1. what is git ?

Git is a revision control system used to track changes in computer files. It's a tool to manage your code & file history while co-ordinating work remotely on those files with others. GitHub is a hosting service for git repositories. Git is the tool, while GitHub is the service to use git.

2. What is the difference between Git and GitHub?

Git is a revision control system, a tool to manage your source code history. GitHub is a hosting service for Git repositories. So they are not the same thing: Git the tool, GitHub the service for projects that use Git.

Git is a distributed version control system written by the creator of Linux, Linus Torvalds, after he got frustrated with proprietary software. Git is similar to other version control systems such as subversion or CVS, but it's distributed. What this means is that if you clone a git project, you have the entire project history. You can commit, branch and tag all on you local machine without interacting with a server at all. If you were working with subversion or another centralized VCS all of your interactions occur with the server.

GitHub and similar services bring all of the benefits of a decentralized VCS to a centralized service. GitHub also stores a copy of your project's repository just like any other developer. Then, you basically designate that as the project's central repository and all the developers push and pull their changes to and from that repository. GitHub takes this a step further by encouraging developers to fork a project's repository and then use that as their own centralized repository. From there they can send "pull requests" to the main project with their changes and then the project maintainers can review them before deciding whether to include them in their project or not.

3. What is scm and svm tool ?

SCM is Software Configuration Management and SVN is a Version Control System tool, which is a subset of SCM. VCS are also called Revision Control and Git, Mercurial are also VCS tools ( Distributed VCS to be more specific. ) SVN, Git, Mercurial, etc. are tools for revision control, one of the processes in SCM.

4.difference between git and other scm tools

Other scm toos

client–server model

Platforms Supported

Linux, Windows, AIX, Solaris, HP UX, i5/OS, OS/390, z/OS

Cost

Cost May or maynot

GIt

Distributed revision control systems:

Platforms Supported

POSIX, Windows, OS X

Cost

Free

**5.What is client–server model**

A computer network diagram of clients communicating with a server via the Internet.

The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.[1] Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web

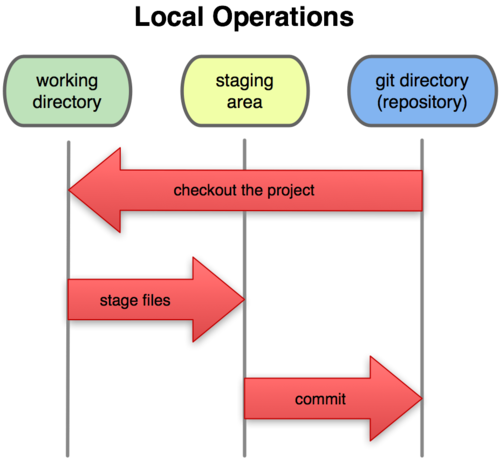
**6.What is Distributed revision control systems:**

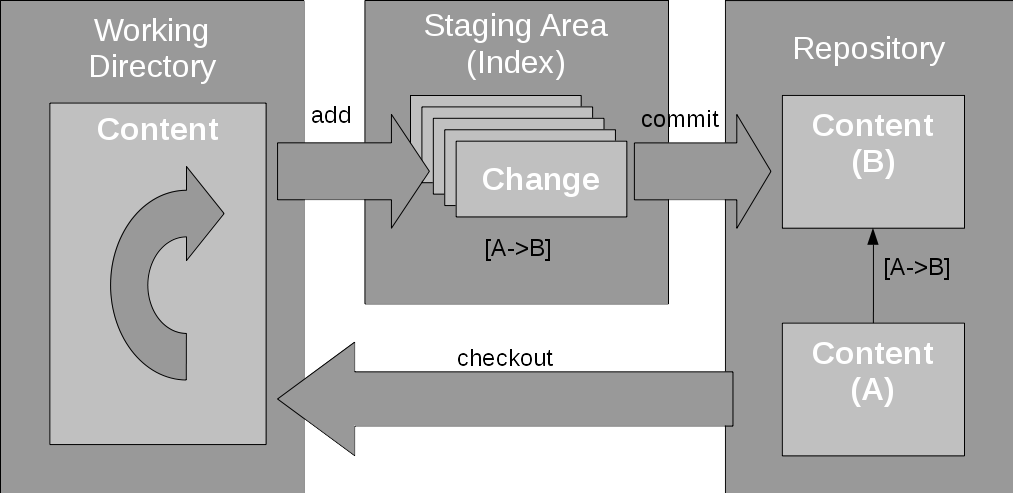
Distributed revision control systems (DVCS) takes a peer-to-peer approach to version control, as opposed to the client–server approach of centralized systems. Distributed revision control synchronizes repositories by exchanging patches from peer to peer. There is no single central version of the codebase; instead, each user has a working copy and the full change history.

**7.Git Commands**

|  |  |  |
| --- | --- | --- |
| Git task | Notes | Git commands |
| [**Tell Git who you are**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config) | Configure the author name and email address to be used with your commits.  Note that Git [strips some characters](http://stackoverflow.com/questions/26159274/is-it-possible-to-have-a-trailing-period-in-user-name-in-git/26219423#26219423) (for example trailing periods) from user.name. | git config --global user.name "Sam Smith"  git config --global user.email sam@example.com |
| [**Create a new local repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-init) |  | git init |
| [**Check out a repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone) | Create a working copy of a local repository: | git clone /path/to/repository |
| For a remote server, use: | git clone username@host:/path/to/repository |
| [**Add files**](https://www.atlassian.com/git/tutorials/saving-changes#git-add) | Add one or more files to staging (index): | git add <filename>  git add \* |
| [**Commit**](https://www.atlassian.com/git/tutorials/saving-changes#git-commit) | Commit changes to head (but not yet to the remote repository): | git commit -m "Commit message" |
| Commit any files you've added with git add, and also commit any files you've changed since then: | git commit -a |
| [**Push**](https://www.atlassian.com/git/tutorials/syncing#git-push) | Send changes to the master branch of your remote repository: | git push origin master |
| [**Status**](https://www.atlassian.com/git/tutorials/inspecting-a-repository#git-status) | List the files you've changed and those you still need to add or commit: | git status |
| [**Connect to a remote repository**](https://www.atlassian.com/git/tutorials/syncing#git-remote) | If you haven't connected your local repository to a remote server, add the server to be able to push to it: | git remote add origin <server> |
| List all currently configured remote repositories: | git remote -v |
| [**Branches**](https://www.atlassian.com/git/tutorials/using-branches) | Create a new branch and switch to it: | git checkout -b <branchname> |
| Switch from one branch to another: | git checkout <branchname> |
| List all the branches in your repo, and also tell you what branch you're currently in: | git branch |
| Delete the feature branch: | git branch -d <branchname> |
| Push the branch to your remote repository, so others can use it: | git push origin <branchname> |
| Push all branches to your remote repository: | git push --all origin |
| Delete a branch on your remote repository: | git push origin :<branchname> |
| [**Update from the remote repository**](https://www.atlassian.com/git/tutorials/syncing) | Fetch and merge changes on the remote server to your working directory: | git pull |
| To merge a different branch into your active branch: | git merge <branchname> |
| View all the merge conflicts:  View the conflicts against the base file:  Preview changes, before merging: | git diff  git diff --base <filename>  git diff <sourcebranch> <targetbranch> |
| After you have manually resolved any conflicts, you mark the changed file: | git add <filename> |
| **Tags** | You can use tagging to mark a significant changeset, such as a release: | git tag 1.0.0 <commitID> |
| CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using: | git log |
| Push all tags to remote repository: | git push --tags origin |
| [**Undo local changes**](https://www.atlassian.com/git/tutorials/undoing-changes) | If you mess up, you can replace the changes in your working tree with the last content in head:  Changes already added to the index, as well as new files, will be kept. | git checkout -- <filename> |
| Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this: | git fetch origin  git reset --hard origin/master |
| **Search** | Search the working directory for foo(): | git grep "foo()" |
| **Rebase** | git-rebase - Reapply commits on top of another base tip | Git rebase branchname |
| **Fetch** | git fetch can fetch from either a single named repository or URL, or from several repositories at once if <group> is given and there is a remotes.<group> entry in the configuration file.  When no remote is specified, by default the origin remote will be used, unless there’s an upstream branch configured for the current branch. | git fetch [<options>] [<repository> [<refspec>…​]]  git fetch [<options>] <group>  git fetch --multiple [<options>] [(<repository> | <group>)…​]  git fetch --all [<options>] |
|  |  |  |
|  |  |  |

**8.Git Architecture**





What is a git ignore?

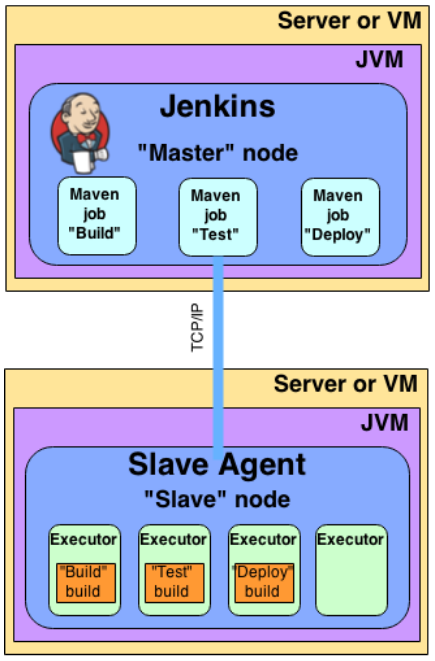
Create a local .**gitignore**. If you create a file in your repository named .**gitignore**,**Git** uses it to determine which files and directories to **ignore**, before you make a commit. A .**gitignore** file should be committed into your repository, in order to share the **ignore** rules with any other users that clone the repository.

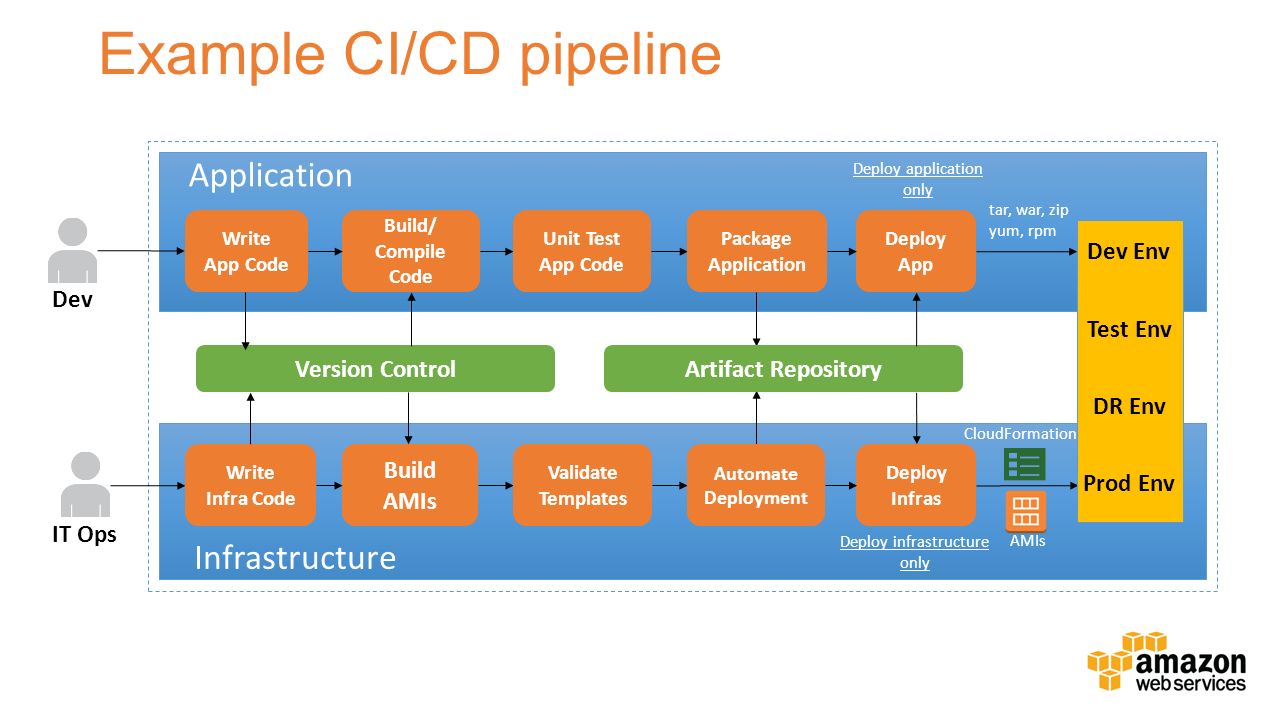
What is Jenkins?

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed

Jenkins Architecture





## Related image

## Jenkins Best Practices

[Always secure Jenkins](https://wiki.jenkins.io/display/JENKINS/Securing+Jenkins).

**This best practice is around authenticating users and enforcing access control on a Jenkins instance**  
I

In larger systems, don't build on the master.

Backup Jenkins Home regularly.

Archive unused jobs before removing them.

Setup a different job/project for each maintenance or development branch you create

Prevent resource collisions in jobs that are running in parallel.

Avoid scheduling all jobs to start at the same time

Set up email notifications mapping to ALL developers in the project, so that everyone on the team has his pulse on the project's current status

Take steps to ensure failures are reported as soon as possible.

Write jobs for your maintenance tasks, such as cleanup operations to avoid full disk problems.

Tag, label, or baseline the codebase after the successful build.

**Jenkins folders**