

Version Control Using Git

Ade Fewings, Aaron Owen

Bangor University

Updated: 2022-02-20

eResearch

- Supercomputing Wales
 - Available to researchers at Bangor
- Research Software Engineers
- Collate expert knowledge into an open and shared centralised repository
 - Yammer
 - Github
 - Workshops
 - Projects
 - Acknowledgements

Training Workshops

- Introduction to the Linux Shell
- **Version Control Using Git**
 - <https://swcarpentry.github.io/git-novice/>
- Programming Principles and Practice using Python
- Advanced Python
- Parallel Processing in Python
- Machine Learning with Python*

See and discuss on the Yammer group.

Suggestions for new training welcome.

Theory: Version Control Systems (VCS)

- A program or set of programs that tracks changes to a collection of files.
- Can recall specific versions of files.
 - No need for *file-new*, *file-new-new*, *file-old*, *file-latest*, *file-working*
- Not restricted to software development.
 - Often used to write books, papers and online tutorials.
- Can be local, centralised and decentralised.
- Enables several team members to work on a project, even on the same file at the same time without affecting each other's work.
 - Users are notified when there is a possibility that changes may conflict.

Theory: Version Control Systems

- Users can revert to a previous version - Unlimited undo.
- See all the changes made to a project, when the changes were made and who made them.
- Include messages on each save (commit) to explain a reason behind it.
- Create branches, where changes to code can be tested without affecting the main branch.
 - Can merge branches back into the main branch.
- Many different version control systems:
 - CVS, SVN, **Git**, Mercurial

Theory: Version Control Systems

- Initialise the VCS within a directory.
- Create or copy a base version of a file.
- VCS will record changes you make each step of the way.
- Are able to save snapshots and include a message.
- Can rewind to the base file and playback each change.



Theory: Version Control Systems

- Think of the changes as separate from the file itself.
- Different sets of changes can result in different versions of the file.
- Two users can make independent sets of changes on the same file.



Theory: Version Control Systems

- If users make changes to different parts of a file.
 - You can incorporate two sets of changes into the same base file
- If users make changes to the same part of a file.
 - This will result in a *conflict* and needs to be reviewed.

Git and GitHub

Git

- *Git* is a command-line tool but Git also integrates with a wide variety of graphical development environments.
 - Free and open-source version control system, originally developed by Linus Torvalds in 2005.
 - We include *command line*, *RStudio* and *Visual Studio Code* practical elements in this course. Please follow which you need.

Github

- *GitHub* is a Git repository hosting service, but it adds many of its own features.
 - Web-based graphical interface.
 - Access control and several collaboration features, such as wikis and basic task management tools for every project.

Create a GitHub account

Sign up

- Visit <https://github.com/>
- Sign up
- Verify account

Create a Personal Access Token

- <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>
- From your GitHub account
 - Settings
 - Developer settings
 - Personal access token
 - Generate new token
 - Fill in the form
 - Click Generate token
 - Copy the token - will be used shortly

Command Line: Configure Git

- Open *Git Bash* and run the configure the following:
 - Set your username.
 - `git config --global user.name "issa16"`
 - Set your email address.
 - `git config --global user.email "issa16@bangor.ac.uk"`
 - Set the default branch for projects
 - `git config --global init.defaultBranch main`
 - Username will be linked to your Git activity.
 - `--global` For every Git project, use these settings

Theory: Line Endings

- Different Operating Systems (OS) using different characters to represent to the end of a line in a file.
- Git uses these characters to compare files, it may cause unexpected issues when editing a file on different machines.
- `dos2unix` - Converts plain text files in DOS format to UNIX format.

Linux and macOS

- `git config --global core.autocrlf input`

Windows

- `git config --global core.autocrlf true`

Command Line: Check Git Config

```
git config --list
```

- You can change your configuration as many times as you want.

Git Config Help

- `git config --help`
- Press **q** to exit, **Spacebar** Go to next screen, **b** Go to the previous screen

Summary

Use `git config` to change the Git configuration and preferences.

RStudio and Git

- Version control doesn't have to be at the command-line. It integrates nicely with Development Environments as well.
- To use with RStudio, you must have these installed:
 - **R**
 - <https://cran.r-project.org>
 - **RStudio**
 - <https://rstudio.com/products/rstudio/download/>
 - **Git**
 - <https://git-scm.com/downloads>
- Once you have a repository in Github (creating a new one through Github's web interface is the easiest way, if necessary) you can check it out in RStudio and perform the full Version Control cycle therein.

Visual Studio (VS) Code and Git

- To use Visual Studio Code, you must have these installed:
 - **Git**
 - <https://git-scm.com/downloads>
 - **Visual Studio Code**
 - <https://code.visualstudio.com/>
- Optional package to work with GitHub
 - **GitHub Pull Requests and Issues extension**
 - <https://code.visualstudio.com/docs/editor/github>
- Once you have a repository in Github (creating a new one through Github's web interface is the easiest way, if necessary) you can check it out in Visual Studio Code and perform the full Version Control cycle therein.
- Tutorial: <https://code.visualstudio.com/docs/editor/github>

Command Line: Create Local Repository

- `cd` Change your working directory to your *home* directory.
- `mkdir hello_world` Create a directory named *hello_world*.
- `cd hello_world` Change your working directory to the *hello_world* directory.
- `git init` Initialise a git repository.
 - Output: `Initialized empty Git repository in /some/path/.git?`
- `git status` Check git status.

On branch main

Initial commit

nothing to commit (create/copy files and use `"git add"` to track)

Command Line: Create Local Repository

Note

- `git init` will include subdirectories and their files

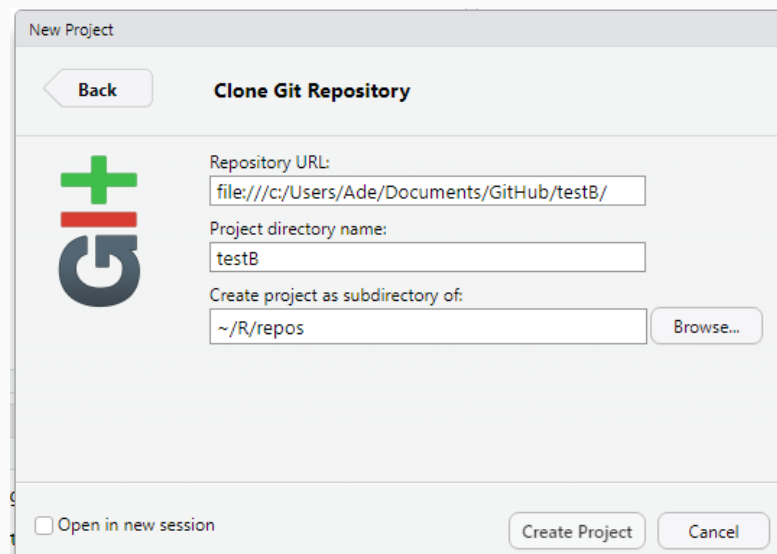
RStudio: Creating a Local Repository

It is not possible to create a local Git repository in RStudio's interface, but only to *clone* one that exists elsewhere - be it on a local file system or on GitHub.

To create a local one, use the command line and run `git init`. In Windows, open *Command Prompt* and run something like this:

- `mkdir hello_world`
- `cd hello_world`
- `git init`

This local repository can then be *cloned* in RStudio via a local URL style like this:



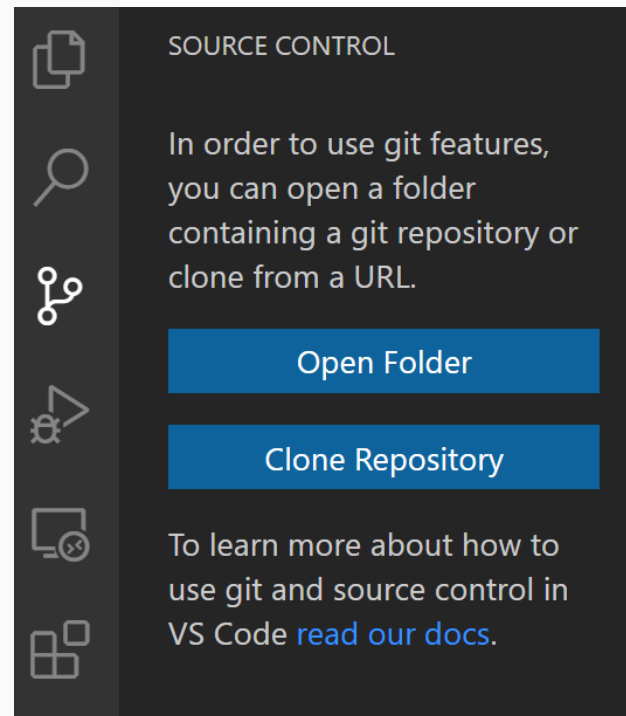
VS Code: Creating a Local Repository

It is not possible to create a local Git repository in VS Code's interface, but you can *clone* one that exists elsewhere or *open* one that is on a local file system.

To create a local git repository, open VS Code and press **CTRL + '** on Linux, macOS or Windows and type the following:

- `mkdir hello_world`
- `cd hello_world`
- `git init`

This local repository can then be *opened* in VS Code via the *Open Folder* option under the *Source Control* tab.



Command Line: Hello World Explained

- We created a directory named *hello_world* to store versions of our files.
- `ls` Empty?
- `ls -a` Hidden files?
- Git uses a special sub-directory to store all the information about a project.
- If we were to delete the `.git` directory, we would lose the project's history.
- `git status`

Summary

- `git init` initialises a repository
- Git stores all of its repository data in the `.git` directory.

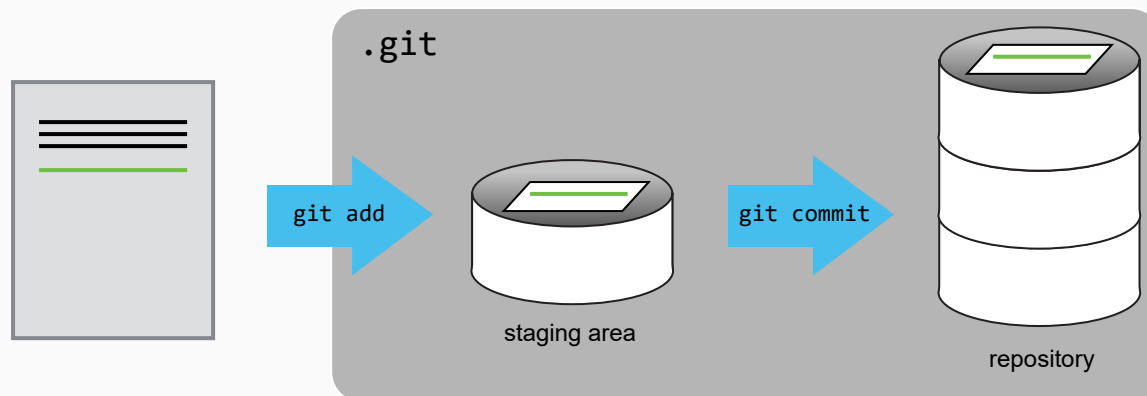
Staging Area & Repositories

Think of Git as taking snapshots or changes over the life of a project.

Files can be stored in

- A project's working directory (user sees)
- The staging area (where the next commit is being built-up)
- The local repository (where the commits are permanently recorded)
- *ADD* specifies what will go in a snapshot (staging area)
- *COMMIT* takes the snapshot and makes a permanent record of it, multiple files typically being changed in one commit to represent a single functional change to the overall application. Useful commit messages are a good idea for future tracking.

Staging Area & Repositories



- *PUSH* takes the permanent record and stores it in the main repository (typically on GitHub)
- *PULL* updates our local working copy (clone) with changes from the repository

Command Line: Tracking Changes

- `cd ~/hello_world && echo "Hello" > data.txt`
 - Inside the `hello_world` directory create a file called `data.txt`
- `cat data.txt` View contents of data.txt.
- `git status` Git has picked up a new file?
- `data.txt` is listed as an **Untracked File**
 - This means that Git is not keeping track of the contents of data.txt
- `git add data.txt`
 - Git will now track this file
- `git status?`

Command Line: Tracking Changes

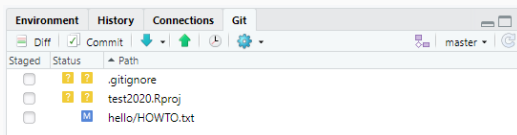
- Git knows to track `data.txt` but hasn't recorded these changes as a commit.
- `git commit -m "Added data.txt"`
- `git status?`
- `git commit` takes everything we have told it to save by using `git add` and stores a copy permanently inside the special `.git` directory
- This permanent copy is called a *commit* or *revision*
- `git log` View recent history for git repository
- `git log --oneline` Reduce the quantity of information

Command Line: Tracking Changes

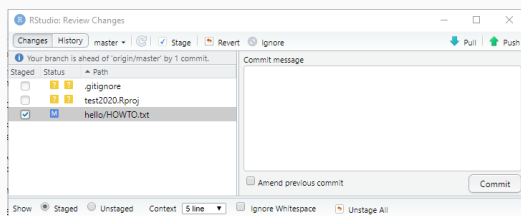
- `echo "Updates" > data.txt` Update data.txt
- `git status?`
- We have made changes, but we haven't told Git we want to save those changes
- `git diff --color-words` Good practise to review the changes first
- `git add data.txt`
- `git commit -m "Updated data.txt"`
- `git add` before `git commit` allows us to be selective on the files we want to commit to our repository

RStudio: Tracking Changes

- Modify a file from the working copy via the RStudio File Browser (bottom-right quadrant), once saved notice the 'Git' tab sees this file as *modified*:



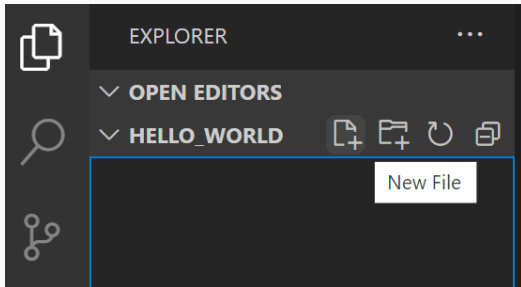
- Now to *commit* this, we need to tick the box next to the file and press the commit button, after which we are presented with the *Review Changes* windows in which we see the commit being staged and also enter our *Commit message*:



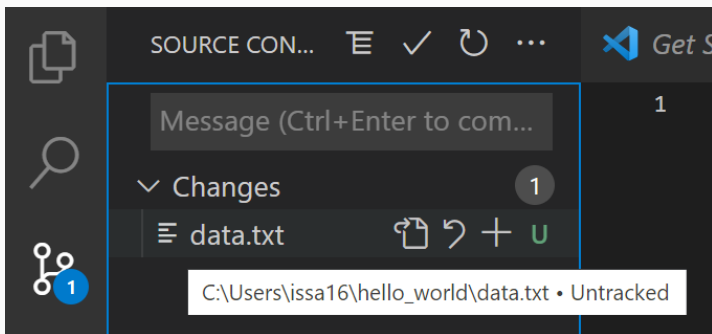
- RStudio will then show the output of the Git commit. We now see the message that "Your branch is ahead of 'origin/master' by 1 commit" showing that we have not pushed this change from the stage to the main repository origin.

VS Code: Tracking Changes

- Inside the `hello_world` directory create a file called `data.txt`

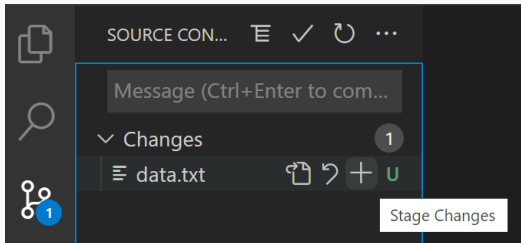


- After doing so, you will see in the *Source Control* tab that the `data.txt` file shows up with the letter **U** besides it. **U** stands for *untracked file*, meaning a file that is new or changed, but has not yet been added to the repository.

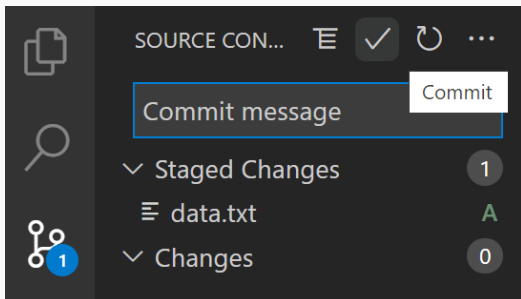


VS Code: Tracking Changes

- Click the **plus** icon (+) by the `data.txt` file listing to track the file by the repository.



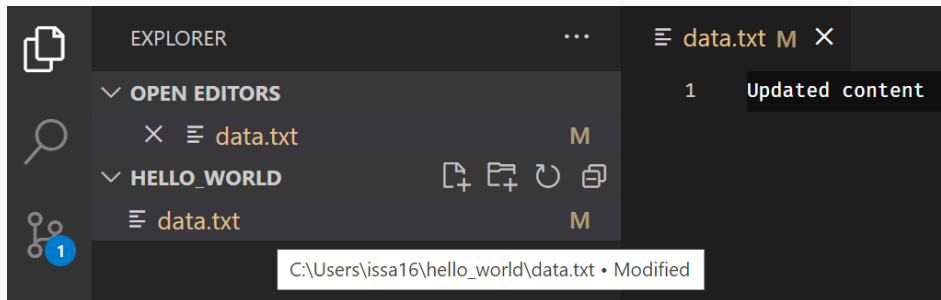
- Once added, the letter next to the file will change to **A**. **A** represents a new file that has been added to the repository. To *commit* the changes, type a commit message into the input box at the top of the *Source Control* tab and click the **check** icon to perform the commit.



- You will notice that there are now no pending changes.

VS Code: Tracking Changes

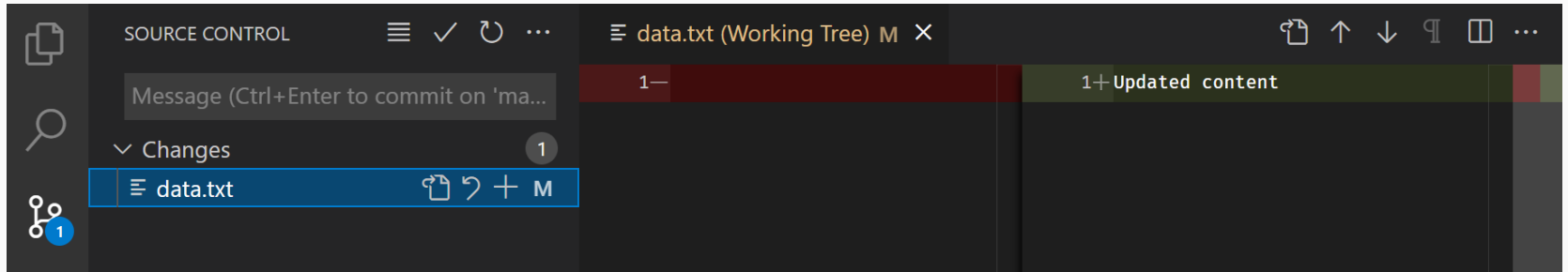
- Modify the contents of `data.txt` and save the file. In the *Source Control* tab you will see that the file has been changed. It will show the letter **M** next to it, which stands for a file that has been modified.



- Click the **plus** icon (+) by the `data.txt` file to add the file to the staging area. To *commit* the changes, type a commit message into the input box at the top of the *Source Control* tab and click the **check** icon to perform the commit.

VS Code: Viewing Changes

To view the changes, open the *Source Control* tab and double-click the `data.txt` file. This will open a typical diff view with the previously committed version of the file on the left and the current version of the file on the right.



Command Line: Using Github

There are a couple of ways to use a Github repo as the backend storage for a repository.

For an existing local repo, connect your local git repo to GitHub like this:

1. Create a repo on Github using the web-based graphical interface
2. In your local shell run the following command to connect the remote repo on Github to your local repo.

```
git remote add origin https://github.com/issa16/test.git  
git branch -M main  
git push -u origin main
```

3. Check the two repos are connected

- `git remote -v`

4. One time operation to store credentials

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

Command Line: Using Github

Alternatively, create the project initially on GitHub and begin local work with a clone thereof:

1. Create a project on Github
2. Clone the repository

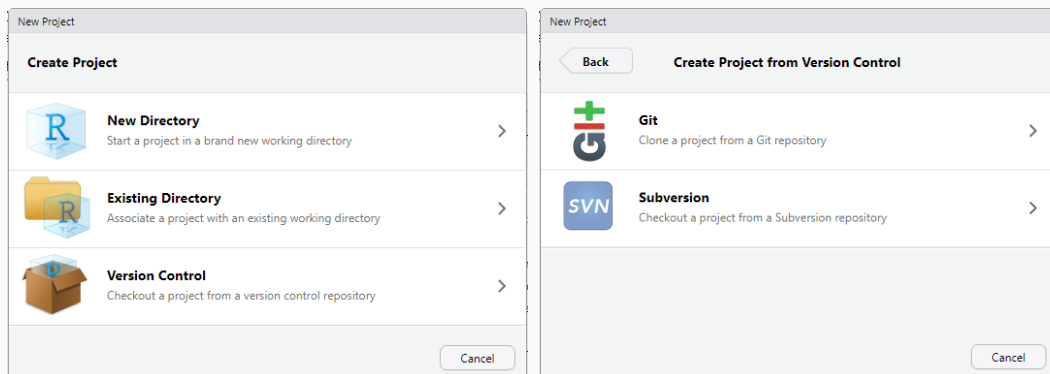
- `cd && git clone https://github.com/issa16/test.git`

We use HTTPS here because it does not require additional configuration. After the workshop, you may want to set up SSH access, which is more secure.

RStudio: Cloning from Github

To *clone* in RStudio from GitHub, we simply use a URL that points to the GitHub repo rather than a local repository.

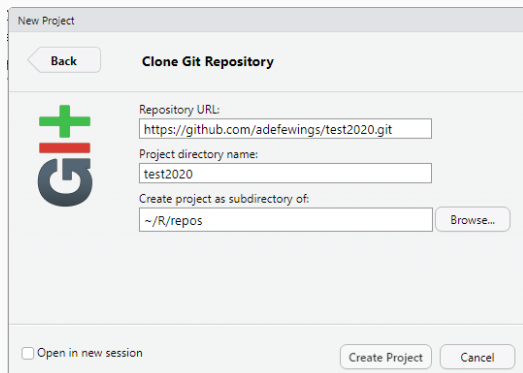
- In RStudio start a new project:
 - File -> New Project -> Version Control -> Git



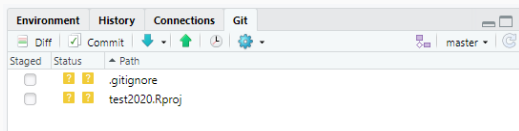
- If *Version Control* not available try restarting RStudio and/or reinstalling the required components as your system paths are most likely incorrect*

RStudio: Cloning from Github

- Put the URL to the Github repository in (from the Github web interface under *Clone or download*), and consider where the local clone will go



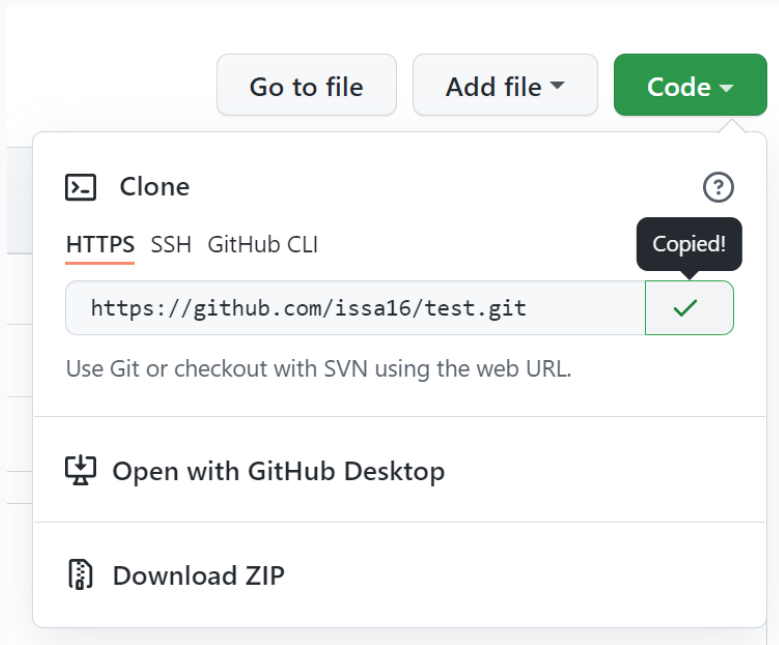
- Click *Create Project*
- RStudio will clone the repository into a working copy and open therein.
- Notice that the new window opened has a *Git* option in the top-right pane:



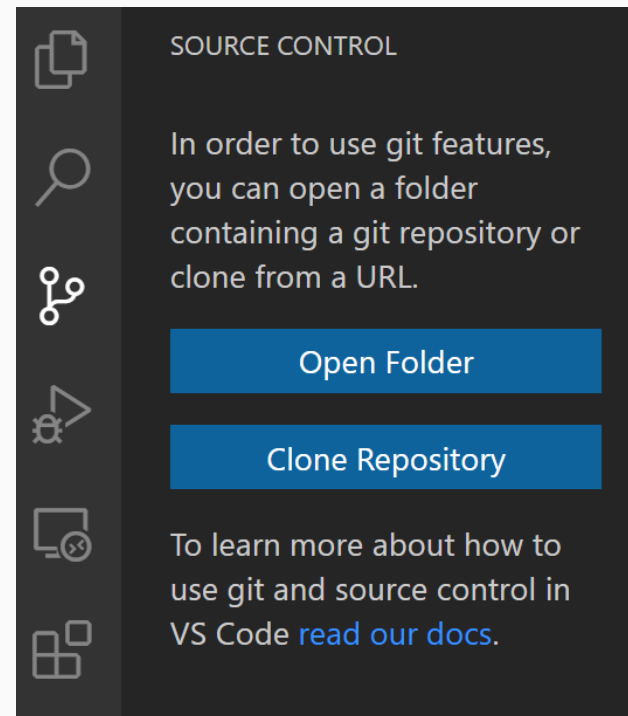
- Buttons here represent the principal Git functions:

VS Code: Cloning from Github

Obtain the Github repository URL by clicking the **Code** button on the repository homepage.

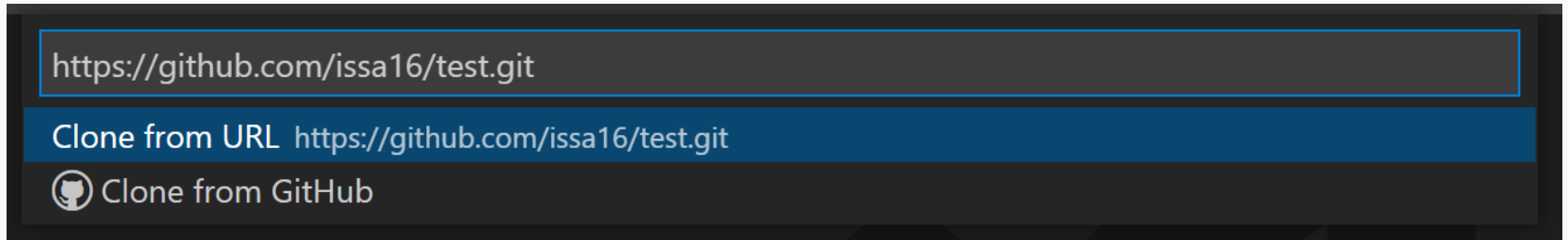


This remote repository can then be *cloned* in VS Code via the *Clone Repository* option under the *Source Control* tab.

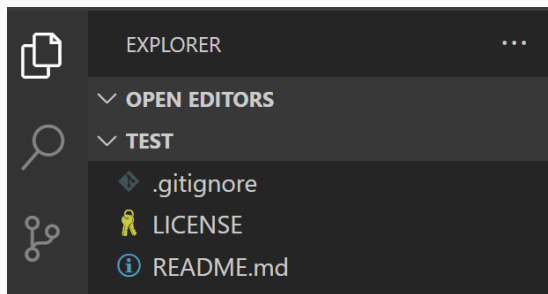


VS Code: Cloning from Github

- Paste in the repository URL and press Enter.



- A window should open to select a location where the local clone will go. VS Code will then download the repository from Github and offer the option to open the files.



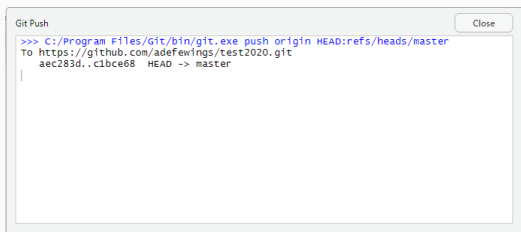
Command Line: Push To Github

Once we have a working clone locally, we can push our changes to GitHub by first *committing* them as above, and then *pushing* them:

- `git push origin main` Push changes from the local repository to Github
- `git pull origin main` Pull changes from Github to the local repository

RStudio: Push to Github

- Once we have staged changes and are ready to commit to the origin, this can be achieved either through the 'Review Changes' window or the main Git tab in the RStudio project editing window.
- Simply select the green up arrow, provide your GitHub credentials if prompted and see a push result:

A screenshot of the 'Git Push' window in RStudio. The window has a title bar 'Git Push' and a 'Close' button. The main area contains a terminal-like output showing the command executed: `>>> C:/Program Files/git/bin/git.exe push origin HEAD:refs/heads/master`. Below this, it shows the remote repository: `to https://github.com/adefewings/test2020.git`. The final line of the output is `aec283d..c1bce68 HEAD -> master`, indicating a successful push of the HEAD branch to the master branch on the specified GitHub repository.

```
Git Push
Close
>>> C:/Program Files/git/bin/git.exe push origin HEAD:refs/heads/master
to https://github.com/adefewings/test2020.git
aec283d..c1bce68 HEAD -> master
```

- Our changes have now been pushed up to the repository on Github.

Command Line: Pull changes from

To *pull* (retrieve) changes to content stored that has been committed and pushed to the repo from elsewhere (perhaps, for example, we have a clone on another machine we use at home), we:

- `git pull origin main`

We can then see the changes:

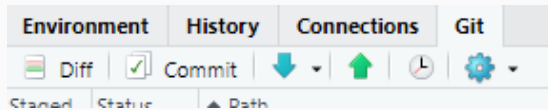
- `ls`

And, we can view the commit logs that were pushed along with any code changes:

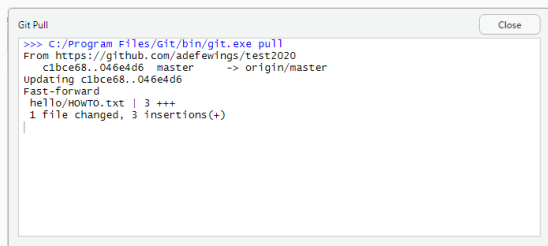
- `git log --oneline`

RStudio: Pull changes from Github

It's trivially easy to pull any changes from the GitHub repository for a working copy (clone) by pressing the blue down button:

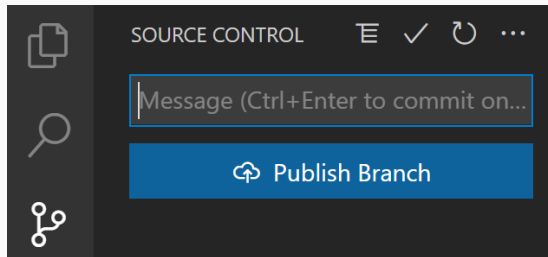


Which will result in display of the Git activity:

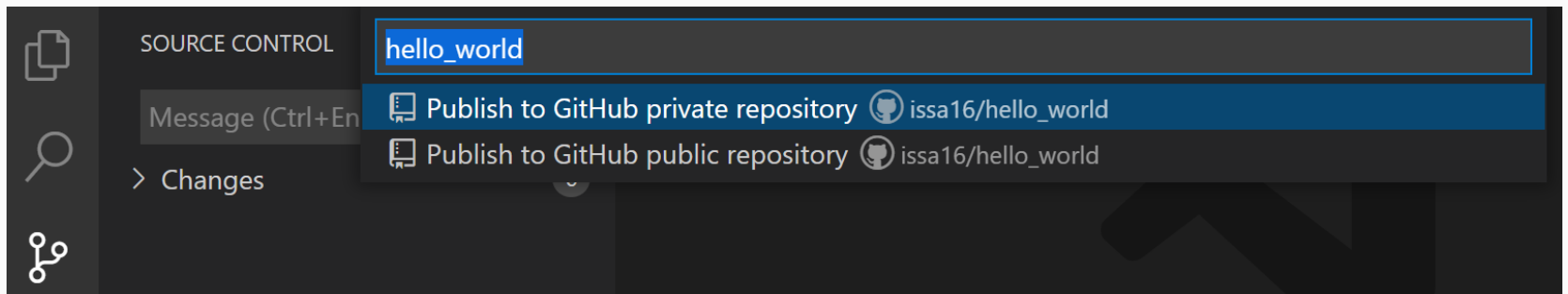


VS Code: Publish Branch

To upload our local git repository to Github, select the *Publish Branch* option that will be available after the initial *commit* in the *Source Control* tab.

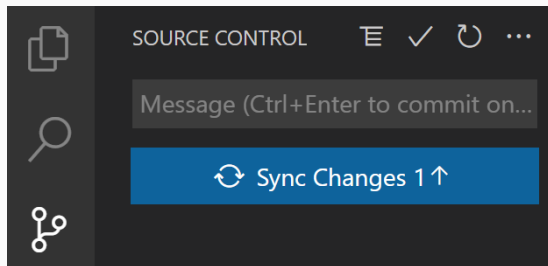


- Follow the prompt, you will be given the choice between a private and public repository.



VS Code: Sync Changes (Push and Pull)

To upload our local commits to our Github repository, and to download any commits from Github to our local git repository, select the *Sync Changes* option that will be available after each *commit* in the *Source Control* tab.



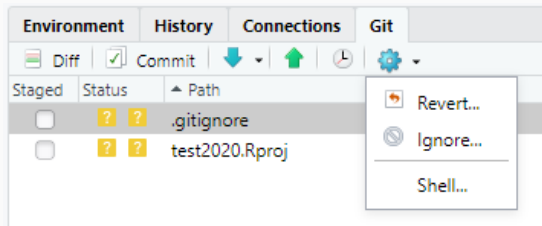
Command Line: Ignoring Files

What if we have files that we do not want to track?

- backups or intermediate files created during data analysis?
- `cd ~/hello_world`
- `touch a.dat b.dat c.dat` Do not want these files to be tracked
- `git status` Always listed and could distract us from changes that matter
- `nano .gitignore`
 - `*.dat` # Tells Git to ignore all .dat files
 - `!a.dat` # Except a.dat
 - `results` # Tells Git to ignore all files in the results directory
- `git status`? Much better
- Need to save our *.gitignore* file to our repository

RStudio: Ignoring Files

Similarly, to ignore files in RStudio, under the 'Git' tab, we can click the Blue Settings Cog icon and choose ignore to adjust the list of ignored files (as itself stored in .gitignore):



VS Code: Ignoring Files

Branches

- A branch represents an independent line of development.
- Think of them as way to request a brand new working directory, staging area and project history.
- List all branches in our repository.
 - `git branch`
- Create a new branch
 - `git checkout -b <name_of_branch>`
- Push updates
 - `git push origin <name_of_branch>`
- Switch branch
 - `git checkout <name_of_branch>`

Github and Collaboration

The power of version control systems comes into its own when we start to collaborate with other people.

It enables different people to develop different parts of a project, some elements experimentally, and eases the merging of these changes.

Version control allows a particular project state at a point in time to be stored forever. Typically this would denote major changes, such as a new version.

As changes are required to be attributed to a user, accountability is also introduced. This, with helpful user interfaces and meaningful messages about change enable a far more collaborative working environment.

Github Project Collaboration

To configure a GitHub hosted repository for collaboration among users, some simple settings are applied in the GitHub interface:

- Settings
- Manage access
- Add people
- The collaborator must accept access
- The collaborator must clone the repository
- Create a file with a unique name
- Add, commit, push to the repo
- Should see changes in GitHub from multiple users

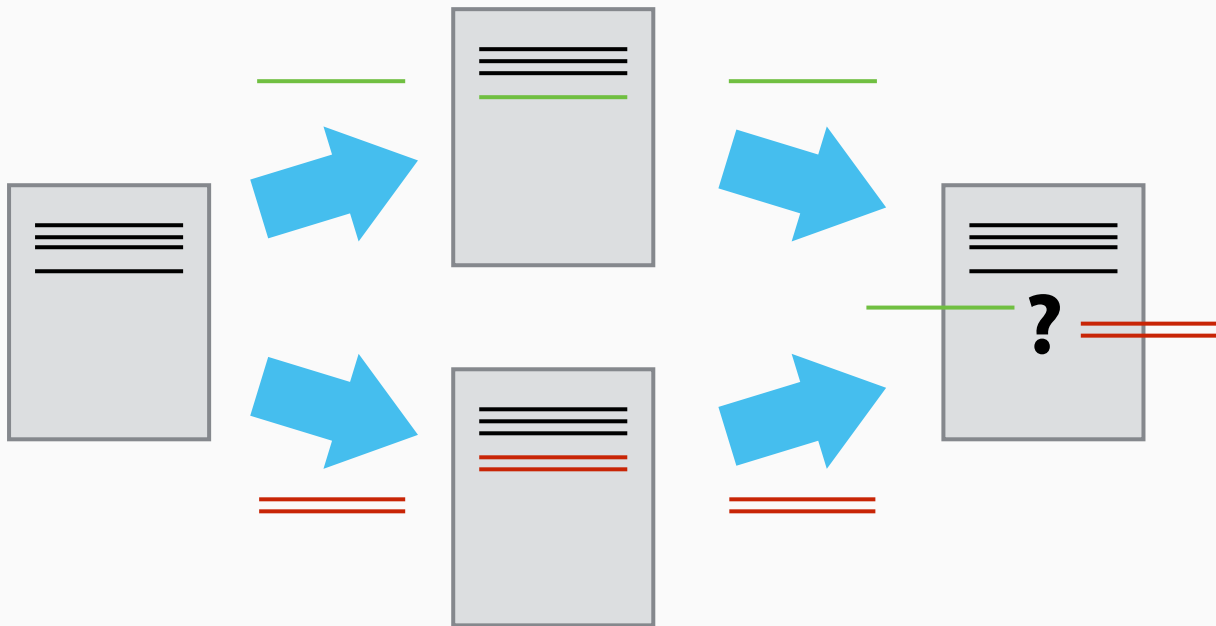
A Basic Collaborative Workflow

- `git pull origin main` Update your local repository
- `git add` Make your changes and stage them
- `git commit -m` Commit your changes
- `git push origin main` Upload the changes to GitHub

Theory: Conflicts

As soon as people can work on a document in parallel, conflicts will happen

- Version Control helps us to manage conflicts by providing tools to resolve overlapping changes



Theory: Conflicts

Conflicts exist when changes to a file in different working copies occur concurrently and thus clash with the repository during commit/push.

E.g.

- All users clone Aaron's repository on Github
- Aaron updates **test.txt**
- All users edit and move to commit and push **test.txt**
- Users can commit the change locally but Git won't let them push it to GitHub
- Why?

Theory: Conflicts

Practical

- Git rejects the push because it detects that the remote repository has new updates that have not been incorporated into the local clone working copy
- What we have to do is pull the changes from GitHub, merge them into the copy we're currently working in, and then push that to Github

Command Line: Conflicts

- `git pull origin main` - Pull in latest updates
- `nano test.txt`

```
<<<<<< HEAD
This is your edit
=====
This is aaron's edit
>>>>>>
```

- Need to decide which line is correct? User process.

Command Line: Conflict Resolution

RStudio: Conflicts

VS Code: Conflicts

Also: GitHub Desktop

GitHub Desktop

Licensing

- When code or other creative work becomes public, it should include a **LICENSE** or **LICENSE.txt** in the base directory of the repository
- It should clearly state under which license the content is being made available
- Reusing creative works without a license is dangerous because the copyright holders could sue you for copyright infringement
- Choose an open source license
- People who incorporate General Public License (GPL'd) software into their own software must make their software also open under the GPL license; most other open licenses do not require this
- The Creative Commons family of licenses allow people to mix and match requirements and restrictions on attribution, creation of derivative works, further sharing, and commercialisation
- People who are not lawyers should not try to write licenses from scratch
- Beerware?!

Citation

Basic

- You may want to include a file called **CITATION** or **CITATION.txt** that describes how to reference your project
- More detailed advice and other ways to make your code citable can be found at the [Software Sustainability Institute](#)
- [Example](#)

Advanced

- [Official Tutorial: Making Your Code Citable](#)

Resources

- <https://swcarpentry.github.io/git-novice/>
- Yammer - Research Computing Community
- <https://github.com/BangorUniversity/Research-Computing-Community>
- <https://guides.github.com/>

