

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 something's 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.

```

[tom@eduroam-143-118 project1 % git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        main.c

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

```

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

```

[tom@eduroam-143-118 project1 % git add main.c
[tom@eduroam-143-118 project1 % git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   main.c

tom@eduroam-143-118 project1 %

```

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.

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

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

```
tom@eduroam-143-118 project1 % git add .gitignore
tom@eduroam-143-118 project1 % git commit -m "added a gitignore file to ignore
the program executable"
[main f21a022] added a gitignore file to ignore the program executable
1 file changed, 1 insertion(+)
create mode 100644 .gitignore
tom@eduroam-143-118 project1 %
```

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

```
tom@eduroam-143-118 project1 % git status
On branch main
nothing to commit, working tree clean
tom@eduroam-143-118 project1 % git log
commit f21a022e4bf9d81624849622cbc38dc7ccc2a812 (HEAD -> main)
Author: Sanders <tomsanders601@gmail.com>
Date:   Sun Mar 3 11:31:07 2024 +0000

    added a gitignore file to ignore the program executable

commit 56d77aff245ff4f039d3f3976eb7bb341c5c9b1e
Author: Sanders <tomsanders601@gmail.com>
Date:   Sun Mar 3 11:17:58 2024 +0000

    first commiit
tom@eduroam-143-118 project1 %
```

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:

```
tom@eduroam-143-118 project1 % nano main.c
tom@eduroam-143-118 project1 % git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.c

no changes added to commit (use "git add" and/or "git commit -a")
tom@eduroam-143-118 project1 % gcc main.c -o program
tom@eduroam-143-118 project1 % ./program
Hello!
tom@eduroam-143-118 project1 % git add main.c
tom@eduroam-143-118 project1 % git commit -m "changed Hi to Hello! in main.c"
[main 7677b2f] changed Hi to Hello! in main.c
 1 file changed, 1 insertion(+), 1 deletion(-)
tom@eduroam-143-118 project1 % git status
On branch main
nothing to commit, working tree clean
tom@eduroam-143-118 project1 %
```

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.)

```

tom@eduroam-143-118 project1 % git log
commit 7677b2f50f683ba4d360919669c6ddb61bc5f23e (HEAD -> main)
Author: Sanders <tomsanders601@gmail.com>
Date:   Sun Mar 3 11:34:01 2024 +0000

    changed Hi to Hello! in main.c

commit f21a022e4bf9d81624849622cbc38dc7ccc2a812
Author: Sanders <tomsanders601@gmail.com>
Date:   Sun Mar 3 11:31:07 2024 +0000

    added a gitignore file to ignore the program executable

commit 56d77aff245ff4f039d3f3976eb7bb341c5c9b1e
Author: Sanders <tomsanders601@gmail.com>
Date:   Sun Mar 3 11:17:58 2024 +0000

    first commiit
tom@eduroam-143-118 project1 % █

```

- 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.

```

tom@eduroam-143-118 project1 % git log | less
tom@eduroam-143-118 project1 % git checkout 56d77aff
Note: switching to '56d77aff'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 56d77af first commiit
tom@eduroam-143-118 project1 % █

```

- 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.

```
tom@eduroam-143-118 project1 % git checkout main
Previous HEAD position was 56d77af first commiit
Switched to branch 'main'
tom@eduroam-143-118 project1 % git status
On branch main
nothing to commit, working tree clean
tom@eduroam-143-118 project1 % █
```