Chapter 2

An Overview of Snapstore

Snapstore was created with two distinct groups of users in mind, non-technical users and technical users. In order to be an attractive system to technical users who use systems like Git, Snapstore needed to support the functionality of powerful version control systems. However, Snapstore also needed to have a smoother learning curve to promote quick startup and attract users who feel overwhelmed by complicated VCSs.

We decided to create Snapstore with an opt-in strategy concerning complexity. Users can download Snapstore and get started right away with simple actions like file storage. Then, if desired, users can explore more advanced features of the system, without fear that they are necessary for their current project needs.

Snapstore uses a simple user-based identification system. Once a user downloads Snapstore, they create an account with a username.

Snapstore operates within a specially designated Snapstore folder which is created upon opening the application for the first time. It is similar to the Dropbox folder; Snapstore only looks at files that are inside of it. Snapstore watches the user's filesystem for changes in order to trigger certain functionality. This allows users to use any editor with Snapstore. The Snapstore folder that Snapstore uses does not need to remain the default folder. Snapstore can be opened using any folder as the Snapstore folder.

2.1 Basic Features

The basic features of Snapstore allow a single user to use the application like they would a file syncing system.

2.1.1 Snapshots

Snapshots allow a user to persistently store all of their edits to a file over time. Whenever a change is made to a file, a snapshot is created with the contents of that file and stored in the local repository. Users can recover this snapshot data by finding this snapshot in the file's history.

Snapshots are also created when a file is created, deleted or renamed. This allows snapshots to capture the true history of a file. Snapshots can be one of six types: create, update, rename, delete, merge, or conflict. These snapshot types correspond to all potential changes made to file.

When creating snapshots for a given file, Snapstore will add the snapshot to that file's *snapshot graph*. This graph represents a history of the file and it shows each snapshot that was taken, along with its relationship to other snapshots of that file. A snapshot has one or more parents (the snapshot(s) taken before it), and it has a child (the snapshot taken after it). The first snapshot in a graph is called the *root*, and the last snapshot in the graph is called the *head*.

To inspect a snapshot from a file's history, navigate to that file within Snapstore's interface. Then, click on the file's name. The snapshot graph will appear at the top of the window. Each node in this graph is a snapshot, and the node on the far right is the head. By clicking on these nodes, the content of the snapshot will appear on the bottom-right of the window. After the correct snapshot is found, it's possible to revert to that snapshot by clicking "Revert", a button above the snapshot content. Reverting to a previous snapshot will create a new snapshot with the same content at the head position and alter the file's content to match the snapshot's content.

2.1.2 Upstreams

Upstreams are used to synchronize edits made by collaborators on a project. They are remote repositories that Snapstore users can connect to when on a network. Whenever a user creates a snapshot, branch, group, or tag, it is sent to the upstream and out to all other users who are collaborating what that user.

If two users are working together on a project, the upstream will synchronize their snapshots as they make them. In the case of multiple snapshots coming in to the upstream at the same time, the upstream will resolve the conflict and push the same ordering of snapshots to both users.

By default, the upstream is the Snapstore server, but users might want to change their upstream so their data passes through a known location. To do so, follow these steps:

1. Download the Snapstore server program onto the machine
2. Run the Snapstore server on that machine
3. Point the Snapstore client to that machine

2.1.3 Local Repository

The *local repository* stores all of a users data offline, on their own computer. Whenever a user makes a snapshot, branch, group, or tag, that data is saved in the local repository.

The local repository allows users to work in a disconnected setting. While offline, Snapstore will populate the local repository based on user actions. When connection is restored, Snapstore will push all new changes to the upstream.

A single user using more than one computer makes use of upstreams and local repositories. In this case, they are collaborating with themselves, on different computers with different local repositories. The user can work on computer A, saving changes to their local repository. When they open Snapstore on computer B, computer A's local repository will push those changes to computer B through the upstream.

2.2 Advanced Features

Snapstore's advanced features allow users to access the more powerful components of a version control system. None of the advanced features are needed to use the system. They provide additional functionality that users might want when working on their projects.

2.2.1 Groups

Groups allow users to group related snapshots. When a user has a collection of snapshots they believe are related, even if they exist across multiple files, they can place them in a group. The user can then give this group a name.

A software team collaborating on a project might want to fix a syntax bug in their program. If one user makes a few snapshots in this process, they can then place them in a group and title it "Fixed syntax bug". Now, these snapshots and this group have all been shared with the other team members through the upstream. Team members can easily locate this group and inspect its snapshots to see how the bug was fixed.

2.2.2 Tags

Tags allow users to specially designate a group as a coherent point in development. When a group of snapshots is particularly significant, such as project completion or a release of a piece of software, users can tag that group.

Tags can only be given to groups who have at most once snapshot per file. This is so that tags can be used for mass file reversion.

For a software release, a user can tag the group of head snapshots "Version 1.3" to signify that the project is in a stable release state. Later, after more snapshots have been created, users can utilize this tag to revert the project to the group of snapshots tagged by "Version 1.3".

2.2.3 Branches

Branches allow users to differentiate between parallel lines of work. Whenever any snapshot, group, or tag is created, it is saved within the user's current branch. Switching to a different branch will load all the files, snapshots, groups, and tags associated with that branch. Branches can be merged together to combine parallal lines of work.

Snapstore provides a default branch called "master" on startup. A user can then create and use branches to maintain multiple versions/releases of the same project, keep the development of major features isolated, and to give users the ability to try out experimental changes without affecting the main line [2].

A branch can also be created by cloning an existing branch. Creating a clone involves choosing a branch to clone, and then selecting the snapshots inside the original branch to bring over to the clone. For all snapshots that are cloned into the new branch, any groups associated those snapshots, and any tags associated with those groups, will be copied to the cloned branch.

Collaborating on Branches

When a branch is shared with another user, that branch's data is sent to that user and it becomes a shared branch. On a shared branch, when any user makes changes, those changes are immediately sent to the other user(s) associated with that shared branch. A user might create a branch to hide certain files from others users. They could keep private files in their master branch and instead create a new branch called "public" that is shared with other people.

When conflicts arise due to multiple users working on the same branch at the same time, Snapstore uses a last-write wins methodology. The last snapshot to reach the server will become the head snapshot for the file. However, no snapshots are lost in the conflict, and the user can revert to a passed over snapshot easily.

An important use case is the one mentioned in the introduction of this paper. If a small software team in school needs to be able to share code and develop in parallel, they can work on a shared branched in Snapstore. Snapstore is especially attractive if they do not have the time or desire to learn a system as complex as Git. In either case, Snapstore allows them to share work and maintain versioning

Merging Branches

Merging two branches compares the snapshot graphs in those two branches. If two snapshot graphs correspond to the same file, then a merge is performed using three way merge and their common ancestor. This will result in a merge snapshot with as many parents as there are snapshots being merged. If there is a merge conflict, then the resulting snapshot will be a conflict snapshot. Like in Git, a conflict snapshot's content will show where the conflict needs to be resolved. Unlike in Git, this merge snapshot is already on the server and saved, no conflict resolution is needed to continue working. By simply fixing the conflict and saving, a new snapshot is created that reflects the fix. Merging two branches will keep all of the group and tag data from both branches.

Snapstore offers many project management solutions using branches. For example, imagine that a web application project has a project manager, a front-end developer, and a graphic artist. The project manager can create two clones of his master branch; one has all the code, and one has all the images of the project. She can then share these cloned branches with the developer and the artist. This shields each worker from the other's work. The project manager can keep an eye on each branch's progress because she has access to both cloned branches; each snapshot one of her employees makes shows up on her machine. The project manager can later merge both branches back onto the main branch and have a completed project.

Snapstore branches can also be used for projects with different workflows. The Linux project uses trusted lieutenants to review patches in sections of the Linux kernel before sending them on to the project owner. Many open source developers send their code directly to these lieutenants for review. Snapstore can achieve a similar hierarchy by having the project owner clone certain aspects of the full project and share those clones with the lieutenants. Then, lieutenants can share these clones with the world and vet incoming snapshots.