# An Introduction to GitHub and Git shell

Andrew Edwards

October 28, 2021

# Definitions

- GitHub – a website that hosts your repositories so that you can easily share code and collaborate with colleagues.
- Repository – essentially a directory containing all your files for a project (plus some files that Git uses). Also used to refer to the GitHub page for your project.
- Git – a program that allows you to efficiently save ongoing versions of your files ('version control') and easily interact with GitHub.

Basically, you work on your files in a repository on your computer, use Git on your computer when you are happy with some changes, and share the files easily on GitHub.

# Contents



I will demonstrate the basics in this video. For each slide I will demonstrate what to do on the screen (e.g. some commands to type) and then leave the slide up. You should pause the video and then repeat what was done by following the instruction on the slide.

There are also a couple of exercises.

The text in the slides is repeated (for easier reference) in the main notes, and there is also a link there to access these slides.

# Creating a new repository

- Sign into your GitHub account, click on the *Repositories* tab, and press the *New* button.
- Give your repository a name. Let's call it `test`.

# Creating a new repository

- Sign into your GitHub account, click on the *Repositories* tab, and press the *New* button.
- Give your repository a name. Let's call it `test`.
- Check *Initialize this repository with a README*.
- Leave *'Add .gitignore'* and *'Add a license'* set to *None*.
- Click *Create repository*.

You now have a new repository on the GitHub website. Next we will `clone` it onto your computer.

# Cloning your new repository

- Copy the full URL (web address) of your test repository.
- Open the Git shell and navigated to your `C:/github` directory (or whatever you called it when you created it in the setup instructions – it's the place you are going to save all your Git repositories).
- run the following command to *clone* your repository:

  ```
  git clone URL
  ```

  where `URL` is the url of your newly created repository (paste should work).

You should now have a subdirectory called `github/test` on your computer.
In Git shell, change to that directory:

```
cd test
```

So 'clone' is Git speak for copying something from GitHub onto your local computer. This example has just one file (the README). But the process is the same for a repository with multiple files and multiple directories (the structure is fully preserved).

# Windows only: Storing your credentials

When you are using the Git shell for the very first time on Windows, issue the following command:

```
git config --global credential.helper wincred
```

This means that you don't have to repeatedly enter your GitHub password (just do it when you are first prompted).

# Creating and committing a new file

- Create a new file, *newFile.txt*, in the `github/test` directory.

# Creating and committing a new file

- Create a new file, *newFile.txt*, in the `github/test` directory.
- Add a line of text at the start of the file and save it.

# Creating and committing a new file

- Create a new file, *newFile.txt*, in the `github/test` directory.
- Add a line of text at the start of the file and save it.
- Check the status of the (`test`) repository by typing `git status` in the Git shell.

# Creating and committing a new file

- Create a new file, *newFile.txt*, in the `github/test` directory.
- Add a line of text at the start of the file and save it.
- Check the status of the (`test`) repository by typing `git status` in the Git shell.
- It should say that you have an 'Untracked file' called *newFile.txt*. You want to tell Git to start tracking it, by using the `git add` command:
  `git add newFile.txt`

# Creating and committing a new file

- Create a new file, *newFile.txt*, in the `github/test` directory.
- Add a line of text at the start of the file and save it.
- Check the status of the (`test`) repository by typing `git status` in the Git shell.
- It should say that you have an 'Untracked file' called *newFile.txt*. You want to tell Git to start tracking it, by using the `git add` command:
  `git add newFile.txt`
- Type `git status` again. (If on MacOS you may see mention of a *.DS_Store* file – ignore that for now).
- You should see that the file is listed as a 'new file' under 'Changes to be committed'.
- Let's now 'commit' it:
  `git commit -a -m "Add newFile.txt file."`
  The commit message should be a useful message saying what the commit encapsulates.

# Creating and committing a new file

- Create a new file, *newFile.txt*, in the `github/test` directory.
- Add a line of text at the start of the file and save it.
- Check the status of the (`test`) repository by typing `git status` in the Git shell.
- It should say that you have an 'Untracked file' called *newFile.txt*. You want to tell Git to start tracking it, by using the `git add` command:
  `git add newFile.txt`
- Type `git status` again. (If on MacOS you may see mention of a *.DS_Store* file – ignore that for now).
- You should see that the file is listed as a 'new file' under 'Changes to be committed'.
- Let's now 'commit' it:
  `git commit -a -m "Add newFile.txt file."`
  The commit message should be a useful message saying what the commit encapsulates.
- Push the commit to GitHub: `git push`

# Creating and committing a new file

- Create a new file, *newFile.txt*, in the `github/test` directory.
- Add a line of text at the start of the file and save it.
- Check the status of the (`test`) repository by typing `git status` in the Git shell.
- It should say that you have an 'Untracked file' called *newFile.txt*. You want to tell Git to start tracking it, by using the `git add` command:
  `git add newFile.txt`
- Type `git status` again. (If on MacOS you may see mention of a *.DS_Store* file – ignore that for now).
- You should see that the file is listed as a 'new file' under 'Changes to be committed'.
- Let's now 'commit' it:
  `git commit -a -m "Add newFile.txt file."`
  The commit message should be a useful message saying what the commit encapsulates.
- Push the commit to GitHub: `git push`
- Check (refresh) the GitHub webpage and see your commit and the uploaded file.

# What just happened?

We just used three of the main Git commands:

- `git add <filename>` – tell Git to start keeping track of changes to this file. You only need to tell Git this once.
- `git commit -a -m "Message."` – committing your changes, which means tell Git you are happy with your edits and want to save them.
- `git push` – this sends your commit to the GitHub website.

You always have your files stored *locally* on your computer (as usual), even if you don't `add` them or `commit` changes.

When you push to GitHub then your colleagues can easily `fetch` (retrieve) them.

# Keyboard aliases (shortcuts)

Now, `git commit -a -m "Message."` is a bit much to type, so we have an alias for it:

```
git com "Message."
```

This is defined in the *.gitconfig* file you installed in the 'git-setup' instructions.

The `-a` means 'commit all changes of files that Git is tracking', and `-m` is to include a message. Since we usually want to do both of these, `git com "Message."` is a useful shortcut. But it is important to realise it is an alias if searching online for help.
Similarly:
`git s` – for `git status`
`git p` – for `git push`
`git d` – for `git diff`
From now on we will use the aliases.

# Edit *Readme.md*

Edit the *Readme.md* file. Add some simple comments describing the project such as: "A test repository for learning Git."

# Edit *Readme.md*

Edit the *Readme.md* file. Add some simple comments describing the project such as: "A test repository for learning Git."

Look over the changes, commit them, and push them to your GitHub repository:

`git s`

`git diff` or the alias `git d` – this gives a simple look at the differences between the last committed version and your current version (of all files; only one in this case)

`git com "Initial edit of Readme.md"`

`git p`

Refresh your GitHub web page and you should see your text (the *Readme.md* file is what is shown on the main page of your repo).

# Exercise 1: create, edit, and commit *simpleText.txt*

1. Create a text file `simpleText.txt` in your local `test` repository. Add a line of text at the start and save it.

# Exercise 1: create, edit, and commit *simpleText.txt*

1. Create a text file `simpleText.txt` in your local `test` repository. Add a line of text at the start and save it.
2. Predict what `git s` will tell you, *then* type it in the Git shell to check.

# Exercise 1: create, edit, and commit *simpleText.txt*

1. Create a text file `simpleText.txt` in your local `test` repository. Add a line of text at the start and save it.
2. Predict what `git s` will tell you, *then* type it in the Git shell to check.
3. Add the file to the repository using the git commands:

   `git add simpleText.txt`

   `git s` – not necessary but useful to check you understand what is changing before you commit

   `git com "Adding simpleText.txt"`

   `git p`

# Exercise 1: create, edit, and commit *simpleText.txt*

1. Create a text file `simpleText.txt` in your local `test` repository. Add a line of text at the start and save it.
2. Predict what `git s` will tell you, *then* type it in the Git shell to check.
3. Add the file to the repository using the git commands:
   `git add simpleText.txt`
   `git s` – not necessary but useful to check you understand what is changing before you commit
   `git com "Adding simpleText.txt"`
   `git p`
4. Add some text to `simpleText.txt`, then `git com "Message"` and `git p`.

# Exercise 1: create, edit, and commit *simpleText.txt*

1. Create a text file `simpleText.txt` in your local `test` repository. Add a line of text at the start and save it.
2. Predict what `git s` will tell you, *then* type it in the Git shell to check.
3. Add the file to the repository using the git commands:
   `git add simpleText.txt`
   `git s` – not necessary but useful to check you understand what is changing before you commit
   `git com "Adding simpleText.txt"`
   `git p`
4. Add some text to `simpleText.txt`, then `git com "Message"` and `git p`.
5. Repeat this a few times to get the hang of it, while intermittently doing `git s` and `git d` to understand what's changing.

# Exercise 1: create, edit, and commit *simpleText.txt*

1. Create a text file `simpleText.txt` in your local `test` repository. Add a line of text at the start and save it.
2. Predict what `git s` will tell you, *then* type it in the Git shell to check.
3. Add the file to the repository using the git commands:
   `git add simpleText.txt`
   `git s` – not necessary but useful to check you understand what is changing before you commit
   `git com "Adding simpleText.txt"`
   `git p`
4. Add some text to `simpleText.txt`, then `git com "Message"` and `git p`.
5. Repeat this a few times to get the hang of it, while intermittently doing `git s` and `git d` to understand what's changing.
6. Keep an eye on your commits by refreshing the GitHub page.

# Adding multiple files at once – slide 1

Often you add multiple files in a new directory. When you run `git s`, you will see a large list of *Untracked files*. They can be added at once by simply adding the whole directory.

- Create a new directory to your `test` repository, using your normal method. Call it `new-stuff`.
- Add a few new test files to that directory called `test1.txt`, `test2.txt`, etc. Put some example text in one or more of them if you want.
- On the command line, check the status:
  `git s`
- You will see a listing showing the `new-stuff` directory in *Untracked files*.
- To add all the new files in preparation for a commit, issue the command:
  `git add new-stuff/`

Continued...

# Adding multiple files at once – slide 2

- Check the status of the repository again: `git s`
- It will now show all files in *Changes to be committed*
- Commit the changes:
  `git com "Added new-stuff directory."`
- Push the changes to GitHub:
  `git p`
- Check your GitHub webpage and see your commit and that the files have been uploaded.
- That works no matter how many files are in your `new-stuff` directory.

Exercise 2: Repeat the above with more files, to practice creating multiple files in a directory and committing your changes.

# Adding multiple files at once – slide 3

- To add multiple files with similar names you can use the wildcard `*` symbol.
- You just added (told Git to keep track of) the new files in your `new-stuff/` directory.
- If you add more new files to that directory, you will have to tell Git to track those also (since they are new – you haven't told Git about them yet).
- Say you have 10 new files in `new-stuff/` called `idea1.txt`, `idea2.txt`, ..., `idea10.txt`.
- Instead of typing

  ```
  git add new-stuff/idea1.txt
  git add new-stuff/idea2.txt
  ```

  etc. (note the `new-stuff/` directory name there) you can just use the wildcard `*`:

  ```
  git add new-stuff/idea*.txt
  ```

  (or even just `git add new-stuff/*.txt`, or `git add new-stuff/*.*`).
- No need to do this now, but this is useful to know.

# The *.gitignore* file

But what if you don't want to add all the files that you create?

Each repository can have a *.gitignore* file, in the root directory of the repository. Such a file has names of files (such as `my-secret-notes.txt`) or wildcard names (such as `*.pdf` or `*.doc`) that will be completely ignored by Git.

When sharing a repository with others, you want to share your *code* (for example, R or Python code) and maybe data, but generally *not* share the output (such as figures that the code generates; more on this later). For reproducible research your colleague (or anyone) should be able to run your code to generate the results.

Some programs you run may make temporary files that don't need to be tracked by Git, the names of which should also be included in your *.gitignore*.

# The *.gitignore* file

When sharing code or collaborating you want to keep your repository as clean as possible and not clutter it up with files that other people don't need.

So when you run `git s` and see untracked files that you don't want to be tracked, add them (or a suitable wildcard expression) to your *.gitignore* file so that they are not added inadvertently.

This will also simplify your workflow (you don't need to keep being reminded that you have untracked files).

If you are on MacOS and you find that directories have a *.DS_Store* file in them, then create (and add and commit) a *.gitignore* file with *.DS_Store* as a line.

# Thoughts/hints regarding commit messages

What to write in `git com "Message"`?
Ideally:

- Want to describe *what* (and sometimes *why*) you did something.
- The *how* is not needed since that will be explained by the actual changes in the code.
- Message should be informative for collaborators (including your future self).

Bad:

```
git com "Tweaked function."
```

Good:

```
git com "Allow plot.biomass() to use extra colours."
```

A good rule of thumb is to complete the sentence "This commit will ...".

# That's the basics

You have now learnt the basics of using Git. By creating a public repository on GitHub you can now release your code to the world!

You can also choose the *private repository* option when creating a repository, so that you can control who can see it.

Next we will show how to collaborate with colleagues, which is where the usefulness of Git will become more apparent.