1 The Git object store

Exercises

1. Create a new folder, and initialise Git.
2. Look inside the `.git` folder.
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 new object in `.git/objects`.
5. Use a Git plumbing command to inspect the object 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?

Bonus exercises

7. Delete the file, then recreate it from the Git object store.
8. What if you create two files with the same contents, but different filenames? What do you see in `.git/objects`?

   What do you expect to see?

Useful commands

`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 <path>`	show the contents of something in the Git object store

2 Blobs and trees

Exercises

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. Use a Git plumbing command to inspect the tree. Make sure you understand what you're looking at.

Bonus exercises

7. Make an edit to your file, add the new version to the Git database, and add the new Git object to the index. Create a new tree, and use a plumbing command to look at the tree.
8. Create another folder inside your main folder, then create a text file inside that folder. Add the new text file to the index. Now create a new tree, and look at the contents of the tree.

   Before you look: talk to your neighbour about what you expect to see in the tree.

Useful commands

`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; write the current index to a tree

3 Creating commits

Exercises

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 has the previous commit as its parent.
5. Use a porcelain `git log` to see the history you've just created.

Bonus exercises

1. A *merge commit* is a commit with more than one parent. How do you think you might create it? Try it.

Useful commands

```
echo "my first commit" | git commit-tree <tree>
```

create a Git commit from a tree

```
echo "next commit" | git commit-tree <tree> -p <commit>
```

create a Git commit from a tree, with a previous commit as its parent

```
git log <commit>
```

show all the parent commits leading up to this commit

4 Refs and branches

Exercises

1. Look in your `.git/refs` folder. Check it's empty
2. Take one of the commits you created in exercise #2, and create a *master* branch. Use a porcelain `git log` to look at all the commits on this branch.
3. Have another look inside the `.git/refs` folder.
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 *development*. Add some more commits, and advance development to your latest commit. Check that *master* is still pointing to your old commit.
6. Use a porcelain `git branch` to inspect your new branches.

Bonus exercises

7. If you created a branch, new commits aren't automatically added. Update HEAD to point to one of your branches. Start creating some new commits, and use `git log` to check that those new commits are being added to the branch.

Useful commands

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