Branching and Merging:-

* Frictionless Context Switching .Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
* Role-Based Code lines. Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work
* Feature Based Workflow . Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
* Disposable Experimentation . Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime).

1. Notably, when you push to a remote repository, you do not have to push all of your branches. You can choose to share just one of your branches, a few of them, or all of them. This tends to free people to try new ideas without worrying about having to plan how and when they are going to merge it in or share it with others.
2. There are ways to accomplish some of this with other systems, but the work involved is much more difficult and error-prone. Git makes this process incredibly easy and it changes the way most developers work when they learn it.

Small and Fast :-

* Git is fast. With Git, nearly all operations are performed locally, giving it a huge speed advantage on centralized systems that constantly have to communicate with a server somewhere.
* Note that this is the best case scenario for SVN - a server with no load with an 80MB/s bandwidth connection to the client machine. Nearly all of these times would be even worse for SVN if that connection was slower, while many of the Git times would not be affected.
* Git was built to work on the Linux kernel, meaning that it has had to effectively handle large repositories from day one. Git is written in C, reducing the overhead of runtimes associated with higher-level languages. Speed and performance has been a primary design goal of the Git from the start.
* Git is one or two orders of magnitude faster than SVN</strong>, even under ideal conditions for SVN.
* One place where Git is slower is in the initial clone operation.Here, Git is downloading the entire history rather than only the latest
* version. As seen in the above charts, it's not considerably slower for an operation that is only performed once.

Distributed:-

* One of the nicest features of any Distributed SCM, Git included, is that it's distributed. This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.

Multiple Backups

* This means that even if you're using a centralized workflow, every user essentially has a full backup of the main server. Each of these copies could be pushed up to replace the main server in the event of a crash or corruption. In effect, there is no single point of failure with Git unless there is only a single copy of the repository.

Any Workflow

* Because of Git's distributed nature and superb branching system, an almost endless number of workflows can be implemented with relative ease.

Subversion-Style Workflow

* A centralized workflow is very common, especially from people transitioning from a centralized system. Git will not allow you to push if someone has pushed since the last time you fetched, so a centralized model where all developers push to the same server works just fine.

Data Assurance:-

* The data model that Git uses ensures the cryptographic integrity of every bit of your project. Every file and commit is checksummed and retrieved by its checksum when checked back out. It's impossible to get anything out of Git other than the exact bits you put in
* It is also impossible to change any file, date, commit message, or any other data in a Git repository without changing the IDs of everything after it.This means that if you have a commit ID, you can be assured not only that your project is exactly the same as when it was committed, but that nothing in its history was changed.

Most centralized version control systems provide no such integrity by default.

Free and Open Source:-