

Introduction to Git

v1.0

Transforming Publishing

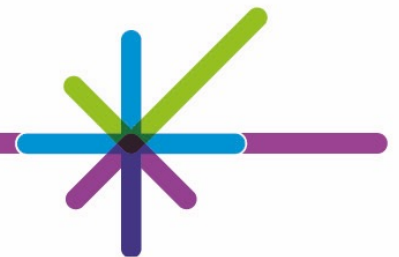
phs.transformingpublishing@phs.scot

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

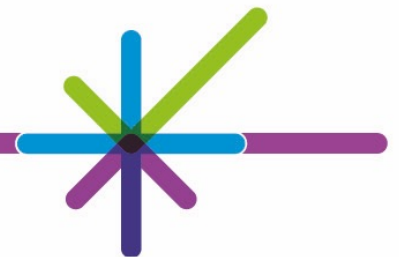
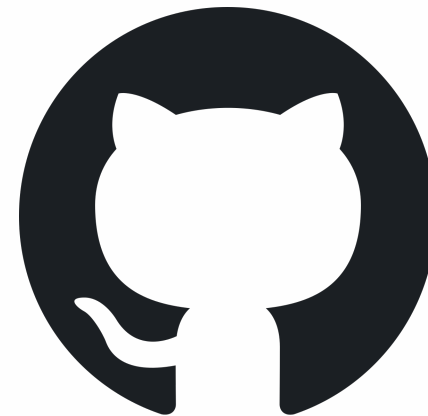
- changes



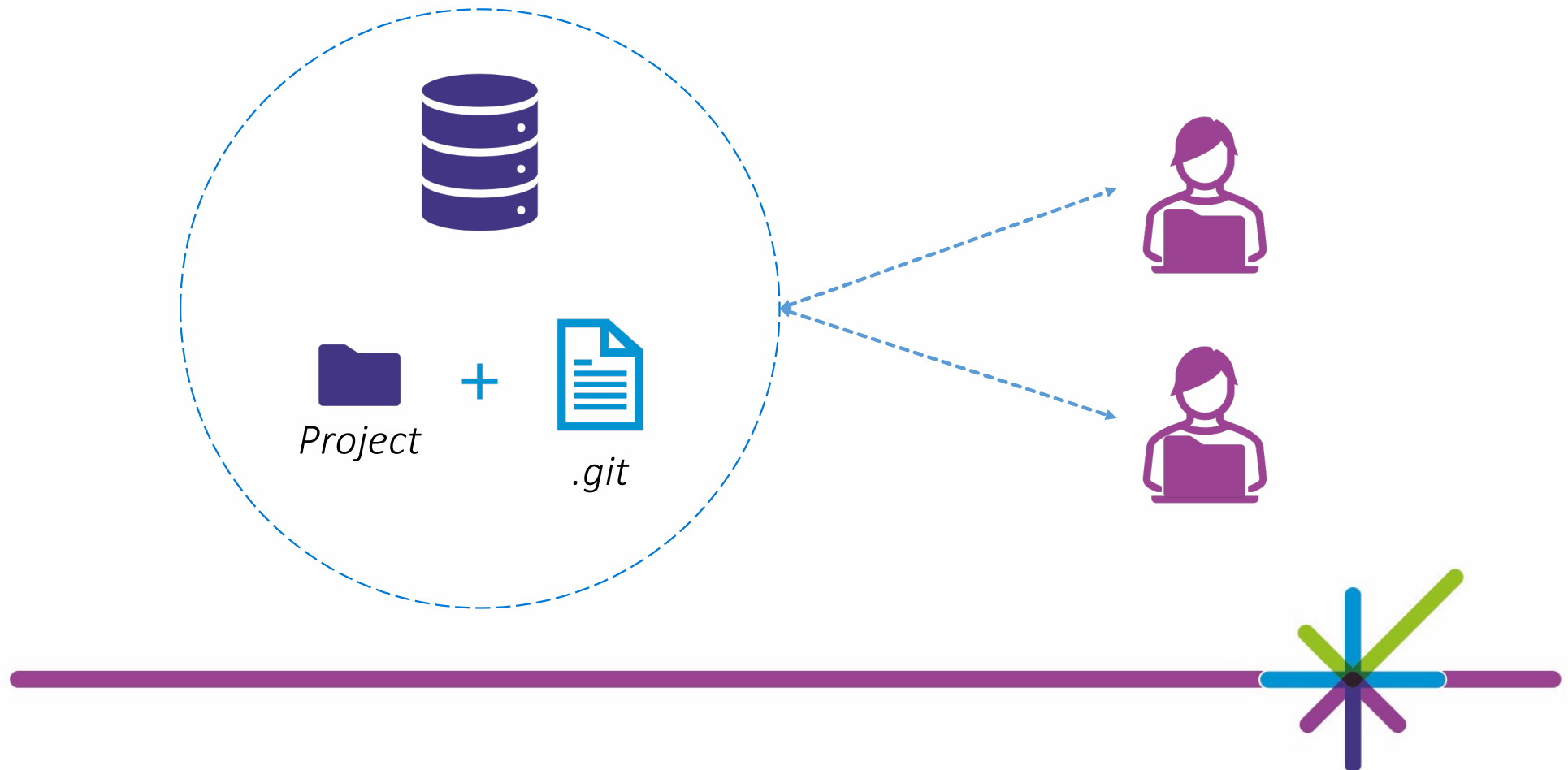
Version Control - Remote

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

- GitHub
- Gitea (secure, internally hosted option)

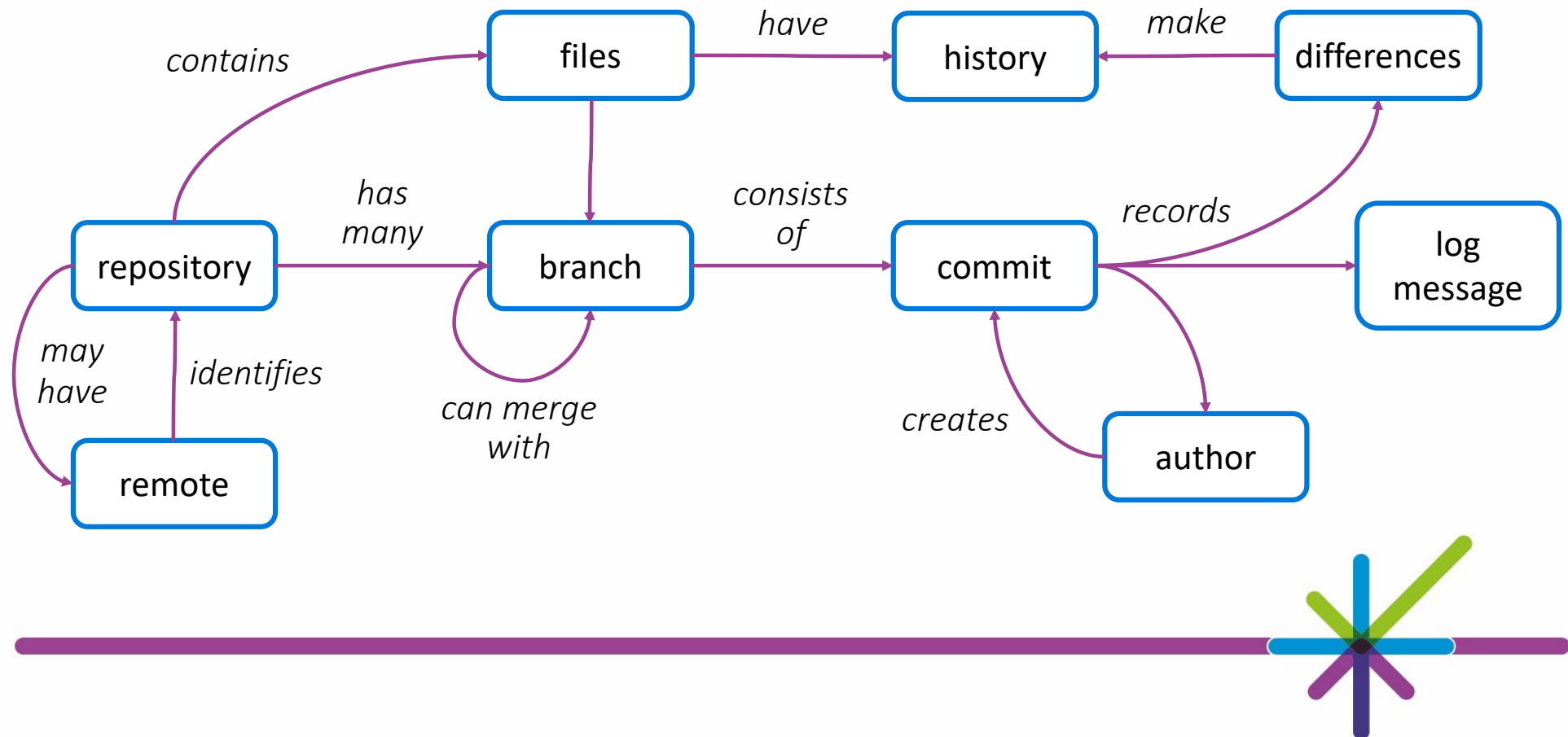


GitHub / Gitea

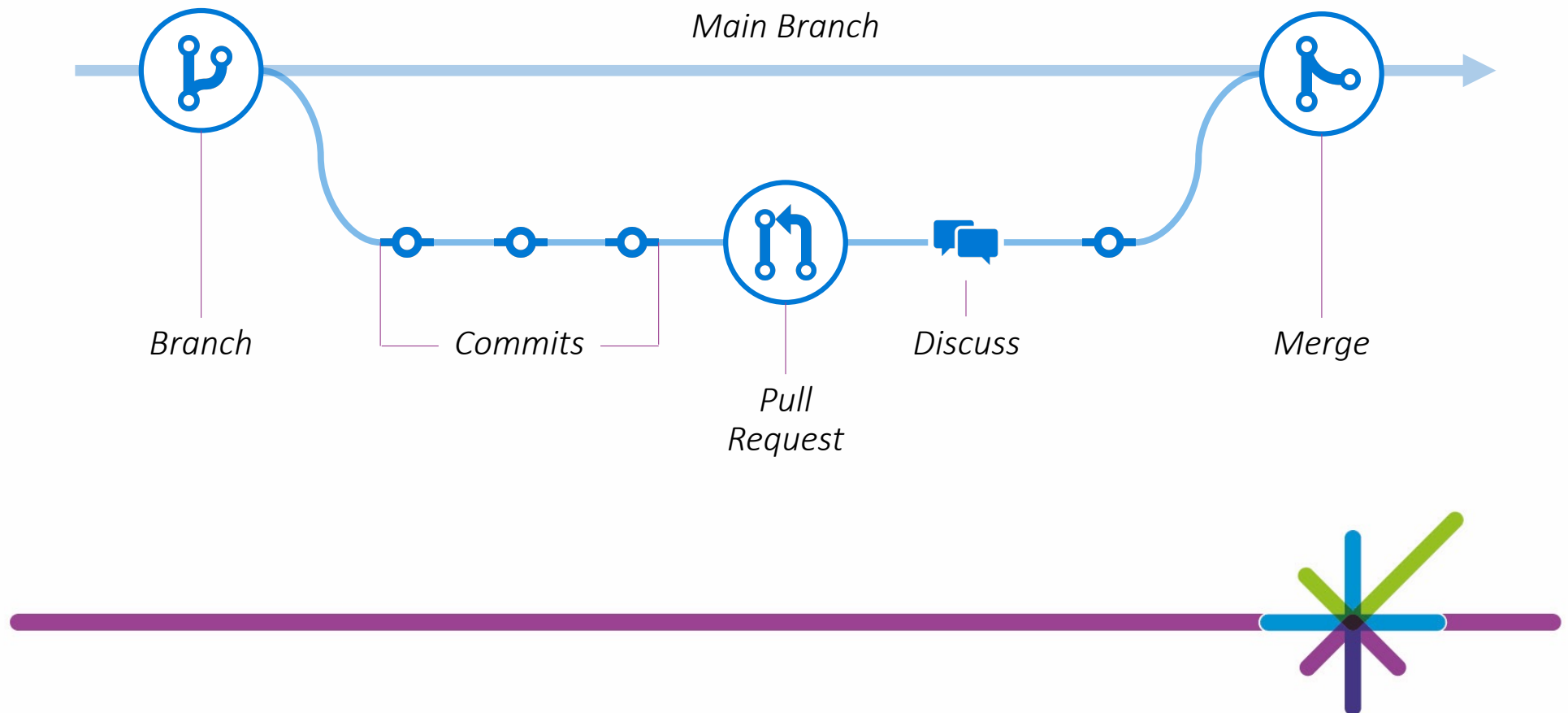


Foundations – the what

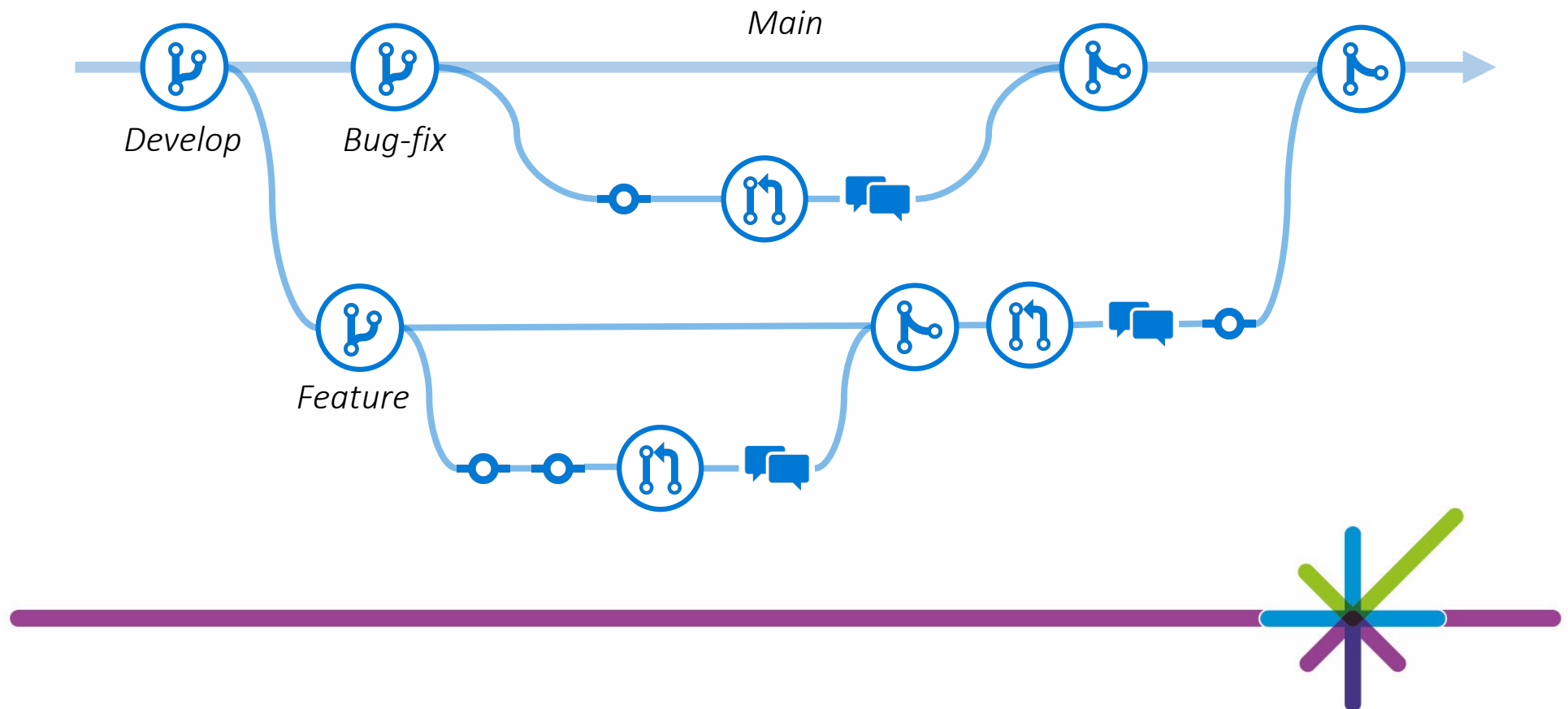
A Concept Map



Basic Workflow



Advanced Workflow



Setup

Checklist

Git installed and configured



GitHub account > added to PHS organisation



RStudio installed and Git software linked



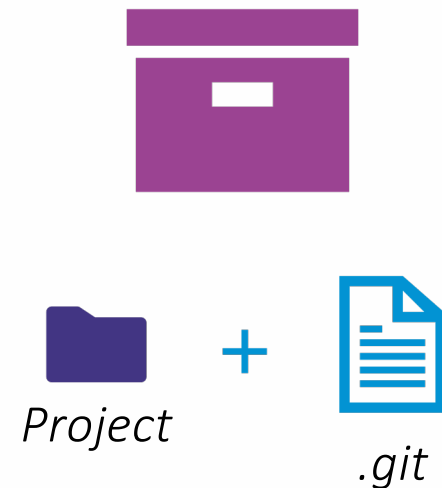
Repositories – the how

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. Go to GitHub repo: github.com/Public-Health-Scotland/learn-git
2. Explore contents of repo
3. Guided clone of repo
4. Open project in RStudio

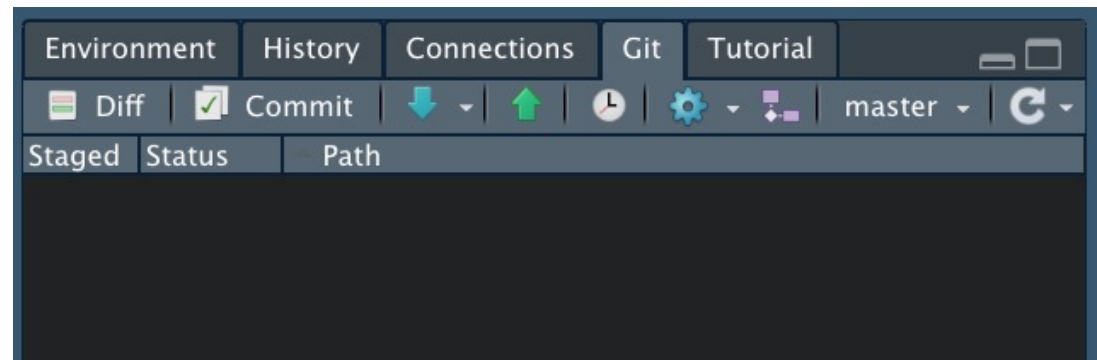


Status

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.

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

```
$ git status
```

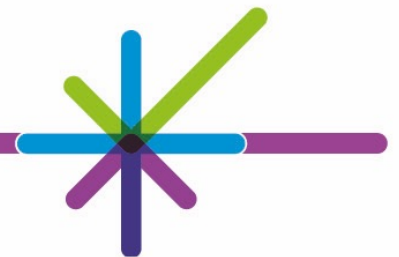
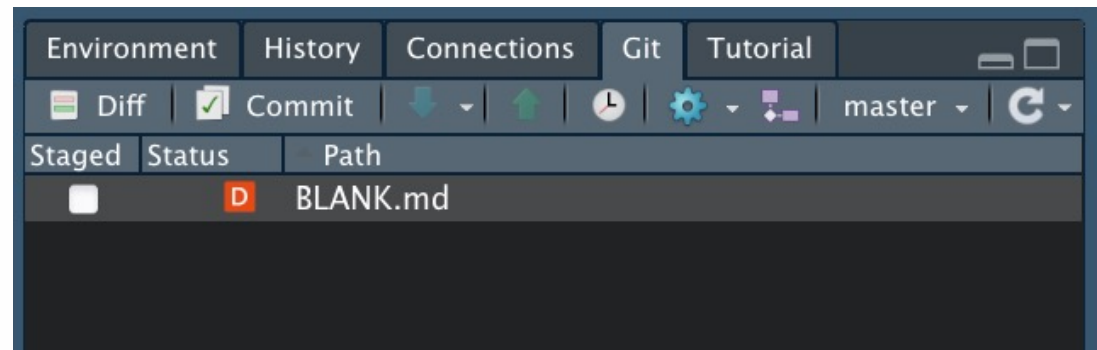


Change

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
```

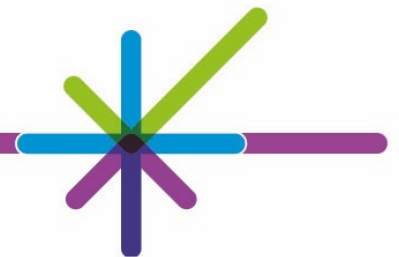
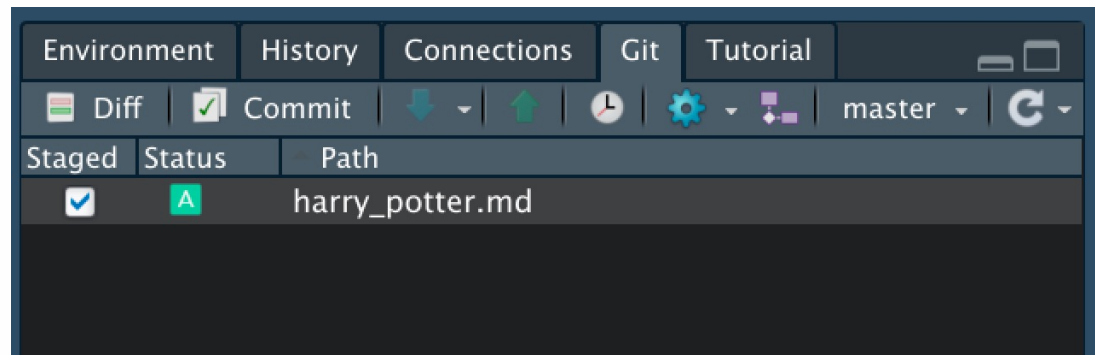


Stage

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

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

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



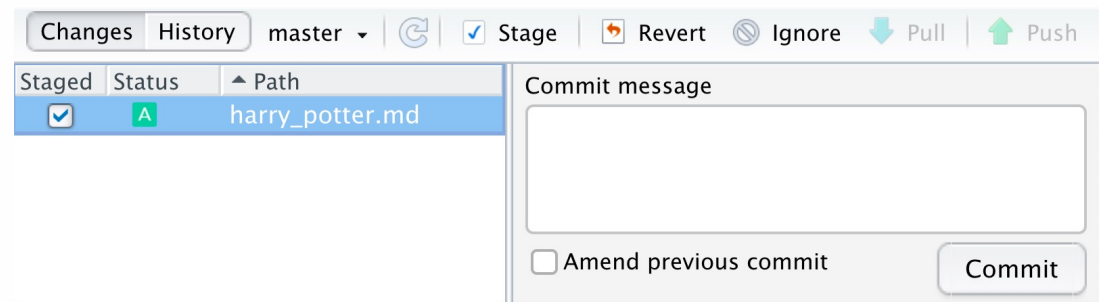
Commit



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.

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.

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



Exercise 1

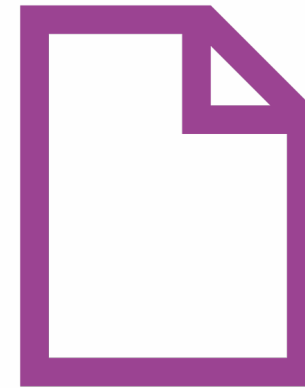
1. Create a text file.
2. Add your name to the first line of the file.
3. Save the file as "<your_name>.md"
4. Stage and commit. Don't forget your message.



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



Exercise 2

1. Create a CSV file with "<name>, <age>, <location>", saved as "<user_details>.csv".
2. Change the .gitignore to ignore all CSV files.
3. Stage and commit only the .gitignore changes.
4. Review the Git status.



History – the how

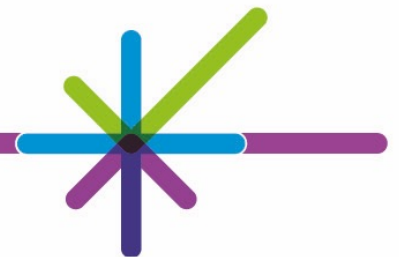
Hash

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

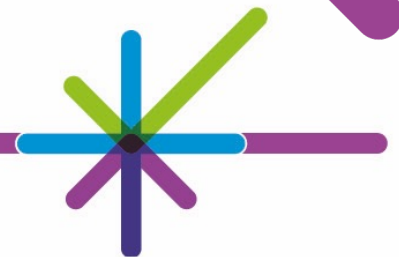
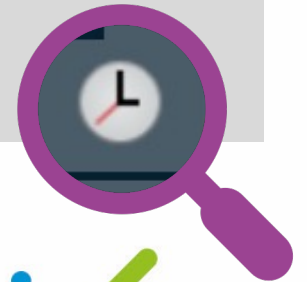
The log is Git's history, it includes the hash, and details of who, when, and what.

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>
```

MD Report.md	
MD Report.md	
@@ -1,3 +1,3 @@	
1	1 # Report
2	2
3	This is a report for the Introduction to Git training.
	No newline at end of file
3	This is the report for the Introduction to Git training in RStudio.
	No newline at end of file

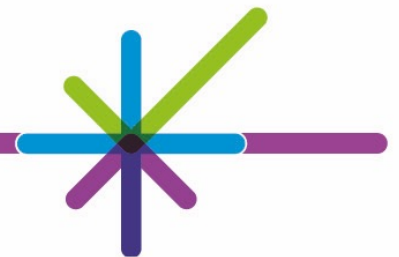


Difference

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.

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

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



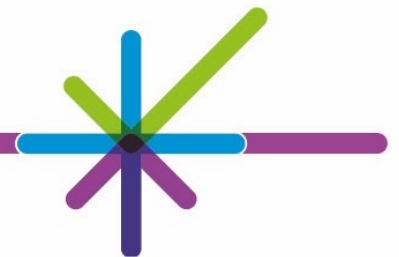
Undo

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).



Exercise 3

1. View the repo's history and find the commit 'update report'.
2. Undo all changes created by this commit.
3. Check the *status*, *stage* and then *commit* with a meaningful message.



Branching – the how

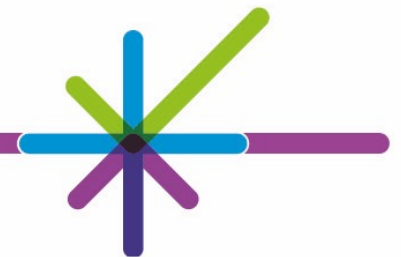
Branching



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.

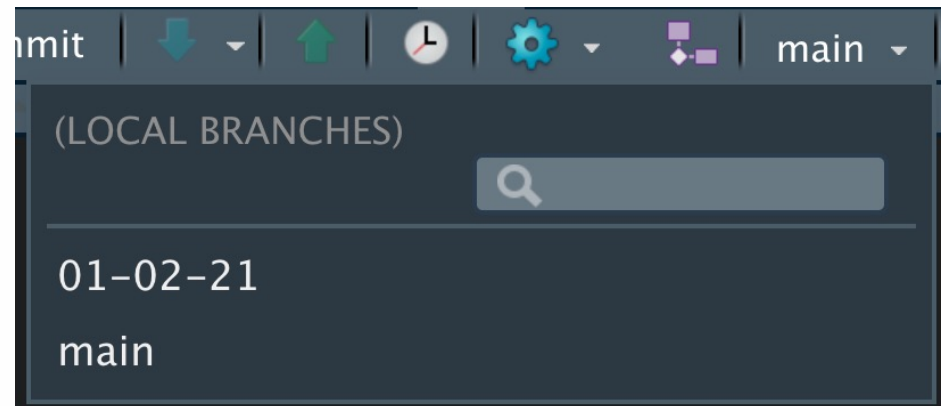


Switch

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.

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

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

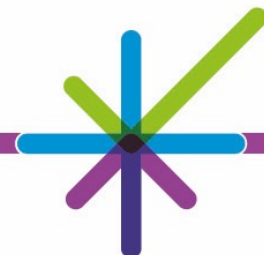


New Branch

Branch Name:

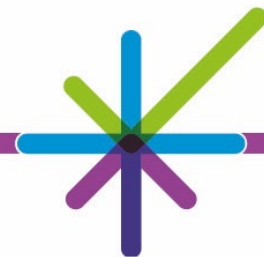
Remote: Add Remote...

Create Cancel



Exercise 4

1. Create a new branch with your name as the branch name.
2. Add a .txt file with a brief reflection on something you've learned/the course does well, and something you'd like to learn/the course could do better. Save as `<name>_reflection.txt`
3. You guessed it, stage and commit!



Merge



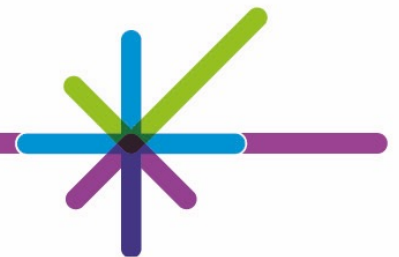
Merging brings the changes from one branch, combining them with another. You should be on the destination branch.

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*



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>
```



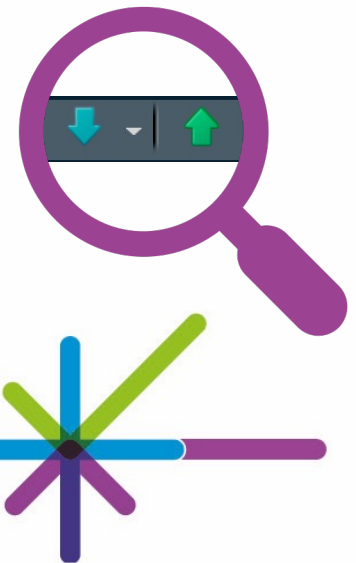
Pulling & Pushing

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.

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

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

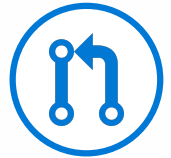
```
$ git push <remote> <branch>
```



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.



Open a pull request

The change you just made was written to a new branch named `feature`. Create a pull request below to propose these changes.

base: master

compare: feature

✓ Able to merge. These branches can be automatically merged.

Update README

Write

Preview

AA B i “ < > ⌂ ☰ ☷ ⌵ @ 📎 ↶

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

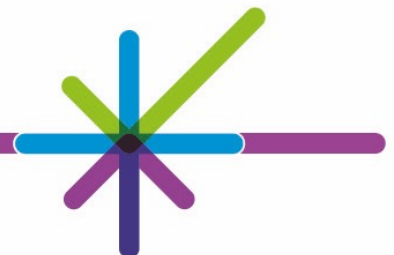
None yet

Projects

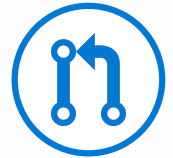
None yet

Milestone

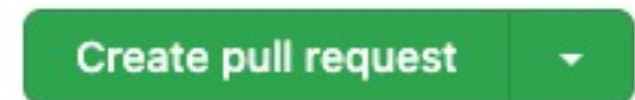
No milestone



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.



Exercise 5

1. Check your remotes, `git remote -v`, you should still be connected to the original GitHub remote.
2. Push the changes from your named branch.
3. Open a pull request on GitHub to merge your named branch with today's dated branch. Add the trainer as your reviewer.



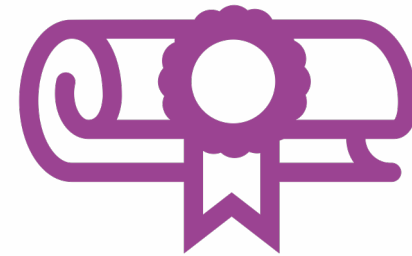
Review

Knowledge Check

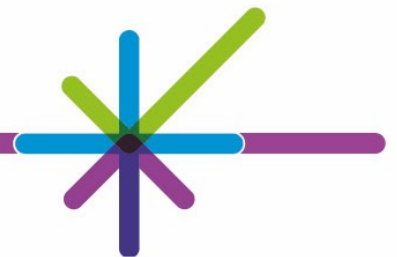
It's time for a quiz!

You'll receive a link or you can click below if you're using the slides.

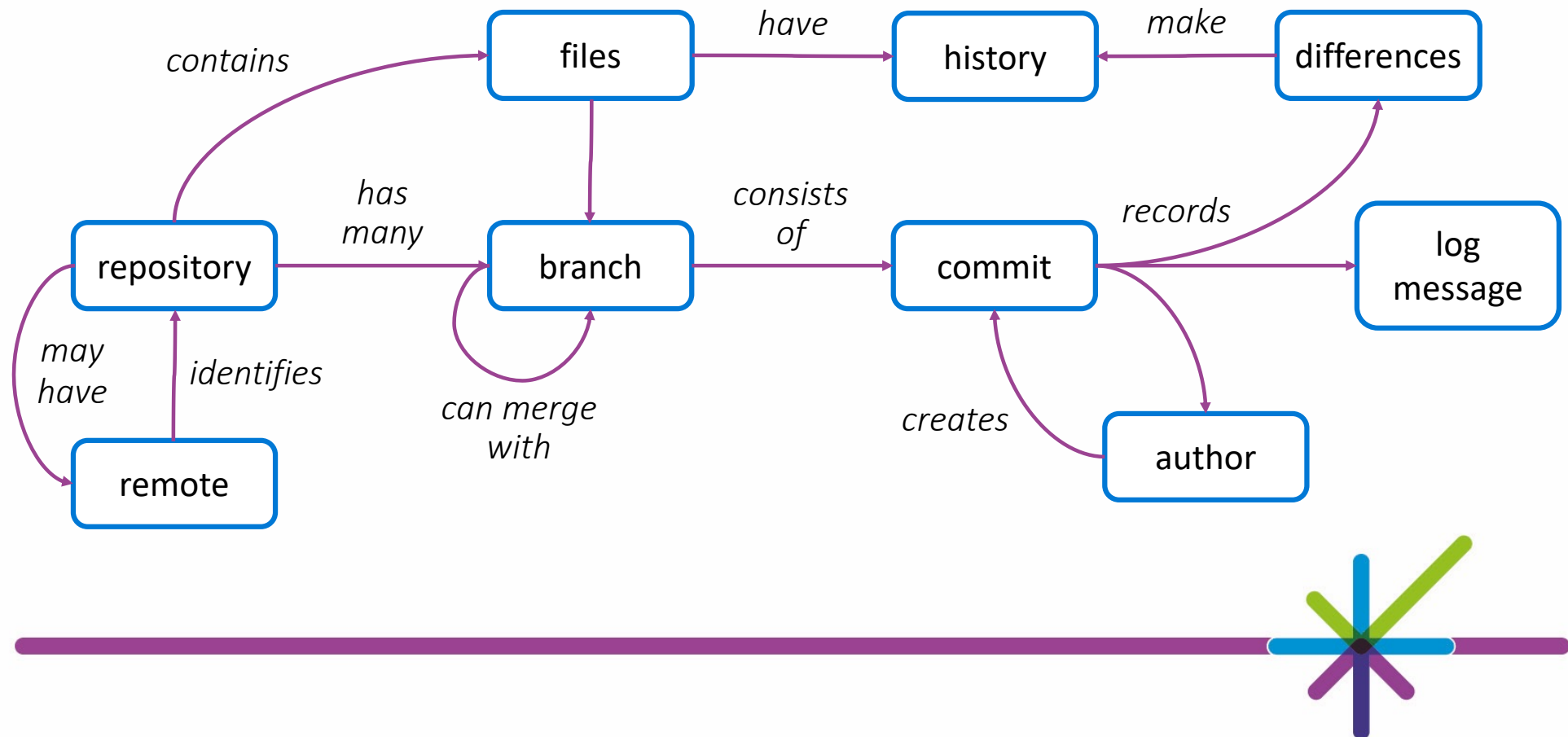
You will receive immediate feedback but we'll come back in 10 mins to review.



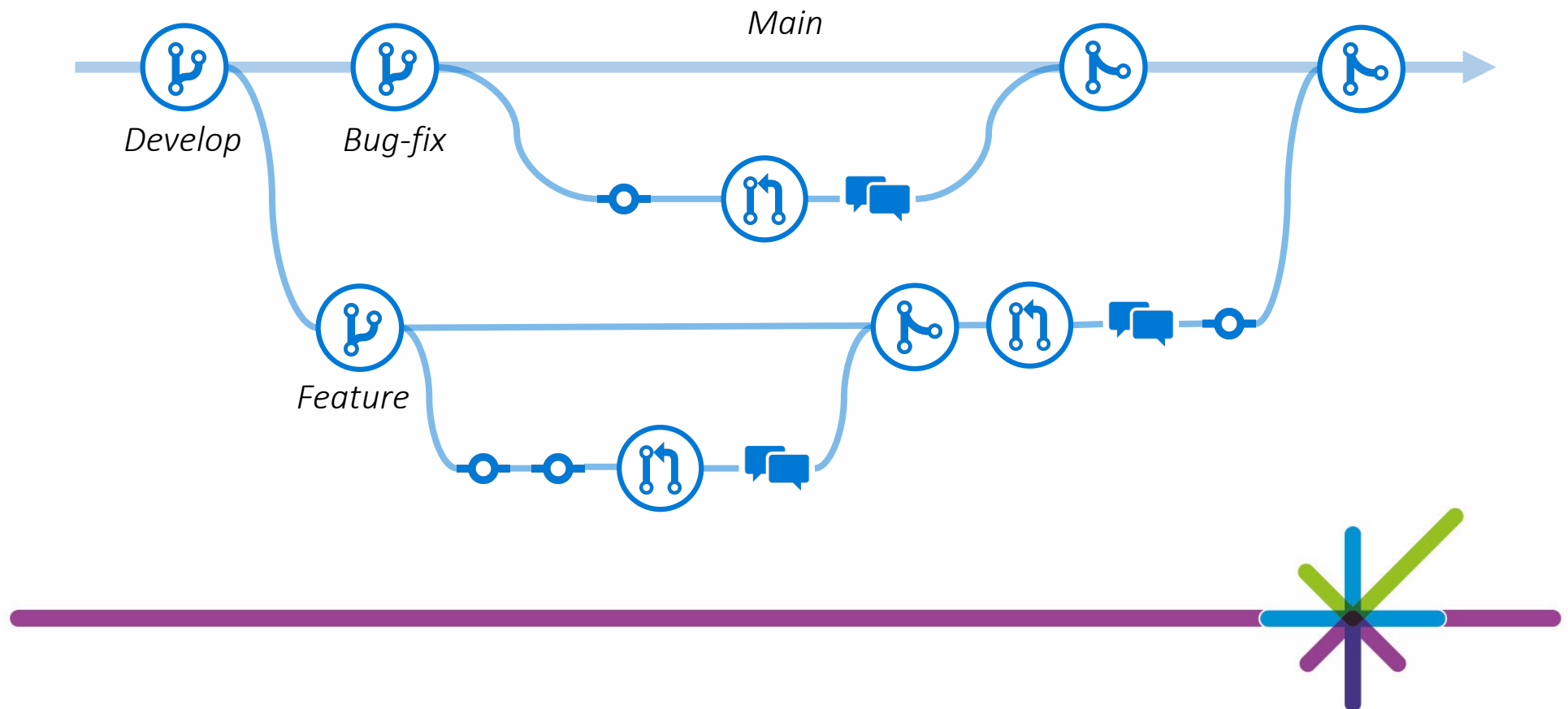
[Start Quiz](#)



A Concept Map



Advanced Workflow



Getting Help

- Google / Stack Overflow
tag queries "[git]" & "[github]"
- [R User Group Teams](#) – [Github and Gitea Queries](#)
- [Transforming Publishing](#)

