**Differences between GIT and SVN:**

1. **GIT is distributed while SVN is centralized**

This is by far the **\*core\*** difference between GIT and other non-distributed version control systems like SVN, CVS etc. If you can catch this concept well, then you have crossed half the bridge. To add a disclaimer, GIT is not the first or only distributed VCS(version control system) currently available. There are other tools like [Bitkeeper](http://www.bitkeeper.com/" \t "_new), [Mercurial](http://mercurial.selenic.com/) etc. which also work on distributed mode. But, GIT does it better and comes with much more powerful features.

GIT like SVN do have centralized repository or server. But, GIT is more intended to be used in distributed mode which means, every developers checking out code from central repository/server will have their own cloned repository installed on their machine. Let’s say if you are stuck somewhere where you don’t have network connectivity, like inside the flight, basement, elevator etc, you will still be able to commit files, look at revision history, create branches etc. This may sound trivial for lot of people but, it is a big deal when you often bump into no-network scenario.  
And also, the distributed mode of operation is a biggest blessing for open-source software development community. Instead of creating patches & sending it through emails, you can create a branch & send a pull request to the project team. It will help the code stay streamlined without getting lost in transport. [GitHub.com](http://www.github.com/) is an awesome working example of that.

There were some rumors that the future version of subversion will be working on distributed mode. But, it’s still an unknown at this point.

1. **GIT stores content as metadata, SVN stores just files:**

Every source control systems stores the metadata of files in hidden folders like .svn, .cvs etc., whereas GIT stores entire content inside the .git folder. If you compare the size of .git folder with .svn, you will notice a big difference. So, the .git folder is the cloned repository in your machine, it has everything that the central repository has like tags, branches, version histories etc.

1. **GIT branches are not the same as SVN branches:**

Branches in SVN are nothing but just another folder in the repository. If you need to know if you had merged a branch, you need to explicitly run commands like *[svn propget svn:mergeinfo](http://jan.baresovi.cz/dr/en/subversion-mergeinfo" \t "_new)* to verify if it was merged or not. [Thanks Ben](http://boxysystems.com/index.php/5-fundamental-differences-between-git-svn/#comment-791)for pointing this feature :).  
So, the chance of adding up orphan branches is pretty big.

Whereas, working with GIT branches is much more easy & fun. You can quickly switch between branches from the same working directory. It helps finding un-merged branches and also help merging files fairly easily & quickly.

1. **GIT does not have a global revision no. like SVN do:**

This is one of the biggest feature I miss in GIT from SVN so far. As you may know already SVN’s revision no. is a snapshot of source code at any given time. I consider that as a biggest breakthrough moving from CVS to SVN.  
Since, GIT & SVN are conceptually different, I don’t know how you can mirror that feature in GIT.

*Update: You can use GIT’s SHA-1 hash key to uniquely identify the code snapshot. It may not exactly replace SVN’s easily readable numeric revision no. but, it kind of serves the same purpose.*

1. **GIT’s content integrity is better than SVN’s:**

GIT contents are cryptographically hashed using [SHA-1](http://en.wikipedia.org/wiki/SHA-1) hash algorithm. This will ensure the robustness of code contents by making it less prone to repository corruption due to disk failures, network issues etc.

**Centralized Version Control:**

There are many version control systems out there. Often they are divided into two groups: “centralized” and “distributed”.

Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy.

“Committing” a change simply means recording the change in the central system. Other programmers can then see this change. They can also pull down the change, and the version control tool will automatically update the contents of any files that were changed.

Most modern version control systems deal with “change sets,” which simply are a group of changes (possibly to many files) that should be treated as a cohesive whole. For example: a change to a C header file and the corresponding .c file should always be kept together.

Centralized version control solves the problems described in the [previous post on What is Version Control?](http://blogs.atlassian.com/2012/02/version-control-diffs-patches/). Programmers no longer have to keep many copies of files on their hard drives manually, because the version control tool can talk to the central copy and retrieve any version they need on the fly.

Some of the most common centralized version control systems you may have heard of or used are CVS, Subversion (or SVN) and Perforce.

**A Typical Centralized Version Control Workflow:**

When you’re working with a centralized version control system, your workflow for adding a new feature or fixing a bug in your project will usually look something like this:

* Pull down any changes other people have made from the central server.
* Make your changes, and make sure they work properly.
* Commit your changes to the central server, so other programmers can see them.

**Distributed Version Control:**

In the past five years or so a new breed of tools has appeared: so-called “distributed” version control systems (DVCS for short). The three most popular of these are Mercurial, [Git](http://www.atlassian.com/git/) and Bazaar.

Distributed means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.

These systems do not necessarily rely on a central server to store all the versions of a project’s files. Instead, every developer “clones” a copy of a repository and has the **full** history of the project on their hard drive. This copy (or “clone”) has *all* of the metadata of the original.

This method may sound wasteful, but in practice, it’s not a problem. Most programming projects consist mostly of plain text files (and maybe a few images), and disk space is so cheap that storing many copies of a file doesn’t create a noticeable dent in a hard drive’s free space. Modern systems also compress the files to use even less space.

The act of getting new changes from a repository is usually called “pulling,” and the act of moving your own changes to a repository is called “pushing”. In both cases, you move change sets (changes to files groups as coherent wholes), not single-file diffs.

One common misconception about distributed version control systems is that there *cannot* be a central project repository. This is simply not true – there is nothing stopping you from saying “this copy of the project is the authoritative one.” This means that instead of a central repository being *required* by the tools you use, it is now optional and purely a social issue.

**Advantages over Centralized Version Control:**

The act of cloning an entire repository gives distributed version control tools several advantages over centralized systems:

* Performing actions other than pushing and pulling change sets is *extremely* fast because the tool only needs to access the hard drive, not a remote server.
* Committing new change sets can be done locally without anyone else seeing them. Once you have a group of change sets ready, you can push all of them at once.
* Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won’t be forced to commit several bug fixes as one big change set.
* Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

**Disadvantages Compared to Centralized Version Control:**

To be quite honest, there are almost no disadvantages to using a distributed version control system over a centralized one. Distributed systems do *not* prevent you from having a single “central” repository, they just provide more options on top of that.

There are only two major inherent disadvantages to using a distributed system:

* If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly.
* If your project has a very long history (50,000 change sets or more), downloading the entire history can take an impractical amount of time and disk space.

The authors and contributors of modern distributed version control systems are working on solving these problems, but at the moment, no bundled, built-in features solve them.