Name: Shivani Shrivastava
Roll no.: 15IT243

SOFTWARE ENGINEERING ASSIGNMENT 3

Comparison of Version Control Tools

Features	**Subversion**	**Git**	**Mercurial**
Repository Model	Client-server	Distributed	Distributed
Concurrency model	Merge or lock	Merge	Merge
Cost	Free	Free	Free
Platforms supported	Unix like, Windows, OS X	POSIX, Windows, OS X	Unix like, Windows, OS X
Storage Method	Changeset and Snapshot	Snapshot	Changeset
Scope of change	Tree	Tree	Tree
Network Protocols	custom(*svn*), custom over ssh, HTTP and SSL(using WebDAV)	custom(*git*), custom over ssh, HTTP/HTTPS, rsync, email, bundles	custom over ssh, HTTP, email bundles (with standard plugin)
Atomic commits	Yes	Yes	Yes
File renames	Yes	Partial	Yes
Merge file renames	Partial	Yes	Yes
Signed revisions	No	Yes	Yes
Merge tracking	Yes	Yes	Yes
End of line conversions	Yes	Yes	Yes

Tags	Partial	Yes	Yes
International support	Yes	Yes	Yes
Unicode filename support	Yes	Yes	Partial
Support large repos	Yes	Partial	Yes
Web interfaces	Apache 2 module included, WebSVN, ViewSVN, ViewVC, Trac, SharpForge, sventon, Springloops	Gitweb, wit, cgit, GitLab, GitHub, gitorious, Trac, Kallithea, Bitbucket, Stash, Springloops, Bonobo Git Server	Bitbucket, Trac, Kallithea
Stand-alone GUIs	Java, KDESVN, OS X[67] (including Finder integration), Nautilus, Qt, RabbitVCS, RapidSVN, SourceTree (OS X), TortoiseSVN (Windows Explorer)	gitk, git-gui (Tcl/Tk), tig, Gitbox (OS X), TortoiseGit, qgit, gitg (GNOME/GTK), (h)gct (Qt), git-cola (Qt), Git Extensions (Windows), GitEye, SmartGit/Hg, Tower, SourceTree (OS X/Windows), Sprout (OS X), GitX (OS X), GitUp (OS X), GitKraken	Hgk (Tcl/Tk), (h)gct (Qt), TortoiseHg (Windows Explorer, Nautilus), MacHg, MacMercurial, Murky, SourceTree (Windows/OS X), TortoiseHg, SmartGit/Hg
Integration and/or plug-ins for IDEs	Anjuta, BBEdit, Eclipse (Subclipse, Subversive), Emacs (standard VC), IntelliJ IDEA (standard in Community and Ultimate Editions), KDevelop (standard), Komodo IDE, MonoDevelop (standard), Netbeans, RabbitVCS (for GEdit), TextMate (SVNMate plugin),	Aptana 3 Beta (Aptana Studio 3 with Git Integration); Eclipse (JGit/EGit); Helix TeamHub; Netbeans (NbGit); KDevelop; Visual Studio (Git Extensions); Emacs (extension for standard VC); SAP Web IDE; TextMate (Git TextMate Bundle); Vim (VCSCommand	IntelliJ IDEA (hg4idea 3rd party plugin), Eclipse (Mercurial Eclipse), NetBeans, Visual Studio 2008, Emacs, Vim (VCSCommand plugin), Komodo IDE, Eric Python IDE, WingIDE

	Visual Studio (AnkhSVN, VisualSVN), WingIDE. See also Comparison of Subversion clients	plugin and fugitive plugin); IntelliJ IDEA >8.1 (standard in Community and Ultimate Editions); Komodo IDE; Anjuta; XCode, WingIDE	
Command aliases	No	[in '.gitconfig' file]	[in '.hgrc' file]
Lock/unlock	lock/unlock	No	No
Rollback	No	reset HEAD^	strip (bundled extension)
Cherry-picking	svnmerge cherry-picking	cherry-pick	graft(core) or transplant(bundled extension)
Incoming/outgoing	status -u	cherry	incoming/outgoing
Grep	No	grep	grep
Record	No	add -p	record (bundled extension)

Table explanation:

- Repository model describes the relationship between various copies of the source code repository. In a client–server model, users access a master repository via a client; typically, their local machines hold only a working copy of a project tree. Changes in one working copy must be committed to the master repository before they are propagated to other users. In a distributed model, repositories act as peers, and users typically have a local repository with version history available, in addition to their working copies.

- Concurrency model describes how changes to the working copy are managed to prevent simultaneous edits from causing nonsensical data in the repository. In a lock model, changes are disallowed until the user requests and receives an exclusive lock on the file from the master repository. In a merge model, users may freely edit files, but are informed of possible conflicts upon checking their changes into the repository,

whereupon the version control system may merge changes on both sides, or let the user decide when conflicts arise. Note that distributed version control almost always implies a merge concurrency model.

- Storage Method: Describes the form in which files are stored in the repository. A snapshot indicates that a committed file(s) is stored in its entirety—usually compressed. A changeset, in this context, indicates that a committed file(s) is stored in the form of a difference between either the previous version or the next.

- Scope of change: Describes whether changes are recorded for individual files or for entire directory trees.

- Network protocols: lists the protocols used for synchronization of changes.

- Atomic commits: refers to a guarantee that all changes are made, or that no change at all will be made.

- File renames: describes whether a system allows files to be renamed while retaining their version history.

- Merge file renames: describes whether a system can merge changes made to a file on one branch into the same file that has been renamed on another branch (or vice versa). If the same file has been renamed on both branches then there is a rename conflict that the user must resolve.

- Signed revisions: refers to integrated digital signing of revisions, in a format such as OpenPGP.

- Merge tracking: describes whether a system remembers what changes have been merged between which branches and only merges the changes that are missing when merging one branch into another.

- End of line conversions: describes whether a system can adapt the end of line characters for text files such that they match the end of line style for the operating system under which it is used. The granularity of control varies. Subversion, for example, can be configured to handle EOLs differently according to the file type, whereas Perforce converts all text files according to a single, per-client setting.

- Tags: indicates if meaningful names can be given to specific revisions, regardless of whether these names are called tags or labels.

- International support: indicates if the software has support for multiple language environments and operating system

- Unicode filename support: indicates if the software has support for interoperations under file systems using different character encodings.

- Supports large repos: Can the system handle repositories of around a gigabyte or larger effectively?

- Web Interface: Describes whether the software application contains a web interface. A web interface could allow the software to post diagnostics data to a website, or could even allow remote control of the software application.

- GUIs: A GUI is a graphical user interface. If a software product features a GUI its functionality can be accessed through application windows as opposed to accessing functionality based upon typing commands at the command prompt such as a DOS interface.

- Plug-ins: functionality is available through an Integrated Development Environment. Minimum functionality should be to list the revision state of a file and check in/check out files.

- command aliases: create custom aliases for specific commands or combination thereof

- lock/unlock: exclusively lock a file to prevent others from editing it

- rollback: remove a patch/revision from history

- cherry-picking: move only some revisions from a branch to another one (instead of merging the branches)

- grep: search repository for lines matching a pattern

- record: include only some changes to a file in a commit and not others

For managing changes to source code over time and keeping track of every modification to the code Git version control tool is chosen over others because of its following advantages:

- **Feature Branch Workflow**: One of the biggest advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge. This facilitates the feature branch workflow popular with many Git users.

- **Distributed Development**: In SVN, each developer gets a working copy that points back to a single central repository. Git, however, is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits. Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits. Distributed development also makes it easier to scale your engineering team. If someone breaks the production branch in SVN, other developers can't check in their changes until it's fixed. With Git, this kind of blocking doesn't exist. Everybody can continue going about their business in their own local repositories. And, similar to feature branches, distributed development creates a more reliable environment. Even if a developer obliterates their own repository, they can simply clone someone else's and start anew.

- **Pull Requests**: Many source code management tools such as Bitbucket enhance core Git functionality with pull requests. A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of changes, but also lets developers initiate discussions around their work before integrating it with the rest of the codebase.

- **Community**: In many circles, Git has come to be the expected version control system for new projects. If your team is using Git, odds are you won't have to train new hires on your workflow, because they'll already be familiar with distributed development.

- **Faster Release Cycle**: The ultimate result of feature branches, distributed development, pull requests, and a stable community is a faster release cycle. These capabilities facilitate an agile workflow where developers are encouraged to share smaller changes more frequently. In turn, changes can get pushed down the deployment pipeline faster than the monolithic releases common with centralized version control systems.