2/5/19

Kaushal Mangipudi, Gerard Gaimari, Robert Kolmos, Kamden Chew

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

Motivation:

Backing up a project is essential to its reliable completion. To backup their work, many programmers use distributed version control systems like Git. Git provides a robust system for tracking changes in a project, and comes with a plethora of features that are invaluable for teams working on projects like pushing/pulling, tagging, branching, checking out past versions, etc. Its power has allowed it to become one of the dominant version control systems out there - a StackOverflow survey found that in 2018, almost 90% of developers used Git to backup their work.[1] Git's popularity does not mean that it is considered perfect. We have had our fair share of mishaps with Git, and have struggled with things like causing merge conflicts by failing to pull changes and reverting past versions of files. As single developers working on small scale projects, a lot of the Git functionality was unnecessary, and was in fact detrimental to success. We were often overwhelmed by commands we almost never needed to use, and sometimes ended up using the wrong command as a result of the excessive complexity. Git, while usable for single users, was far from optimal. It felt bloated at times, and we were often bogged down and confused by functionality that we would almost certainly never use. We're not alone in our Git woes - studies have shown that a convincing majority of users believe that Git's UI and documentation need improvement.[2] Additionally, many users believe that Git is too complex and has too many features, which can overwhelm even experienced developers, and make it very difficult for inexperienced users to use.[2]

There are many wrappers for Git like GitKraken that aim to make Git more user-friendly. They simplify many commands down to a couple button clicks, while also making it easier for teams to coordinate.[3] However, these wrappers still include the overwhelming and excessive amount of functionality that can confuse users. A lot of this functionality is necessary for large teams working on projects, but for single users, it only bogs them down. For example, merge conflicts are often inadvertent in large scale projects, but for a single user, they would mostly occur as the result of user error, and can and should be handled in a way to prevent the problems from occurring in the first place.

In conclusion, Git's design clearly caters to larger-scale, multi-user projects that may require a more feature-rich environment, and begs for an alternative for smaller, single-user projects. Git Wrappers simplify many aspects of Git, but still contain unnecessary functionality that single users don't need.

Proposed Solution:

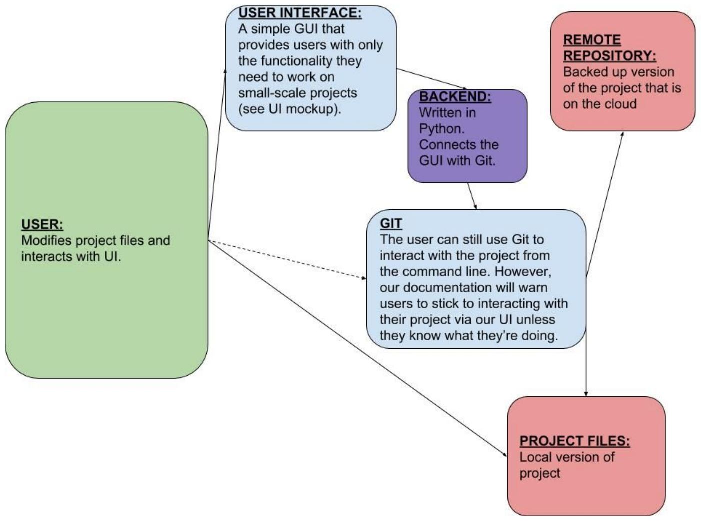
GitUp is a product with the goal of building on reliable, automatic, and secure backup features by providing users with robust version tracking of files in an intuitive way. GitUp will achieve this goal by using relevant features from the Git version control system and abstracting their usage for the purpose of backing up individual projects created by single users.

GitUp will be designed for users working on single-developer projects, with a focus on inexperienced users who are not comfortable with Git. To use GitUp, a user will open the application and select the project they want to work on. GitUp will automatically track project file versions and update the backup whenever it detects a change. It will provide simple version restoration with granularity ranging from the entire project to individual file changes.

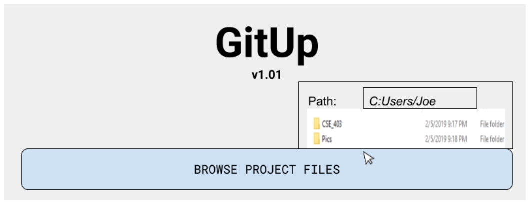
No other system has been created with the idea of single user projects in mind. Existing products are sufficient, but not ideal, for this use case. GitUp will be created to fill the gap that Git leaves in terms of catering to smaller-scale single-user projects, with a focus on being simple and intuitive to use. Our final product aims to (1) automatically maintain a project backup, (2) automatically group project changes in a meaningful way, (3) be intuitive to users with little/no experience with other version control systems, and (4) give users with a simple way to view past versions of a project (down to file level) and revert.

Architecture:

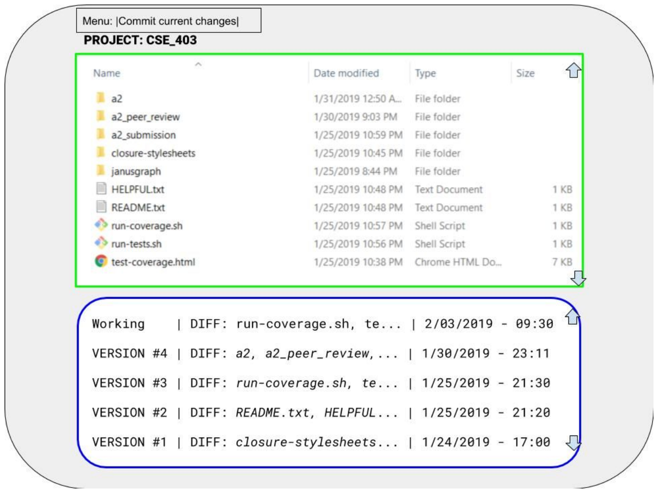
After the user creates a Git repository and clones it locally, they can interact with their repository with GitUp. GitUp's architecture will be based around a simple GUI that aims to emulate the Windows file explorer interface. The GUI is elaborated on in the next section. When the user interacts with GitUp's GUI, GitUp will use Git commands in the background to provide the necessary functionality. The user can still interact with their project using Git in the command line, but we will include copious warnings in our documentation to stress that only experienced developers that know what they're doing should ever do so. GitUp will use Git to help sync the local and remote versions of the project. Whenever a user opens a project in GitUp, it will try to to use Git to automatically pull the project. If it fails, it will issue a warning to the user that any changes they make might lead to merge conflicts down the road. In addition to periodic automatic pushing and the manual pushing provided in the GUI's functionality, GitUp will attempt to automatically push when it is closed, once again warning the user if the push was unsuccessful.


USER INTERFACE: A simple GUI that provides users with only the functionality they need to work on small-scale projects (see UI mockup).
REMOTE REPOSITORY: Backed up version of the project that is on the cloud
BACKEND: Written in Python. Connects the GUI with Git.
USER: Modifies project files and interacts with UI.
GIT The user can still use Git to interact with the project from the command line. However, our documentation will warn users to stick to interacting with their project via our UI unless they know what they're doing.
PROJECT FILES: Local version of project

GUI Design:


GitUp
v1.01
Path:
C:Users/Joe
CSE_403 2/5/2019 9:17 PM File folder
Pics 2/5/2019 9:18 PM File folder
BROWSE PROJECT FILES

When the user opens GitUp, they are greeted by the menu to the left. If they click on Browse Project Files, another window pops up that closely emulates the file explorer. The user will be able to type a path, and all directories in that path will be visible. The user can click on the project they wish to work on, and then GitUp will open up that project.



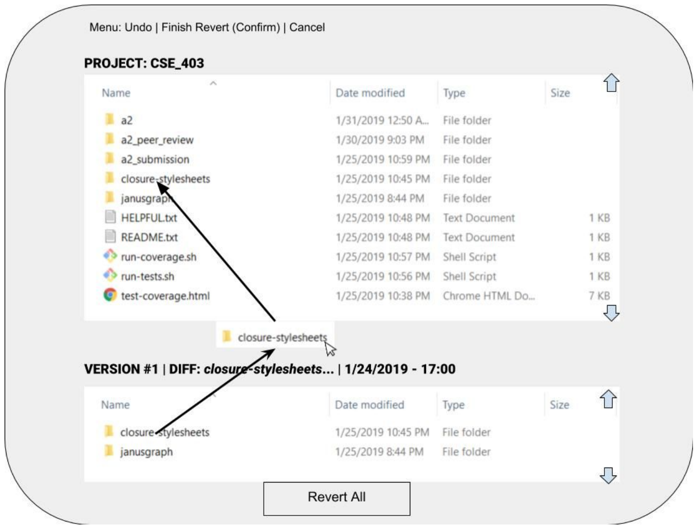
Menu: |Commit current changes|

PROJECT: CSE_403

Name	Date modified	Type	Size
a2	1/31/2019 12:50 A...	File folder	
a2_peer_review	1/30/2019 9:03 PM	File folder	
a2_submission	1/25/2019 10:59 PM	File folder	
closure-stylesheets	1/25/2019 10:45 PM	File folder	
janusgraph	1/25/2019 8:44 PM	File folder	
HELPFUL.txt	1/25/2019 10:48 PM	Text Document	1 KB
README.txt	1/25/2019 10:48 PM	Text Document	1 KB
run-coverage.sh	1/25/2019 10:57 PM	Shell Script	1 KB
run-tests.sh	1/25/2019 10:56 PM	Shell Script	1 KB
test-coverage.html	1/25/2019 10:38 PM	Chrome HTML Do...	7 KB

```
Working    | DIFF: run-coverage.sh, te... | 2/03/2019 - 09:30
VERSION #4 | DIFF: a2, a2_peer_review,... | 1/30/2019 - 23:11
VERSION #3 | DIFF: run-coverage.sh, te... | 1/25/2019 - 21:30
VERSION #2 | DIFF: README.txt, HELPFUL... | 1/25/2019 - 21:20
VERSION #1 | DIFF: closure-stylesheets... | 1/24/2019 - 17:00
```

Once the user selects a project, they are greeted by the menu to the left. The user is shown the current state of all the files (bordered in green) as well as a history of the groups of changes that have been made (bordered in blue). Hovering over a past version displays all changed files. The user can manually commit all current changes by clicking 'Commit current changes'.



PROJECT: CSE_403

Name	Date modified	Type	Size
a2	1/31/2019 12:50 A...	File folder	
a2_peer_review	1/30/2019 9:03 PM	File folder	
a2_submission	1/25/2019 10:59 PM	File folder	
closure-stylesheets	1/25/2019 10:45 PM	File folder	
janusgraph	1/25/2019 8:44 PM	File folder	
HELPFUL.txt	1/25/2019 10:48 PM	Text Document	1 KB
README.txt	1/25/2019 10:48 PM	Text Document	1 KB
run-coverage.sh	1/25/2019 10:57 PM	Shell Script	1 KB
run-tests.sh	1/25/2019 10:56 PM	Shell Script	1 KB
test-coverage.html	1/25/2019 10:38 PM	Chrome HTML Do...	7 KB

closure-stylesheets

VERSION #1 | DIFF: *closure-stylesheets...* | 1/24/2019 - 17:00

Name	Date modified	Type	Size
closure-stylesheets	1/25/2019 10:45 PM	File folder	
janusgraph	1/25/2019 8:44 PM	File folder	

Clicking on a group of changes brings up the revert changes view. After selecting a given commit (referred to in the UI as version #1) the user is able to drag and drop files from this commit into the master version to revert the files to that version. Clicking "Revert All" reverts all changed files to their past versions. Otherwise, after dragging and dropping, the user can either undo some drags, confirm the revert, or cancel it altogether. Clicking on a directory in the lower menu will open up the directory's contents that have changed, and clicking on a file in the lower menu opens up that version of the file.

Technologies:

The UI will be implemented with Tkinter[4] for Python since this library is platform independent allowing a single build to produce a GUI for OS X, Linux and Windows 10 if necessary. Tkinter provides a basic UI kit that should be sufficient to create a UI resembling the above mockups. Python was chosen for GUI implementation so we can easily interface with the GitPython[5] API that GitUp will use to utilize Git in order to interact with the local project files and the remote repository.

Challenges and Risks:

Design Challenges:

One fundamental design 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 using a group of experienced developers and a group of inexperienced developers who have kindly agreed to help us. 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.

One 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 challenging since it's not a functionality provided by Git, and machines will need to know when another machine has a lock on a file. A second 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. We can group commits using a method that is time-based, file-based or both. If we decide to group commits by time, then we might choose to group commits that have happened since the last pre-defined time interval without changes. If we instead decided to group commits by file, then we might group all commits that edit a particular file.

Implementation Schedule:

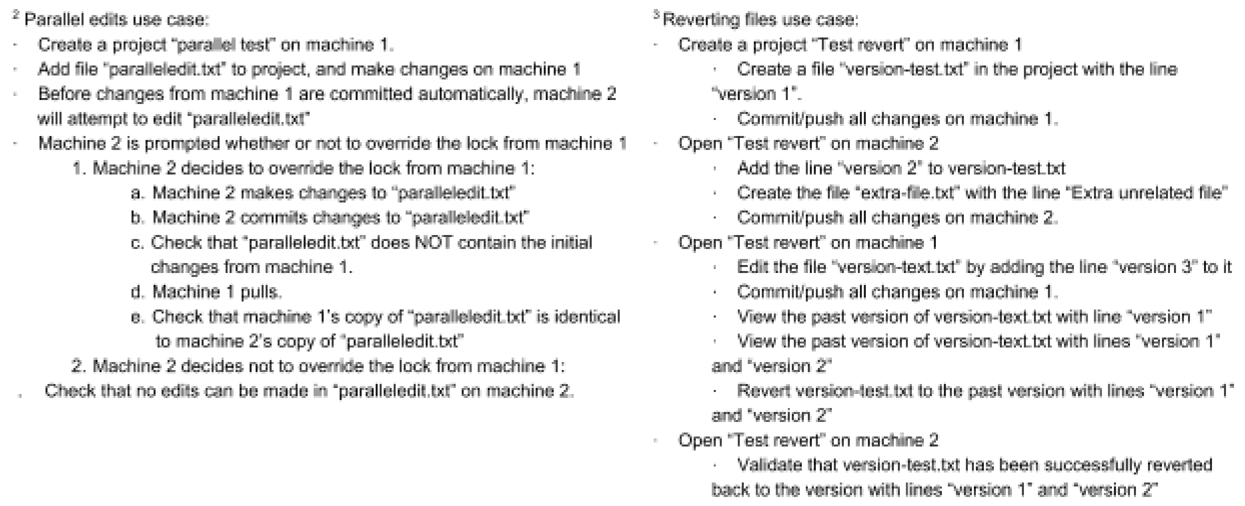
Date	Goal(s)	Test(s)	Implementers	Evaluators
02/04	Finish preliminary user usage data and finish designing a preliminary UX/UI	Be able to run example use cases through UX flowchart. Results of survey collected.	Rob, Kamden	Gerard, Kaushal
02/11	Automated pushing/pulling	Be able to complete the basic project use case [1]	Kaushal, Gerard	Rob, Kamden
02/18	Edge case avoidance/File reverting	Be able to complete the parallel edits use case [2] and reverting files use case [3]	Gerard, Rob	Kaushal, Kamden
02/25	Begin work on smarter grouping of changes	Collect customer data on what users think a 'smart grouping' of changes are, and write a smart grouping use case/test based off the customer data	Kaushal, Kamden	Rob, Gerard
03/04	Smarter grouping of changes	Be able to complete the smart grouping [4] and final use cases [5]	Kaushal, Gerard	Rob, Kamden
03/11	Collect and respond to feedback on finished GitUp	Feedback fully collected.	Kamden, Kaushal	Rob, Gerard
03/18	Feedback Implementation	Dependent on feedback received. Fixes implemented address most important user concerns with GitUp	Kamden, Gerard	Kaushal, Rob

Testing & Evaluation:

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.
    - Validate that version-test.txt has been successfully reverted back to the version with lines "version 1" and "version 2"



[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:
- 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.

Evaluating GitUp's success will require that we not only test that the implementation is working as expected, but that it achieves the goal of making project management for new/novice developers easy and intuitive. To make sure that our product hits this mark, we have a pool of new/novice developers who are willing to try out GitUp after it's implementation is complete. We will be able to evaluate if our product is useful to its target audience by seeing if our pool of novice testers are able to navigate the system easily, or at least more easily than competitors. If we find that GitUp is an easier version control system for inexperienced users to use, then we will have filled the glaring gap that Git leaves behind in regards to simple and easy version control for novice developers who are focused on small-scale single-user projects. GitUp can fill this niche while also serving as a gentle introduction for new developers into version control systems.

Work Used:

[1]https://insights.stackoverflow.com/survey/2018
[2]https://spderosso.github.io/onward13.pdf
[3]https://www.gitkraken.com/git-client
[4]https://docs.python.org/3/library/tk.html#tkinter
[5]https://gitpython.readthedocs.io/en/stable/

Hours worked on proposal: 50 (team total)