Git Learning Notes

1. Terminology

- A file is called a blob.
- A dictionary is called a tree.
- A snapshot of the whole file is called a commit.
- A directed acyclic graph of commit is the history.
- An object is a blob,tree or commit.
- A reference is a string alias to the hashcode of a object.
- The master reference usually points to the latest commit in the main branch of development.
- HEAD is a special reference to "where we currenctly are in the history".
- A *Git repository* is the data objects and references.
- Staging area is a mechanism allowing you to specify which file you want to commit, not the whole work dictionary.

2. Git Data Model

We first defne three types of data models:

```
// a file is a bunch of bytes
type blob = array<byte>

// a directory contains named files and directories
type tree = map<string, tree | blob>

// a commit has parents, metadata, and the top-level tree
type commit = struct {
    parents: array<commit>
    author: string
    message: string
    snapshot: tree
}
```

An object is a blob,tree,or commit:

```
type object = blob | tree | commit
```

In Git data store, all objects are content-addressed by their SHA-1 hash.

```
objects = map<string, object>

def store(object):
    id = sha1(object)
    objects[id] = object

def load(id):
    return objects[id]
```

3. Commands

Basics

`<reversion>` here can be hash or reference

- `git help <command>` : get help for a git command
- `git init` : creates a new git repo, with data stored in the .git directory
- `git status` : tells you what's going on
- `git add <filename>` : adds files to staging area
- `git commit` : creates a new commit
- `git log` : shows a flattened log of history
- `git log --all --graph --decorate` : visualizes history as a DAG
- `git diff <filename>` : show changes you made relative to the staging area
- `git diff <revision> <filename>` : shows differences in a file between snapshots
- `git checkout <revision>` : updates HEAD and current branch
- `git cat-file -p <hash>` : The instruction to catch a tree or blob

Branching and Merging

- `git branch` : shows branches
- `git branch <name>` : creates a branch
- `git checkout -b <name>` : creates a branch and switches to it
  - same as `git branch <name>` ; `git checkout <name>`
- `git merge <revision>` : merges into current branch
- `git mergetool` : use a fancy tool to help resolve merge conflicts
- `git rebase` : rebase set of patches onto a new base

Remotes

- `git remote` : list remotes
- `git remote add <name> <url>` : add a remote
- `git push <remote> <local branch>:<remote branch>` : send objects to remote, and update remote reference
- `git branch --set-upstream-to=<remote>/<remote branch>` : set up correspondence between local and remote branch
- `git fetch` : retrieve objects/references from a remote
- `git pull` : same as git fetch; git merge
- `git clone` : download repository from remote

Undo

- `git commit --amend` : edit a commit's contents/message
- `git reset HEAD <file>` : unstage a file
- `git checkout -- <file>` : discard changes

Advanced Git

- `git config` : Git is highly customizable
- `git clone --depth=1` : shallow clone, without entire version history
- `git add -p` : interactive staging
- `git rebase -i` : interactive rebasing
- `git blame` : show who last edited which line
- `git stash` : temporarily remove modifications to working directory
- `git bisect` : binary search history (e.g. for regressions)
- `.gitignore` : specify intentionally untracked files to ignore