

ATLAS

GIT methodology

[Branch Methodology](#)

[Branch Naming conventions](#)

[Live deployment](#)

[Development workflow](#)

[Repository Structure](#)

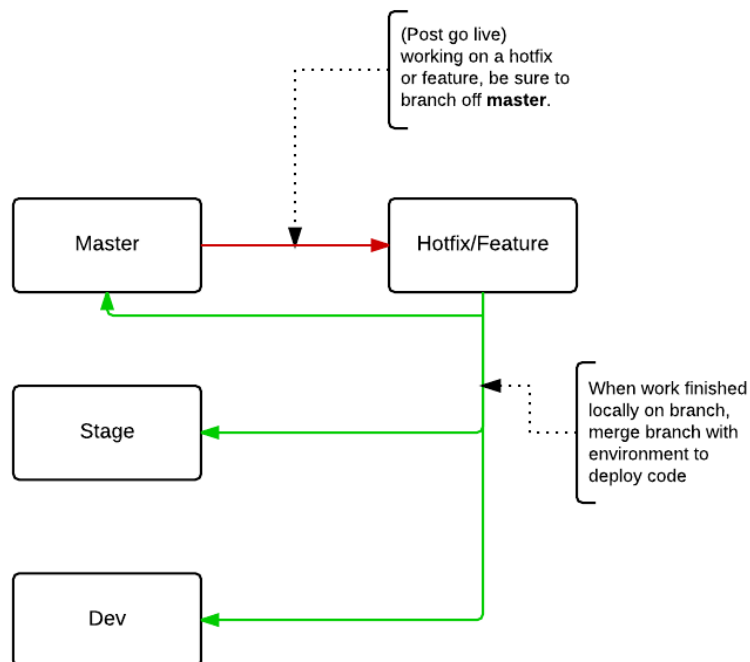
Branch Methodology

We use 3 main branches to represent the various environments used on a project. Generally, only 2 of the environments will be used on a project, but if we have maintenance work happening in parallel to a new feature, for the same project we might use both **stage** and **dev** for client review.

The following branches will be defined for **all** project repositories.

- **master** - this represents the code that should be present on the live server.
- **stage** - this branch represents the code that is visible on the client stage environment. It should be stable code that is suitable for the client to review and test pending approval for deployment to production.
- **development** or **dev** - this branch should be used to store a dev version of the code that should be pushed onto the “external development” server. Once code is pushed to this it will be automatically pushed.


When working on a feature or hotfix, branch off **master** and always work within that branch. Then once happy with the work, merge into the branch required environment branch.





Branch Naming conventions

To avoid confusion when multiple engineers are working on a project, and one engineer has to pick up a task from another, or deploy a task they haven't worked on, we must use a consistent naming convention for our branching e.g. **505_example**


We use a naming convention that relates individual branches back to their corresponding task in ActiveCollab, basic format being **{task_number}_{short_description}** e.g.




#505: Example task




FELD - Consolidated Maintenance | Task #505 | Created by Courtney Adesile on Nov 18, 2020

 **Add a Subtask**

 **Add a Dependency**

Discussion



For the task above the branch name is **505_example**

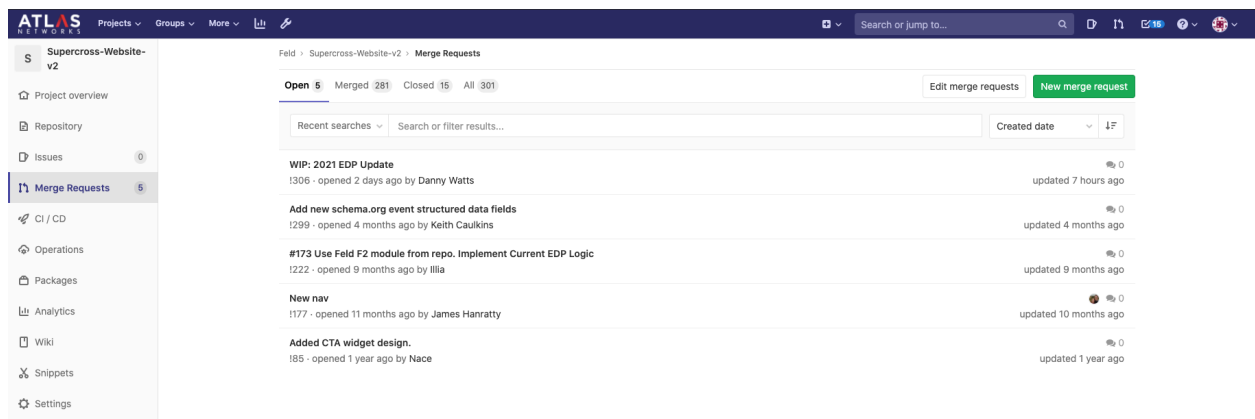
1. ActiveCollab task id, this is **critical** must be present when creating a branch
2. Small description / title of task - *try to keep this short*

IMPORTANT: When creating a feature branch, you **MUST** branch off **master**. If you don't create your feature/hotfix branch off master, you could potentially merge code still in development, into the master branch (and production environment).

Live deployment

For internal and client review, deploy your work to the stage environment by merging your feature branch into the stage branch for the project. Use the same process if you want to deploy to the development environment (merge into the **dev** branch).

After your work has been reviewed and any feedback/amends completed, it will be ready for live deployment. For live deployment we use **merge requests**, we use gitlab to create merge requests e.g.



Go to the project repository in Gitlab, then from the left-hand navigation goto **merge requests** and click on the **new merge request** button which will take you to the form below:

New Merge Request

Source branch


atlargeinc/imgor-new


Select source branch

Target branch


atlargeinc/imgor-new

master

 Merge branch 'master' into 221_cart_menu_item
Courtney Adesile authored 4 days ago



7eb03ea5



Compare branches and continue

Your feature branch will need to be pushed to the remote repository to be visible in the form above. Once the merge request has been created, it will be the responsibility of the project lead to review and action the merge request.

Development workflow

All developers should work and test on their local machines in the first instance. Once a feature they have been working on has been completed and tested it should be committed to their local repository. These changes can then be pushed up to the main git repository.

Code from the main repository will be deployed to the dev platform and stage platform as required. If the client has agreed that a version on the stage platform is complete, a merge request will be created and the feature deployed onto the clients production environment.

It is our aim to catch bugs whilst in a local or external development environment. Both of these are non-client focussed and should not be distributed to the client under any circumstances. As a team we should test code on the “external dev” each time a new version is pushed out.

Repository Structure

All projects will have to following file/directory structure:

```
./site|deploy  
./static  
./docs  
./assets  
./env  
./env/local  
./env/dev  
./env/stage  
./env/prod
```

- the **site/deploy** directory will contain the web site files, picture this as an “htdocs”.
- the **static** directory static markup for the website produce by the frontend engineer.
- the **docs** directory will contain any files provided by the client. Files should be managed using further sub directories.
- the **assets** directory will contain any design assets which are needed in the repository e.g. PNG dumps of page designs etc
-
- the **env** directory will contain any per-environment configuration, settings.php, htaccess, custom php configuration, robots.txt.