# Introduction to Version Control

*phs.datascience@phs.scot*

Public Health Scotland

# Who are we?

**Russell McCreath –** Senior Information Analyst

Co-leading the development and delivery of data science training across Public Health Scotland. Also working on transformation projects bringing automation, testing, and reproducibility to legacy products.

**Ciara Gribben –** Senior Information Analyst

Co-leading the development and delivery of data science training across Public Health Scotland. Also working on a range of COVID analytical projects.

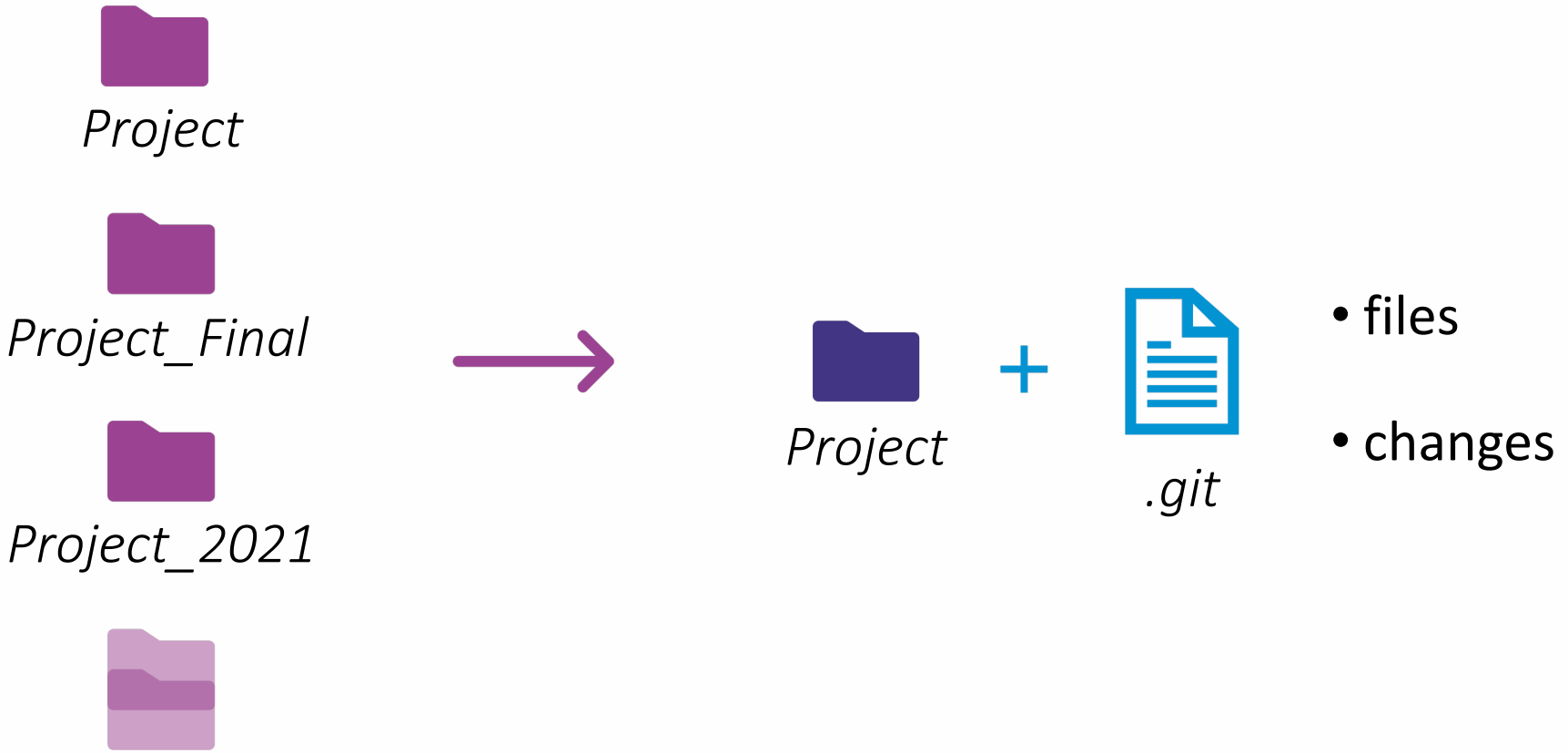
# Introduction – the why

# Version Control

*Tracking and maintaining a full history of changes on a local project which can be shared to collaborate (distributed version control). The version we're focusing on is called Git.*

- Which changes were made?
- Who made the changes?
- When were the changes made?
- Notification of when work conflicts occur.

# Git

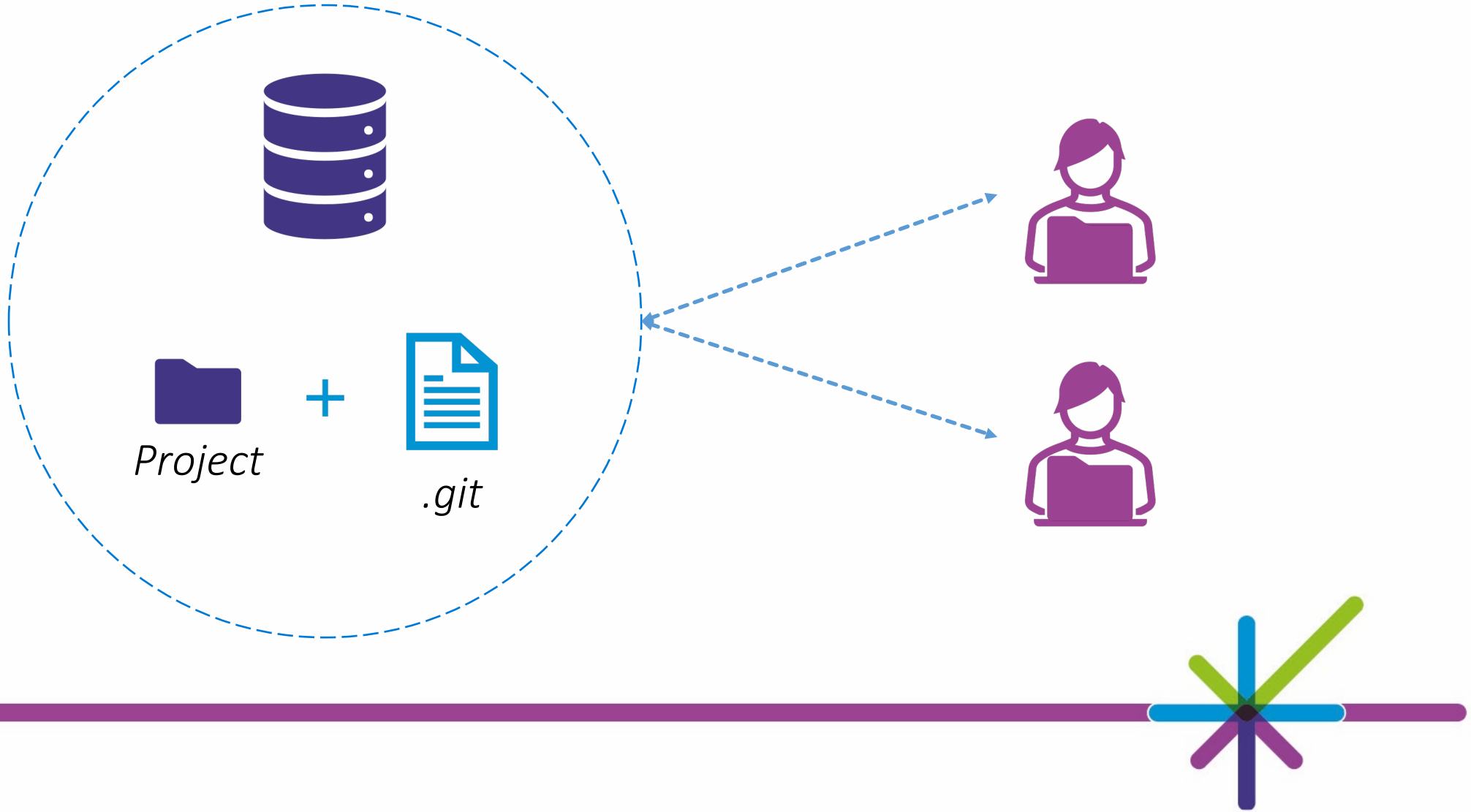

Project
Project_Final
Project_2021
Project
.git
• files
• changes

# Version Control - Remote

*Maintaining a remote version of the project in a central server, accessible by all collaborators with tools to manage workflow.*

- GitHub
- Gitea (our secure, internally hosted option)
- GitLab, Bitbucket, Azure Repos, AWS CodeCommit…

# Version Control - Remote



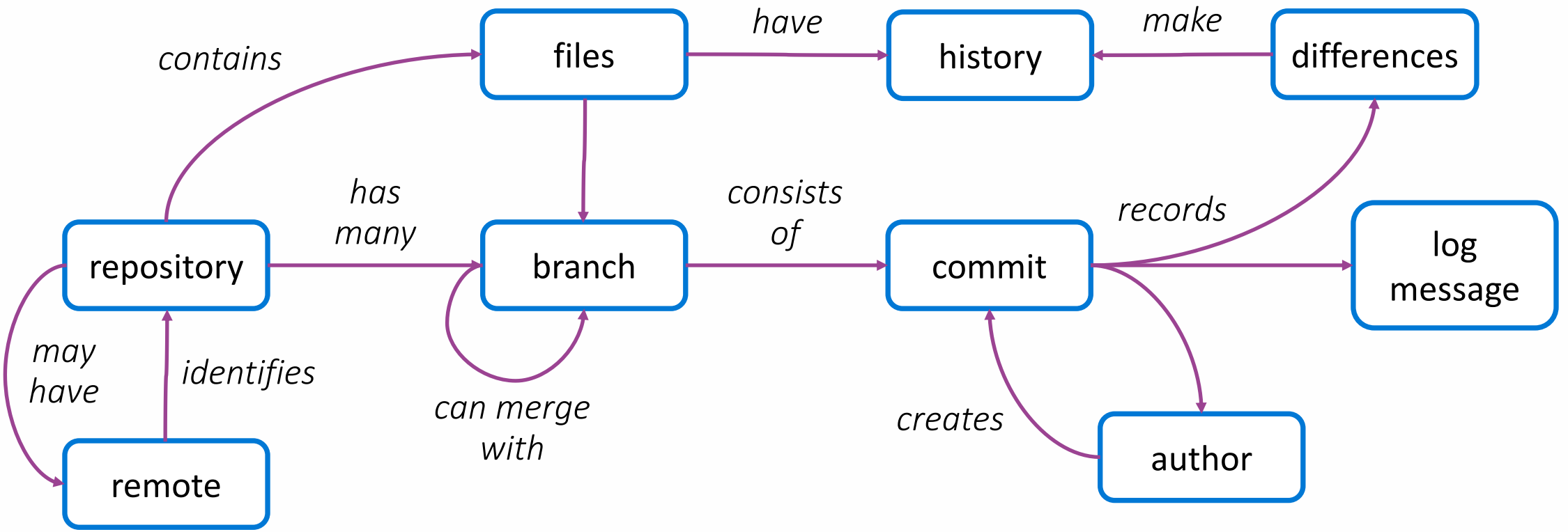
# Demo

1. View the Public Health Scotland GitHub organisation

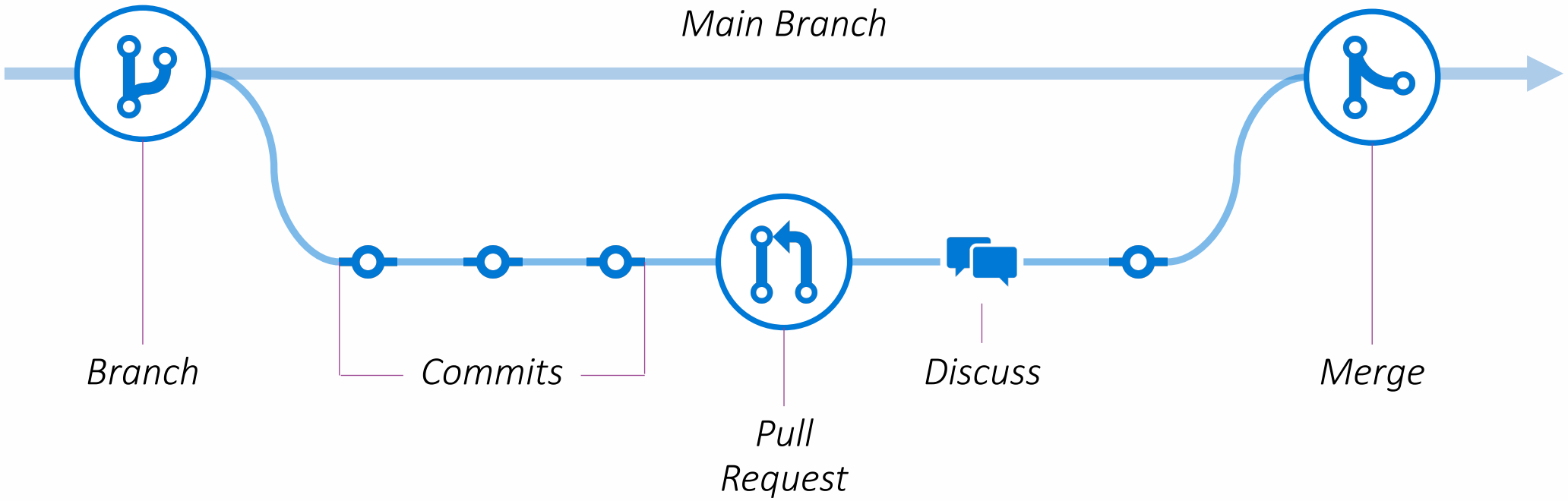
2. View some of the repositories

# Foundations – the what

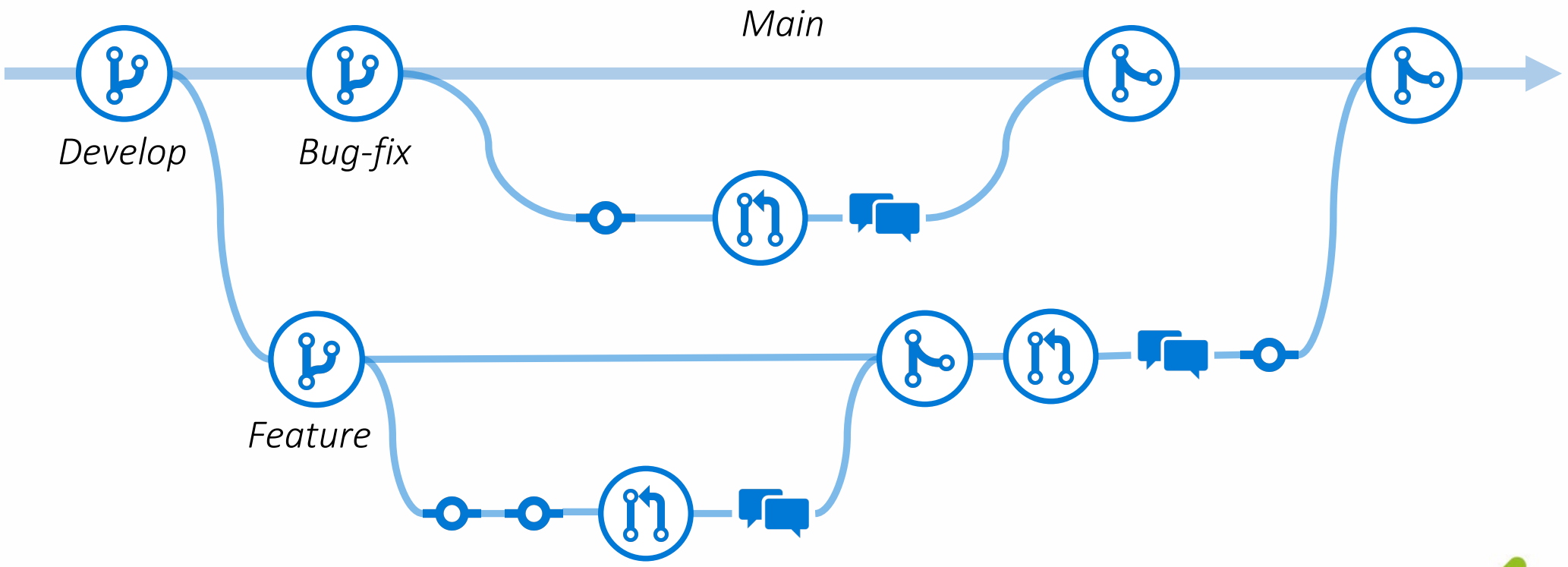
# A Concept Map



# Basic Workflow



# Advanced Workflow



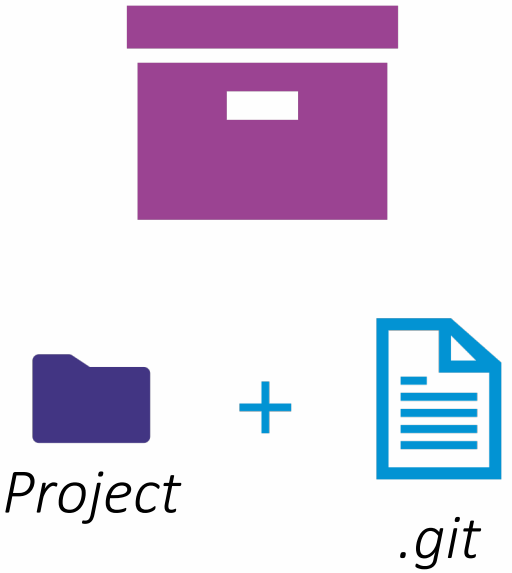
# Repositories – the how

# Repository (Repo)

Git projects have two parts which make a repository:

- user generated files/directories
- .git directory

*The .git directory is located at the root of the repository, it stores all the information Git records, and should never be edited directly.*



# New Repo

You can start using Git at any stage in a project, the earlier the better. This will add the .git directory ready to start tracking work.

```
$ git init
```

*In RStudio:*

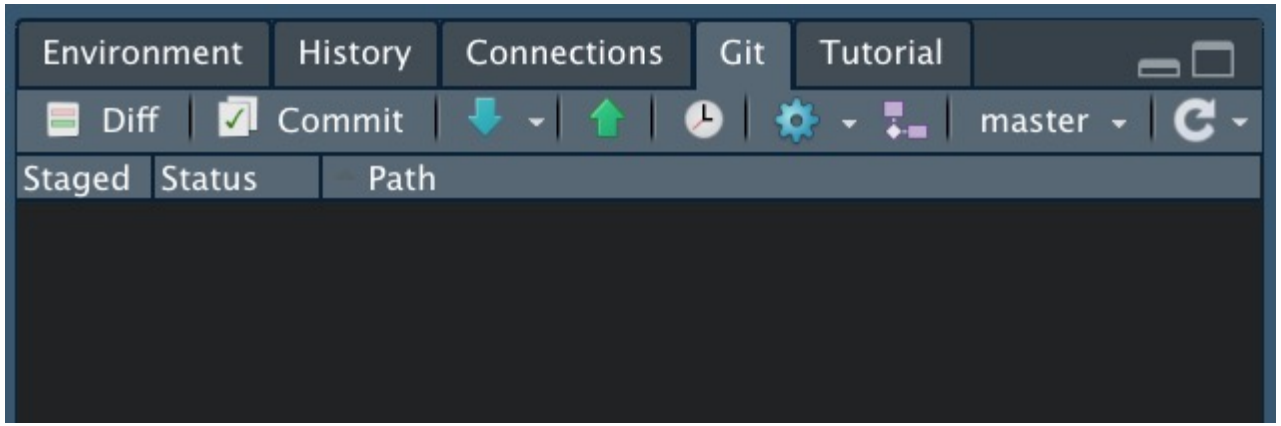
- *New Project: select 'Create a git repository' during setup*
- *Existing Project: go to Tools > Project Options > Git/SVN > and select Git as the version control system.*

# Activity 1

# Status

As you work on your project, you'll frequently want to check the Git status. This will display a list of files that have been modified since the last commit.

```
$ git status
```



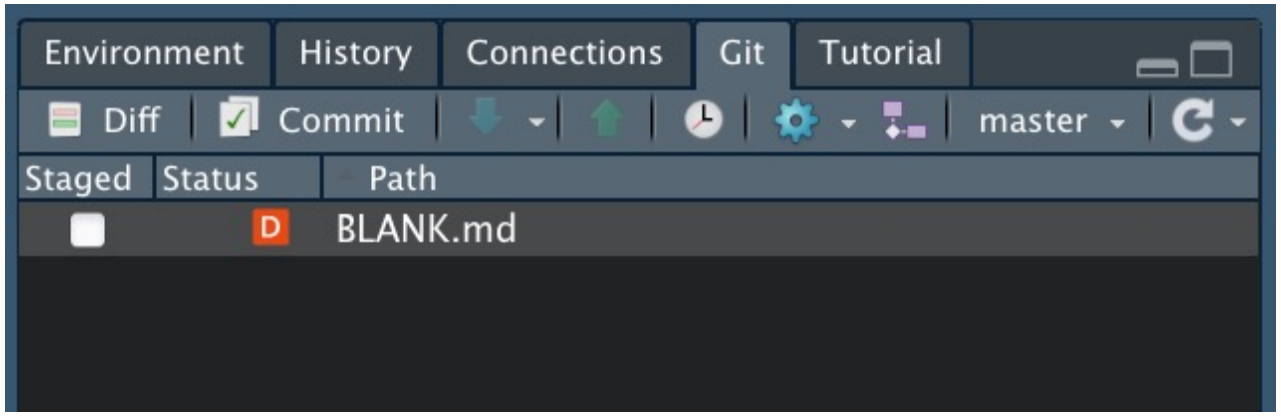

*As we've just cloned the repo, there are no changes to see… yet!*

# Change

While you work, Git is tracking. Any changes made will show as part of the Git status.

*We've deleted the file called 'BLANK.md'. The status shows this and lets us know it has been deleted (red D).*

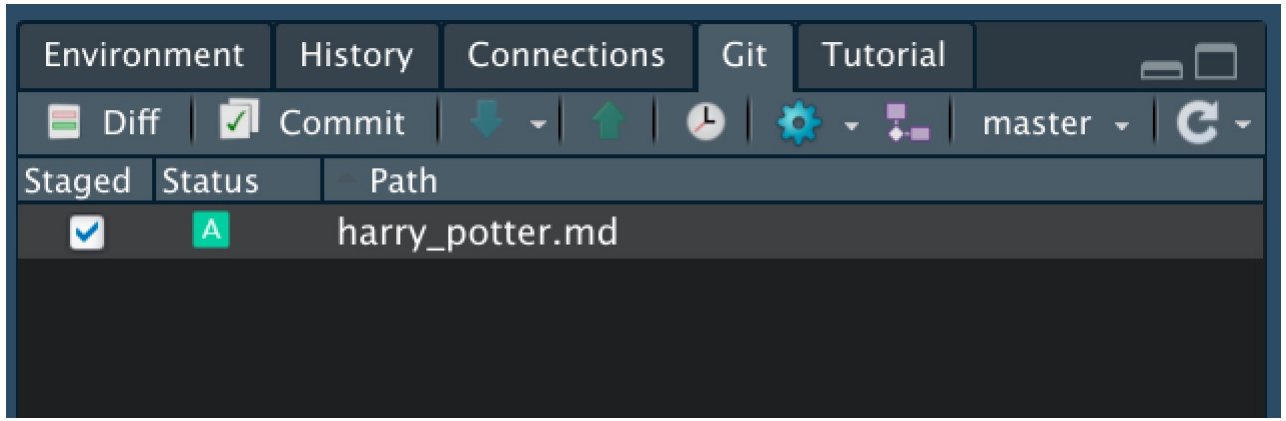
```
$ git status
```




# Stage

When we have made changes and want to build a commit (a snapshot of changes), we need to stage them using the checkboxes.

```
$ git add <filename>
```

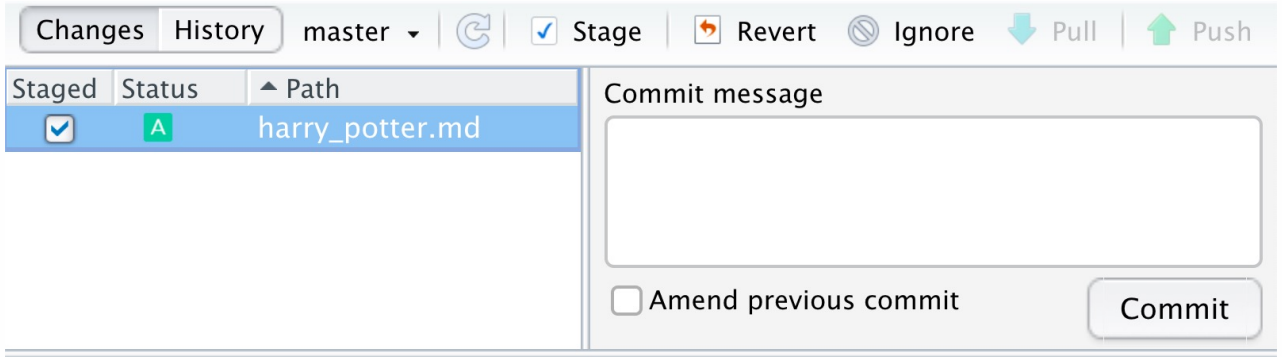



*This is like putting together a parcel to send in the mail. You can freely add and remove things (changes), "building" your commit.*

# Commit

A commit takes everything that has been staged to generate a historical log of changes. You also add a meaningful message to identify changes and reasoning.

```
$ git commit -m "<message>"
```



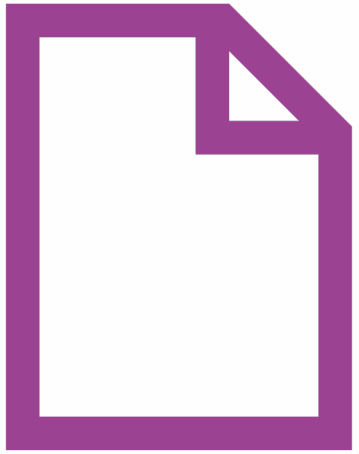

*To continue the analogy, this is posting your parcel. Once it's sent, the parcel is gone, and you can't change what's inside.*

# Activity 2

# Ignore

What about files we don’t want to track (data, configuration, logs, etc.)?
Well, Git provides us with the .gitignore file.

*This file can take exact strings for filenames and directories, or wildcard patterns. As such "build" and "*.pdf" will ignore directories (and anything in it) or files called ‘build’ and any PDF file.*



*.gitignore*

# Activity 3

# History – the how

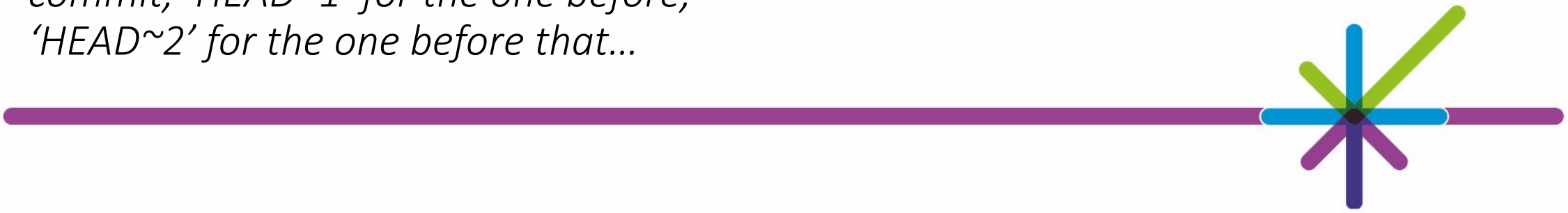
# Hash

Every commit has a unique ID, called a hash, generated from the content of the commit. This is a long hexadecimal string, but when directly referencing, the first 6-8 characters will most certainly be enough.

*If using command line, a special label, 'HEAD', is used to represent the last commit, 'HEAD~1' for the one before, 'HEAD~2' for the one before that…*

```
fc75a7ce604a13357f050f0f8735da
4c2396ef97


9521c18c0143a36a18553522bbeeb5
1ebc29ae41
```



# Log

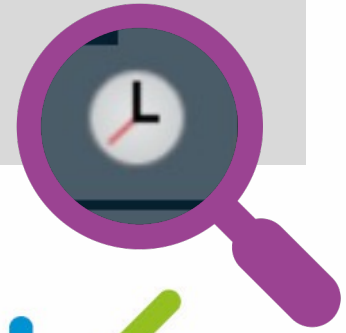
The log is Git's history, it includes the hash, and details of who, when, and what.

*All commits are shown in the log, this is an example of how one is displayed. RStudio's interface is more friendly, click on the clock to explore this repo's history.*

```
$ git log
```

```
commit  fc75a7ce604a13357f050f0f8735d
Author: Git Learner <learner@git.com>
Date:   Mon Feb 01 13:31:26 2021 +0000

    initial commit
```

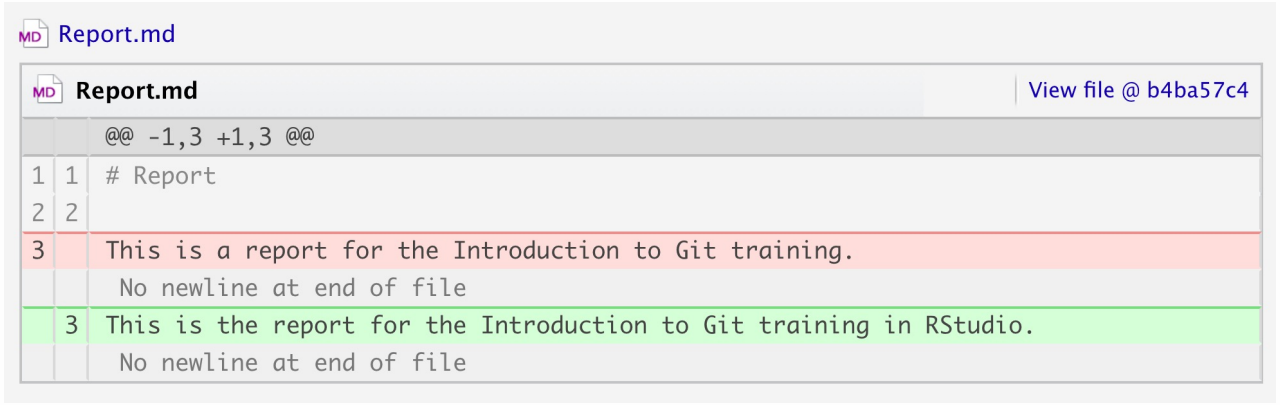


# Show

Taking a step into one of the commits, Git allows us to see the specific details and changes that the commit has made.

*RStudio's interface uses colour to highlight the added and deleted components from files.*

```
$ git show <hash>
```




# Difference

While you can see the changes one commit has made, you can also compare the difference between any two commits. You'll see this command come up again.

```
$ git diff <hash1>..<hash2>
```


*RStudio currently doesn't have an interface for this but the diff will show the difference for all files compared to the last commit.*

# Undo

To revert changes to a previous commit, you effectively load the state of the previous commit, and then provide a new commit. This allows a full history where you can make a small undo or even undo your undoing.

```
$ git checkout <hash> <file|.>
```

*'file' – checkout specific filename*
*'.' – checkout full commit for all files*

*RStudio doesn't have a user interface for this. However, you can view the history, see what changes were made, copy them to your working files, and then make a commit (if you'd rather avoid the terminal).*

# Activity 4

# Branching – the how

# Branching

Branching is incredibly powerful, allowing us all to work on different things at one time. Changes in one branch do not affect others, until they are merged.

```
$ git branch

  feature-branch
* main
```

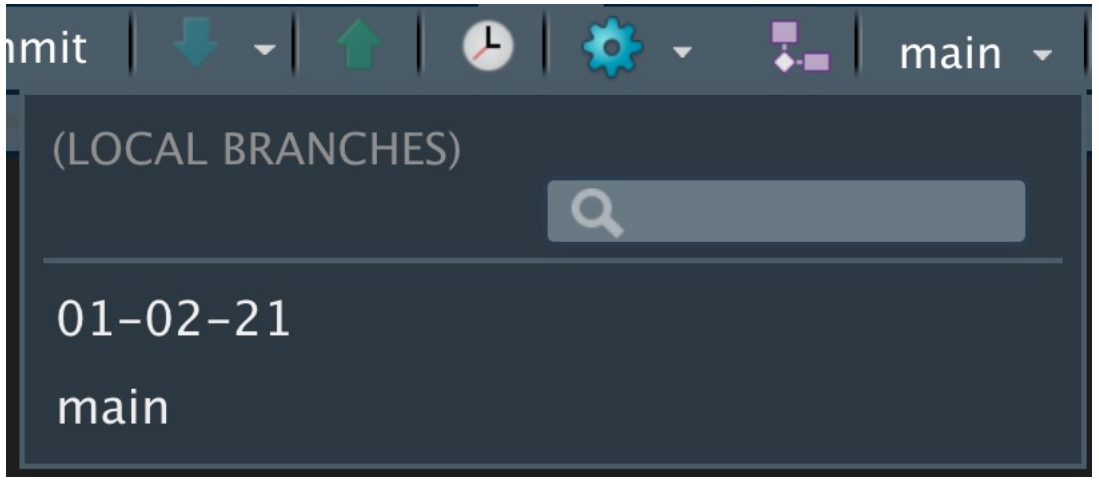
*You have already been working on a branch, called 'main'. This is the default name for the primary branch (you may also see 'master', an older terminology, being used). However, it's best to protect this branch as "production ready" and develop features in feature branches.*

# Switch

We've seen the checkout command previously; we can use this to switch to an existing branch.

*RStudio provides a dropdown of all existing branches with a search functionality.*

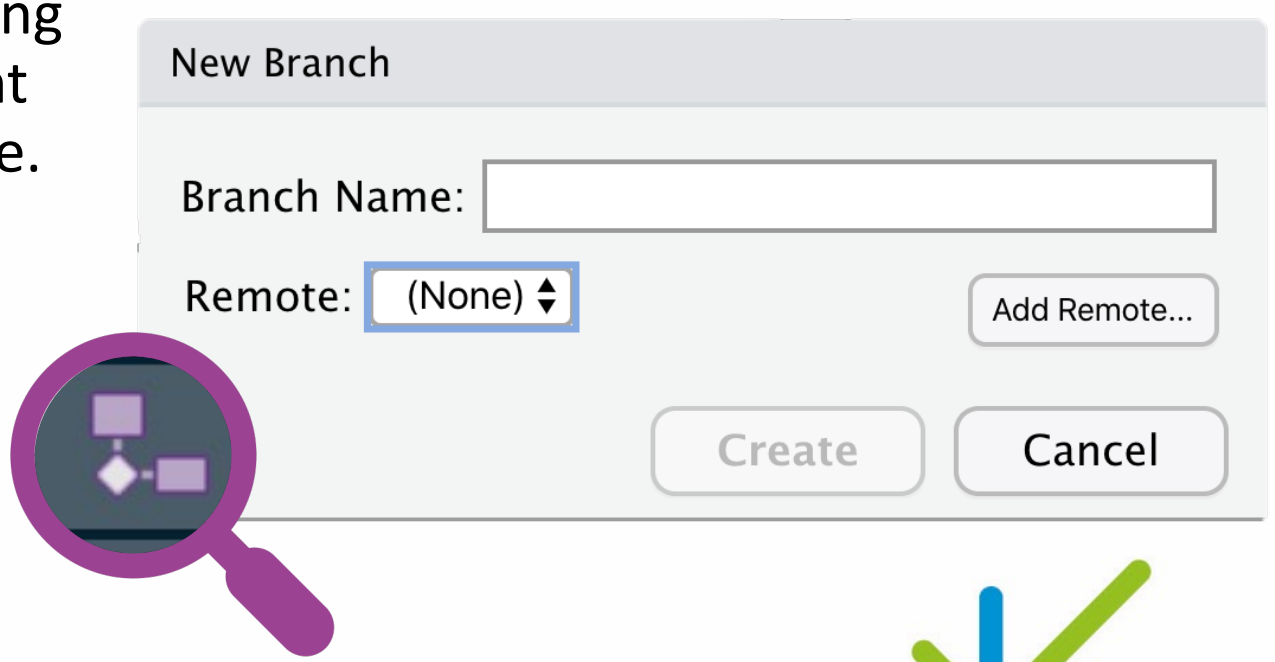
```
$ git checkout <branch>
```




# New Branch

When creating a branch, it's typical to start working on that branch straight away. When we create a branch using these methods, it copies the current branch and switches to the new one.

```
$ git checkout -b <branch>
```




*It's a common workflow, in PHS, to create analyst branches (analyst name) rather than for a feature.*

# Activity 5

# Merge

Merging brings the changes from one branch, combining them with another. You should be on the destination branch.

*RStudio doesn't have an interface for this function. So, we need to use the terminal and write some Git commands.*

```
$ git merge <source> <dest>
```

```
commit  fc75a7ce604a13357f050f0f8735d
Author: Git Learner <learner@git.com>
Date:   Mon Feb 01 13:31:26 2021 +0000

    initial commit
```

# Conflicts

Conflicts occur when there are overlapping changes. Git will add markers inside the file to identify the issue.

Conflicts are resolved by removing the markers, making required changes, then committing those changes.

```
<<<<<<< destination-branch-name
...changes from the destination branch...
=======
...changes from the source branch...
>>>>>>> source-branch-name
```

# Deleting Branches

When using branches for features, once they've been merged, there's no longer a need for them. Remember that merging is combining all changes. So, we can delete them using this command:

```
$ git branch -d <branch>
```

# Remotes – the how

# Remotes

Git meets collaboration with remotes. A remote provides a centralized repository that can pulled (downloaded) to your local computer, and then pushed (uploaded) back to the remote. Different services also provide other services to work collaboratively; opening discussions, issues, project planning, etc.

- *GitHub*
- *Gitea*
- *GitLab, Bitbucket, Azure Repos, AWS CodeCommit…*

# Check Remotes

The remote command shows the remote names connected (origin is the default), adding –v will print URLs alongside the names.

```
$ git remote -v
```

If you clone a project, Git automatically remembers the remote. It is also possible to have multiple remotes.

# Add/Delete Remotes

It’s possible to connect any two Git repositories in this way but generally they should share a common history in some way.

```
$ git remote add <name> <URL>
```

```
$ git remote rm <name>
```

# Pulling & Pushing

When collaborating, pulling changes allows changes made available in the remote to be merged with the local copy of the branch. This should be done regularly to make sure updates are received.

```
$ git pull <remote> <branch>
```

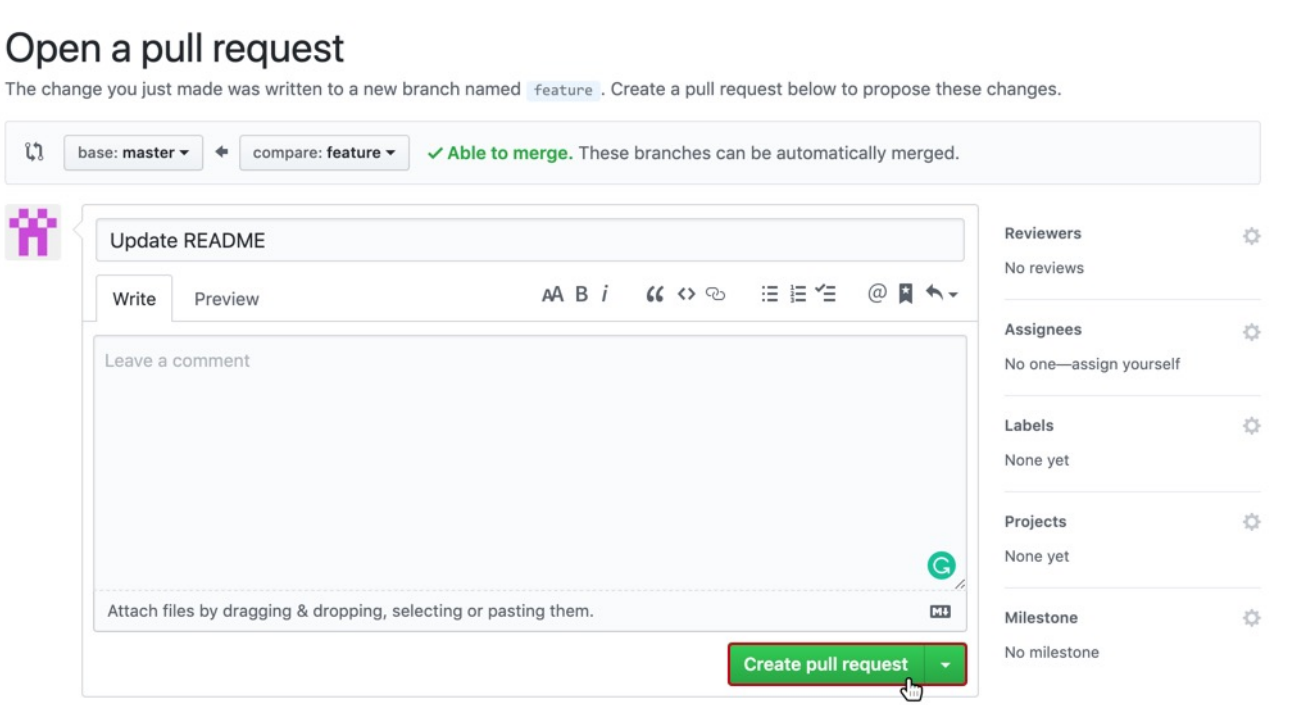
```
$ git push <remote> <branch>
```

The complement of pulling is pushing, this shares changes made locally back to the remote.

# Pull Request

When merging a branch, it's advised to utilise a pull request. This is like a code review stage and adds some protection to any higher branches. It's also a great way to learn for the reviewee and reviewer.

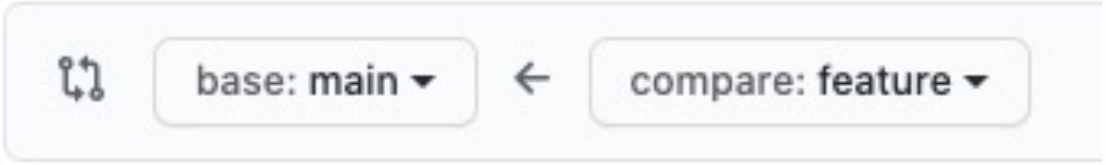
*The pull request must be generated from the remote.*



# Pull Request

1. On GitHub, go to ‘Pull requests’ tab and click ‘New’.

2. Select the branch you're merging to as ‘*base*’ and the feature branch as ‘*compare*’.



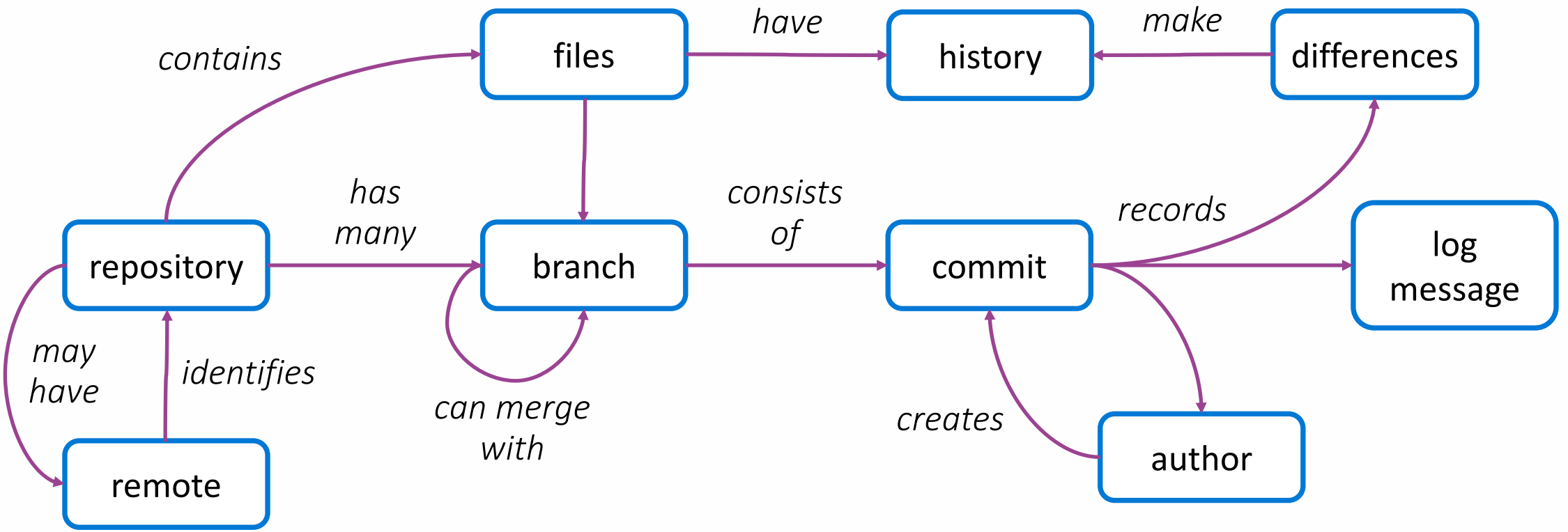

3. Add a title and comments.


4. Further options are available on the right menu, e.g., adding specific reviewers.

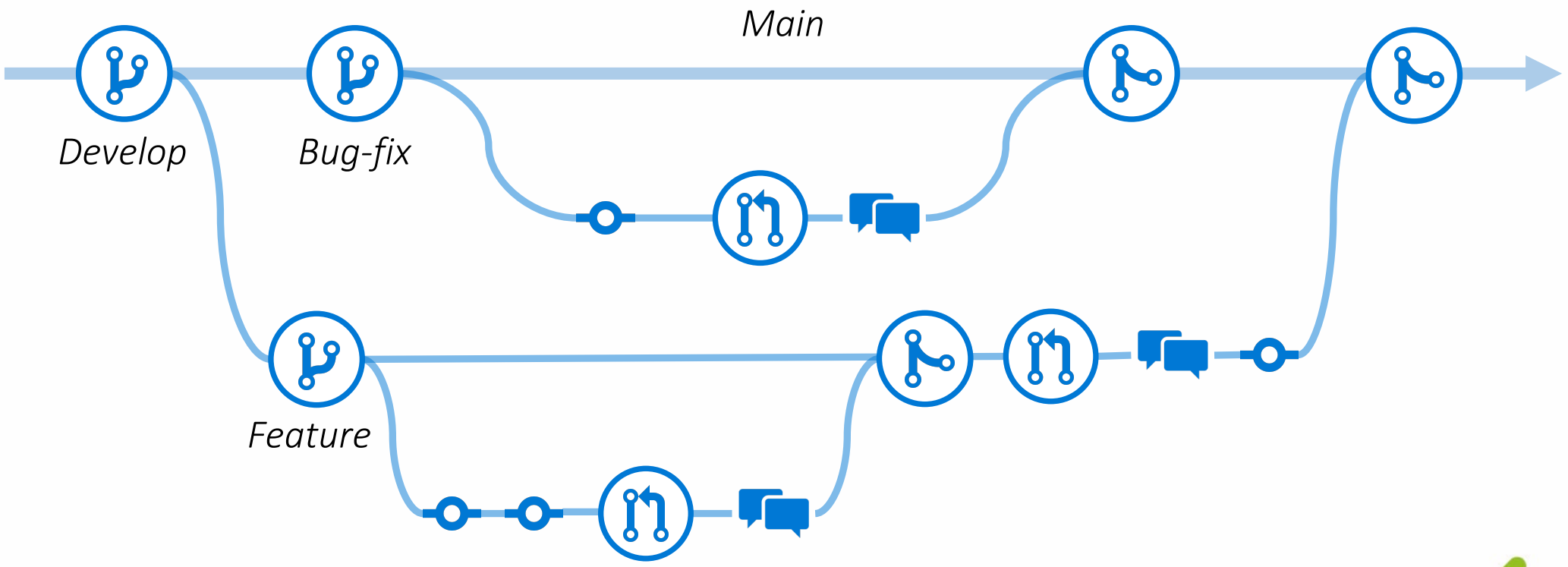
# Activity 6

# Review

# A Concept Map



# Advanced Workflow



# Next Steps

- Access the classroom and get practical with it: https://classroom.github.com/a/LBBzSptF

- Google / Stack Overflow

  tag queries "[git]" & "[github]"