# Git

## Branches

Basically all our repositories should be arranged like this:

- `master` branch for production
- `staging` branch for staging
- `development` branch to create your development branches from

### Naming Convention

We have a naming convention for the branches which is as follows. Please replace `feature` with `bugfix`, `hotfix` or whatever if needed:

```
feature/code/TICKET-#-description
```

> ℹ We will use the alias of `/...` as postfix to branches whenever we are referring to this convention in this guide. For example `feature/...` refers to a feature branch with the naming convention described above.

## Creating a New Branch

### Feature Branches

When you start on a new feature, make sure to do so in a branch prefixed with `feature/...` as described in . To create a new branch make sure you are in the `development` branch first. Then create new branch from it, using the naming convention.

```
git pull origin development
git checkout development
git checkout -b feature/tjv/GRETA-123-updating-the-install-command
```

> ℹ
> - **Bugfix** branches use the prefix `bugfix/...`
> - **Hotfix** branches use the prefix `hotfix/...` (s. )

# Merging

To merge branches, you should use Private (https://app.clickup.com/2457460/docs/2azvm-512/2azvm-624). That is because we use the merge event to trigger our GitLab pipelines to handle our Private (https://app.clickup.com/2457460/docs/2azvm-512/2azvm-386)

## Merge Flow

Make sure that you only merge your branches into `development`. Then you can merge `development` into `staging` and if needed you can then push from `staging` into the `master` branch.

- `feature/...` ➡ `development` ➡ `staging` ➡ `master`
- `bugfix/...` ➡ `development` ➡ `staging` ➡ `master`

# Hotfixes

Hotfixes are an exception to the usual *Merge Flow* described above. Hotfixes are crucial bugfixes that need to be applied as soon as possible. For example a hotfix for the production environment should not be fixed for the `development` branch and then wait for the deployment to production until the next version. It should rather be fixed for the `master` branch and then deployed directly to the production environment.

So here is the process to apply for hotfixes in the production environment.

1. Checkout the `master` branch on your local environment.
2. Create a `hotfix/...` branch for your fix.
3. Merge your `hotfix/...` branch into the `master` branch through gitlab.

- `hotifx/...` ➡ `master`

# Clean up your Branches

Sometimes you need to clean up your branches. You can use the following command to remove all branches that have already been merged, except for your current branch, master, development and staging.

```
git branch --merged | egrep -v "(^\*|master|development|staging)" | xargs git branch -d
```

 **Tip:** Create an alias for it in your `~/.bashrc` / `~/.bash_aliases` or `~/.zshrc` / `~/.zsh_aliases` file, depending on which terminal you use (bash or zsh).

```
# alias
alias gbDA='git branch --merged | egrep -v "(^\*|master|development|staging)" | xargs
git branch -d'
```

## Assume unchanged

Sometimes you need to do some changes only for testing. To make sure you don't commit and push them, you can update the index of your stage to make git assume a given file is still unchanged, even though you have made changes to it.

```
git update-index --assume-unchanged src/Controllers/SomeControllerWithDDInIt.php
```

And when you are done with your tests and want to continue working with the given file just like with every other in your git repo, then you can revert the assumption by git with the following command.

```
git update-index --no-assume-unchanged src/Controllers/SomeControllerWithDDInIt.php
```