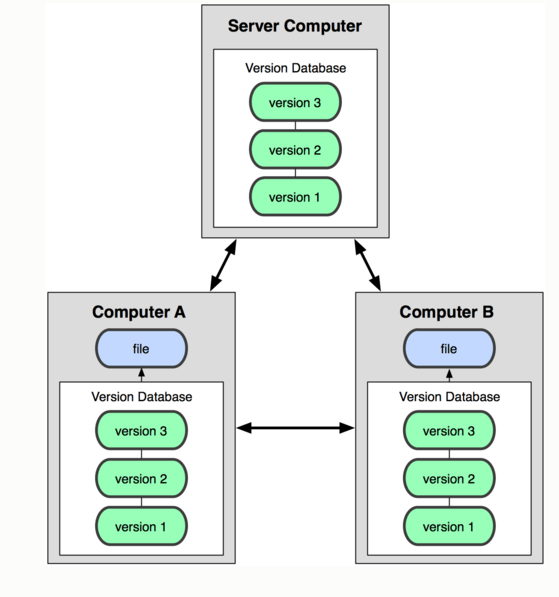
**GIT**

* GIT is a Distributed Version Control System (DVCS).
* CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”
* Mercurial and Git treat all repositories as equal
* In a DVCS (such as Git, Mercurial, Bazaar or Darcs ), clients don’t just check out the latest snapshot of the files: they fully mirror the repository.
* Unlike Centralized Version Control Systems (CVCSs), the distributed nature of Git allows you to be far more flexible in how developers collaborate on projects. In centralized systems, every developer is a node working equally on a central hub.
* In Git, however, every developer is potentially both a node and a hub – that is, every developer can both contribute code to other repositories and maintain a public repository on which others can base their work and which they can contribute to.
* This opens a vast range of workflow possibilities for your project and/or your team, so we’ll cover a few common paradigms that take advantage of this flexibility.
* We’ll go over the strengths and possible weaknesses of each design; you can choose a single one to use, or you can mix and match features from each.



**GIT Repository: -**

You can get a Git project using two main approaches. The first takes an existing project or directory and imports it into Git. The second clones an existing Git repository from another server.

**GIT cloning:-**

* Clone from upstream URL
* Copies complete upstream repository into a local repository.
* Creates origin tracking branch.
* Tracks upstream master branch
* Introduces master development branch
* Initially the same place as origin

If you want to get a copy of an existing Git repository – for example, a project you’d like to contribute to – the command you need is Git clone . If you’re familiar with other VCS systems such as Subversion, you’ll notice that the command is "clone" and not "checkout". This is an important distinction – instead of getting just a working copy, Git receives a full copy of nearly all data that the server has. Every version of every file for the history of the project is pulled down by default when you run Git clone. In fact, if your server disk gets corrupted, you can often use nearly any of the clones on any client to set the server back to the state it was in when it was cloned

**Git Configuration:-**

The configuration options recognized by Git fall into two categories: client-side and server-side. Most of the options are client-side – configuring your personal working preferences. Many configuration options are supported, but a large fraction of them are only useful in certain edge cases. We’ll only be covering the most common and most useful here. If you want to see a list of all the options your version of Git recognizes, you can run

**Git branch: -**

When you make a commit, Git stores a commit object that contains a pointer to the snapshot of the content you staged. This object also contains the author’s name and email, the message that you typed, and pointers to the commit or commits that directly came before this commit (its parent or parents): zero parents for the initial commit, one parent for a normal commit, and multiple parents for a commit that results from a merge of two or more branches.

List, create, or delete branches.

$ Git branch testing

**Git tag: -**

Create, list, delete or verify a tag. Add a tag reference in refs/tags/

$ Git tag testing

**Git checkout: -**

Checkout a branch or paths to the working tree.

$ Git checkout <branch>

**Git add:**

Add file contents to the index

$ Git add build.xml

**Git commit-:**

Record changes to the repository

$Git commit -m “build file”

**Git pull remote repository *:-***

Get changes from a remote repository and merge them into your own repository

**Git status**

The main tool you use to determine which files are in which state is the git statuscommand. If you run this command directly after a clone

**GIT HUB**

GitHub is the single largest host for Git repositories, and is the central point of collaboration for millions of developers and projects. A large percentage of all Git repositories are hosted on GitHub, and many open-source projects use it for Git hosting, issue tracking, code review, and other things. So while it’s not a direct part of the Git open source project, there’s a good chance that you’ll want or need to interact with GitHub at some point while using Git professionally.

**GIT Protocol**

This is a special daemon that comes packaged with Git; it listens on a dedicated port (9418) that provides a service similar to the SSH protocol, but with absolutely no authentication. In order for a repository to be served over the Git protocol, you must create the git-daemon-export-ok  file – the daemon won’t serve a repository without that file in it – but other than that there is no security. Either the Git repository is available for everyone to clone or it isn’t. This means that there is generally no pushing over this protocol. You can enable push access; but given the lack of authentication, if you turn on push access, anyone on the internet who finds your project’s URL could push to your project. Suffice it to say that this is rare.

**GIT Bugs:**

Bugs in Git can be reported directly to the mailing list (see above for details). Note that you do not need to subscribe to the list to send to it. You can help us out by attempting to reproduce the bug in the latest released version of Git, or if you're willing to build Git from source, the next branch. Sometimes an attempted fix may be pending in this branch, in which case your feedback as to whether the fix worked for you will be appreciated.