Collaboration with Git

Patrick McCann

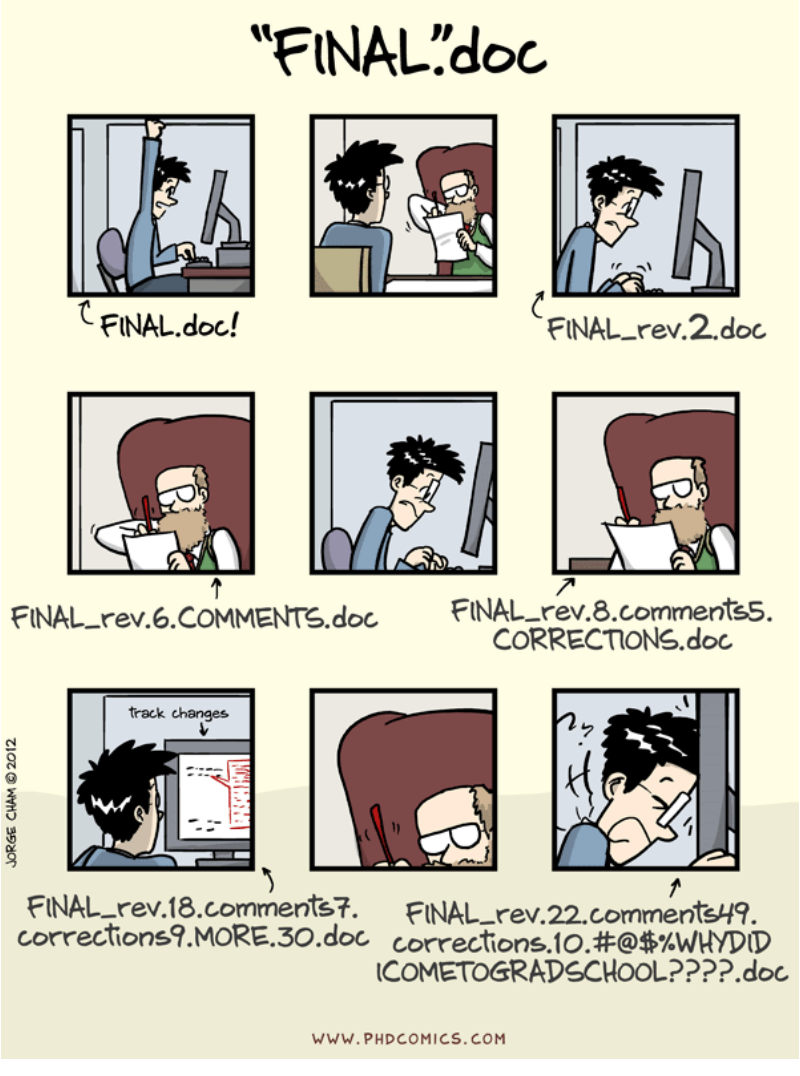
Research Computing, University of St Andrews

Software Carpentry

Much of this presentation will draw on the Software Carpentry lesson Version Control with Git.

That lesson goes into topics in more detail, with examples and exercises. There are regular Software Carpentry workshops at the University, open to all researchers.

Why use Version Control?



Version control systems record the details of changes made to a base version of a document or documents.

As well as allowing you to track changes over time - and to move back and forward between versions - they allow collaborators to maintain differing versions and provide mechanisms for resolving those differences.

You get to decide what changes get grouped together in a *commit*, marking a new version.

A project's commit history and metadata make up a *repository*. Repositories can be kept in sync between different computers.

Why use Git?

git

Version control systems have been around since the 1980s - you may have heard of e.g. CVS or Subversion.

Modern systems like Git and Mercurial are *distributed*, so they don't need a central server to host repositories.

Git has become the de facto standard.

Why use GitHub?

GitHub

GitHub.com has become the most popular platform for hosting Git Repositories - especially public repositories for open-source software.

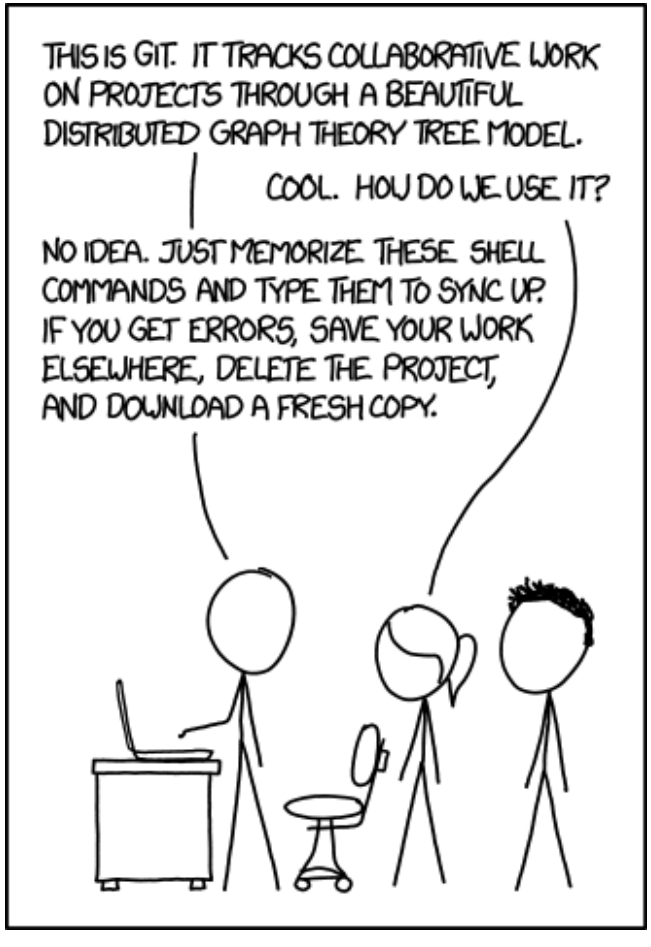
Others platforms include BitBucket and GitLab.com.

The University has an instance of GitLab (VPN) available to researchers - this is not available to users outside the University.

GitLab

GitHub also provides some additional features which are particularly useful to academic researchers.

Using Git



https://xkcd.com/1597/

There are a number of ways to work with Git on your computer:

- The command-line interface referenced in the XKCD cartoon
- Terminal based user interfaces like gitui and lazygit
- Graphical user interfaces like GitHub Desktop and SourceTree
- As a feature (or plugin) of development tools and editors like RStudio.

Other user interfaces can be considered as wrappers around the command-line interface. They tend to hide some of the details of how Git works.

This presentation includes screenshots from GitHub Desktop alongside the equivalent commands.

A Git Repository

- my-project
  - .git
  - data
  - src
  - test
  - .gitignore
  - LICENSE.txt
  - README.md
  - run.sh

- my-project
  - **.git**
  - data
  - src
  - test
  - .gitignore
  - LICENSE.txt
  - README.md
  - run.sh

.git is where Git stores the metadata about the project.

We (almost) never edit its contents directly.

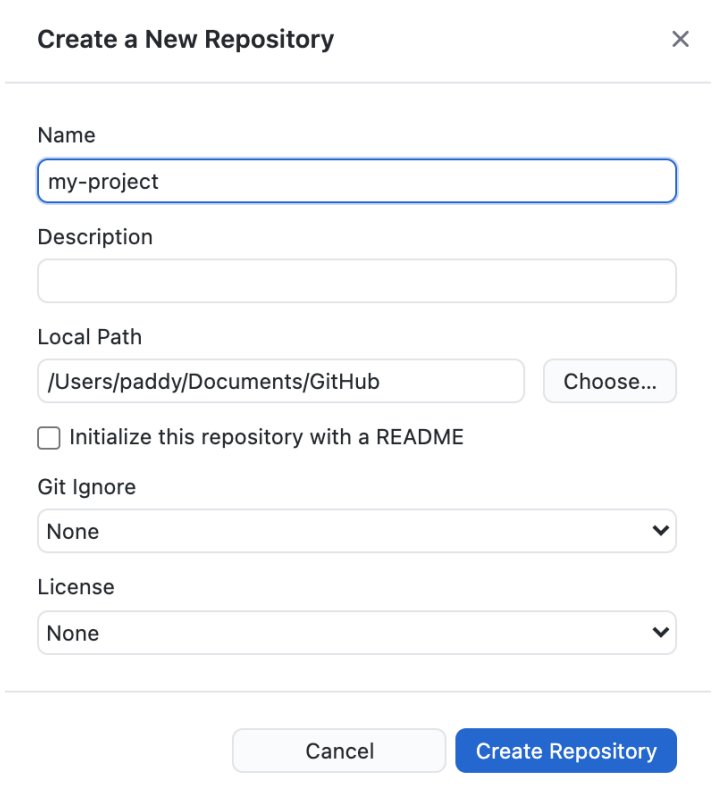
Technically, **.git** is the *Git Repository*, but you will often see **my-project** described as such.

- my-project
  - .git
  - data
  - src
  - test
  - **.gitignore**
  - LICENSE.txt
  - README.md
  - run.sh

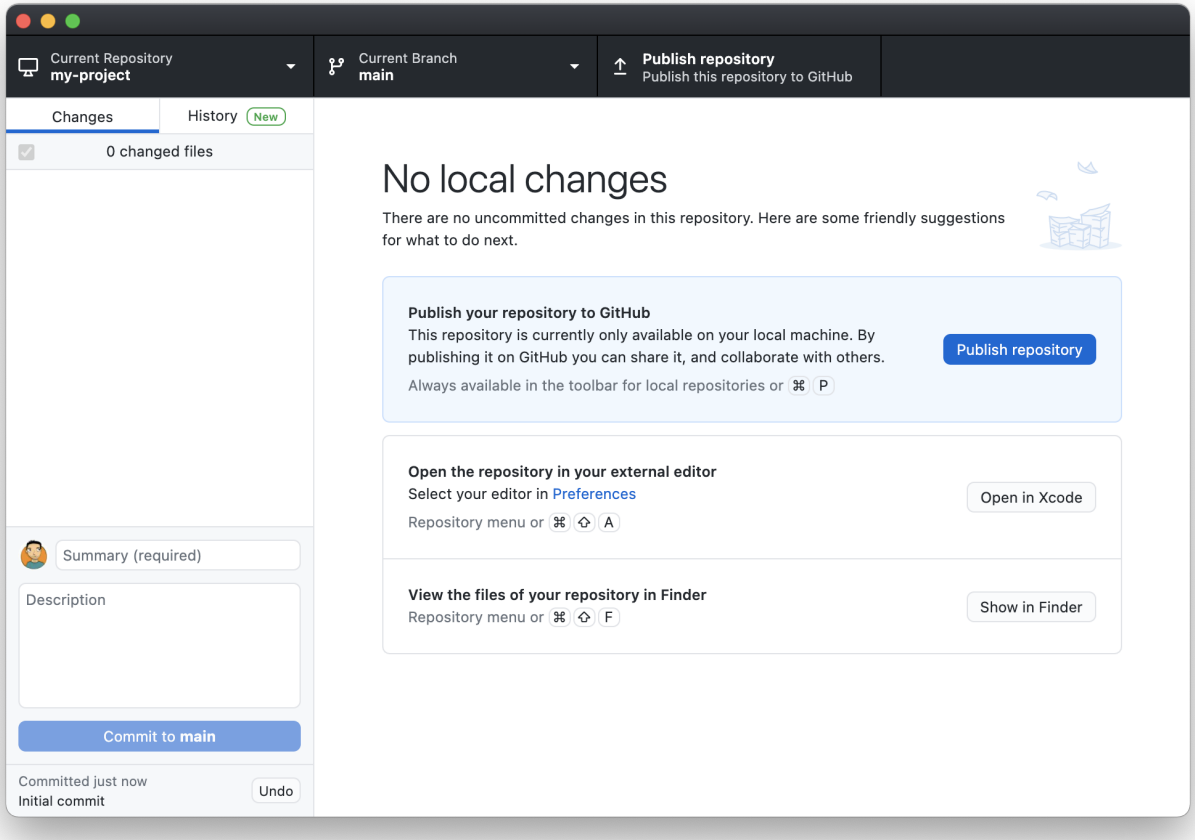
Many Git-managed projects will have a **.gitignore** file, which allows us to exclude some files from version control.

Initialising

```
$ cd /Users/paddy/Documents/GitHub
$ mkdir my-project
$ cd my-project
$ git init
```



```
$ ls -a
.        ..       .git
$ git status
On branch main
nothing to commit, working tree clean
```

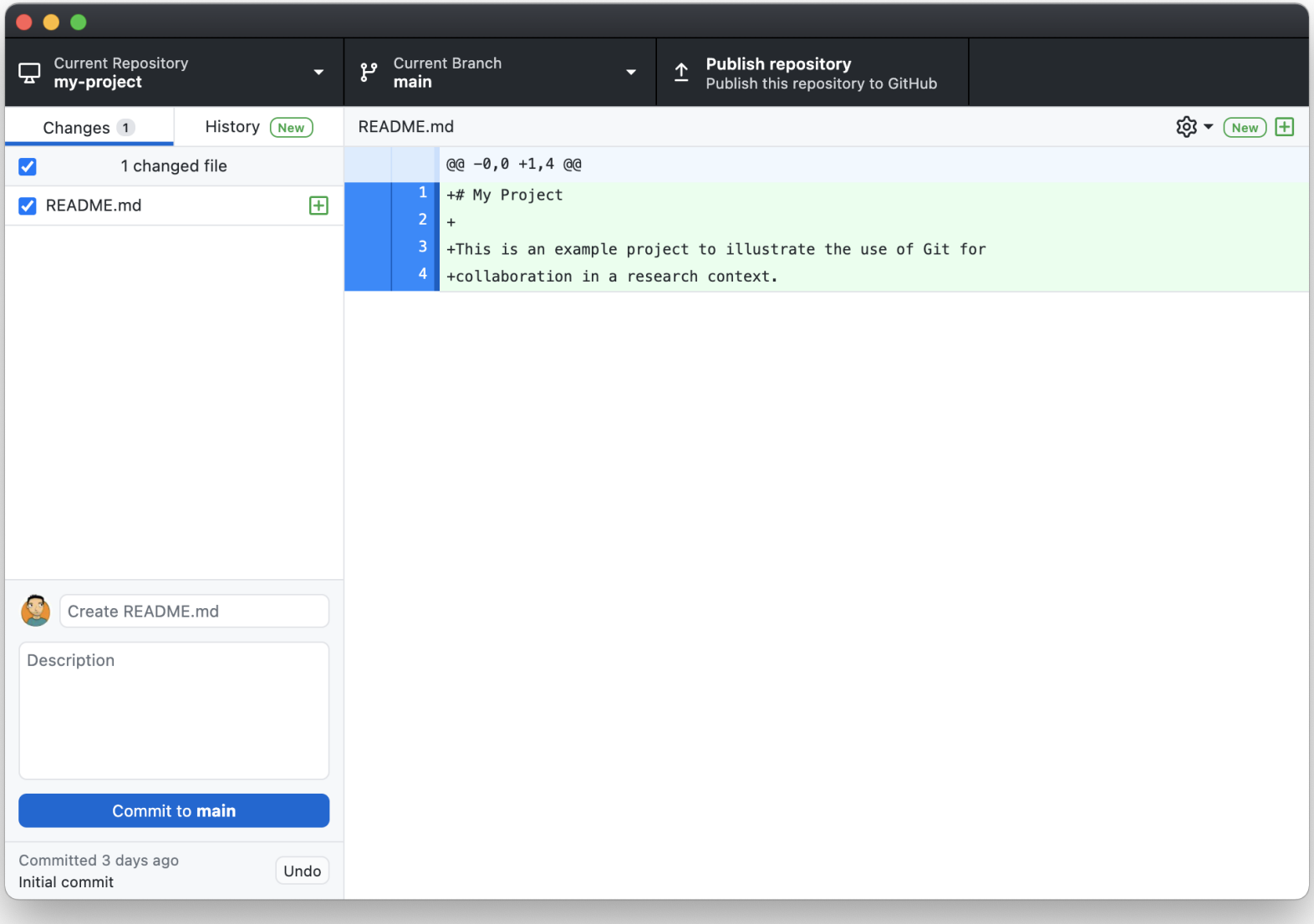


Adding and Committing

We can add a file to our project folder using any text editor or, indeed, any piece of software which allows us to save files.

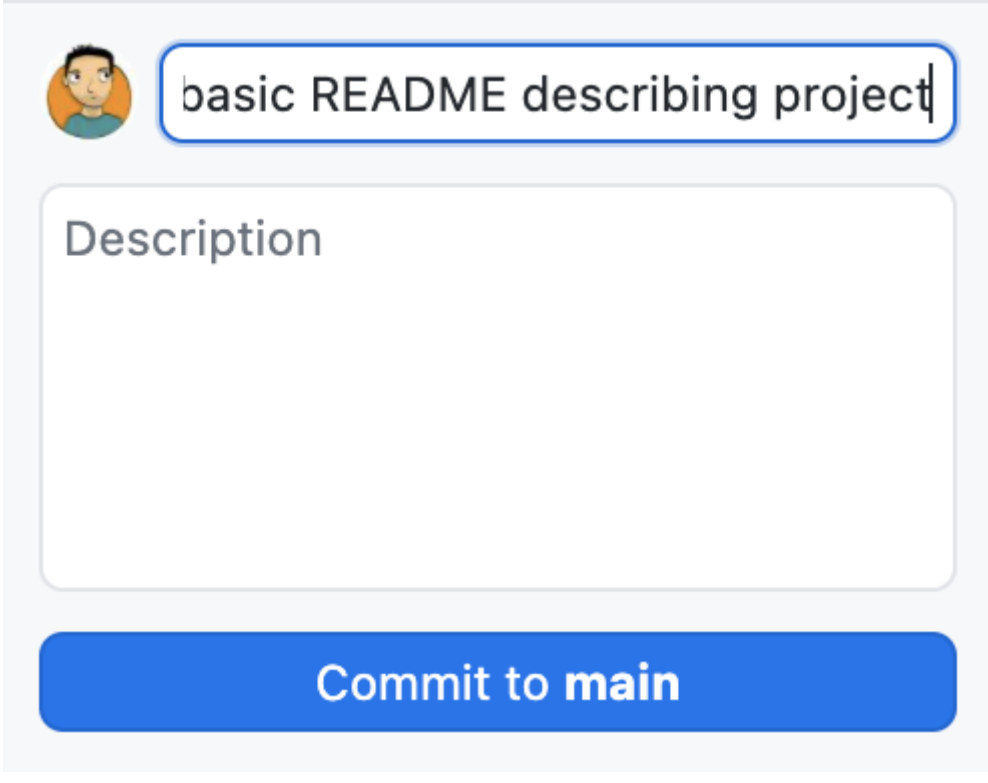
Here, we use a text editor to create a README file using markdown syntax and save it as `README.md`.

```
$ ls -a
.        ..       .git     README.md
$ cat README.md
# My Project

This is an example project to illustrate the use of Git for
collaboration in a research context.
```


Current Repository
my-project
Current Branch
main
Publish repository
Publish this repository to GitHub
Changes 1
History New
1 changed file
README.md
Create README.md
Description
Commit to main
Committed 3 days ago
Initial commit
Undo
README.md
New
@@ -0,0 +1,4 @@
+# My Project
+
+This is an example project to illustrate the use of Git for
+collaboration in a research context.

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add
$ git add README.md
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   README.md
```

```
$ git commit -m 'Adds basic README describing project'
```



Viewing the Log

Ignoring Things

Remotes - Pushing and Pulling

Git in RStudio

Collaboration

Branching

Forking

Automation

Open Science

Licensing

Citation

Zenodo and Pure