Version Contol Tools

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

A graphic or web designer may want to keep every version of an image or layout; a Version Control System (VCS) is very helpful in such cases. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

Various Version Control Systems

SVN

Subversion is developed as a project of the Apache Software Foundation, and as such is part of a rich community of developers and users. Subversion is an open source version control system. Founded in 2000 by CollabNet, Inc., the Subversion project and software have seen incredible success over the past decade. Subversion has enjoyed and continues to enjoy widespread adoption in both the open source arena and the corporate world.

Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance.

Bazaar

GNU Bazaar is a distributed and client–server revision control system sponsored by Canonical. Bazaar can be used by a single developer working on multiple branches of local content, or by teams collaborating across a network. Bazaar is a version control system that helps you track project history over time and to collaborate easily with others.

Comparison between the version control tools mentioned.

1. Subversion and Git

   1.1. Differences

      a. Git is much faster than Subversion

      b. Subversion allows you to check out just a subtree of a repository; Git requires you to clone the entire repository (including history) and create a working copy that mirrors at least a subset of the items under version control.

c. Git's repositories are much smaller than Subversions

d. Git was designed to be fully distributed from the start, allowing each developer to have full local control.

e. Git branches are simpler and less resource heavy than Subversion's

f. Git branches carry their entire history

g. Git provides better auditing of branch and merge events

h. Git's repo file formats are simple, so repair is easy and corruption is rare.

i. Backing up Subversion repositories centrally is potentially simpler - since you can choose to distributed folders within a repo in git

j. Git repository clones act as full repository backups

k. Subversion's UI is more mature than Git's

l. Walking through versions is simpler in Subversion because it uses sequential revision numbers (1,2,3,..); Git uses unpredictable SHA-1 hashes. Walking backwards in Git is easy using the "^" syntax, but there is no easy way to walk forward.

1.2. Git's Major Features Over Subversion

a. Distributed Nature

Git was designed from the ground up as a distributed version control system. In a distributed VCS like Git every user has a complete copy of the repository data stored locally, thereby making access to file history extremely fast, as well as allowing full functionality when disconnected from the network. It also means every user has a complete backup of the repository. If any repository is lost due to system failure only the changes which were unique to that repository are lost. If users frequently push and fetch changes with each other this tends to be a small amount of loss, if any.

In a centralized VCS like Subversion only the central repository has the complete history. This means that users must communicate over the network with the central repository to obtain history about a file. Backups must be maintained independently of the VCS. If the central repository is lost due to system failure it must be restored from backup and changes since that last backup are likely to be lost.

b. Access Control

Due to being distributed, you inherently do not have to give commit access to other people in order for them to use the versioning features. Instead, you decide when to merge what from whom.

In git, users are able to have version control of their own work while the source is controlled by the repo owner.

c. Branch Handling

Branches in Git are a core concept used everyday by every user. In Subversion they are more cumbersome and often used sparingly.

The reason branches are so core in Git is every developer's working directory is itself a branch. Even if two developers are modifying two different unrelated files at the same time it's easy to view these two different working directories as different branches stemming from the same common base revision of the project.

d. Performance (Speed of Operation)

Git is extremely fast. Since all operations (except for push and fetch) are local there is no network latency involved to:

- Perform a diff.
- View file history.
- Commit changes.
- Merge branches.
- Obtain any other revision of a file (not just the prior committed revision).
- Switch branches.

e. Smaller Space Requirements

Git's repository and working directory sizes are extremely small when compared to SVN.

One of the reasons for the smaller repo size is that an SVN working directory always contains two copies of each file: one for the user to actually work with and another hidden in .svn/ to aid operations such as status, diff and commit. In contrast a Git working directory requires only one small index file that stores about 100 bytes of data per tracked file. On projects with a large number of files this can be a substantial difference in the disk space required per working copy.

As a full Git clone is often smaller than a full checkout, Git working directories (including the repositories) are typically smaller than the corresponding SVN working directories.

f. Line Ending Conversion

Git's advantage over Subversion is that you do not have to manually specify which files this conversion should be applied to, it happens automatically (hence autocrlf).

1.3. Subversion's Major Features Over Git

a. User Interfaces Maturity

Currently Subversion has a wider range of user interface tools than Git. For example there are SVN plugins available for most popular IDEs. Also Windows Explorer shell extension along with a number of native Windows and Mac OS X GUI tools available in ready-to-install packages.

Whereas Git's primary user interface is through the command line. There are two graphical interfaces: git-gui (distributed with Git) and qgit, which is making great strides towards providing another feature-complete graphical interface. There are some user interface tools in development for Git, namely TortoiseGit, a port of TortoiseSVN. There is also Git Extensions, another explorer shell extension.

b. Single Repository

Since Subversion only supports a single repository there is little doubt about where something is stored. Once a user knows the repository URL they can reasonably assume that all materials and all branches related to that project are always available at that location. Backup to tape/CD/DVD is also simple as there is exactly one location that needs to be backed up regularly.

Since Git is distributed by nature not everything related to a project may be stored in the same location. Therefore there may be some degree of confusion about where to obtain a particular branch, unless repository location is always explicitly specified. There may also be some confusion about which repositories are backed up to tape/CD/DVD regularly, and which aren't.

c. Binary Files

Detection and Properties:

Subversion can be used with binary files (it is automatically detected; if that detection fails, you have to mark the file binary yourself). Just like Git.

Only that with Git, the default is to interpret the files as binary to begin with. If you _have_ to have CR+LF line endings, you have to tell Git so. Git will then autodetect if a file is text (just like Subversion), and act accordingly. Analogous to Subversion, you can correct an erroneous autodetection by setting a git attribute.

Change Tracking:

In an earlier version of git seemingly minor changes to binary files, such as adjusting brightness on an image, could be different enough that Git interprets them as a new file, causing the content history to split. Since Subversion tracks by file, history for such changes is maintained.

d. Partial Checkout/Bandwidth Requirements

With Subversion, you can check out just a subdirectory of a repository. This is not possible with Git. For a large project, this means that you always have to download the whole repository, even if you only need the current version of some sub-directory.

In other cases, requirements other than the raw repository size provide the motivation for wanting a partial checkout, e.g. access control (you can't restrict read access to part of the repository with Git) or directory layout requirements. There is no general solution for this problem other than to split the original Git repository into multiple repositories, then cloning one of the new repositories.

e. Shorter and Predictable Revision Numbers

First, as SVN assigns revision numbers sequentially (starting from 1) even very old projects such as Mozilla have short unique revision numbers (Mozilla is only up to 6 digits in length). Many users find this convenient when entering revisions for historical research purposes. They also find this number easy to embed into their product, supposedly making it easy to determine which sources were used to create a particular executable.

As Git uses a SHA1 to uniquely identify a commit each specific revision can only be described by a 40 character hexadecimal string, however this string not only identifies the revision but also the branch it came from. In practice the first 8 characters tends to be unique for a project, however most users try to not rely on this over the long term. Rather than embedding long commit SHA1s into executables Git users generate a uniquely named tag.

Secondly, SVN's revision numbers are predictable. If the current commit is 435 the next one will be 436. It's very easy then to go through a few sequential revisions to, e.g. look at differences, revert to an old revision to find when a regression was introduced, etc. Furthermore, without looking up any additional information, you know that commit 436 was done after 435. Similar actions and knowledge from git requires looking at the log.

f. Ability to Represent Richer Histories

The benefits of Git's branch and merge handling that are mentioned above come with a downside that can occasionally surface: they slightly restrict your freedom. For instance, it is not possible to commit to two branches at once with Git, but it is in Subversion. There are other restrictions as well;

Git may sometimes be less flexible than CVS and Subversion.

CVS allows a branch or tag to be created from arbitrary combinations of source revisions from multiple source branches. It even allows file revisions that were never contemporaneous to be added to a single branch/tag. Git, on the other hand, only allows the full source tree, as it existed at some instant in the history, to be branched or tagged as a unit. Moreover, the ancestry of a git revision makes implications about the contents of that revision. This difference means that it is fundamentally impossible to represent an arbitrary CVS history in a git repository 100% faithfully."

2. Bazaar and Git

2.1. Differences

a. Git was designed for the needs of the Linux kernel community, to match the processes applicable there and the brilliance of the people executing them.

b. Bazaar, on the other hand, was designed to be suitable for a wide range of people, workflows and environments.

c. In Git the features were the primary focus, whereas in Bazaar the clean user interface and command set were given careful attention.

d. Bazaar has a wider audience in mind. Whereas Git may well be the right choice for a specialized community or team.

e. Bazaar in contrast to Git is platform independent and will work without problems in mixed development environments---anywhere where Python is available.

f. Bazaar is slower compared to Git.

g. Git is more storage efficient when comared with Bazaar.

2.2. Bazaar's major features over Git

a. Windows Support

Windows occupies 85% of the operating system market. While it is possible to run Git on top of Windows using the Cygwin layer, that would bring a mixed operating system environment into the picture for regular Windows users. Cygwin is a POSIX environment where handling of case insensitivity and filename globbing are different from what Windows users expect — not to mention the commands.

Git currently has beta-level native Windows support, code-named MSysGit. However the complete command set is not yet supported, many open problems remain, and the installer package is available only as a third-party project.

Bazaar, on the other hand, includes a native Windows port and installer. The port feels like a regular Windows program, and the installer includes the graphical front end for Windows, TortoiseBzr. Olive is available as a graphical front end for Linux.

b. Direct Support for more workflows

There's more than one way to collaborate together. The best way depends on a whole range of factors. Your team or community will have one primary workflow model — and the one encouraged by Git is a very good one — but different groups within your community will undoubtedly mix and match as circumstances dictate. Bazaar's UI directly supports a larger range of work flows than Git.

Take for example a centralized SVN-style workflow, which is well supported in Bazaar's command set. In Bazaar, it is possible to commit directly to the central server. In Git it would need two actions: a local commit, followed by a push to remote host. Similarly Bazaar supports SVN-style checkout, whereas in Git you may have to download whole — possibly big — repository.

In principle any work flow is possible in Git, but the actions are usually more demanding and less intuitive than in Bazaar.

c. User Interface

Users want tools that stay out of their way. Bazaar generally achieves this better than Git. Git's strength is a simple core data storage model, but it has weaknesses on the UI level. Git quickly gets complicated with concepts like staging area, dangling objects, detached heads, etc — concepts that you don't need to know at all in Bazaar.

The key differences between the UI of the two tools are:

- Directories are branches. In Git they are branch containers where you switch to different views.
- Empty directories cannot be versioned in Git.
- Within a branch, changes directly made are emphasized over changes from a merge.
- Revision objects are simple: r1, r2 etc. In git they are SHA1s which are represented by 40 character long hexadecimal encoding.
- Git's automatic merge and commit may create problems.
- Git has over 150 different commands. The UI between these commands is not consistent, and there is no unified GNU --longoption convention support.
- Bazaar uses familiar commands known to Subversion and CVS users. Git contains a whole new vocabulary: for example, commit into repository is very different in Git.

d. Better Storage Model

Bazaar can efficiently share revisions between branches through shared repositories. These are completely optional — a standalone tree has its own repository by default. But if a parent directory has been configured as a shared repository, the revisions are stored and shared there.

For efficiency, this is actually far more flexible and powerful than Git's default model. For example, the one repository can be used on a developer workstation for storing revisions from:

- release branches
- a tracking branch of the main development trunk
- topic branches for each fix or feature currently being worked on
- temporary QA or integration branches for reviewing other changes.

e. Easier Administration

Bazaar's directory-is-a-branch model has administration advantages. Security can be applied to different branches by using existing operating system access control facilities. Bazaar's plug-in architecture is also a good thing w.r.t. cost of ownership. Plugins are typically easier to upgrade and share than enhancements made outside such a framework.

In a commercial environment, one of the arguments against adopting DVCS tools is that central VCS tools encourage daily check-ins which fits in well with the daily backup cycle. That's the wrong trade-off — it leads to fragile code being checked in, trunk quality dropping, other developers grabbing those changes before they are ready, and lost time all around. That's bad enough for a centrally located in-house team. For distributed teams and open source communities, it’s even worse. The right solution for backing up the work of developers using a DVCS is a central backup server that developers can push changes to daily. Bazaar’s shared repositories can be useful on that backup server. In particular, while Bazaar users with laptops are in the office, they can alternatively bind branches to ones on that server if they want backups to happen implicitly.

Both Bazaar and Git have tools for detecting junk and inconsistencies in repositories. Bazaar also has an upgrade tool for switching between file formats and a reconfigure tool for changing how a branch is configured, e.g. lightweight vs heavyweight, standalone vs shared repository.

f. Robust renaming

Git prides itself on being a “content manager” and deriving what got renamed using heuristics. This mostly works, but breaks under certain merge conditions. If you want your team or community to collaborate without fear of breaking merges, Bazaar's robust renaming is essential.

g. Better asynchronous sharing

When changes in a branch are ready for sharing and you wish to share asynchronously (e.g. via email instead of advertising a public branch), Bazaar handles this better than Git. The recommended way to do this in Git is via the format-patch command which generates a set of normal patches which can be applied with its am command. Bazaar implements this via the send command, which generates an intelligent patch known as a "merge directive".

2.3. Git’s major features over Bazaar

a. Speed

It's true that Git is really fast at many operations and that Bazaar was once quite slow. It is still a bit slow for big histories (> 10 000 changesets). Git is also likely to be faster than Bazaar for network operations.

b. Storage Efficiency

Git has a strong reputation for efficiently storing data and claims on its home page that it tops every other open source VCS in this area.

On thorough testing on repositories with long history is needed to confirm storage efficiency. There are some indications that Git is more efficient in repositories with long history. Git repositories can be packed very efficiently with clever tweaking. Subsequent revisions added to Git repository will cause the repository to grow, but recent Git releases will repack the repository regularly and automatically.

c. Cryptographic Content Validation

Linus made cryptographic strength integrity checking a core part of Git’s design. As revisions are named using their SHA, it's next to impossible to attack a Git repository. Bazaar explicitly chose to make its revision identifiers UUIDs instead of SHAs. This doesn't mean that Bazaar is less secure — the integrity of each revision is still validated using SHAs in Bazaar. If that isn’t enough security, Bazaar can be configured to digitally sign every commit.

VCS used for the project

For the project on Doctor Appointment Booking System the Git VCS would be used.

Name: Snigdha Taduru

Roll No: 15IT149