Introduction to Git

Learning Outcomes

- What is version control and Git, and why is it useful?
- Setting up Git
- Repositories
- Committing your work
- Branches
- Merging your work
- Merge conflicts
- Pulling and pushing

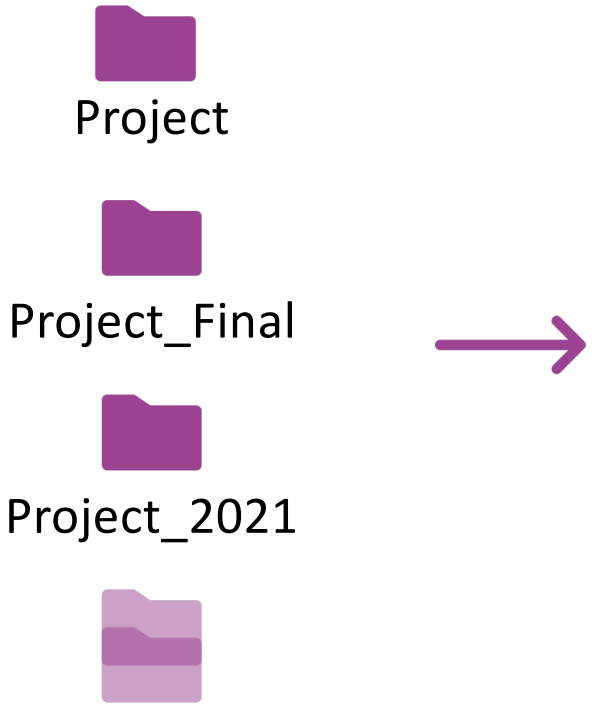
Introduction – the why

Version Control

Tracking and maintaining a full history of changes on a local project which can be shared to collaborate (distributed version control). The system 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

.git

- files
- changes

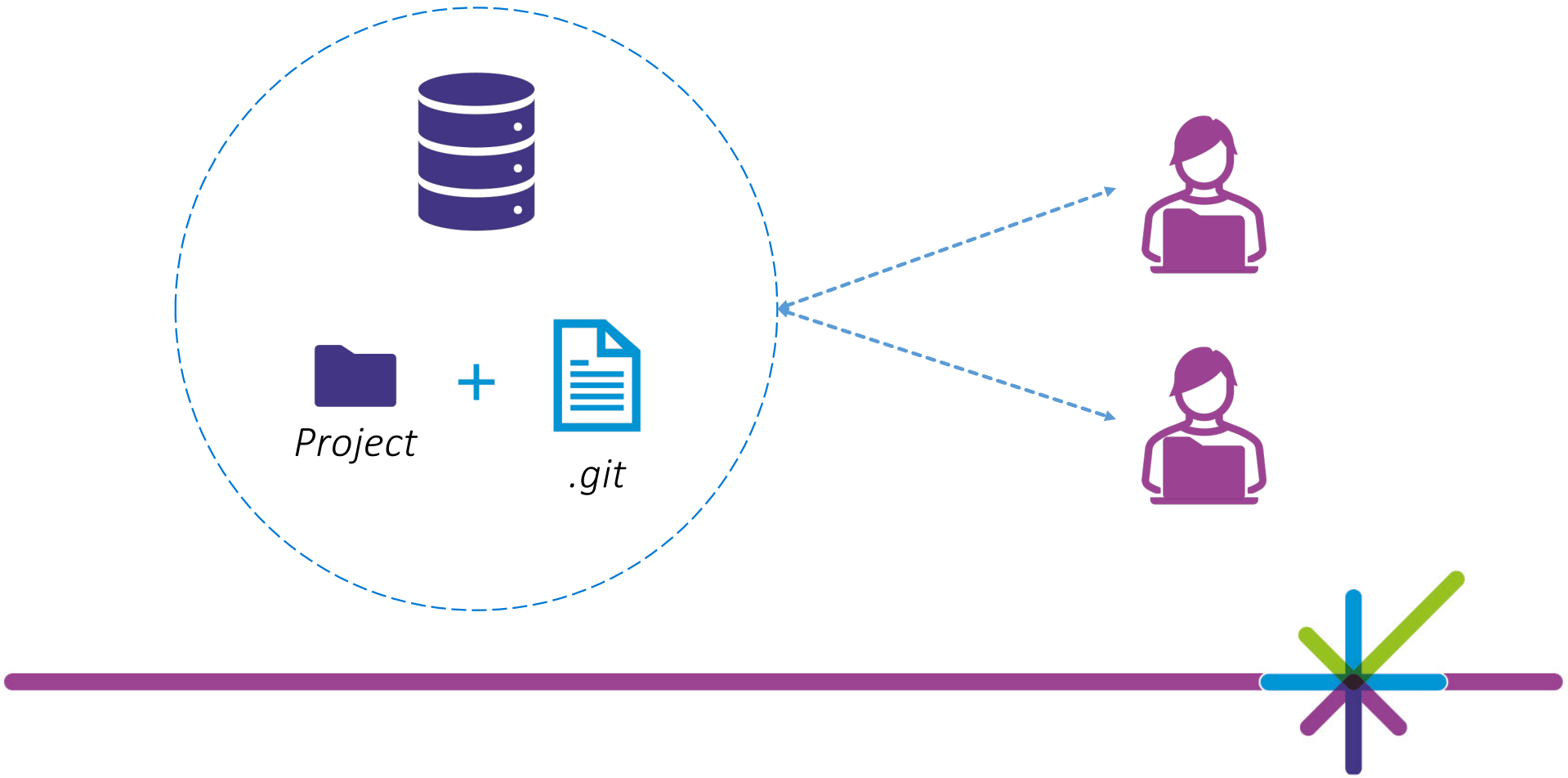
Version control handles all project files with a .git file to track every change.

Version Control - Remote

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

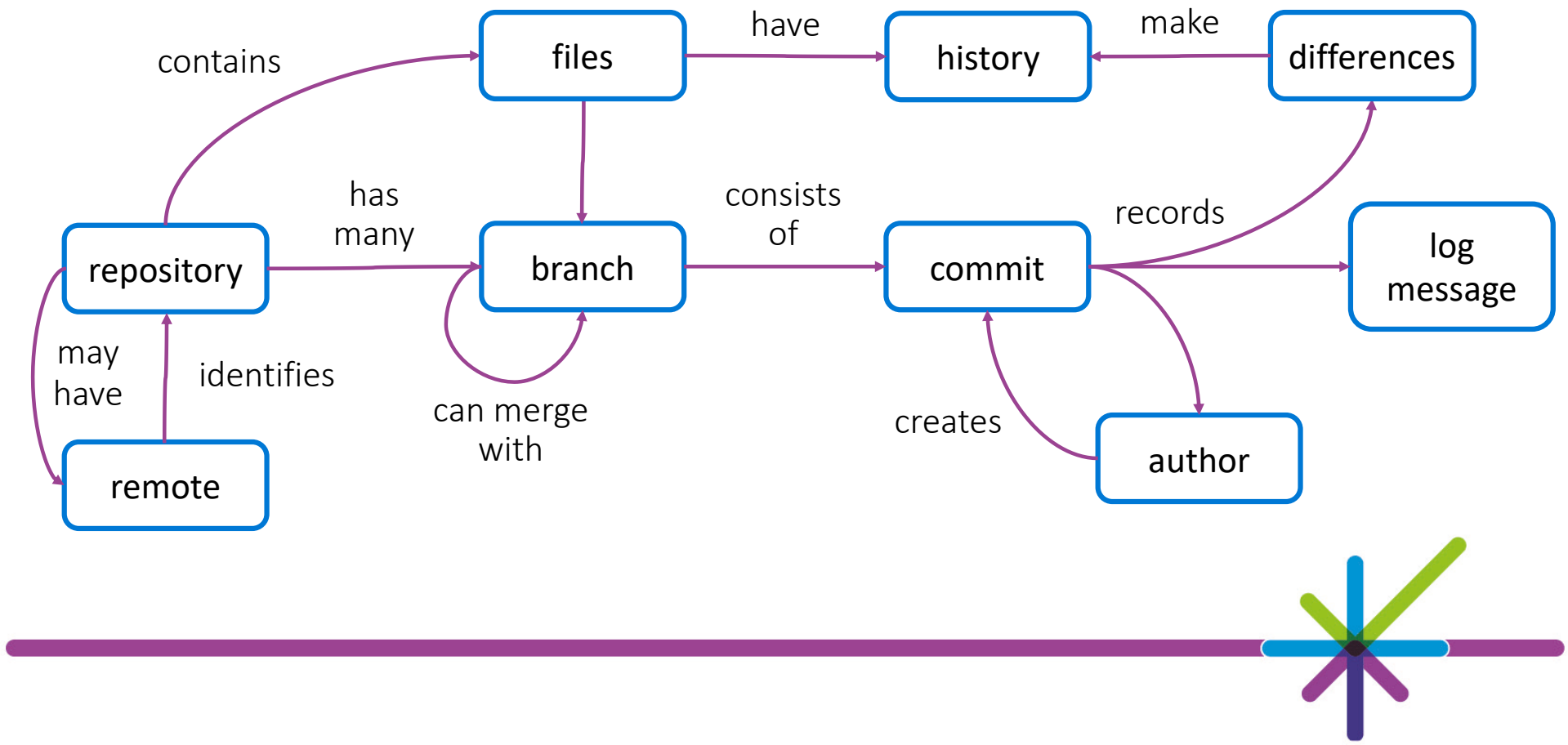
- **GitHub**
- **Gitea** (secure, internally hosted option)

GitHub / Gitea

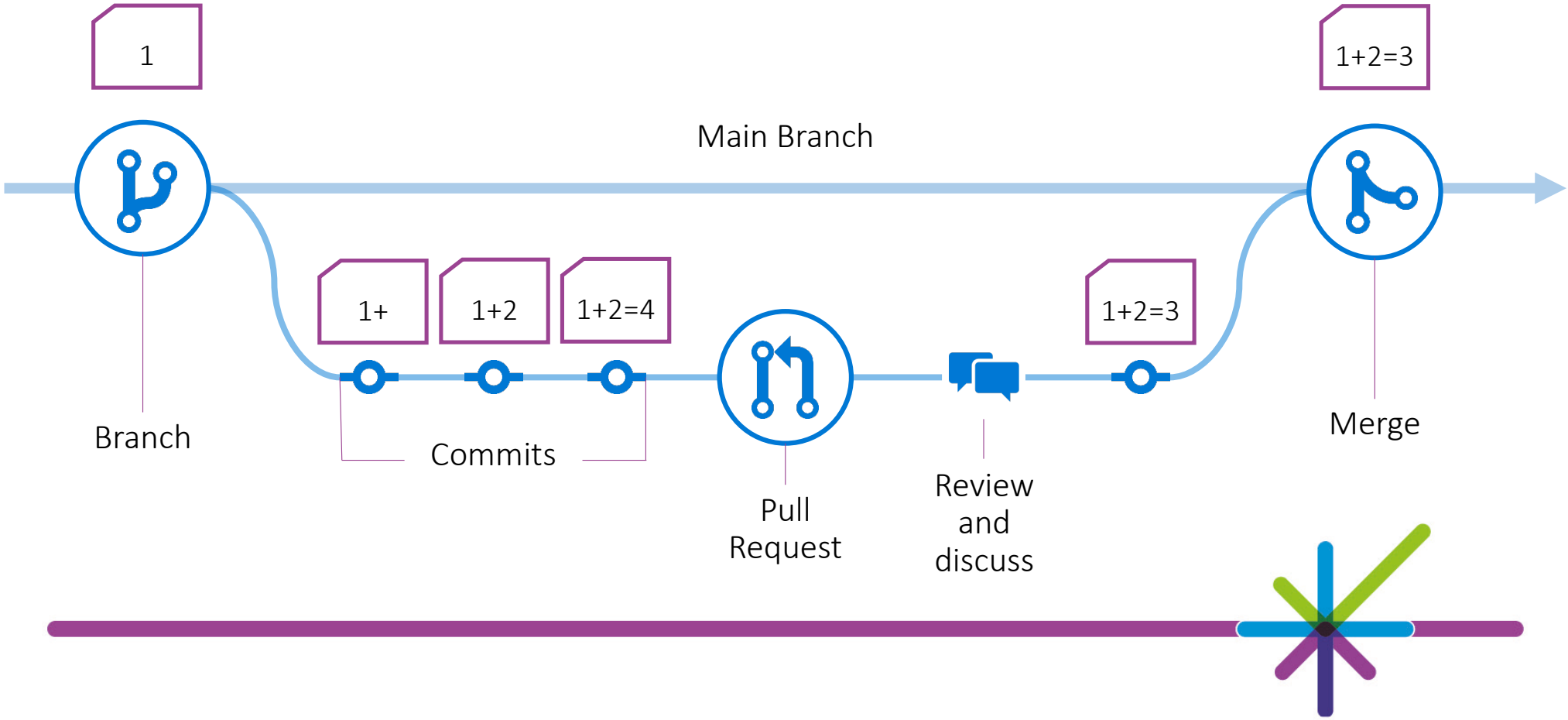


Foundations – the what

A Concept Map



Basic Workflow



Setup

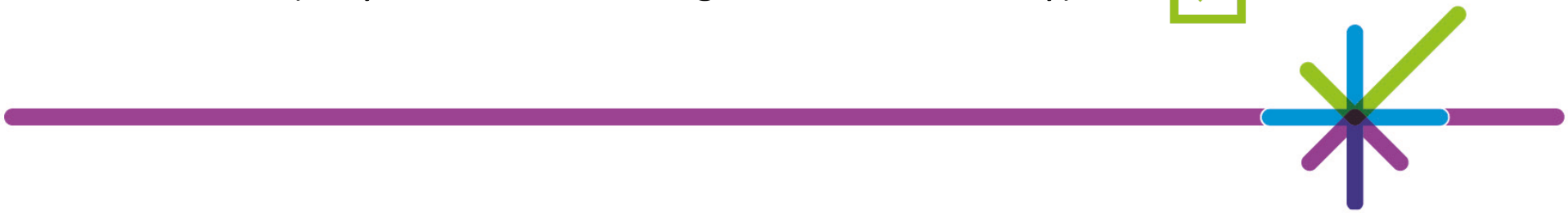
Checklist

Access to Posit Workbench 

GitHub account > added to PHS organisation 

Check Posit Workbench setup and configure SSH key 

Install Git (only needed if working on machine locally) 

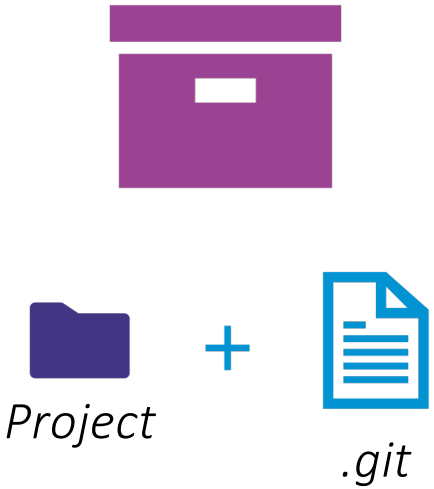


Repositories – the how

Repository (Repo)

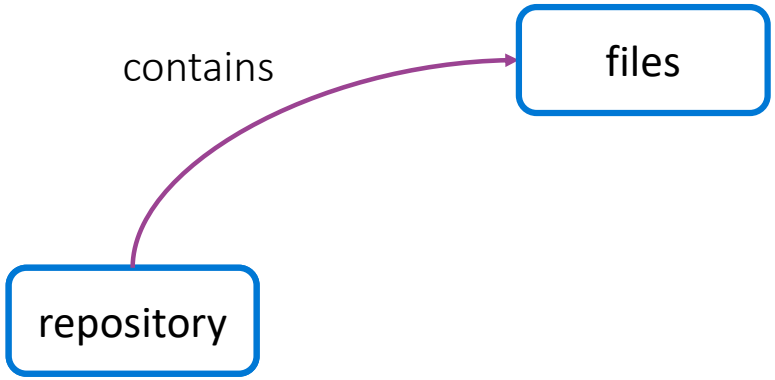
A repo is the directory where project files are stored and is made of 2 parts:

- user generated files/directories
- .git directory



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

A Concept Map



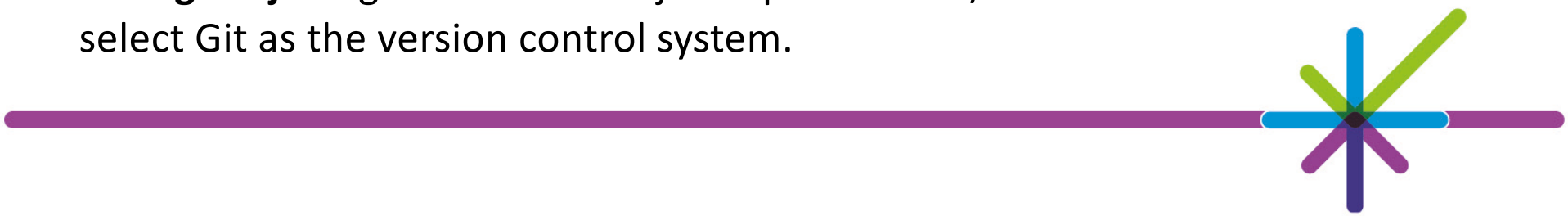
New Repo

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**: Go to New Project > New Directory > New Project:
  select 'Create a git repository' during setup (leave new session box unchecked and RStudio will restart)
- **Existing Project**: go to Tools > Project Options > Git/SVN:
  select Git as the version control system.



Does Git know who you are?

If you haven't used Git in RStudio before you need to set information about who you are. This is done in the terminal and only needs to be completed once.

```
git config --global user.email <email address> [use your PHS email address here]
```

```
git config --global user.name <your name> [use your GitHub username here]
```

Cloning a Repo

Bringing a repo that is hosted elsewhere into a local folder on your computer.

```
git clone <my repo url>
```

In RStudio:

- Go to New Project > Version Control > Git

  enter URL details (leave new session box unchecked and RStudio will restart)

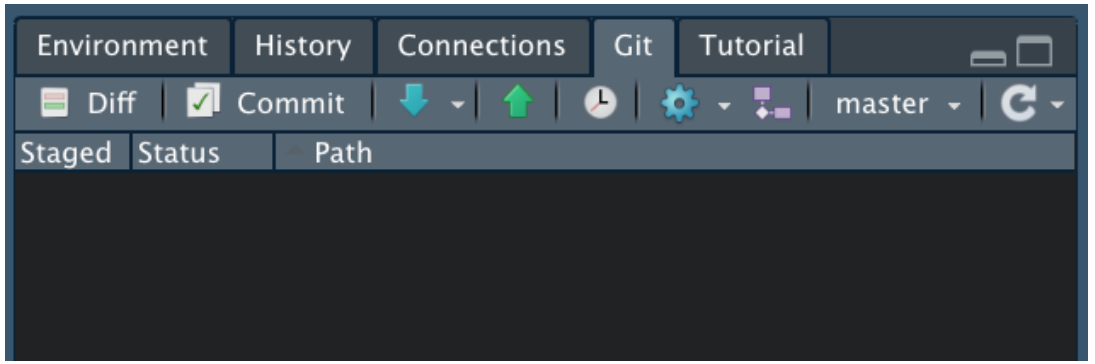
Activity

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

This will display a list of files that have changed since the last commit.

```
$ git status
```



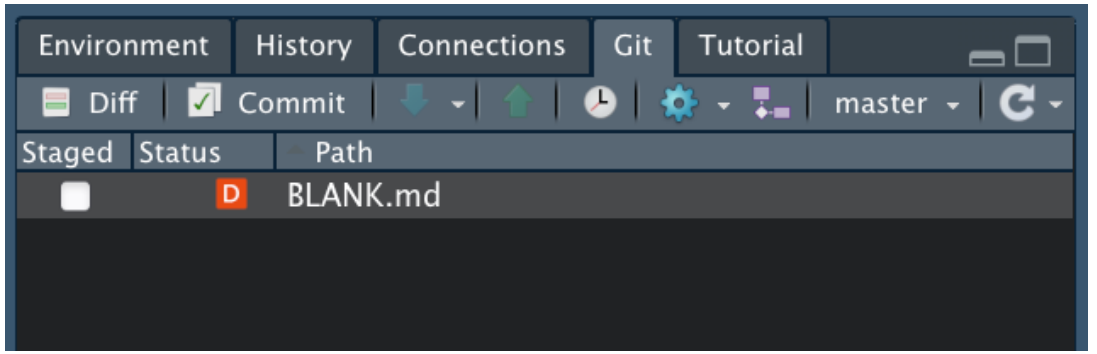
As we've just cloned the repo, there are no changes to see… yet!

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).

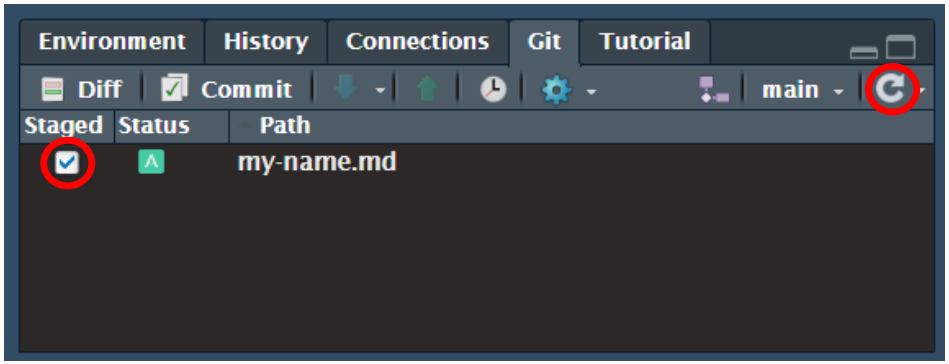
```
$ git status
```



Stage

When we have made changes and want to build a **commit** (a snapshot of changes), we need to stage them using the checkboxes. This interface may need refreshed to see latest changes.

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

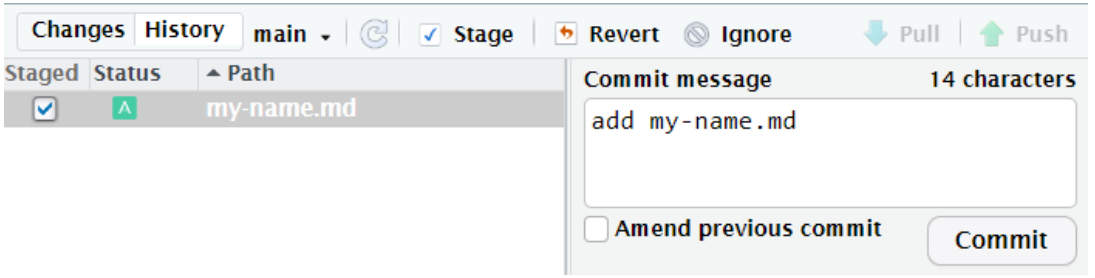


This is like putting together a parcel to send in the mail. You can freely add and remove things (changes), “building” your commit.

Commit

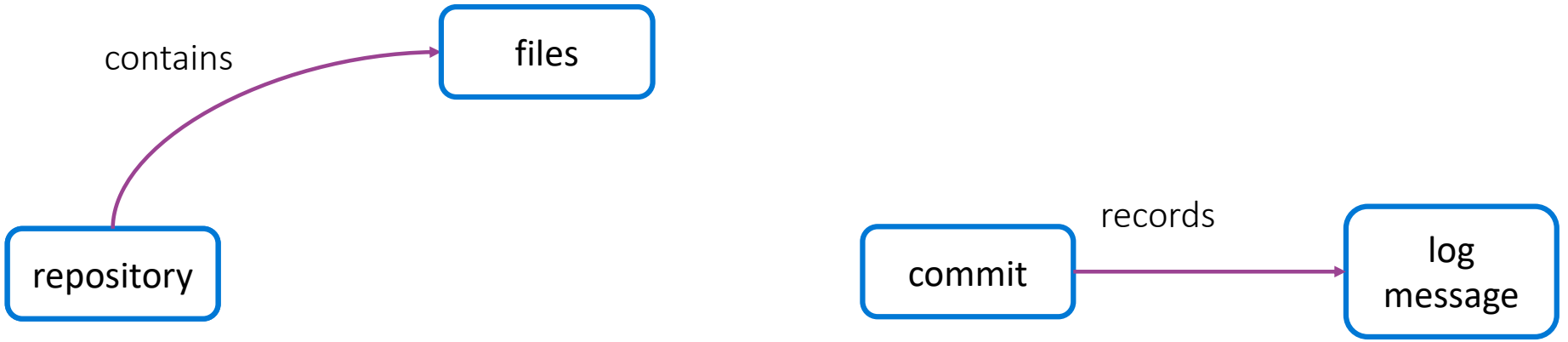
A commit takes everything that has been staged to generate a historical log of changes. You also add a meaningful message to identify changes and reasoning.

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



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

A Concept Map



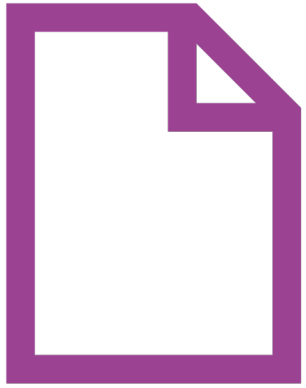
Exercise 1

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.

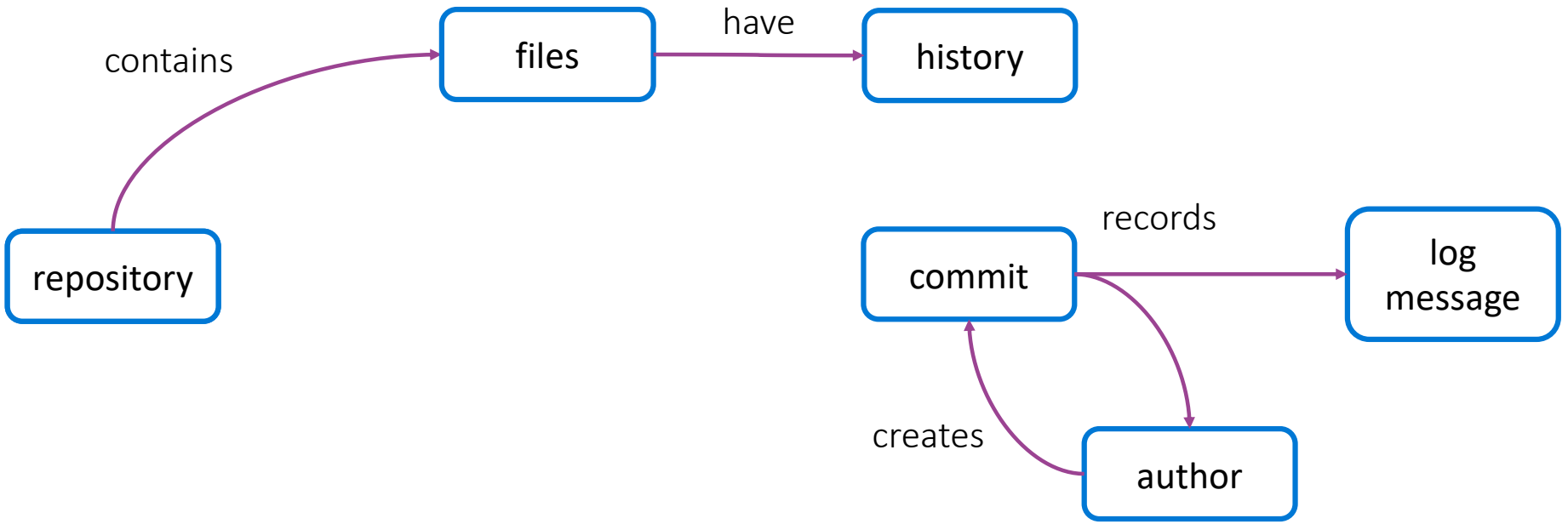
All commits are shown in the log, this is an example of how one is displayed. RStudio's interface is more friendly, click on the clock to explore this repo's history.

```
$ git log
```

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

    initial commit
```

A Concept Map

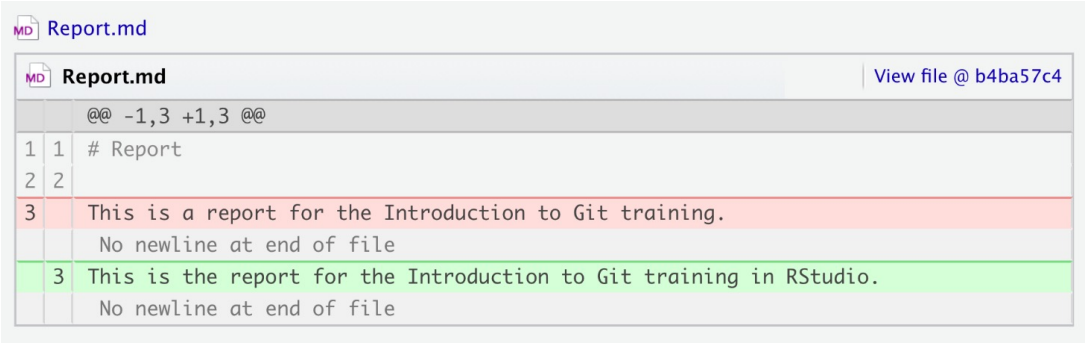


Show

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>
```



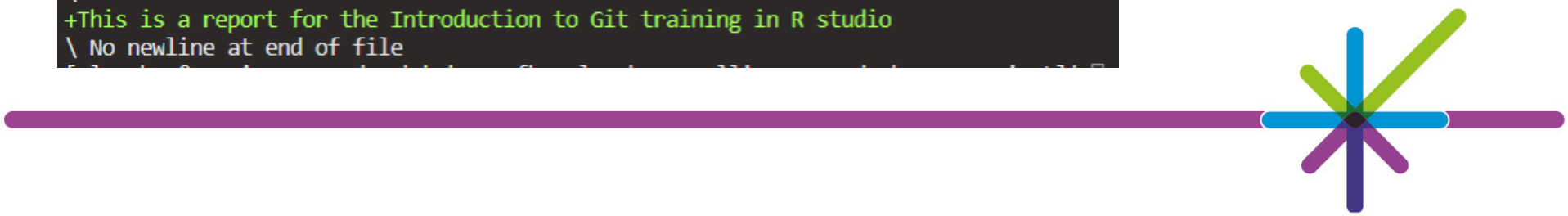
Show – terminal window view example

```
$ git log -1 Report.md

$ git show 211381617b888d5f0ebc9034fc5856bf97fdb595
```

```
diff --git a/Report.md b/Report.md
index dad6bed..6744c63 100644
--- a/Report.md
+++ b/Report.md
@@ -1,3 +1,3 @@
 # Report

-This is a report for the Introduction to Git training
\ No newline at end of file
+This is a report for the Introduction to Git training in R studio
\ No newline at end of file
```



Difference

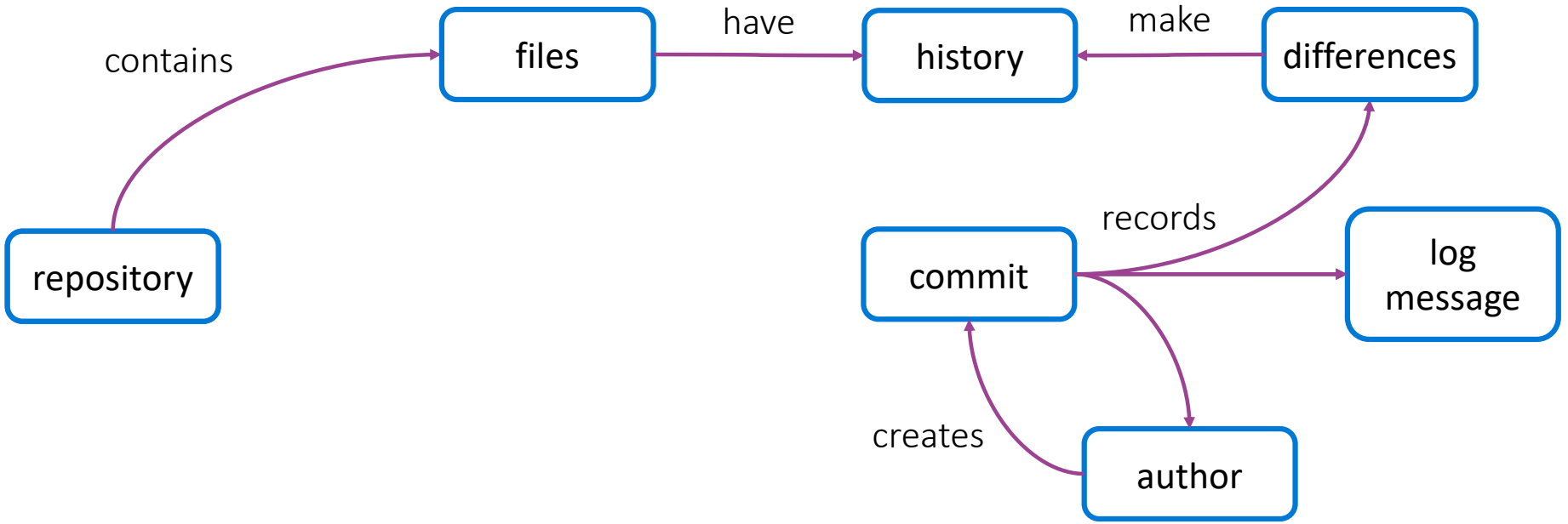
While you can see the changes one commit has made, you can also compare the difference between any two commits. You'll see this command come up again.

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

```
$ git diff <branch>
```

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

A Concept Map



Undo

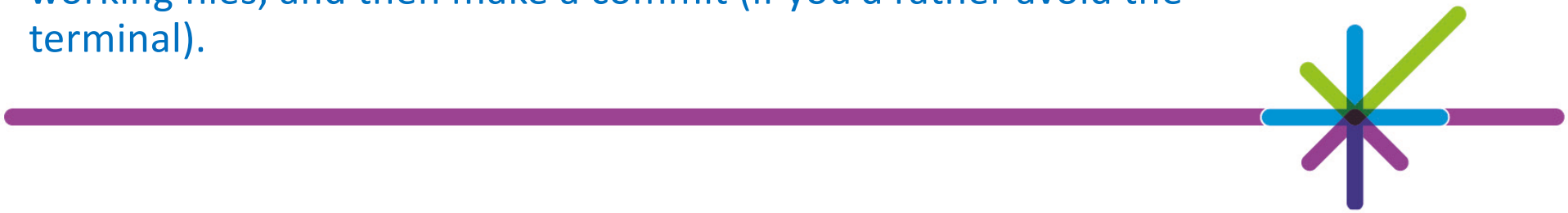
To revert changes to a previous commit, you effectively load the state of the previous commit, and then provide a new commit. This allows a full history where you can make a small undo or even undo your undoing.

```
$ 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. Manually undo all changes created by this commit.

3. Check the **status, stage** and then **commit** with a meaningful message.

Branching – the how

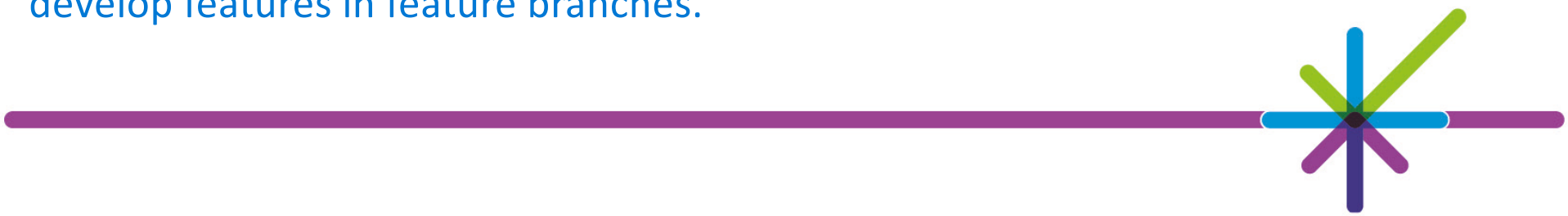
Branching

Branches are parallel versions of the repo, allowing simultaneous work. Changes in one branch do not affect others, until merged.

```
$ git branch

  feature-branch
* main
```

You've already been working on a branch, called 'main' ('master' on old implementations), this is the default primary branch. However, it's best to protect this branch as "production ready" and develop features in feature branches.

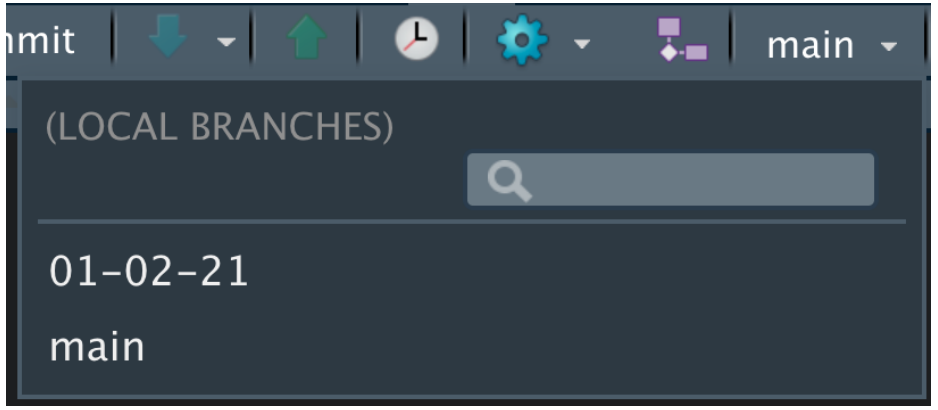


Changing Branches

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

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

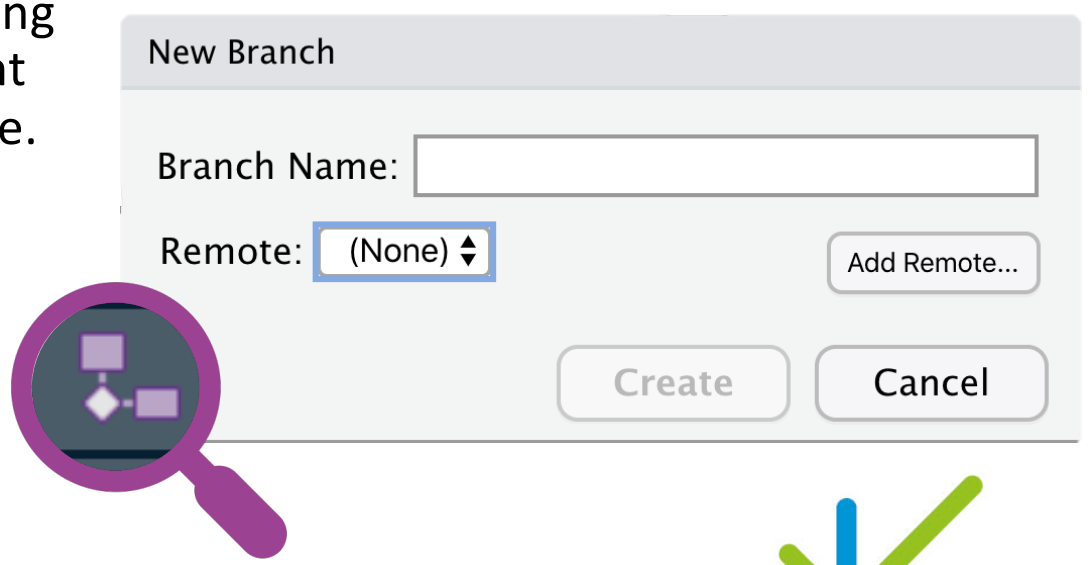
RStudio also provides a dropdown of all existing branches with a search functionality.



New Branch

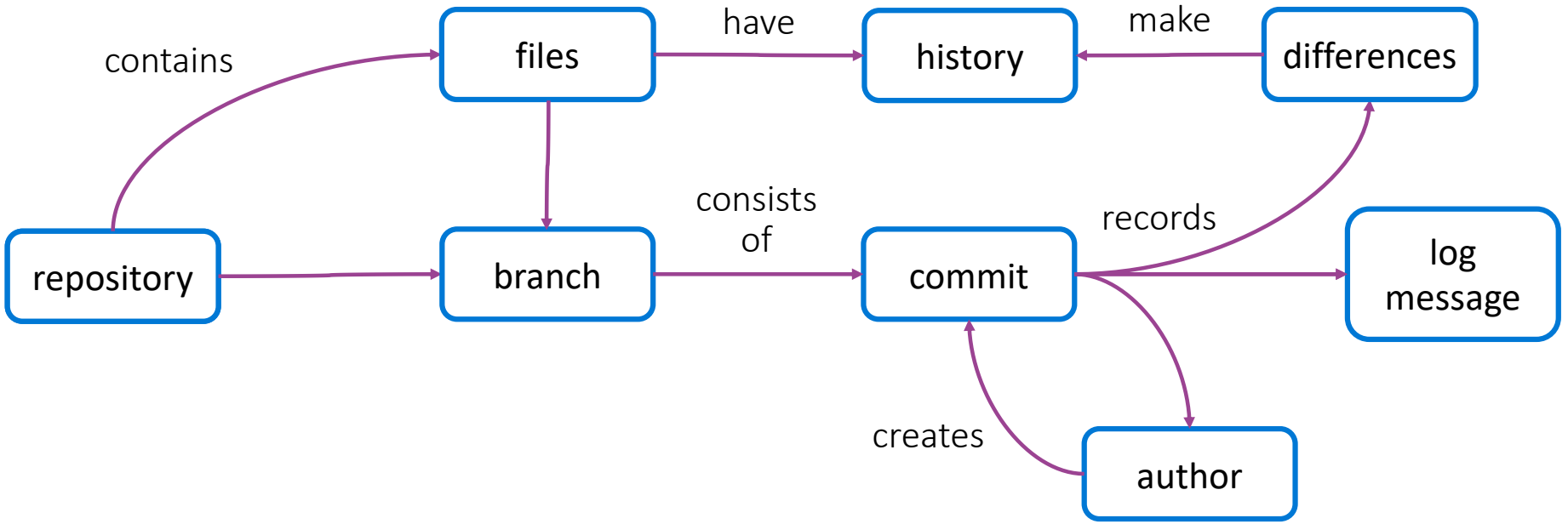
When creating a branch, it's typical to start working on that branch straight away. When we create a branch using these methods, it copies the current branch and switches to the new one.

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



Branches can't have spaces, and, in PHS, the workflow is to sometimes create analyst branches (analyst name) rather feature branches.

A Concept Map



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. Save as 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.

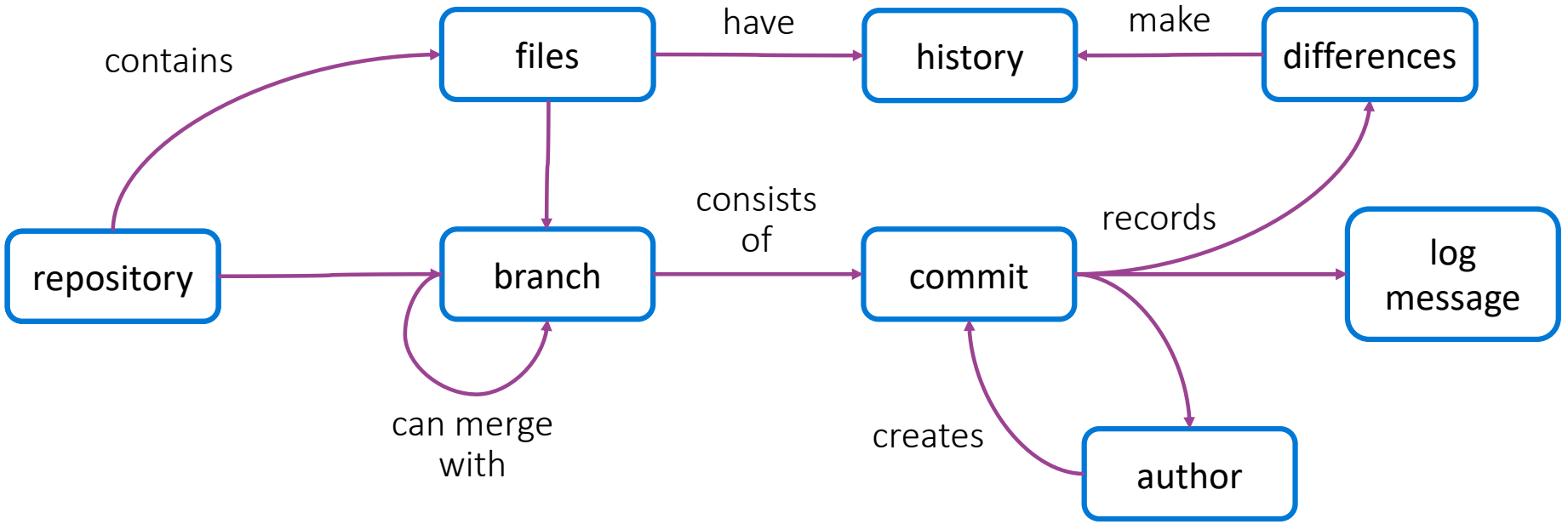
RStudio doesn't have an interface for this function. So, we need to use the terminal and write some Git commands.

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

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

    initial commit
```

A Concept Map



Conflicts

Conflicts occur when you are merging two branches and there are overlapping changes. Git doesn't know which version of the code to keep.

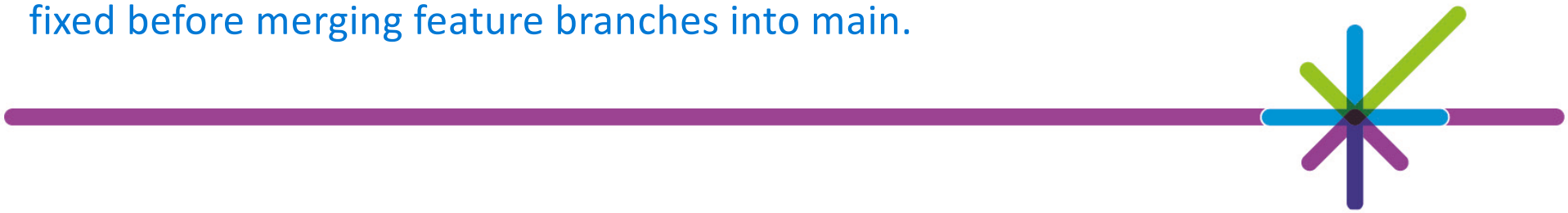
Git will add markers inside the file to identify the issue.

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

Resolving Conflicts

Conflicts are resolved by removing the markers, making required changes to the code (keep one change, two or none), then committing those changes. GitHub has an interface for resolving merge conflicts, but Gitea does not.

To avoid conflicts, use branches and agree on scope of work to prevent overlap. The latest version of the main branch should also be pulled into your feature branch and any discrepancies fixed before merging feature branches into main.

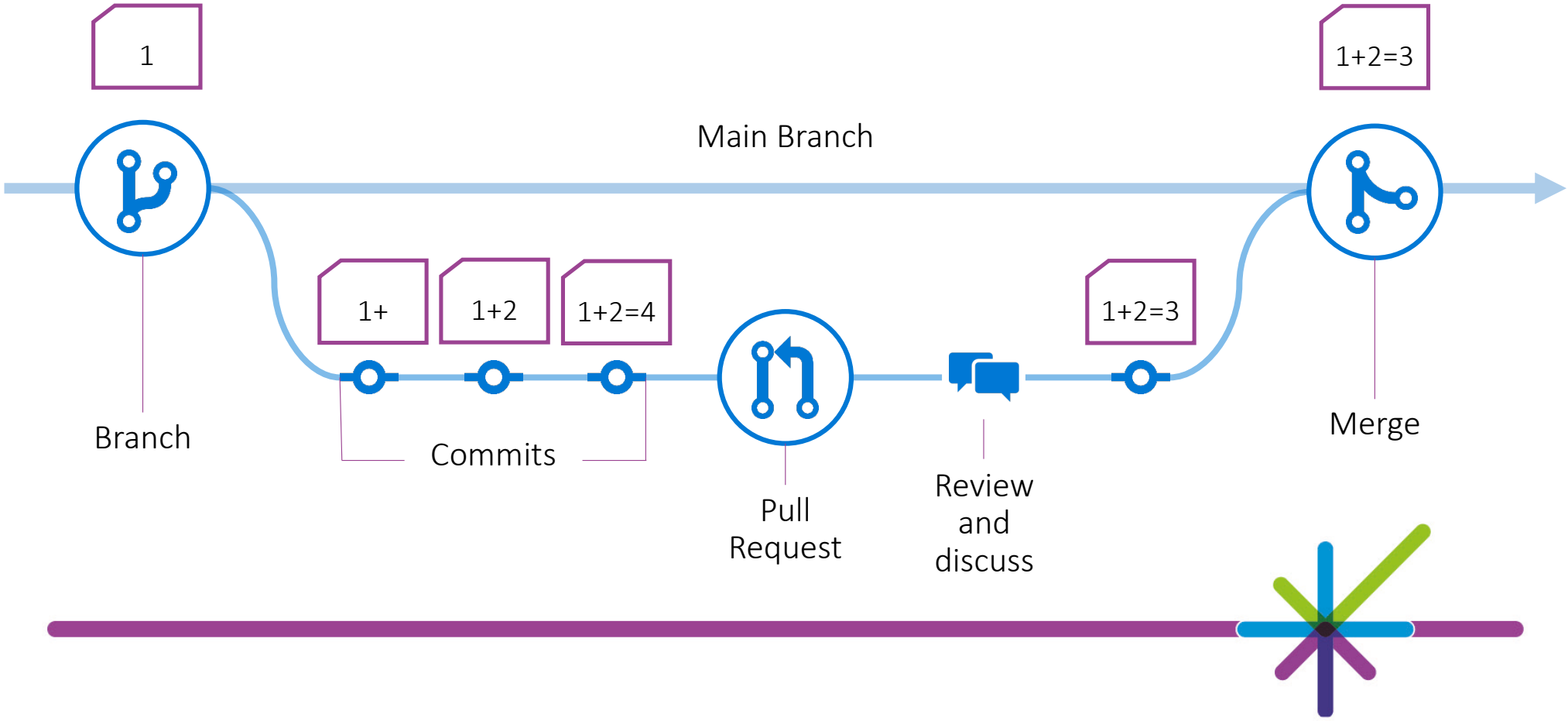


Deleting Branches

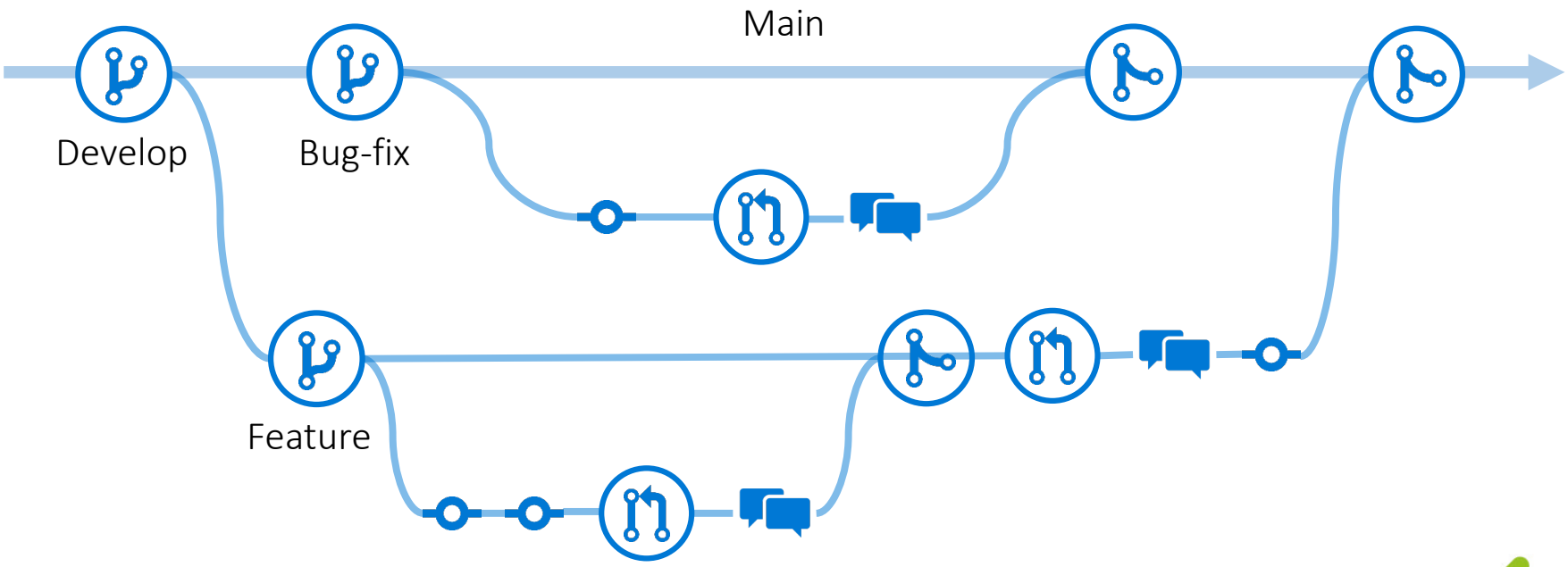
When using branches for features, once they've been merged, there's no longer a need for them. Remember that merging has combined all changes. As such, it's best practice to delete the branch.

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

Basic Workflow



Advanced Workflow



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.

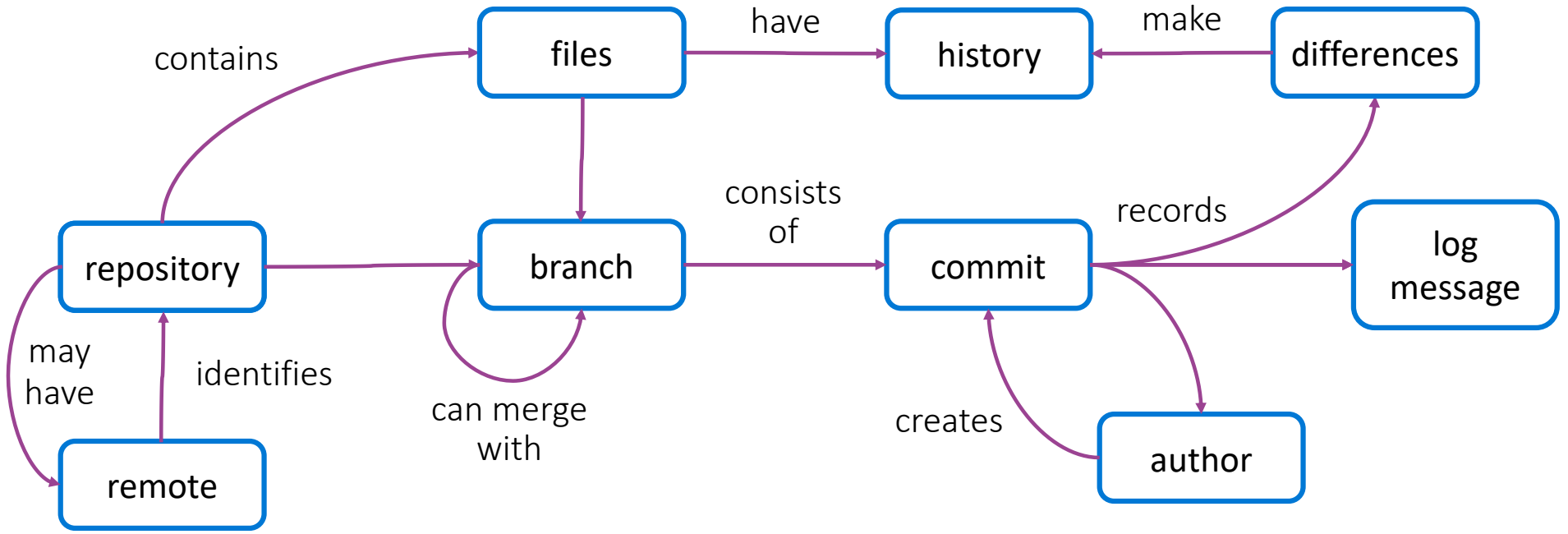
Add/Delete Remotes

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>
```

A Concept Map



Pulling & Pushing

Pulling allows changes made available in the remote repo 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>
```

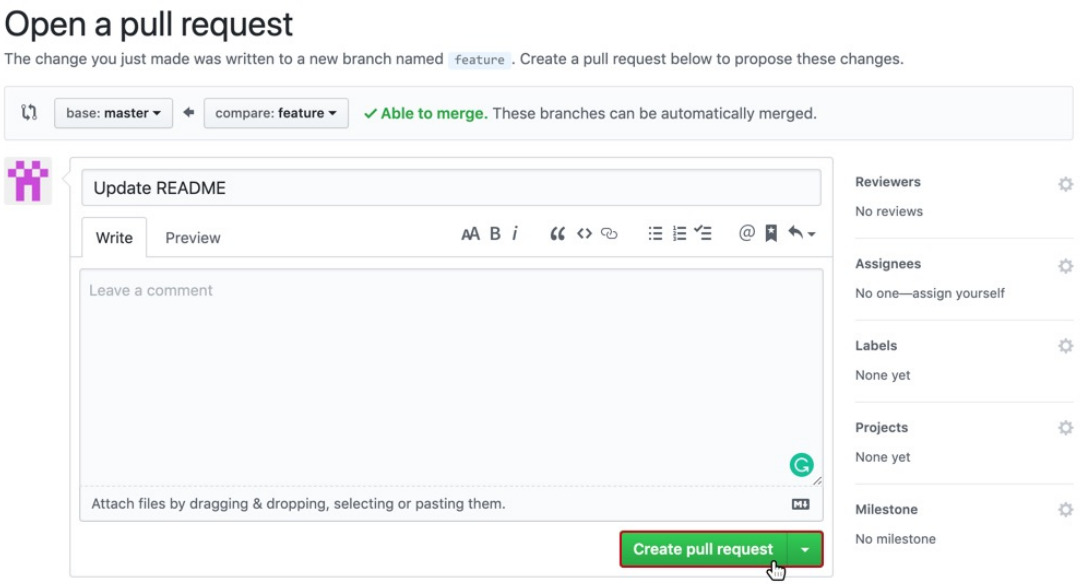
Pushing shares changes made locally and committed, back to a remote repo that is hosted on, for example, GitHub.

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

Pull Request

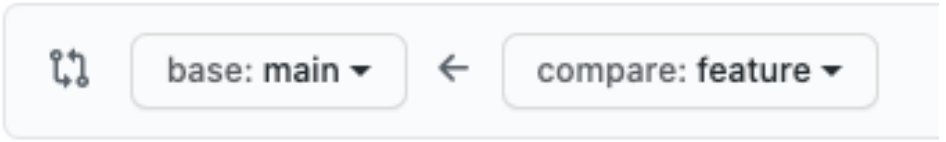
When merging a branch, it's advised to utilise a pull request. This is like a code review stage and adds some protection to any higher branches. It's also a great way to learn for the reviewee and reviewer.

The pull request must be generated from the remote.



Pull Request: How-To

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.

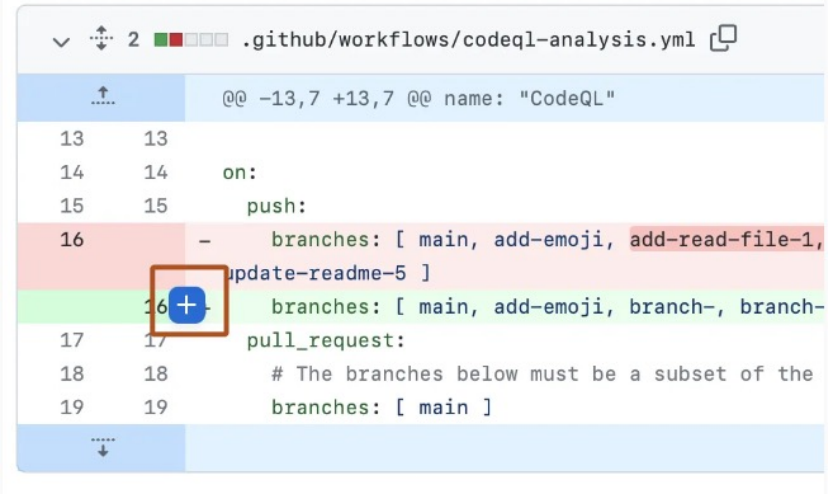


Code Review – How to

You can review code and add comments in GitHub and Gitea.

Pull requests > Select PR

In the Files Changed tab, you can add comments to specific lines. You can also add a general comment and “Approve” or “Request changes” for the overall PR.



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.

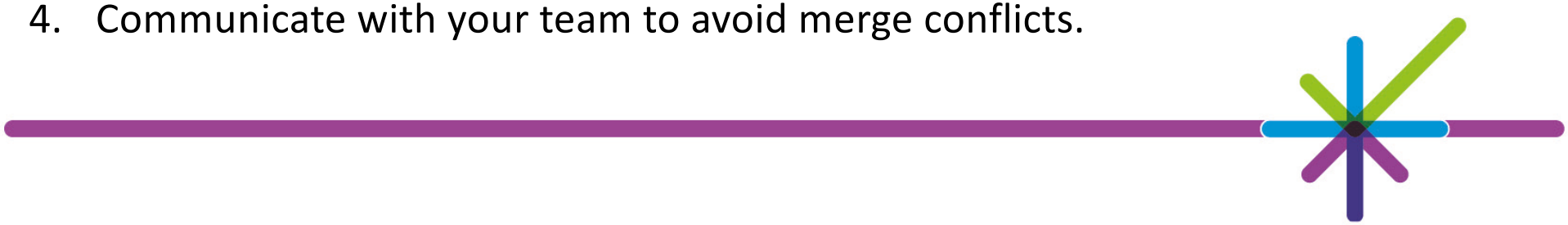
Remotes - Tips

1. When creating a new branch, ensure you're on the branch you want to branch from (usually main) and pull it to get the latest changes from the remote.

2. Keep commits small and targeted, with useful messages.

3. Pull from the remote regularly, especially when collaborating, to keep local files up to date.

4. Communicate with your team to avoid merge conflicts.



Activity

1. Go to GitHub repo: github.com/Public-Health-Scotland/learn-git

2. Guided code review of pull requests

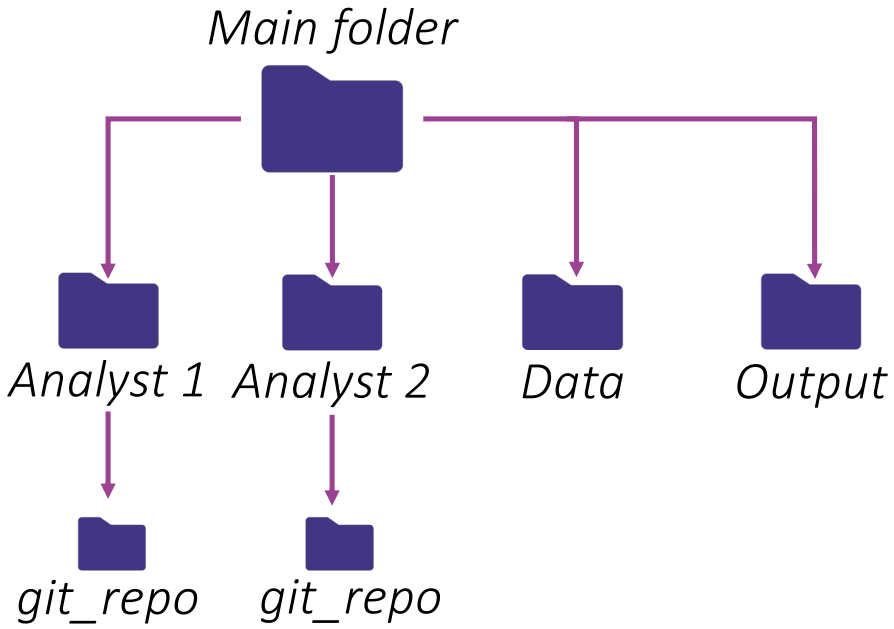
3. Handle a git conflict.

Using Git in your team

When using Git in your team, it is good practice for each analyst to have their own folder and clone of the repository.

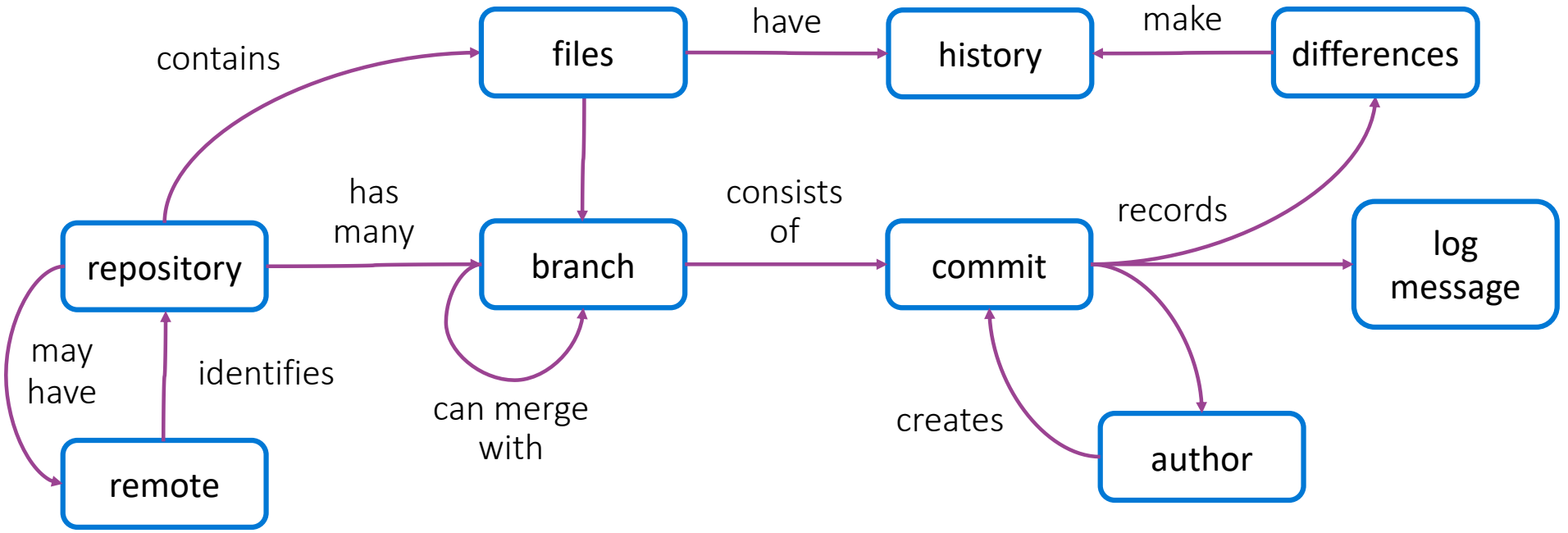
Depending on your team's folder structure, you can have analyst folders in each work folder (image) or one analyst folder with multiple git repo's in.

Avoid having Data and Output folders in the repo folder, but still include data files in your gitignore file to be safe.

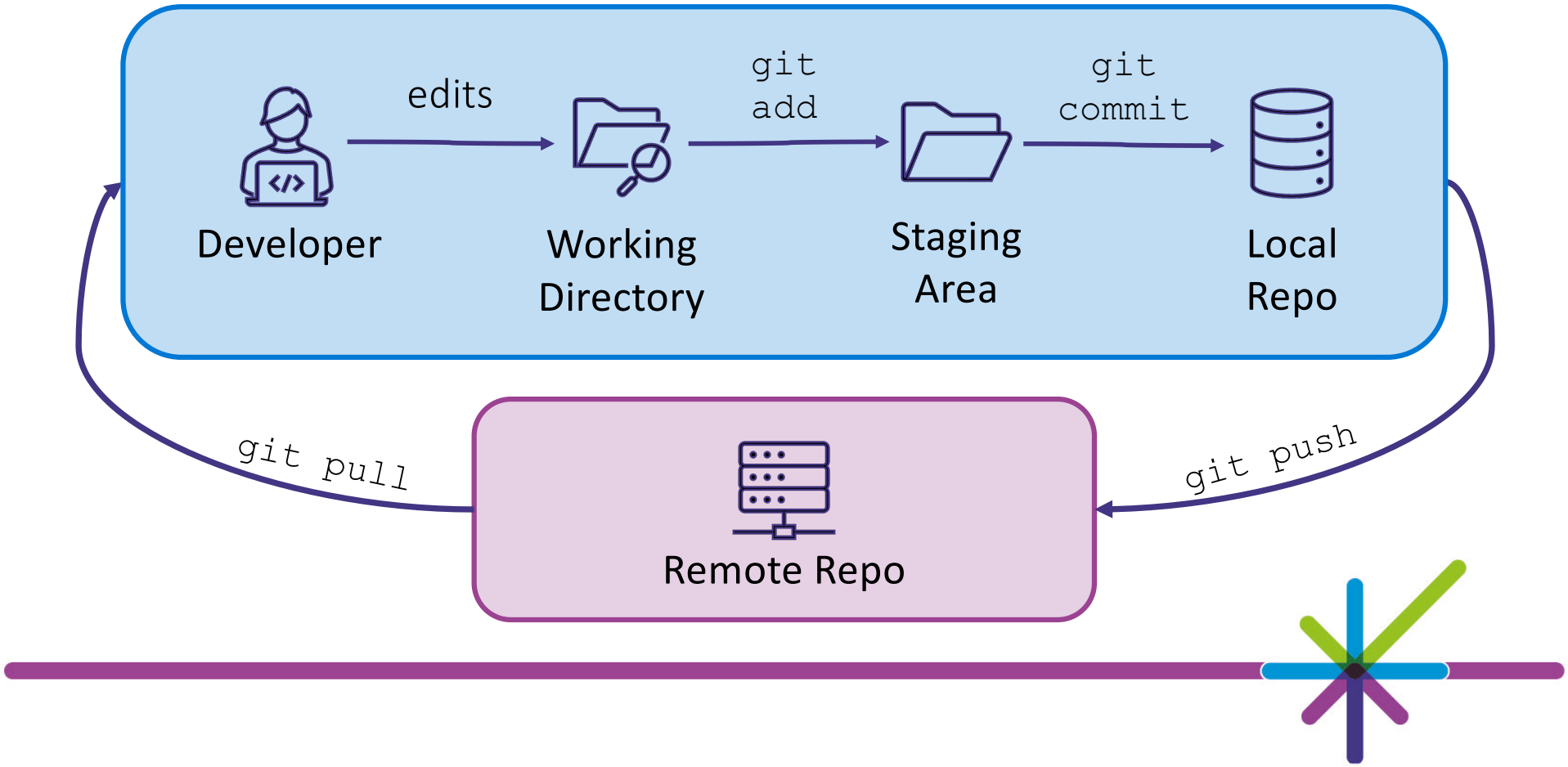


Review

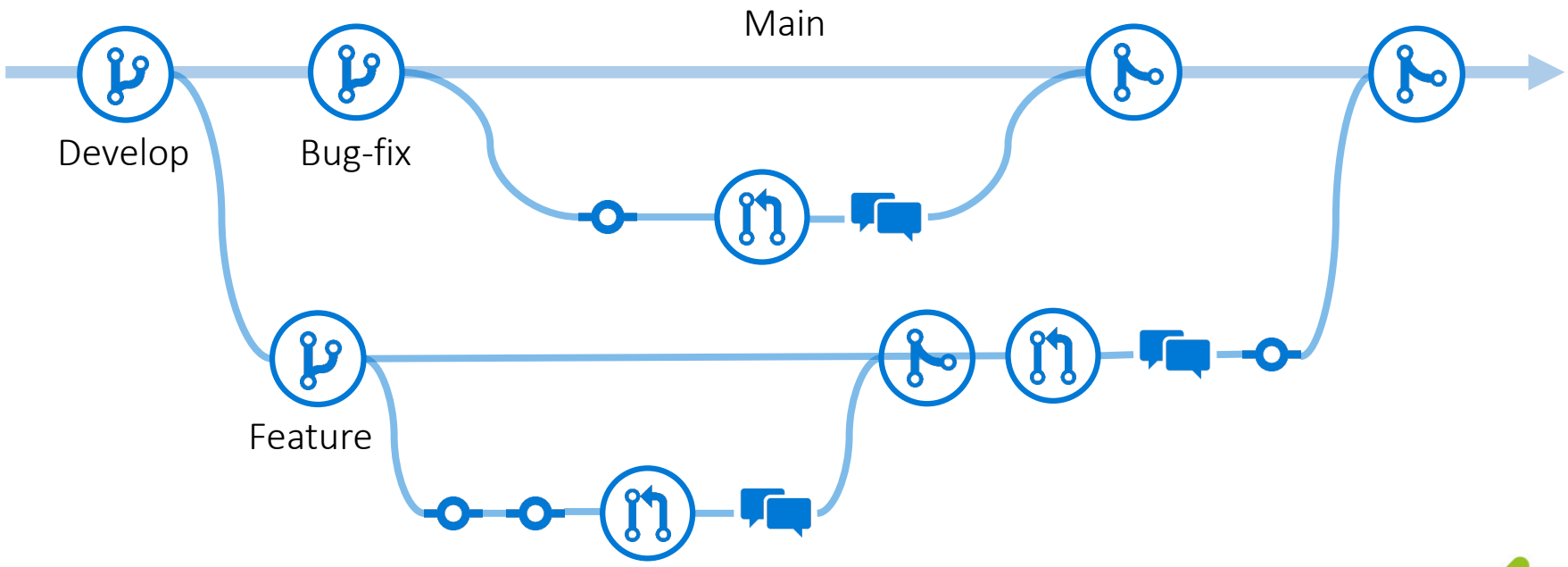
A Concept Map - review



Git System


edits
git
add
git
commit
Developer
Working
Directory
Staging
Area
Local
Repo
git pull
git push
Remote Repo

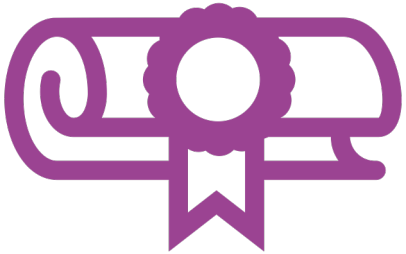
Advanced Workflow - review



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

Getting Help

- PHS Quick Guide to Git & GitHub
- Google / Stack Overflow

  tag queries "[git]" & "[github]"
- PHS Data and Intelligence Forum – Github and Gitea Queries

Course feedback

Please take a few minutes to complete the course feedback form

PHS Introduction to Git Training - Feedback Form (office.com)