Meant to break down Gitlet instructions, organize classes + pseudocode

- **blobs**: Essentially the contents of files.
- **trees**: Directory structures mapping names to references to blobs and other trees (subdirectories).
- **commits**: Combinations of log messages, other metadata (commit date, author, etc.), a reference to a tree, and references to parent commits. The repository also maintains a mapping from branch heads (in this course, we've used names like `master`, `proj2`, etc.) to references to commits, so that certain important commits have symbolic names.
    - *branch heads*: certain important symbolic commits
- **Branches:** Maintaining related sequences of commits.

Having our metadata consist only of a timestamp and log message.
A **commit**, therefore, will consist of a **log message**, **timestamp**, a **mapping of file names to blob references**, a **parent reference**, and (for merges) a **second parent reference**

- **two objects** with **exactly the same content** will have the **same id on all systems**
- case of **blobs**, "same content" means the same file contents
- case of **commits**, it means the same metadata, the same mapping of names to references, and the same parent reference

cryptographic hash function called SHA-1 (Secure Hash 1), which produces a 160-bit integer hash from any sequence of bytes

All of this stuff **must** be stored in a directory called `.gitlet`, just as this information is stored in directory `.git` for the real git system (files with a `.` in front are hidden files. You will not be able to see them by default on most operating systems. On Unix, the command `ls -a` will show them.)

Git Commands:

- **Init:** Creates a new Gitlet version-control system in the current directory. This system will automatically start with one commit: a commit that contains no files and has the commit message initial commit (just like that, with no punctuation)
    - has a single branch: `master`, which initially points to this initial commit, and `master` will be the current branch.
- **Add**: adds a copy of the file as it currently exists to the *staging area* (see the description of the `commit` command)
    - adding a file is also called *staging* the file. The staging area should be somewhere in `.gitlet`.
- **Commit:** taking snapshots to track the saved files,
    - A commit will only *update* files it is tracking that have been staged at the time of commit, in which case the commit will now include the version of the file that was

staged instead of the version it got from its parent → A commit will save and start tracking any files that were staged but weren't tracked by its parent.
    - The bottom line: By default a commit is the same as its parent. Staged and removed files are the updates to the commit.
    - After the commit command, the new commit is added as a new node in the commit tree.

Any changes made to files after staging or removal are ignored by the `commit` command, which *only* modifies the contents of the `.gitlet` directory. For example, if you remove a tracked file using the Unix `rm` command (rather than Gitlet's command of the same name), it has no effect on the next commit, which will still contain the deleted version of the file.

- rm
- log → : Starting at the current head commit, display information about each commit backwards along the commit tree until the initial commit, following the first parent commit links, ignoring any second parents found in merge commits. (In regular Git, this is what you get with `git log --first-parent`). This set of commit nodes is called the commit's *history*.
- Global-log
- Find
- Status
- Checkout***
- Branch***
- Rm-branch***
- Reset
- Merge****(Deadly)

checkout would be used for experimenting down a logical path that you didn't want to affect ur main project
revert would be used for when you are finished and screwed something up in a new commit

This project requires reading and writing of files. In order to do these operations, you might find the classes java.io.File and java.nio.file.Files helpful. Actually, you may find various things in the java.io and java.nio packages helpful. Be sure to read the gitlet.Utils package for other things we've written for you. If you do a little digging through all of these, you might find a couple of methods that will make the io portion of this project *much* easier! One warning: If you find yourself using readers, writers, scanners, or streams, you're making things more complicated than need be.

Serialization:**

In order to successfully complete your version-control system, you'll need to remember the commit tree across commands

- have to design not just a set of classes to represent internal Gitlet structures during execution → need parallel representation as files within your .gitlet directories, which will carry across multiple runs of your program.

Ex.

```
import java.io.Serializable;
class MyObject implements Serializable {  ...
}
```

Ex.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
...
   MyObject obj;
   File inFile = new File(someFileName);
   try {
      ObjectInputStream inp =
         new ObjectInputStream(new FileInputStream(inFile));
      obj = (MyObject) inp.readObject();
      inp.close();
   } catch (IOException | ClassNotFoundException excp) {  ...
      obj = null;
   }
```

- convert obj to a stream of bytes and store it in the file whose name is stored in someFileName.

There is, however, one annoying subtlety to watch out for: Java serialization follows pointers. That is, not only is the object you pass into writeObject serialized and written, but any object it points to as well. If your internal representation of commits, for example, represents the parent commits as pointers to other commit objects, then writing the head of a branch will write all the commits (and blobs) in the entire subgraph of commits into one file, which is generally not what you want. To avoid this, don't use Java pointers to refer to commits and blobs in your runtime objects, but instead to use SHA-1 hash strings. Maintain a runtime map between these strings and the runtime objects they refer to. You create and fill in this map while Gitlet is running, but never read or write it to a file.

You might find it convenient to have (redundant) pointers commits as well as SHA-1 strings to avoid the bother and execution time required to look them up each time. You can store such pointers your serializable while still avoiding having them written out by declaring them "transient", as in

```
  private transient MyCommitType parent1;
```

---

java gitlet.Main.init, git init:

- set up info to set up the classes, saves important information

Head pointer: pointer to a commit that is the most recent commit being tracked in our repo
Head is what I'm looking at, Master is tracking the most recent commit

- detached head means the head is not on a branch

`master` is a reference to the end of a branch. By convention (and by default) this is usually the main integration branch, but it doesn't have to be.

`HEAD` is actually a special type of reference that points to another reference. It may point to `master` **or it may not** (it will point to whichever branch is currently checked out). If you know you want to be committing to the `master` branch then push to this.

Branch: named pointer to a specific commit
Working Directory: folder on the computer where I'm actually storing my code

- nano Hello.txt → adds Hello.txt into our working directory
- java gitlet.Main add Hello.txt
    - git add Hello.txt
- java gitlet.Main commit "Created Hello.txt"
    - git commit -m "Created Hello.txt"

nano World.txt
nano Hello.txt
java gitlet.Main add World.txt
java gitlet.Main add Hello.txt
java gitlet.Main commit "Created World.txt"
java gitlet.Main commit "Created Hello.txt"

```
java gitlet.Main checkout [commit id] -- [file name]
```

SHA: family of cryptographic hashes, makes collisions very rare

- Uniquely identify everything with a SHA-1 hash

Classes for Branch, Commit, Staging, Trees

- Tree consists of branches + committing, maybe handles logs as well

- Write init inside of gitlet folder
- Blobs can be stored within a hashmap, can be serialized into bytes and saved