Comparison of Git and Mercurial Distributed Version Control Systems (DVCS)

**1. Background - Git & Mercurial -** Git was designed to be a low level VCS. Characteristics include rapid branching, fast merging and diffs (Ref 12, Ref 9). Git allows bespoke front ends to use its low level VCS commands. Non linear development history can be viewed using Git tools. Each commit in Git is a branch. The branch structure can be constructed with the parental commits of the change (Ref 12). Each developer has a local history for their repository and changes can be copied into it from another repository. Branches can be merged like local branches (Ref 12). Mercurial was designed to be scalable, give high performance, advanced branching and merging and allow collaborative working. Mercurial also comes with a web interface unlike Git to allow sharing of repositories (Ref 13).

**2. Common Features and Differences**

**Installation -** Mercurial andGit can be installed on Linux, Solaris, Mac OS and Windows operating systems using various types of installers. The installation option depends on the operating system selected (Ref 12).

**Help -** Mercurial and Git command line help lists available commands and descriptions for each (Ref 8 and Ref 1).

**User Interface -** Mercurial uses command line or GUI interfaces. The directed acyclic graph Git and Mercurial repository can be viewed (Ref 4). The Git software has git bash interface which uses UNIX like command line operation (Ref 4). Mercurial has support for IDEs plug-ins for Eclipse (Mercurial Eclipse), Netbeans and Visual HG and HgSccPackage (Visual Studio). Mercurial also has context menu in Windows Explorer. Git has IDE plug-in's for Eclipse (Egit) and Netbeans IDE (Ref 12). Git has 130 commands and advises users which to run based on the current situation encountered (intuitive). Mercurial has less commands and a simpler interface and makes for easier learning than Git (Ref 4).

**Importing Repositories from other version control systems -** Mercurial also has support for importing from various VCS including Git repositories (Ref 13). Importing to Git from other VCS repositories such as subversion is troublesome i.e. author information (requires transformation) but history can be imported with no problems. Git has a fast-import tool and online tools to convert non git repositories (Ref 2).

**Portability of GIT and Mercurial for Development -** Git libraries exist for use by popular languages such as Java (JGit), Python (Dulwich) and ANSI C (libgit2).They can be embedded into applications (Ref 12). Mercurial has an evolving internal Mercurial API. Mercurial recommend not developing against this but developing against the API for their command line interface which is stable. One such API is the hgapi Python API. (Ref 10).

**Source Code Hosting -** Git and Mercurial repositories can both be hosted on BerliOS, Bitbucket, Code Plex, GNU Savannah, Google Code, JavaForge and SourceForge. Mercurial repositories can also be hosted on Alioth, Assembla, Kiln, Project Kenai and git repositories can also on Cloud forge, Git hub and Gitorious. (Ref 12 and Ref 3).

**Making a local repository -** Mercurial and Git repository contains all the files of a project chosen to be stored in the repository as well as the history file. Both DVCS have the ability to clone over the network remembering its source repository it was cloned from to allow Pull/Fetch of data/files (Ref 8 and Ref 1).A repository in Git or Mercurial consists of the actual \git. or \.hg subfolder which stores the history information. Files and directories that exist with theses folders form the working directory. The working directory contains a snapshot view of the project in a point of time (Ref 8, Ref 1).

**Tracking files -** Add and status commands are used to add and track files in Mercurial and Git. (Ref 2). Files can also be untracked. In Mercurial paths to files are tracked and directories are not. The remove command can be used to untrack files. When a file is removed the file history is not affected. Users can revert the deletion using revert command. Mercurial does not deal with files that are manually deleted and therefore missing(Ref 8). The track states in Git are unmodified, modified and staged. Untracked files are those that are not in the staging area or in the last snapshot taken of the working directory. Cloning a repo for the first time the files in the clone are unmodified and tracked. Removing files in Git is similar to Mercurial but unlike Mercurial a remove command is used to remove the file from tracking (staged area) and commits change (so file is not shown as untracked) (Ref 2).

**Viewing and referring to changes -** Mercurial - Changes are recorded in changesets which can be viewed using log command which gives a summary. Changes made are listed as changesets (referring to 1 or more files) including user(username and email address), date (date and time) and comment added by user when the change was made. The changesets are referred to by their unique hexa-decimal number identifier that is common between clones of a repository. Revisions can be viewed singly by the revision number or hexa-decimal identifier or batch(Ref 8).Similarly Git changes can be viewed by viewing the log file. A unique 40 digit hexadecimal id (unique to any copy of a repository) , author (username and email address), date/time and commit message is saved with each commit and can be viewed. Both DVCS allow users to refer to commits as well as to transitive commits i.e. via specified branch names (Ref 1).

**Making Changes and Reviewing these -** In Mercurial multiple clones of repositories can be made locally to allow users to do different tasks in parallel if required. There is not much overhead and it is not costly. Mercurial determines automatically what has been modified and what hasn't without user involvement (Ref 8).Changes can be committed to the repository, and stored in the changeset via commit command which creates a new changeset. The username is mandatory and is usually the local username (host name) and email address. (Ref 8).Like Mercurial Git has a status command to view the status and any changes to files in the repository. This includes the current branch that the change(s) that the files relate to. The changes will be detected and listed and the user is automatically told which Git commands (add and commit commands) to run to add the file(s) to the repository and to commit the changes (intuitive) similar to Mercurial (Ref 1). Branches in Git are a pointer to a commit (fast commit). Changes in files are shown in the git status as "modified" instead of "M" that is used in Mercurial (Ref 1). Git allows actual changes between files to be viewed via the diff command (Ref 1). Like Mercurial, Git's equivalent review commands status and diff is used until users are happy with changes.

**Committing Changes. -** In Mercurial the commit changes allows users to add a message and commit the change to the changeset. Changes can be viewed and reviewed using status and log commands. Mercurial allows commits to be aborted by allowing users to exit the commit without saving changes. The last change can be viewed using tip command (Ref 8). Git and Mercurial commit assigns author information, date, time, message and branch information (Ref 1). In both DVCS files to be tracked are added using add before they are committed. Batch tracking and committing is available in both (Ref 1).

**Sharing Changes -** In Mercurial changes can be shared between users (pulled) in a repository using >hg incoming command to tell the user the changes that the >hg pull command would make to the target repository. NB. The >hg incoming command does not pull the actual changes to the target repository and gives the user a level of control to see changes that would occur if the data was pulled without actually doing the pull from the source repository. Users can specify to pull a specific changeset by reference to its hexa-decimal value in the hg pull command (Ref 8). Similar functionality exits in Git to pull snapshots and review before changes.

**Updating the repository -** In Mercurial changes that have been pulled are stored initially in the repository sub folder of the working directory and do not initially show up in the working directory. For changes to be stored in the working directory the update command has to be executed. This allows users to go to a revision of the data including previous revisions when running the update and not having conflicts with any pulled changes. (Ref 8).

**Pulling/Fetching Changes** Mercurial allows all heads or single branch via revision id, to be pulled (fetched) using >hg pull command and relevant switches. In Git pull is a fetch. By default cloned repositories in Git get all the branches from the repository being cloned, saved as remote tracking branches in Git. A single branch can be fetched and is stored similar to Mercurial like an unnamed head (anonymous) (Ref 6). In Git after every pull there is no need to issue an update command like in Mercurial (Ref 7).

**Pushing changes to a repository -** In Mercurial pushing to a repository is similar to pulling. There are commands in Mercurial to check to see what changes will be pushed to the target repository before changes are pushed. If there are no changes to push then there is a message to state this. (Ref 8). All heads are pushed by Mercurial by default and a specific head can be specified to push instead by specifying the tip revision using the revision number (local repository, revision id, or named branch (embedded branch name). Commits are saved into the cloned repository with the same named branch as a name. In Git pushes to repositories will happen if the destination repository contains the same local branches. Matching (for more than one local branch), all, or single branch are the types of branch that can be involved in pushes in Git (Ref 6).

**Repositories shared over a network -** In Git and Mercurial pushing/pulling repositories can be done over a network and in the cloud(Ref 8 and Ref 13) (see source code hosting section above). In Git and Mercurial available protocols include HTTP, rsync , FTP, ssh or Git protocols to publish and share repositories and Git has the ability access Subversion repositories. IDE plug-ins are supported and can be used with Git because CVS Server emulation exists in Git. (Ref 4).

**Creating a new project -** Both Mercurial and Git allow users to create projects with new repositories instead of just cloning existing repositories (Ref 2 and Ref 8).

**Merging Work -** Mercurial allows the merging of work from 2 or more repositories or merging different changes made to different repositories. The revision history in target repository stays the same and the revision that is pulled for merging from the source repository keeps its changeset id but its revision number changes. It is added to the revision history of the destination repository as to create a divergent history. The Mercurial DVS stores the parent of each change with the change being seen as the child. In Mercurial a new change does not have a parent and is called the head. The divergent history described above has two heads one for the most recent change in the target repository and one for the recent change in the source repository that was chosen to merge with the source repository. If one or more heads exist in a repository there are commands to allow users to view and then merge the changes from the resulting 2 heads in the repository. The contents of the affected source file in the working directory is changed to show the merge and in the 2 heads. The merge is not saved until committed. The 2 heads are merged into a new tip revision and is given the 2 previous heads as its 2 parents (Ref 8). Merging and reviewing changes using Diff in Git is very quick compared to related commands in Mercurial (Ref 7). The merge strategies in Git are part of the Git toolkit (Ref 12).

**Managing conflicting changes when merging -** Compared to Git Mercurial does not itself have any software itself for handling conflicting changes i.e. where there are 2 changes , one each in identical places in the versions to merge (Ref 8). Similarly Git has been designed with multiple algorithms available that allows users to manually edit incomplete merges or to automatically merge if it can do so (Ref 12).

**Extending Functionality -** Mercurial has new commands, processes or plug-ins that run on server mode. (Ref 8). Git on the other hand caters for scripting (Ref 3).

**Copying Files -** Mercurial propagates code changes from source files up to copies of these files when copies of files are to merge. Git does not do this (Ref 8).

**Renaming Files -** Git uses rename detection and Mercurial uses rename tracking(Ref 4 and Ref 6).

**History Model in Git and Mercurial -** The history model in Git is built from blob, tree and commit whereas Mercurial uses file, manifest and changeset. A changeset is identified in Mercurial using a SHA1 hash code which is similar to how changes are identified in Git. However the difference is that a revision number - incremented integer is added to identify revisions in the local repository which makes communicating between developers easier instead of referring to SHA1 number. The history model in both Git and Mercurial is stored in a Directed Acyclic Graph (DAG).(Ref 11). Git stores its history data like a hash tree and contains extra data at the node and leaves levels. The Id for a change depends on the history of changes up to the commit. Git saves history as a snapshot and not a changeset (Ref 12, Ref 1).

**Branch Model In Git and Mercurial -** In Git and Mercurial both support different branches in the same repository.Branches containing thousands of files takes seconds in git when switching and is fast compared to Mercurial which has more overhead **(**Ref 4 and Ref 7)**.** In Git a branch is a clone of a repository and can be identical to other repositories that are also a clone of this repository. Developers can switch between branches quickly and easily as the branches are lightweight i.e. references are stored to branches via the last commits (as pointer to commit) of each branch. New commits are added to the relevant branch and is given a new reference (moving reference) (Ref 11). Git also has branches that remotely track branches in the remote repository (Ref 6). Mercurial branches are called HEADS and changeset identifiers are used to refer to these heads (via SHA1 hash code or local repository integer revision number). Mercurial also has named branches which allow users to specify a working directory for the branch (repository) (Ref 8 & Ref 11) The named branch or default is embedded in the commit, This concept of embedded branches is not in Git (Ref 6). All changesets that are committed will automatically be assigned to the same named branch. New branches can also be created for older revisions for example the update command can specify a specific revision number and the branch command can be run to create a new branch for this revision. Branches may be diverged as a new line of development at a diverging point or in the middle of a line of development (sequence of changesets) is taken. If a new branch name is specified when there is a diverging line of development the new line of development will take on the parent changesets branch name. (Ref 8 and Ref 11).Users can switch between branches and specifying which named branch to switch to. This becomes the tip of the branch. Tip can be found where multiple HEADs exist. Mercurial branches is based on anonymous heads compared to Gits tracked branches. Git's commits may not belong to the branch that the commit has been made to. Mercurial documentation advises long lived branches which is branching by cloning (Ref 6). Clashing branch name is possible from same named branches in 2 different repositories with different contents because of global namespace in Mercurial but not Git (Ref 6).

**Behaviour of the repository -** Both DVCS have similar branching models but the movement of history is different in both. In Mercurial undo changes is difficult compared to GIT. Git has a tracking branch which allows selective push or pull of branches to or from repositories allowing changes to be followed from a repository (Ref 11). Mercurial branching is considered simpler because when all heads are pulled in the remote repository and brought into the target repository users can interactively or automatically merge heads (Ref 11). When pushing changes users are warned if two heads look to be created (Ref 11).

**Staging -** Git has a staging area / index which is exposed to the user for them to interact with (Ref 2). Mercurial has no staging area but has DirState so system knows files to save for the next commit (Ref 11).

**Common Differences Table1 - Functionality and Features in Git & Mercurial (Ref 4).**

|  |  |  |
| --- | --- | --- |
|  | **Git Software** | **Mercurial Software** |
| **Atomic Commits** | **Yes** | **Yes** |
| **Permission Keeping** | **Execution bit only** | **Execution bit only** |
| **History Model** | **Snapshot** | **Changeset** |
| **Written In** | **C, Bourne Shell, Perl** | **Python and C** |
| **Repository Size** | **(O) patch, Big O notation** | **(O) patch, Big O notation** |
| **Concurrency Model** | **Merge** | **Merge** |
| **Latest Release (Updated)** | **1.8.0 (21/10/2012) Stable** | **2.3.2** |
| **Operating System** | **Mac OS X, Windows, Linux, Solaris** | **Unix like, Windows, Mac OS X, Solaris** |
| **Bug Tracker Integration** | **No** | **Trac (via plug-in)** |
| **Staging Area** | **Yes** | **No** |
| **Licence** | **GPL v2** | **GPL v2** |
| **Cost** | **Free** | **Free** |
| **File/Directory Name tracking** | **Rename Detection** | **Rename tracking** |
| **Repository Model** | **Distributed** | **Distributed** |
| **Network Protocols** | **Custom, custom over ssh, rsync, HTTP, email bundles** | **HTTP, custom over ssh, email bundles (with standard plug-in)** |
| **Built in Web Server** | **No** | **Yes** |
| **Merge Tracking** | **Yes** | **Yes** |
| **Pre/Post Event hooks/triggers** | **Yes** | **Yes** |
| **End of Line Conversions** | **Yes** | **Yes** |
| **Tags** | **Yes** | **Yes** |
| **File Renames** | **Yes (implicit)** | **Yes** |
| **Merge File Renames** | **Yes** | **Yes** |
| **Symbolic Links** | **Yes** | **Yes** |
| **External: Branch** | **Yes** | **No** |
| **Open Source** | **Yes** | **Yes** |
| **RCS Keyword** | **Yes (not recommended)** | **Via bundled plug-in** |
| **Signed Revisions** | **Yes** | **Yes** |
| **Revision IDs** | **SHA-1 hashes** | **Numbers, SHA-1 hashes** |
| **Shallow checkout/clone** | **Yes** | **Bugzilla extension** |
| **Sub Directory Checkout/Clone** | **No** | **No** |

**References**

1. Blewitt, A. (2011). *mercurial and git technical comparison.* Available: http://alblue.bandlem.com/2011/03/mercurial-and-git-technical-comparison.html. Last accessed 6th Nov 2012.

2. Chacon, S. (2009). *Pro Git (Online version of Book).* Available: http://git-scm.com/book. Last accessed 6th Nov 2012.

3. Contreras, F. (2011). *mercurial-vs-git-its-all-in-the-branches.* Available: http://felipec.wordpress.com/2011/01/16/mercurial-vs-git-its-all-in-the-branches/feed/. Last accessed 6th Nov 2012.

4. Diffen. (2012). *Git (software) vs Mercurial (software).* Available: http://www.diffen.com/difference/Git\_(software)\_vs\_Mercurial\_(software). Last accessed 6th Nov 2012.

5. Losh, S. (2010). *the-real-difference-between-mercurial-and-git.*Available: http://stevelosh.com/blog/2010/01/the-real-difference-between-mercurial-and-git. Last accessed 6th Nov 2012.

6. Narebski, J. (2009). *git and mercurial compare and contrast.* Available: http://stackoverflow.com/questions/1598759/git-and-mercurial-compare-and-contrast. Last accessed 6th Nov 2012.

7. O'Sullivan.B ,Worth. C. (2007). *A tour of git: the basics.* Available: http://cworth.org/hgbook-git/tour/. Last accessed 6th Nov 2012.

8. O'Sullivan, B. (2009). *Mercurial: The Definitive Guide.* Available: http://hgbook.red-bean.com/read/. Last accessed 6th Nov 2012.

9. Soria Parra,D. (2007). *GIT VS. MERCURIAL.* Available: http://blog.experimentalworks.net/2007/08/git-vs-mercurial/. Last accessed 6th Nov 2012.

10. Various et al, http://mercurial.selenic.com/wiki/MercurialApi?action=info. (2012). *The Mercurial API.* Available: http://mercurial.selenic.com/wiki/MercurialApi. Last accessed 6th Nov 2012.

11. Various et al, http://mercurial.selenic.com/wiki/GitConcepts?action=info. (2012). *Mercurial for Git users.* Available: http://mercurial.selenic.com/wiki/GitConcepts. Last accessed 6th Nov 2012.

12. Wikipedia. (2012). *Git (software).* Available: http://en.wikipedia.org/wiki/Git\_(software). Last accessed 6th Nov 2012.

13. Wikipedia. (2012). *Mercurial.* Available: http://en.wikipedia.org/wiki/Mercurial. Last accessed 6th Nov 2012.