1. **What is the difference between build and deploy?**

**Build** means to **Compile the project**.

**Deploy** means to **Compile the project & Publish the output**

1. **What is the purpose of continuous integration for a development team?**
   * At integration stage, build failures are cached
   * For each code commit changes an automatic build report notification generates
   * To notify developers about build report success or failure, it is integrated with LDAP mail server
   * Achieves continuous integration agile development and test driven development
   * With simple steps, maven release project is automated
   * Easy tracking of bugs at early stage in development environment than production
2. **What is a version control system? What are the advantages and features of a typical version control system?**

Version control systems are a category of software tool for helping a software team manage changes to source code over time.

Advantages of Version Control Systems are:

* + Automatic backups: If you accidentally delete some file (or part of a file) you can undelete it. If you change something and want to undo it, the VCS can do so.
  + Sharing on multiple computers: VCSes are designed to help multiple people collaboratively edit text files. This makes sharing between multiple computers (say your desktop and laptop) particularly easy. You do not need to bother if you always copied the newest version; the VCS will do that for you. Even if you are offline and change files on both computers, the VCS will merge the changes intelligently once you are online.
  + Version control and branching: Say you published some class notes as a pdf and want to fix some typos in them while simultaneously working on the notes for next year. No problem. And you only need to fix the typos once, the VCS will merge them to the other versions.

1. **How PXE accomplishes its objective? What are the key advantages of PXE?**

The Preboot Execution Environment (PXE) is an industry standard client/server interface that allows networked computers that are not yet loaded with an operating system to be configured and [boot](http://searchwinit.techtarget.com/definition/boot)ed remotely by an administrator. The PXE code is typically delivered with a new computer on a [read-only memory](http://searchcio-midmarket.techtarget.com/definition/read-only-memory) chip or boot disk that allows the computer (a [client](http://searchenterprisedesktop.techtarget.com/definition/client)) to communicate with the network [server](http://whatis.techtarget.com/definition/server) so that the client machine can be remotely configured and its operating system can be remotely booted.

Advantages of PXE are:

* The client machine or workstation does not require a storage device or operating system.
* Network extension and the addition of new client computers is made easier because PXE is vendor-independent.
* Maintenance is simplified because most tasks are performed remotely.
* Centralized data storage provides information security.

1. **What are the pros and cons between centralized and distributed version control system?**

The main difference between the two classes is that Centralized VCSs keep the history of changes on a central server from which everyone requests the latest version of the work and pushes the latest changes to. This means that everyone sharing the server also shares everyone’s work. Sourceforge.net uses this type of versioning in their projects.

On the other hand, on a Distributed VCS, everyone has a local copy of the entire work’s history. This means that it is not necessary to be online to change revisions or add changes to the work. “Distributed” comes from the fact that there isn’t a central entity in charge of the work’s history, so that anyone can sync with any other team member. This helps avoid failure due to a crash of the central versioning server. Open source projects, such as Mozilla Firefox, tend to use this type of versioning.

1. **Explain various major git commands?**

Major GIT commands:

* **git config**

Sets configuration values for your user name, email, gpg key, preferred diff algorithm, file formats and more. Example: git config --global user.name "My Name" git config --global user.email "user@domain.com" cat ~/.gitconfig [user] name = My Name email = user@domain.com

* **git init**

Initializes a git repository – creates the initial ‘.git’ directory in a new or in an existing project. Example: cd /home/user/my\_new\_git\_folder/ git init

* **git clone**

Makes a Git repository copy from a remote source. Also adds the original location as a remote so you can fetch from it again and push to it if you have permissions. Example: git clone git@github.com:user/test.git

* **git add**

Adds files changes in your working directory to your index. Example: git add .

* **git rm**

Removes files from your index and your working directory so they will not be tracked. Example: git rm filename

* **git commit**

Takes all of the changes written in the index, creates a new commit object pointing to it and sets the branch to point to that new commit. Examples: git commit -m ‘committing added changes’ git commit -a -m ‘committing all changes, equals to git add and git commit’

* **git status**

Shows you the status of files in the index versus the working directory. It will list out files that are untracked (only in your working directory), modified (tracked but not yet updated in your index), and staged (added to your index and ready for committing). Example: git status # On branch master # # Initial commit # # Untracked files: # (use "git add <file>..." to include in what will be committed) # # README nothing added to commit but untracked files present (use "git add" to track)

* **git branch**

Lists existing branches, including remote branches if ‘-a’ is provided. Creates a new branch if a branch name is provided. Example: git branch -a \* master remotes/origin/master

* **git checkout**

Checks out a different branch – switches branches by updating the index, working tree, and HEAD to reflect the chosen branch. Example: git checkout newbranch

* **git merge**

Merges one or more branches into your current branch and automatically creates a new commit if there are no conflicts. Example: git merge newbranchversion

* **git reset**

Resets your index and working directory to the state of your last commit. Example: git reset --hard HEAD

* **git stash**

Temporarily saves changes that you don’t want to commit immediately. You can apply the changes later. Example: git stash Saved working directory and index state "WIP on master: 84f241e first commit" HEAD is now at 84f241e first commit (To restore them type "git stash apply")

* **git tag**

Tags a specific commit with a simple, human readable handle that never moves. Example: git tag -a v1.0 -m 'this is version 1.0 tag'

* **git fetch**

Fetches all the objects from the remote repository that are not present in the local one. Example: git fetch origin.

* **git pull**

Fetches the files from the remote repository and merges it with your local one. This command is equal to the git fetch and the git merge sequence. Example: git pull origin

* **git push**

Pushes all the modified local objects to the remote repository and advances its branches. Example: git push origin master

* **git remote**

Shows all the remote versions of your repository. Example: git remote origin

* **git log**

Shows a listing of commits on a branch including the corresponding details. Example: git log commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe Author: User <user@domain.com> Date: Mon May 3 09:24:05 2010 +0300 first commit

* **git show**

Shows information about a git object. Example: git show commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe Author: User <user@domain.com> Date: Mon May 3 09:24:05 2010 +0300 first commit diff --git a/README b/README new file mode 100644 index 0000000..e69de29

* **git ls-tree**

Shows a tree object, including the mode and the name of each item and the SHA-1 value of the blob or the tree that it points to. Example: git ls-tree master^{tree} 100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 README

* **git cat-file**

Used to view the type of an object through the SHA-1 value. Example: git cat-file -t e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 blob

* **git grep**

Lets you search through your trees of content for words and phrases. Example: git grep "www.siteground.com" -- \*.php

* **git diff**

Generates patch files or statistics of differences between paths or files in your git repository, or your index or your working directory. Example: git diff

* **git archive**

Creates a tar or zip file including the contents of a single tree from your repository. Example: git archive --format=zip master^ README >file.zip

* **git prune** Removes objects that are no longer pointed to by any object in any reachable branch. Example: git prune

1. **What are Jenkins Best Practices?**

Jenkins best practices:

* Secure Jenkins
* Treat your master(s)
* Backup Jenkins home
* Choose and maintain your plugins
* Schedule jobs
* Break jobs down to pieces
* Jenkins need free disk-space
* Limit project names to a sane (e.g. alphanumeric) character set
* The most reliable builds will be clean builds, which are built fully from Source Code Control.
* Always configure your job to generate trend reports and automated testing when running a Java build
* Archive unused jobs before removing them.

1. **What is the Puppet configuration management tool, and how does it work?**

Puppet is an open source IT automation tool that allows IT organizations to encode the configuration of services as a policy, which the framework then audits and enforces.

[Puppet configuration consists of](http://searchitoperations.techtarget.com/news/450401507/Puppet-Enterprise-features-appeal-but-upgrades-a-roadblock) a language, client-server processes and the Resource Abstraction Layer.

The language allows the description of a server configuration with an abstraction of the resources that an administrator already thinks in: users, groups, packages, files, cron, mount and services, to name a few.

The relationships between the resources are also specified. For example, a service depends on a configuration file, and that file depends on a package being installed. The relationships provide order as the policy is applied and allow Puppet to restart dependent services when their configurations change.