

# Collaborating with Git

Teamwork makes the dream work

Presented by:  
Joe Wilson and Helen Richardson



# Objectives

- Why should you collaborate using Git
- Working with Branches
  - Branch Naming
  - Working with Branches
  - Creating a Branch
  - Changing Branches
  - Merging Branches
    - Updating
    - Pull Requests
    - Merge Conflicts
  - General Guidelines
- Exploring GitHub and the Open-Source Community



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

*"Merge branch 'asdfasjkfdlas/alkdjf' into sdkjfls-final"*

# Why should you collaborate using Git?

Collaborating with others on a project can sometimes get messy.

Changes can be tangled together, and hard work might be overwritten accidentally and lost.

Git allows you to create branches and separate out those changes.

Any new changes are first made to the branches, keeping the main branch clean and stable.

Once the changes have been made, reviewed and approved, they can be merged safely into the main branch.

This model of using branches with a shared repository is called the **Shared Repository Model**.



# Different ways to collaborate and write code

## Remote Repository Hosting Platforms

- **GitHub**
- GitLab
- Bitbucket



## Integrated Development Environments

- **GitHub Codespaces (VSCode)**
- RStudio
- Visual Studio Code (Desktop)
- PyCharm



# Naming your branch

It is good to have an agreed scheme in your team and your repositories.  
We typically advise:

**<branch type>/<project id>-<initials>-<ticket number>-<brief  
description>**

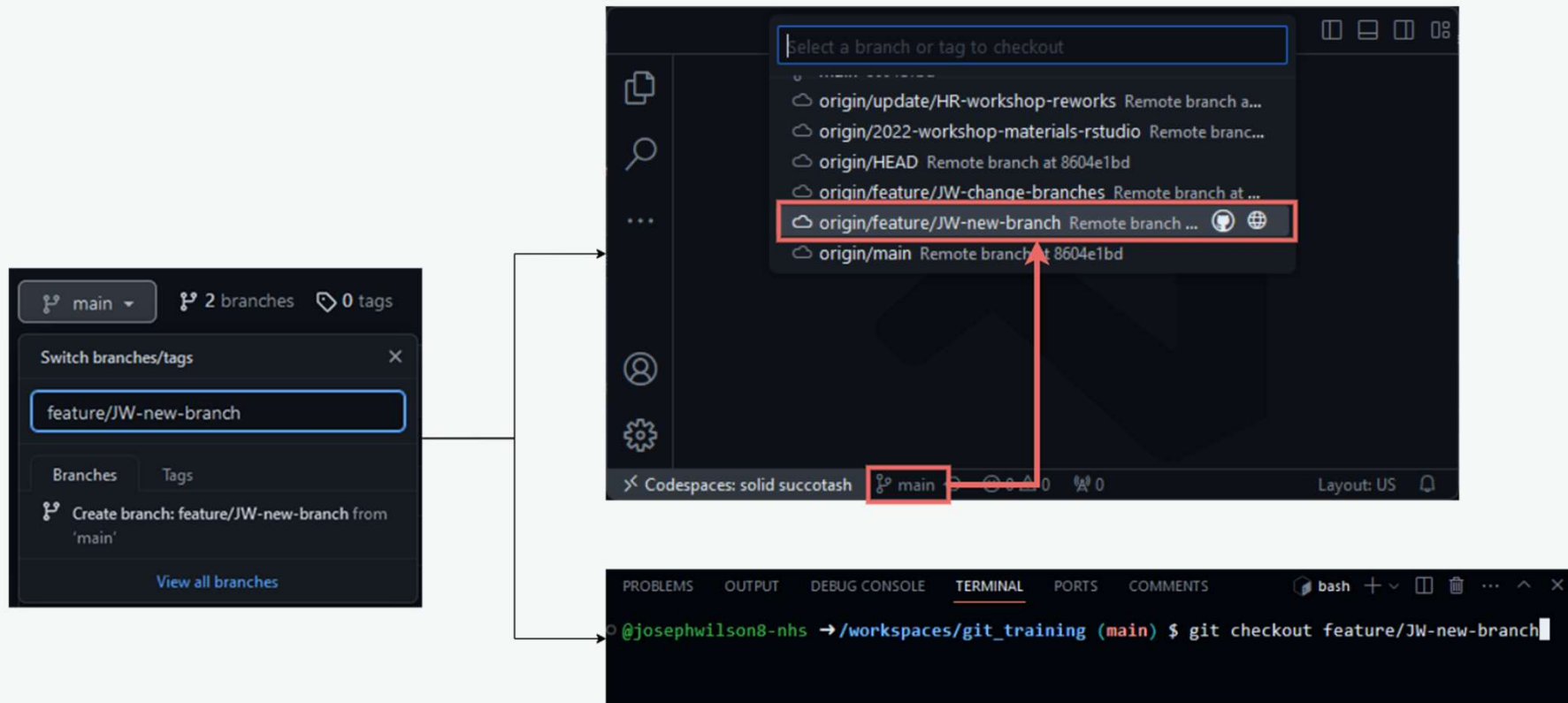
`feature/JW-123-example_feature`

Preface your branch names with the type of branch it is going to be.  
Typically, the types you will have would be:

- **feature/ - for adding new features**
- **release/ - for preparing or preserving a batch of changes for delivery**
- **hotfix/ - for emergency fixes to your main branch (if it is protected)**

# Creating a branch

We can do this remotely via GitHub and pull it in to our local repository.



# Creating a branch

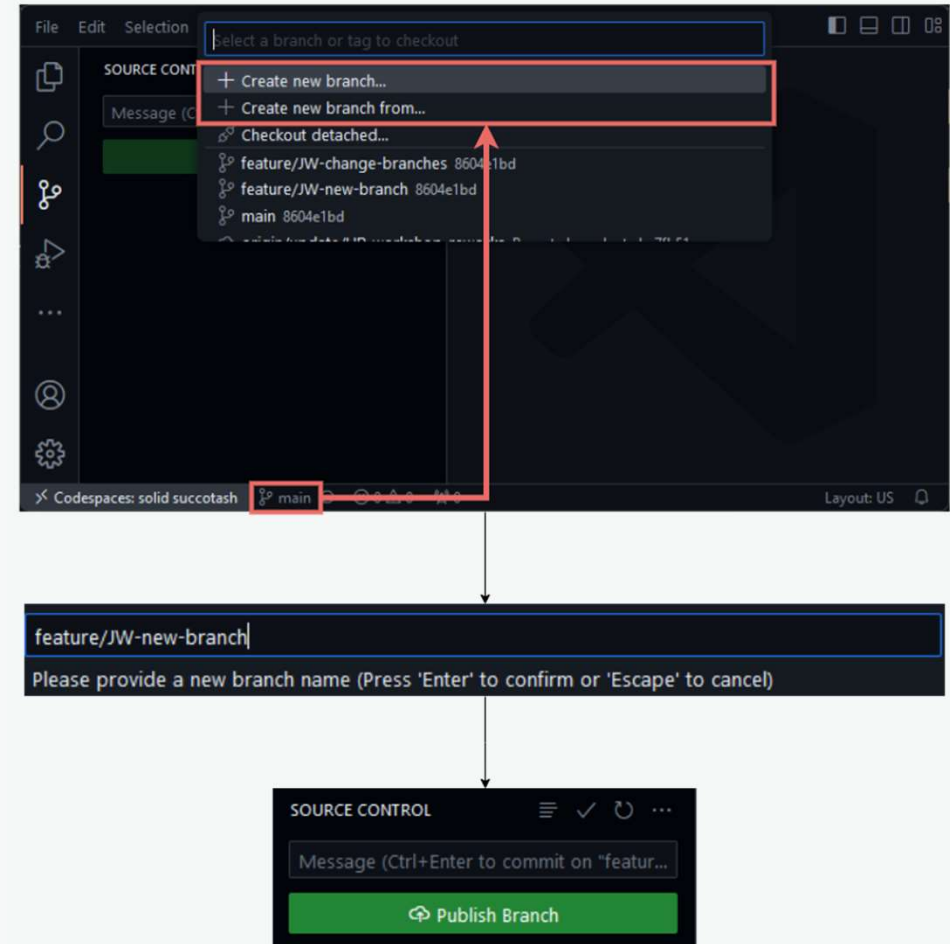
We can also create the branch locally and then push it up to GitHub from our Codespace. We can do this using the UI or the Terminal/Shell.

## Using the Terminal

```
git checkout -b feature/JW-new-branch
```

```
git push --set-upstream origin  
feature/JW-new-branch
```

## Using the UI





# Let's try it out

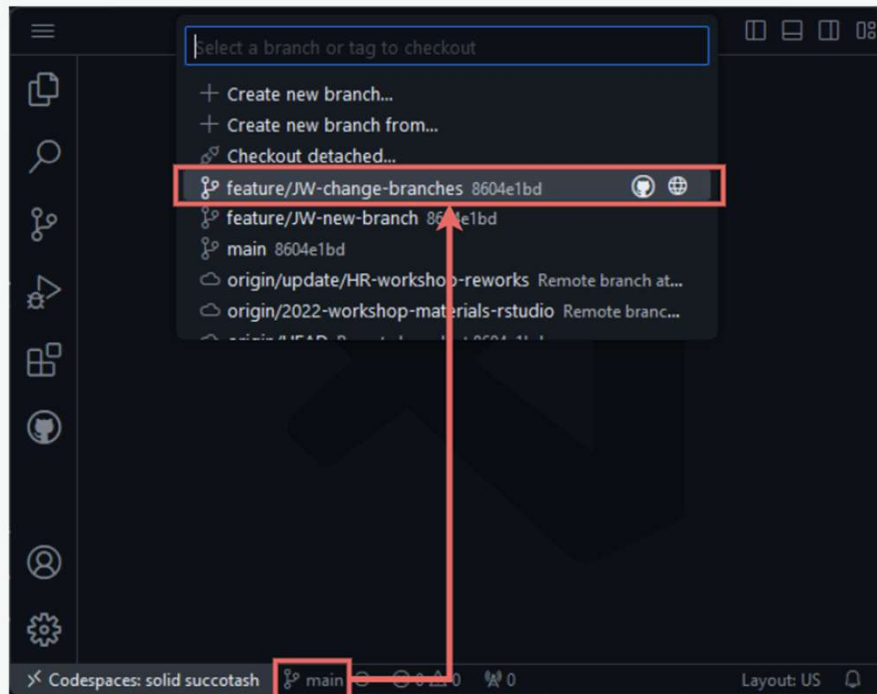




# Changing branch

It is useful to know how to change between branches.  
Again this can be done via the UI or the Terminal.

## Using the UI



## Using the Terminal

```
git checkout feature/JW-change-branches
```



# Let's try it out





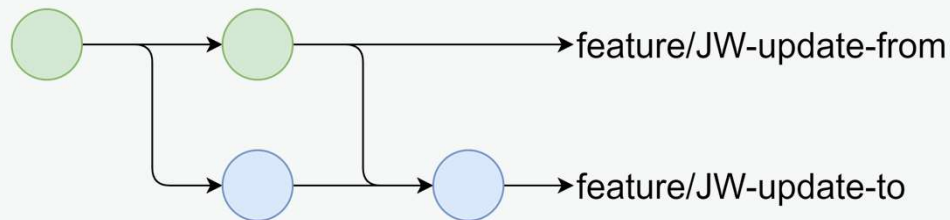
**Let's have a  
short break**



# **Welcome back!**

# Updating your branch (Merging)

As you collaborate and work on multiple branches in your project's repository, you might want to ensure that your branch is updated with changes from other branches. This allow us to prevent future conflicts or resolve current conflicts.



We can merge branches together in the terminal:

```
git checkout feature/JW-update-from  
git pull  
git checkout feature/JW-update-to  
git merge feature/JW-update-from
```

Commands to run before  
merging to ensure that you  
are working with the latest  
version



# Let's try it out



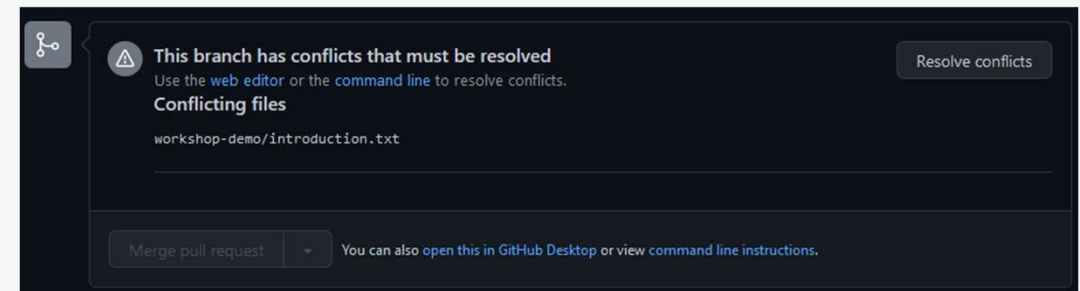
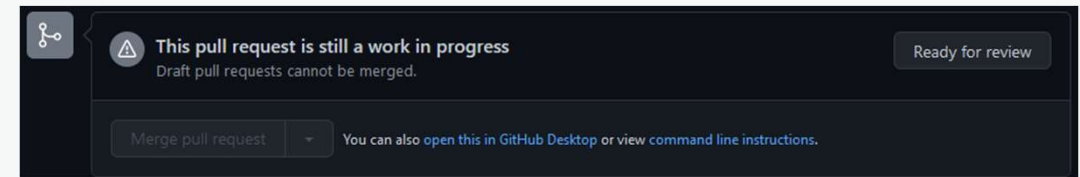
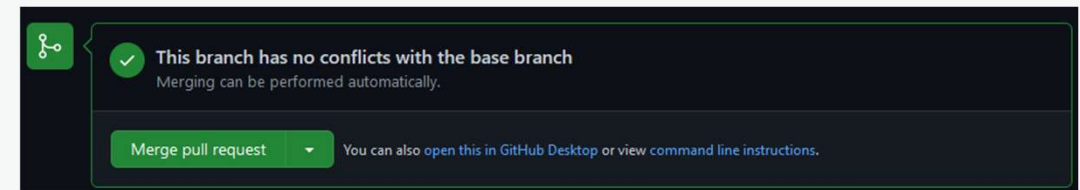
# Pull Requests (Merging)

Pull Requests can also be called Merge Requests and can be an important part of the collaborative process.

Pull Requests are normally used when you want to merge your changes into the main branch.

Pull Requests allow you to assign a reviewer, allowing someone to peer review your code. Make sure to update your branch before you make your Pull Request.

GitHub will show a warning if there is a conflict occurring and a merge cannot take place safely.





# Let's try it out



# Merge Conflicts

While Git is an extremely powerful tool, it needs human intervention to resolve situations where there are conflicting changes.

Conflicts are usually caused by the same lines being changed on both branches.

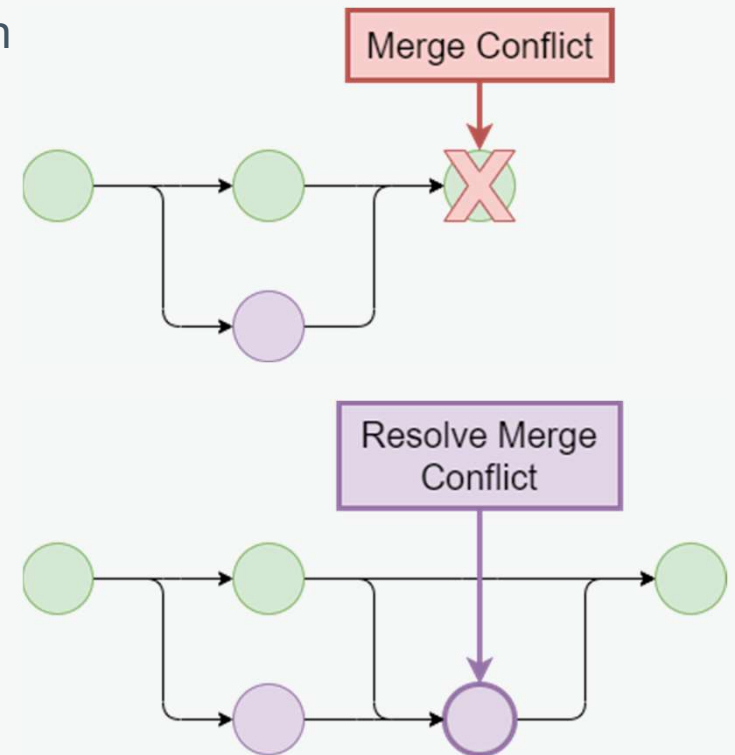
It will attempt to automatically merge the branches, like it did when we updated our branches, but when it encounters a conflict it will stop auto-merging and show a CONFLICT error, highlighting the files where conflicts are occurring.

```
Auto-merging practice/temperatures_function.py
```

```
CONFLICT (content): Merge Conflict in  
practice/temperatures_function.py
```

```
Automatic merge failed; fix conflicts and then  
commit the result.
```

Git will clearly mark the conflicting changes in the files and still automatically merge non-conflicting changes.



# Viewing the Merge Conflicts

Different editors give us different tools for handling these merge conflicts, though the principles remain the same.

Open the file(s) listed in the CONFLICT error and scroll to find the conflict. The conflict will be marked by these strange lines:

```
<<<<<<< HEAD
```

```
My name is Joe and I work for NHS Digital.
```

```
=====
```

```
I am a imposter and I work at Imposter Inc
```

```
>>>>>>> feature/JW-conflicting-branch
```

The top bit above the double line is what we have in our branch, represented by the <<<<<<< HEAD statement.

The bottom bit is the same line but saved in the branch we are merging from, represented by the >>>>>>> feature/JW-conflicting-branch statement.

# Resolving Merge Conflicts

To resolve the merge conflict, delete the line which you do not wish to keep, as well as anything to do with marking the conflict. So, if we wanted to keep only what we had on the source branch, we would delete as so:

~~<<<<<< HEAD~~

My name is Joe and I work for NHS Digital.

~~=====~~

~~I am a imposter and I work at Imposter Inc~~

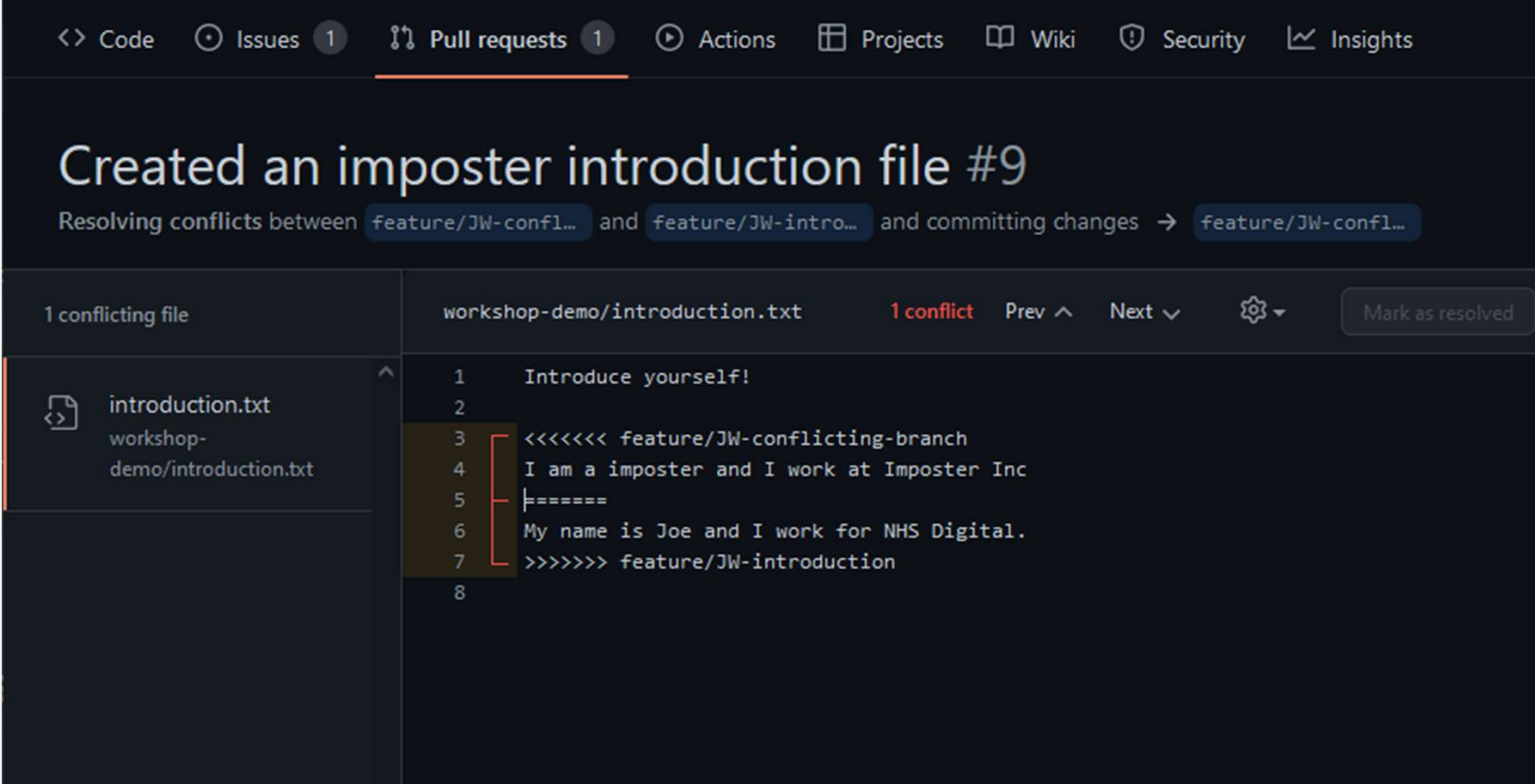
~~>>>>>> feature/JW-conflicting-branch~~

Then once all conflicts are resolved we can stage and commit our merge.

If we wanted to cancel the merge, we can do so simply with the following command:

`git merge --abort`

# Resolving Merge Conflicts - GitHub



The screenshot shows a GitHub pull request interface. At the top, navigation tabs include Code, Issues (1), Pull requests (1), Actions, Projects, Wiki, Security, and Insights. The main heading is "Created an imposter introduction file #9". Below this, it states "Resolving conflicts between feature/JW-confli... and feature/JW-intro... and committing changes → feature/JW-confli...".

On the left, a sidebar indicates "1 conflicting file" and lists "introduction.txt" with the path "workshop-demo/introduction.txt".

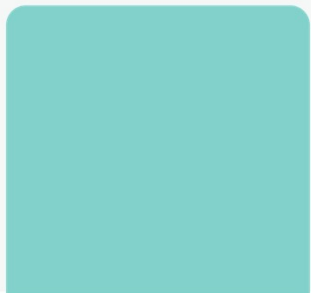
The main area displays the file "workshop-demo/introduction.txt" with a "1 conflict" status. It shows line numbers 1 through 8. Lines 3 through 7 are highlighted with a red bracket, indicating the conflict. The content of these lines is:

```
1   Introduce yourself!
2
3   <<<<<<< feature/JW-conflicting-branch
4   I am a imposter and I work at Imposter Inc
5   =====
6   My name is Joe and I work for NHS Digital.
7   >>>>>>> feature/JW-introduction
8
```

Navigation controls include "Prev ^", "Next v", a settings gear icon, and a "Mark as resolved" button.



# Let's try it out





**Let's have a  
short break**



# **Welcome back!**



# General Guidelines



The main branch of your repository should be kept in a good stable state.



Developers should create feature branches from the main branch and work on them.



Feature branches should be short lived. This prevents challenges when merging later.



Pull requests should be used and reviews completed by your collaborators when merging back into the main branch.



Once a Pull Request is approved and merged, the feature branch should be trimmed and deleted.



# Exploring GitHub and the Open-Source Community

Hopefully, you now have some knowledge and curiosity to start your own Git Collaboration journey.

As well as creating your own repositories, you can also explore other repositories published publicly:

- You can explore the code bases, cloning the repository to your local system (although you probably won't have permissions to contribute)
- Raise issue if you spot any problems
- Fork the repository so you have your own copy to alter and use.
- Use the code as a package in another project.

Check out these interesting repositories:

[nhs-r-community](#) / [git training](#)

[NHSDigital](#) / [rap-community-of-practice](#)

[nhs-r-community](#) / [NHSRdatasets](#)

[NHSDigital](#) / [Smoking-Drinking-and-Drug-Use-Report-Code](#)

[NHSDigital](#) / [nhsuk-react-components](#)

# Useful Resources

[NHS England RAP CoP](#)

[GitHub Cheat Sheet](#)

[Git Docs](#)

[VS Code Source Control Documentation](#)

[How to use Git with R and RStudio](#)

[Atlassian Git Tutorials](#) – Bitbucket orientated

*Disclaimer: NHS England is not affiliated or responsible for any external websites or companies*



*If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything.*

# Contact us

## NHS England RAP Squad



[datascience@nhs.net](mailto:datascience@nhs.net)



[NHS-R Slack Channel](#)



[Gov Data Science  
Slack Channel](#)

**Helen Richardson**  
[github.com/helrich](https://github.com/helrich)



**Joe Wilson**  
[github.com/josephwilson8-nhs](https://github.com/josephwilson8-nhs)



---

## Thank You



**@nhsengland**



**company/nhsengland**



**england.nhs.uk**