hamedafarag on Sep 16, 2024 Maintainer

Introduction

We are currently facing a challenge due to changes in the product team's strategy for releasing stable features into production.

We have two different teams, each with its own scope, features, and deployment schedules. However, both teams contribute to a single codebase.

Please check these two documents to check the current applied git flow and the implemented pipelines.

Proposed Solutions

- Reusable Components, Helpers, Utilities, or Styles

Reusable components, helpers, utilities, or styles should be implemented through a separate workflow. Developers should handle these separately from the feature implementation itself. The following guidelines should be followed:

- **Implementation**: Develop the reusable component, helper, utility, or style.
- **Unit Testing**: Ensure comprehensive unit testing, especially for utilities or helpers.
- **Documentation**: Provide full documentation by writing an MDX file in the Storybook documentation project.

Branching

- Create a branch from `master` with the prefix `reuse/`.
  > **"reuse"** stands for reusable "stuff".
- Develop and implement your reusable component, helper, utility, or style. Write unit tests and documentation.
- > If you are unsure whether a function, utility, or component will be reusable, implement it within your feature. In the PR, mention that it may be nominated for reuse so a decision can be made.
  > Use unit tests for helpers and utilities.
  > Use Storybook for React components and styles.
- Create a new PR to merge the branch into the main `master` branch.
  > Any developer wants to use this component, they must rebase from the `master` to get this component

    - If there is a bug or an enhancement needed for the reusable stuff, create a branch from `master` and follow the same steps for creating new reusable components, helpers, utilities, or styles.
- **Third-Party Library**
    - For reusable components based on shadcn/ui, create a single Storybook entry and include:
        - The reference link for the component.
        - The version number.
        - Any additional features or enhancements, with full examples.
- **Styles**
    - We are currently using TailwindCSS. Use the default Tailwind class names. If any changes are needed, override the Tailwind setup in **tailwind.config.ts** and/or **ui/src/root.css**.

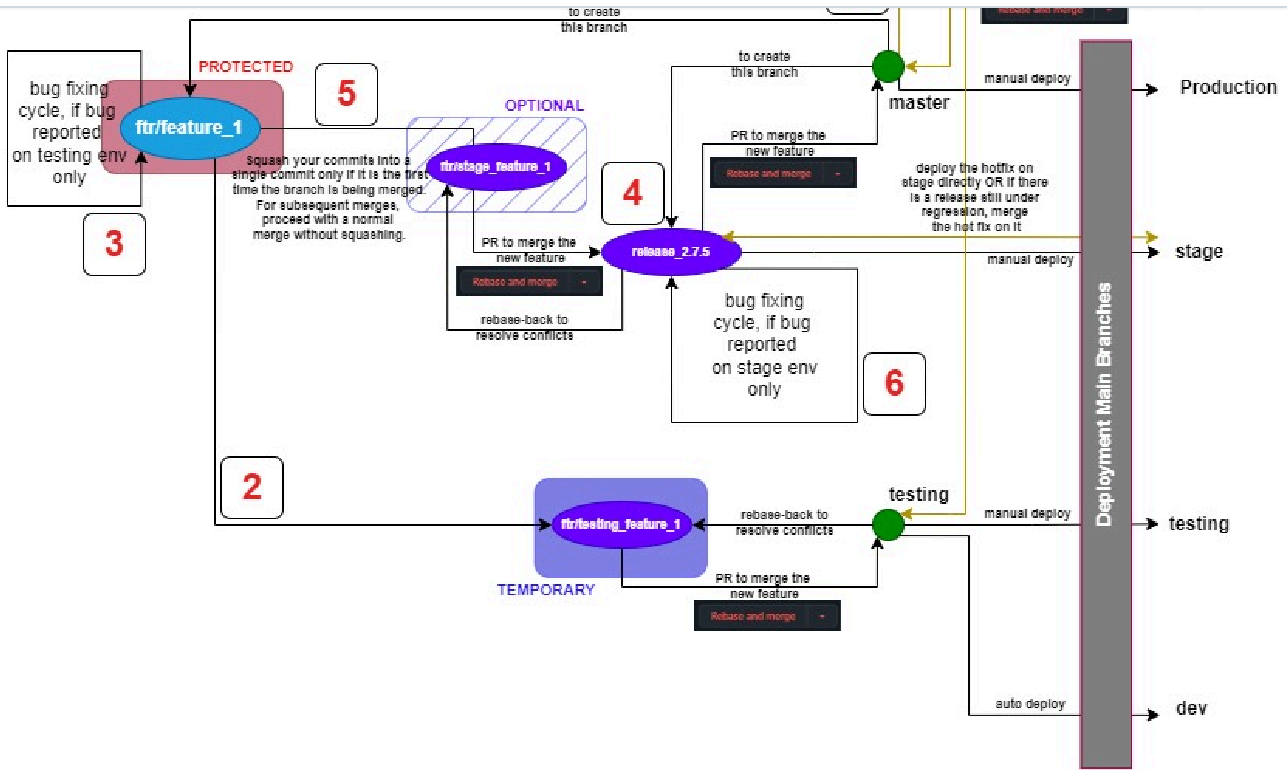
> Previously, we used Bootstrap, but it is now deprecated, and no more Bootstrap components should be used.

Action Items

- Create a pipeline to ensure the following:
    - The branch prefix is correct.
    - The parent branch is `master`.
    - Unit tests cover at least 80% of the code (code coverage).
    - If a PR with the prefix `reuse/` has been merged into `master`, automatically a new deploy will be pushed to GitHub Pages with the new updates.
        > A pipeline will run automatically called "Deploy Storybook Docs to Pages" (*docs.yml*).
- ~~We need to change the `docs` branch to be based from the `master`~~. [DEPRECATED]
    > ~~Change the `docs` branch name to another name [*CASE HAS BEEN REPORTED BY SARAH*]~~
- ~~Create a PR template for the reusable stuff~~ [DONE]
    - ~~**Component template**~~
    - ~~**Helpers/Utils template**~~
    - ~~**Styles template**~~

- Releasing Strategy

Shifting our mindset on managing Git Flow to focus more on releasing stable features can be a bit tricky. Let's now explain the following diagram.



to create this branch
PROTECTED
ftr/feature_1
bug fixing cycle, if bug reported on testing env only
3
5
OPTIONAL
ftr/stage_feature_1
Squash your commits into a single commit only if it is the first time the branch is being merged. For subsequent merges, proceed with a normal merge without squashing.
PR to merge the new feature
Rebase and merge
rebase-back to resolve conflicts
4
release_2.7.5
to create this branch
PR to merge the new feature
Rebase and merge
master
manual deploy
Production
deploy the hotfix on stage directly OR if there is a release still under regression, merge the hot fix on it
manual deploy
stage
bug fixing cycle, if bug reported on stage env only
6
Deployment Main Branches
2
ftr/testing_feature_1
TEMPORARY
rebase-back to resolve conflicts
testing
manual deploy
testing
PR to merge the new feature
Rebase and merge
auto deploy
dev

This Git flow diagram outlines a branching strategy designed to manage feature development, bug fixing, and releases in a way that minimizes conflicts and prevents the release of unstable features. It is particularly suited for N number of teams working in parallel on the same codebase.

Full Cycle as an Example

Mr. Neo has been assigned a new task titled "ZN-001: Hack the Matrix"

1. Create branch `ftr/zn-001_hack_matrix` from the `master` branch.
2. Implement the full feature, write the needed unit tests, write the documentation, etc.
3. Time to integrate with the `testing` branch
    i. create an intermediate branch (temp, if there is any conflict) named `ftr/testing_zn-001_hack_matrix` from `ftr/zn-001_hack_matrix`
    ii. merge back (rebase) `testing` branch into this intermediate branch to resolve any conflicts.
    iii. create a PR to merge `ftr/testing_zn-001_hack_matrix` into `testing` branch
        a. **DON'T FORGET TO SQUASH ALL COMMITS TO BE ON COMMIT (INTERACTIVE REBASE)**

> If there is an exception or something goes wrong, **Mr. Neo** should fix the issue in the main feature branch `ftr/zn-001_hack_matrix`, merge it again into the intermediate branch `ftr/testing_zn-001_hack_matrix` or create a new intermediate branch, resolve any new conflicts with the `testing` branch, and then create a new PR.

> For bug fixing, **Mr. Neo** should address any bugs reported by the quality team in the main feature branch `ftr/zn-001_hack_matrix`. He should then merge this branch again into the intermediate branch `ftr/test_zn-001_hack_matrix` or create a new intermediate branch, resolve any new conflicts with the `testing` branch and create a new PR.

4. Time to release on the stage

    > We have to create a release branch `release/*` for the next the relase from the `master` branch.

    i. Start to create a PR to merge `ftr/zn-001_hack_matrix` into the `release/*`, if there is any conflict you have to create an intermediate branch `ftr/stage_zn-001_hack_matrix`
    ii. Merge the `ftr/zn-001_hack_matrix` or `ftr/stage_zn-001_hack_matrix` into the `release/*` using `**Rebase and merge**` option.

5. Bug fixes for the stage

    i. Any reported bug, we have to create a new branch from the `release/*` to fix the issue.
    ii. Create a new PR to merge (rebase) the fix into the `release/*`.
    iii. Deploy the `release/*` into stage environment.

6. Hot fixes

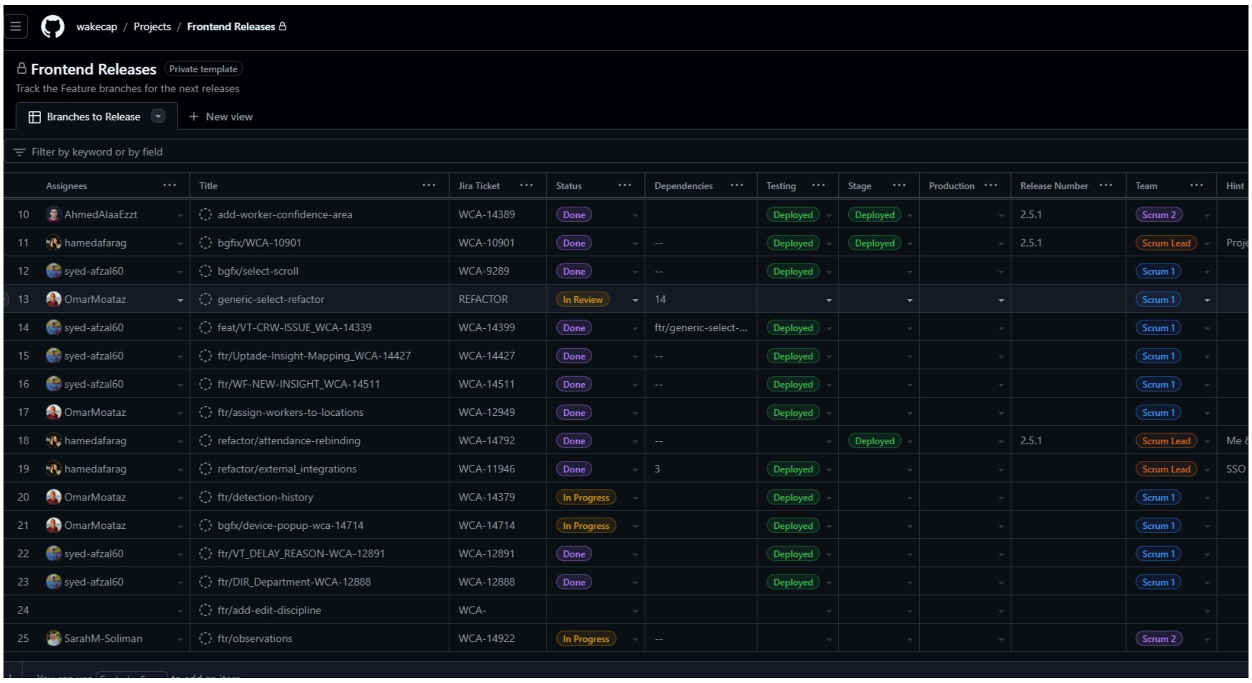
    i. The hacking task threw an exception due to Agent Smith working against it. We need to deploy a quick fix to support the hacking process.
        a. Mr. Neo should create a new branch from `master` named `htfx/zn-002`.
        b. Create a PR to merge this fix into ..., we have to secnarios here
            a. Deploy directly the hot fix directly to stage.
            b. Merge the PR into the `release/*` branch, if we have a release still in the regression phase.
        c. if the hot-fix is working fine, merge the hot-fix into the `master` branch and deploy it to production
        d. Don't forget to merge the hot-fix back into `testing` branch and if there is any conflict, as ususal you have to create an intermediate branch.

7. Feature Dependencies
    When implementing a feature that depends on changes from another branch not yet merged into the `main` branch, follow these steps:

    i. Create an intermediate branch:
        - Name it descriptively (e.g., `feature/intermediate-dependency-integration`)
        - Implement the necessary changes from the dependency branch
    ii. Complete the integration in the intermediate branch:
        - Ensure all required functionality is working correctly
        - Perform thorough testing to verify the integration
    iii. Cherry-pick the relevant commits:
        - Select the specific commits from the intermediate branch
        - Apply them to your original feature branch
    iv. Prioritize deployment order:
        - Ensure the dependency branch is merged and deployed first
        - Only then proceed to merge your feature branch

   - **Master**: The production-ready code.
   - ***Testing**: Used for testing new features for the **Quality Team**.
2. **Feature Branches**:
   - **ftr/feature_1**: A branch for developing a specific feature.
     - We Will keep this branch till we release the feature to the production. (Red Zone)
   - ***ftr/testing_feature_1**: A temporary branch for integrating the feature into the testing branch (Blue Zone).
   - **ftr/stage_feature_1**: A temporary branch for integrating the feature into the stage branch (Blue Zone).
3. **Bug Fixing Cycle**:
   - **ftr/feature_1**: The feature branch where bugs are fixed.
4. **Hot Fixes**: hot fix should be from master (create a flow for the hotfixes)
   - **htfx/hotfix_1**: The branch that created from `master` and contains the hotfix.

> ## Sequential Deployment of Releases
>
> We should implement a policy of deploying releases one at a time. In cases where multiple releases are pending, they will be deployed in sequence rather than concurrently.
>
> ## Developer Responsibility for Feature Branches
>
> Each developer is fully accountable for their feature branch throughout its complete lifecycle. This responsibility extends from the initial development phase through to the final delivery in the production environment.
>
> ## Centralized Release Branch Communication
>
> The team should maintain a centralized list of all branches scheduled for inclusion in the upcoming release.


wakecap / Projects / Frontend Releases
Private template
Branches to Release
New view
Filter by keyword or by field

Frontend Releases

Track the Feature branches for the next releases

	Assignees	Title	Jira Ticket	Status	Dependencies	Testing	Stage	Production	Release Number	Team	Hint
10	AhmedAlaaEzzt	add-worker-confidence-area	WCA-14389	Done		Deployed	Deployed		2.5.1	Scrum 2	
11	hamedafarag	bgfix/WCA-10901	WCA-10901	Done	--	Deployed	Deployed		2.5.1	Scrum Lead	Proje
12	syed-afzal60	bgfx/select-scroll	WCA-9289	Done	--	Deployed				Scrum 1	
13	OmarMoataz	generic-select-refactor	REFACTOR	In Review	14					Scrum 1	
14	syed-afzal60	feat/VT-CRW-ISSUE_WCA-14339	WCA-14399	Done	ftr/generic-select-...	Deployed				Scrum 1	
15	syed-afzal60	ftr/Uptade-Insight-Mapping_WCA-14427	WCA-14427	Done	--	Deployed				Scrum 1	
16	syed-afzal60	ftr/WF-NEW-INSIGHT_WCA-14511	WCA-14511	Done	--	Deployed				Scrum 1	
17	OmarMoataz	ftr/assign-workers-to-locations	WCA-12949	Done		Deployed				Scrum 1	
18	hamedafarag	refactor/attendance-rebinding	WCA-14792	Done	--		Deployed		2.5.1	Scrum Lead	Me &
19	hamedafarag	refactor/external_integrations	WCA-11946	Done	3	Deployed				Scrum Lead	SSO
20	OmarMoataz	ftr/detection-history	WCA-14379	In Progress		Deployed				Scrum 1	
21	OmarMoataz	bgfx/device-popup-wca-14714	WCA-14714	In Progress		Deployed				Scrum 1	
22	syed-afzal60	ftr/VT_DELAY_REASON-WCA-12891	WCA-12891	Done		Deployed				Scrum 1	
23	syed-afzal60	ftr/DIR_Department-WCA-12888	WCA-12888	Done		Deployed				Scrum 1	
24		ftr/add-edit-discipline	WCA-								
25	SarahM-Soliman	ftr/observations	WCA-14922	In Progress	--					Scrum 2	

- `bgfx/` : Prefix for the bug fixes branches.
- `htfx/` : Prefix for the hot fixes branches.
- `release/` : Prefix for the branch that contains the stable feature for deployment into staging and production environments.
- `reuse/` : Prefix for the reusable stuff branches.
- `chore/` : Updating build tasks, package manager configs, etc; no production code change
- `docs/` : Changes to documentation
- `test/` : Adding tests, refactoring test; no production code change

Action Items

- ~~Each feature's author should keep track with the feature branch till we merged it into the master.~~ [DONE]
- No more merge squash, it will be normal merge or rebase.
- Find a good way to rename the feature branches

  > Comments By [Sarah Soliman]
  >
  > - Feature Branches: **ftr/feature_1 =>** so we should write here **ftr/[featureName_JiraTicket]**?
  > - What if two or more developers are working on a story, will each developer add the feature name as a prefix, then their subtask (ex: **ftr/[featureName]_[Subtask]**)? Or should we add another prefix in this case? For example **epic/[epicName]_[JiraTicket]** and then each developer will base from the epic branch **ftr/[epicName]_[Subtask]** or **[epicName]/[Subtask]**.
  >
  > Comment By [Omar]
  >
  > - Frr/zn-001-hack-matrix should be called ftr/hack-matrix-zn-001 because we need branch name auto complete to kick in after typing hack and then clicking tab. Useful for people who use the command line to do their git work.

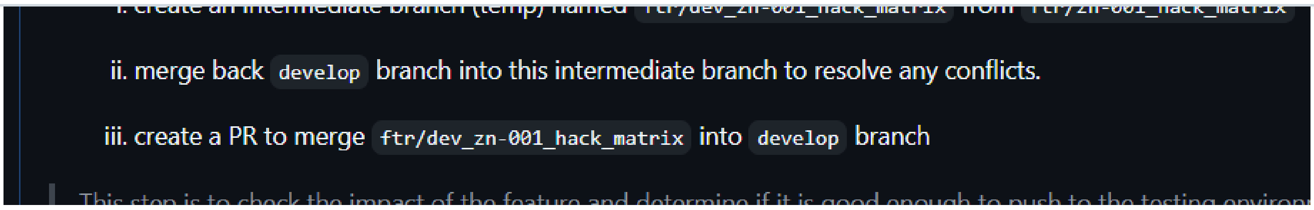
- ~~Auto Deploy the develop branch after any merging.~~ [DONE]
- Extract rules from this flow and start to create scripts ans husky checks
- Prepare GitHub Actions to validate
  - Branch name.
  - Branch's Parent name.
  - Create Tag automatically after merging the release into the `master`

↑ 1

8 comments · 4 replies

Oldest Newest Top

hamedafarag on Sep 16, 2024 Maintainer Author


i. create an intermediate branch (temp) named `ftr/dev_zn-001_hack_matrix` from `ftr/zn-001_hack_matrix`

ii. merge back `develop` branch into this intermediate branch to resolve any conflicts.

iii. create a PR to merge `ftr/dev_zn-001_hack_matrix` into `develop` branch

This step is to check the impact of the feature and determine if it is good enough to push to the testing environ

For the first point, creating an intermediate branch is optional. You should only create one if conflicts occur in your Pull Request (PR).

1

0 replies

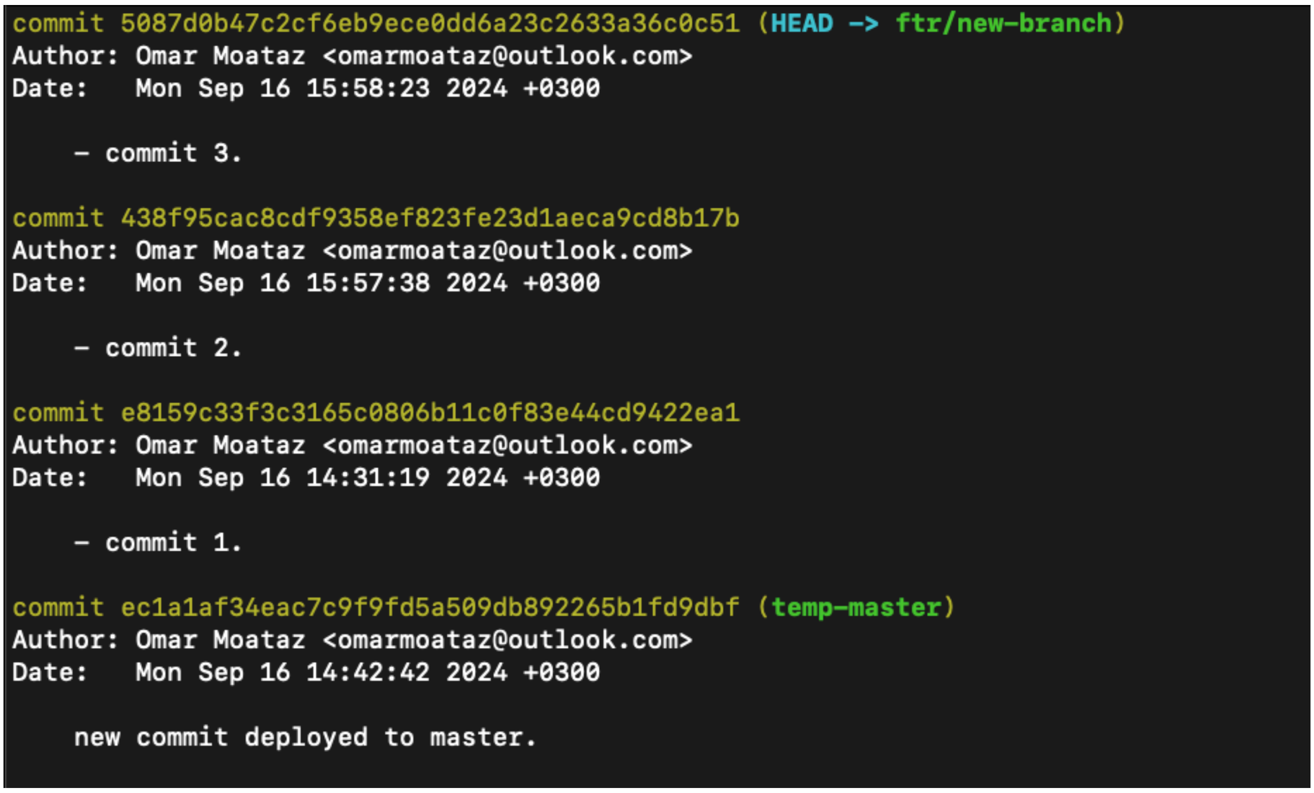
Write a reply

OmarMoataz on Sep 17, 2024 Collaborator

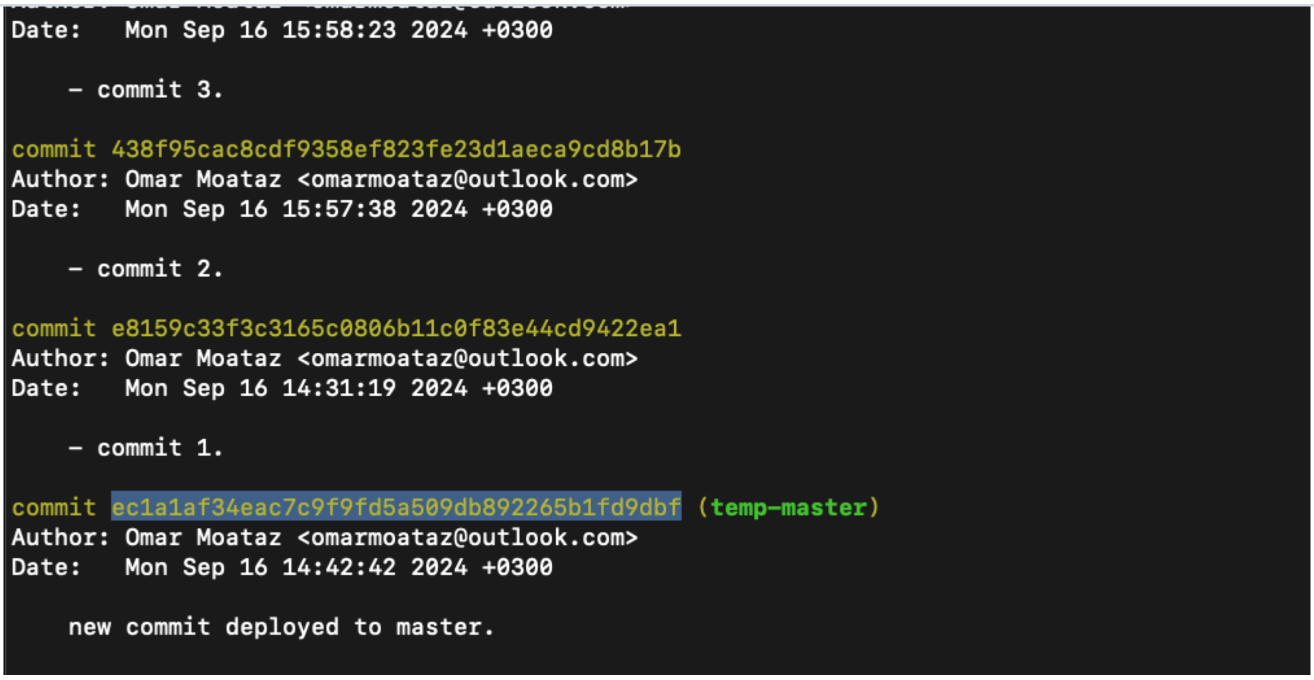
edited

3. Use interactive rebasing to create a single commit PR as follows:

git log:

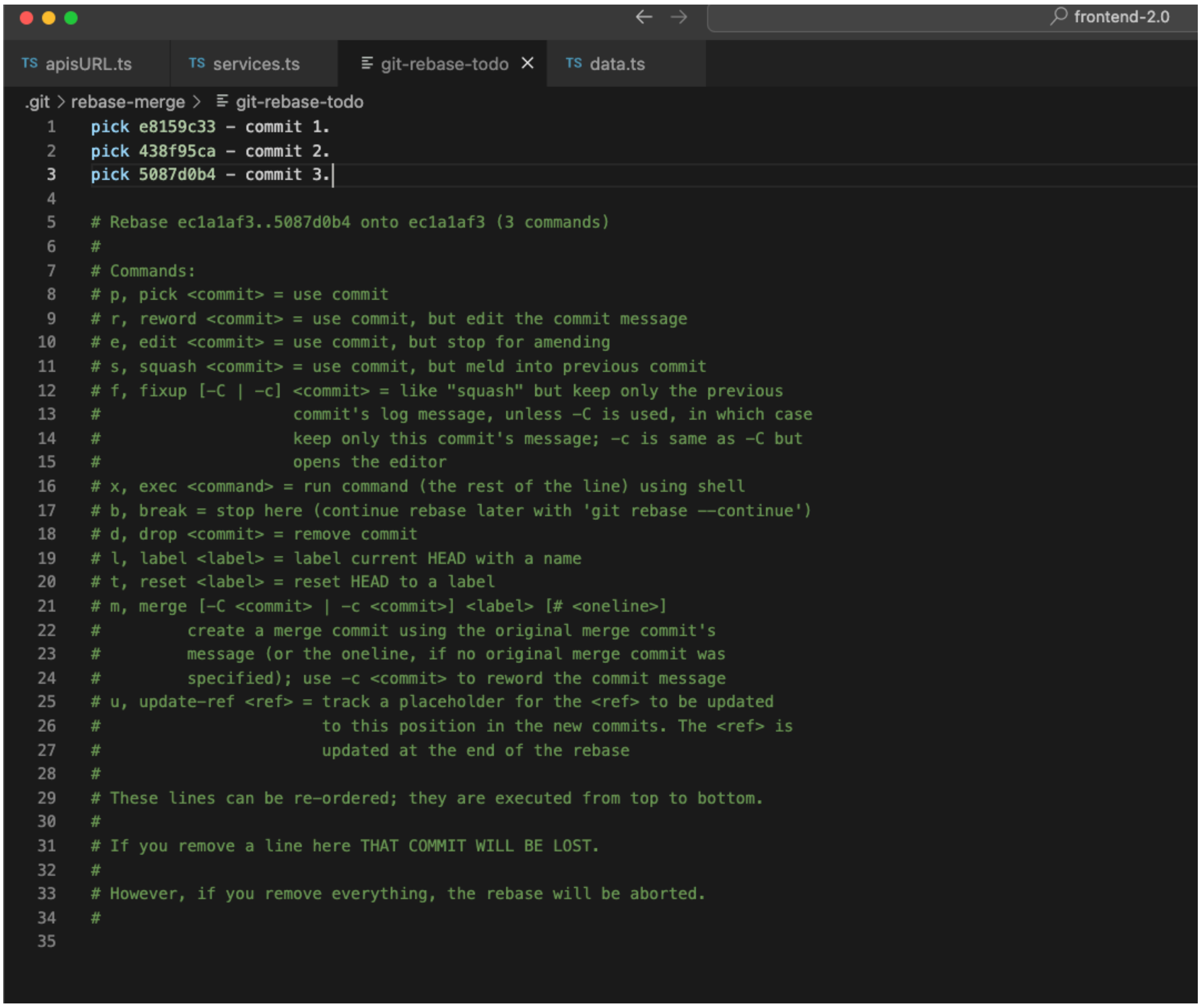

```
commit 5087d0b47c2cf6eb9ece0dd6a23c2633a36c0c51 (HEAD -> ftr/new-branch)
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 15:58:23 2024 +0300

    - commit 3.

commit 438f95cac8cdf9358ef823fe23d1aeca9cd8b17b
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 15:57:38 2024 +0300

    - commit 2.

commit e8159c33f3c3165c0806b11c0f83e44cd9422ea1
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 14:31:19 2024 +0300

    - commit 1.

commit ec1a1af34eac7c9f9fd5a509db892265b1fd9dbf (temp-master)
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 14:42:42 2024 +0300

    new commit deployed to master.
```

check the commit hash of the base commit (i.e. the commit hash of the whatever branch you based on):

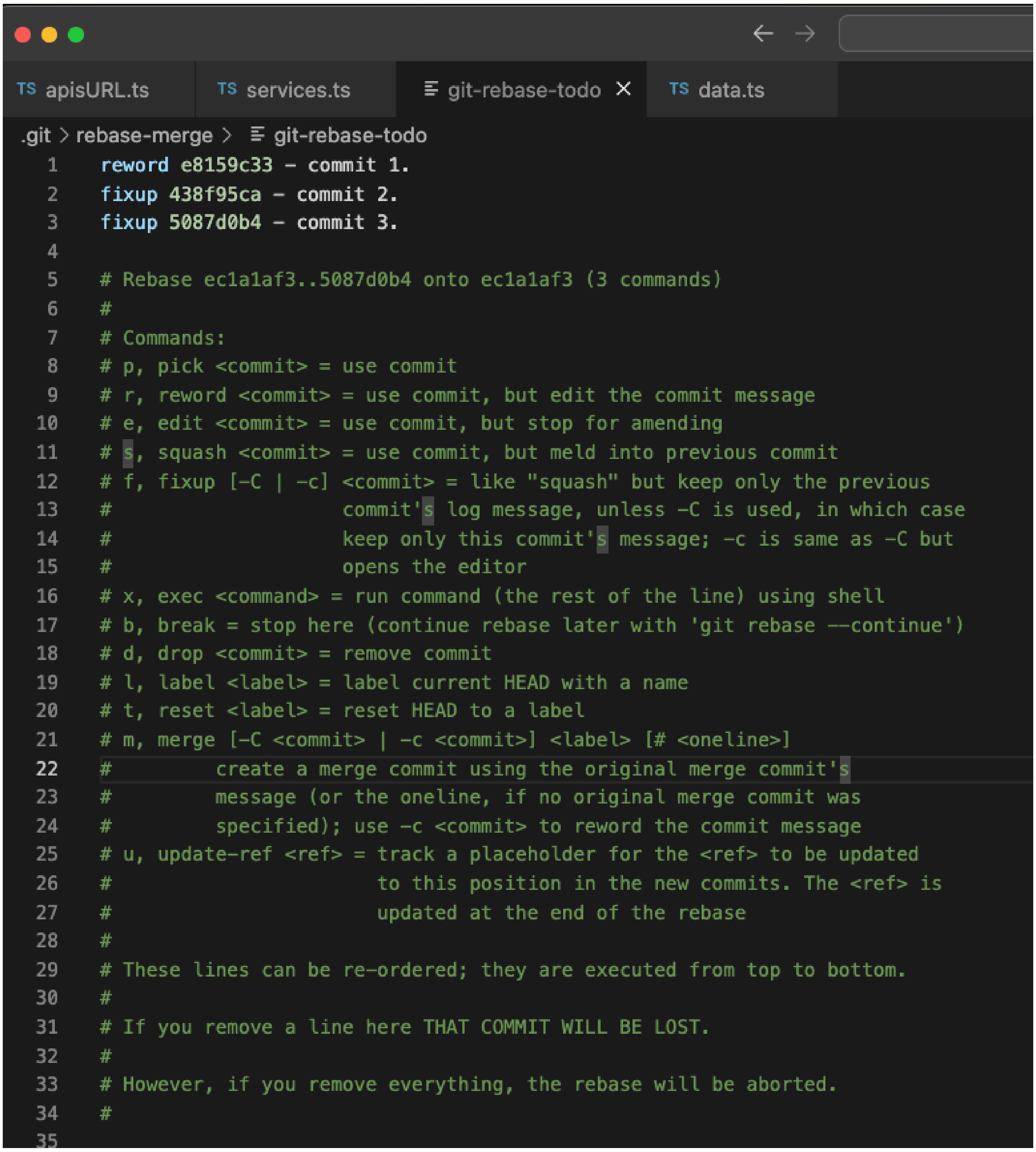


```
Date:   Mon Sep 16 15:58:23 2024 +0300

    - commit 3.

commit 438f95cac8cdf9358ef823fe23d1aeca9cd8b17b
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 15:57:38 2024 +0300

    - commit 2.

commit e8159c33f3c3165c0806b11c0f83e44cd9422ea1
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 14:31:19 2024 +0300

    - commit 1.

commit ec1a1af34eac7c9f9fd5a509db892265b1fd9dbf (temp-master)
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 14:42:42 2024 +0300

    new commit deployed to master.
```

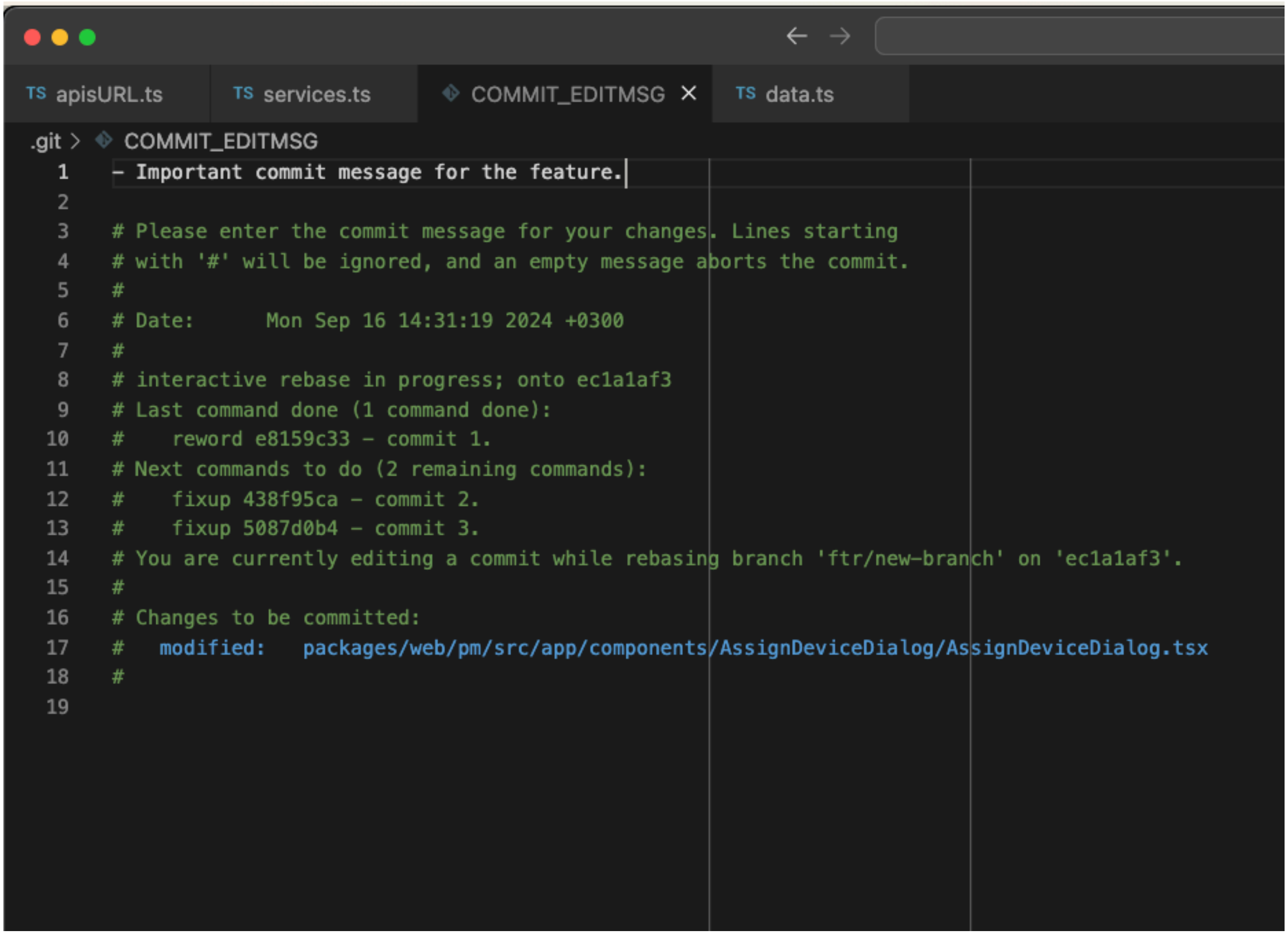
use the command git rebase -i {commit_hash} e.g. git rebase -i ec1a1af34eac7c9f9fd5a509db892265b1fd9dbf



```
pick e8159c33 - commit 1.
pick 438f95ca - commit 2.
pick 5087d0b4 - commit 3.

# Rebase ec1a1af3..5087d0b4 onto ec1a1af3 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#         create a merge commit using the original merge commit's
#         message (or the oneline, if no original merge commit was
#         specified); use -c <commit> to reword the commit message
# u, update-ref <ref> = track a placeholder for the <ref> to be updated
#                       to this position in the new commits. The <ref> is
#                       updated at the end of the rebase
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#

```



```
reword e8159c33 - commit 1.
fixup 438f95ca - commit 2.
fixup 5087d0b4 - commit 3.

# Rebase ec1a1af3..5087d0b4 onto ec1a1af3 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#         create a merge commit using the original merge commit's
#         message (or the oneline, if no original merge commit was
#         specified); use -c <commit> to reword the commit message
# u, update-ref <ref> = track a placeholder for the <ref> to be updated
#                       to this position in the new commits. The <ref> is
#                       updated at the end of the rebase
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#

```



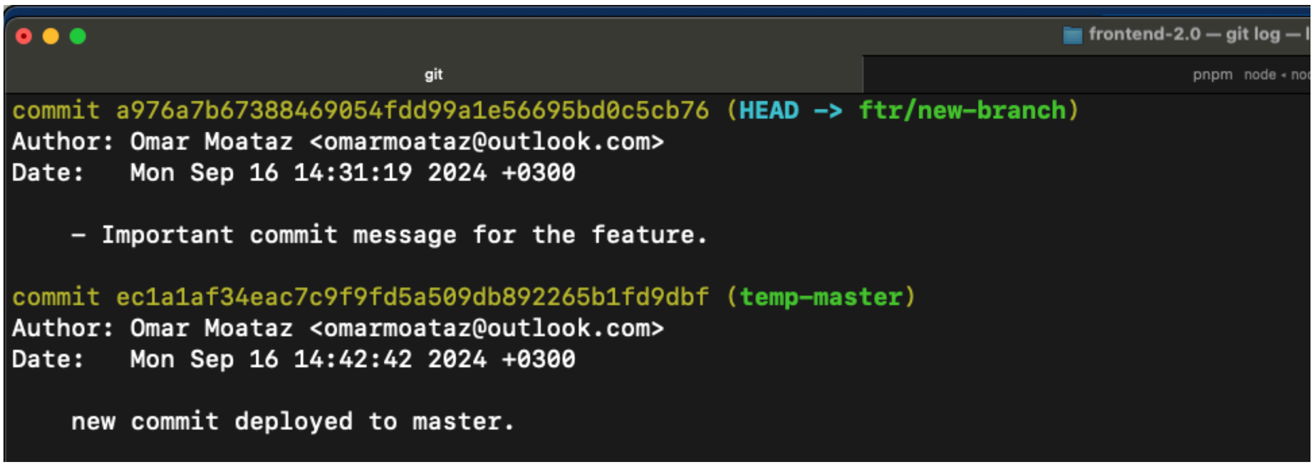
```
- Important commit message for the feature.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Mon Sep 16 14:31:19 2024 +0300
#
# interactive rebase in progress; onto ec1a1af3
# Last command done (1 command done):
#    reword e8159c33 - commit 1.
# Next commands to do (2 remaining commands):
#    fixup 438f95ca - commit 2.
#    fixup 5087d0b4 - commit 3.
# You are currently editing a commit while rebasing branch 'ftr/new-branch' on 'ec1a1af3'.
#
# Changes to be committed:
#   modified:   packages/web/pm/src/app/components/AssignDeviceDialog/AssignDeviceDialog.tsx
#

```

here git is giving you the opportunity to modify the message of the first commit you made.

Note: we're rewording because we want to change the commit message for that specific comment which will end up being the
only commit we have left.

We modified the message like we wanted and we save and quit.

if you rerun git log:



```
commit a976a7b67388469054fdd99a1e56695bd0c5cb76 (HEAD -> ftr/new-branch)
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 14:31:19 2024 +0300

    - Important commit message for the feature.

commit ec1a1af34eac7c9f9fd5a509db892265b1fd9dbf (temp-master)
Author: Omar Moataz <omarmoataz@outlook.com>
Date:   Mon Sep 16 14:42:42 2024 +0300

    new commit deployed to master.
```

↑ 2 🚀 1

0 replies

OmarMoataz on Sep 17, 2024 Collaborator edited

as per the discussion with the team, we will choose to create a parent branch for the story and then 2 sub branches for subtasks and then we will use either a rebase or merge strategy to sync the branches and then the parent branch

↑ 1 1 reply

syed-afzal60 on Sep 17, 2024 Collaborator

This strategy is for when we have two developers working on a same story

Write a reply

OmarMoataz on Sep 17, 2024 Collaborator

https://hakdogan26.medium.com/understanding-git-merge-vs-rebase-b1d6e4bccb1d
@syed-afzal60 link

↑ 2 0 replies

Write a reply

AhmedAlaaEzzt on Sep 18, 2024 Collaborator

Hello team,
Right now I'm working on the search feature I should create a protected feature for it, and create a temp branch for the dev and test.

↑ 1 2 replies

hamedafarag on Sep 18, 2024 Maintainer Author

Based on our discussion about the new Git flow, you should create a feature branch from the main branch (`master`). However, due to our current situation, you need to create it from the `testing` branch instead.

After ensuring it is ready to merge, create a new pull request from the feature branch into the `testing` branch so that the quality team can test it.

Hint: Be mindful of conflicts and ensure that your feature branch contains only your work.

❤️ 1

AhmedAlaaEzzt on Sep 18, 2024 Collaborator

ok now it's clear thank you

Write a reply

hamedafarag on Oct 14, 2024 Maintainer Author

UPDATES:

Based on our hands-on experience with this Git flow, we have the following updates:

- There will be two primary branches: `master` and `testing`.
- `master` serves as the production branch.
- Automatic deployments to the development environment will occur from the `testing` branch whenever a feature branch is merged into it.
- The `release/*` branch is a temporary branch used for deploying to stage and production, and it consolidates feature branches that are ready for release.
- The feature branch is the cornerstone of this Git flow.
- Before merging a feature branch, ensure that it is rebased with the interactive option, consolidating it into a single commit.
- Bug fixes should be made within the same feature branch. However, do not use interactive rebasing on bug fix commits to avoid changing the hash of already merged commits.

↑ 1

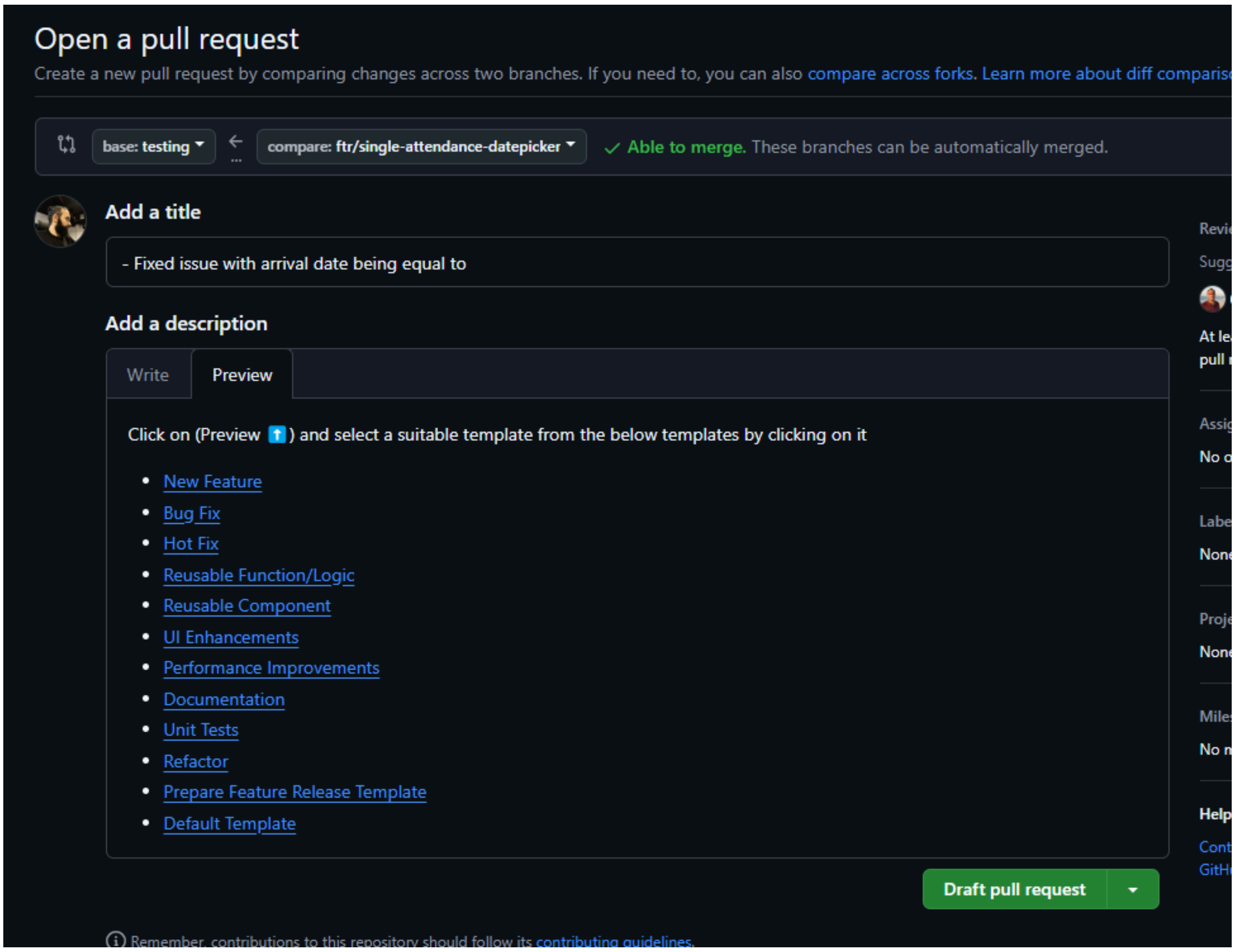
1 reply

hamedafarag on Oct 14, 2024 Maintainer Author

Main Discussion Updated based on these points

Write a reply

We have now multiple pull request templates, each one for a different purpose


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparis

base: testing ← compare: ftr/single-attendance-datepicker ✓ Able to merge. These branches can be automatically merged.

Add a title

- Fixed issue with arrival date being equal to

Add a description

Write Preview

Click on (Preview ⬆️) and select a suitable template from the below templates by clicking on it

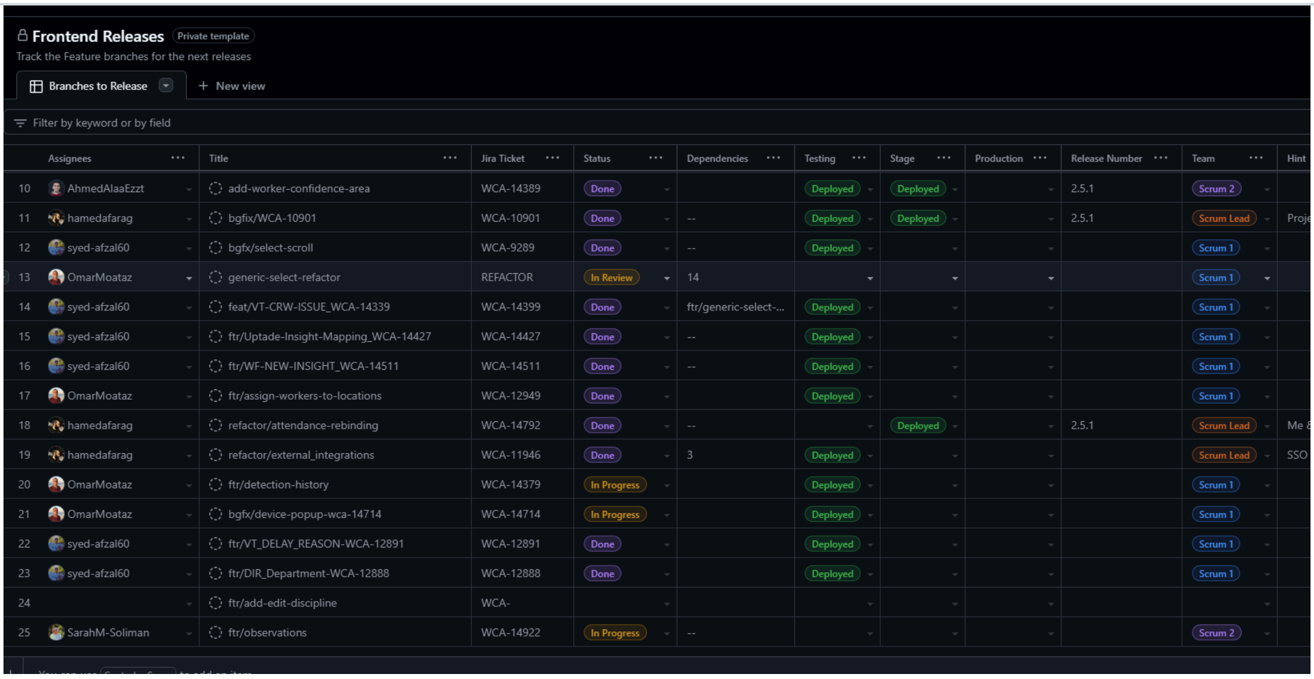
- New Feature
- Bug Fix
- Hot Fix
- Reusable Function/Logic
- Reusable Component
- UI Enhancements
- Performance Improvements
- Documentation
- Unit Tests
- Refactor
- Prepare Feature Release Template
- Default Template

Remember, contributions to this repository should follow its contributing guidelines.

↑ 1 ☺ 0 replies

Write a reply

hamedafarag on Oct 14, 2024 Maintainer Author



Frontend Releases Private template

Track the Feature branches for the next releases

Branches to Release | + New view

Filter by keyword or by field

	Assignees	Title	Jira Ticket	Status	Dependencies	Testing	Stage	Production	Release Number	Team	Hint
10	AhmedAlaaEzzt	add-worker-confidence-area	WCA-14389	Done		Deployed	Deployed		2.5.1	Scrum 2	
11	hamedafarag	bgfix/WCA-10901	WCA-10901	Done	--	Deployed	Deployed		2.5.1	Scrum Lead	Proje
12	syed-afzal60	bgfx/select-scroll	WCA-9289	Done	--	Deployed				Scrum 1	
13	OmarMoataz	generic-select-refactor	REFACTOR	In Review	14					Scrum 1	
14	syed-afzal60	feat/VT-CRW-ISSUE_WCA-14339	WCA-14399	Done	ftr/generic-select-...	Deployed				Scrum 1	
15	syed-afzal60	ftr/Uptade-Insight-Mapping_WCA-14427	WCA-14427	Done	--	Deployed				Scrum 1	
16	syed-afzal60	ftr/WF-NEW-INSIGHT_WCA-14511	WCA-14511	Done	--	Deployed				Scrum 1	
17	OmarMoataz	ftr/assign-workers-to-locations	WCA-12949	Done		Deployed				Scrum 1	
18	hamedafarag	refactor/attendance-rebinding	WCA-14792	Done	--		Deployed		2.5.1	Scrum Lead	Me
19	hamedafarag	refactor/external_integrations	WCA-11946	Done	3	Deployed				Scrum Lead	SSO
20	OmarMoataz	ftr/detection-history	WCA-14379	In Progress		Deployed				Scrum 1	
21	OmarMoataz	bgfx/device-popup-wca-14714	WCA-14714	In Progress		Deployed				Scrum 1	
22	syed-afzal60	ftr/VT_DELAY_REASON-WCA-12891	WCA-12891	Done		Deployed				Scrum 1	
23	syed-afzal60	ftr/DIR_Department-WCA-12888	WCA-12888	Done		Deployed				Scrum 1	
24		ftr/add-edit-discipline	WCA-								
25	SarahM-Soliman	ftr/observations	WCA-14922	In Progress	--					Scrum 2	

↑ 1 ☺ 0 replies

Write a reply