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.

2.1 Basic Features

This basic system doesn't need many of the features offered by Snapstore, and so they are hidden away. The basic features allow a single user to use the application for file storage and backup.

Snapstore uses a simple user-based identification system. Once a user downloads Snapstore, they create an account with a username. With this, they can immediately begin to use Snapstore to whatever capacity they wish.

Snapstore operates within a specially designated Snapstore folder which is cre-

ated upon opening the application for the first time. It operates very similarly to the Dropbox folder; Snapstore only looks at files that are inside of it and nothing else. 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, allowing users to have more than one Snapstore folder.

2.1.1 Snapshots

Once the user is logged in and starts making edits to files in their Snapstore folder, Snapstore will begin saving their data. By default, any edits to a file made in the Snapstore folder will result in the creation of a *snapshot*. The snapshot is essentially a copy of a file at a specific moment in time. Snapshots are also created when a file is created, deleted or renamed. This allows users to see the true history of a file, even if it has been moved, renamed, deleted, or reverted.

Snapshots can be one of four types: create, update, rename, or delete. The different types of snapshots will show up as different colors in the snapshot graph so that the user can see the difference.

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 (i.e. the current snapshot) is called the *head*.

2.1.2 Upstreams

When connected to a network, all changes and edits are sent to an *upstream* automatically. An upstream is a remote repository that allows Snapstore data to be stored persistently, in case of local system crashes. The default upstream is Snapstore's

server, and it is available right away, when the application is opened for the first time.

Snapstore, at its most basic, can be used for personal backup. Once Snapstore is downloaded and connected to a network, all snapshots are sent to an upstream for persistent storage. Any user can use this setup for personal backup. They can even decide where to send these snapshots (described below in section 2.2.4); perhaps a machine in their home is all they need.

One benefit of the upstream model that any Git user can sympathize with is how it handles the case of a single user using multiple machines. A Git user needs to pull and push from every machine they use in order to get work they have commited from other machines. Snapstore eliminates that by having all snapshots automatically saved on whatever server the user wants. Then, when they log in to another machine, all changes are automtically pulled in from the server and reflected on the user's machine.

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. Rather, they provide additional functionality that users might want when working on their projects.

2.2.1 Groups

Snapshots can also be grouped for projects and files where this is necessary. Groups of snapshots do not necessarily need to contain all snapshots from the same file. For example, a user might want to group the head snapshots for every file in a branch. Or, they might collect multiple snapshots from the same snapshot graph where they fixed a bug.

2.2.2 Tags

Groups of snapshots can be given tags. These tags provide a label for a group that represents a coherent development point. For example, the group of head snapshots for a branch might be tagged "Version 1.3" to signify that the project is in a stable release state. Only groups that have at most one snapshot per file (vertical groups) can be tagged. Tags help administrative duties such as setting deadlines, labeling work, and marking milestones.

2.2.3 Branches

Snapstore uses the *branch* to differentiate separate lines of development. Branches hold files (represented by their snapshot graphs) and their group and tag data. Branches can be shared between users so that they can collaborate on files within them. When Snapstore is opened for the first time, a master branch is created. No new branches ever need to be created to use Snapstore.

New branches can be created for multiple reasons. One reason might be to simplify the Snapstore folder for an individual's use. Keeping every single file in the master branch would become excessive, and new branches can clean up a user's workspace. Another reason might be to hide certain files from users you would like to share a branch with. A user might keep private files in their master branch and instead create a new branch called "public" that is shared with other people. A user can also 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].

Cloned branches can also be created from branches. Creating a clone involves choosing a branch to clone, and then selecting the files inside the original branch to bring over to the clone. This allows the new branch to have common snapshot ancestors with the original branch, for the purpose of later merging. Cloned branches will inherit all of the related tags and groups from its parent branch. This means that for all snapshots that are cloned into the new branch, any groups associated those

snapshots, and any tags associated with those groups, will be in 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. When the new user loads that branch, it changes their file system within their Snapstore folder to match the branch's head snapshots. On a shared branch, when any user makes changes, those changes are immediately sent to the other user(s) associated with that shared branch, offering immediate collaboration.

When conflicts arise due to multiple users working on the same branch at the same time, they are dealt with using the DESQ algorithm (described in section 4.4). This algorithm uses a last-write wins methodology, so 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.

Snapstore is designed with every industry in mind, and these features also benefit the non-technical user. A legal team working on an array documents might use Snapstore to maintain versions of those documents and to manage parallel work on the same document. Team leaders can easily keep track of progress, and those working on the documents can see the snapshots from other team members immediately.

Merging two branches is possible with Snapstore and facilitates parallel development in teams. Merging two branches compares the snapshot graphs in those two branches. If two snapshot graphs correspond to the same file, they are are merged together. Other files are simply added to the merged branch.

Files that have a common ancestor will merge their head snapshots. This will result in a snapshot with as many parents as there are snapshots being merged. This

new, "merge" snapshot will merge the file contents from the head snapshots to produce a new snapshot. If there is a merge conflict, that will be reflected in the snapshot and file. Much like a merge conflict in Git, the file will show where the conflict needs to be resolved. Unlike Git, however, 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. If a snapshot that ends up in the merged branch is a member of a group, then the group will end up in the merged branch. Similarly, if a group ends up in the merged branch that had a tag, that tag will end up in the merged branch.

With these more advanced collaboration tools, Snapstore offers many project management solutions. By adding branches and clones of branches, a project manager can shield certain parts of the project from certain workers. 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 from the code folder and the images folder 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. Then, the project manager can 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, Linus. Many open source developers send their code directly to these lieutenants for review. This hierarchy allows a huge project like Linux to function efficiently. 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, they can share these clones with the world and vet incoming snapshots.

2.2.4 Local Repository

All changes made on a user's local computer are saved in the *local repository*. A user can have multiple local repositories on one or more machines to shield certain edits from other repositories. The local repository allows users to work in a disconnected setting. While offline, Snapstore will still watch the filesystem and save changes. When connection is restored, Snapstore will push all new changes to the server.

The location of which upstream repository a local repository is connected to can be changed according to a user's needs. A user can designate another machine as their backup by following a few easy 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

This allows Snapstore to be a viable use case for sensitive information and even for data backup in the home.