Version Control Walkthrough with git

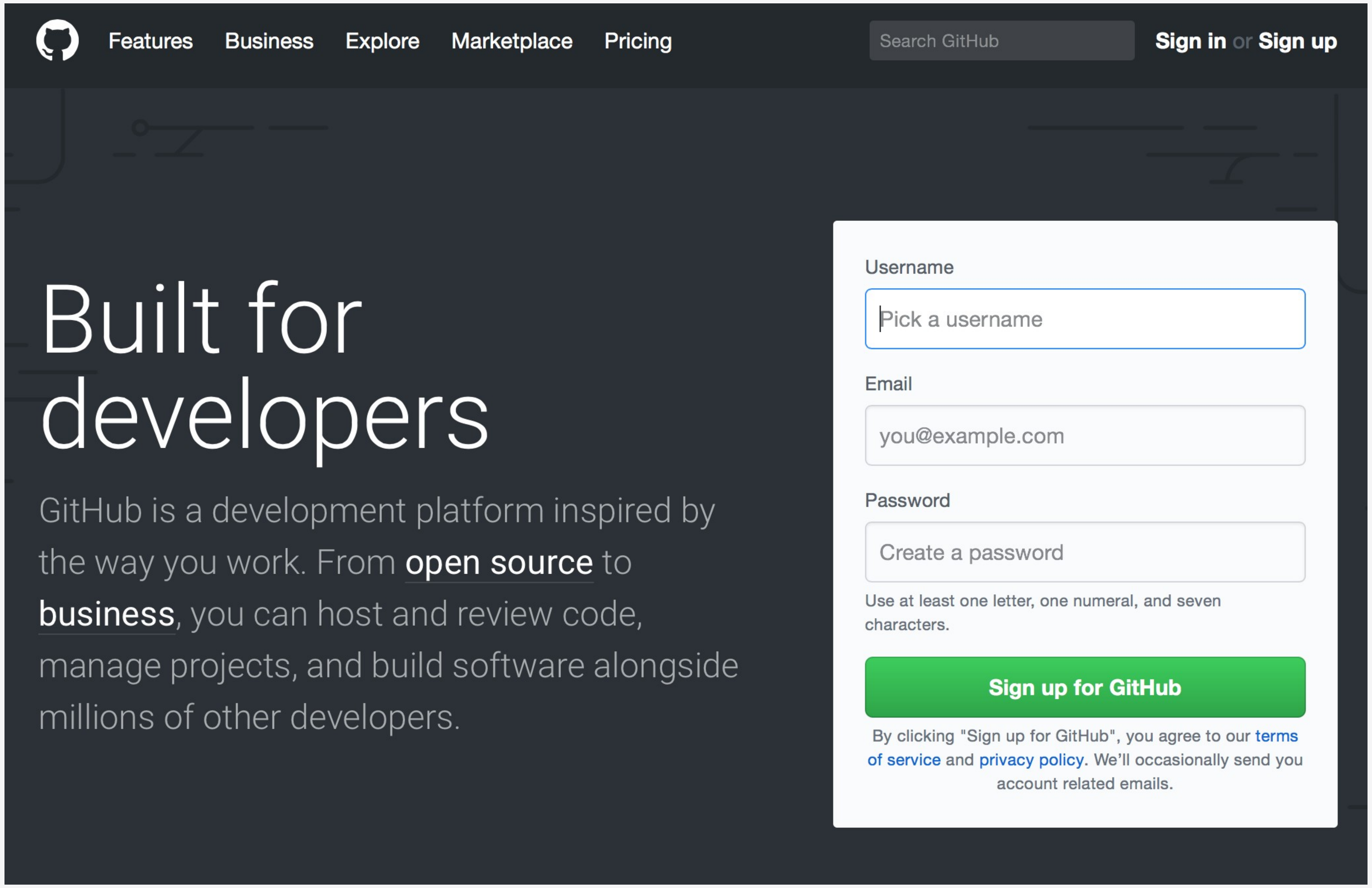
Mark Slater based on slides from Ben Morgan

THE UNIVERSITY OF WARWICK

UNIVERSITY OF BIRMINGHAM

A Version Control Walkthrough with Git

- We haven’t actually written anything yet, and this is the perfect time to get mpags-cipher under version control so you can use it through the whole course. We’ll be using **git** as our version control system, with **github.com** acting as a central repository.
- This will allow us to to use the **‘Github Classrooms’** feature which will mean you will have a clean repo to start each day from. You can then apply your changes to this and we can add comments, changes, etc. directly in github.
- Aims of the walkthough:
  - *Create a repo from the Github Assignment, get a working copy into Visual Studio Code, add files, commit changes and make tags*
  - *Show diffs and logs for the commits we’ve made*
  - *Push our local changes to the central repo, Pull changes from another repo*



Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside millions of other developers.

Username

Pick a username

Email

you@example.com

Password

Create a password

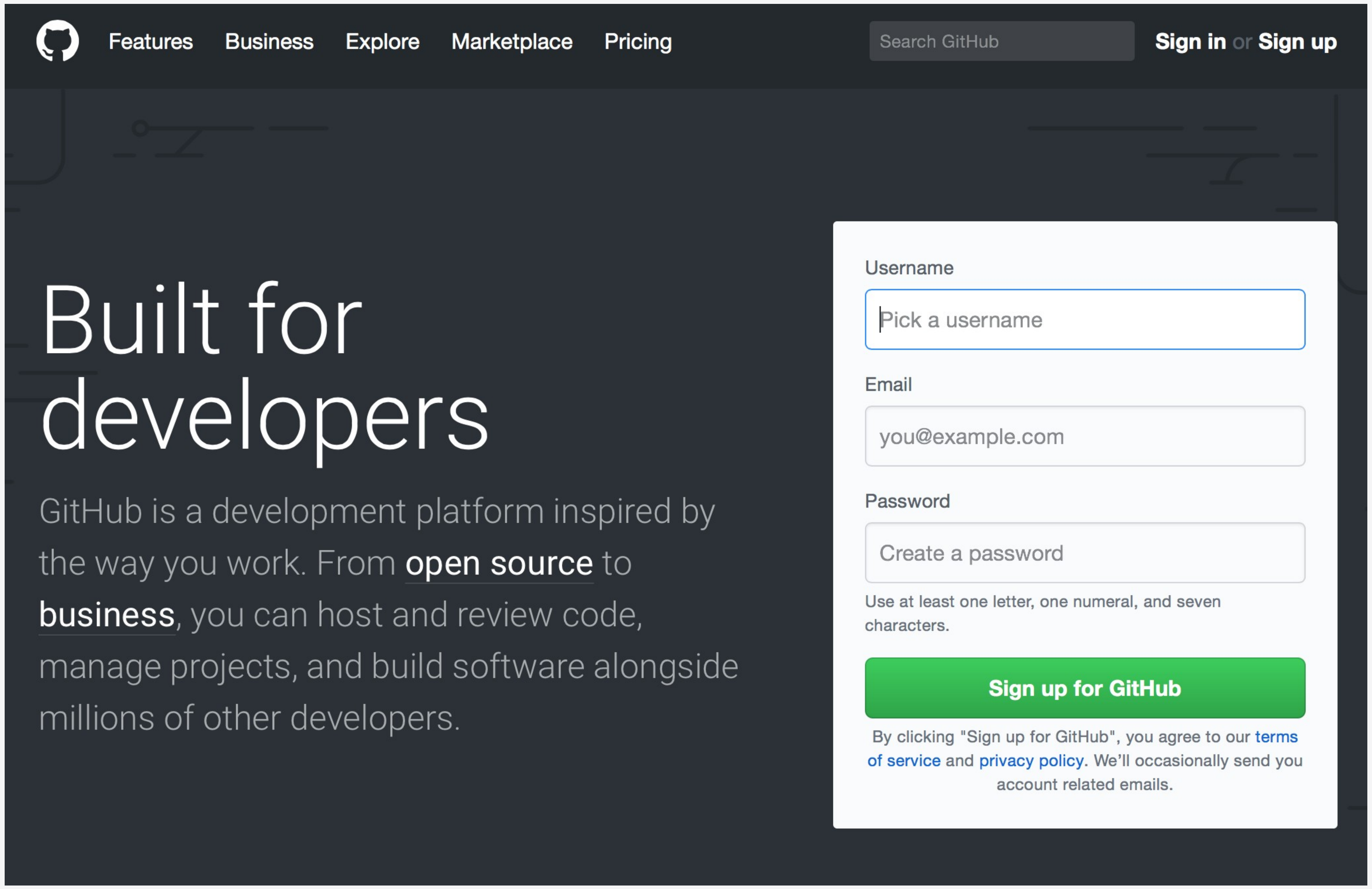
Use at least one letter, one numeral, and seven characters.

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We'll occasionally send you account related emails.

Tools you'll need

1: Creating a Github Account

Whilst git is completely distributed, to help with working in several locations and to supply solutions, you'll create a repository to be an authoritative one. This will be created for you when you go to the link for each assignment on each day. However, before you can do anything in Github, you need to sign up for an account.



Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside millions of other developers.

Username: Pick a username

Email: you@example.com

Password: Create a password

Use at least one letter, one numeral, and seven characters.

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We'll occasionally send you account related emails.

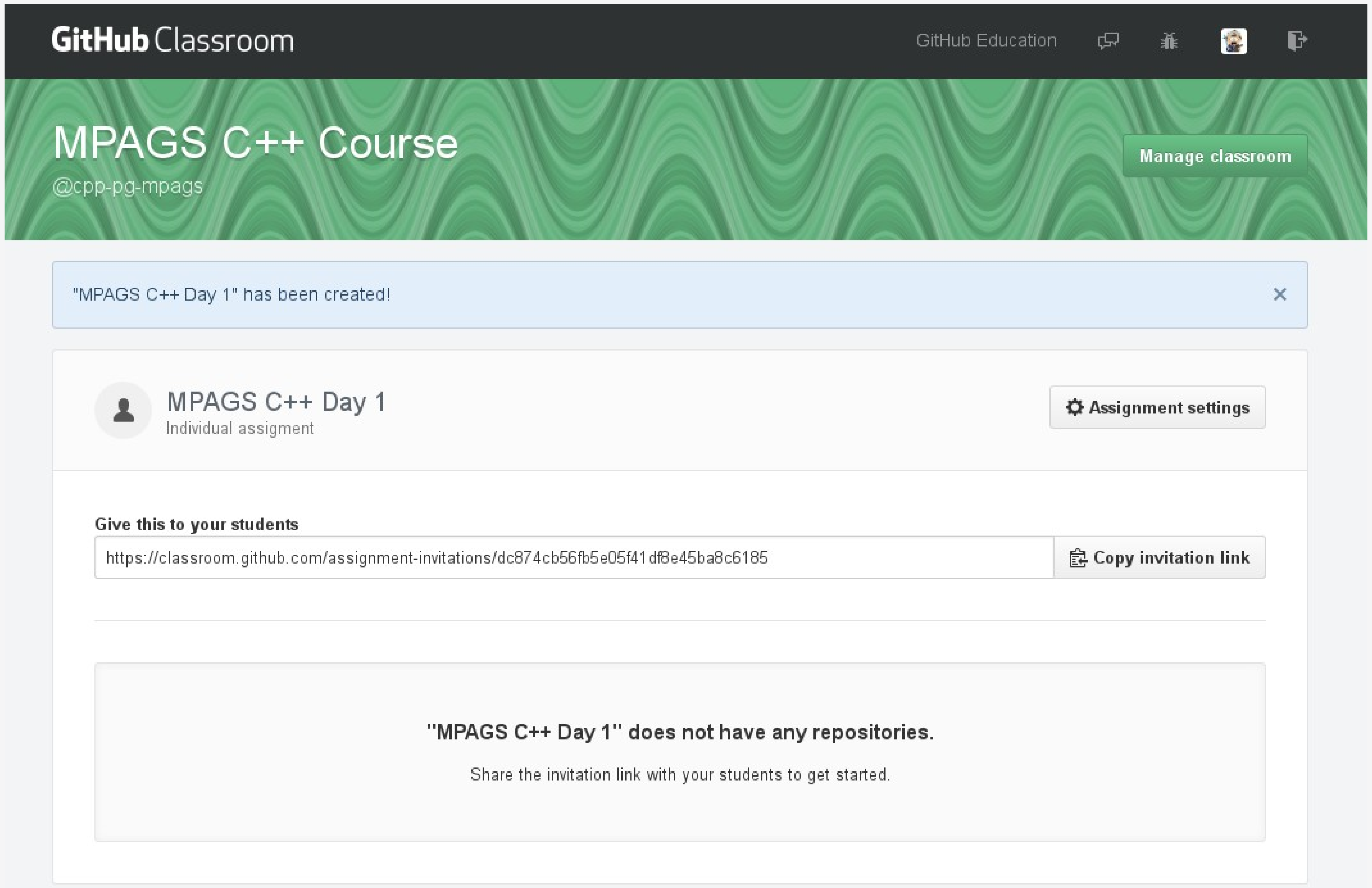
Notes

Other hosting services exist for git, though github is currently the most popular.

Similar hosting services exist for other VCSs.

2: Creating Your mpags-cipher Repository

You can create repositories in github in a number of ways: Creating one from scratch on Github, pushing an already existing local repo, forking an existing repo on Github, etc. However, for this course, the base repo that you'll be working from each day will be created for you from a template repo created by us. This will also include the docker container definition that will provide all the tools we'll need. To create the repo for Day one, go to the link on the course Day 1 page.



"MPAGS C++ Day 1" has been created!

MPAGS C++ Day 1
Individual assigment

Give this to your students
https://classroom.github.com/assignment-invitations/dc874cb56fb5e05f41df8e45ba8c6185

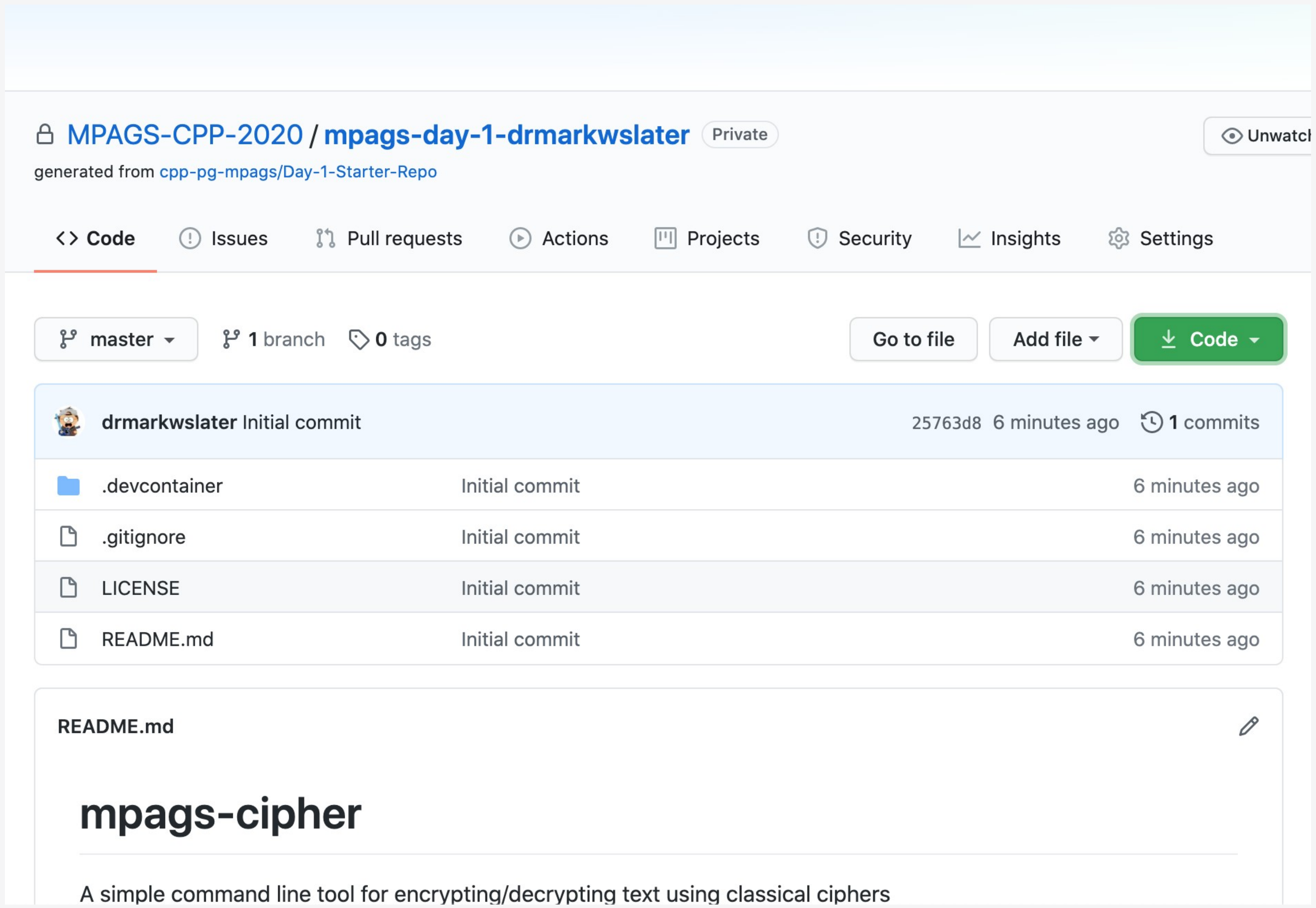
"MPAGS C++ Day 1" does not have any repositories.
Share the invitation link with your students to get started.

Notes

Though you will be creating a new repository for each day of the course, the previous repos will be kept until at least the end of the course for you to refer back to.

3: Your mpags-cipher Repository

After accepting the invitation to the classroom assignment, you should be sent a link to the repository you'll use for today (there will be new links for the other days!). This will be stored in the MPAGS-CPP group but you're welcome to 'fork' it to your own account at a later date if you want!


MPAGS-CPP-2020 / mpags-day-1-drmarkwslater Private

generated from cpp-pg-mpags/Day-1-Starter-Repo

drmarkwslater Initial commit 25763d8 6 minutes ago 1 commits

.devcontainer	Initial commit	6 minutes ago
.gitignore	Initial commit	6 minutes ago
LICENSE	Initial commit	6 minutes ago
README.md	Initial commit	6 minutes ago

README.md

mpags-cipher

A simple command line tool for encrypting/decrypting text using classical ciphers

Notes

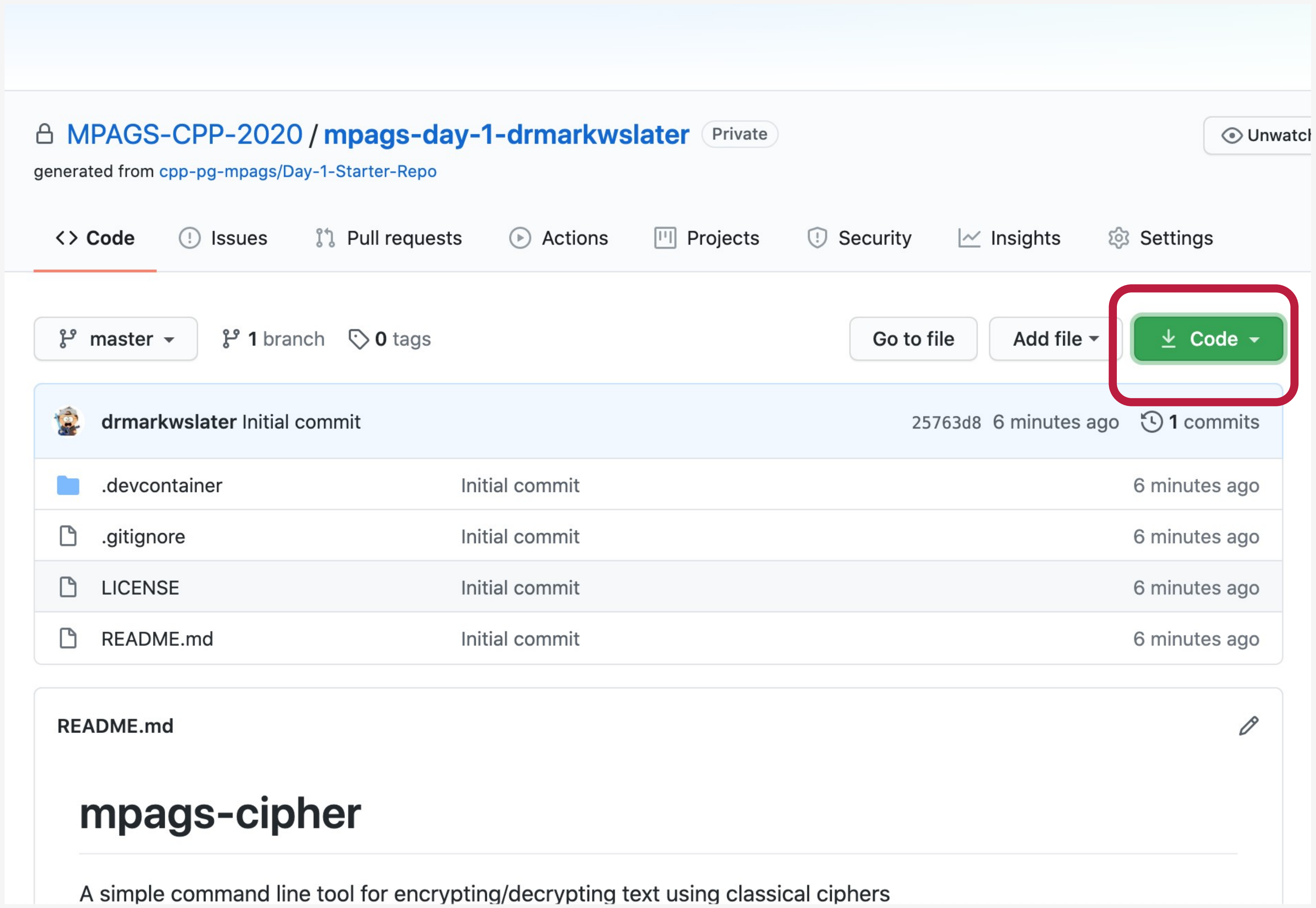
Though you don't need to for this course, you can create repos from scratch within github or from the command line (using `git init`) and then push to github.

4: Accessing Your Repo with VS Code and Docker

To access your repo through the container, start Visual Studio Code and select 'View → Command Palette'. In the dialog that pops up, start typing 'clone repository' and then select 'Clone Repository in Container Volume'. In the next box, paste the URL of your repo and then select 'Create Unique Volume'.

Note that if you just want to clone a repository from the command line, use:

```
$ git clone <repourl> <localdir>
```



MPAGS-CPP-2020 / mpags-day-1-drmarkwslater Private

generated from cpp-pg-mpags/Day-1-Starter-Repo

drmarkwslater Initial commit 25763d8 6 minutes ago 1 commits

.devcontainer	Initial commit	6 minutes ago
.gitignore	Initial commit	6 minutes ago
LICENSE	Initial commit	6 minutes ago
README.md	Initial commit	6 minutes ago

README.md

mpags-cipher

A simple command line tool for encrypting/decrypting text using classical ciphers

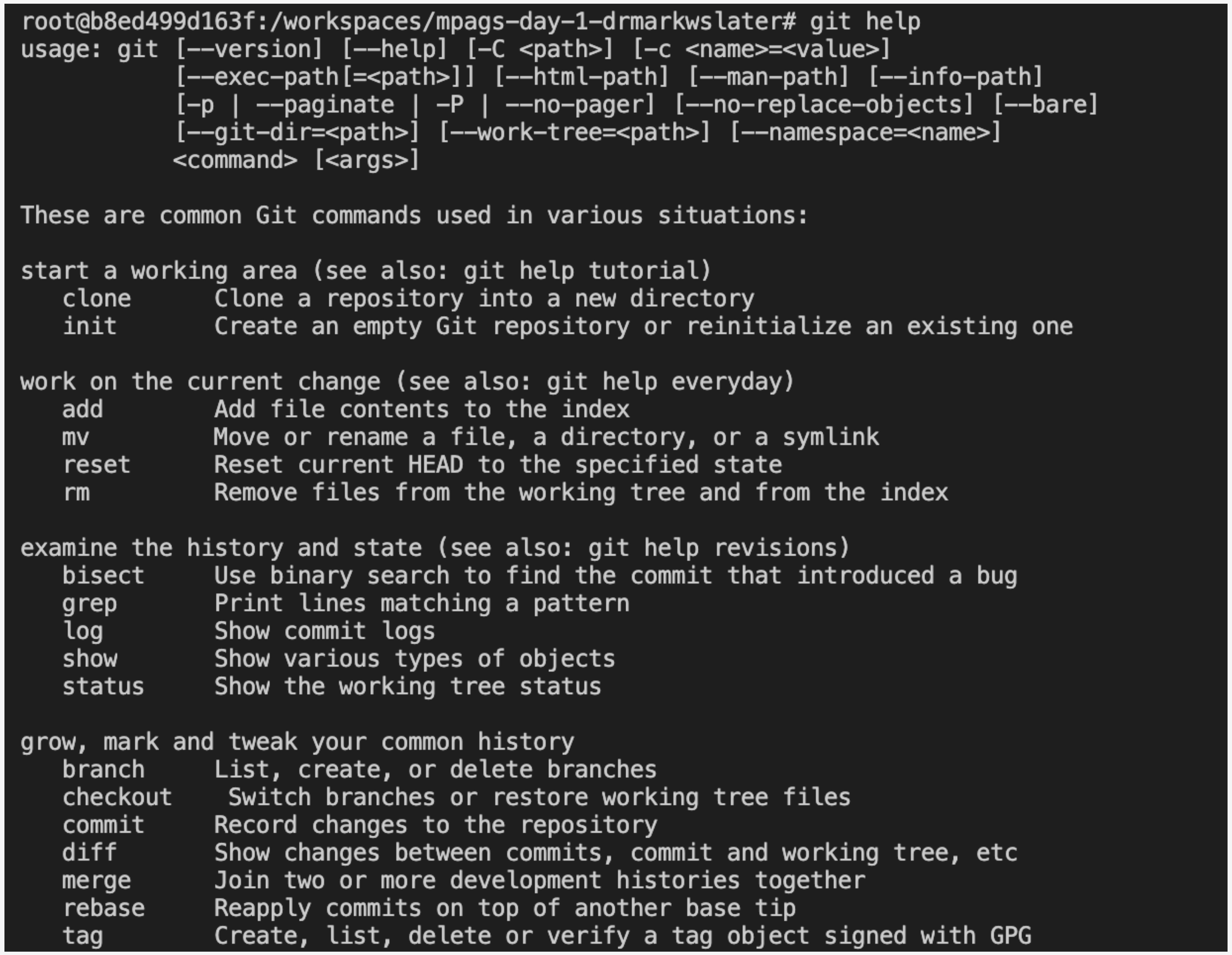
Notes

We'll use the **https** protocol to clone the repository as we haven't setup SSH keys yet and github has stopped supporting username+password authentication. The URL for cloning your GitHub repo can be found on the right hand side of its front page, as shown on the left. Click on the 'https' tab and copy the URL there.

5: Command Line Git: Getting Help

With the repo in place and the container started, we can now start working with git. In VSC, you can access the Terminal of the container by going to 'View → Terminal'. This will operate just as a normal Linux terminal – try doing the following to get help on git:

```
$ git help
```



```
root@b8ed499d163f:/workspaces/mpags-day-1-drmarkwslater# git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout    Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG
```

Notes

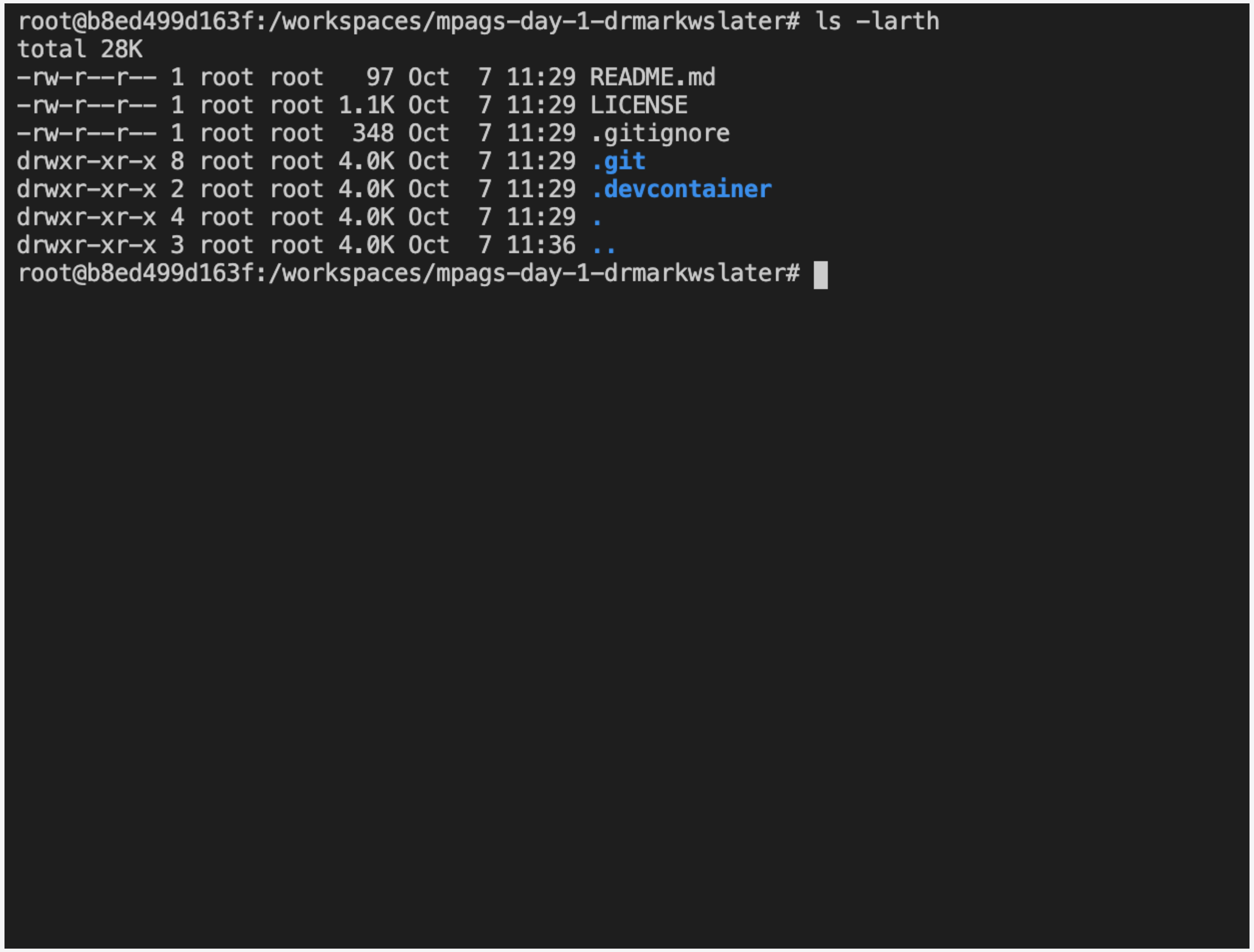
As the output of git help notes, to get help on a specific command, simply append its name to git help. It will open a man style page for the command (simply use 'q' to exit this).

Of course, also refer to text books and online resources such as git-scm.com.

6: Repository Structure

If not there already, change into your repository's directory and run `ls -larth` to see all the files. Apart from the files you'll have seen on the original GitHub site, there's an extra directory, `.git.` This is git's "database" holding the complete history of changes plus configuration information. It's this holding of the complete project history that allows the distributed version control

Note that in the images I have run: `alias ls='ls --color'` so directories are in blue



```
root@b8ed499d163f:/workspaces/mpags-day-1-drmarkwslater# ls -larth
total 28K
-rw-r--r-- 1 root root   97 Oct  7 11:29 README.md
-rw-r--r-- 1 root root 1.1K Oct  7 11:29 LICENSE
-rw-r--r-- 1 root root  348 Oct  7 11:29 .gitignore
drwxr-xr-x 8 root root 4.0K Oct  7 11:29 .git
drwxr-xr-x 2 root root 4.0K Oct  7 11:29 .devcontainer
drwxr-xr-x 4 root root 4.0K Oct  7 11:29 .
drwxr-xr-x 3 root root 4.0K Oct  7 11:36 ..
root@b8ed499d163f:/workspaces/mpags-day-1-drmarkwslater# 
```

Notes

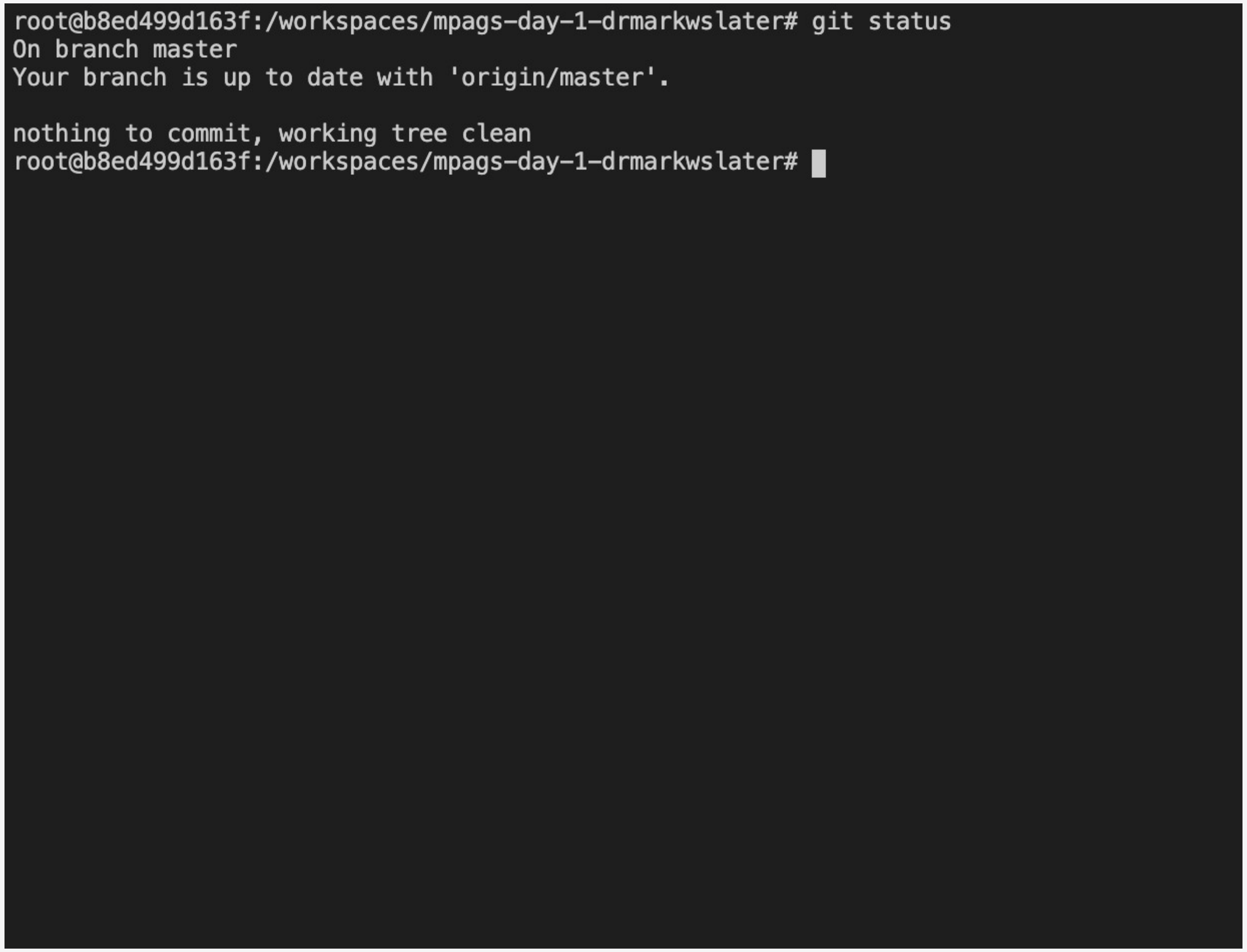
As this directory holds all of your changes, be very careful not to delete it!

We won't dig into the content or structure of .git in this course. If you're interested in learning more about this, the main Git references cover it in detail.

7: Viewing Repository Status

As we add and edit files, it's useful to keep track of the repository status without changing anything. With git, simply use the **`status`** command to view the current repository status:

```
$ git status
```



```
root@b8ed499d163f:/workspaces/mpags-day-1-drmarkwslater# git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
root@b8ed499d163f:/workspaces/mpags-day-1-drmarkwslater# 
```

Notes

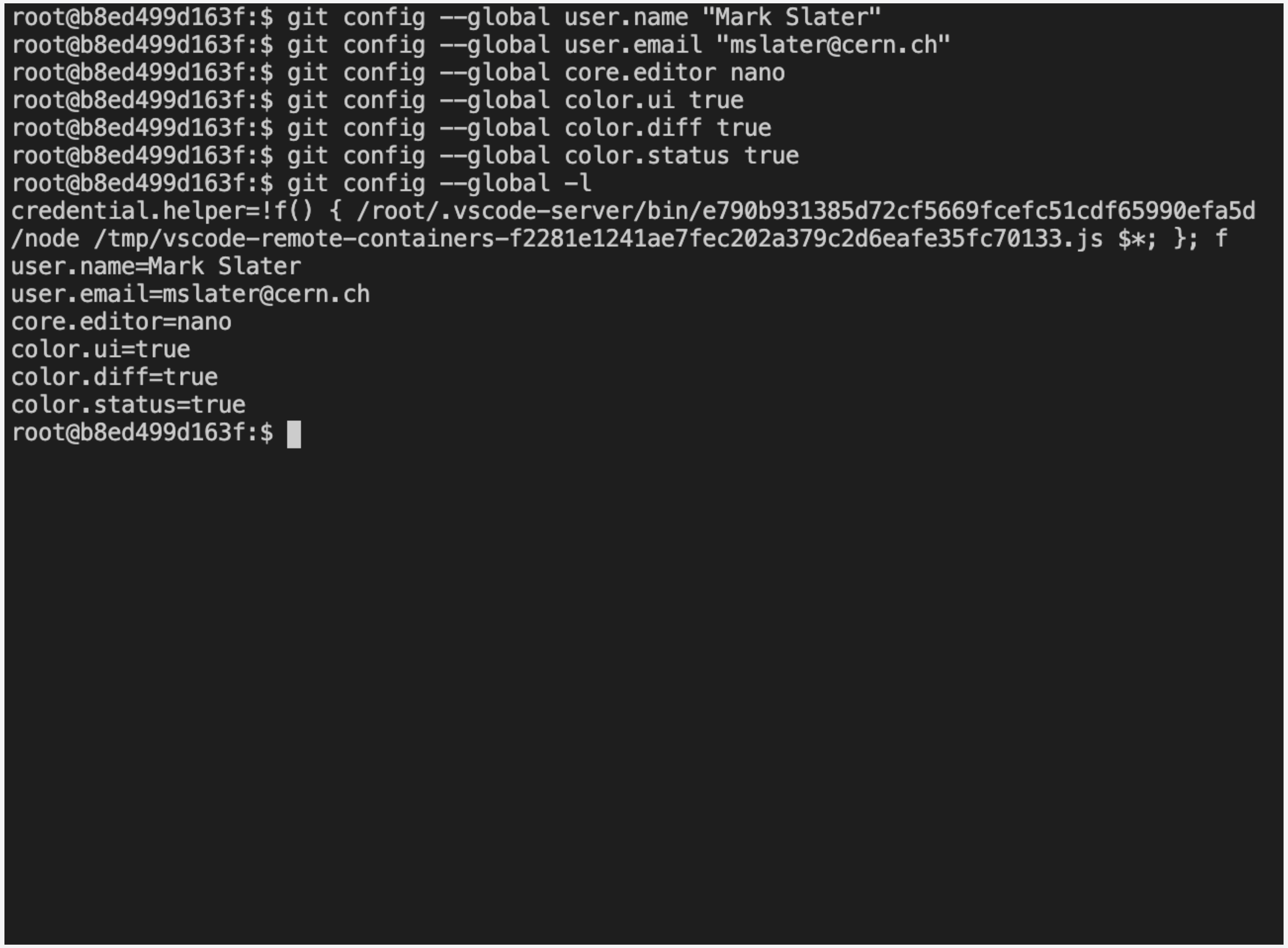
Of course at the moment, there's not much to report as we have not made any changes yet

Get into the habit of running git status regularly to see what you've changed.

8: Configuring Git

Before we start to use **`git`**, it's useful to configure it with the details to record in commit messages, an editor to use to write messages and to display changes using colour markup. The **`config`** command sets parameters, <key>, that **`git`** knows about to required values

```
$ git config --global <key> <value>
```



```
root@b8ed499d163f:$ git config --global user.name "Mark Slater"
root@b8ed499d163f:$ git config --global user.email "mslater@cern.ch"
root@b8ed499d163f:$ git config --global core.editor nano
root@b8ed499d163f:$ git config --global color.ui true
root@b8ed499d163f:$ git config --global color.diff true
root@b8ed499d163f:$ git config --global color.status true
root@b8ed499d163f:$ git config --global -l
credential.helper=!f() { /root/.vscode-server/bin/e790b931385d72cf5669fcefc51cdf65990efa5d
/node /tmp/vscode-remote-containers-f2281e1241ae7fec202a379c2d6eafe35fc70133.js $*; }; f
user.name=Mark Slater
user.email=mslater@cern.ch
core.editor=nano
color.ui=true
color.diff=true
color.status=true
root@b8ed499d163f:$ 
```

Hints

Git can use the **`EDITOR`** environment variable rather than **`core.editor`**

You can configure git options globally (**`--global`**) or locally in a repository (**`--local`**)

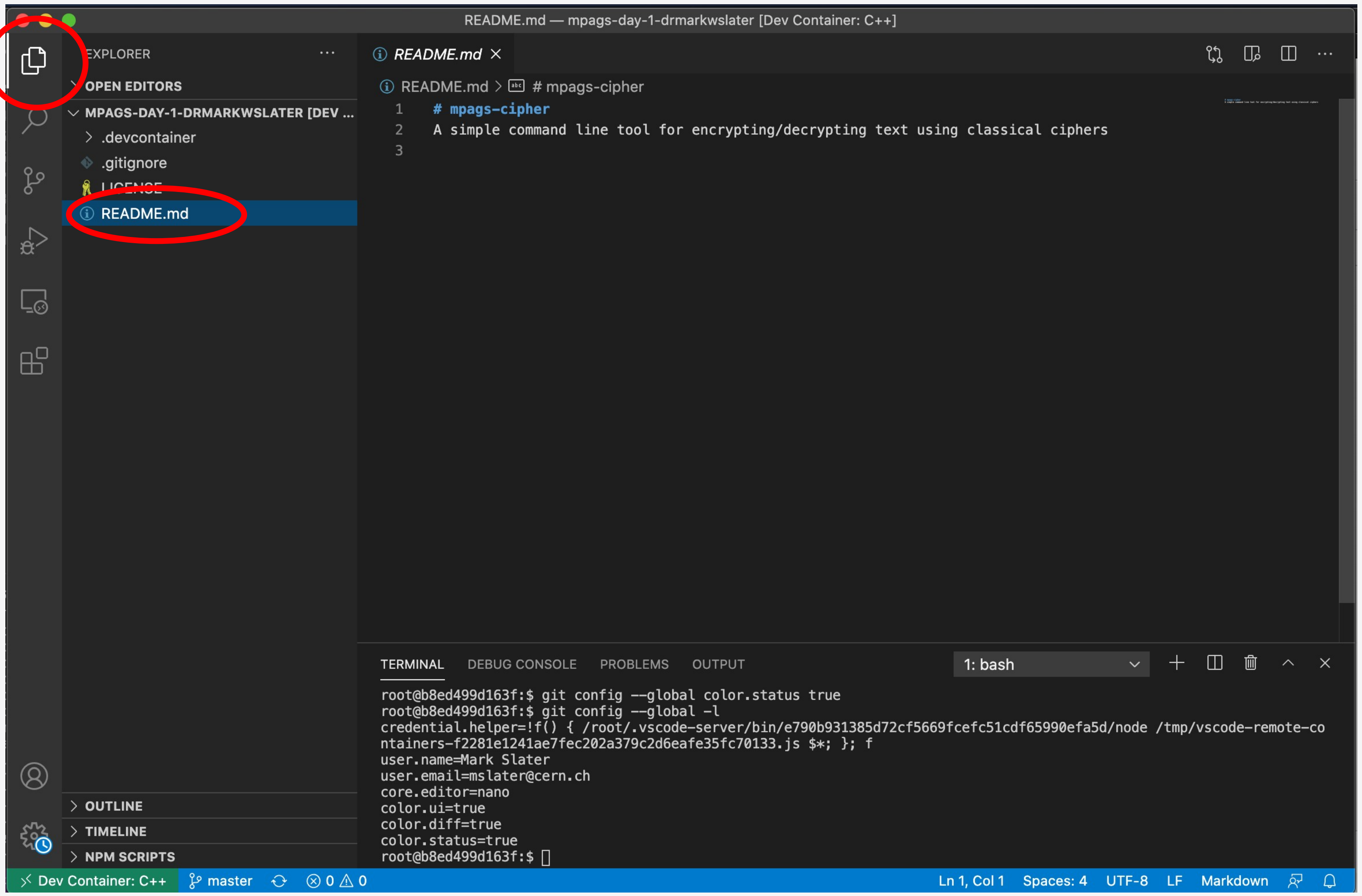
Also see the [Git SCM Book](#)

and try out some options!

9: Improving the README file

We're going to start our project by improving the **README** for **mpags-cipher**. This is a file that sits in the top level of the project and provides some basic information about the project, how to install it, and other details such as author/copyright/license details. Our **README** is in plain text, using Markdown formatting. We use Markdown because it is human readable but easily convertible to other formats.

In the 'Explorer' tab, click on '**README.md**' to open it. Edit the file and then save it.



```
# mpags-cipher
A simple command line tool for encrypting/decrypting text using classical ciphers
```

```
root@b8ed499d163f:$ git config --global color.status true
root@b8ed499d163f:$ git config --global -l
credential.helper=!f() { /root/.vscode-server/bin/e790b931385d72cf5669fcefc51cdf65990efa5d/node /tmp/vscode-remote-co
ntainers-f2281e1241ae7fec202a379c2d6eafe35fc70133.js $*; }; f
user.name=Mark Slater
user.email=mslater@cern.ch
core.editor=nano
color.ui=true
color.diff=true
color.status=true
root@b8ed499d163f:$
```

Notes

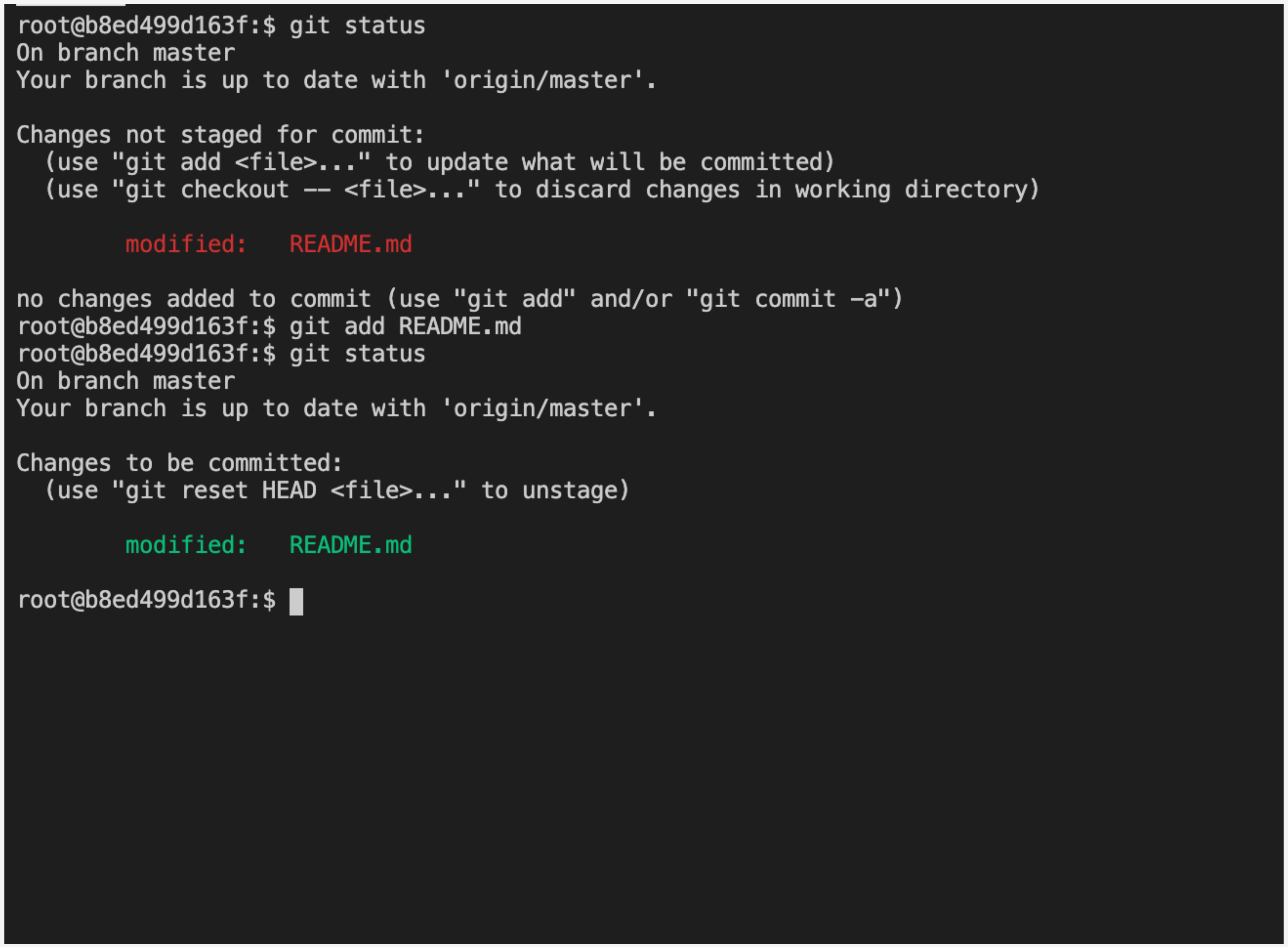
Add a bit more detail about the project, and create placeholder sections for "How to Install" and "Authors". Use the link above for a format guide

'#' marks major sections, and those with '##' are subsections. When we upload our project to github, we'll see how these are displayed.

10: Staging Changes for Commit

Having saved **README.md**, if you run **git status** again, git can see it's been changed. However, git marks these changes as "unstaged", i.e not yet ready to be committed to its database. To tell git we want to "stage" the changes, use the **git add** command on the file(s):

```
$ git add README.md
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
root@b8ed499d163f:$ git add README.md
root@b8ed499d163f:$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

root@b8ed499d163f:$ 
```

Notes

"staged" files

- Ready to be commited

"unstaged" files

- Changed but not staged

"untracked" files

- Not tracked by git yet

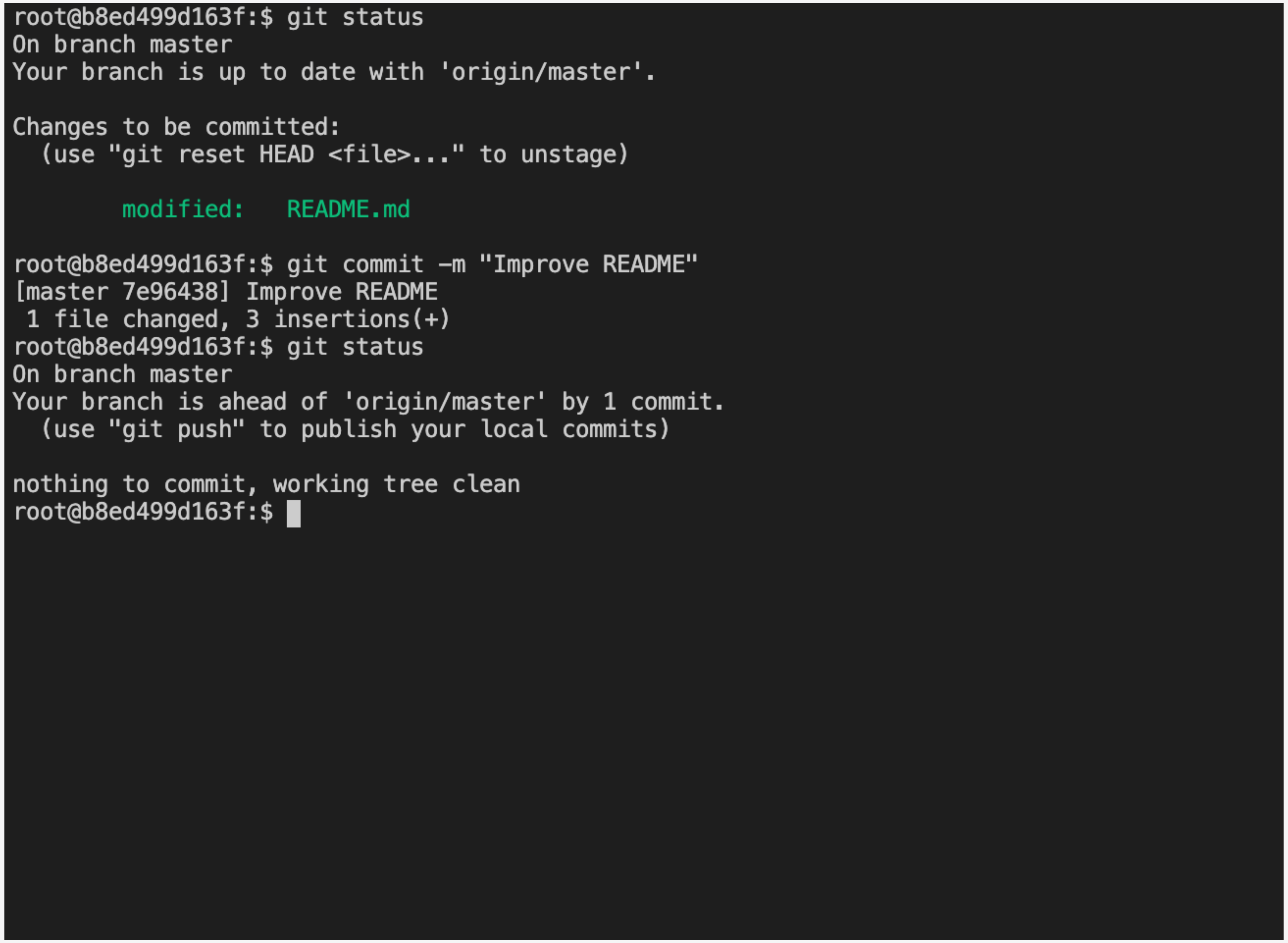
"deleted" files

- Deleted by git and ready for removal

11: Committing Changes

You'll have noticed that when you ran **`git status`** after adding **`README.md`** it only says "*Changes to be committed*". Git stages changes before committing them to the repository. To store the changes we use the **`commit`** command with a message describing the changes:

```
$ git commit -m "Improve README"
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

root@b8ed499d163f:$ git commit -m "Improve README"
[master 7e96438] Improve README
 1 file changed, 3 insertions(+)
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@b8ed499d163f:$
```

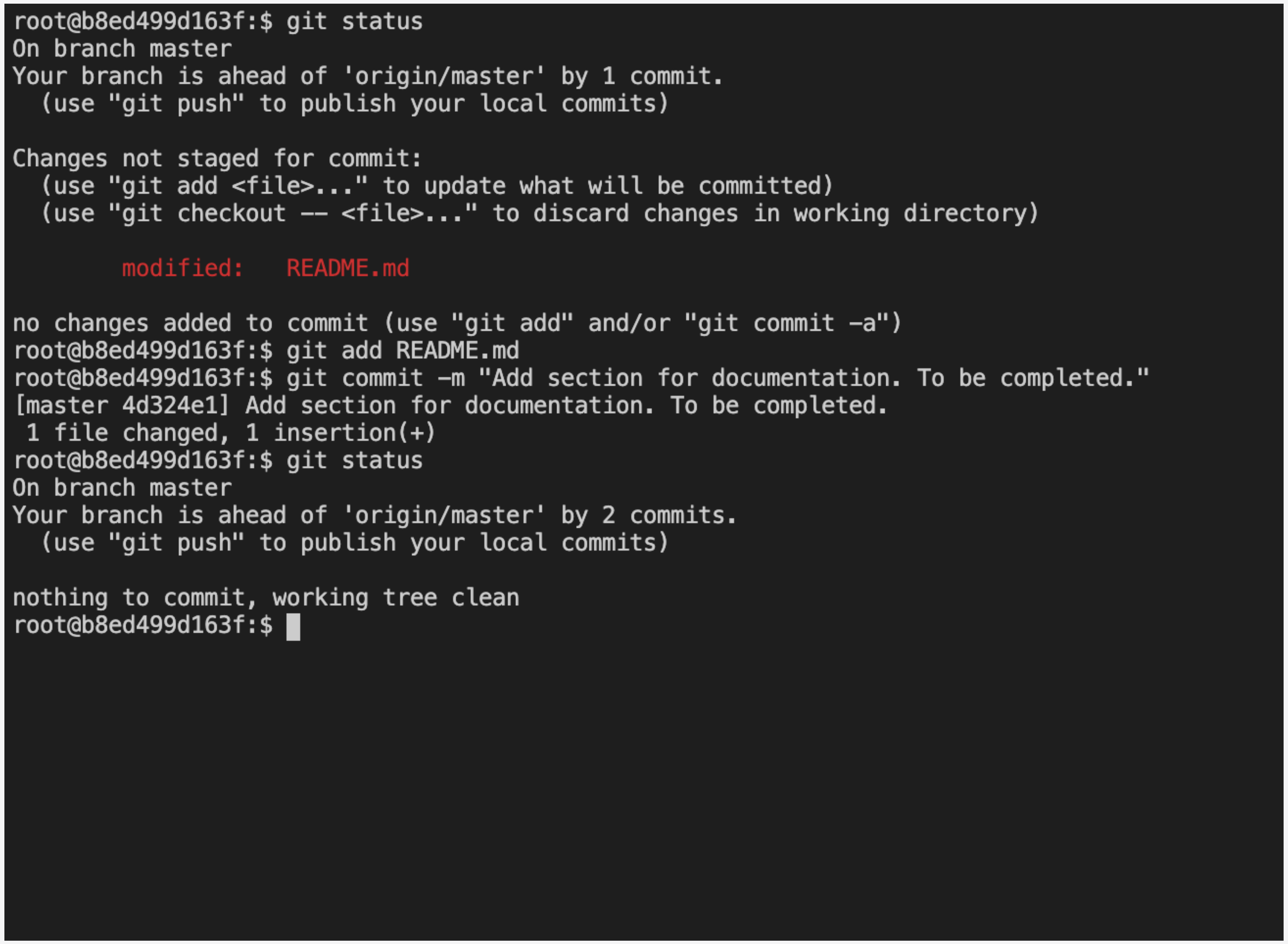
Notes

The staging area is a place to queue up (or remove) changes before they are committed. This is useful when we start to deal with changes across multiple files

A "commit" is a snapshot of the repository. After the commit, **`status`** shows that we're ahead of where we cloned from (GitHub).

12: Making Further Changes

Now we've staged and committed **`README.md`**, we can continue to make changes. So edit your **`README.md`**, for example, add an empty section "Documentation". Save the file and run **`git status`** again. As before, git recognises we've made changes, but these are not yet staged. To actually update the repository, we run **`git add`** again to stage the change then **`git commit`** to update the repository. This cycle of staging and committing is the basic git workflow.



```
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
root@b8ed499d163f:$ git add README.md
root@b8ed499d163f:$ git commit -m "Add section for documentation. To be completed."
[master 4d324e1] Add section for documentation. To be completed.
 1 file changed, 1 insertion(+)
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@b8ed499d163f:$ 
```

Try This

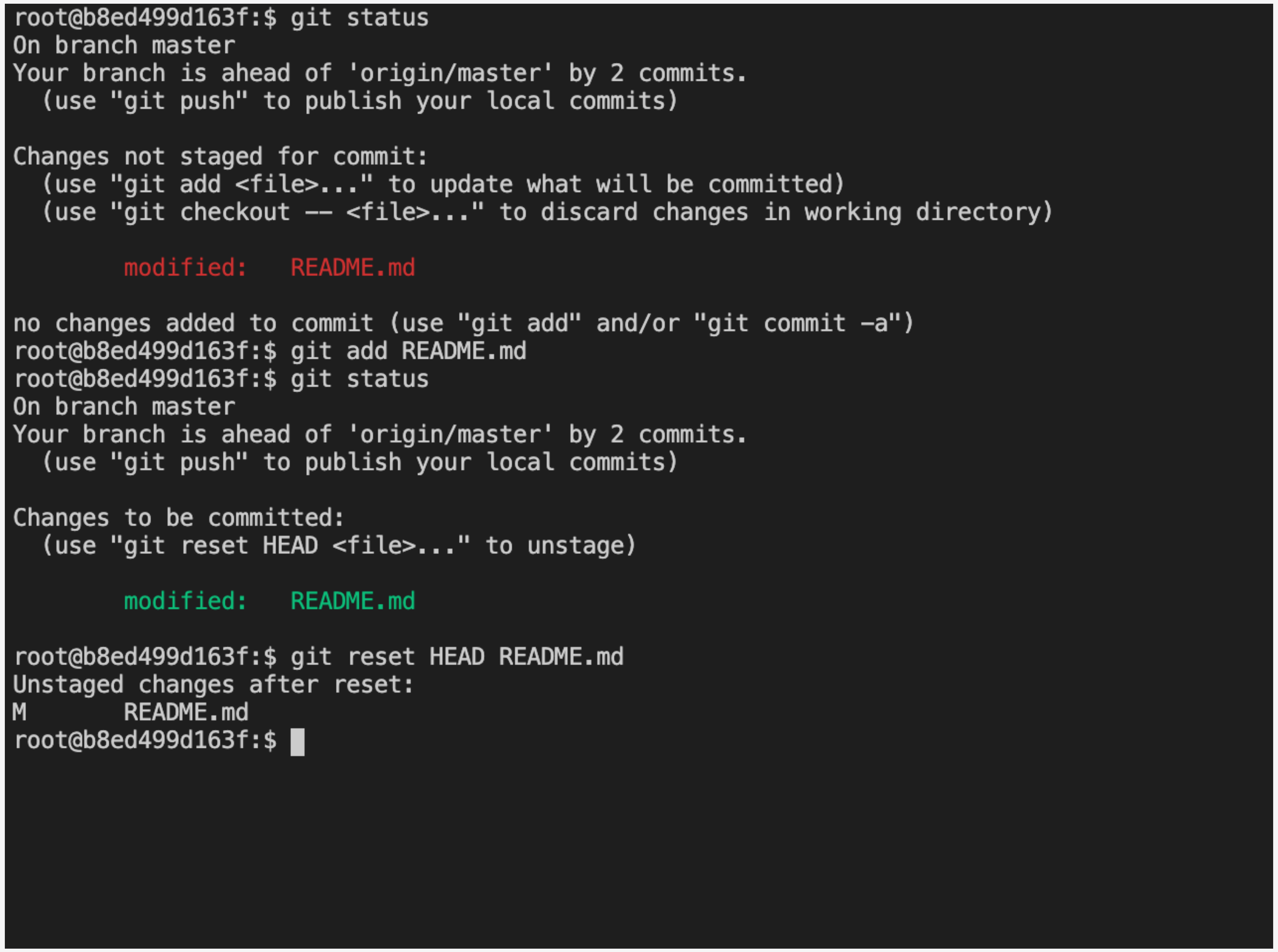
Make a few more edits to **`README.md`** and use **`git add`** and **`git commit`** for each to get into the feel of staging and committing.

Remember to use **`git status`** regularly to see what's happening!

13: Unstaging Changes

So you've staged a change, and then you realise it either breaks something or you want to add something else. As you may have noticed, git status actually tells you what to do in this case. So, make a change, stage it up and then use `git reset` to unstage it:

```
$ git reset HEAD README.md
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
root@b8ed499d163f:$ git add README.md
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

root@b8ed499d163f:$ git reset HEAD README.md
Unstaged changes after reset:
M       README.md
root@b8ed499d163f:$ 
```

Notes

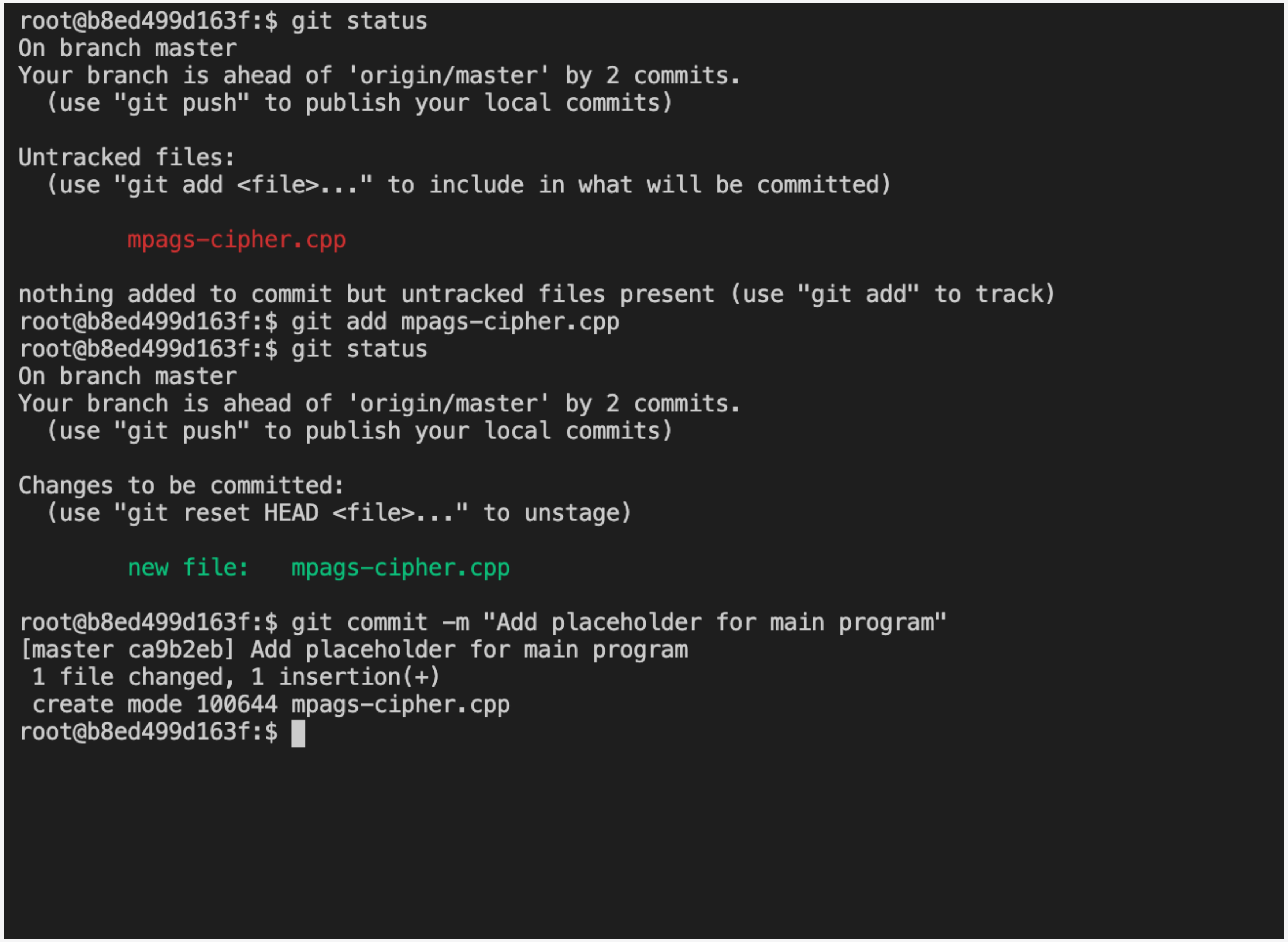
If you have a change staged then make further changes, just use `git add` to append these to the staging area.

Note that you need to do this if you've staged a file then made further changes to it.

14: Adding New Files

Staging/committing new files is the same as working with existing files. Create a new file named **`mpags-cipher.cpp`** with a single line "**`// mpags-cipher.cpp`**" and save it. Run **`git status`** again, and git recognises a new "Untracked" file, so just use **`git add`** to track it:

```
$ git add mpags-cipher.cpp
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

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

        mpags-cipher.cpp

nothing added to commit but untracked files present (use "git add" to track)
root@b8ed499d163f:$ git add mpags-cipher.cpp
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   mpags-cipher.cpp

root@b8ed499d163f:$ git commit -m "Add placeholder for main program"
[master ca9b2eb] Add placeholder for main program
 1 file changed, 1 insertion(+)
 create mode 100644 mpags-cipher.cpp
root@b8ed499d163f:$ 
```

Try This

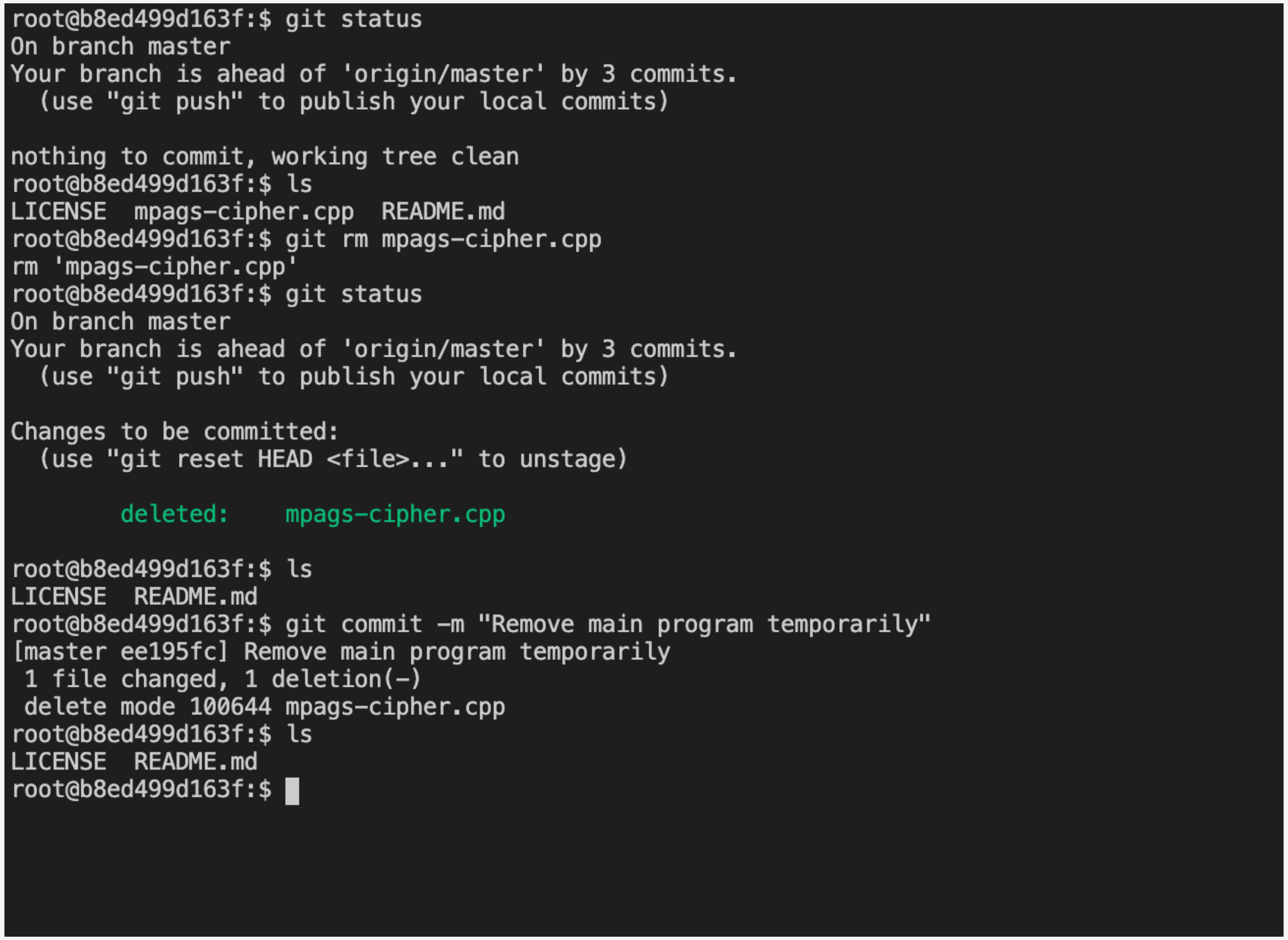
Once staged, git recognises **`mpags-cipher.cpp`** as a "**`new file`**". However, the next commit step is identical, so just commit as normal!

Commits can contain both new files and modifications to existing ones. Note that you can run **`git add`** with multiple files at once.

15: Removing Files

Files can also be removed, but note that git's **`rm`** command removes the file(s) from both the repository and local disk! Git regards a deletion as a change, so **`rm`** also stages the deletion for commit (though the physical file has been deleted). Try removing the **`mpags-cipher.cpp`** file:

```
$ git rm mpags-cipher.cpp
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@b8ed499d163f:$ ls
LICENSE  mpags-cipher.cpp  README.md
root@b8ed499d163f:$ git rm mpags-cipher.cpp
rm 'mpags-cipher.cpp'
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    mpags-cipher.cpp

root@b8ed499d163f:$ ls
LICENSE  README.md
root@b8ed499d163f:$ git commit -m "Remove main program temporarily"
[master ee195fc] Remove main program temporarily
 1 file changed, 1 deletion(-)
 delete mode 100644 mpags-cipher.cpp
root@b8ed499d163f:$ ls
LICENSE  README.md
root@b8ed499d163f:$ 
```

Try This

After using **`git rm`**, make the commit.

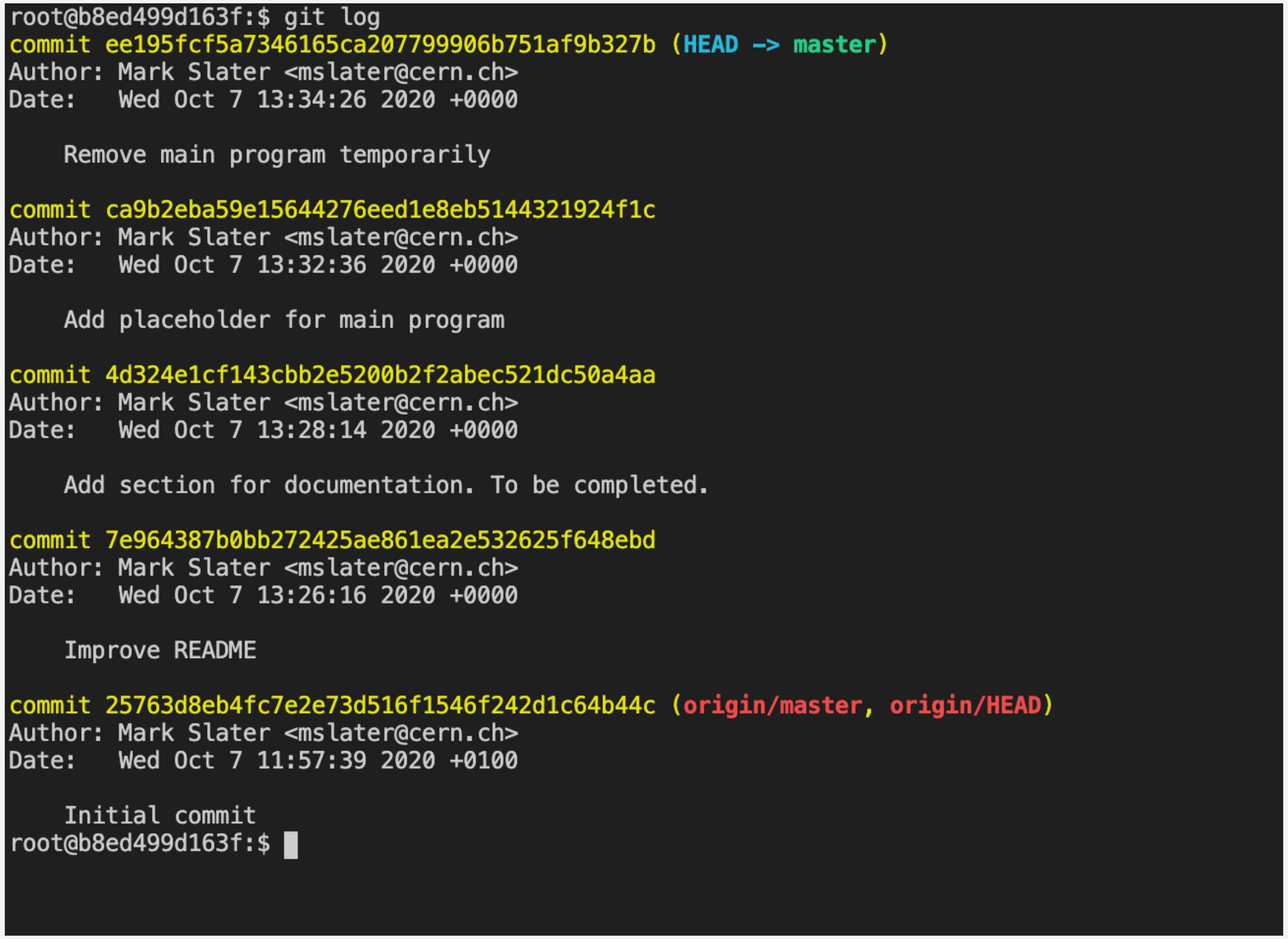
As with all changes, you can stage deletions along with additions and modifications.

Like **`git add`**, **`git rm`** can be run with several files at once.

16: Viewing Logs

We've now made a few commits to our repository, so how do we go back and see what we've done and why? Just use the **`log`** command!

```
$ git log
```



```
root@b8ed499d163f:$ git log
commit ee195fcf5a7346165ca207799906b751af9b327b (HEAD -> master)
Author: Mark Slater <mslater@cern.ch>
Date:   Wed Oct 7 13:34:26 2020 +0000

    Remove main program temporarily

commit ca9b2eba59e15644276eed1e8eb5144321924f1c
Author: Mark Slater <mslater@cern.ch>
Date:   Wed Oct 7 13:32:36 2020 +0000

    Add placeholder for main program

commit 4d324e1cf143cbb2e5200b2f2abec521dc50a4aa
Author: Mark Slater <mslater@cern.ch>
Date:   Wed Oct 7 13:28:14 2020 +0000

    Add section for documentation. To be completed.

commit 7e964387b0bb272425ae861ea2e532625f648ebd
Author: Mark Slater <mslater@cern.ch>
Date:   Wed Oct 7 13:26:16 2020 +0000

    Improve README

commit 25763d8eb4fc7e2e73d516f1546f242d1c64b44c (origin/master, origin/HEAD)
Author: Mark Slater <mslater@cern.ch>
Date:   Wed Oct 7 11:57:39 2020 +0100

    Initial commit
root@b8ed499d163f:$
```

Try This

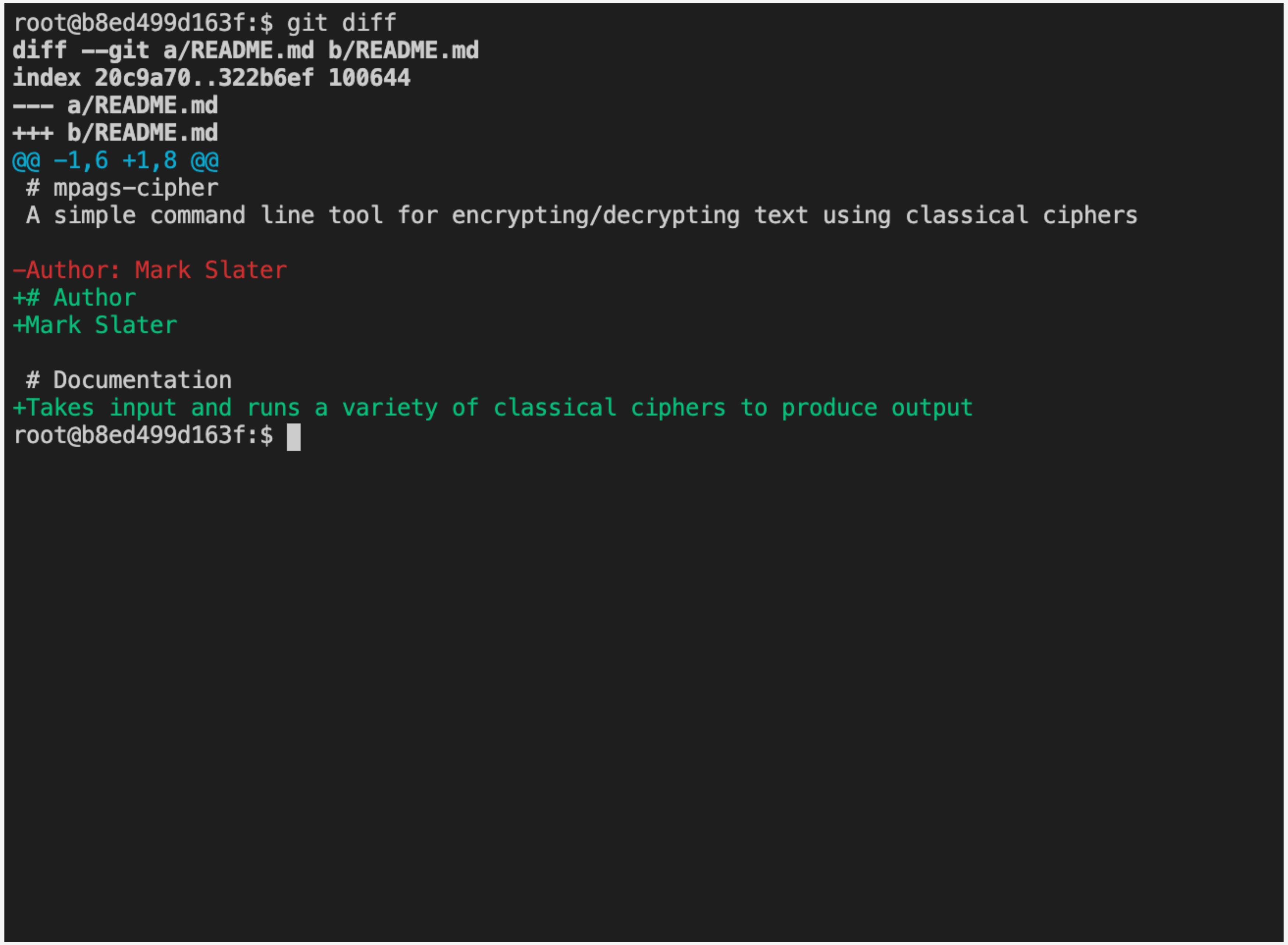
Plain log displays everything! To get the N most recent commits, use **`git log -nN`**.

You can also use **`git log --summary`** to get a more detailed overview, though it doesn't show much as we've only worked with one file.

17: Viewing Changes

The basic log command shows the timeline of changes, but not what changed. To see what actually changed between commits, we can use **`git log -p`** or the **`diff`** command. Without any arguments, it shows a diff between the last commit and any **unstaged** changes:

```
$ git diff
```



```
root@b8ed499d163f:$ git diff
diff --git a/README.md b/README.md
index 20c9a70..322b6ef 100644
--- a/README.md
+++ b/README.md
@@ -1,6 +1,8 @@
 # mpags-cipher
 A simple command line tool for encrypting/decrypting text using classical ciphers

-Author: Mark Slater
+# Author
+Mark Slater

 # Documentation
+Takes input and runs a variety of classical ciphers to produce output
root@b8ed499d163f:$ 
```

Notes

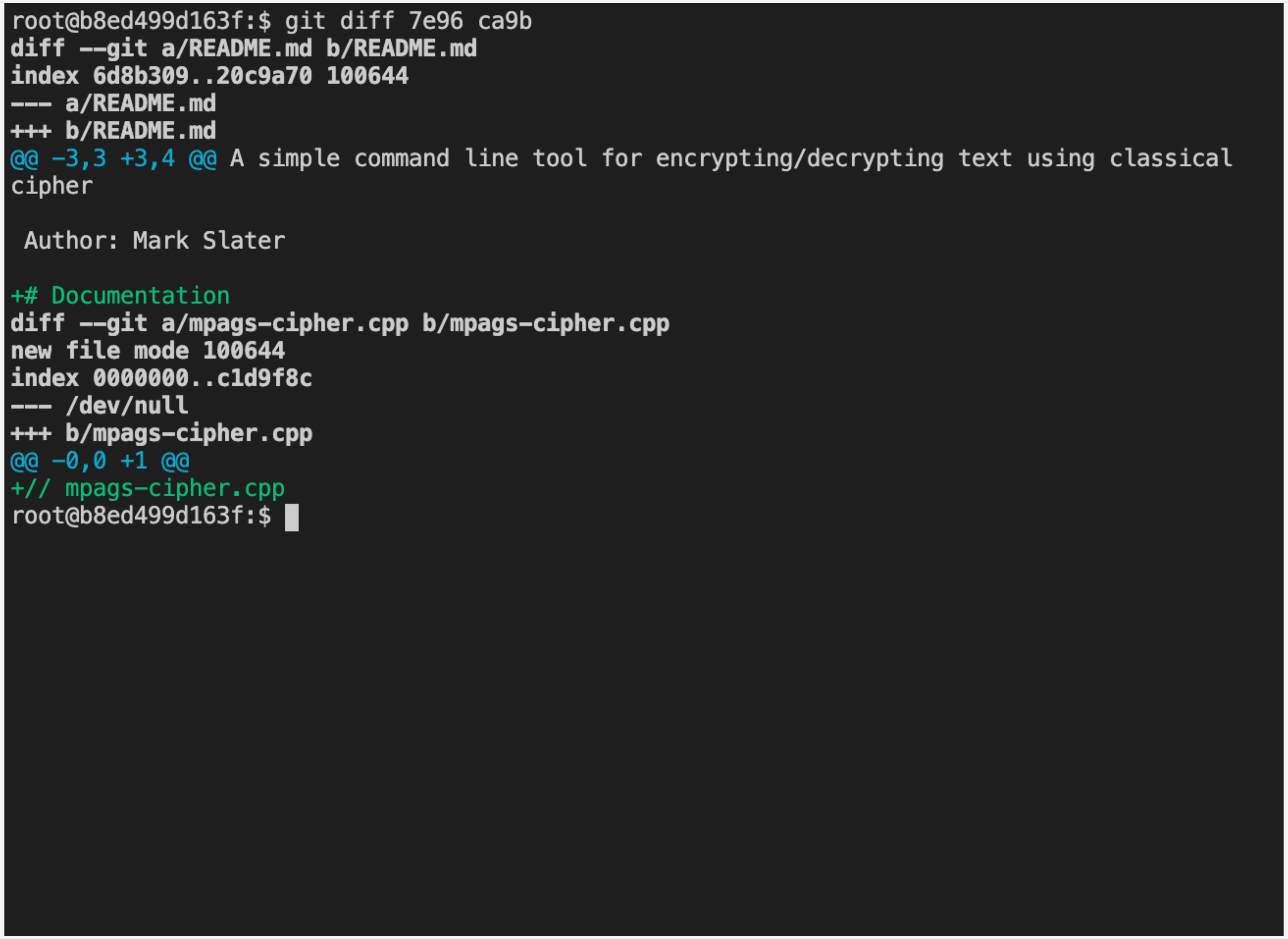
Git shows difference using the standard diff format for additions/removals. On the left, additions are in green, removals in red.

Depending on the default configuration, the diff may be output to a pager, in which case use ‘q’ to quit.

18: Changes between Commits

As you'll have seen in using **`git log`**, git labels commits using a 40 character hash code. You can use these labels to view differences between any two commits, though because hashes are unique, you don't have to type out 80 characters, e.g:

```
$ git diff 7e96 ca9b
```



```
root@b8ed499d163f:$ git diff 7e96 ca9b
diff --git a/README.md b/README.md
index 6d8b309..20c9a70 100644
--- a/README.md
+++ b/README.md
@@ -3,3 +3,4 @@ A simple command line tool for encrypting/decrypting text using classical
cipher

 Author: Mark Slater

+# Documentation
diff --git a/mpags-cipher.cpp b/mpags-cipher.cpp
new file mode 100644
index 0000000..c1d9f8c
--- /dev/null
+++ b/mpags-cipher.cpp
@@ -0,0 +1 @@
+// mpags-cipher.cpp
root@b8ed499d163f:$
```

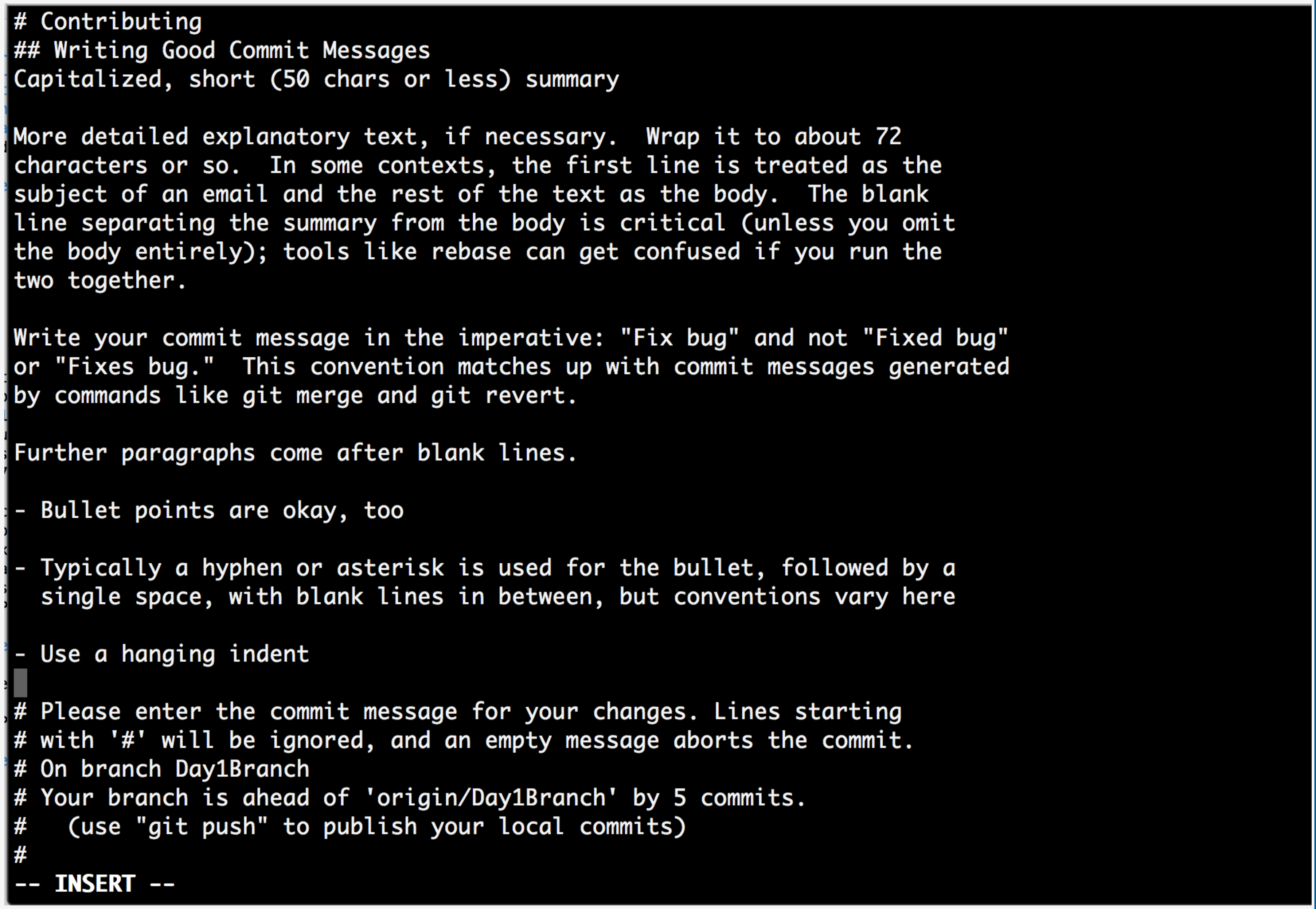
Try This

The commit specifier needs to contain enough characters to uniquely identify the commit. Note that your hashes will differ!

The arguments to git diff can take a variety of forms. See **`man gitrevisions`** for more details, or the more helpful Git SCM Book! Try some of these out.

19: Writing Good Commit Messages

Our edits so far have been simple and confined to one file. In these cases, a single line commit message using **`git commit -m "commit message"`** is completely sufficient (e.g. "Fixed typographic errors"). As we start to make more involved commits involving several files, then we need to provide more detail. Because of the way git works with patches and email, it tends to recommend the specific style of commit message listed below.



```
# Contributing
## Writing Good Commit Messages
Capitalized, short (50 chars or less) summary

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Write your commit message in the imperative: "Fix bug" and not "Fixed bug"
or "Fixes bug."  This convention matches up with commit messages generated
by commands like git merge and git revert.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, followed by a
  single space, with blank lines in between, but conventions vary here

- Use a hanging indent

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch Day1Branch
# Your branch is ahead of 'origin/Day1Branch' by 5 commits.
#   (use "git push" to publish your local commits)
#
-- INSERT --
```

Why?

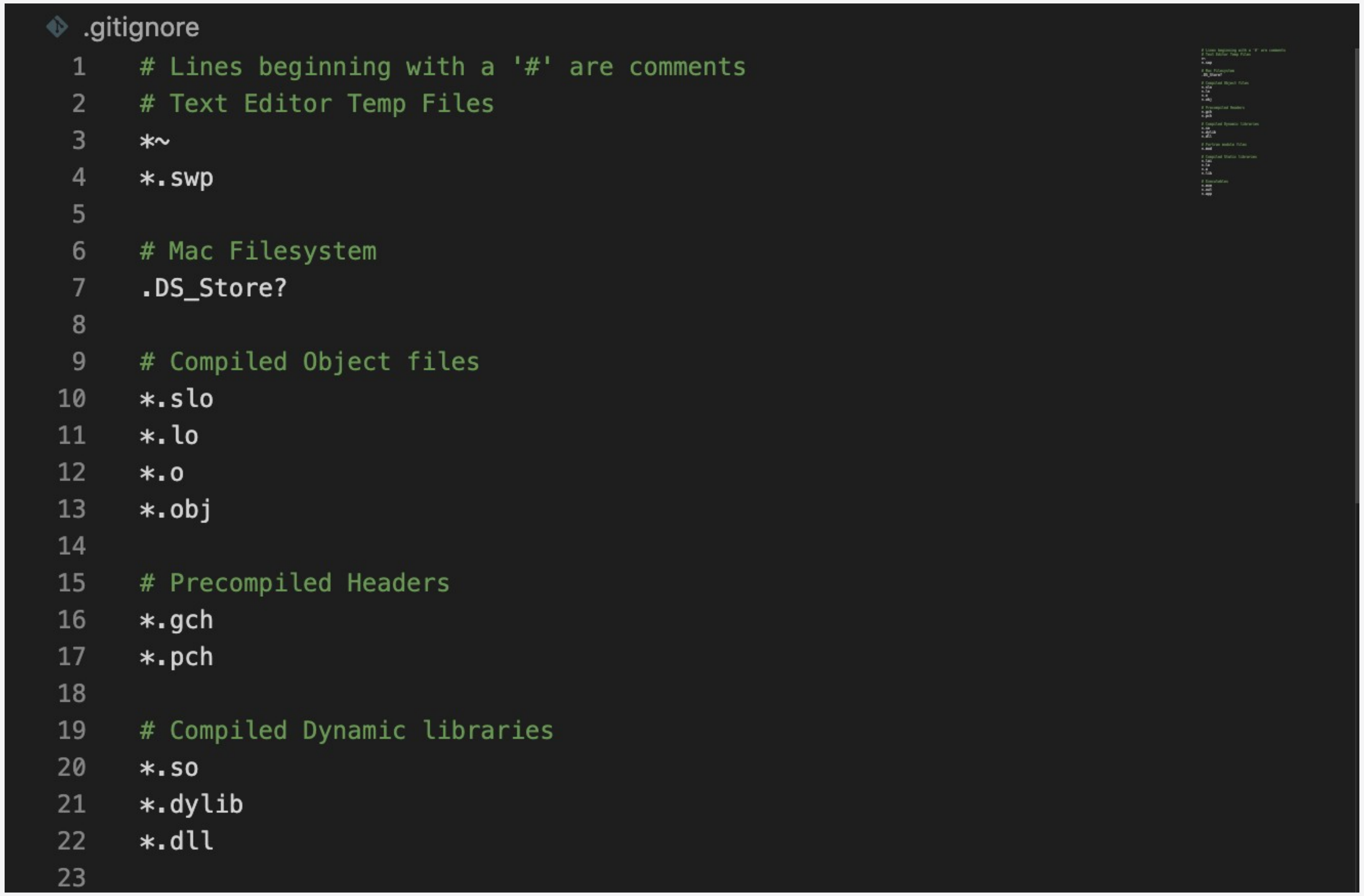
There's a good example in the text on the left (taken from a post by Tim Pope).

If you "fixed a bug" you should say which bug, and how it was fixed. You might also say (and include in the commit) that a test has been added to check for the bug in the future.

20: The .gitignore File

When you run **`git status`**, git will report any files it doesn't track ("untracked"). In some cases we'll have files that we don't want git to track, for example files generated by the build or text editor temporaries, but we may accidentally add them to the repository (e.g. by **`git add .`**).

Git uses the .gitignore file in our repository (provided by upstream) to determine what to ignore. This contains a list of filename patterns that git should ignore and already contains patterns for C++ compiled objects. Have a look at the .gitignore provided in the repository and make sure it looks like below:


.gitignore

```
# Lines beginning with a '#' are comments
# Text Editor Temp Files
*~
*.swp

# Mac Filesystem
.DS_Store?

# Compiled Object files
*.slo
*.lo
*.o
*.obj

# Precompiled Headers
*.gch
*.pch

# Compiled Dynamic libraries
*.so
*.dylib
*.dll

```

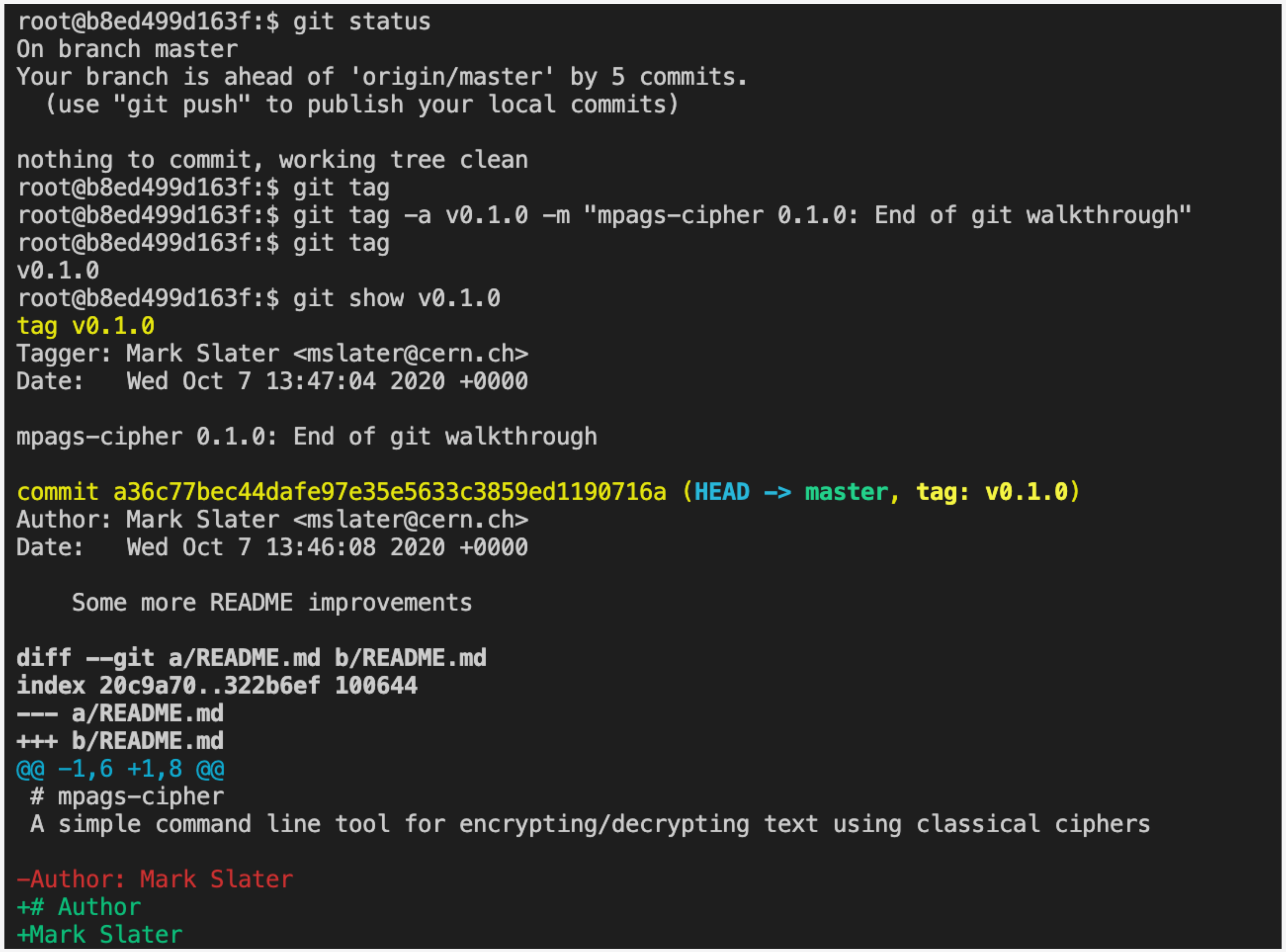
Notes

You can also have a global ignores file. You could have a file named **`.global_gitignores`** in your **`HOME`** directory. Git can be made aware of this file by setting the **`core.excludesfile`** variable to point to it in the global **`git config`**

21: Tagging

We've seen that in git, commits are described by a 40 character hash. At certain points in development, we'll want to mark a commit as a usable, stable piece of work. The hashes aren't an easy way of marking these points, so instead we create a "tag". Current tags are listed via:

```
$ git tag
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@b8ed499d163f:$ git tag
root@b8ed499d163f:$ git tag -a v0.1.0 -m "mpags-cipher 0.1.0: End of git walkthrough"
root@b8ed499d163f:$ git tag
v0.1.0
root@b8ed499d163f:$ git show v0.1.0
tag v0.1.0
Tagger: Mark Slater <mslater@cern.ch>
Date:   Wed Oct 7 13:47:04 2020 +0000

mpags-cipher 0.1.0: End of git walkthrough

commit a36c77bec44dafe97e35e5633c3859ed1190716a (HEAD -> master, tag: v0.1.0)
Author: Mark Slater <mslater@cern.ch>
Date:   Wed Oct 7 13:46:08 2020 +0000

    Some more README improvements

diff --git a/README.md b/README.md
index 20c9a70..322b6ef 100644
--- a/README.md
+++ b/README.md
@@ -1,6 +1,8 @@
 # mpags-cipher
 A simple command line tool for encrypting/decrypting text using classical ciphers

-Author: Mark Slater
+# Author
+Mark Slater
```

Try This

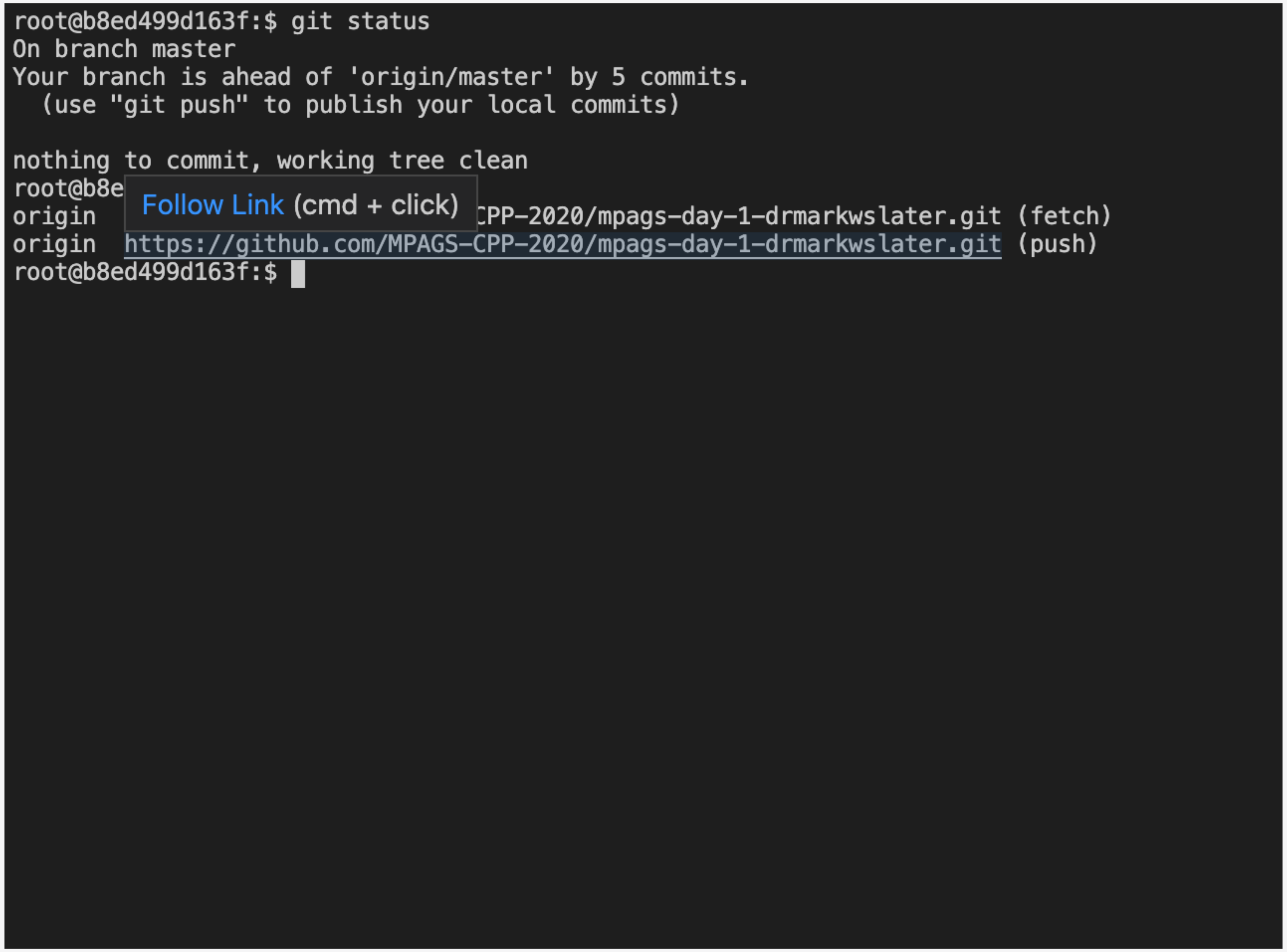
Follow the steps on the left to tag your repository,

Tags can have any name, but git projects tend to use 'vMAJOR.MINOR.PATCH' for version numbers. "Annotated" tags are the best to use to begin with, as they can take extra info about the tag. Use show to see this info.

22: Sharing Changes between Repositories

Whilst we've made commits to our repository, these are all local as we work on a copy of the repository. If you go back to the GitHub page for your project and refresh the browser, you'll see that this is still in its original state. The distributed nature of git means it can track a set of repositories, which we can view with the remote command:

```
$ git remote -v
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@b8e
origin           CPP-2020/mpags-day-1-drmarkwslater.git (fetch)
origin  https://github.com/MPAGS-CPP-2020/mpags-day-1-drmarkwslater.git (push)
root@b8ed499d163f:$
```

Notes

Because we cloned from Github, the default "origin" remote points to it.

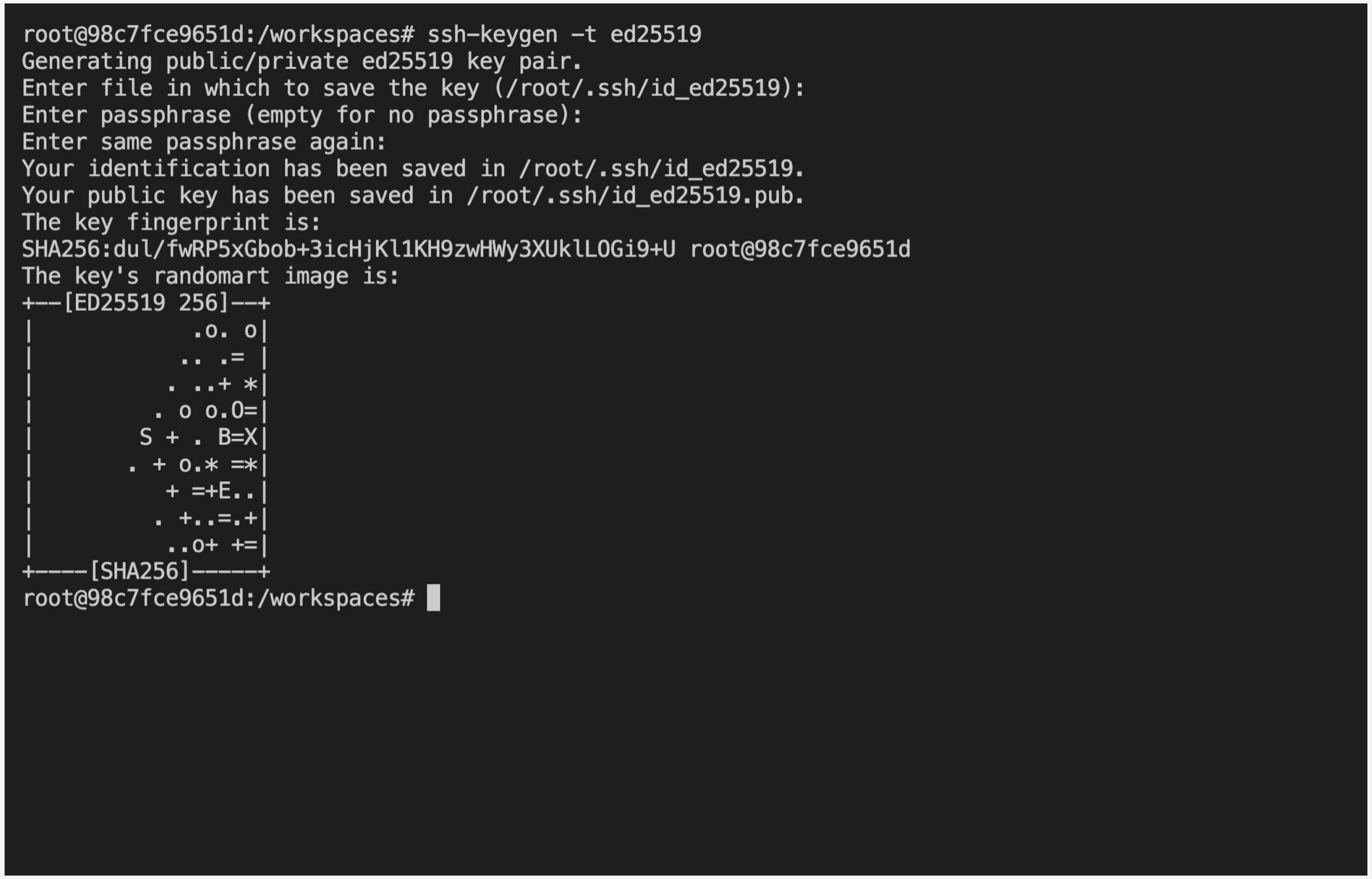
The "-v" flag gets git to show the full URLs.

As we'll see, we can have multiple remotes.

23: Preparing to Send your Repository to github

We now want to send our changes to github. However, to do this we need to generate an **ssh key** consisting of a **public** part and a **private** part. You send the **public** part to github and keep the **private** part on your machine and using using the combination, github can authenticate you. To generate the key, run the following:

```
$ ssh-keygen -t ed25519
```



```
root@98c7fce9651d:/workspaces# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519.
Your public key has been saved in /root/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:dul/fwRP5xGbob+3icHjKl1KH9zwHWy3XUklLOGi9+U root@98c7fce9651d
The key's randomart image is:
+--[ED25519 256]--+
|            .o. o|
|           .. .= |
|          . ..+ *|
|         . o o.O=|
|        S + . B=X|
|       . + o.* =*|
|          + =+E..|
|         . +..=.+|
|          ..o+ +=|
+----[SHA256]-----+
root@98c7fce9651d:/workspaces# █
```

Notes

During this process you will be asked to provide a passphrase. This will ‘unlock’ the ssh key when you push to github.

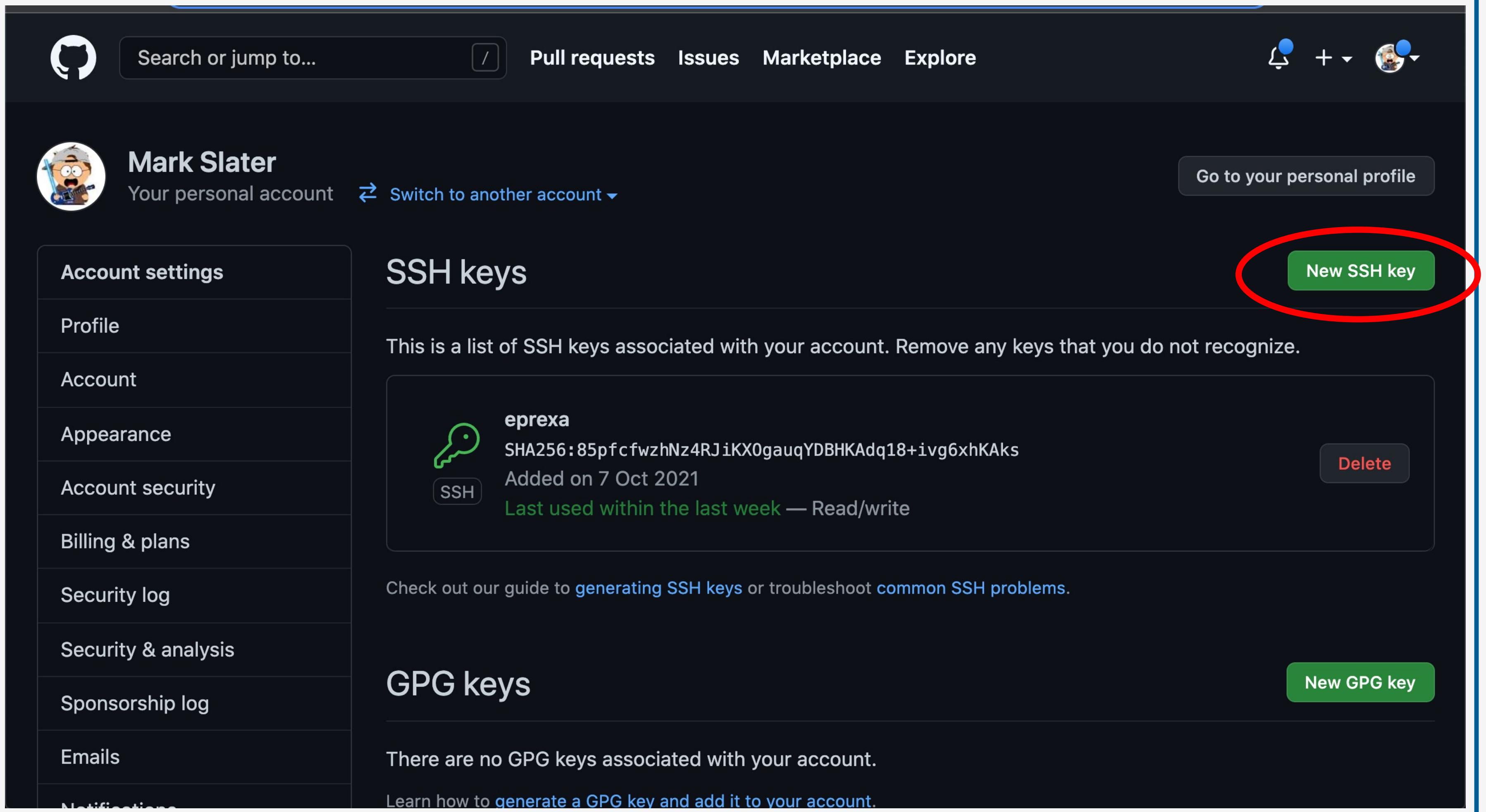
Though this is not sent over the internet and is useless without the private part of the key, you should still ensure it’s not easily guessable!

24: Adding the Public SSH Key Part to Github

You now have two additional files: `~/.ssh/id_ed25519` and `~/.ssh/id_ed25519.pub`. The second of these is the **public** part which you can freely give out – **DON'T GIVE OUT THE PRIVATE PART!** To provide this to github, go to:

https://github.com/settings/keys

And copy and paste the contents of `~/.ssh/id_ed25519.pub` file as well as giving it a title.



SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

eprexa
SHA256:85pfcfwzhNz4RJiKX0gauqYDBHKAdq18+ivg6xhKAks
Added on 7 Oct 2021
Last used within the last week — Read/write

Check out our guide to generating SSH keys or troubleshoot common SSH problems.

GPG keys

There are no GPG keys associated with your account.

Learn how to generate a GPG key and add it to your account.

Notes

You can store many keys on github so if you are using several machines, you can either copy the private and public parts to the different machines or generate different keys for each.

You will have to create a new key for each day of the course as we use a different image each time. Don't forget to delete the old ones from your github account!

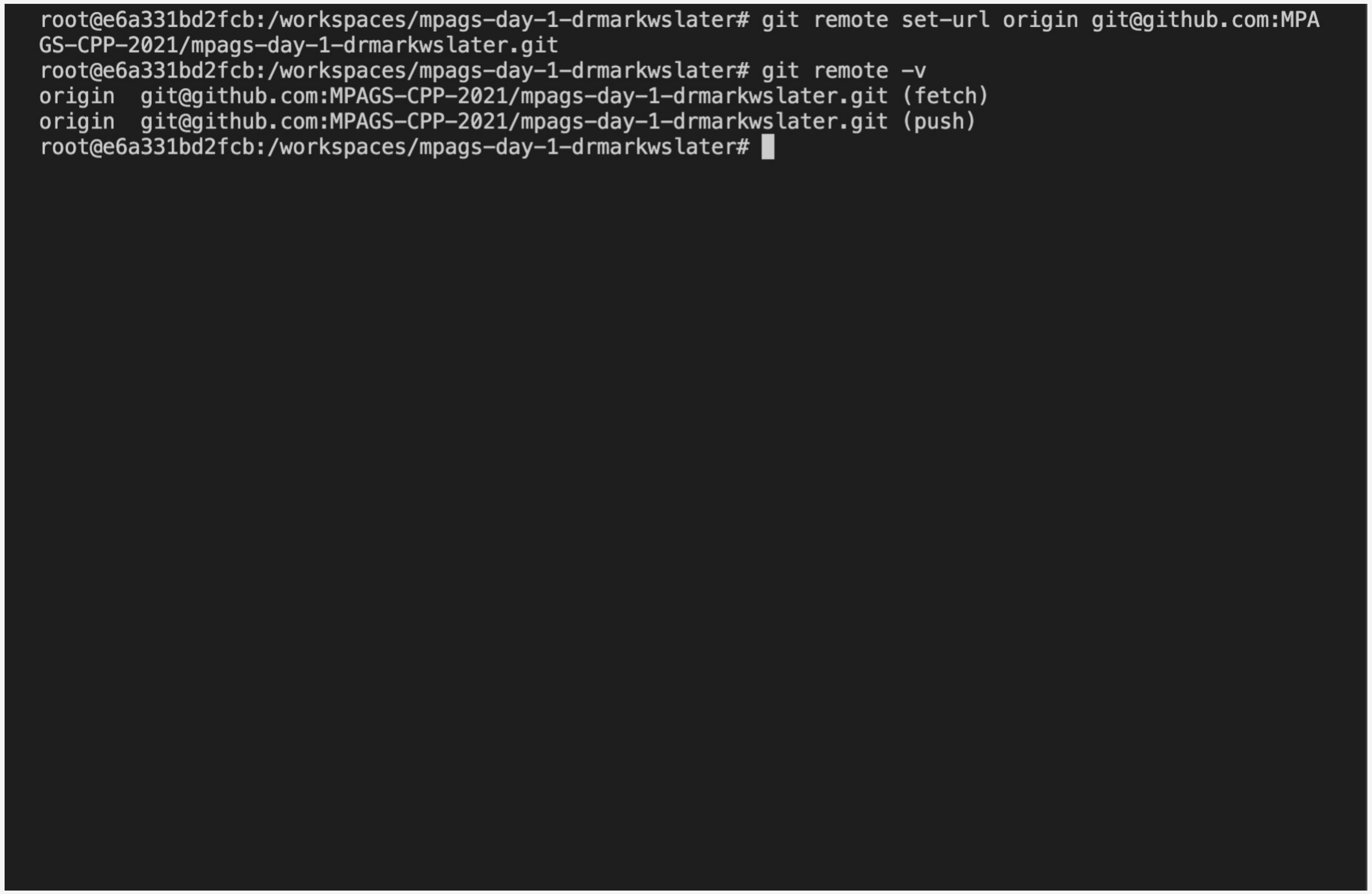
25: Changing the remote URL

We need to make one more change – when we downloaded the repo we used ‘https’ but we want to **push** our changes using **ssh**. To change this, we will change the URL or the ‘origin’ remote:

```
$ git remote set-url origin <remote-url>
```

The URL you need can be found by just replacing the ‘https://github.com/’ part to ‘git@github.com:’, e.g.

https://github.com/MPAGS-CPP-2021/mpags-day-1-drmarkwslater.git → git@github.com:MPAGS-CPP-2021/mpags-day-1-drmarkwslater.git



```
root@e6a331bd2fcb:/workspaces/mpags-day-1-drmarkwslater# git remote set-url origin git@github.com:MPA
GS-CPP-2021/mpags-day-1-drmarkwslater.git
root@e6a331bd2fcb:/workspaces/mpags-day-1-drmarkwslater# git remote -v
origin  git@github.com:MPAGS-CPP-2021/mpags-day-1-drmarkwslater.git (fetch)
origin  git@github.com:MPAGS-CPP-2021/mpags-day-1-drmarkwslater.git (push)
root@e6a331bd2fcb:/workspaces/mpags-day-1-drmarkwslater# █
```

Notes

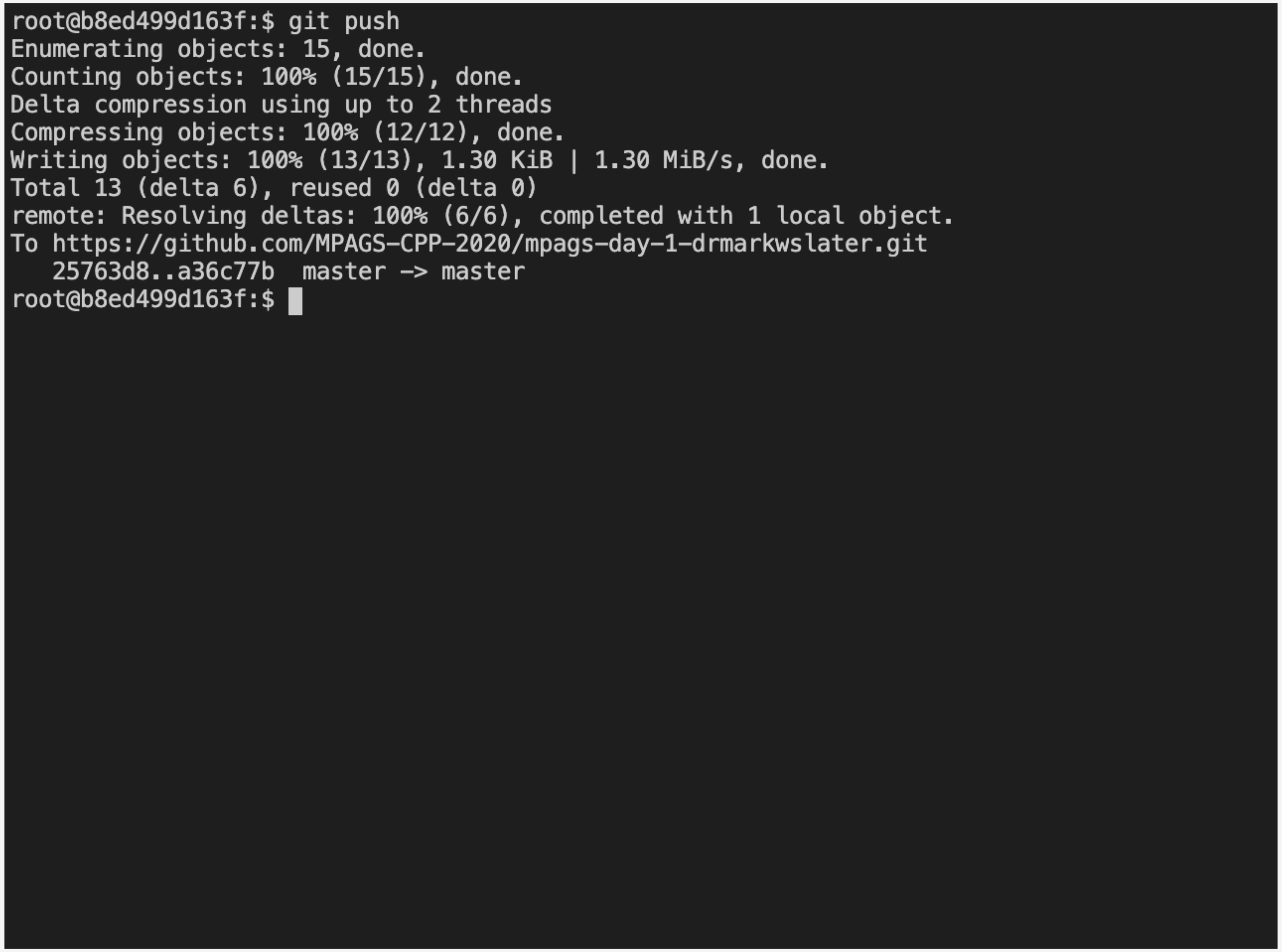
You can also find the URL from the gihub interface for your repository. Go to ‘Code’ and select ‘SSH’. The URL can be copied from there.

Along with generating a new SSH key for each day, you will also need to change the remote URL each time as well.

26: Pushing your Repository to Github

We're now ready to to send our changes to github. To do this, we use **`git push`**. This can be supplied with the name of the remote to push to ('origin' usually), and the "refspec" or branch we want to share. In our case though, we can just use the defaults (origin and the current branch - Day1Branch):

```
$ git push
```



```
root@b8ed499d163f:$ git push
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 2 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 1.30 KiB | 1.30 MiB/s, done.
Total 13 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), completed with 1 local object.
To https://github.com/MPAGS-CPP-2020/mpags-day-1-drmarkwslater.git
   25763d8..a36c77b  master -> master
root@b8ed499d163f:$ 
```

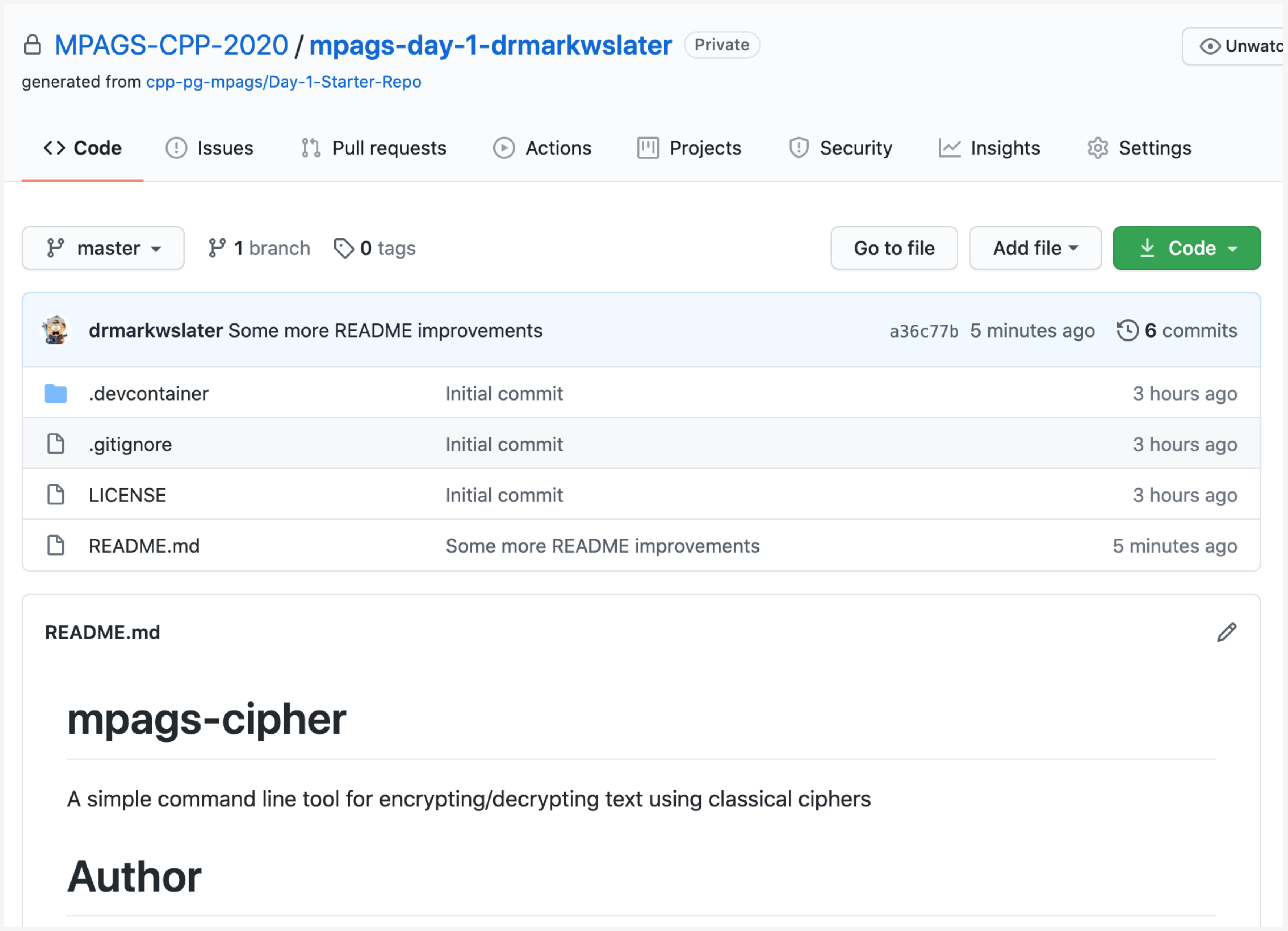
Notes

We won't cover Branching, but they can be thought of as separate sequences of commits. They're used to partition development, such as implementing new functionality, without interfering with others.

27: Viewing Changes on GitHub

After running `git push`, go back to your browser and refresh the page for your repository. You should now see that it's updated with the commits you've made up to the point you pushed.

It provides a very nice interface for browsing changes, so explore the viewing options and see how these map to the git command line arguments.



drmarkwslater Some more README improvements a36c77b 5 minutes ago 6 commits

.devcontainer	Initial commit	3 hours ago
.gitignore	Initial commit	3 hours ago
LICENSE	Initial commit	3 hours ago
README.md	Some more README improvements	5 minutes ago

README.md

mpags-cipher

A simple command line tool for encrypting/decrypting text using classical ciphers

Author

Notes

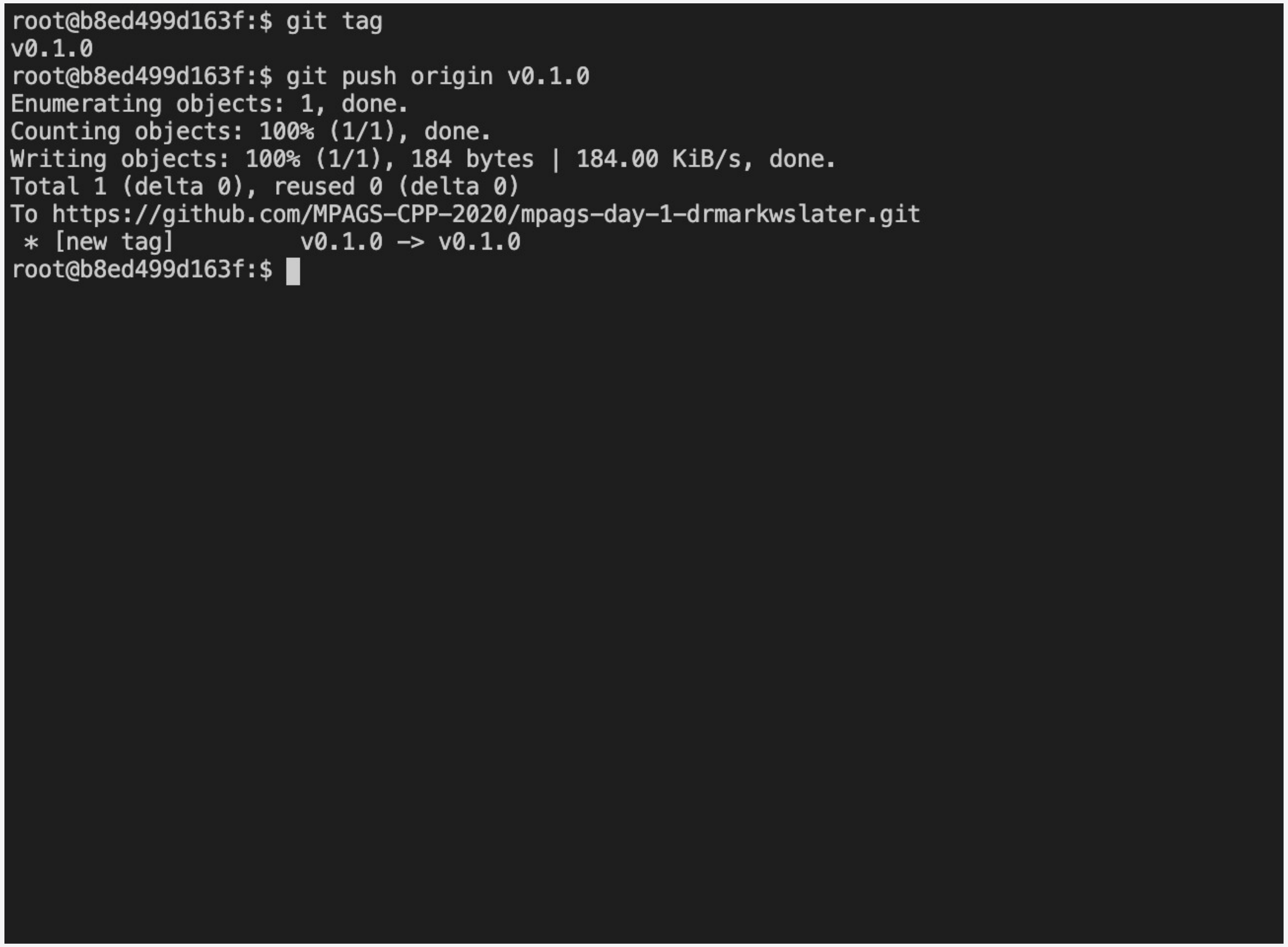
Now we can see the advantage of using Markdown format, as Github has rendered it nicely for us.

We could have used other markup styles as well.

28: Pushing Tags

Like any other repository "refspec", tags can be pushed to a remote repository. However, **`push`** does not **`push`** tags by default (you can see this as Github does not list your tag yet under "releases"). To do this, we have to either specify the tag name or use the **`--tags`** argument:

```
$ git push origin v0.1.0
```



```
root@b8ed499d163f:$ git tag
v0.1.0
root@b8ed499d163f:$ git push origin v0.1.0
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 184 bytes | 184.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To https://github.com/MPAGS-CPP-2020/mpags-day-1-drmarkwslater.git
 * [new tag]         v0.1.0 -> v0.1.0
root@b8ed499d163f:$ 
```

Notes

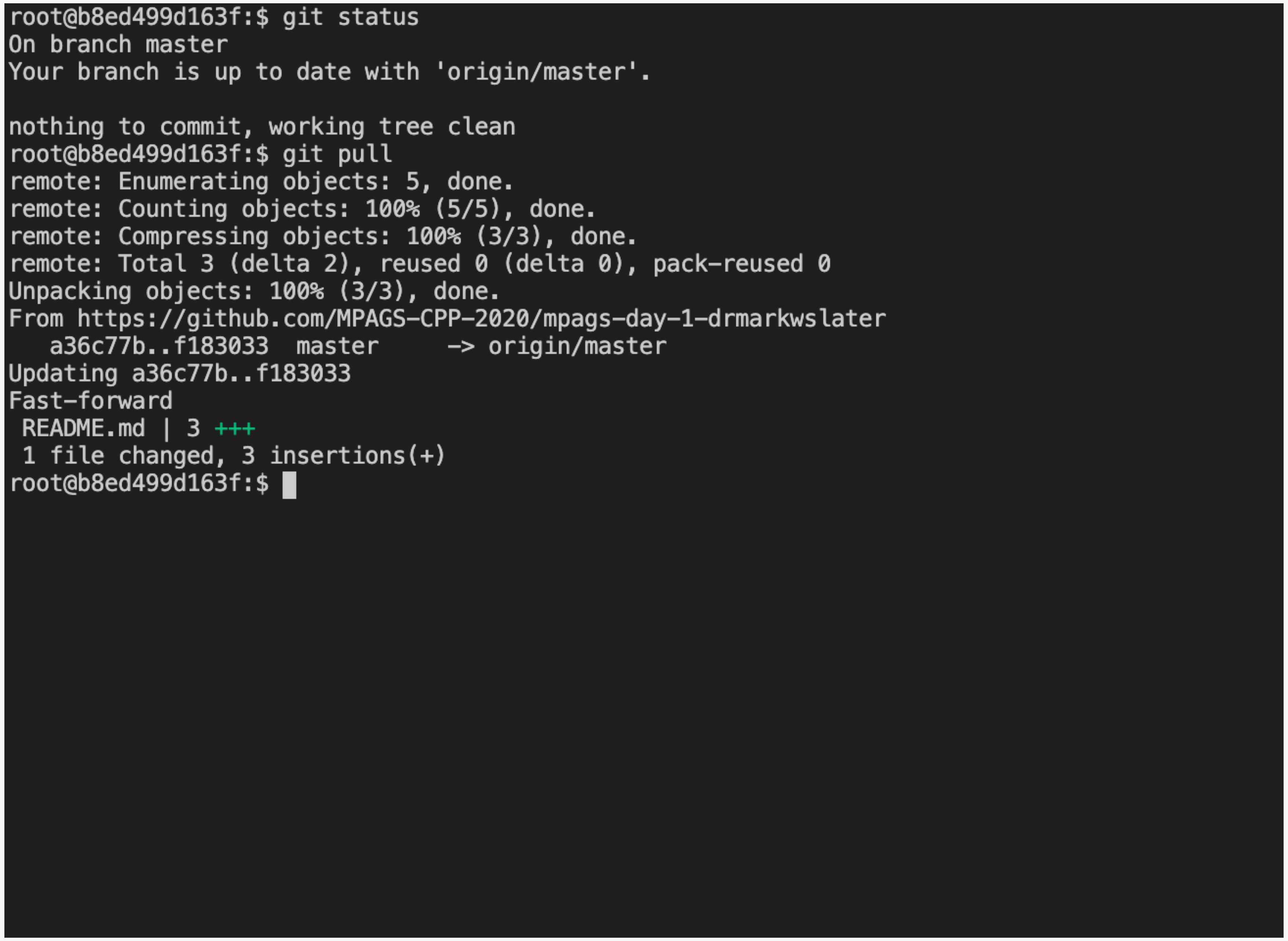
You should always push tags so they appear when others pull from your remote repo (including you!).

If you look on your github repository, you should see a '1' next to the "releases". Clicking on this will take you an interface where you can download a source archive for your code at the tag!

29: Pulling Changes from Github

At present we're only working with a single copy of our GitHub repo. Later you may obtain other copies, e.g. on your Laptop or another Location, so commits will get pushed to Github that other copies don't yet have. To update the current copy of the repo, we can use git pull:

```
$ git pull origin
```



```
root@b8ed499d163f:$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
root@b8ed499d163f:$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/MPAGS-CPP-2020/mpags-day-1-drmarkwslater
   a36c77b..f183033  master     -> origin/master
Updating a36c77b..f183033
Fast-forward
 README.md | 3 +++
 1 file changed, 3 insertions(+)
root@b8ed499d163f:$
```

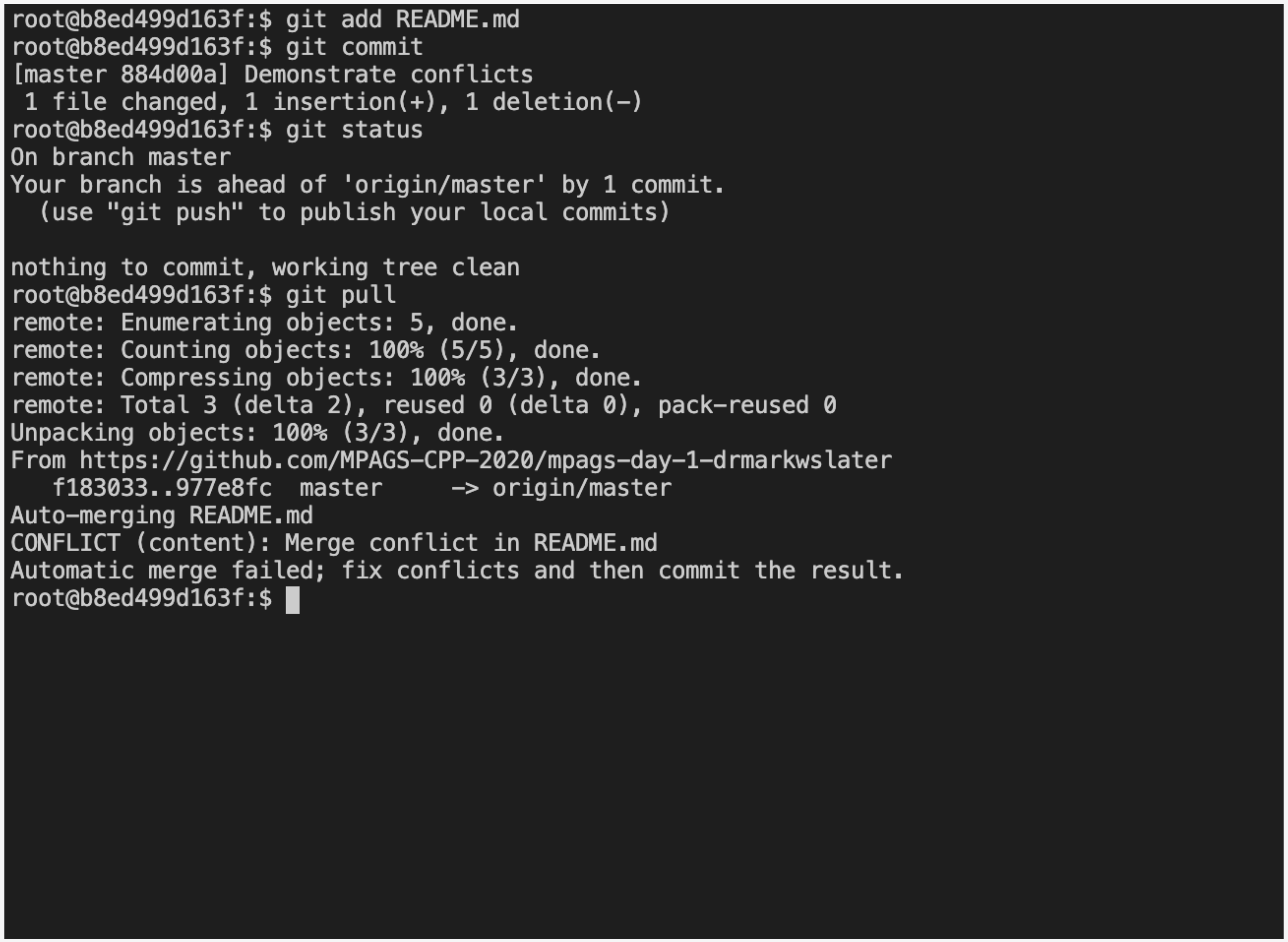
Try This

Using the web editor on github.com, make a change to your README file. Return to your main copy, then `git pull` to get those changes. Git will report what changes have been made.

Note that git pull is two steps: i) Fetch changes, ii) Merge changes.

30: Git Conflicts

Git is very smart at merging content changes in files, but it is not infallible - e.g. if a single word has changed on the same line, which one should be preferred? When conflicts occurs, git will warn us about them, and git status can be used to review them. Make an edit to README.md from github and then alter the same line in a different way in your local copy of the repo. After committing, pull the changes from the remote repository. `git status` will say there are conflicts that can't be automatically dealt with. We need to edit the file(s) to resolve the conflict, then add/commit just as we did for any other edit.



```
root@b8ed499d163f:$ git add README.md
root@b8ed499d163f:$ git commit
[master 884d00a] Demonstrate conflicts
 1 file changed, 1 insertion(+), 1 deletion(-)
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@b8ed499d163f:$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/MPAGS-CPP-2020/mpags-day-1-drmarkwslater
   f183033..977e8fc  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
root@b8ed499d163f:$ 
```

Notes

Conflicts are most common in collaborative development, but can also happen in your own work, so it’s worth learning how to resolve them!

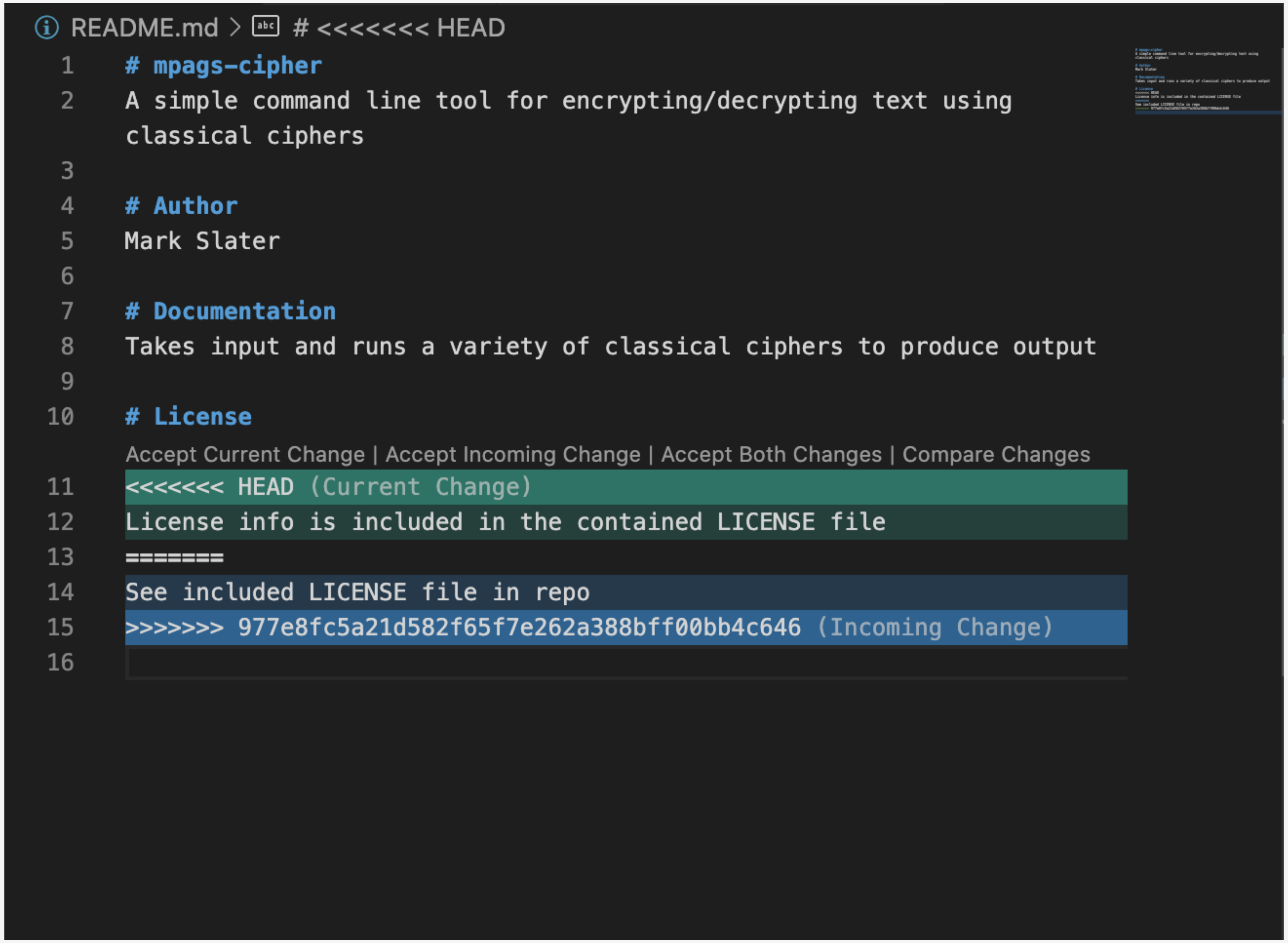
31: Viewing Conflicts

Git marks conflicts in files using a special markup block showing the conflicting content

```
<<<<<<< HEAD
local content
=======
remote content
>>>>>>> refspec
```

The HEAD block shows our local content.

'=======' divides the sections, and after this is shown the conflicting content. This ends with the "refspec" of the commit causing the conflict


README.md > # <<<<<<< HEAD
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
# mpags-cipher
A simple command line tool for encrypting/decrypting text using classical ciphers

# Author
Mark Slater

# Documentation
Takes input and runs a variety of classical ciphers to produce output

# License
<<<<<<< HEAD (Current Change)
License info is included in the contained LICENSE file
=======
See included LICENSE file in repo
>>>>>>> 977e8fc5a21d582f65f7e262a388bff00bb4c646 (Incoming Change)
```

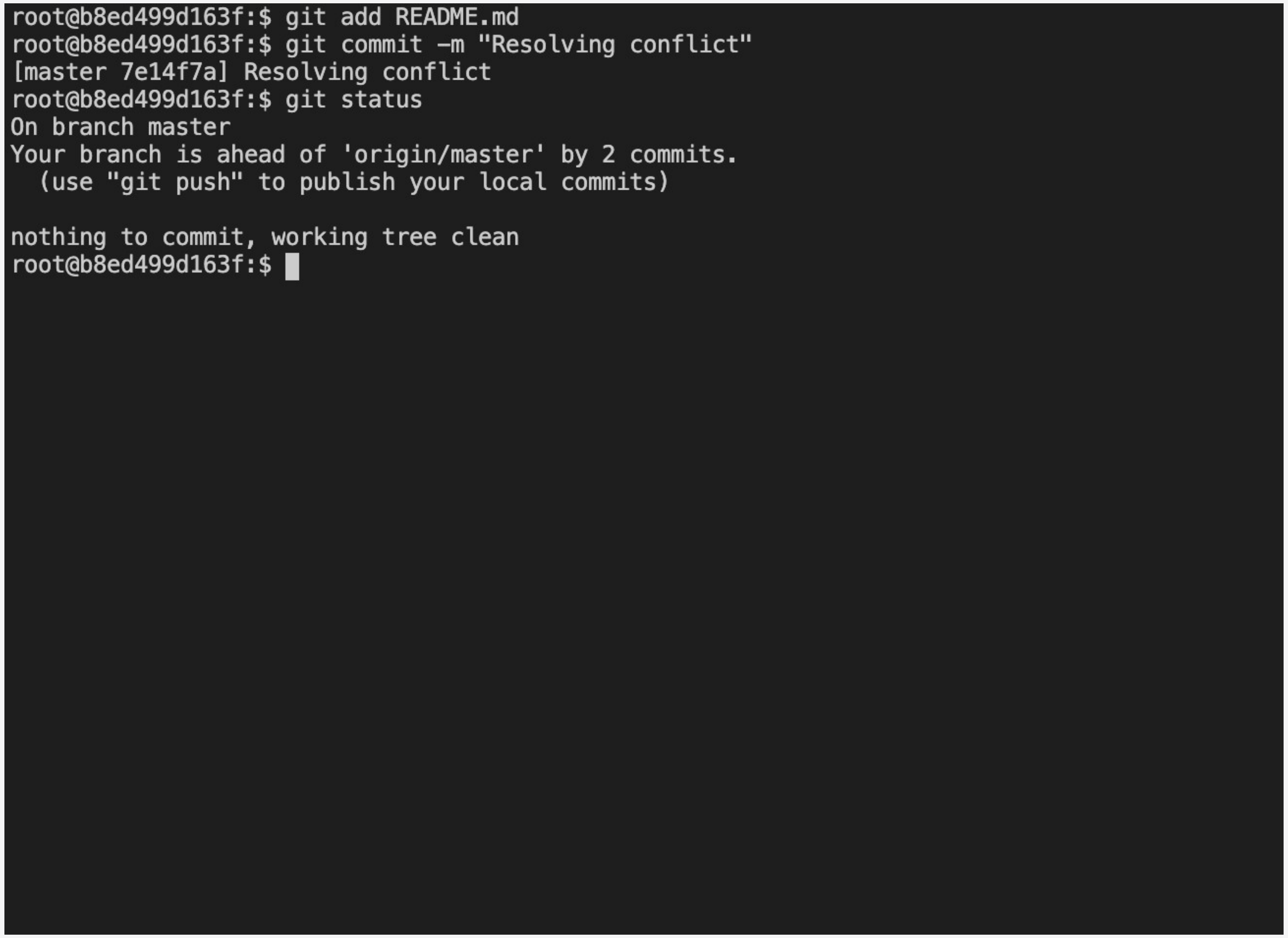
Notes

There may be more than one conflict block in a file!

This is where having a good syntax aware text editor helps. Add-ons are usually available specifically to highlight and handle git syntax like the conflict block. Visual Studio Code highlight them by default.

32: Resolving Conflicts

The content of the conflict has to be resolved manually, and is up to you (it may be a simple merge, choosing one or the other, or more complex). Once you've done this, remove the git markup (**`<<<<<<< HEAD`**, **`=======`** and **`>>>>>>> refspec`**) and save the file. Using **`git status`** will still show it as unmerged, but we can now use **`git add`** to stage it, followed by **`git commit`** to commit it and all the other changes brought in. Finally, push the changes to your repository!



```
root@b8ed499d163f:$ git add README.md
root@b8ed499d163f:$ git commit -m "Resolving conflict"
[master 7e14f7a] Resolving conflict
root@b8ed499d163f:$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@b8ed499d163f:$ 
```

Notes

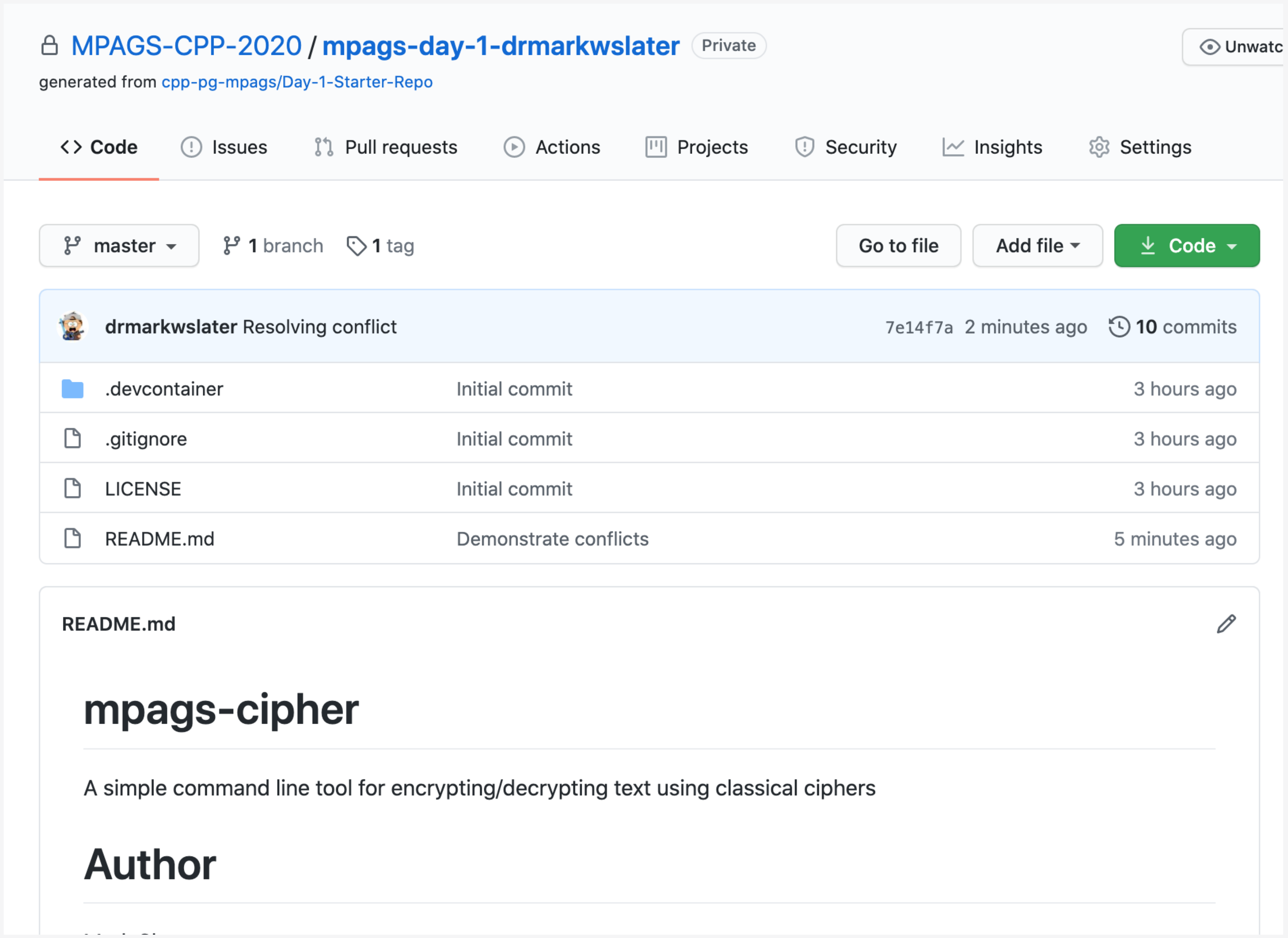
Don't be frightened of conflicts, git provides all the tools to help you resolve them!

Though you are unlikely to encounter conflicts in this course as you will be using your own repo, it's good to know how to deal with them.

33: And we’re done

That about covers the basic usage of git and github. All of the techniques are applicable to other VCSs you may work with, of particular importance being **the writing of good commit messages** so you (and your collaborators) know not only what changes were done, but why!

Through the course, remember to commit your work regularly when you have got something working (NEVER commit code that doesn’t work for you!). Push to GitHub regularly. Use tags to mark feature/task completion, again, the tag should work!


MPAGS-CPP-2020 / mpags-day-1-drmarkwslater Private

generated from cpp-pg-mpags/Day-1-Starter-Repo

drmarkwslater Resolving conflict 7e14f7a 2 minutes ago 10 commits

.devcontainer	Initial commit	3 hours ago
.gitignore	Initial commit	3 hours ago
LICENSE	Initial commit	3 hours ago
README.md	Demonstrate conflicts	5 minutes ago

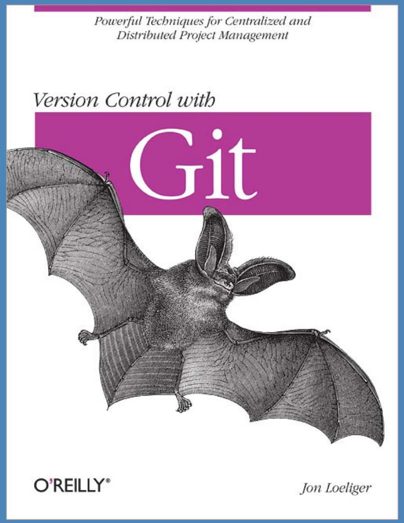
README.md

mpags-cipher

A simple command line tool for encrypting/decrypting text using classical ciphers

Author

Don’t Forget Resources


Powerful Techniques for Centralized and Distributed Project Management
Version Control with
Git
O'REILLY
Jon Loeliger

Homework Hand In with Git

- Through github classrooms, you will get a repository for each day.
- **Once you’re happy with your code from the end of each session, email us and we can have a look at your repos and leave comments!**