Git Exercises (1)

Git Documentation

Git comes with *extensive* documentation. Run:

```
apropos git
```

To see all of it, or run:

```
apropos git -a tutorial
```

To find documentation relating to git and (the `-a`) tutorials. Read any of the documentation you think might be useful with the `man` command.

Task: There is a man page that documents the *everyday git* commands you might want to use. Find it with `apropos` and read it with the `man` command.

You might also want to read the `gittutorial` pages...

Configuring your identity

Git is all about tracking changes to source code. That means it needs to know *who* made *what* changes.

Run the following two lines to set up git correctly. You only need to do this once when you install git, but not every time you create a new repository.

```
git config --global user.name "YOURNAME"
git config --global user.email "YOUREMAIL"
```

The name and email address aren't actually sent anywhere or checked... they're just listed with alongside the changes you make to the code so if somethings wrong later programmers know who to blame (see `man git-blame`). You can put anything you like here (git will happily accept `-` as your email address, and it does not send you email).

This alters the *global* git configuration (the settings applied to *every* git repository you work on), but you can also make these changes on a repository by repository basis. Just drop the `--global` and run the command inside the git repository you want to apply the changes to. This is useful if you're *Bruce Wayne* and need to keep your public and private development projects separate (or if you do subcontracted development work).

A sample project and repository

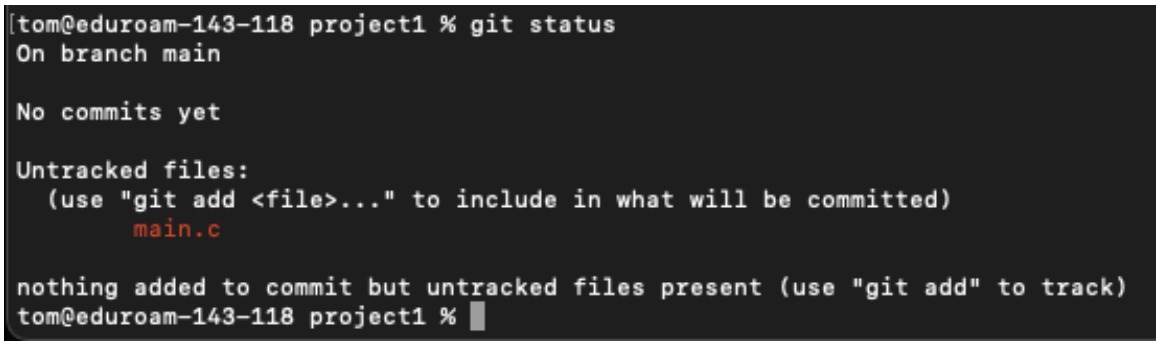
Let's say you want to start developing a C program. Let's make a folder:

```
mkdir project1
cd project1
git init
```

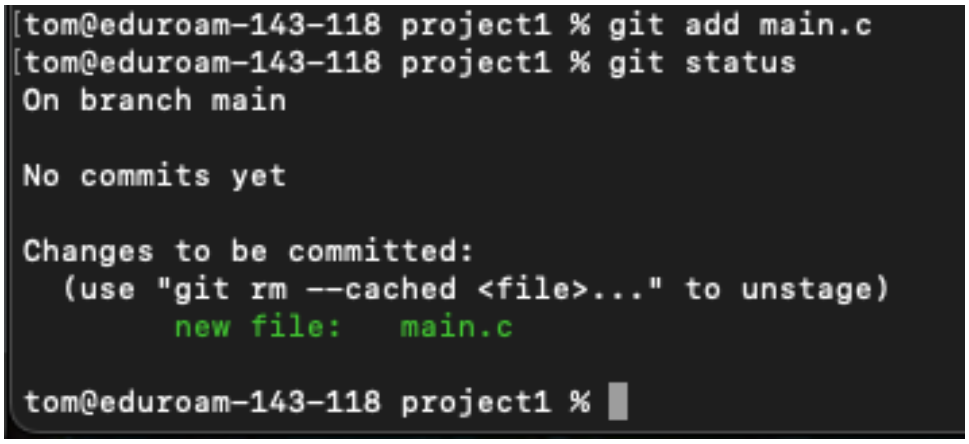
The last command created an empty git repository in a subfolder called `.git`. We can check with `git status` to see whether there are any changes, and git reports `nothing to commit`.

Create a file `main.c`, with your text editor and add some sample content like this (you should be able to copy-paste into your terminal):

Do a `git status` and you will see `main.c` in red under *untracked files* - this is a new file that git does not know about yet.



Do `git add main.c` followed by another `git status` and the file is now green under *files to be committed*.



Commit the file with `git commit -m "first file"` or something like that - you need double quotes if you want spaces in your commit message. Try `git status` again and you should see *nothing to commit, working tree clean* which means git is up to date with your files. Try `git log` and you will see that there is now one commit in the log.

- without the `-m` flag you will be put into a default text editor (usually vim)

```
tom@eduroam-143-118 project1 % git commit -m "first commiit"
[main (root-commit) 56d77af] first commiit
 1 file changed, 8 insertions(+)
 create mode 100644 main.c
tom@eduroam-143-118 project1 % git status
On branch main
nothing to commit, working tree clean
tom@eduroam-143-118 project1 %
```

Ignoring files

Compile your code with `gcc main.c -o program`, and check with `./program` that it runs and prints *Hi*. (If you get an error that `stdio.h` doesn't exist, then you have installed gcc but not the C development libraries Hint: `man apt-file`.)

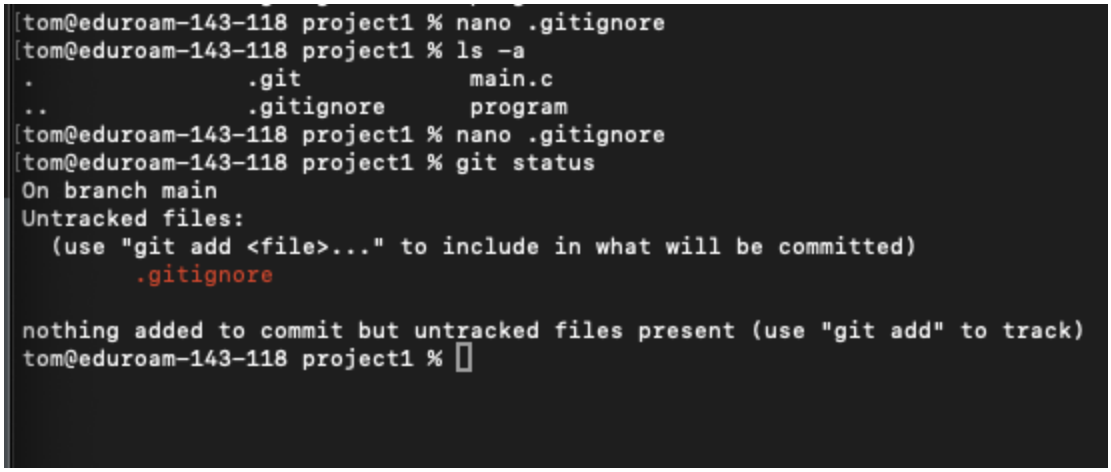
If you look at `git status` now, the program file shows as untracked, but we do not want to commit it: the repository works best when you store only your source code, and anyone who needs to can check out a copy and build from there. Among other things this means that people on different platforms e.g. linux and mac, intel and ARM and so on can each compile the version that works for them.

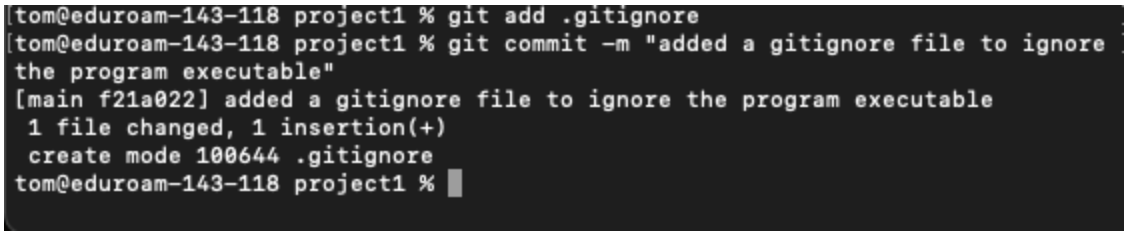
```
tom@eduroam-143-118 project1 % git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        program

nothing added to commit but untracked files present (use "git add" to track)
tom@eduroam-143-118 project1 %
```

So we want to tell git to ignore the program and changes in it, which we do by creating a file called `.gitignore` and adding an expression on each line to say which file(s) or folders to ignore - you can use `*.o` to select all object code files, for example.

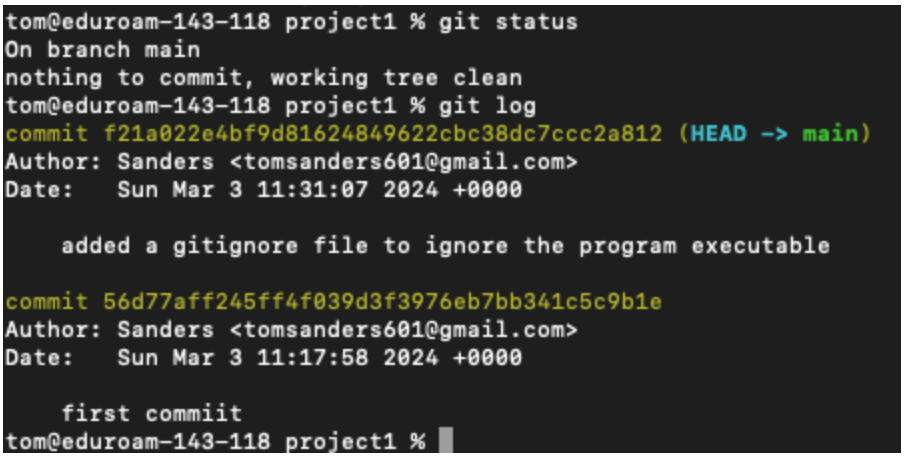
- Create a file `.gitignore` and add the single line `program` to it.

- Do another `git status` and notice that while the program is now ignored, the ignore file is marked as new. This file does belong in the repository, so add it and commit it.



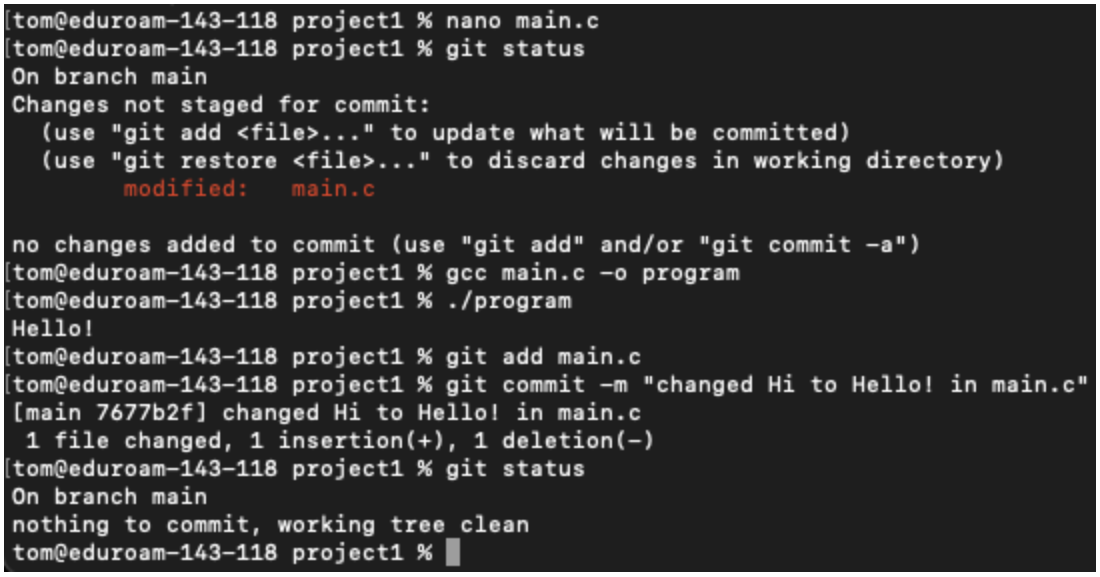


- Check that `git status` reports *clean* again, and that `git log` contains two commits.



Commit and checkout

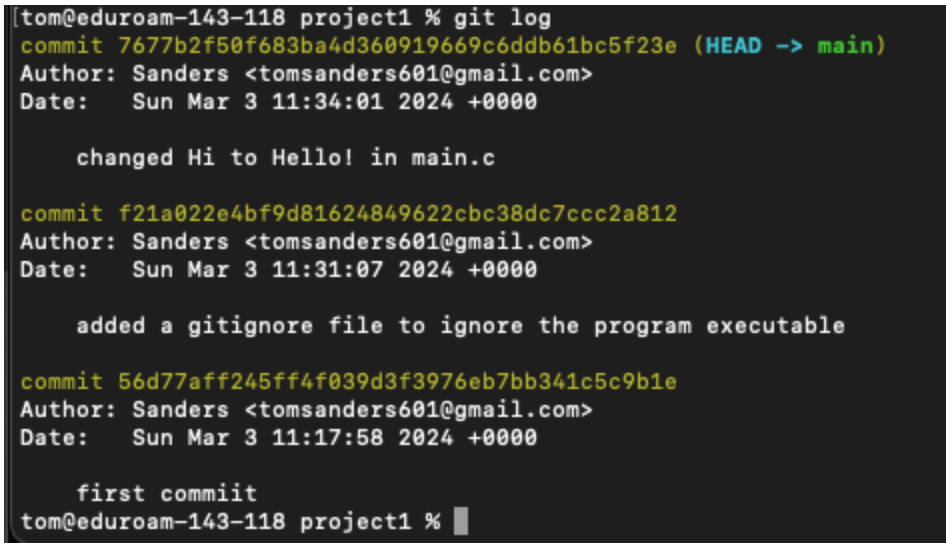
As you develop, you should regularly code, commit, repeat. To practice this, change *Hi* to *Hello* in the program, rebuild and run the program, then add and commit the source file again - check with `git status` at the end that you get *clean* again:



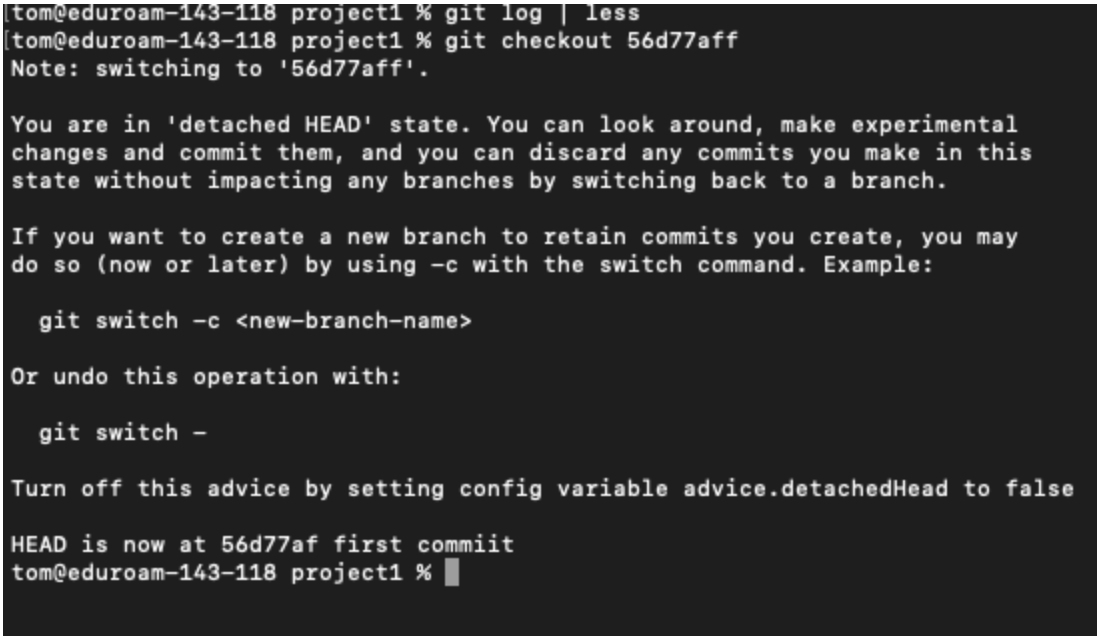
The command `git add .` adds all new and changed files and folders in the current folder in one go, and is typically the quickest way to add things when you want to commit all your changes since the last commit.

Sometimes you want to go back and look at another commit, or undo a commit that broke something - this is when you want a checkout.

- Use `git log` to show the history of your commits. (When you have more than one screen, `git log |less` lets you scroll.)



- Note the first 6 or so characters of the commit hash of the commit where you added the ignore file, but before changing *Hi* to *Hello*. You need at least 6 characters, but only as many so that it's not ambiguous to git which commit you mean.
- Run `git checkout HASH` where HASH is the 6 or however many you need characters of the commit in question. Git will print a warning about the HEAD pointer.



- Check the source file, and notice that it is now back on *Hi*.

- Use `git checkout main` to return to the latest version of your files, and git will set up the HEAD pointer again ready to accept new commits.