Git Internals Workshop

May 10, 2019

*NIX Commands

`mkdir <path>`	Creates a directory/folder
`ls <path>`	Lists the contents of a directory
`find <path> -type f`	Finds all files in a given directory
`cat <path>`	Prints contents of a file to `stdout`
`echo <message> > <path>`	Overwrites a file with the given message

`git` Commands

`git init`	Creates and populates `.git/` directory
`git hash-object -w <path>`	Saves an object to the Git object store and prints the hash
`git cat-file -p <object hash>`	Prints the contents of an object in the Git object store
`git cat-file -t <object hash>`	Prints the type of an object in the Git object store
`git update-index --add <path>`	Adds a file to the Git index
`git ls-files`	Lists the files in the current index
`git write-tree`	Saves the index as a tree in the Git object store and prints the hash
`echo <message> | git commit-tree <tree hash>`	Saves a commit pointing to a tree in the Git object store and prints the hash
`echo <message> | git commit-tree <tree hash> -p <commit hash>`	Saves a commit as above, but with (at least one) parent commit
`git log <commit hash>`	Prints all the commits previous to the one specified by the hash
`git log <commit hash> --all --decorate --oneline --graph`	Prints the log as a easily human parsable graph
`git update-ref refs/heads/<branch> <commit hash>`	Creates a named branched that points at the given commit
`git rev-parse <branch>`	Prints the commit hash to which the branch points
`git branch`	Lists all the branches in the repository
`git symbolic-ref HEAD refs/heads/<branch>`	Sets `HEAD` to point at the specified branch
`git update-ref refs/tags/<tag> <object hash or reference>`	Tags a git object with a lightweight tag
`git tag -a <tag> <object hash or reference> -m <message>`	Tags a git object with an annotated tag

Exercises

Git Object Store

1. Create a new directory and initialize git
2. Look inside the `.git` folder. Make sure you understand its contents.
3. Create a file and put some contents in it.
4. Save that file to the git object store. Find it in the git object database.
5. Use a plumbing command to see the contents of the object you just saved. *Try steps 3-5 several times until you are comfortable saving files to the database*

6. Make a change to a file. Save the changed file to the object database. Before you find it, what do you expect to see? Were you right?
7. *Bonus* Delete a file and recreate it from the git object store.
8. *Bonus* Change the name of a file, and save it to the object database. Before you find it, what do you expect to see? Were you right?

Blobs and Trees

1. Add a file from part 1 to the index.
2. Find the index file in the `git` folder.
3. Check the contents of the index with a plumbing command. Confirm the contents with `git status`.
4. Save the index as a tree in the object database. Find the file in the database.
5. Look at the contents of the tree object. Make sure you understand each line. *Repeat 1-5 until you are comfortable saving trees to the database.*
6. Change a file and add it to the index. Save the tree to the object database. Before you look at the contents of the tree, what do you expect to see? Were you right?
7. Save a subdirectory with files to your database. Inspect both the root tree and the subtree. Make sure you understand each of the trees' contents.

Commits

1. Create a commit from a tree in part 2.
2. Find your commit object in the object database.
3. Check the contents of the commit you created. *Repeat 1-3 to create a linear history.*
4. Check the commit history with `git log`.
5. *Bonus* Create a second commit off your initial commit. Create a merge commit from your two "branches."

References

1. Look inside your `.git/refs` directory. Check that there are two subdirectories.
2. Take a commit from part 3 and create the "master" branch.
3. Find the branch in the `.git/refs` directory.
4. Create a new commit and the check that master has not changed. Advance master to the newest commit.
5. Create a second branch. Add some more commits, and advance this branch to the newest commit.
6. Create a lightweight tag on the most recent commit.
7. Create an annotated tag on the same commit referred to by master.
8. *Bonus* Try to delete a branch using only file manipulation.