/meminfo**

Ex :

% cat /proc/meminfo

MemTotal: 16250912 kB

MemFree: 3281056 kB

MemAvailable: 10404492 kB

Buffers: 1101852 kB

Cached: 4654684 kB

SwapCached: 129304 kB

Active: 7930860 kB

Inactive: 2892144 kB

Active(anon): 4118480 kB

Inactive(anon): 1197660 kB

Active(file): 3812380 kB

Inactive(file): 1694484 kB

Unevictable: 236 kB

There are other commands also which gives memory info :

free - m

vmstat

top

htop

#### [Explain the command used to count every occurrence of the term “warn” in all the files appearing under the current directory, and its subdirectories, recursively?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1122)

To list every occurrence of the term “warn” on a separate line,

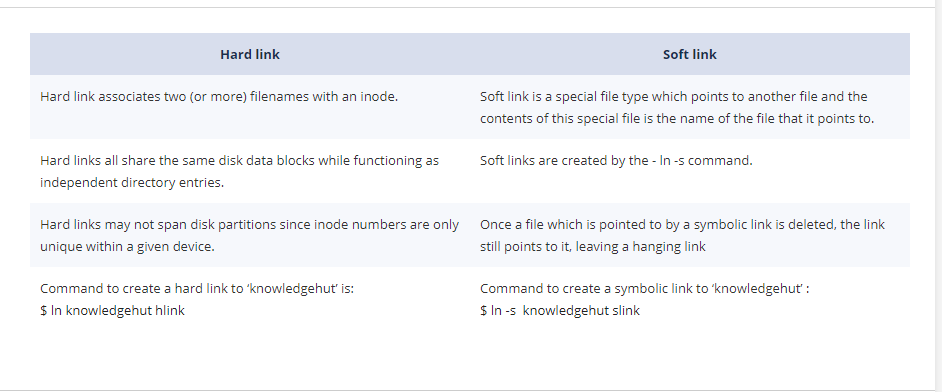
run grep -o warn <path>. Adding the r flag to the command makes recursive search for every file under the given path, and the I flag ensures that matches in binary files are ignored.

In addition, the w flag can be included to match the exact term only, and ignore superstrings such as “warnings”, and to make the search case-insensitive, the i flag can be added.

**% grep -iworI warn | wc -l**

**12**

[**Explain the major differences between Hard Link and Soft Link?**](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1124)



#### [Explain the usage of SED command with examples?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1125)

SED command in UNIX stands for stream editor, which is used to make changes to file content.

It can be used to find and replace strings or patterns without opening a file

The default behavior is that  the SED command replaces the first occurrence of a pattern in each and it won’t replace the second, third or multiple occurrence in the line.

If we use the ‘g’ option along with the above command then SED command will replace all Unix strings with Linux globally ( g stands for globally) .

sed 's/unix/linux/g' sample.txt

#### [How to remove blank lines from a file - sample.txt in a single command line ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1126)

sed '/^$/d' sample.txt

Here “^” symbol represents the starting point of a line and “$” represents the end of the line. Whereas “^$” represents the empty lines , d stands for delete .

#### [How to get a list of currently running processes and resource utilization in Linux?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1127)

Top is the command used to get the list of running processes and resource utilization (RAM or CPU usage). It gives  all the information about each process running on the host :

* Process ID (PID)
* Owner of the process(USER)
* Priority of process(PR)
* Percentage of CPU (%CPU)
* Percentage of memory
* Total CPU time spends on the process.
* Command used to start a process.

**Sample Output :**

PID USER      PR  NI  VIRT  RES SHR S %CPU %MEM    TIME+  COMMAND  
7629 greys     20 0 749m 291m  28m S 1 7.4 16:51.40 firefox  
19935 greys     20 0 133m 14m  10m S 0 0.4 2:38.52 smplayer  
   1 root      20 0 4020  880 592 S 0  0.0 0:00.96 init  
   2 root      15 -5 0    0 0 S 0 0.0   0:00.00 khutreadd  
   3 root      RT -5 0    0 0 S 0 0.0   0:00.04 datamigration/0  
   4 root      15 -5 0    0 0 S 0 0.0   0:00.90 ksoftirqd/0  
   5 root      RT -5 0    0 0 S 0 0.0   0:00.00 watchdog/0  
   6 root      RT -5 0    0 0 S 0 0.0   0:00.06 datamigration/1

Most commonly used options with the top  command are below –

top -u -> Process by a user.

top – i -> exclude idle tasks

top -p -> Show a particular process

#### [What is the command to list all the opened files of user–kunand?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1128)

lsof -  ‘LiSt Open Files’ is used to find out which files are opened or are in use .

Example :

# lsof -u kunand

COMMAND  PID   USER FD TYPE     DEVICE SIZE/OFF NODE NAME

sshd    1838 kunand  mem REG   253,0 122436 190247 /lib/libselinux.so.1

sshd    1838 kunand  mem REG   253,0 255968 190256 /lib/libgssapi\_krb5.so.2.2

sshd    1838 kunand  mem REG   253,0 874580 190255 /lib/libkrb5.so.3.3

#### [What are Zombie process in UNIX and how to find them ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1129)

Zombie process is a process whose execution is completed but have not been removed from the process table.

When a program forks and the child finishes before the parent , kernel has some of the child information .

In case parent needs to check for child's exit status - parent calls 'wait()'. Child is said to be a zombie process in the duration child terminating and the parent calling 'wait()'.

Execute the below command

ps aux | grep Z

child will have a 'Z' in its status field to indicate zombie process

Also this command will give details of all zombie processes in the processes table.

#### [What is the command to kill Zombie processes ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1130)

As zombie processes are already dead , the user  Cannot kill something which is already dead.

Execute the below command :

kill -s SIGCHLD pid

Replace the pid with Parent process ID, so that parent process will remove all the child processes that are dead.

#### [What is the command to find space consumed by each directory ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1131)

This can be checked by using du command ( disk usage)

du –sh . | grep G    - lists all the directory which has GIGS in Size.

$ du –sh . | grep G

5       ./Default/AppData/Local/Microsoft/Windows/WinX/Group1

25      ./Default/AppData/Local/Microsoft/Windows/WinX/Group2

60      ./Default/AppData/Local/Microsoft/Windows/WinX/Group3

#### [What is the command to find hostname from IP Address and vice versa ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1132)

Below commands can be used :

**nslookup** - to find the IP address from a hostname or vice-versa.

**ipconfig or ifconfig** - based on whether the host is Windows or Unix

**hostname -i** - on Linux

#### [Explain the usage of 'touch' command?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1133)

touch command is used to create an empty filename

Example :

$ touch knowledgehut.txt

Also for existing files or directories, touch command changes the last access time to current time.

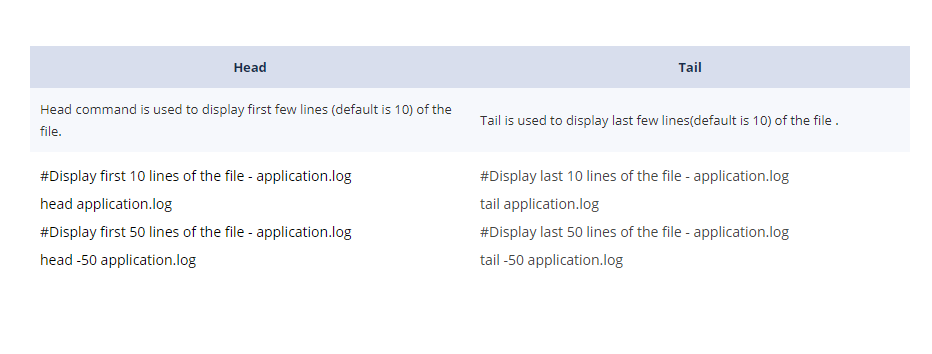
#### [What is the primary difference between head and tail commands in Linux ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1134)

#### [Write the command to print content of line 10 to 20 from input.txt to output.txt ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1135)

This can be done using 'sed' command :

# Here 'p' to print and '-n' to not print each line

sed -n 10,20p input.txt > output.txt



#### [What are aliases and how to create them with examples ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1137)

Aliases are abbreviated shortcuts used to represent a command or a group of commands executed with or without custom options.

#Alias for log directory

alias logs="cd /user/application/logs"

These aliases can be put in the ~/.bash\_aliases file.

To have the aliased command on any existing terminal, user needs to source ~/.bashrc from the terminal :

source ~/.bashrc

#### [Mention each system Calls used For Process Management in Linux ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1138)

Below are the system calls used for Process management:

fork () :- create a child process

exec() :- Execute a new program

wait():- wait until the process finishes execution

exit():- Exit from the process

getpid():- get the unique process id of the process

getppid():- get the parent process unique id

nice():- run a program with modified scheduling priority

#### [Explain LILO in details?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1139)

LILO is a boot loader for Linux. LILO stands for Linux Loader that is used to load Linux into memory.

It is used mainly to load the Linux operating system into main memory so as to begin operation.

Lilo handles some tasks such as locating the kernel, identifying other supporting programs, load memory and starts the kernel. The configuration file of lilo is located at “/etc/lilo.conf”. Lilo reads this configuration file and it tells Lilo where to place the bootloader.

#### [What is the command to find all the files and directories having 777 permissions ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1140)

find –perm option is used to find files based on permissions

Here "." or period denotes the current directory

$ find . -perm 777

./IBSO\_utilities/venv/Scripts/easy\_install-3.7.exe.manifest

./IBSO\_utilities/venv/Scripts/easy\_install.exe.manifest

./IBSO\_utilities/venv/Scripts/pip.exe.manifest

./IBSO\_utilities/venv/Scripts/pip3.7.exe.manifest

./IBSO\_utilities/venv/Scripts/pip3.exe.manifest

**[Explain different types of channel bonding available?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3535)**

The seven types of modes available, as mentioned below:

mode=0 (balance-rr): This mode is based on the Round-robin policy and it is the default mode. This mode provides load balancing and fault tolerance kind of features. It routes the packets in the Round-robin fashion that means from the first available slave through the last.

mode-1 (active-backup): This mode is based on the Active-backup policy. Only one is active in this band,

and another slave will act only when others fail. The MAC address of this bond will be available only on the adapter part to avoid confusing switch. This mode will also provide fault tolerance.

mode=2 (balance-xor): This mode sets an XOR (an exclusive or) mode that is the source MAC address is XOR’d with destination MAC address which provides load balancing and fault tolerance feature. Each destination MAC address the same slave selected.

mode=3 (broadcast): This method is based on the broadcast policy that will transmit everything on all the slave interfaces. It provides the fault tolerance feature. This can be used for only specific purposes.

mode=4 (802.3ad): This mode is called the Dynamic Link Aggregation mode that has it created aggregation groups having the same speed. It requires a switch that supports the IEEE 802.3ad dynamic link. The slave selection for outgoing traffic will be done based on the transmit hashing method. This can be changed from the XOR method via the xmit\_hash\_policy option.

mode=5 (balance-tlb): This mode is known as Adaptive transmit load balancing. The outgoing traffic will be distributed based on the current load on each slave and the incoming traffic is received by the current slave. If incoming traffic fails, the failed receiving node is replaced by the MAC address of another slave.

mode=6 (balance-alb): This mode is known as adaptive load balancing.

**[Defines UMASK and how can it be helpful on a Linux server?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3536)**

UMASK is user file-creation mode to determine permissions of newly created files. When any user creates a file or directory under Linux or UNIX, the default setting of the permissions was applied on the basis of UMASK mentioned in the config file. By default, UMASK id022 but you can change it complete system or for a particular user. Any files can have 3 types of permission read, write and execute, numeric representation is 777 for full permissions to a user, group and other users. When any user creates a file on the system with default UMASK 022 then files will get 755 permissions. 755 means read, write & execute to the user and read & execute to the group and other users.

UMASK is a very crucial command to control file & directory security. The system administrator can control file access permissions using UMASK in an efficient manner.

**[Share the difference between cron & anacron?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3537)**

As a system administrator or application admin, You need to execute multiple tasks or reports on a regular basis. We can automate or schedule such activities in Linux system using cron  & anacron.

We can use either cron or anacron as per our requirement but both have its features. Both cron and anacron are daemons processes.

Corn assumes your system is running continuously and online for execution. anacron can works when your system is not online 24X7. If our system is off and we have a job scheduled during this time, the job never gets executed.

Anacron uses timestamp file to check when was last time command or task was executed if schedule task or process missed the schedule due offline system. On the other hand, corn executes the required task a predefined schedule.

anacron every hour or day and check all required execution to execute whereas corn runs every minute to perform required action.

Corn job can be configured by any normal user but Anacron can be scheduled only by the superuser.

Corn is best when you can not expect a delay in execution time whereas Anacron is good when we can expect action in set intervals instead of specified timestamp.

**[What is the difference between ext2 and ext3 file systems?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3538)**

The ext3 file system is an enhanced version of the ext2 file system. The most important difference between the Ext2 and Ext3 is that Ext3 supports journaling.

Ext2 is a legacy file system has loys shortcomings. In case of a system crash or unexpected power failure or unclean reboot of the system, the system administrator needs to check all ext2 mounted drives for consistency. This needs to be performed an e2fsck program. This is a time-consuming process and during this time, any data on volumes is unreachable.

Ext3 is a newer filesystem with supports journaling.  Journaling feature in ext3 file systems eliminates the requirement of consistency check of the file system in case of a system crash or unclean reboot. The only possible situation of consistency check requirement in ext3 is with hardware failures. In such a case, recovery time depends on hardware speed, storage performance, and system resources. File size and a number of files do not create any impact, normally journaling complete consistency check in a few seconds.

**[How shadow passwords are given by in Linux?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3539)**

In Linux, User passwords save in /etc/passwd system file. This file accessible to all users and visible passwords is a security risk. Linux comes up with a feature of shadow password or encrypted password. In Shadow password, Passwords are encrypted before saving in /etc/passwd system file. This gives you security from unauthorized system access. The pwconv command is used for providing shadow passwords. This command creates the file as /etc/shadow and changes all passwords to ‘x’ in the /etc/passwd file. This functionality may require additional installation of shadow suite.

The original password is encrypted after creating shadow password by an encryption key

Encryption key saves along with the encrypted password for a further refresh

When a user using a shadow password tries to connect, the system decrypts the password before connecting.

**[Explain Window Manager in Linux?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3540)**

Window Manager is client software that controls icons, placement of icons, the appearance of the window after login to the system. This is desktop management software. As we know LINUX is an open-source operating system, We have a long list of WM software available in the market. The system administrator can install and configure it as per user or environment requirements. One thing before using WM software, they will consume additional resources on the system.

The/etc/.xinitrc file is hidden system files allows you to change the window manager while login from any or particular user account. The prefix of “.” In the file, the name shows that it is hidden file and you will not be able to view it with the normal ls command.  WM gives enhance user experience or add-on features.

Some of the popular command for WM or desktop management are:-

The KDE = startkde

The Gnome = gnome-session

The Blackbox = Blackbox

The FVWM = fvwm

The Window Maker = wmaker

The IceWM = icewm

**[Is it safe to use Telnet in LINUX system? Do we have any alternative for this?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3541)**

No, TELNET is not a secure way of communication. Talent sends data and sensitive information in plain text over the network and that can be easily accessible and readable by anyone. This gives the open opportunity to the hacker to hurt your system. As a System administrator, you need to close all possible security risk for your system and having Talent is one of the tops in the list.

SSH (Secure Shell) is a secure alternative of talent. SSH is completely secure and replaces legacy telnet usage. SSH save user identity, password, and data from the network attacks. Linux comes up with a free version of SSH known as OpenSSH. For extended features, We can also use paid versions of SSH.

**[What is the difference between the name-based virtual hosting and IP based virtual hosting? Explain the scenario where the name-based virtual hosting seems useful?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3542)**

The Virtual hosts are used to host the multiple domains on a single apache instance. We can have one virtual host for each IP your server has, or the same IP Address but different ports, or the same IP Address, the same port but different hostnames. The latter is called "the name-based vhosts".

In IP-based virtual hosting, we can run more than one web site on the same server machine, but each web site has its own IP Add while In Name-based virtual hosting, we can host multiple websites on the same IP address. But for this to succeed, we have to put more than one DNS record for your IP address in the DNS database. In the production shared web hosting environment, getting the dedicated IP address for every domain hosted on the server is not feasible in terms of the cost. Most of the customers won't be able to afford the cost of having a dedicated IP Add. Here is the place where the concepts of the Name-based virtual hosting find its place.

How the system administrator can manage and monitor memory usage in Linux?

Memory monitoring and usage management are one of the critical system administrator requirement. It’s always required to keep the system under monitoring to check if memory is low or any user or process is over-consuming it. Linux comes up with multiple commands that you can use to monitor and manage the usage. Different ways to check memory usage:-

**Free:** Free command gives details of memory used, free, cache and total. By default values are in KB but you can pass -m to have values in MB.

/proc/meminfo: This is a system file to monitor memory. It will give you 6 entries Total Active RAM, Total Inactive RAM, User Active RAM, User Inactive RAM, File Active RAM & File Inactive RAM.

**Vmstat:** VMSTATS give you memory statistics.

Top: Top command gives you memory usage and total RAM. This command also used for monitoring.

**[Briefly define ls command and options?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3543)**

This is one of the most basic and useful commands. This command (“ls”) is used by a normal user or system administrator on a regular basis. This command is used to list down files and directories in the present working directory.

“ls” Command comes up with multiple options:-

* l This option will show file & directory permissions in rwxrwxrwx format for user, group & others
* This option allows you to view hidden & system files
* i This option gives you inode details of files
* s Helps you to list down file size

“ls” without any options will list down all files & directories in plain text. This command gives more desired outputs after clubbing it with grep & less command that allows your filter the list or highlights the required file. This command is also compatible with input-output redirection option which is very helpful for logging.

**[What is Samba Share?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4263)**

Samba is an open-source software suite that runs on the Unix/Linux based platforms but it is able to communicate with the Windows clients like a native application. So Samba is able to provide the service by employing the Common Internet File System (CIFS).

At the heart of the CIFS is the Server Message Block (SMB) protocol.  Samba does this by performing the four key things –

* The File & print services
* The Authentication and Authorization
* The Name resolution
* The Service announcement (browsing)

Samba can be run on many different platforms including Linux, Unix, OpenVMS and the operating systems other than the Windows and allows users to interact with a Windows client or server natively. It can basically be described as Standard Windows interoperability suite of the programs for Linux and Unix.

**[What is the difference between ctrl+z and ctrl+c?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4264)**

When we have a process in progress which handle your prompt, there were some signals (orders) that we can send to theses process to indicate what we need:

Control+C sends SIGINT which interrupts the application. Usually causing it to abort, but a process is able to intercept the signal and do whatever it likes: for instance, from the Bash prompt, try Ctrl-C. In Bash, it cancels whatever you've typed and gives you a blank prompt (as opposed to the quitting Bash)

Control+Z sends SIGTSTP to foreground application, effectively putting in the background on suspended mode. This is very much useful when we want the application to continue its process while we are doing another job in the current shell. When we finish the job, we can go back into the application by running FG (or %x where x is the job number as shown in jobs).

**[Sometimes NSCD die itself and DNS resolving doesn't occur properly. How can we avoid NSCD for DNS and is there any disadvantage to bypass it?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4265)**

NSCD means Name Service Cache Daemon which provides a cache for the most common name service requests. When resolving a user, group, service the process will first try to connect to the NSCD socket (something like /var/run/nscd/socket2).

If NSCD died, then the connection will fail and NSCD won't be used for same and that should not be a problem.

If NSCD in a hung state, then the connection may hang or succeed. If this succeeds then the client will send the request. Now, we can configure NSCD to disable caching for any type of the database (for instance by having enable-cache hosts no in the /etc/nscd.conf for hosts database).

However, if NSCD is in a hung state, it may not be able to even give that simply won't do the answer, so that won't necessarily help. NSCD is a caching daemon, it's meant to improve the performance. Disabling it would potentially make those lookups slower. However, that's only true for some kind of databases. For the instance, if user/service/group databases are only in small files (/etc/passwd, /etc/group, /etc/services), then using NSCD for those will probably bring little benefit if any. NSCD will be very useful for the host's database.

**[What is the difference between /dev/random and /dev/urandom for generating random data?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4266)**

The Random Number Generator gathers noise of environment from the device drivers and other sources into the entropy pool. It also keeps an estimate of the number of bits of the noise in an entropy pool. It is from this entropy pool and will generate random numbers.

/dev/random will only be returning Random bytes from the entropy pool. If the entropy pool is empty, reads to /dev/random will be blocked until the additional environmental noise will be gathered. This is suited to high-quality randomnesses, such as the one-time pad or key generation.

/dev/urandom will return as many random bytes requested. But if the entropy pool is empty, this will generate data using SHA, MD5 or any other available algorithm. It never blocks the operations. Due to which, the values are vulnerable to the theoretical cryptographic attack, though no known methods will exist.

For cryptographic purposes, we should really use the /dev/random because of the nature of data it returns. Possible waiting should be considered as an acceptable tradeoff for the sake of the security, IMO. When we need random data fast, we should use the /dev/urandom of course.

Both /dev/urandom and the /dev/random are using exact same CSPRNG (a cryptographically secure pseudorandom number generator). They can only differ in very few ways that have nothing to do with the “true” randomness and then /dev/urandom is the preferred source of cryptographic randomness on the UNIX-like systems.

**[What are logical volume manager and its requirements on Linux servers?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4267)**

LVM is a short form of logical volume manager requires to resize filesystem size. This size of LVM can be extended and reduced using lvextend and lvreduce command respectively.  We can think of LVM as dynamic partitions, meaning that we can create/resize/delete LVM partitions from the command line while our Linux system is running: here is no need to reboot the systems to make kernel aware of the newly-created or resized partitions.

**LVM provided functions:**

* A logical drive can extend over more than one hard-disk. They are not limited by the size of one single disk, rather by the total aggregate in size.
* LVM can be used to create a (read-only) snapshot of any LV (Logical Volume). You can revert the original LV to the snapshot at a later time, or delete the snapshot if you no longer need it.
* LVM also supports writeable snapshots. LVM allows freezing an existing Logical Volume in time, at any moment, even while the system is running.

**[You are getting "filesystem is full" error but 'df' shows there is free space. Explain the problem here?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4268)**

This is very much possible that we have free storage space but still we cannot add any new data in the file system because all the Inodes are consumed as the df -I command will show that. This may happen in a case where the file system contains a very large number of very small-sized files. This will consume all the Inodes and though there would be free space from a Hard-disk-drive point of view but from a file system point of view no Inode available to store any new file.

A storage unit can contain numerous small files. The inode structure fills up before the data storage of disk, no more files can be copied to the disk. Once inode storage is freed up in the structure, new files can be written to storage.

**[What shell does a Linux Administrator assign to a POP3 mail-only account?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4269)**

POP3 mail is the only account that is assigned to the /bin/false shell. However, assigning bash shell to the POP3 mail only gives user login access, which is always avoided. /bin/nologin can also be used for the same. This shell access is provided to the user when we don’t want to give shell access to the user. The user cannot access the shell service and it rejects shell login on the server as in Telnet. It is mainly for the security of all shells.

POP3 is basically used for downloading the mail to the mail program. So for illegal downloading of all emails on the shell, this account is assigned to the /bin/false shell or the /bin/nologin. These both shells are the same as they both do the same work of rejecting the user login to the shell.

The main difference between these two shells is that the false shell shows the incorrect code and any unusual coding when a user login to the shell. But the nologin shell simply tells that no account is available. So nologin shell is used often in the Linux.

**[Explain list of daemon responsible for event tracking in Linux system & signals given to the syslogd?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4270)**

syslogd daemon process facilitates the event tracking in a Linux system and logs useful information or future analysis. syslogd provides two system utilities, one for logging and other for the kernel messages. syslogd mainly reacts to the set of signals given by users.  
  
**Some of the signals given to syslogd:**

* **SIGHUP:** This signal instructs syslogd to perform a re-initialization step. All open files are then closed, the configuration file (default is /etc/syslog.conf) will be reread and the syslog facility will be started again.
* **SIGTERM:** syslogd will die.
* **SIGINT, SIGQUIT:** If debugging is enabled these are then ignored, otherwise syslogd will die.
* **SIGUSR1:** Switch the debugging on/off. This option can only be used if the syslogd is started with the - d debug option.
* **SIGCHLD:** Wait for Childs if some were born, because of the waiting messages.

**[How the system administrator can manage and monitor memory usage in Linux?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4271)**

Memory monitoring and usage management are one of the critical system administrator requirement. It’s always required to keep the system under monitoring to check if memory is low or any user or process is over-consuming it. Linux comes up with multiple commands that you can use to monitor and manage the usage. Different ways to check memory usage:-

* **Free:** Free command gives details of memory used, free, cache and total. By default values are in KB but you can pass -m to have values in MB.
* **/proc/meminfo:** This is a system file to monitor memory. It will give you 6 entries Total Active RAM, Total Inactive RAM, User Active RAM, User Inactive RAM, File Active RAM & File Inactive RAM.
* **Vmstat:** VMSTATS give you memory statistics.
* **Top:** Top command gives you memory usage and total RAM. This command also used for monitoring.

**[What is NFS and Benefits of NFS?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4272)**

Full form of NFS is Network File System. NFS is used for sharing of the files and folders between Linux/Unix systems by Sun Microsystems in late 1980. NFS helps you in mounting your local file systems or drive over a network and remote \ client hosts can use it as it mounted locally on their system. With the help of the NFS, we can set up file sharing between the cross-operating system, Unix to Linux system and vice versa. If you want to use Linux system mount on windows, you need to use SAMBA\CIFS in place of NFS.

**Benefits of NFS:-**

* You can access remote files as local
* NFS uses client/server architecture for file sharing
* NFS support file sharing cross-operating systems
* NFS helps in building centralized storage solutions
* Users can access their data irrespective of physical location
* No manual refresh needed for new files
* NFS can be secured with Firewalls and Kerberos

**[Can we think of systemd over init system?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4273)**

Systemd is the first process of the Linux system and very well designed process in compare with init.

Systemd is multithreaded and faster than init. Systemd is standard processes to control programs need to be run during Linux boot. It was conceived from the top, not just to fix the bugs, but to be a correct implementation of all the base system services. A systemd, may refer to all packages, utilities and the

libraries around the daemon. It was designed to overcome all the shortcomings of init. It is itself a background process which is designed to start the processes in parallel, and thus reducing the boot time and computational overheading. It has a lot of other features as compared to init.

* Systemd support multiple useful commands with unified command-line interfaces
* Systemd makes the boot process much simpler
* Systemd support backward compatibility
* Systemd supports SELinux integration
* Systemd is an incredibly fast service

**[Explain /proc filesystem?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4274)**

/proc is a virtual file system that provides detailed information about the Linux kernel, hardware, and running processes. /prod is a generic file available in all flavours of Linux. Files under /proc directory named as Virtual files. These files are created when the system boots up and dissolve on shutdown. It contains information about running processes and works as an information zone for the kernel.

/proc is also a hidden tool for a system administrator for analyzed and troubleshooting performance and system bottleneck related issues.

These virtual files have unique qualities. Most of them are listed as zero bytes in size as they reside in memory, not on disk. Virtual files such as the /proc/interrupts, /proc/meminfo, /proc/mounts, and the /proc/partitions provide an up-to-the-moment glimpse of system’s hardware. Others: /proc/filesystems file and /proc/sys/ directories provide system configuration information and interfaces. These are tools for a system administrator to troubleshoot and analyze the issues.

**[Explain briefly the procedure for re-installing Grub in Linux Server?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4275)**

Boot Loader is a package that loads operating system to memory during boot. Windows comes up with its own boot loader whereas Linux gives you to select boot loader as per your environment and requirement.

GNU GRUB or GRUB (Grand Unified Bootloader) is a type of boot loader package that supports multiple operating systems. It allows feasibility of selecting the required OS during boot. GNU GRUB gives the option to select the operating system to load during boot. GNU GRUB is an advanced level of legacy GRUB.

* Unlimited number of boot entries supported
* Dynamically configurable, run-time changes can be made during boot
* Easy to install or execute from any device
* It can be helpful in loading the operating system from the network or decompressing during boot
* It also supports Windows and DOS
* Only CLI (Command Line Interface) is available
* Installation and configuration steps have a slight difference on the basis of OS distribution used

**[Explain Command Line Interface?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4276)**

Command Line Interface is also known as CLI. This is an interface for users to interact and instruct system in command line fashion. CLI is the basis on text-based interact to accept user request and response. While comparing with GUI, CLI is lightweight and consume less CPU & Memory resources.

considering the GUI of different versions and flavour, User needs to change there way of working and need additional learning. Whereas CLI is independent of this and allows the user to use any Linux system in the same manner. CLI also comes up with help option so that users need not remember all commands and option and they can refer help or man page for details options and definitions.

**Advantages**

* Easy, Fast & Flexible
* Very less load on CPU or Memory

**Disadvantages**

* Commands syntax is a bit tricky and tough to learn
* Long Text inputs can be problematic
* In Compare of GUI, bit tuff to use

**[What does SELinux mean in Linux?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3523)**

SELinux is known as Security-enhanced Linux. In today’s world data is everything, Protecting your server and keep is up to server is a major challenge. Linux kernel gives security option of SELinux which is designed to protect the server from misconfigurations and unauthorized data access/modification. It helps in defining policy for accessing programs and files.

SELinux comes with 3 modes Enforcing, Permissive and Disabled.

1. Enforcing mode: Default mode to enforce the policies on the system, restrict access and log actions.
2. Permissive mode: Here, SELinux was enabled but does not enforce security policies. This result warning and logging for any restricted access.
3. Disabled mode: When SELinux is turned off and the security policies do not protect the server.

**[What is Linux Loader?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3524)**

Linux Loader or Linux Boot Loader is also known as LILO, a combination of initial 2 characters of Linux LOader.

LILO loads Linux operating system into main memory to boot system and to start working on it. Multiple operating systems like Windows & Mac OS comes up with their respective boot loaders.  When you install Linux OS, you need to install a special boot loader for it. We have multiple boot loaders available in the market, LILO is one of them.

When the system started, BIOS performs some initial tests and transfers control to the Master Boot Record. Now, LILO loads the Linux OS and starts it. The best part of using LILO is that it allows fast boot of Linux OS.

**[Explain /bin, /usr, /sbin, /user/bin and /user/sbin Linux directories?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3525)**

**/bin:** Its critical directory used to bring the system online in single-user mode to repair it. This contains executable programs(can say scripts) for this.

**/sbin:** This directory holds commands needed to boot the system in normal condition but not executed or used by normal users.

**/usr:** This is one of the largest directories in the Linux system, mostly mounted from a separate partition. Birnies and files all programs are installed reside here.

**/usr/bin:** This directory contains programs, executables and scripts not used for boot process but used by users to execute. Most of the programs or executable executed by users rather than root

**/usr/sbin:** Program binaries or executables required for a system administrator is kept under this directory. This program binaries or executables are not required for boot process or normal users.

**[Define cron job and steps to configure a cron job to run every week “At 00:00 on Sunday.”?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3526)**

cron Job is similar to Task Scheduler in windows. corn is a software utility which schedules a command or script on your server to run automatically at a mentioned time and date. cron jobs can be very useful to automate repetitive tasks according to our need.

For example, we need to delete some temporary files every week to conserve our disk space. Once we have a script in place doing required action, We can set up a cron job to perform a certain action on a specific time. Scripts executed as a cron job are typically used to modify files, directories or databases. However, they can perform other tasks that do not modify data on the server, like sending email notifications.

We need to enter below lines in crobtab by:

 testmachine@myworld-linux:~$ crontab -e

0 0 \* \* 0 /path/to/command

Where,

* 1 => is the Minute (0-59)
* 2 => is the Hours (0-23)
* 3 => is the Day (0-31)
* 4 => is the Month (0-12 [1 == January])
* 5 => is the Day of the week(0-7 [7 or 0 == sunday])
* /path/to/command – Script or command name want to schedule

**[What is the ctrl+D error?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3527)**

ctrl+D error is one of the common errors occur when the root user tries to make any Permanent entry in fstab file & by mistake changes the path location of mounted file systems in os, it gives error while rebooting the system and fails to load.

fstab is a system configuration file on the Linux operating system that contains information about major filesystems. This file is located in /etc directory and can be viewed like “cat /etc/fstab”.

We have set steps to recovery system from ctrl+D error.

1. Boot system using bootable CV/DVD of Linux of the current version installed
2. Read ctrl+D error for the exact issue
3. Connect using Root User and it will connect you in single-user mode
4. Access /etc/fstab file and make required changes
5. Try normal system reboot

**[What is the FTP Server?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3528)**

FTP is the simplest file transfer protocol to exchange files to and from a remote computer or network system. Similar to Windows, Linux, and UNIX operating systems they also have built-in command-line prompts that can be used as FTP clients to establish FTP connection. FTP works in Client-Server architecture to communicate and transfer the file during an established FTP session.

When the Client initiates a connection to the server, it’s called Passive Connection. Whereas when Server initiates a connection to the client, it’s called Active Connection.

In Phase 1, When Connection initiated with the server, User credentials are passed for authentication. This is the control connection phase. In Phase 2, When actual data is transferred between client & server, This is data connection phase.

**[Explain are different process states in Linux?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3529)**

When we execute “ps aux”  in Linux terminal console, we can see multiple states of processes running in the system under the STAT column.

**R:** Process is running with CPU or waiting for CPU (Running or Runnable)

**S:** Process is waiting for the set event to complete, Like an input from terminal (SLEEP)

**D:** Process is with uninterruptible sleep stats which cannot be changed or killed \ rollback. The only way to go away is the reboot system.

**Z:** Process in Zombie status means the process is already killed but process information and data still exist in the process table.

**T:** Process either completed or terminated by the operating system or user. This is also known as Terminated /  Completed.

**[Explain Root and it’s significance in the Linux system?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3530)**

The root is the most privileged account in Linux for the system administrator. The root user has you full access to the system to perform all kind of access. The root is default account of Linux, created with Linux installation only. The root user is also known as the Root account or superuser. Due to uncontrolled access of Root account, this account needs to be secured and used crucially and carefully.

Some of the functions can be performed by the Root account:-

1. The root can create/delete/ modify any user, user group, files, directory and permissions
2. The root can connect by using any user without password
3. System / Library / function calls
4. Managing hidden & config files
5. System administrator commands
6. Etc.

**[What is the numeric and alphabetical method for granting permissions?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3531)**

Granting permission on files and directories is one of the crucial and data security stuff for Linux System Administrator. Permission on any file or directory is a combination of 9 alphabets. First 3 from the left represents owner access, then middle 3 represents user group access and the last 3 are for other users who are not the owner or part of a group having access.

**rwxrwxr**

**wx**

We have several ways to grant permissions using the numeric and alphabetical method. Having knowledge of these shortcuts makes administrators life easy.

rwx = 111 in binary = 7

rw- = 110 in binary = 6

r-x = 101 in binary = 5

r-- = 100 in binary = 4  
  
So when you mention 765 numeric code to provide access permission, User will get like:- **rwxrw-r**

**[What is IRIX Mode in top command?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3532)**

CPU usage percentage is very common confusion in LINUX. Sometimes, User or new system administrator complaint that CPU is showing >100% which is some configuration issues or VM issues. But this is not true.

Linux treats all processors individually and when you run top common, it should CPU usage per processor wise. Here, System is not considering all processors as a whole and showing usage of each processor individually which lead to total PU utilization to >100%.

To resolve this confusion, IRIX mode has been introduced. This is the default mode of Linux now. In Irix Mode, System considers all processors as a whole and CPU usage can vary from 0 to 100 only. IRIX feature was introduced in Solaris and adopted by Linux later.

**[What is the difference between the tar, gz, and zip?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3533)**

Tar & ZIP are two most commonly used utilities in Linux system

TAR is archiver utility which will archive the selected files or directories. Extension of TAR is .tar.

gz is known as gunzip used compress files only. Extension of gz is .gz. You can use gz on TAR to compression directories achieved by TAR.

ZIP is archiver and compression utility for files and directory. Extension of ZIP is .zip.

The benefit of TAR can be applied on directories. Sometimes, you do not want to compress the filer but want to bundle them, TAR is perfect for it. TAR with GZIP is the best combination. Like GZIP, we also have bzip2 which use a completely new algorithm to compress files has given less size in compare to gzip.

**[List down some of major Linux distributions?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-3534)**

LINUX is an open-source operating system which allows users to modify kernel as per their requirement. This facilitates the different part of Linux to be deployed, modified and tested by a different organization. This result in multiple flavors of Linux available in the market and each has its own feature.

Major Linux distributions are as below:-

Ubuntu: It’s the most common and well-known distribution. It has lots of ree installed apps for user’s easiness. It’s very easy to use and available in the command line and GUI both.

Red Hat Enterprise: Red Hat Enterprise Linux or RHEL is commercial Linux distribution. It stale, tested, user-friendly and most important NOT free to use.

Debian: Debian is one of the fastest and user-friendly Linux version.

Linux Mint: Its a special type of distribution works on the windows system as well. This for beginners to get hands-on the Linux system.

Fedora: Fedora is not in use on high numbers due to less stability. It supports a GNOME3 desktop environment by default.

**[Briefly explain the process of sending an email?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4250)**

The basic steps of this process are mentioned below for your convenience.

**Step 1:** After composing a message and send, your email client - whether it's Outlook Express or Gmail - connect to the domain's SMTP server. This server can name many things; a standard example would be smtp.local.com.

**Step 2:** In this step, the email client communicates with the SMTP server by giving your email address, the recipient's email address, the message body, and any attachments.

**Step 3:** Now SMTP server has complete details to processes the recipient's email address - especially its domain. If the domain name is the same as the sender's, the message is routed directly over to the domain's POP3 or IMAP server - no routing between servers needed. If the domain is different, though, the SMTP server will have to communicate with the other domain's server.

**Step 4:** To deliver email and to find the recipient's server, the sender's SMTP server has to communicate with the DNS or Domain Name Server. The DNS will take the recipient's email domain name and translates this into an IP address. The sender's server cannot route an email properly with a domain name alone; an IP address is a unique number that is assigned to every computer that is connected to the Internet. By knowing information, an outgoing mail server can perform its work more efficiently.

**Step 5:** Once SMTP server got the recipient's IP address, it can connect to its SMTP server. This isn't done directly, though; instead, the message is routed along with a series of unrelated SMTP servers until it arrives at its destination.

**Step 6:** The recipient's SMTP server scans the incoming message. If the domain and user name are valid, it forwards the message along to the domain's POP3 or IMAP server. From there, it is placed in a send mail queue until the recipient's email client allows it to downloaded. At that point, the message can read by the recipient.

**[Define Linux Kernal and can we modify it?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4251)**

The kernel is the lowest level of software that controls all hardware and communicates with users for functions needs to perform. Linux Kernel is the core of the operating system that provided a user interface to perform user commands and control associated hardware. Linux Kernel is a layer that provides the ability to the user to control system hardware, develop applications on the operating system. All underline hardware with the system is communicable though Kernel only. The kernel gives you the independence to use software and programming language by your choice, Kernel is capable enough to convert them in machine language to control the required hardware subsystem.

Linux Kernal is free and open-source software and as per General Public License (GPL), it becomes legal for anyone to edit it.

Open-source software authorized you to distribute software with the source. This privileged people to review and add features as per their requirements. This gives a win-win solution to the complete community.

**[What is Network Teaming?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4252)**

Network Teaming also is known as Ethernet Channel Bonding that enables two or more Network Interfaces Card (NIC) to work as a single virtual NIC card. This means the machines will be supposed to work on the virtual one and which may increase the bandwidth and provides redundancy of NIC Cards. This helps us in achieving redundant links, load balancing or fault tolerance networks in the production system. If one physical NIC is down or can say unplugged, it will automatically move the resources to other NIC card. Channel/NIC bonding works with the help of the bonding driver in Kernel.

**2 main types of Network Teaming:-**

1. **Load balancing (Round-Robin):** In Load balancing, Both network cards are active all the time and dividing the workload for better data transmission. This provides better resource utilization and faster output.
2. **Active-Backup: In Active-Backup**: One network card is active at one time and another card will keep passive for backup. In case of active network card failure, Passive card will start behaving as active to fulfil the user request.

**[What are the different types of Shells used in Linux?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4253)**

Shell is an interpreter which converts scripts or executables to machine action. Shell prompt is a Command-line interface as well as GUI (Graphical User Interface) that takes inputs from the user and executes selected program according to that. Shell Scripts can be combined in a package for automation and schedule background tasks. Shell Scripts will be saved using extension .sh and scheduled using corn jobs.

**Some majorly known shells as below:**

* sh: sh is known as Bourne Shell (sh). This is the original UNIX shell which has limited functionality. Command path is located at /bin/sh and /sbin/sh.
* bash: Default shell in most Linux installations and compatible with the legacy sh shell. bash is the short name of Bourne Again Shell. Command path is /bin/bash.
* ksh: High-level programming language shell known as Korn Shell. Command path is /bin/ksh.
* Tcsh/csh: Tcsh or csh is also known as C Shell. This works on C language syntax. Command path is / bin/csh.

**[What is the difference between the mail client and the mail server?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4254)**

Email client primarily is a desktop or mobile application that enables users to receive and send emails directly on the desktop or mobile. Typically, email client requires an email address to be set up, mail server details & connectivity to the mail server to configure and use email service. These configuration and settings include email address, password, POP3/IMAP and SMTP address, port number, email aliases, and other related preferences.

A mail server or an email server is a server that supports email function in network and support clients to handles and delivers e-mail over a network. This can be over intranet or internet. Email server receives emails from client computers and delivers them to other mail servers after proper authentication and authorization. Mail servers use MTA (Mail transfer agent) with SMTP (Simple Mail Transfer Protocol) to support email transmission. You have used any open source free MTA or any paid version the basis of your requirements and security policies.

**[Share 5 Linux commands you use on a regular basis in your environment?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4255)**

We use several commands in our day to day Linux activities and support. You can also use help or man page for a list of commands and the available options for quick reference. Most basic sand majorly uses are as below:

* **cd**: You can use this command to change your working directory. You can easily reach you your home directly by “CD /” from any working location.
* **PWD**: This command displays the full path of your current working directory. Very use full while reviewing different directories during troubleshooting or system analysis.
* **ls**: This command displays complete of files and directories in your current working directory. You can additional parameter -l for permission details and -a for hidden or config files. We have lots of option to use with ls commands to have the required output.
* **Mkdir / rmdir**: You can use “**Mkdir**” command to create a new directory and “**rmdir**”  This command to remove or delete the directory.
* **Su**: This is a special command to log in with a specific user or switches user from one to the other.

**[Explain SMTP  server in Linux with any open-source Mail transfer agent?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4256)**

SMTP(Simple Mail Transfer Protocol) is a push protocol and is used to send the mail whereas POP (post office protocol) or IMAP (internet message access protocol) are used to retrieve those emails at the receivers or client-side. The SMTP server on Linux is very fast, reliable and secure. Also, it supports POP3, IMAP and webmail access. Linux systems in a network can use the SMTP server to send alerts notifications. The mail transfer agent is an application use SMTP to transmit Email over the network. We have some of the most popularly used open-source Mail transfer agents like POSTFIX, SENDMAIL, EXIM, QMAIL, MUTT, ALPINE etc.. Each agent has its own advantages and disadvantages. You can review your system and can install the required one as per need.

Postfix is a free and open-source mail MTA (Mail Transfer Agent). This application used to send and receive the email. It is responsible for routing and delivering electronic mail. This is a cross-platform and most popular system.

**[What are network zones?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4257)**

First, We need to understand the difference and relation between the network connection and a network interface. A network interface can have many connections but one connection is only bounded to a specific network interface. Network connections are unclassified by default. It’s a system or network administrator's responsibility to create a zone with specific details to enable a level of trust by creating firewall policies.

Network Zone explains the trust level of a network connection. Creating Zones helps in identifying the secure network or unsecured network. Your system can have large access in a secure zone and limited to other zones or unsecured networks. This helps Network administrators to plan the level of monitoring for different networks.

**The initial network zones:**

|  |  |
| --- | --- |
| trusted | Fully trusted connections. All the incoming traffic is allowed. |
| home work internal | Partly trusted connections. User/administrator defines open services. |
| DMZ | Mostly untrusted connections, the demilitarized zone. |
| Public external | Mostly untrusted connections. User/administrator defines the open services. |
| block | Fully untrusted connections. No incoming traffic is allowed. |
| drop | Fully untrusted connections. All packets are dropped immediately. |

**[Explain Physical Volume, Logical volume & volume group?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4258)**

Physical volume or Physical disk is the first layer of the disk management. It’s considered as physical disks connected to the system. It can be local to the system or from SAN storage. Normally Datacenter and storage team manage it. Any disk addition and expansion can be done if space is available at this level.

A volume group is the second layer or middle layer between physical volume and logical volume. Volume group club all physical volume and display them as single storage to the system for further partitioning and usage. Due to large system in today’s environment, Application or Database need bigger space than an available physical disk. Volume group allows to clod multiple physical disk as one volume. This leverage system team to use bigger disks with any splitting at their end.

Logical volume or logical disk is the management of volume group to divide disk as per usage instead of allocation physical storage directly. Logical Volume Management (LVM) partitions can allocate across the physical drives and be resized like traditional disks.

**[What’s swap space and its usage?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4259)**

Swap space is like pagefile in windows. Swap space is virtual memory where disk reserved for swap will behave like actual RAM.

In Swap space, Some amount of physical disk to hold some transaction or data temporarily. Ideally, this data should be residing in RAM or memory but due to a situation where memory is under pressure, the system moves some transaction or data into swap space. RAM is always costlier than disk space and disk performance is increasing day by day. Physical Memory need proper management for cost-effectiveness and swap spaces help in using the disk as a physical memory or temporary and least used data. The system will consider this as a part of memory only. Swapping of memory to and from physical storage is managed by systems memory management. By default, this is an auto driven process and does not require any manual intervention.

For more accurate memory management We also have the tools to manage swap space as per our requirements

**[What is Input/output Redirection?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4260)**

By default, the standard input device is Keyboard and the standard output device is a Display screen. But to automate the processes or to pass the output of one process to another process these standard ways does not work. Linux feature of directing input & outputs data to and from processes is called Input/Output Redirection. Input / Output redirection is a required feature for good programming and shell scripting. It’s used for taking input and showing results as per requirement. Input from user and passing to other process is called input redirection whereas if process further pass its output to another process or function that will be output redirection for that and input redirection for receiving one.

 In Linux, we have three redirections available as below: -

* Input Redirection: ‘<’ symbol is used for input redirection to the new process.
* Output Redirection: ‘>’ symbol is used for output redirection to a new process or log file.
* Error Redirection: It is denoted as STDERR(2). This is very useful for error handling

**[What basics measures could you take to secure an ssh connection? For Linux users, it is frequent to access servers by ssh. But are we sure that the communication established is really good secured?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4261)**

Data & system security is one of the biggest challenges today. We need to secure our system from all possible vulnerabilities. The system should be on the basis of best practices either adopted from and best practices either from approved vendors or from inhouse from experts. SSSH has secured shell services used to connect a Linux system in a secured manner. SSH is is the most common tool for a system administrator for better system management and security. SSH gives some advance features that need proper knowledge and expertise to use. SSH gives more feature to the user.

Some of the very simple steps to secure ssh services as below:-

1. Disable/enable Root User
2. Reset Root user password
3. Disable password-based login and enable key-based logins
4. Change standard ports of the system and assign random ports
5. Restricted network access from the system
6. Use a whitelist approach to allow only authorized IPs to connect
7. **[What are system calls used for process management in Linux?](https://www.knowledgehut.com/interview-questions/linux" \l "collapse-beginner-4262)**
8. Service Calls provide a feature to use operating system services. It provides an interface between the process and the operating system for better functionality and requirements. System calls are not for beginners, it needs some level of expertise to use. System call provides additional control over the system. Processes are the most basic unit on Linux System and process management need some system calls in Linux, some of them are:-

| **System calls** | **Explanation** |
| --- | --- |
| Fork | Creates a new process |
| Exec | Execute the program |
| Wait | Force process to wait |
| Exit | Exit/terminate the process |
| Clone | Creates Child Process |
| Exit\_Group | Exit/terminate all threads in the process |
| Nice | Change the priority of the running process |
| Getppid | Find parent ID of the process |
| Vfork | Create Child Process and block parent |

#### [Explain different file permissions or access modes in Linux?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1120)

All the files and directories in Linux  have below 3 permissions, represented by a three digit octal value .-

**Read** -    It provides the ability to read the contents of a file (represented by 'r' in the first position "r--")

**Write** -   It Provides the ability to edit or delete the content of a file (represented by 'r' in the second position "-w-")

**Execute** - It Provides the ability to execute the file (represented by 'x' in the third position "--x")

The octal value is calculated as the sum of the permissions:

“read” is 4

“write” is 2

“execute” is 1

#### [Explain the usage of ps command in Linux with examples ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1119)

**ps** (i.e., process status) **command** is used to provide information about the currently running processes, including their PIDs (process identification numbers). A process is a running  instance of a program. Every process is assigned a unique PID by the.  
**Example:**

$ ps -ef

$ ps -ef | grep tomcat

#### [How to check os version in Linux command line ?](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1118)

Type any one of the following command to find os name and version in Linux:

cat /etc/os-release

lsb\_release -a

hostnamectl

Example –

**%** hostnamectl

Static hostname: WDFL41000139D

Icon name: computer-vm

Chassis: vm

Machine ID: bfc98d9a56631ccde8f8578d58347195

Boot ID: dc39baafe82849b39413507cfd395b54

Virtualization: microsoft

Operating System: SUSE Linux Enterprise Server 12 SP2

CPE OS Name: cpe:/o:suse:sles:12:sp2

Kernel: Linux 4.4.121-92.98-default

Architecture: x86-64

**%**lsb\_release -a

LSB Version: n/a

Distributor ID: SUSE

Description:  SUSE Linux Enterprise Server 12 SP2

Release:    12.2

Codename:   n/a

Also type the following command to find Linux kernel version:

**uname -r**

[**What is the difference between UNIX and LINUX?**](https://www.knowledgehut.com/interview-questions/linux#collapse-beginner-1117)

| **Unix** | **Linux** |
| --- | --- |
| Unix is an Operating System and Linux is mainly a Kernel for Linux Based OS. | Linux is a Unix-Like based OS, it means the core functionalities behind is similar to UNIX OS. |
| Unix is an operating system having some common command as that of Linux. | Linux is an operating system having some common command as that of Unix. |
| Unix uses Command Line Interface. | Linux uses Graphical User Interface with an optional Command Line Interface. |
| Unix is mainly used in Server Systems, Mainframes and High-End Computers | Linux is mainly used in Home Based PC, Mobile Phones, Desktops, etc. |
| Unix has a rigid requirement of the Hardware. Hence, cannot be installed on every other machine. | Linux is very flexible and can be installed on most of the Home Based Pcs. |
| Different Versions of Unix are: AIS, HP-UX, BSD, Iris, etc. | Different Versions of Linux are: Ubuntu, Debian, OpenSuse, Redhat, Solaris, etc. |
| Both Unix are written in C & Assembly language. | Both Linux are written in C & Assembly language. |