Sequential Git Flow

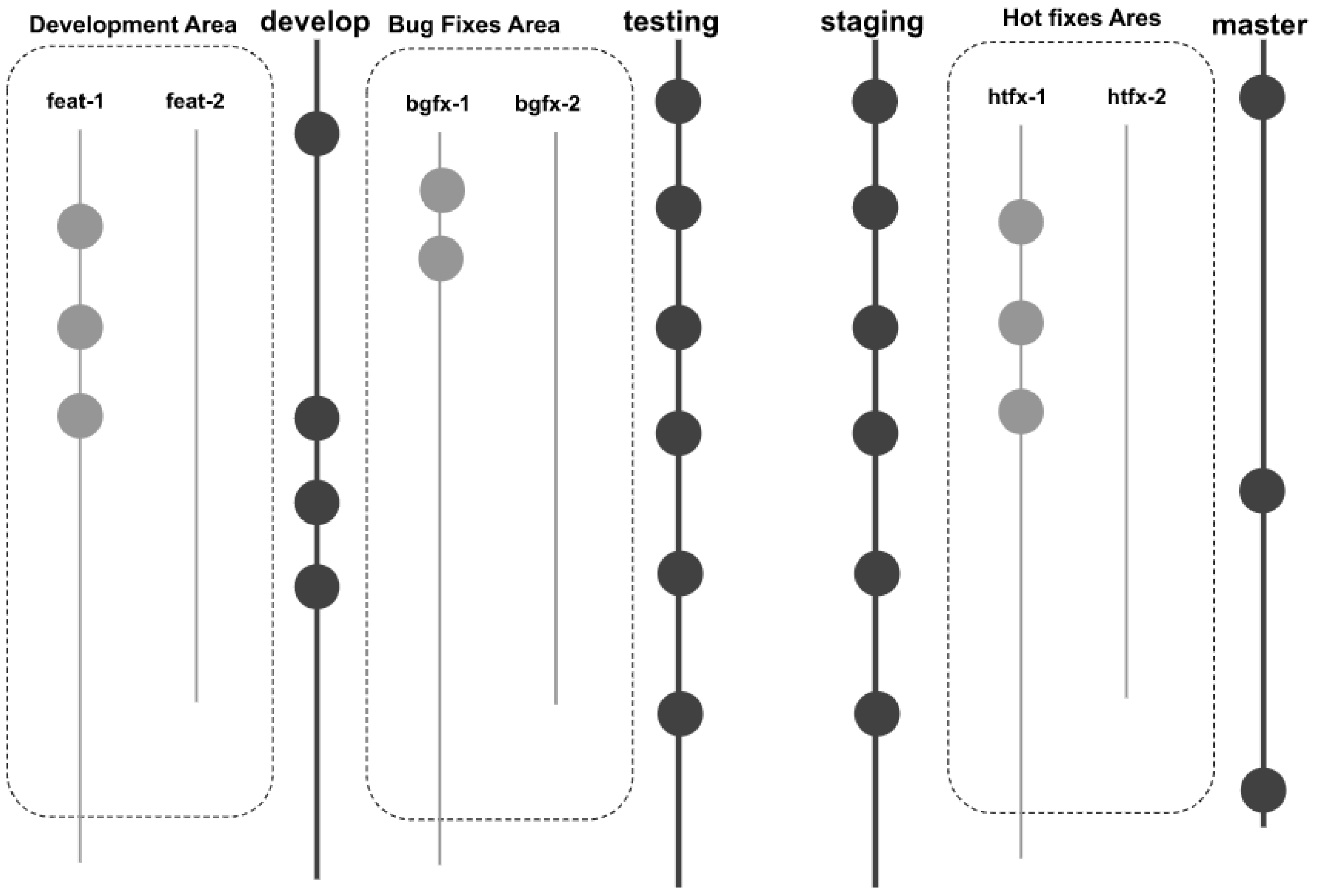
We have for the frontend repository "four main branches", each branch presents mainly an environment as follows,

- **develop** branch -> **development** environment
- **testing** branch -> **testing** environment
- **stage** branch -> **stage** environment
- **master** branch -> **production** environment

Also we have something called areas, and we have three areas, **development area**, **bug fixing area** and **hot fixes area**.

Staging is a replica from the production and the last line of defense for our features and fixes before deploying them to the production environment.

The following figure will present the above description



Git Flow

The main reason for creating Git Guideline is to avoid "cherry-picking" action as much as we can, we may need it but in very rare cases and not in normal cases or in the daily routine when we need to merge our code.

Creating a new "feature branch"

1. Create a new branch (***ftr/feature_1***) from **develop** branch.
2. Work normally and push your commits.
3. Time to merge your code, Obviously, you need to merge your code into **develop** branch as the first step for testing your code with the whole system.
    1. You have to create a PR to merge your branch (***ftr/feature_1***) into **develop** branch.
4. Create PR and follow the PR Template, for merging your code.
5. If your feature is ready for testing create a PR to merge the develop branch into the testing branch (MERGE DEVELOP INTO TESTING).
6. If your feature is ready for release, create a PR to merge the testing branch into the staging branch (MERGE TESTING INTO STAGING), and then create a PR to merge the staging branch into the master branch (MERGE STAGING INTO MASTER).

Creating a new "bug fix branch"

1. Create a new branch (***bgfx/bug_1***) from **testing**.
2. Fix the issue normally and push your commits.
3. Time to merge your fix, you need to merge your code into **testing** branch, as the bug reported from the testing environment.
4. Create PR and follow the PR Template, for merging your code.
5. If the bug is fixed on testing, then we need to merge back the testing branch into the develop branch to update the develop with that fix.
    1. Usually we do this by creating a PR to merge the testing branch into the develop branch at the end of the day.

Creating a new "hot fix branch"

1. Create a new branch (***htfx/htfx_1***) from **master** or **staging** it will be based on reported bug.
2. Fix the issue normally and push your commits.
3. Time to merge your fix, you need to merge your code into **master** or **staging** branch, as the bug reported from one of them.
4. Create PR and follow the PR Template, for merging your code.
5. If the bug reported from **staging** and this bug also exist in the production, then we need to merge also (***htfx/htfx_1***) into the master to update the production with that fix.
    1. Or we can merge the stage branch into the master branch bases on the situation.
6. If the bug is fixed on stage or production and everything is fine, we need to merge back to update all the main branches.
    1. Usually we do this by creating a PR to merge the testing branch into the develop branch at the end of the day.

Prefixes, Branches and tags name

We will use the following **prefixes** following with a Slash `/`

- **ftr/**, for Feature branch.
- **bgfx/**, for Bug Fix branch.
- **htfx/**, for Hot Fix branch.
- **refactor/**, for Refactoring branches.
- **exp/**, for Experimental branches - for POCs, trying new library, new implementation of existed feature - .

Branch name should contains, **Jira Ticket Number** and a **short descriptor** of the task with **hyphens (-)** as separators

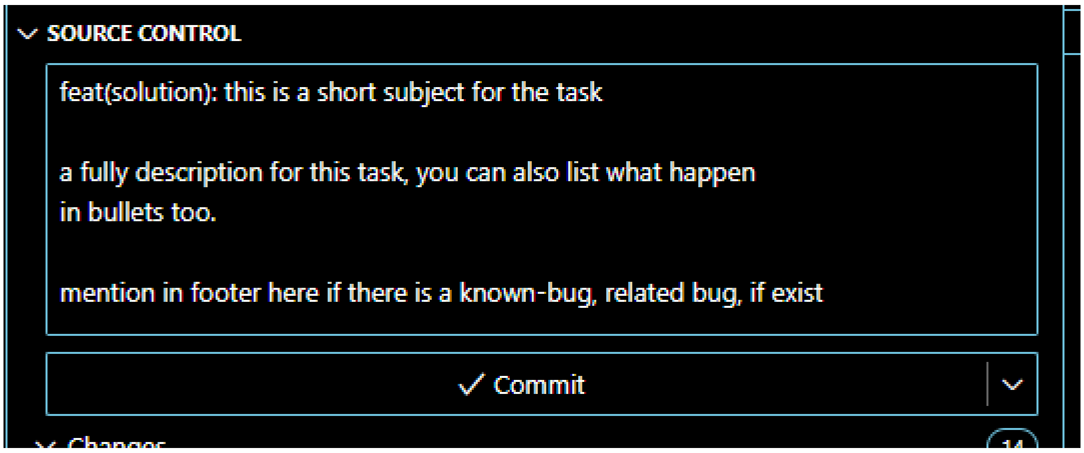
> Example: `ftr/WCA-78453-create-login-form`

Tags name should contains only version number `v1.0.2` ,

How to write a commit [ref]

Frontend solution use commitlint as a checker for the commit message. You can commit through two ways only

1- **VSCode**

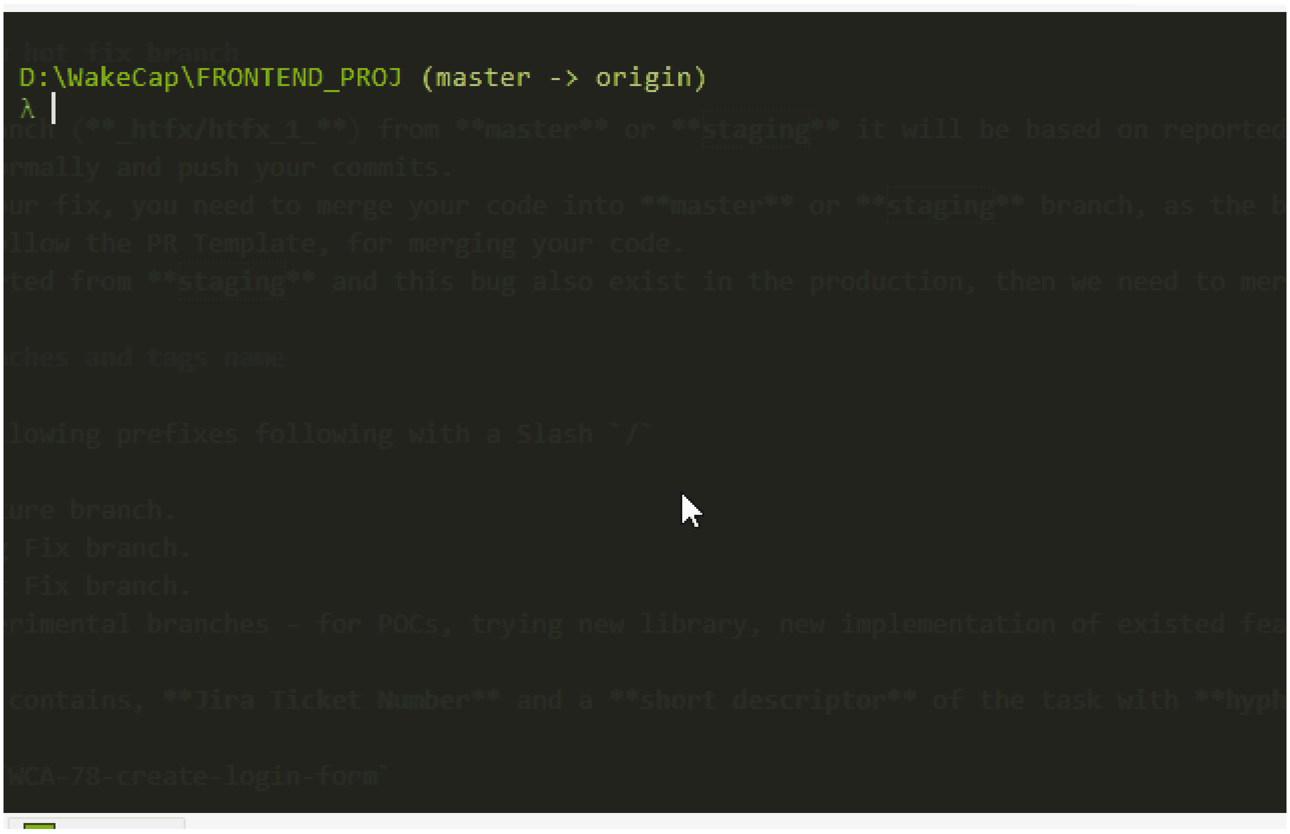


feat(solution): this is a short subject for the task

a fully description for this task, you can also list what happen
in bullets too.

mention in footer here if there is a known-bug, related bug, if exist

2- **command-line**



```
D:\WakeCap\FRONTEND_PROJ (master -> origin)
λ |
```

Here is how to write a commit message, and you cannot push your commit unless you follow the pattern

```
feat(scope): short subject

body

footer
```

Copy

The **type** value must be one of: [**build**, **chore**, **ci**, **docs**, **feat**, **fix**, **perf**, **refactor**, **revert**, **style**, **test**]

The **scope** value must be one of:

```
[
  "docs",
  "solution",
  "web_root",
  "web_nav",
  "web_admin",
  "web_pm",
  "web_mt",
  "shared_ui",
  "shared_tui",
  "shared_icons",
  "shared_helpers",
    "shared_comps"
  ]
```

Copy

> **Scope** will has a new values if we have a new projects in the future, for each new project we should add a new value for the scope in `.commitlint.json` file.

How to write a PR

The Pull Request is an event that takes place in software development when a contributor/developer is ready to begin the process of merging new code changes with the main project repository.

Not only are PR descriptions helpful context for the team, they also train you to more concisely and effectively communicate, which is commonly an undervalued skill. Improving your communication skills will make you a better programmer and teammate.

Pull Request Subject Template
`[ENV][TYPE](JIRA_TICKET_NUMBER): Short Subject for this Pull Request`

Example `[DEV][FEAT](WCA-125): Add Login Form`

Pull request Description Template, Copy and Paste in the description area, if it is not populated automatically from bitbucket/github.

```
# Description

Please include a summary of the changes and the related issue. Please also include r

## Jira Tickets

- WCA-1101

## Type of change

**Please delete options that are not relevant.**

- [ ] Bug fix (non-breaking change which fixes an issue)
- [ ] New feature (non-breaking change which adds functionality)
- [ ] Breaking change (fix or feature that would cause existing functionality to not
- [ ] This change requires a documentation update

## Known Bugs & Impacts

Please write down any impact your code may cause it, and also if there is any known-

# Checklist:

- [ ] My code follows the style guidelines (Tailwind, Figma) of this project
- [ ] My Styles respect the RTL approach.
- [ ] My Code respect the Semantic HTML
- [ ] I have performed a self-review of my code
- [ ] I have commented my code, particularly in hard-to-understand areas
- [ ] My changes generate no new warnings
- [ ] New and existing unit tests pass locally with my changes
- [ ] I have written a storybook for documenting my reusable component or code.
- [ ] I have added tests that prove my fix is effective or that my feature works
- [ ] I have made corresponding changes to the documentation

- [ ] I support my code with **data-/\*** attribute for automation tests.
- [ ] I have written a storybook for documenting my reusable component or code.

## Screenshots

Paste here any screenshot from your component, screen, page, or whatever it needs tc
```
Copy

Hints

1. Branches will be deleted after each merge.
2. Don't forget to merge back the main targeted into the branch you want to merge before creating the PR.
3. PR will not review unless the PR has a proper description and the subject contains Jire Ticket number.
4. Merging will be a Squash merge not a normal. but in some cases (between the main branches), we may need to merge normally.
5. A new tag will be created after each merge into **master** branch.
6. At the end of the day, the Git master - git master is a role will be played by any one in the team - responsible for merging back the main branches as follow

```
- MERGE MASTER INTO STAGE
- MERGE STAGE INTO TESTING
- MERGE TESTING INTO DEVELOP
```
Copy

7. Feature Flag feature will be introduced somehow and this will be handled case by case and will be from the frontend side only.
8. Don't ever push code directly to the main branches!.
9. Make Pull Request small as much as you can, you can plan to create a feature branch and create sub-branches from this branch and create multiple PRs to merge the code into this feature branch, this will be easier in reviewing and will make the reviewing process faster.(**#share-your-work-early**)

Quick Tutorial

hamedafarag Aug 8, 2024 Member

This is popular Git Flow and it works great with one team. but today we have a several problems specially the problems related to releasing a new build to any environment.

What are the problems?

1. How to release selected features?
2. We have two teams, working on the same code base, how to release two different releases without touching or grab any other feature from the other team?
3. How to implement a reusable component by Team A, and Team B want to use this component?

↑ 1 0 replies