GitUp: A Simple, Portable Backup Manager with Practical Version Control

Problem:

Backing up a project is essential to its reliable completion. This process ranges anywhere from copying files to an external drive to using an automated service to store files online. Products such as Google Drive, Microsoft OneDrive, Adobe Creative Cloud and Dropbox offer automated backup services to protect projects from being lost. Some also offer version tracking that allows users to restore intermediary changes to files. Cloud-backup products are more reliable, secure, and easy to use than traditional external drive backups, but they have their fair share of disadvantages. Their backup speed and frequency is limited by internet speed because project data must be overwritten on the cloud server when files are changed. This, along with space constraints on cloud servers, can limit project size. Cloud backup products are also woefully inadequate at keeping track of and comparing previous versions of a project. Previous versions only exist for a given time frame and changes are not grouped meaningfully.

Distributed version control systems such as Git, Mercurial and Subversion solve some of the issues cloud backup systems face by tracking changes between multiple repositories. Version control systems provide a far more robust system for tracking changes in a project, but with that added functionality comes a whole host of new problems. Version control systems that allow multiple users to collaborate have added complexity that is unnecessary when used by a single user as a backup solution. Additionally, granular change tracking in version control systems is a difficult to learn, manual process. New users frequently find themselves in “edge cases”, as a result of ignorant or improper usage of version control software.

Intermediary to cloud-backup and version control systems are version control wrappers such as GitKraken or IDE extensions. Wrappers provide a GUI but do little to simplify or automate interaction between users and project files for backup purposes. The portability and flexibility of a wrapper is also limited by the IDE or system it uses. For example, a user might write code in Eclipse and documentation in Word for a single project but these programs do not share version control wrapper extensions.

Proposed Solution:

GitUp is a product with the goal of building on reliable, automatic, and secure backup features by providing users with fast backup speed, granular version tracking, efficient use of storage space, and portability. GitUp will achieve this goal by extracting relevant features from the Git version control system and abstracting their usage for the purpose of backing up individual projects created by single users.

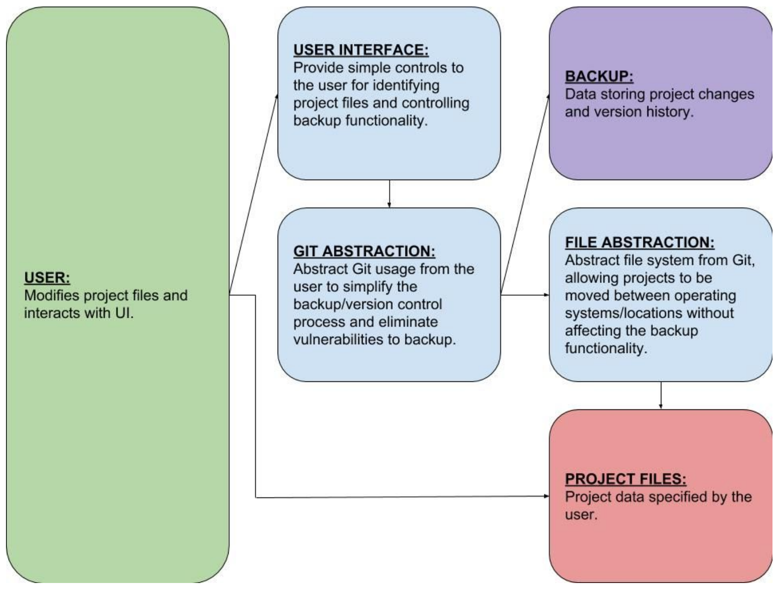
GitUp will be designed with a variety of single user project types of users in mind, from less experienced developers, to graphic designers, to authors. Its simplicity will ensure that it remains accessible to a wide range of users. To use GitUp, a user will copy a self-contained GitUp file to the root directory of their project, where they specify project metadata (name, description etc.) and a backup location (local or online), which can be changed without affecting GitUp's services. GitUp will automatically track project file versions and update the backup whenever it detects a change. It will maintain a compact backup repository and provide simple version restoration with granularity ranging from the entire project to individual file changes.

No other system has been created with the idea of single user projects in mind. Existing products are sufficient, but not optimal, for this use case. GitUp will be created to be the optimal backup solution for single-user projects, with a focus on being simple and intuitive to use.

Product Goals:

1. Automatically create and maintain a project backup (create repository, local → centralized, online → distributed)
2. Automatically group project changes in a meaningful way (automatic add → commit → push)
3. Intuitive to users with little to no experience with other version control systems (hide unnecessary Git features, determined from user survey)
    a. Difficult to end up in git "edge cases".
4. Provide users with a straightforward way to view past versions of a project (down to the file level) and revert. (automatic pull/checkout/reset)

Architecture:


USER:
Modifies project files and interacts with UI.
USER INTERFACE:
Provide simple controls to the user for identifying project files and controlling backup functionality.
GIT ABSTRACTION:
Abstract Git usage from the user to simplify the backup/version control process and eliminate vulnerabilities to backup.
BACKUP:
Data storing project changes and version history.
FILE ABSTRACTION:
Abstract file system from Git, allowing projects to be moved between operating systems/locations without affecting the backup functionality.
PROJECT FILES:
Project data specified by the user.

GitUp's architecture will include a text-based user interface, a Git abstraction layer, a file abstraction layer and a backup repository, as shown in the diagram above. Through its implementation, GitUp will differentiate itself from other backup systems by focusing on single-user projects and avoiding many of its alternatives' weaknesses. GitUp will be faster than cloud-backup systems with more advanced version tracking. It will be easier to use than version control systems, less vulnerable to user error, simpler, and more automated/portable than version control wrappers.

<u>Challenges and Risks:</u>

Design Challenges:

One fundamental risk that will need to be constantly addressed during the development of GitUp is that its design does not satisfy the problem it aims to solve, i.e. GitUp does not provide single users with almost all the functionality they need and/or it is bloated with too much functionality and difficult/unintuitive to use. To mitigate this risk, the GitUp team will conduct comprehensive user research at all stages of the development process. When collecting information, we will focus on determining the most useful version control features needed in a backup system, user desires with respect to backup frequency, size and speed, and how we can ensure that user interaction with the system is simple and automated.

Technical Challenges:

One technical challenge that needs to be addressed is actually integrating our UI with Git so that running UI commands will lead to the necessary Git functionality being utilized. To mitigate this problem, we will devote time in the first two weeks to studying existing open source Git wrappers to understand how they integrate with their UI to ensure that we are able to integrate Git into GitUp. Another technical challenge is if one machine has uncommitted changes, but you want to make edits to that same file on a different machine now. At this point, the first machine has a “lock” on the file, since it shouldn’t be allowed to be edited on two machines simultaneously (leads to merge conflicts). Implementing locking on files could end up being technically challenging since machines will need to know when another machine has a lock on a file. A third technical challenge is grouping automatic commits together in meaningful ways that users will able to use. To that end, after identifying a paradigm by which we group commits, we will try to identify the most recent commit in the group of commits, and only allow the user to see that version when viewing past versions.

<u>Implementation Schedule:</u>

Date	Goal(s)	Test(s)
02/04	Finish preliminary user usage data and finish designing a preliminary UX/UI	Be able to run example use cases through UX flowchart. Results of survey collected.
02/11	Automated pushing/pulling	Be able to complete the basic project use case [1]
02/18	Edge case avoidance/File reverting	Be able to complete both the parallel edits use case [2] and reverting files test [3]
02/25	Begin work on smarter grouping of	Collect detailed customer data on what

	changes	users value most for a ‘smart grouping’ of changes, and finish writing a smart grouping use case/test based off the results of our data collection.
03/04	Smarter grouping of changes	Be able to complete the smart grouping use case [4] and final use case [5]
03/11	Begin and finish collecting feedback on finished Gitup, Begin feedback response.	Feedback fully collected.
03/18	Feedback Implementation	Dependent on feedback received. Fixes implemented address most important user concerns with GitUp

<u>Testing & Evaluation:</u>
These use cases are intended as examples of specific behavior that should work in order for the relevant phase of the project to be considered finished. These are not the exact tests that will be written as part of the test suite, but are meant as high level behavioral requirements. It is assumed that test suites for each phase will be written in order for the phase to be considered completed.

[1] Basic project use case: This test is intended to ensure that the basic functionality of automatic pushing/pulling works. In order for this part of the project to be considered successful a user should be able to:

- Create a project “hello world” on machine 1.
- Add file helloworld.txt to project, and make changes on machine 1, wait for automatic commiting of changes before proceeding.
- Pick up project on machine 2.
- Make changes to helloworld.txt on machine 2.
- Check that changes become visible on machine 1.

[2] Parallel edits use case:

- Create a project “parallel test” on machine 1.
- Add file “paralleledit.txt” to project, and make changes on machine 1
- Before changes from machine 1 are committed automatically, machine 2 will attempt to edit “paralleledit.txt”
- Machine 2 is prompted whether or not to override the lock from machine 1
    1. Machine 2 decides to override the lock from machine 1:
        a. Machine 2 makes changes to “paralleledit.txt”
        b. Machine 2 commits changes to “paralleledit.txt”
        c. Check that “paralleledit.txt” does NOT contain the initial changes from machine 1.

    d. Machine 1 pulls.
    e. Check that machine 1’s copy of “paralleledit.txt” is identical to machine 2’s copy of “paralleledit.txt”
  2. Machine 2 decides not to override the lock from machine 1:
    a. Check that no edits can be made in “paralleledit.txt” on machine 2.

[3] Reverting files use case:

- Create a project “Test revert” on machine 1
  - Create a file “version-test.txt” in the project with the line “version 1”.
  - Commit/push all changes on machine 1.
- Open “Test revert” on machine 2
  - Add the line “version 2” to version-test.txt
  - Create the file “extra-file.txt” with the line “Extra unrelated file”
  - Commit/push all changes on machine 2.
- Open “Test revert” on machine 1
  - Edit the file “version-text.txt” by adding the line “version 3” to it
  - Commit/push all changes on machine 1.
  - View the past version of version-text.txt with line “version 1”
  - View the past version of version-text.txt with lines “version 1” and “version 2”
  - Revert version-test.txt to the past version with lines “version 1” and “version 2”
- Open “Test revert” on machine 2
  - Validate that version-test.txt has been successfully reverted back to the version with lines “version 1” and “version 2”

[4] Smart grouping case:
We are choosing to delay defining what smart change grouping means exactly until we receive feedback from users as to how they value various styles of grouping commits (grouping based on files, grouping based on time changes have been made, etc.).

[5] Final use case:

- Run all permutations of orderings of previous phase tests, and verify that all tests still pass when called in any order.

The Future w/ GitUp:
GitUp will be the connection between cloud backup service, version control system and version control wrapper which enables individuals to use the most essential features of the trifecta. With GitUp, projects that would require multiple separate platforms to maintain can be aggregated into one with a guarantee that changes can be controlled safely and specifically. The first stable version is focused on individual users but later versions will map core functionality to team projects, *getting* cross disciplinary teams *up* to speed with the powerful project management of GitUp.
Hours worked on proposal & presentation: 35 (team total)