Mission Statement

------------------

High level summary of project
What are you doing?
Who is it for?

We are building a simplified source control tool to help introduce new programmers to the concepts and best practices of using revision control systems. Our tool is going to support the essential operations for source control: push/pull, commit/amend, and rollback. In addition we are incorporating web-based code review and collaboration. This project is targeted at new programmers, typically in college, who are starting to work on multi-developer projects for the first time.

Problem

--------

What is the problem you are solving?
What is the need you are filling?

The problem we are solving is that source control is complicated. Despite being an essential and everyday component of software engineering, it is not a major focus of study and programmers often have the mentality of “know as little as possible to get by”. By simplifying the source control process we are able to teach the fundamentals of source control that can then be applied to more complicated scenarios and tools. In other words, git is difficult to learn from scratch and there aren’t clear guidelines on best practices. We want to change that.

Context

--------

Describe the context or circumstances
What are the existing solutions?
Why is something new needed here?

The circumstances somebody would use this involve learning source control for the first time. Whether it’s in a classroom or just a group of friends doing a side project, we hope that our tool will be an aid to help collaboration and make the transition to industry software engineering. Currently the best solution is reading “git - the simple guide” and hoping everything works out. This means using an incredibly powerful program -- git -- in a very simple and straightforward way, which makes the majority of git’s feature set and complexity irrelevant and confusing.

Customer

---------

Who is the customer?
What is the value to them?
Are there other stakeholders?
What is their interest?

Our customer is anybody who wants to learn source control. We are able to offer them a gentle introduction to usage and practices of source control while letting them continue with their other work. The idea is to make source control collaboration seamless as simply and quickly as possible. Because this isn't a product for sale, the stakes are a bit lower. The other stakeholders would be other source control companies that either want more people to use them (ie. Mercurial vs git vs Subversion), or educational resources like Codeacademy or Lynda. The education sites would like to include this element to improve their product.

Challenges

What are the critical make-or-break features?
Examples: form, budget, responsiveness
Other special circumstances?

There are a few key features that our version control system would need to have. It needs to be able to support commits, pushing to the "cloud", pulling from the cloud, and rolling back to a previous version. Once we have these features, we also need to ensure that they perform at a comparable level to existing tools. Specifically, our source revision tool cannot be unusably slow, or inaccurate/unreliable in how it handles merge conflicts. We will also need to make sure that our tool eventually is in a relatively memory-efficient state so that it is able to accommodate even somewhat large projects. If our implementation is too bloated, we will not be able to offer it under free/cheap tier hosting platforms, and so our revision control system will not be able to be offered for free.

Expertise

What special technologies or knowledge will you need?
How will you acquire the necessary expertise?
What sources are available, accessible?
Explain the underlying theory of the problem domain

There are five main areas that we will need to focus on to make our product successful. Firstly, we will need to set up a database of some kind, which means we will need to understand how to represent diffs in data structures, and then how to store them efficiently. We will need to investigate the different types of databases (relational, object, key-value) to decide which is most appropriate. Secondly, we will need to set up a server which handles push/pull requests and deals with merge conflicts. In order to use our server we will need to, thirdly, get a good understanding of networking. We will need our network calls to be relatively efficient. Fourthly, we will need to determine how to represent diffs locally on a machine so that we can handle local changes and commits. And finally, we will need to learn about web-programming so that we can create a UI which displays code for online code review (stretch goal).

First and foremost, we will likely acquire much of the expertise using online tools (google, stack overflow). Furthermore, Elena and Kalina are in a Cloud Applications class this semester, which means they will have some expertise knowledge on databases. Most of the networking information we need is likely online in tutorials/docs. To understand how to represent diffs, we may or may not be using git/mercurial as a model (since they have already solved this very difficult problem). Much of the extremely domain specific knowledge will be surrounding source revision, so we will need to make ourselves very familiar with tools such as git.

Source revision works, as it does today, on the fundamental concept that you only need to store 'diff', or changes to the files (rather than storing entire copies of every file).

Risks

------

What are the risks to success?
What are your contingency plans?
How can you mitigate these risks?
Examples?

The biggest risk we face is discovering that this project's scope is too large for us to complete it in the time given. While we have some idea of what this will entail, there are a lot of moving pieces and we do not know the full picture. It is possible that we will quickly discover we cannot realistically implement a working revision control system. To minimize this risk, we are planning to focus initially on implementing things in the simplest way possible. Once we have a working proof of concept, we will work on improving performance. This may mean that initially, we do not focus on the algorithm for merging conflicts, and instead just present the user with the contents of both files when we encounter a file changed by multiple people. Or we may choose to implement everything in python, although it may be too slow, and then focus on implementing some pieces in a more efficient language such as C.

Ethical questions

-----------------

Are there ethical questions about the project?
Who is impacted?
What is the underlying ethical issue?
How will you investigate and

mitigate this issue?

While we are not working with personal data, we are working with intellectual property. We therefore need to ensure that our implementation is at least somewhat secure in terms of who is allowed to do what. For instance, it may be reasonable to allow all users to pull from any repo (if repos are assumed to be public), but it is not reasonable to allow all users to push to any repo. In cases like this, the victims would be the users who rely on the version control system to maintain a backed-up and up-to-date copy of their project. If it is possible for that to be compromised through malicious use, then the risk may not be worth it. To mitigate this, we will strictly limit write privileges to our underlying database to ensure that only authorized users can make changes to a repo.

Goal

What is the desired result?
Describe what success looks like

Our base goal is to create a source revision tool with a simple, easy to user interface. Our stretch goals include implementing code syncing features for uncommitted changes (to facilitate code review), implementing an online interface to display uncommitted changes, and implementing stacked commits that can be reordered. The first two stretch goals are features that git and mercurial currently do not have, but we believe that all source revision tools (even bare-bones ones) should include this because, in industry, software engineers have to do code review. The third stretch goal is to implement stacked diffs that can be reordered, which is something that git and mercurial already have, but we believe that it's a very useful feature which new source control users would benefit from learning about.