

# Version Control Walkthrough with

---




Mark Slater based on slides from Ben Morgan

# 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 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 walkthrough:
  - *Create a repo from the Github Assignment, get a working copy, 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*

FeaturesBusinessExploreMarketplacePricing

Search GitHubSign in or Sign up

# 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

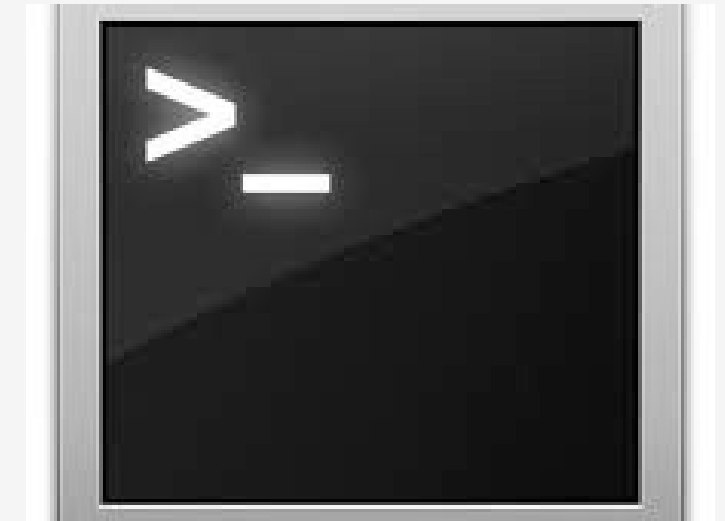
Password

Create a password

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

Sign up for GitHub

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



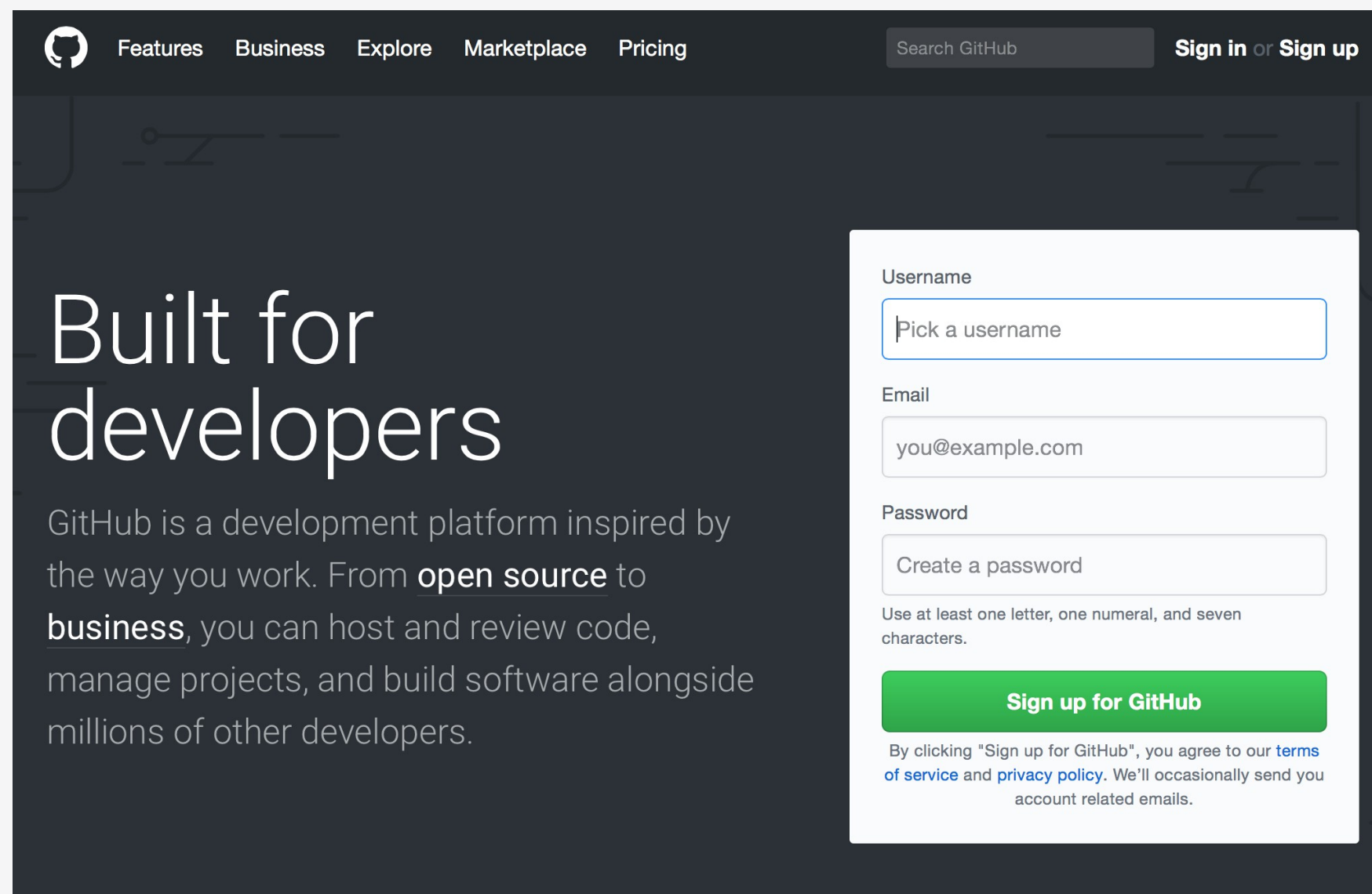
```
SyntaxHighlight.hpp + (~) - VIM
1 #ifndef SYNTAX_HIGHLIGHTING_EDITOR
2 #define SYNTAX_HIGHLIGHTING_EDITOR
3
4 class SyntaxHighlightingEditor {
5 public:
6     enum YourEditor {VIM, EMACS, KATE, GENIE};
7
8     void learn_features() const;
9     void be_productive() const;
10 };
11 #endif
12
```

syntaxHighlight.hpp[+][cpp]

# 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.



The screenshot shows the GitHub homepage with a dark theme. The navigation bar at the top includes links for Features, Business, Explore, Marketplace, and Pricing, along with a search bar and 'Sign in or Sign up' buttons. The main content area features the text 'Built for developers' and a description of GitHub as a development platform. A sign-up form is overlaid on the right side of the page, containing fields for Username, Email, and Password, a 'Sign up for GitHub' button, and a disclaimer about terms of service and privacy policy.

Username

Pick a username

Email

you@example.com

Password

Create a password

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

**Sign up for GitHub**

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 VCS.

# 3: 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. To create the one for Day one, go to the link on the course Day 1 page.

The screenshot shows the GitHub Classroom interface for the 'MPAGS C++ Course'. At the top, there's a dark header with 'GitHub Classroom' on the left and 'GitHub Education' with icons on the right. Below this is a green banner with the course name 'MPAGS C++ Course' and the handle '@cpp-pg-mpags'. A 'Manage classroom' button is on the right. A light blue notification bar states '"MPAGS C++ Day 1" has been created!'. Below this, the 'MPAGS C++ Day 1' assignment is listed as an 'Individual assignment' with an 'Assignment settings' button. A section titled 'Give this to your students' contains a URL: <https://classroom.github.com/assignment-invitations/dc874cb56fb5e05f41df8e45ba8c6185> and a 'Copy invitation link' button. At the bottom, a message says '"MPAGS C++ Day 1" does not have any repositories.' with a subtext: '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.

# 4: 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 (they will be new links for the other days!). This will be stored in the cpp-pg-mpags group but you're welcome to 'fork' it to your own account at a later date if you want!

## 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.

cpp-pg-mpags / **MPAGS-Code**

Unwatch 2

Star 0

Fork 0

Code

Issues 0

Pull requests 0

Projects 0

Wiki

Insights

Settings

No description, website, or topics provided.

Edit

Add topics

5 commits

9 branches

0 releases

1 contributor

MIT

Branch: Day1Branch

New pull request

Create new file

Upload files

Find file

Clone or download

This branch is even with master.

Pull request

Compare



drmarkwslater Added a LICENSE file

Latest commit 56a0909 on Oct 17, 2016



.gitignore

Added a .gitignore file

a year ago



LICENSE

Added a LICENSE file

a year ago



README.md

Added a basic README

a year ago



README.md

mpags-cipher



# 5: Command Line Git: Getting Help

With the repo in place we move back to command line git to get and work with it. Whilst we'll walk through the steps, if you need help, just ask! Also use `man git` for plenty of useful information. Git provides a fast command line help interface, so you can also just type

```
$ git help
```

```
EPSC02PN49MFVH8:~ slatermw$ git help
usage: git [--version] [--help] [-C <path>] [-c name=value]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
          <command> [<args>]

The most commonly used git commands are:
add          Add file contents to the index
bisect       Find by binary search the change that introduced a bug
branch       List, create, or delete branches
checkout     Switch branches or restore working tree files
clone        Clone a repository into a new directory
commit       Record changes to the repository
diff         Show changes between commits, commit and working tree, etc
fetch        Download objects and refs from another repository
grep         Print lines matching a pattern
init         Create an empty Git repository or reinitialize an existing one
log          Show commit logs
merge        Join two or more development histories together
mv           Move or rename a file, a directory, or a symlink
pull         Fetch from and integrate with another repository or a local branch
push         Update remote refs along with associated objects
rebase       Forward-port local commits to the updated upstream head
reset        Reset current HEAD to the specified state
rm           Remove files from the working tree and from the index
show         Show various types of objects
status       Show the working tree status
tag          Create, list, delete or verify a tag object signed with GPG
```

'git help -a' and 'git help -g' list available subcommands and some concept guides. See 'git help <command>' or 'git help <concept>' to read about a specific subcommand or concept.

```
EPSC02PN49MFVH8:~ slatermw$
```

## 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](https://git-scm.com).

# 6: Getting Your Repository

To obtain a local copy of our repository on GitHub, we **clone** it. This command takes the URL of the remote repository and a local directory where we want to create the **clone**. Various protocols for the URL are supported, including HTTPS and SSH

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

## Notes

We'll use the HTTPS protocol to clone as this is the easiest to use. The URL for cloning your GitHub repo can be found on the right hand side of its front page, as shown on the left.

cpp-pg-mpags / **MPAGS-Code**

Unwatch 2 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided.

Add topics

5 commits

9 branches

0 releases

1 contributor

MIT

Branch: Day1Branch

New pull request

Create new file

Upload files

Find file

Clone or download

This branch is even with master.

drmarkwslater Added a LICENSE file

Latest commit 56-0000 on Oct 17, 2016

.gitignore

Added a .gitignore file

a year ago

LICENSE

Added a LICENSE file

a year ago

README.md

Added a basic README

a year ago

README.md

mpags-cipher



# 7: Cloning your Repository

It's good practice to separate different projects, so we'll clone our repo under a dedicated directory. Open a terminal session and check that you're in the **HOME** directory and **cd** to it if not. Create a directory named **mpags** and **cd** into this. Finally, use **git clone** to clone your GitHub repository into a directory named **mpags-cipher.git**

The reason for a two-level directory structure will become apparent once we start to use CMake.

```
EPSC02PN49MFVH8:~ slatermw$ cd $HOME
EPSC02PN49MFVH8:~ slatermw$ mkdir mpags
EPSC02PN49MFVH8:~ slatermw$ cd mpags/
EPSC02PN49MFVH8:mpags slatermw$ ls -ltrah
total 0
drwxr-xr-x+ 111 slatermw 1311385079 3.7K 19 Oct 09:34 ..
drwxr-xr-x 2 slatermw 1311385079 68B 19 Oct 09:34 .
EPSC02PN49MFVH8:mpags slatermw$ git clone https://github.com/cpp-pg-mpags/mpags-c-course-day-1-d
rmarkwslater.git mpags-cipher.git
Cloning into 'mpags-cipher.git'...
remote: Counting objects: 15, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 15 (delta 0), reused 15 (delta 0), pack-reused 0
Unpacking objects: 100% (15/15), done.
Checking connectivity... done.
EPSC02PN49MFVH8:mpags slatermw$ ls -ltrah
total 0
drwxr-xr-x+ 111 slatermw 1311385079 3.7K 19 Oct 09:34 ..
drwxr-xr-x 3 slatermw 1311385079 102B 19 Oct 09:34 .
drwxr-xr-x 6 slatermw 1311385079 204B 19 Oct 09:34 mpags-cipher.git
EPSC02PN49MFVH8:mpags slatermw$
```

## Notes

You can keep the **mpags** directory somewhere other than in **HOME** if you wish.

**Make sure to use your github repo in the clone URL!**

Naming the directory holding the repository with a **“.git”** extension is not required, but helpful in marking its nature.

# 8: Repository Structure

Change into the `mpags-cipher.git` directory and run `ls -lart` 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

```
EPSC02PN49MFVH8:mpags slatermw$ ls -lart
total 0
drwxr-xr-x+ 111 slatermw 1311385079 3.7K 19 Oct 09:34 ..
drwxr-xr-x   3 slatermw 1311385079 102B 19 Oct 09:34 .
drwxr-xr-x   6 slatermw 1311385079 204B 19 Oct 09:34 mpags-cipher.git
EPSC02PN49MFVH8:mpags slatermw$ cd mpags-cipher.git/
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ ls -lart
total 24
drwxr-xr-x   3 slatermw 1311385079 102B 19 Oct 09:34 ..
-rw-r--r--   1 slatermw 1311385079  97B 19 Oct 09:34 README.md
-rw-r--r--   1 slatermw 1311385079 1.1K 19 Oct 09:34 LICENSE
-rw-r--r--   1 slatermw 1311385079 348B 19 Oct 09:34 .gitignore
drwxr-xr-x  13 slatermw 1311385079 442B 19 Oct 09:34 .git
drwxr-xr-x   6 slatermw 1311385079 204B 19 Oct 09:34 .
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ ls .git
HEAD      config     hooks      info        objects     refs
branches  description index       logs        packed-refs
```

## 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.

# 9: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is up-to-date with 'origin/Day1Branch'.
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.

# 10: 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>
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git config --global user.name "Mark Slater"
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git config --global user.email "mslater@cern.ch"
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git config --global core.editor vim
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git config --global color.ui true
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git config --global color.diff true
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git config --global color.status true
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git config --global -l
filter.lfs.clean=git lfs clean %f
filter.lfs.smudge=git lfs smudge %f
filter.lfs.required=true
user.name=Mark Slater
user.email=mslater@cern.ch
core.excludesfile=/Users/slatermw/.gitignore_global
core.editor=vim
difftool.sourcetree.cmd=pandoc --to=markdown
difftool.sourcetree.path=
mergetool.sourcetree.cmd=/Applications/SourceTree.app/Contents/Resources/opendiff-w.sh "$LOCAL"
"$REMOTE" -ancestor "$BASE" -merge "$MERGED"
mergetool.sourcetree.trustexitcode=true
color.ui=true
color.diff=true
color.status=true
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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!

# 11: 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.

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

## 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.

```
-uu-:---F1  README.md    All L1    (Fundamental)-----
Loading image...done
```



# 12: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is up-to-date with 'origin/Day1Branch'.
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")
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git add README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is up-to-date with 'origin/Day1Branch'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## Notes

### “staged” files

- Ready to be committed

### “unstaged” files

- Changed but not staged

### “untracked” files

- Not tracked by git yet

### “deleted” files

- Deleted by git and ready for removal

# 13: 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"
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is up-to-date with 'origin/Day1Branch'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git commit -m "Improve README"
[Day1Branch b2849a3] Improve README
 1 file changed, 2 insertions(+), 1 deletion(-)
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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).

# 14: 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.

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ emacs README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' 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")
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git add README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git commit -m "Add section for documentation. To be completed"
[Day1Branch 8574ec4] Add section for documentation. To be completed
 1 file changed, 1 insertion(+)
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 2 commits.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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!

# 15: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ emacs README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' 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")
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git add README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git reset HEAD README.md
Unstaged changes after reset:
M   README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.

# 16: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ emacs mpags-cipher.cpp
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 3 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)
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git add mpags-cipher.cpp
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 3 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

EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git commit -m "Add placeholder for main program"
[Day1Branch a0e8dfd] Add placeholder for main program
 1 file changed, 1 insertion(+)
 create mode 100644 mpags-cipher.cpp
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.



# 17: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 4 commits.
(use "git push" to publish your local commits)
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ ls
LICENSE      README.md    README.md~   mpags-cipher.cpp
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git rm mpags-cipher.cpp
rm 'mpags-cipher.cpp'
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 4 commits.
(use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    mpags-cipher.cpp

EPSC02PN49MFVH8:mpags-cipher.git slatermw$ ls
LICENSE      README.md    README.md~
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git commit -m "Remove main program temporarily"
[Day1Branch 389e891] Remove main program temporarily
 1 file changed, 1 deletion(-)
 delete mode 100644 mpags-cipher.cpp
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ ls
LICENSE      README.md    README.md~
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.

# 18: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git log
commit 389e891221e00346b669c835c3956cb67fe1b76b
Author: Mark Slater <mslater@cern.ch>
Date: Thu Oct 19 11:07:52 2017 +0100

    Remove main program temporarily

commit a0e8dfd1f0462e2c42a4727271b448f051e73685
Author: Mark Slater <mslater@cern.ch>
Date: Thu Oct 19 11:04:54 2017 +0100

    Add placeholder for main program

commit b1aa13d01aec7e92d9fa2ad1c59cfc594822a1b6
Author: Mark Slater <mslater@cern.ch>
Date: Thu Oct 19 11:03:58 2017 +0100

    Additional notes in README

commit 8574ec432ee4c4ab3cba7db8fc1b9df2c6b42be7
Author: Mark Slater <mslater@cern.ch>
Date: Thu Oct 19 10:09:26 2017 +0100

    Add section for documentation. To be completed

commit b2849a36a95606fffbcb8715abc9266f5ddcd203f
Author: Mark Slater <mslater@cern.ch>
Date: Thu Oct 19 10:04:36 2017 +0100

    Improve README

EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.

# 19: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ emacs README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git diff
diff --git a/README.md b/README.md
index ee4f93c..574420a 100644
--- a/README.md
+++ b/README.md
@@ -2,4 +2,7 @@
A simple command line tool for encrypting/decrypting text using classical ciphers. Created for
the MPAGS C++ course.

# Documentation
-Takes input and runs a variety of classical ciphers on to produce the output
+To be completed in Week 3
+
+# Author
+Mark Slater
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.

# 20: Changes between Commits

As you'll have seen in using `git log`, git labels commits using a 40 character hash code (cf subversion's revision numbers). 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 8574 b1aa
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git diff 8574 b1aa
diff --git a/README.md b/README.md
index 79da959..ee4f93c 100644
--- a/README.md
+++ b/README.md
@@ -2,3 +2,4 @@
 A simple command line tool for encrypting/decrypting text using classical ciphers. Created for
 the MPAGS C++ course.

# Documentation
+Takes input and runs a variety of classical ciphers on to produce the output
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.

# 21: 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.



# 22: 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:

```
# 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

# Fortran module files
*.mod

# Compiled Static libraries
*.lai
*.la

-uu-:---F1 .gitignore Top L1 (Fundamental)-----
Loading image...done
```

## 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`

# 23: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 6 commits.
(use "git push" to publish your local commits)
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git tag
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git tag -a v0.1.0 -m "mpags-cipher 0.1.0: End of git walkthrough"
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git tag
v0.1.0
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git show v0.1.0
tag v0.1.0
Tagger: Mark Slater <mslater@cern.ch>
Date: Thu Oct 19 13:34:41 2017 +0100

mpags-cipher 0.1.0: End of git walkthrough

commit 03b2ef4f603a4804194938cb5fd390ea3836ec39
Author: Mark Slater <mslater@cern.ch>
Date: Thu Oct 19 12:22:00 2017 +0100

    Updated the README file

diff --git a/README.md b/README.md
index ee4f93c..574420a 100644
--- a/README.md
+++ b/README.md
@@ -2,4 +2,7 @@
A simple command line tool for encrypting/decrypting text using classical ciphers. Created for the MPAGS C++ course.
```

## 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.

# 24: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 6 commits.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git remote -v
origin  https://github.com/cpp-pg-mpags/mpags-c-course-day-1-drmarkwslater.git (fetch)
origin  https://github.com/cpp-pg-mpags/mpags-c-course-day-1-drmarkwslater.git (push)
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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.

# 25: Pushing your Repository to Github

To send our changes to a remote repository, 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git push
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 16, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (16/16), 1.57 KiB | 0 bytes/s, done.
Total 16 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 1 local object.
To https://github.com/cpp-pg-mpags/mpags-c-course-day-1-drmarkwslater.git
   56a0909..03b2ef4  Day1Branch -> Day1Branch
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## Notes

We use the https protocol, so git will ask you for your github user/pass to push!

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.

# 26: 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.

The screenshot shows the GitHub interface for a repository named 'mpags-c-course-day-1-drmarkwslater' under the user 'cpp-pg-mpags'. The repository was created by GitHub Classroom. It features 11 commits, 1 branch, 0 releases, and 1 contributor. The 'Day1Branch' is selected. A table of recent commits is shown, including updates to the README file and the addition of .gitignore and LICENSE files. The README content is visible below, titled 'mpags-cipher', describing it as a simple command line tool for encrypting/decrypting text using classical ciphers.

Commit	Message	Time
drmarkwslater	Updated the README file	Latest commit 03b2ef4 2 hours ago
	Added a .gitignore file	a year ago
	Added a LICENSE file	a year ago
	Updated the README file	2 hours ago

## mpags-cipher

A simple command line tool for encrypting/decrypting text using classical ciphers. Created for the MPAGS C++ course.

## 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.



# 27: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git tag  
v0.1.0  
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git push origin v0.1.0  
Counting objects: 1, done.  
Writing objects: 100% (1/1), 183 bytes | 0 bytes/s, done.  
Total 1 (delta 0), reused 0 (delta 0)  
To https://github.com/cpp-pg-mpags/mpags-c-course-day-1-drmarkwslater.git  
* [new tag]          v0.1.0 -> v0.1.0  
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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!

# 28: 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
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is up-to-date with 'origin/Day1Branch'.
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git pull
remote: Counting objects: 3, 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/cpp-pg-mpags/mpags-c-course-day-1-drmarkwslater
   03b2ef4..e46a49f  Day1Branch -> origin/Day1Branch
Updating 03b2ef4..e46a49f
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## Try This

`git clone` another copy of your github repo, make some commits, then `push` to github. 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.

# 31: 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.

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ emacs README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git add README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git commit
[Day1Branch 6be1cd7] Demonstrate conflicts
1 file changed, 1 insertion(+), 1 deletion(-)
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch and 'origin/Day1Branch' have diverged,
and have 1 and 1 different commit each, respectively.
(use "git pull" to merge the remote branch into yours)
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git pull
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## 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!

# 32: Viewing Conflicts

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

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

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

```
# mpags-cipher
A simple command line tool for encrypting/decrypting text using classical ciphers. Created for \
the MPAGS C++ course.

<<<<<< HEAD
# How to use MPAGS Cipher
=====
# How to use MPAGS-CIPaoslmksca
>>>>>> e23388288944f6d40aef76e95c87802b34f11156

# Documentation
To be completed in Week 3

# Author
Mark Slater

-uu-:---F1  README.md      All L1      (Fundamental)-----
Loading image...done
```

## 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.

# 33: 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!

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ emacs README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git add README.md
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git commit -m "Resolving conflict"
[Day1Branch 0f10c0d] Resolving conflict
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ git status
On branch Day1Branch
Your branch is ahead of 'origin/Day1Branch' by 2 commits.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

## Notes

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

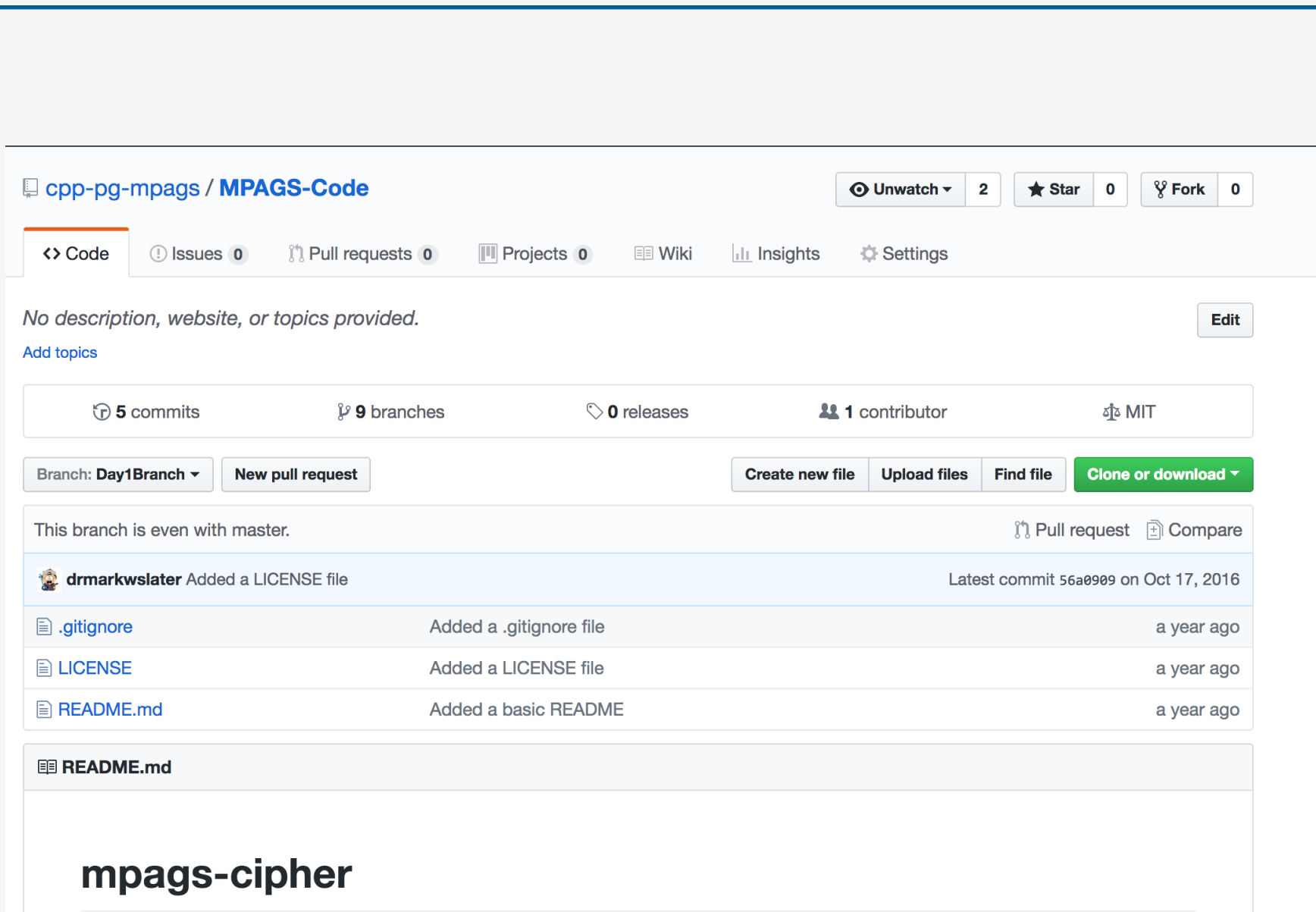
Though you are unlikely to encounter conflicts in this course as you will be using your own repo, it's good to know how to deal with them.



# 34: 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!



The screenshot shows a GitHub repository page for 'cpp-pg-mpags / MPAGS-Code'. At the top, there are buttons for 'Unwatch', 'Star' (0), and 'Fork' (0). Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Insights', and 'Settings'. The main content area shows 'No description, website, or topics provided.' with an 'Edit' button. Below this, statistics show '5 commits', '9 branches', '0 releases', '1 contributor', and 'MIT' license. A 'Branch: Day1Branch' dropdown is present, along with buttons for 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. A commit history table lists three commits: 'drmarkwslater Added a LICENSE file' (latest commit 56a0909 on Oct 17, 2016), '.gitignore Added a .gitignore file' (a year ago), 'LICENSE Added a LICENSE file' (a year ago), and 'README.md Added a basic README' (a year ago). Below the table is a 'README.md' section with the text 'mpags-cipher'.

## Don't Forget Resources



**github**  
SOCIAL CODING

# 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 two day session, email us and we can have a look at your repos and leave comments!