CS 61BL Project 2: `Gitlet Design Document`

Design Document Due:	Saturday, July 13 at 11:59pm
Code Due:	Friday, July 26 at 11:59pm

1 Design Document Philosophy

Design documents are important tools for software engineers. Writing down designs on paper helps us to plan out our project and notice any flaws before actually trying to implement it in code. While design docs may take additional time to create, they actually speed up the development process because problems are addressed sooner and it is clear how everything fits together. This also gives you a chance to get someone more experienced (your TA!) to help you improve your design. Improving ideas on your design doc are much easier than searching through your code!

2 Requirements

Before you start writing any code for your project, you should create a plan for each feature and convince yourself that your design is correct. For this project, you must **submit a design document** and **attend a design review** with your TA.

2.1 Design Document Guidelines

With your partner, come up with ideas for your Gitlet implementation. Once you finalize your design, write your implementation details down. You could use Google Docs, Dropbox Paper, Markdown Editors, and much more. What matters is that it's in a presentable format that will be easy for you and your TA to read. Your document should be detailed enough that your TA or another student taking CS 61BL could fully recreate your project using just your Design Doc. **We will be enforcing a 4-page limit to your design docs**.

We recommend that your Gitlet implementation follows this format.

1. **Data structures and functions** – Write down any class definitions, global variables, or major functions that you will be adding or using (if it already exists). Include a **brief explanation** the purpose of each modification. Your explanations should be as concise as possible. Leave the full explanation to the following sections. You may cut this section short if you find you are running out of space, but be sure to include important details like what classes you are creating. See here for an example of a potential `Commit.java` class.

2. **Algorithms** – This is where you tell us how your code will work. Your description should be at a level below the high level description of requirements given in the spec. We have read the project spec too, so it is unnecessary to repeat or rephrase what is stated there. On the other hand, your description should be at a level above the code itself. Don't give a line-by-line run-down of what code you plan to write. Instead, you should try to convince us that your design satisfies all the requirements, **including any uncommon edge cases**.

The length of this section depends on the complexity of the task and the complexity of your design. Simple explanations are preferred, but if your explanation is vague or does not provide enough details, you will be penalized. Here are some tips:

- For complex tasks, like determining merge conflicts, we recommend that you split the task into parts. Describe your algorithm for each part in a separate section. Start with the simplest component and build up your design, one piece at a time. For example, your algorithms section for Merge Conflicts could have sections for:
  - Checking if a merge is necessary
  - Determining which files (if any) have a conflict
  - Representing the merge conflict in the file
- Use **bold** for variable names and *italics* for function names to easily distinguish these from the rest of your text. You can use your own style, but keep it consistent throughout the document.
- Lists can make your explanation more readable. If your paragraphs seem to lack coherency, consider using a list.

3. **Persistence** – Describe your strategy for ensuring that we never lose our gitlet state or files. Here are some tips for writing this section:
   - This section should be structured as a **list of all times we need to record our state/files**. For each case, you should prove that your design ensures correct behavior.
   - You should also aim to make your persistence efforts as efficient as possible, in terms of time and storage space.
   - Persistence issues revolve around the idea that you should *never* lose the data that is important. A good strategy for reasoning about persistence is to identify which pieces of data are needed across multiple calls to Gitlet. Then, prove that the data remains consistent for all future calls.
4. **Rationale** – Tell us why your design is better than the alternatives that you considered, or point out any shortcomings it may have. You should think about whether your design is easy to conceptualize, how much coding it will require, the time/space complexity of your algorithms, and how easy/difficult it would be to extend your design to accommodate additional features. Remember, Gitlet only has a fraction of the features that Git has!

A sample design document written for Project 0 can be found here.

2.2 Design Review Guidelines

Your design review will be a short meeting with your TA discussing your design doc. You should be prepared to defend your design and answer any design-related questions your TA may have for you. This will happen during lab on Monday, July 15. Sign-ups will be released on Friday, July 12.

3 Deliverables

Your project grade is out of 36 points, and will be made up of 2 components:

- 6 points: Design Document and Design Review
- 30 points: Code