A Plumber's Guide to Git

Alex Chan • @alexwlchan

Image: *Model of the 'Optimus' water closet.*
Science Museum, Wellcome Images.

#1 The Git object database

1. Create a new folder, and initialise Git.
2. Look inside the .git folder. Do you understand what each file does?
3. Using a text editor, create a file and write some text.
4. Use a Git plumbing command to save your file to the Git database. Check you can see the file in .git/objects.
5. Use a Git plumbing command to inspect the file in the database.
6. Make an edit to your file, then save the new version to the Git database. What do you see in .git/objects? *Before you look: what do you expect to see?*
7. [Bonus] Delete the file, then recreate it from the Git object store.
8. [Bonus] What if you create two files with the same contents, but different filenames? What you see in .git/objects? *What do you expect to see?*

#1 The Git object database

- `mkdir <path>`
  create a folder
- `ls .git`
  list the contents of the .git folder
- `find .git/objects -type f`
  list the files in .git/objects
- `git init <path>`
  initialise Git in a folder
- `git hash-object -w <path>`
  add a file to the Git object store
- `git cat-file -p <hash>`
  show the contents of something in the Git object store (-p means "pretty")

#2 Blobs and trees

1. Take the file you created in exercise #1, and add it to the index.
2. Check that you can see an index file in your .git folder.
3. Use a plumbing command to check that you've added it to the index. Then use a porcelain git status to see the result.
4. Create a tree from the current index.
5. List all the files in .git/objects. Can you see the tree you just created?
6. Inspect the tree with a plumbing command. Check you understand what you're looking at.
7. [Bonus] Make an edit to your file, add the new version to the database, add the new object to the index. Create a new tree, and use plumbing to look at the tree.
8. [Bonus] Create another folder inside your main folder, then create a text file inside it. Add the new text file to the index. Now create a new tree, and look at its contents. *Before you look: what do you expect to see in the tree?*

#2 Blobs and trees

- `git update-index --add <path>`
  add a file to the Git index
- `git ls-files`
  list the files in the current index
- `git write-tree`
  create a new tree object; write the current index to a tree
- `find .git/objects -type f`
  list the files in .git/objects
- `git cat-file -p <hash>`
  show the contents of something in the Git object store

#3 Creating commits

1. Take one of the trees you created in exercise #2, and create a commit object.
2. List all the files in .git/objects. What do you see? Inspect the object that's just been created in the Git object store.
3. Make an edit to one of your files. Create a tree that contains the updated file. Create another commit from the new tree, with the original commit as its parent.
4. Repeat step 3 a couple of times. Create several commits that form a linear history – each one with the previous commit as its parent.
5. Use a porcelain git log to see the history you've just created.
6. [Bonus] Commits have associated messages. There's a porcelain command git commit --amend, which allows you to change the message associated with the latest commit (if Git knows which commit this is). *If you were to run this command, would it create a new commit object, or edit the old commit object?*
7. [Bonus] Can you replicate the effect of this porcelain command using the plumbing commands you already know?

#3 Creating commits

- `echo "my first commit" | git commit-tree <tree>`
  create a Git commit from a tree
- `git "another commit" | git commit-tree <tree> -p <commit>`
  create a Git commit from a tree, with another commit as its parent
- `git log <commit>`
  show all the parent commits/history leading up to this commit
- `git commit --amend -m "my new message"`
  change the message on a commit

#4 Refs and branches

1. Look in your .git/refs folder. Check it doesn't contain anything.
2. Take one of the commits you created in #3, and create a master branch. Use a porcelain git log to look at all the commits on this branch.
3. Have another look in the .git/refs folder. What do you see?
4. Make some more edits, add some more commits, and use git log to check that master hasn't changed. Advance master to your latest commit.
5. Create a second branch dev. Add some more commits, and advance dev to your latest commit. Check that master still points to your old commit.
6. Use a porcelain git branch to inspect the branches you've just created.

#4 Refs and branches

- `git update-ref refs/heads/<branch> <commit>`
  create a Git branch which points at the given commit
- `git log <branch>`
  show all the commits/history on this branch
- `git branch`
  list all the branches in a repository
- `git symbolic-ref HEAD refs/head/<branch>`
  set HEAD to point at a particular branch