2/5/19

Kaushal Mangipudi, Gerard Gaimari, Robert Kolmos, Kamden Chew

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

Motivation:

Backing up any project is essential to its reliable completion, and software projects are no exception. Today, more and more inexperienced non-developers work on software projects, and while they are the only coder and their projects are relatively small, they still need to backup their work. Due to their lack of experience with software development, non-developers need different things in a backup system than experienced programmers. A backup system that satisfies all the needs of non-developers must:

1. **Offer long-term tracking of many past versions of files and allow reverting to said past versions.** It's nearly impossible to determine when specific changes were made to a piece of code.
2. **Offer a quick and easy way to compare past versions with one another.** Often times, the differences between versions of a file can be very small and hard to see with the naked eye, as bug fixes often involve changing only a couple lines.
3. **Be easy to use.** Non-developers are already short for time. Due to their inexperience, they already spend far more time writing their actual code than actual developers. Even worse, since coding isn't their primary job, they have less time to code than full time developers. They don't have the time to learn a complicated system, so any backup system they use must be fast to learn.
4. **Be as automated and out of the way as possible**. Non-developers are already out of their depth. Unlike software developers who use version control systems, they are not used to using some third party system to back up their work after saving their work. As a result, it is unreasonable to expect them to constantly remember to take the extra step of manually backing up their work after they save it regularly.

There are a plethora of backup tools available for non-developers to use, but unfortunately, none of them meet all of their needs. Existing backup tools can be divided into four broad categories:

1. **Cloud Sync services (OneDrive, Google Drive, DropBox, etc.)**. They are very easy to use and automatically save a user's work and sync it with other copies of the user's work on other machines and the cloud. Some of them, like Google Drive, let you compare a version with the directly proceeding or preceding version to see what changes are made. However, Cloud Sync services don't have reliable, long-term version tracking. They save a few of the past versions and give you the option of reverting to them. However, they only save the past versions for a couple of weeks at the longest. If a change you want to undo happened a while back, or if the change you want to undo isn't cleanly isolated to one of the past versions, it will be impossible to easily undo the change.
2. **Cloud Backup services (BackBlaze, Mozy, Carbonite, etc.).** Like Cloud Sync services, for the most part, they are easy to use and automatically backup a user's work. They store more past versions of a user's work for a long time, so they allow a user to more easily undo changes they made, even if the changes happened a while ago. However, they don't provide the user with an easy way to identify what the changes were. If a user wanted to compare two past versions of a file with one another, they'd need to open up a text comparison tool and paste both versions of their file into it, which is a waste of time and very disruptive to their workflow.
3. **Dedicated Version Control Systems (git, Mercurial, Subversive, etc.).** They offer comprehensive, near eternal version tracking and allow a user to easily compare past versions of a

file with each other. However, they are very difficult to use and require users to interrupt their work flow to backup their work. Git, by far the most popular version control system[1], was reported to have a very steep learning curve and an inadequate UI and documentation by over 70% of its users[2]. If experienced developers are overwhelmed by git's enormous amount of functionality, non-developers will certainly struggle even more. Additionally, these version control systems use command line interfaces, which can be very hard to use if you have limited experience with them.

4. **VCS 'Simplifiers' (GitKraken, Gitless, TortoiseHg, SmartGit, Tower, GitHub Desktop, etc.).** These tools purport to make VCSs easier to use, and they do, but only for developers already familiar with how VCSs work. Some of them still rely on command line interfaces, and all of them still require a user to understand how VCSs work. That isn't a problem for developers, but non-developers don't have the understanding, and don't have the time to obtain it.

Proposed Solution:

GitUp aims to preserve the reliability, automatic nature, and security that existing cloud-based backup systems provide while also providing users with the robust version tracking of files they need for software projects in an intuitive way. GitUp will be designed for users working on single-developer projects, with a focus on inexperienced users who need to code but work outside of software. To use GitUp, a user will open the application and login to their GitHub account. Once they are logged in, they 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. Users who work outside of software and have never used version control systems won't be able to use them without investing considerable amounts of time to learning them. GitUp will allow these inexperienced users to enjoy robust version tracking without having to learn about the underlying architecture of git or some other version control system.

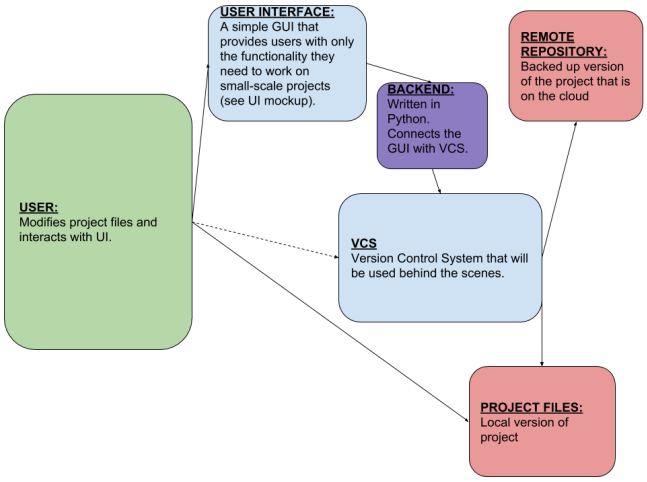
GitUp's Target Functionality:

1. The user should be able to create a new GitHub project or open an existing one
2. Any locally saved changes should be automatically backed up on the remote repository
3. The local version of the project will be periodically synced with the remote repository
4. Automatically backed up past versions will be grouped by files modified and/or time modified before being visible to users
5. The user should be able to view past versions of files and revert a file to a past version if they desire to
6. The risk of merge conflicts will be almost completely eliminated, and the user will be warned in advance about any potential merge conflicts that could occur
7. A very inexperienced user with minimal programming experience and no experience with version control systems should be able to do all of the above with the help of the user manual

User Interface:

See GitUp's User Manual for a detailed explanation of what GitUp's UI will look like and how users will interact with it.

Architecture:



GitUp will be designed to work on Linux machines, though we may expand it to other Operating Systems if time permits us to. It will use a version control system behind the scenes. This could have been any version control system, but it was decided we'd use git since it's the version control system we are most familiar with and because we plan on using OAuth to link GitUp with the user's GitHub account. The user can work on a project using multiple devices, but each project can only be worked on by a single user, and the user should only use one device at a time. GitUp's architecture will be based around a simple GUI that aims to emulate the interface of cloud-base backup systems like Google Drive and Dropbox, with additional functionality involving viewing and reverting past versions of files. 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 technically interact with their project using git in the command line, but the user will never need to, as GitUp will handle all interactions between the local and remote repositories, and will in fact be encouraged not to in the user manual. GitUp will use git to help sync the local and remote versions of the project. GitUp will automatically pull the project. If it fails, it will issue a warning to the user that any changes they make might to certain files might lead to merge conflicts down the road. Since there is a single user, they can know to avoid modifying those files until they have been properly synced. If an automatic push fails, GitUp will periodically reattempt to push the file to the remote repository. If the user closes GitUp before a push succeeds, GitUp will issue a warning that the file is out of sync.

Implementation:

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. GitUp will automatically push by becoming aware of file opens and closes through Linux's inotify API.[6] When a user sets up a project, GitUp begins monitoring file opens, and closes in the directory associated with this project and it's subdirectories. When GitUp sees that a file has been opened it stores the current state of the file, and begins checking intermittently to monitor the magnitude of changes the user has made. When the user

closes the file, or GitUp determines that a significant amount of changes have been made, GitUp automatically commits the changes to this file. If the file is still open when a commit is made, GitUp updates the stored current state of the file, and proceeds monitoring changes. When a file is closed GitUp stops intermittently monitoring it for changes. We will display a warning to a user about potential merge conflicts whenever an automatic push or pull fails, and to do so, we will use a popup implemented with Tkinter to do so. We will not put any hard blocks on modifications that could lead to merge conflicts, because the user is guaranteed to have seen a warning about merge conflicts since they are the only user working on the project. If the user still causes a merge conflict despite the warning, we will automatically resolve the merge conflict, though we haven't determined how exactly we will do so. With regards to grouping commits together to give the user a big picture of changes made to their project over time, we plan on grouping commits together by the files modified and/or the times they were modified. We will save our grouping data in a file that will be pushed to the repository in GitHub but will remain invisible to the user when using GitUp's interface. The exact algorithm to group commits will be finalized after testing.

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 developers familiar with git and a group of inexperienced developers with little to no version control experience 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 how we plan on handling merge conflicts when the user causes one despite numerous warnings to be careful. We see three potential solutions to this. The first is to automatically replace the file with the version that is being pushed. A second possibility would be to save a renamed version of the file to the repository that the user could look at. The third would be to utilize a GUI based merge conflict resolution tool to help the user manually resolve the conflict. Given the complexity of difficulty of the third option, we are leaning towards either the first option or the second.

Another technical challenge is grouping automatic commits together in meaningful ways that users will able to use to get a big picture idea of how their project has changed over time. To that end, we will group commits using a method that is time-based and/or file-based. 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. Either way, we will need to fine tune the intervals which we will use to divide all changes into groups of commits. It will be difficult to choose numbers to use for this, so we'll need to test various frequencies to determine the best one to use.

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 and OAuth	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 syncing warnings use case [2] and reverting files use case [3]	Gerard, Kaushal	Rob, 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, Kamden	Rob, Gerard
03/11	Collect and respond to feedback on finished GitUp	Feedback fully collected.	Rob, Gerard	Kaushal, Kamden
03/18	Feedback Implementation	Dependent on feedback received. Fixes implemented address most important user concerns with GitUp	Rob, Gerard, Kaushal	Kamden

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:

- Open GitUp on machine 1
- Login to GitUp using a GitHub account
- Create a project "hello world" on machine 1.

- Add file helloworld.txt to project, and make changes on machine 1, and save the changes
- Verify that the changes were automatically pushed.
- Close GitUp on machine 1
- Open GitUp on machine 2
- Open the “hello world” project on machine 2.
- Verify that the version of “hello world” on machine 2 is up to date
- Make changes to helloworld.txt on machine 2, and save
- Close GitUp on machine 2
- Open GitUp on machine 1
- Verify that helloworld.txt is up to date

[2] Merge conflict use case:

- Create a project “parallel test” on machine 1.
- Add file “paralleledit.txt” to project, and make changes on machine 1
- Turn off internet on machine 1
- Save the changes on machine 1
- Verify that the user is warned that the automatic push failed
- Close GitUp, verifying that the user is warned once again that their changes haven’t been backed up
- Turn on the internet in machine 1
- Verify that the changes are pushed to the remote repository
- Close GitUp
- Verify that there are no warnings about merge conflicts
- Turn off the internet on machine 2
- Open GitUp on machine 2, and open parallel test
- Verify that a warning appears about not being able to sync the local version of parallel test with the remote version

[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 non-developers easy and intuitive. To make sure that our product hits this mark, we have a pool of non-developers with little to no version control experience 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. To that end, we have standardized testing instructions and surveys for each phase of development. We'll have the users attempt to implement the Song assignment from CSE 142 using GitUp to backup their code. We'll give them 3 hours to do so, and ask them to use various parts of GitUp's functionality (working on multiple machines, reverting to a past version, etc.). The instructions give general goals that we want the testers to be able to achieve, and they will attempt to satisfy those goals in a time limit by only using the user manual for assistance. After they complete the test (or run out of time), we have them fill out a survey about their experience. Specifically, we'll have them answer questions about how using GitUp impacted their workflow, how easy the UI was to use, and any additional feedback. Then, we'll have them do the same with git. 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. Additionally, we'll ask a group of developers who are familiar with git to try making their own project using GitUp and fill out a survey focused on whether they were able to successfully do so using GitUp and multiple machines.

Feedback Response:
Zhen expressed concerns with the motivation behind GitUp during our presentation. Specifically, he thought that we shouldn't talk about cloud sync services. We believe that they are relevant competitors to GitUp, so we left that section in, but we recognized that we weren't clear about why they were relevant, so we've expanded upon that in our motivation section.

Martin expressed concern that we didn't link an actual URL in our User Manual to download GitUp. We recognize that this is an issue, but we haven't finalized how we will distribute GitUp to users. We plan to finalize this in a couple days, at which point we will update the manual.

Work Used:

[1]https://insights.stackoverflow.com/survey/2018 **Accessed 2/12/19**
[2] **What's Wrong with Git? A Conceptual Design Analysis. Santiago Perez De Rosso, Daniel Jackson. Computer Science and Artificial Intelligence Lab at MIT.** ***Presented at the Onward 2013 Conference, p. 37-51.***
Accessed 2/12/19 at https://spderosso.github.io/onward13.pdf
[3]https://www.gitkraken.com/git-client **Accessed 2/12/19**
[4]https://docs.python.org/3/library/tk.html#tkinter **Accessed 2/12/19**
[5]https://gitpython.readthedocs.io/en/stable/ **Accessed 2/12/19**

Hours worked on report (team total): 87