

# Introduction to Git

**Eric Jiang (@lorderikir)**

Hi, I'm Eric Jiang 

- Currently Software Engineer, Digital Transformation, eSolutions.
- I founded MonPlan
- I also maintain other apps like GeckoDM and MARIE.js
- Also have worked at Localz too where I worked on the React Native Core App

A big thanks to our swag/  
resource sponsor -  
**GitHub**

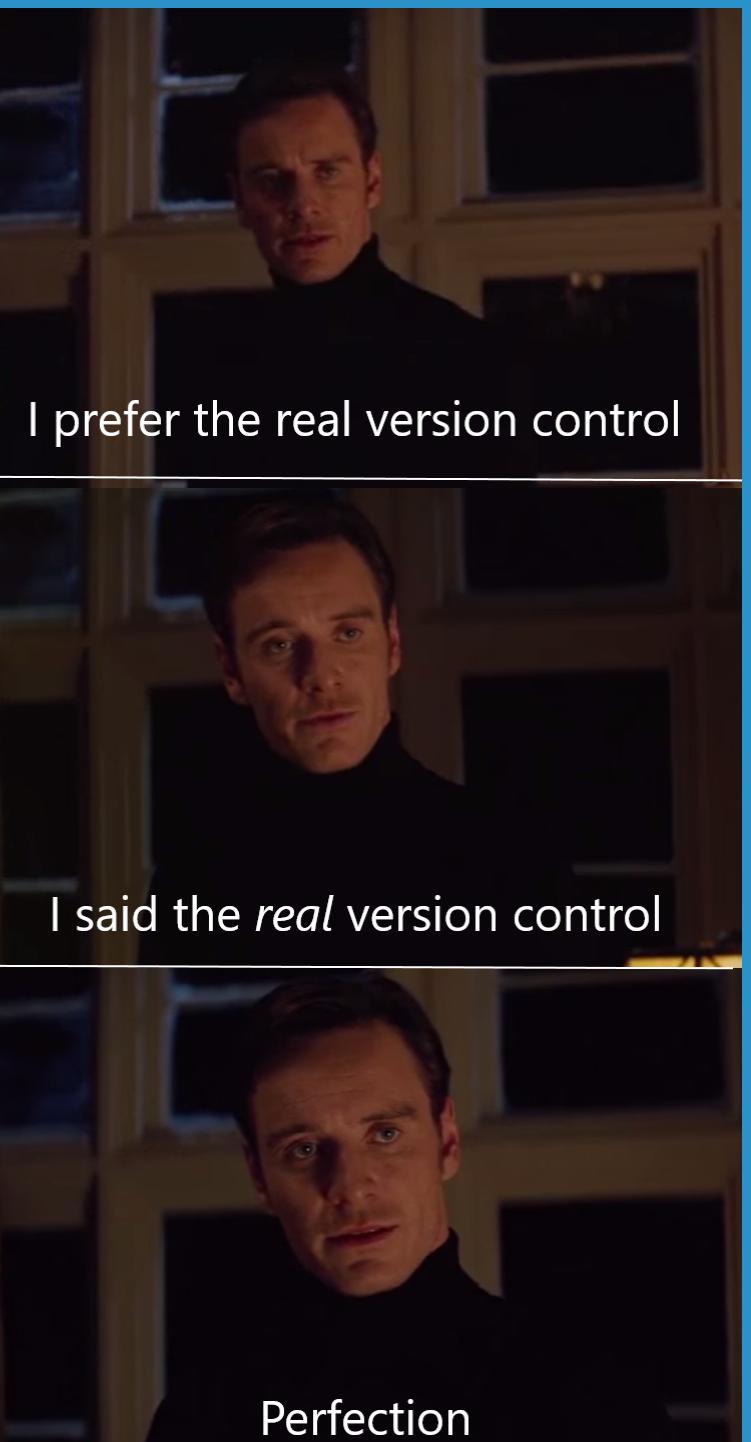
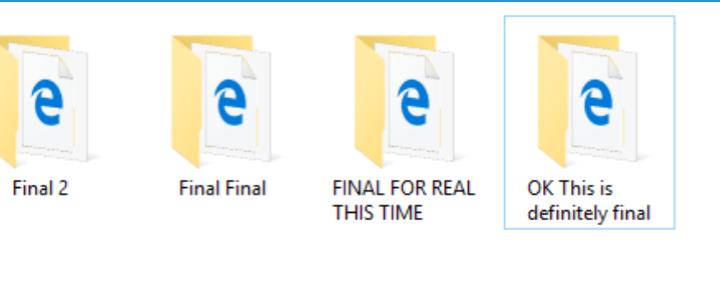
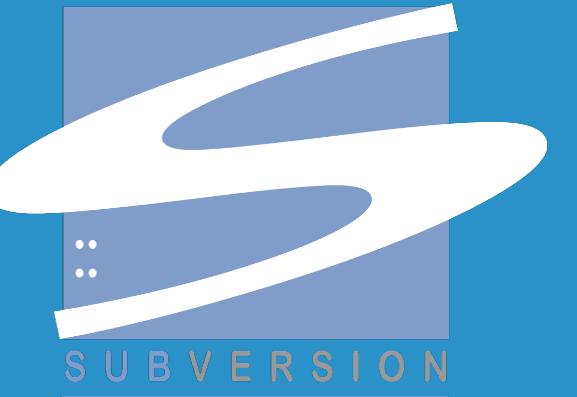
# So what is today about?

# So, I love writing code & also love working in teams

*But what if there was a way that I could remember how the code looked like throughout its entire development lifecycle, for example if something went wrong and I want to go back to a previous version?*

## In comes Git

# First of all, what is Git?



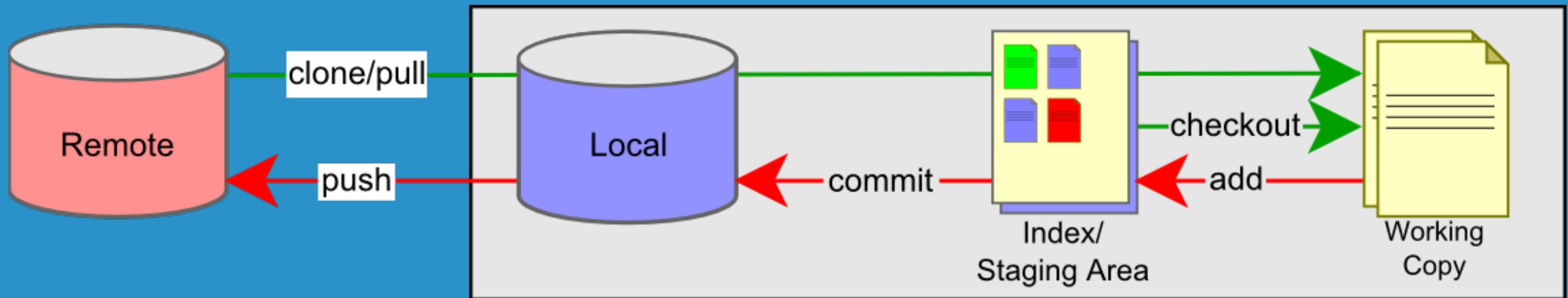
# Git

**Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people**

- **Git-SCM Website**

*Also if you want to go into Software Development at some companies, you are bound to use Git or some kind of version control like SVN/Mercurial*

# How Git Works



# Some Terminology

## Repository

**The Git repository is stored in the same directory as the project itself, in a subdirectory called .git. Note differences from central-repository systems like CVS or Subversion:**

- There is only one .git directory, in the root directory of the project.
- The repository is stored in files alongside the project. There is no central server repository.

# Some Terminology

## Commit

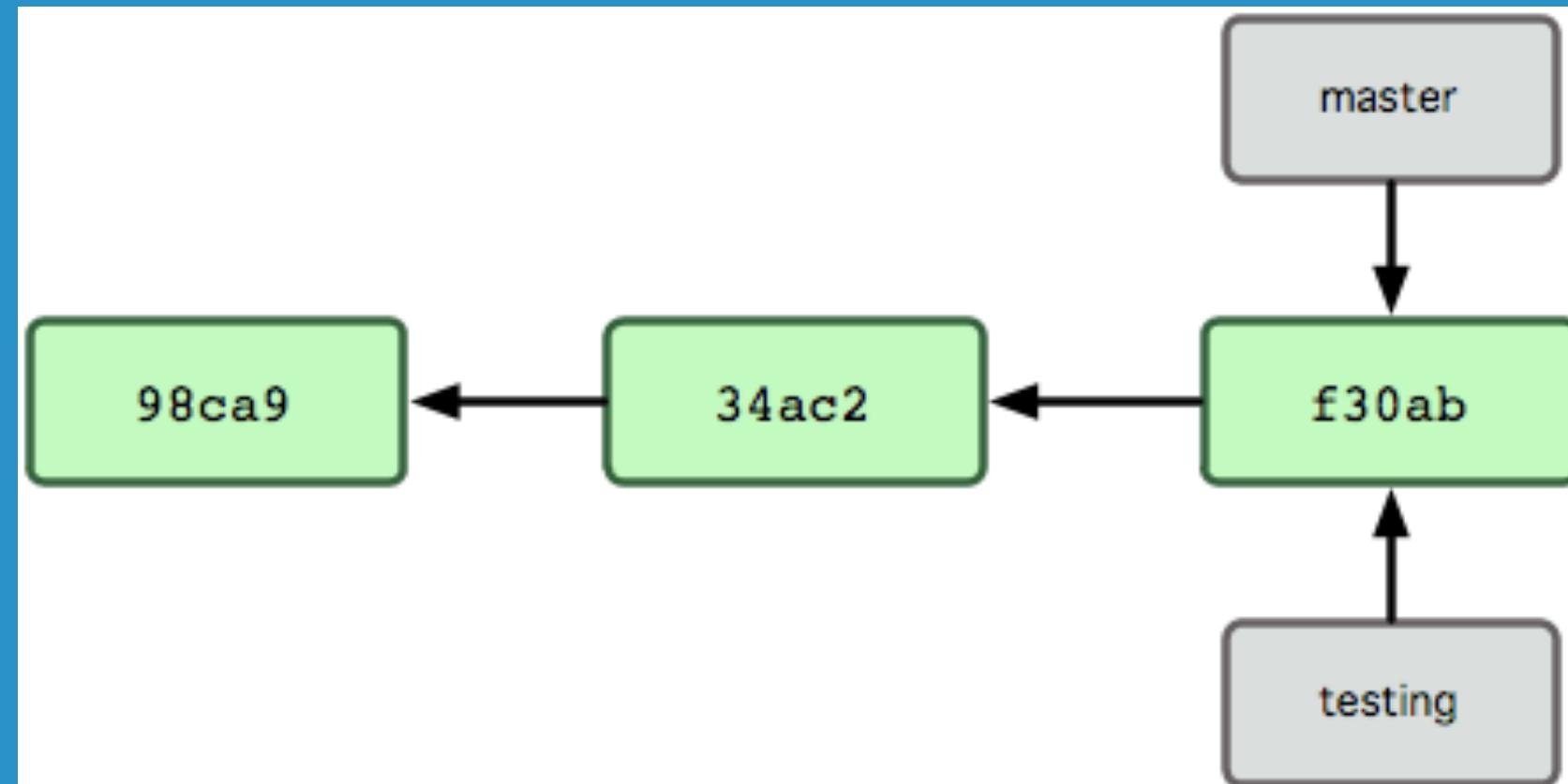
**The git commit command captures a snapshot of the project's currently staged changes**

- A reference to some changes to a file (removals or deletions in files, creation/moving/deletition of files)
- Imagine the commit as a save of the changes to the file(s)
- A commit also contains the Commit Title/Name, and Commit Description

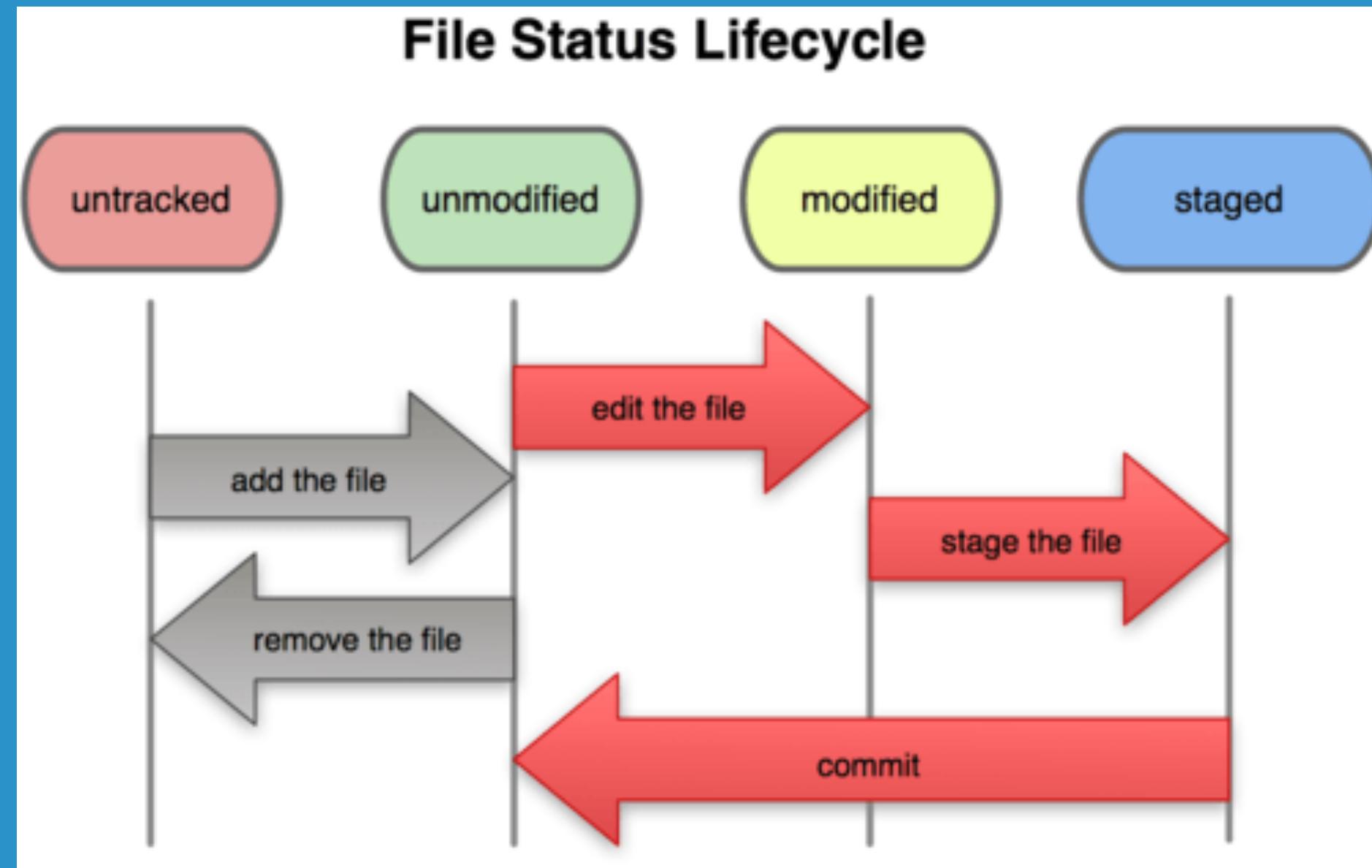
# Some Terminology

## Branches

A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master



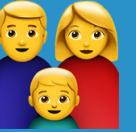
# Git File Lifecycle



# Some Basic Commands

Command	Description
git clone	<b>Clones a repository locally</b>
git add	<b>Stages changes to file(s) for a commit</b>
git commit	<b>Creates a commit (set of changes)</b>
git push	<b>Push changes to the hosted repo</b>

# Using Git within teams

Well, working with teams  may be hard. There are generally two ways you can work off a repository.

- Using Branches
- Using Forks
- Open Source projects tend to use Forks, while:
- a lot of companies internally uses what is known as GitFlow which uses branches!

# Use Branches 🌳 for Versioning Control (GitFlow)

- 1. Make a branch with the feature, bug, hotfix you are working on.**
- 2. Every time you commit and push up**
- 3. Make a Pull Request**
- 4. Merge the pull request**

**One of the best workflows is known as *GitFlow***

# In GitFlow, our Branches follow some naming conventions...

- master: branch is the key branch, typically for our production/public-facing version
- develop: *unstable*, most of the PRs should go here
- staging: this branch is occassionally but not always used, this matches our QA/Testing environment
- feat/\*, fix/\*, etc.: are 'for purpose' branches, these branches are for development

*This slide has been adapted from my CI-CD talk*

So we know that  
development is done  
incrementally

# Imagine we using Git within our practices

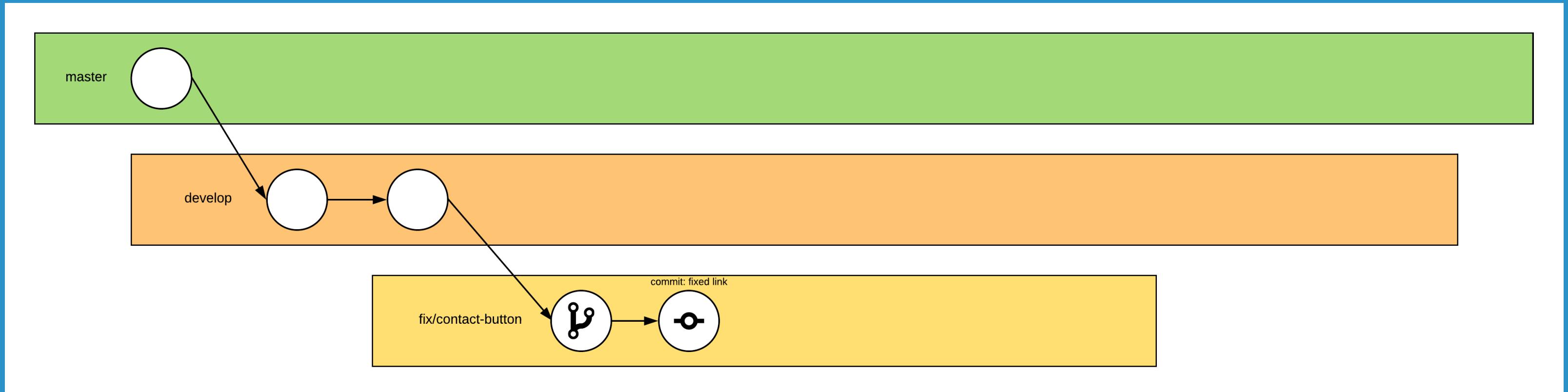
**And one of my team-mates, has found a bug within  
one of our buttons.**

# So, he creates a new branch to fix the bug



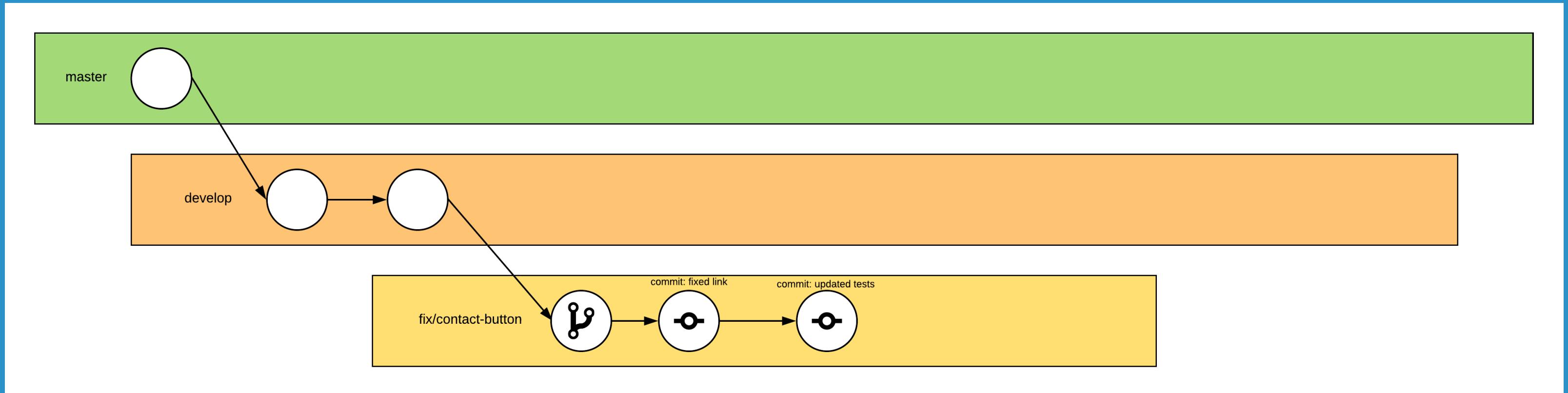
```
# update our develop branch  
git checkout develop  
git pull  
# we create a new branch  
git branch fix/contact-button  
# we make the new branch the new working branch  
git checkout fix/contact-button
```

# He fixes the code and stages the change in commits



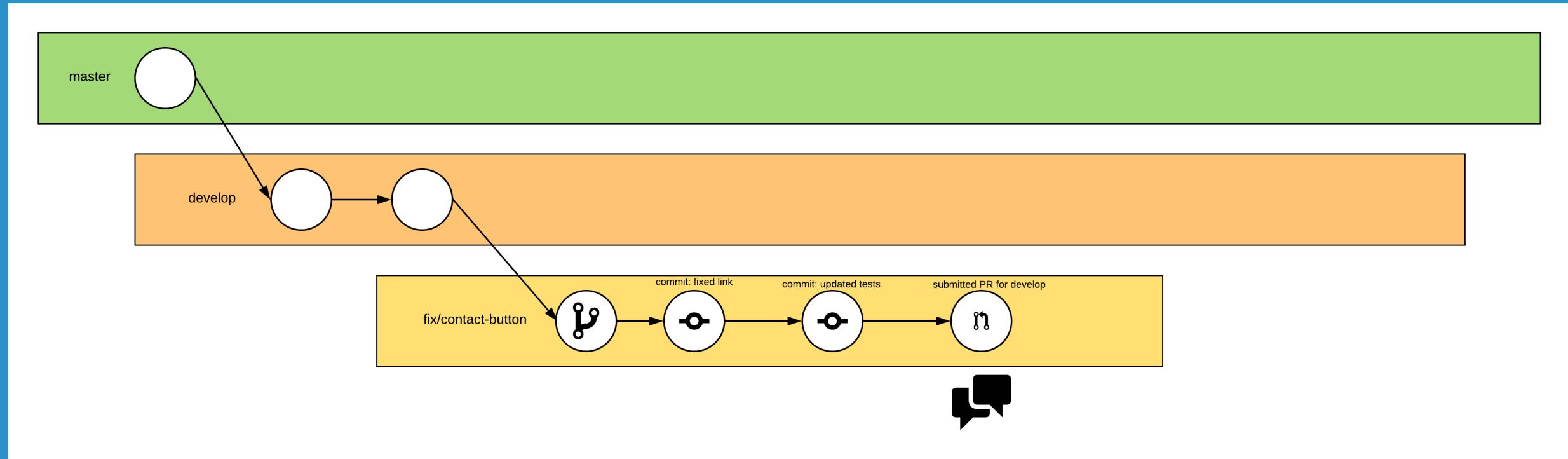
```
git add .
git commit -m "new commit"
git push
```

# He fixes the code and stages the change in commits



```
git add .
git commit -m "new commit"
git push
```

# He then makes a PR into my develop or master branch



Where we discuss his proposed changes

A screenshot of a Bitbucket pull request page. The URL is https://bitbucket.org/monash-university/monplan/pull-requests/298/feature-notification-announcements/diff. The page shows a merged pull request #298 titled "Feature/notification announcements" by Eric Jiang at 63c7419. The target branch is develop. The pull request has 1 approval and 1 review. The description includes a list of changes: "I F\*\*KING FIXED IT", "feat: frontend rendering Announcements", "chore: eslint cleanup", "feat: initial admin component", "added material-ui time picker", "feat: admin page for adding announcements", and "claenup". There is one comment from Saurabh Joshi suggesting to add cookies and create a table for announcement viewing. The file changes section lists 18 files with their respective additions, deletions, and types (M for modified, A for added). The sidebar on the left shows navigation links for Source, Commits, Branches, Pull requests (selected), Pipelines, Deployments, Downloads, Graphs, Boards, and Settings.

Monash University / MonPlan / monplan / Pull requests

## Feature/notification announcements

#298 MERGED at 63c7419 → feature/notification-an... ➔ develop

Revert Approve 1

Overview Commits Activity

Author Eric Jiang

Reviewers SJ

Description

- I F\*\*KING FIXED IT
- feat: frontend rendering Announcements
- chore: eslint cleanup
- feat: initial admin component
- added material-ui time picker
- feat: admin page for adding announcements
- claenup

Comments (1)

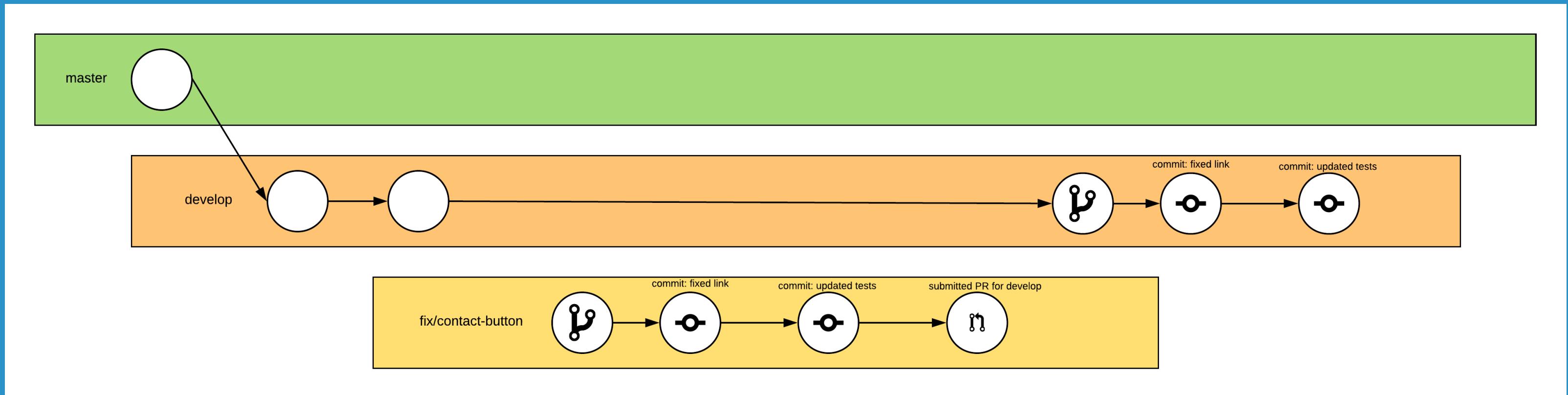
SJ Saurabh Joshi  
Can you add cookies so that the announcements does not show again? Also create a table where it remembers that each user has viewed a particular announcement.  
Reply • Like • Delete • Create task • 2019-04-11

What would you like to say?

Files changed (18)

Change Type	File Path
+25 -2 M	backend/src/main/java/edu/monash/monplan/controllers/NotificationsController.java
+5 -0 M	backend/src/main/java/edu/monash/monplan/models/request/NotificationRequest.java
+5 -0 M	backend/src/main/java/edu/monash/monplan/service/NotificationService.java
+2 -0 M	frontend/package.json
+61 -0 A	frontend/src/components/Base/Announcement/Announcement.js
+30 -0 A	frontend/src/components/Base/Announcement/styles.js
+68 -1 M	frontend/src/components/Base/Main.js
+1 -0 M	frontend/src/constants/limits.js
+154 -0 A	frontend/src/containers/AdminDashboard/NotificationCreationContainer/NotificationCreationContainer.js
+2 -0 A	frontend/src/containers/AdminDashboard/NotificationCreationContainer/index.js
+6 -0 A	frontend/src/containers/AdminDashboard/NotificationCreationContainer/strings.js
+8 -1 M	frontend/src/index.js

# We then merge the Changes



# This would also work for...

- Upgrades to the codebase
- Refactoring our legacy code
- Upgrading frameworks to newer versions

# Why is using GitFlow important?

- We separate production code and our 'work-in-progress' (WIP) code.
- We have a clearer understanding of what each developer is working on
- We can branch off WIP branches and merge changes in
- Relatively easier (not always) to fix merge conflicts
- Some CI/CD tools only run off branches (not PRs)
- We can set our CI/CD to deployment so that it can deploy off branches (i.e. develop to *dev*, master to *staging* or *qa* and deploy to *prod*)

# Key notes



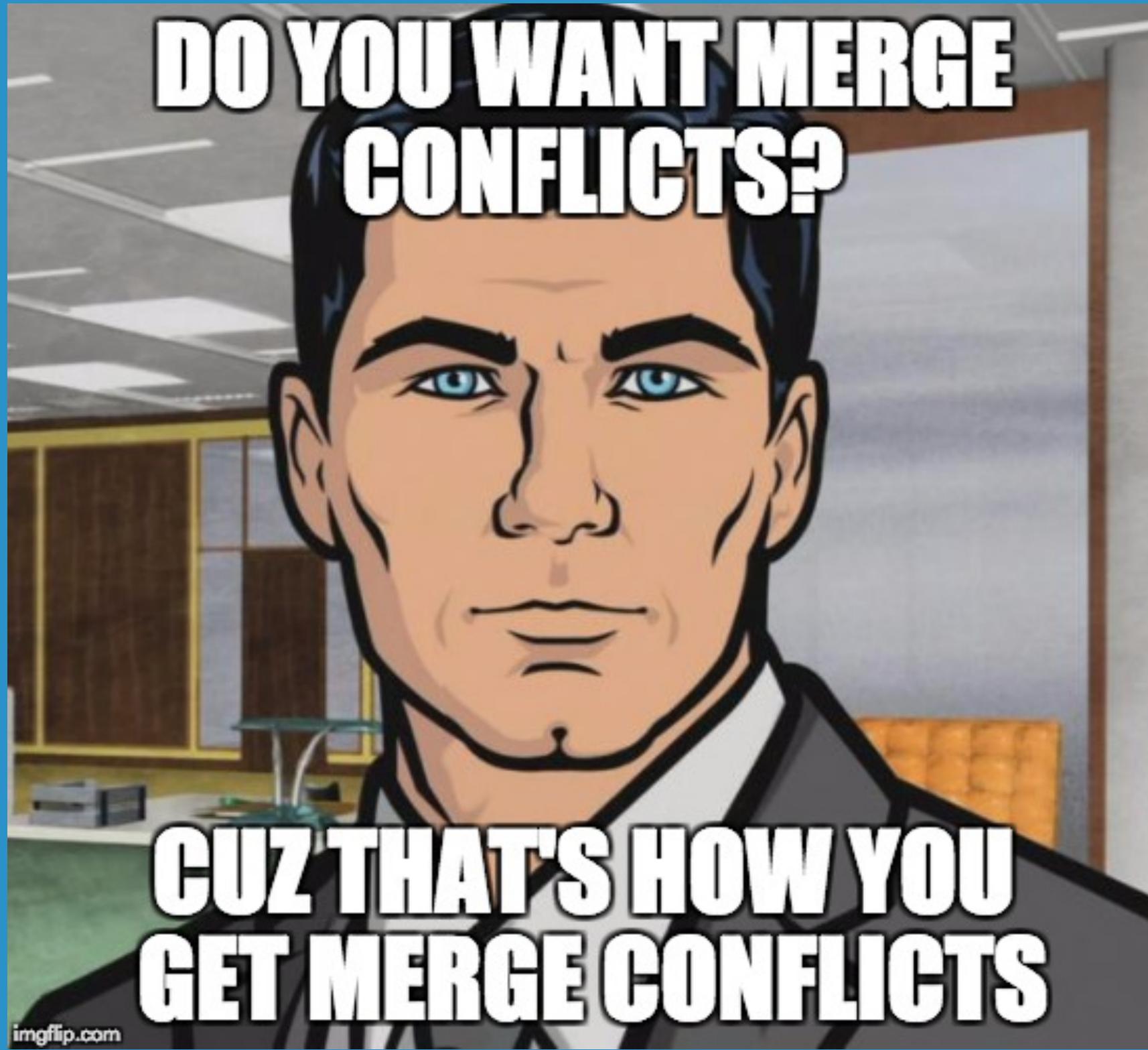
- Version Control over Development is really important as it helps keep 'backups' and you can see the changes
  - You can always see who pushed out the broken code with `git blame` 😈
- Git is always useful as you can always revert broken code or changes
- Branching and forking is basically the same,
  - when working we typically use branches over forks as we can solve merge conflicts more easily (and locally)

# Please DO NOT ever git push --force



# Key things to look out for.

- Merge conflicts are always the hardest part
- Be careful of git merge and git rebase commands.
  - This is because rebase always applies your changes last (assumes you are always correct)
  - When merging between branches and fixing conflicts always work with a team-mate



imgflip.com

Got it? ^\\_(ツ)\_/^-

Let's do some interactive exercises

1. Go to <https://github.com> to create an account if you don't have one (use your student email address as the primary email)
2. Go to <https://lab.github.com> and sign up to GitHub Learning Lab
3. We're going to do the *Introduction to GitHub* course

The screenshot shows a web browser window with the title bar "Introduction to GitHub | GitHub". The address bar displays the URL <https://lab.github.com/githubtraining/introduction-to-github>. The browser's toolbar includes various icons for file operations, search, and navigation. Below the toolbar, the GitHub interface features a "Learning Lab" logo, "For Organizations", and "Docs" links. A user profile icon is visible in the top right corner.

The main content area is titled "githubtraining / Introduction to GitHub". It describes the course as "Your sidekick for getting started on GitHub" and lists categories "Git" and "GitHub". A call-to-action box titled "Join this course" explains that it's a quick introduction to GitHub, and contains a green "Join this course" button.

A large central box contains text about GitHub's capabilities and the benefits of using GitHub Learning Lab. It highlights that GitHub is used for advanced technologies, community building, and tool integration. It also mentions the course's goal of helping users become all-star developers by managing notifications and merging pull requests.

Below this, a section titled "In this course, you'll learn how to:" provides a bulleted list of skills:

- Communicate in issues
- Manage notifications
- Create branches
- Make commits
- Introduce changes with pull requests

# So merge conflicts, what are they and how do we resolve them?

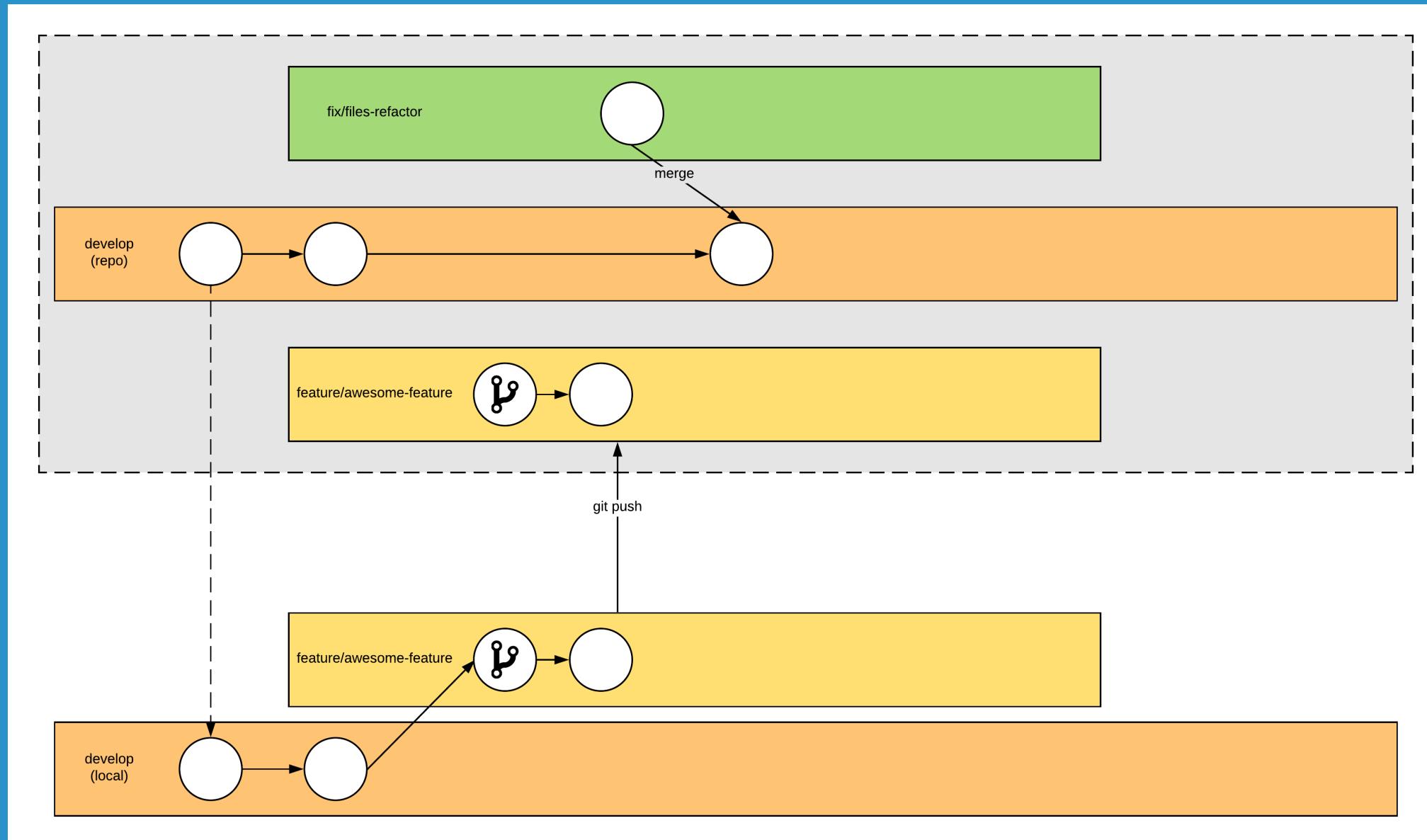
# Merge Conflicts

- essentially, when we are merging changes to code and *Git* sees different changes on the same file
- this most likely happens when someone already committed to the branch from earlier on, and hasn't applied their changes on to the branch that we are using
- therefore, the Git history (of our changes) 'diverges' as each commit is a different hash

# Here's one of the best and easiest ways to resolve a conflict

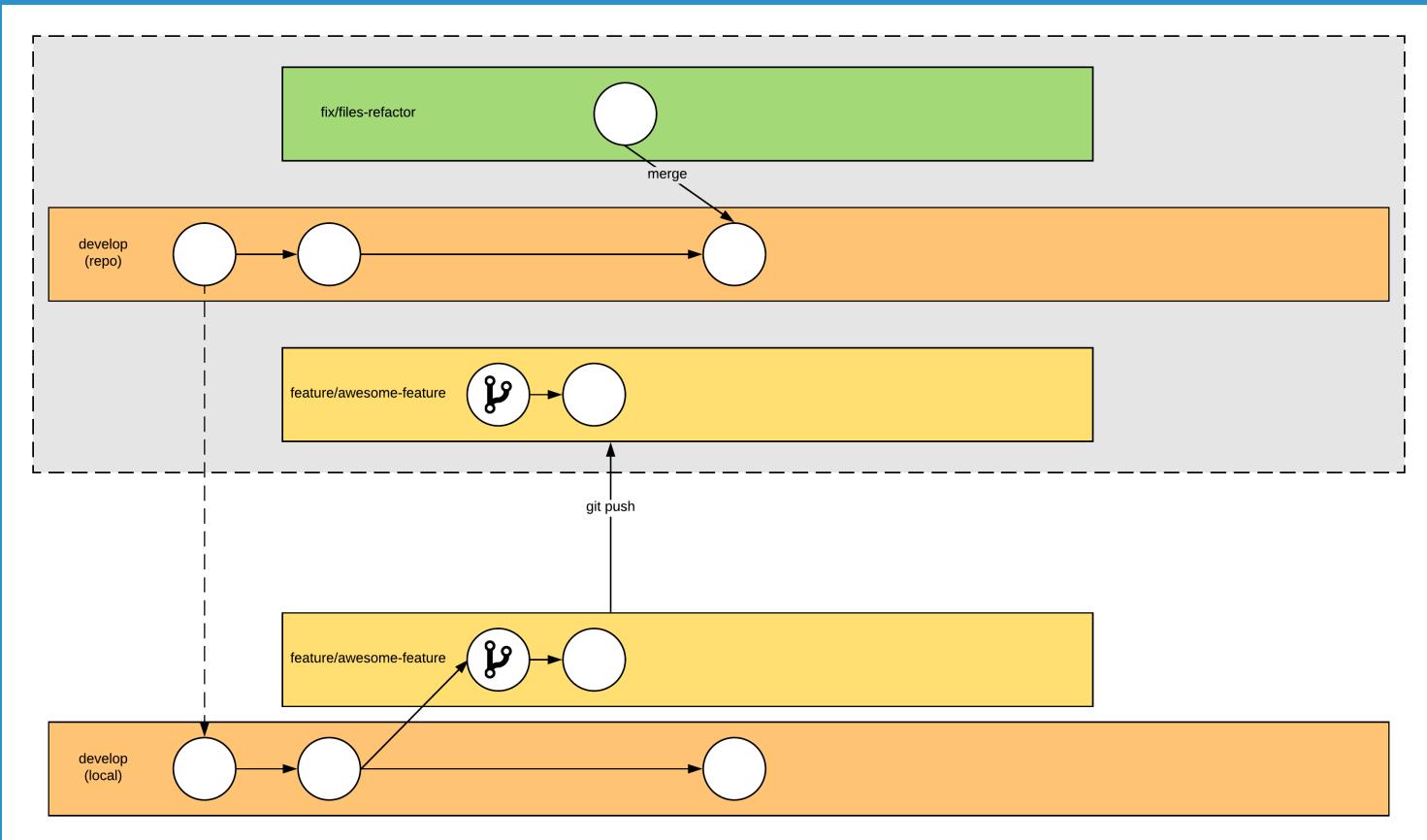
- 1. We go to the target branch and pull down the latest changes**
- 2. We then 'checkout' our current working branch and create a new branch off the working branch**
- 3. We then attempt to merge our target branch into our new branch**
- 4. Resolve Conflicts (by choosing the right pieces of code you want), VSCode makes this really easier**
- 5. Merge the new branch into our current branch**
- 6. Merge the current branch into the target branch**

# This is the current state of the branches



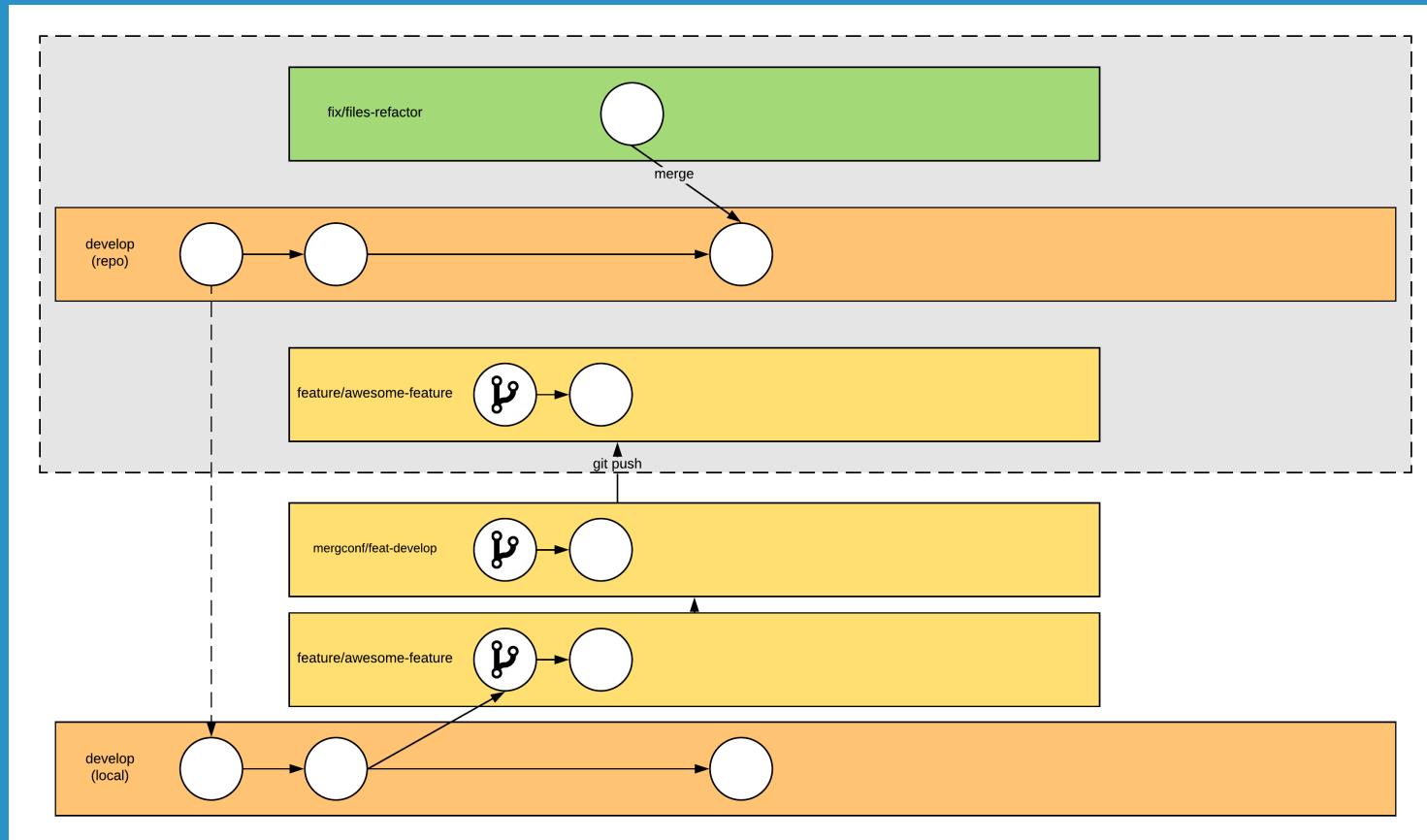
## We go to the target branch and pull down the latest changes

```
git checkout develop  
git pull  
git checkout feature/awesome-feature  
git checkout mergconf/feat-develop  
# ...
```



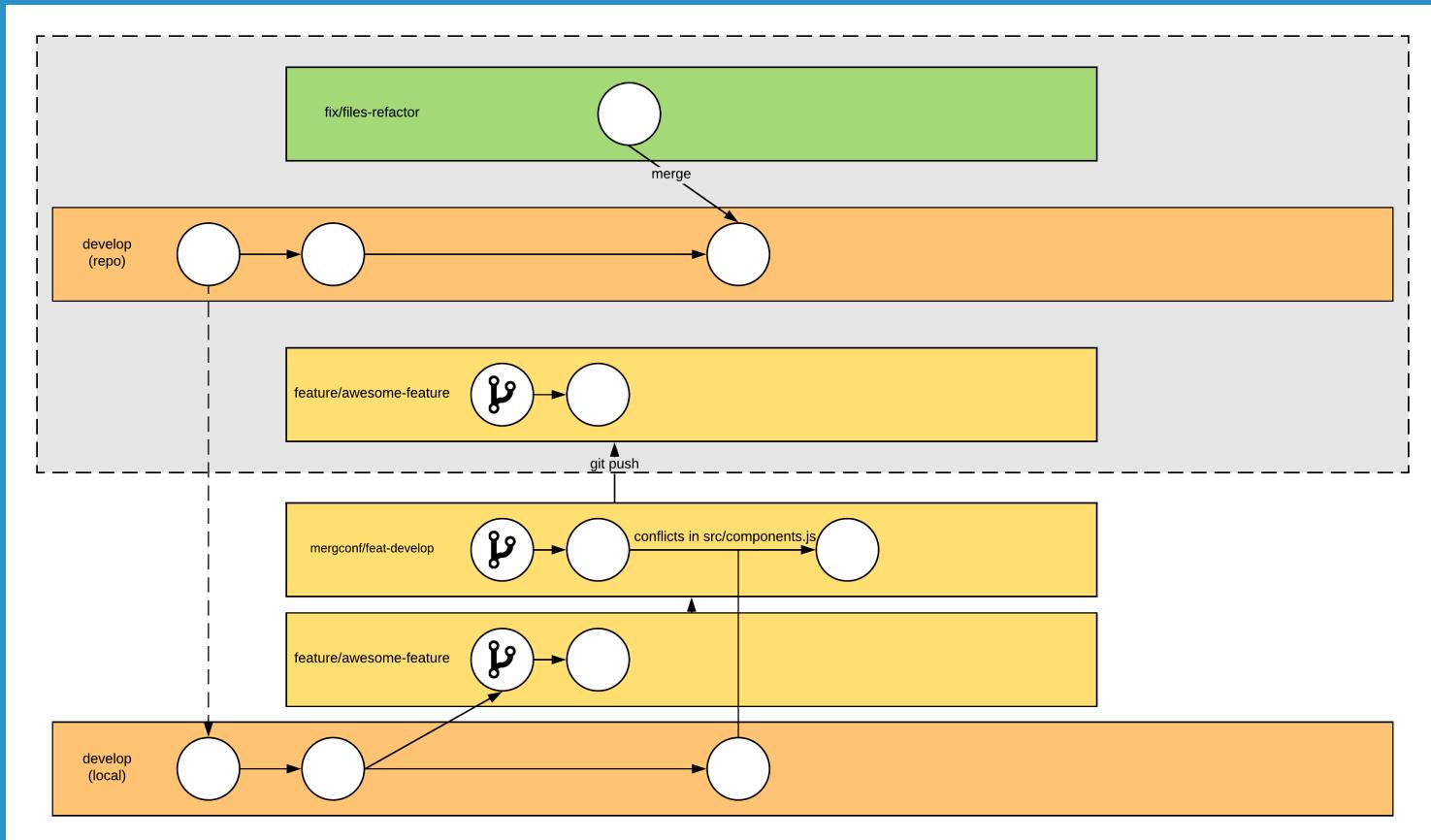
We then 'checkout' our current working branch and create a new branch off the working branch

```
git pull  
git checkout feature/awesome-feature  
git checkout mergconf/feat-develop  
git merge develop  
# ...
```



## We then attempt to merge our target branch into our new branch

```
git pull  
git checkout feature/awesome-feature  
git checkout mergconf/feat-develop  
git merge develop  
# ...
```



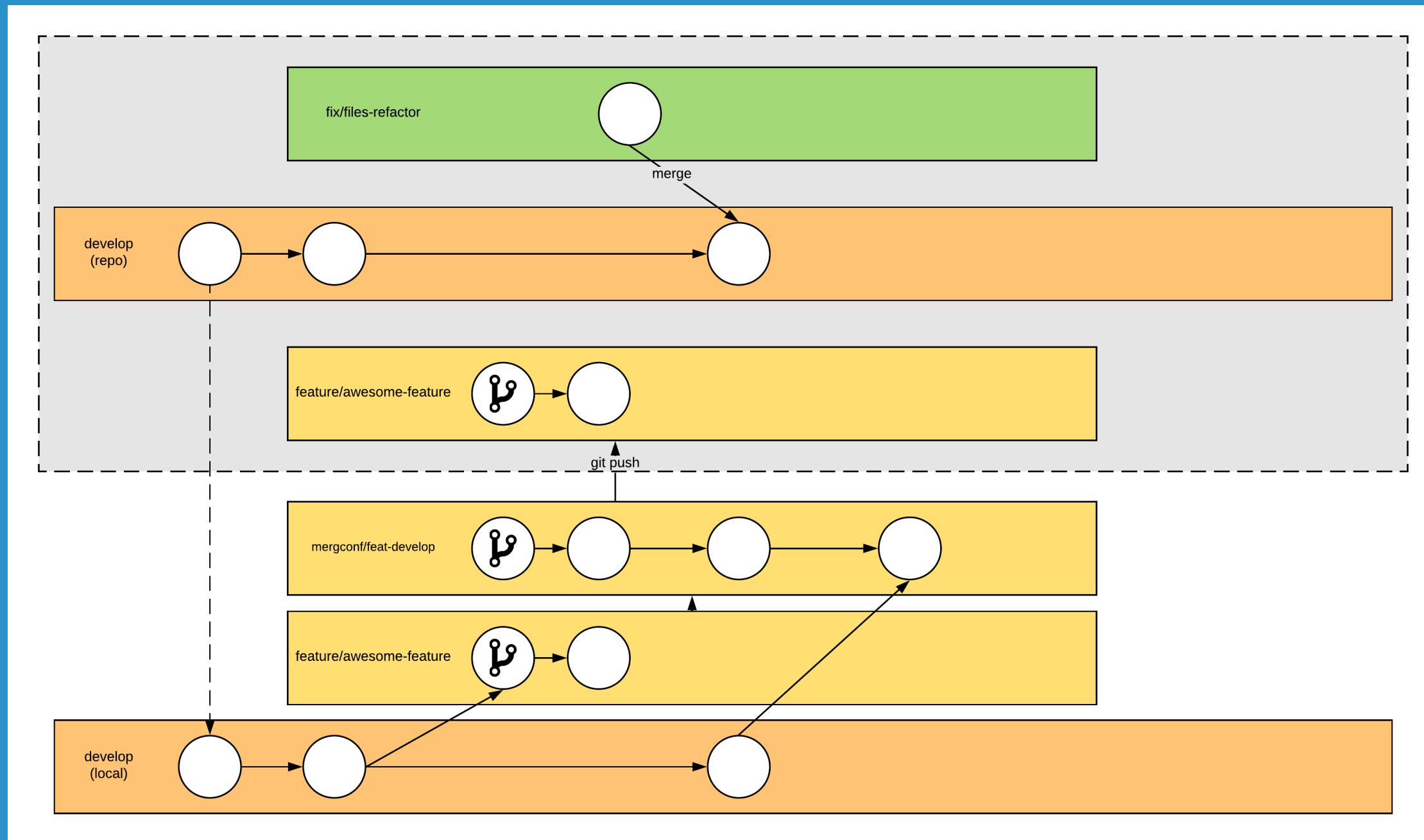
# Resolve Conflicts (by choosing the right pieces of code you want), VSCode makes this really easier

The screenshot shows a conflict in a TypeScript file named `walkThroughPart.ts`. The conflict is between the current local changes (labeled `<<<<< HEAD (Current Change)`) and incoming changes (labeled `>>>>> Test (Incoming Change)`). The code editor displays the following code:

```
406 → → → → → → snippet: i
407 → → → → → → });
408 → → → → → });
409 → → → → });
410 <<<<< HEAD (Current Change)
411 → → → → this.updateSizeClasses();
412 → → → → this.multiCursorModifier();
413 → → → → this.contentDisposables.push(this.configurationService.onDidU
414 =====
415 → → → → this.toggleSizeClasses();
416 >>>>> Test (Incoming Change)
417 → → → → if (input.onReady) {
418 → → → →   input.onReady(innerContent);
419 → → → → }
420 → → → → this.scrollbar.scanDomNode();
421 → → → → this.loadTextEditorViewState(input.getResource());
422 → → → → this.updatedScrollPosition();
423 → → → });
424 → }
```

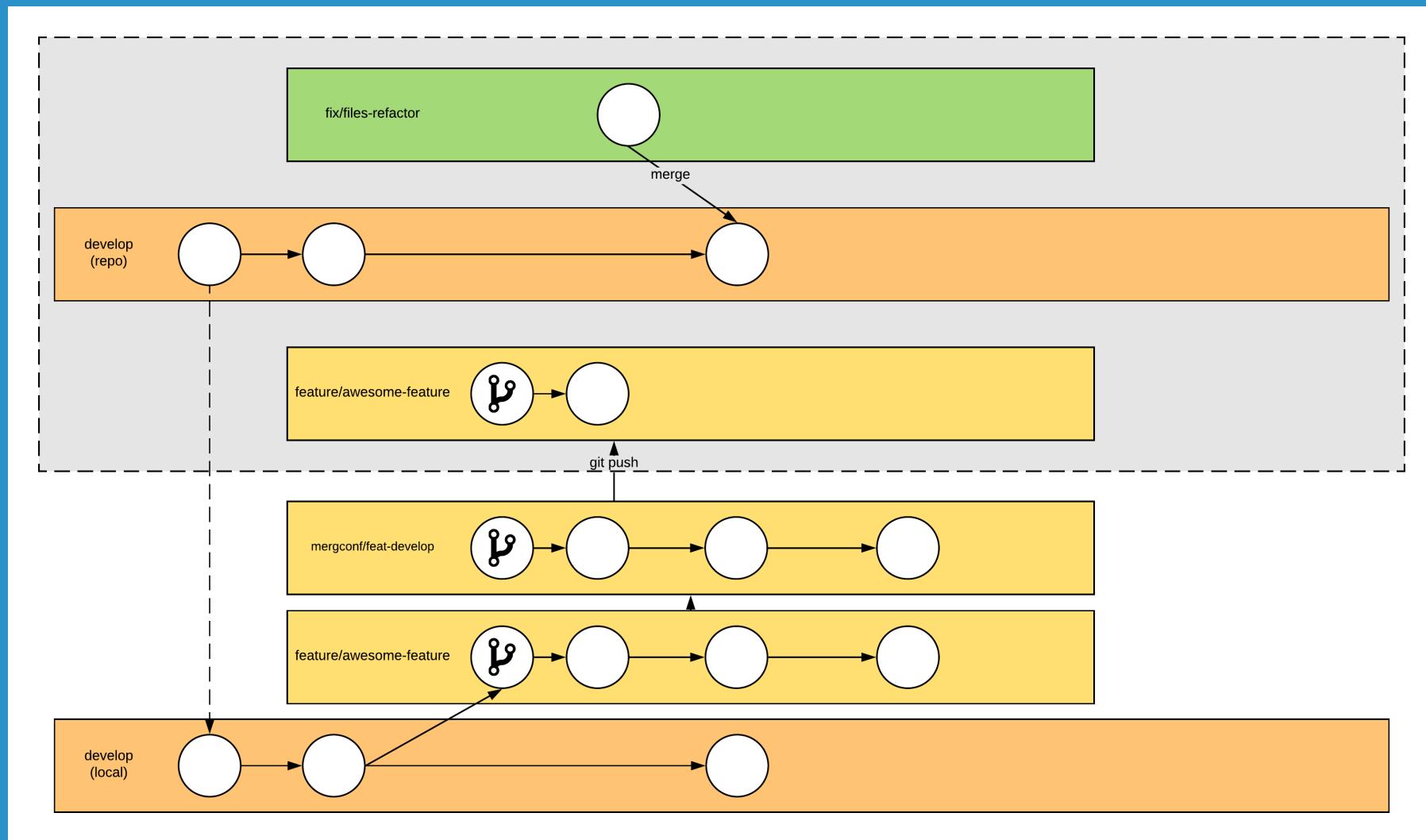
The status bar at the bottom indicates the file is at line 1, column 1, with tab size 4, UTF-8 encoding, TypeScript 2.3.3, and TSLint enabled.

# Here's how the state of the branches are:



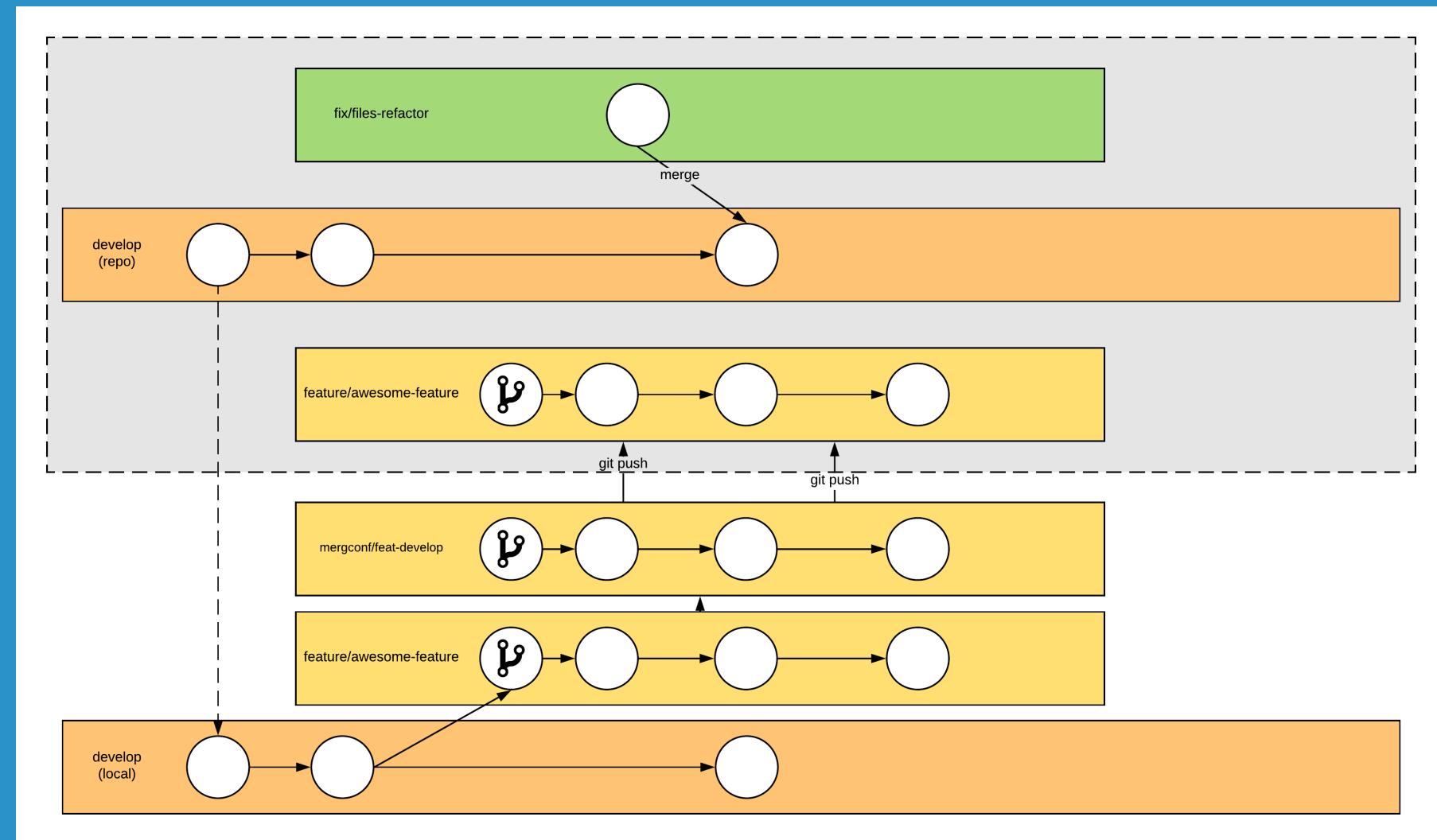
# Merge the new branch into our current branch

```
git checkout feature/awesome-feature #get to current working branch  
git merge mergconf/feat-develop  
git push # push to repo
```



# Push changes to repository

```
git checkout feature/awesome-feature #get to current working branch  
git merge mergconf/feat-develop  
git push # push to repo
```



# Our Conflicts would have been solved now!



# Questions?



# Additional Resources

**Make sure to check out the GitHub Education Pack at <https://education.github.com/pack>, which includes:**

- free GitHub Pro for students
- AWS + credits, free one year algolia,
- free 2 year DataDog,
- one year free domain from namecheap,
- and heaps more!

# Thank You!!

You can follow me on my social accounts:

- Twitter: [@lorderikir](#)
- GitHub: [@lorderikir](#)
- LinkedIn [@ericjiang97](#)