Commits and branches

When creating branches and commits, it is essential to follow a consistent naming convention. This ensures that our team can easily understand the purpose of each commit and branch.

Branch Naming Convention:

There are 4 Permanent branch names : `master` (legacy), `main` , `staging` , `develop`

When developing :

Branches should be named according to the JIRA / linear ticket number associated with the task. The format should be `feat/PG-123` , `fix/PG-123` , or `test/PG-123` depending on the type of task.

Type in front of "/" : This describes the type of change. It must be one of the following: `feat` , `fix` , `docs` , `style` , `refactor` , `perf` , `test` , `hotfix` , `release`

Must be one of the following:

- **build**: Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
- **ci**: Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)
- **docs**: Documentation only changes
- **feat**: (or feature - legacy) A new feature
- **fix**: A bug fix

- **hotfix**: an urgent fix that needs quick release to production (use carefully)
- **release**: a package of features / fixes etc in a new version
- **perf**: A code change that improves performance
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **style**: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
- **test**: Adding missing tests or correcting existing tests

Commit Naming Convention

Commits should follow the Conventional Commits specification. Each commit message should consist of a header, optional body, and optional footer. The commit message should be structured as follows:

1. **Type:** This describes the type of change. It must be one of the following: `feat`, `fix`, `docs`, `style`, `refactor`, `perf`, `test`

   For example,
   `feat` for a new feature and `fix` for a bug fix.

   Must be one of the following:

   - **build**: Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
   - **ci**: Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)
   - **docs**: Documentation only changes
   - **feat**: A new feature
   - **fix**: A bug fix
   - **perf**: A code change that improves performance
   - **refactor**: A code change that neither fixes a bug nor adds a feature

- **style**: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
- **test**: Adding missing tests or correcting existing tests

2. **Scope:** This is optional and refers to the scope of the commit. It could be anything specifying the place of the commit change.
3. **Subject:** This is a brief description of the change.
4. **Body and Footer:** These are optional and used to provide more detailed descriptions about the change and any breaking changes or issues closed by the commit.
5. **BREAKING CHANGE:** a commit that has a footer `BREAKING CHANGE:`, or appends a `!` after the type/scope, introduces a breaking API change (correlating with `MAJOR` in Semantic Versioning). A BREAKING CHANGE can be part of commits of any *type*.

For example: `feat(JIRA-123): add new login button`.

See : https://www.conventionalcommits.org/en/v1.0.0/#specification for full specificaitons