Git Workflow

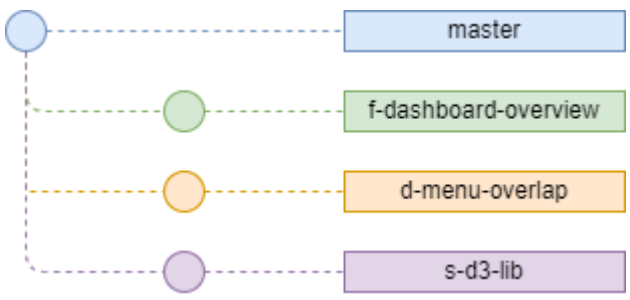
Creating Working Branches

A working branch is *always* created from a main branch such as `master` or a release branch. Each branch is named based on the role of the branch. See the *Naming Conventions* section of Branch Management page.

A working branch is then developed until completion or until it is ready to be tested for release, at which point a merge request is created to complete the review and merge process (see Merging section below).

Example:

> Kaitlyn creates a feature branch `f-dashboard-overview` in order to begin development of an application feature that was requested by the business team. At the same time, Armin begins working on fixing a defect that was introduced in a previous release and creates a branch called `d-menu-overlap`. He is able to fix the defect very quickly and immediately begins working on adding the D3.js library into the application dependencies and testing it for graphing capability in the application. He creates a branch called `s-d3-lib`.



Merging Branches for Testing and Release

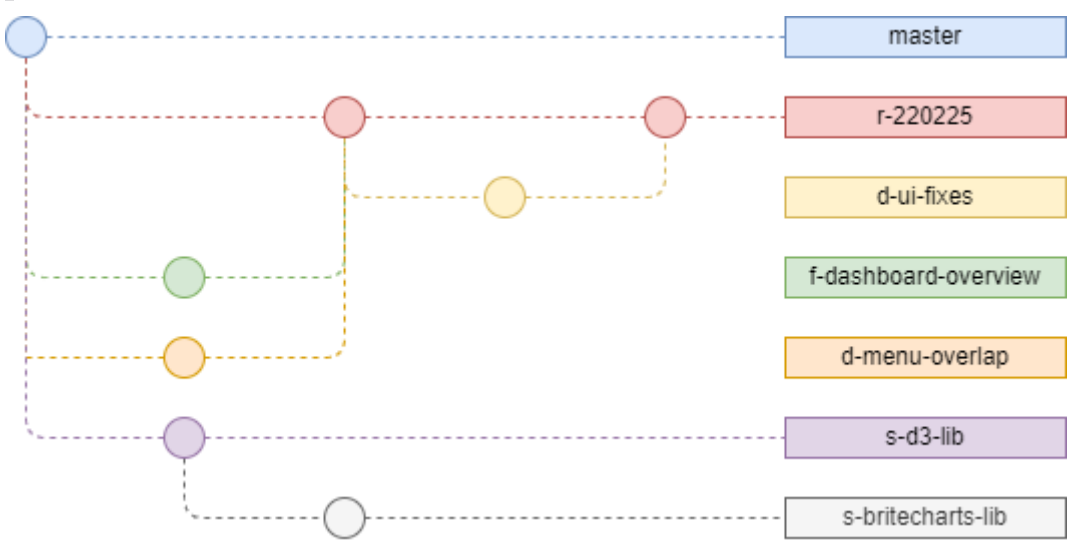
When work is completed on a branch, a merge request (or pull request) is created. Merge requests from a working branch are *always* targeted at a release (or testing) branch and never at `master` (or other main branch). See the Hotfixes section below for the one exception to this rule.

The release branch (or 'release candidate' designated with the `r-` prefix) is a special working branch into which all other working branches are merged for testing before release. In some cases, the release branch may be deployed to a UAT environment first to get business validation. Once tested and/or validated, the release branch can then be merged into the `master` (or another main branch).

See the Merge Requests page for information on how merges actually happen. Never merge branches into each other directly unless you are doing so downstream and you own both branches.

Example, continued:

> Business asks Jamal to showcase the latest feature that Kaitlyn had developed by deploying it to a UAT environment for testing. At the same time, Jamal notices that Armin has also fixed an existing issue, so he creates a release branch called `r-220225` and merges all of the merge requests that were reviewed and approved and immediately deploys it to a UAT environment. Business notices small misalignments of a couple of UI elements and one typo. Jamal asks Kaitlyn to make the quick fixes and then merges those fixes (`d-ui-fixes`) back into the release branch. Meanwhile, Armin has realized that a higher-level library based on D3.js is much more suitable for the application's needs and creates a new branch for testing out Britecharts called `s-britecharts-lib`.

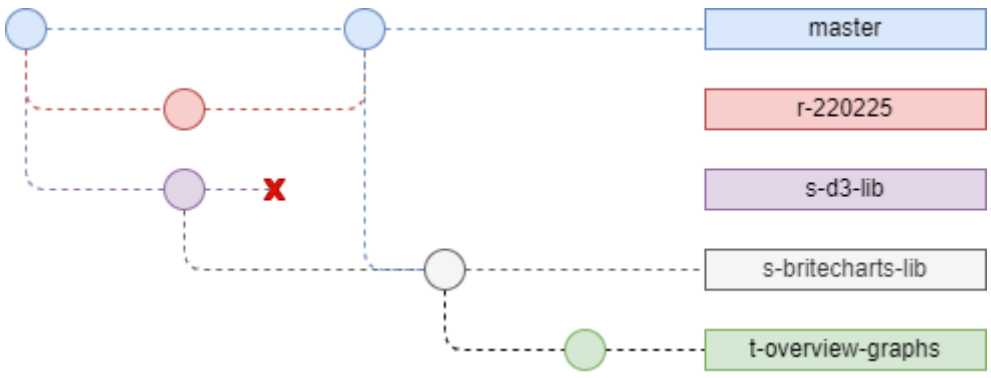


Once a working branch is merged into a release branch, it is deleted. The same strategy is followed for merging release branches into the main branch. If a branch is "abandoned" or is no longer needed, it is the responsibility of the branch owner to delete it. The repository should remain clean and only contain active branches.

After a release is made the main branch may be ahead of a particular working branch if that working branch has not been included as part of a recent release. In this case, it is a good idea to update the working branch with the latest `master` (or other main branch).

Example, continued:

> The latest release candidate was approved during the UAT phase and Jamal merged the branch `r-220225` into `master` and labeled the release. All of the working branches, including the release branch, were deleted as part of the merge strategy. Armin continues to focus all of his efforts on implementing the Britecharts library and is ready to test its capabilities in the dashboard overview module that Kaitlyn had implemented by rendering some graphs. In order to access this module, Armin merges the branch `master` into his working branch `s-britecharts-lib` so it is up to date and contains all of the latest code from the last release. He then creates a new test branch from his working branch to showcase some sample graphs to demonstrate and explain to his team how the graphing library works and how well it performs. He calls this branch `t-overview-graphs`. The intent of this branch is to demonstrate and test the feature out thoroughly rather than to actually release it.



Hotfixes

There may be times when a business-critical defect is found in a package and needs to be urgently fixed. This is where a special working branch of type 'hotfix' (designated by prefix `h-`) comes into play.

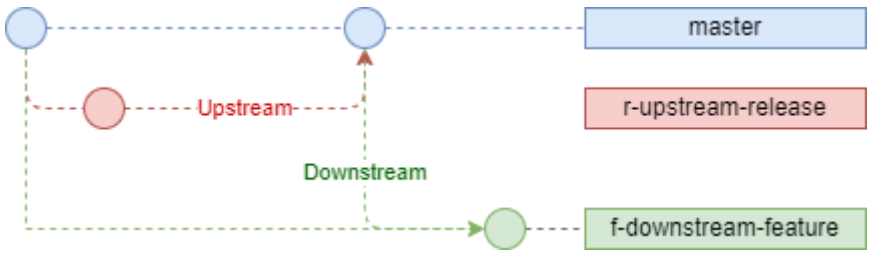
The hotfix branch is the only type of branch that may be created from `master` (or another main branch), worked on, and then merged directly back in without needing to be merged into a release branch first.

When this occurs, all other working branches *must* be updated with the latest code from the main branch. In other words, after its release, the hotfixed issue must be updated in every branch that exists in the repository.

You can read more about Defect Severity Levels and which types of defects would require hotfixes.

Merge Requests

A merge request (or pull request) is to be made for any upstream merge. All conflicts are to be resolved before a merge request is flagged for review, and all requests are to be reviewed and approved before a merge is completed.



Before Creating a Merge Request

Before creating a merge request it is the responsibility of the branch owner to ensure:

1. The branch is fully tested, is functionally complete and release-ready
2. All unit and integration tests pass
3. The code conforms to the Style Guideline

In some cases, a large enough feature or test branch may need to be deployed to a UAT environment for validation by a business team or even a customer before being submitted for review.

Creating a Merge Request

Once the branch is ready to be merged, a merge request is created and reviewers are assigned.

When creating a merge request include a title briefly describing the feature, enhancement, issue fixed or other work that is included in the branch.

In the description section of the merge request, include a link to the corresponding task or card in the project management tool used to manage tasks. You may also include additional notes or a list of major changes to which special attention is required.

Resolving Conflicts

Conflicts should be resolved before assigning reviewers to the merge request while ensuring other developers' work is not affected (i.e. most of the time keeping both HEAD and new branch content).

It is always a good idea to update your working branch if it is behind `master` which helps address conflicts early.

Conducting Reviews

When reviewing merge requests, ensure that:

1. The changes make functional sense
2. The solution is the most sensible and efficient possible
3. The code conforms to the Style Guideline

The merge request is where most of the communication about the code takes place. Do not hesitate to ask questions and leave comments within the merge request.

Completing the Merge

Once the merge request has been reviewed, all comments have been resolved and the merge has been marked as approved, then it can be completed.

When conducting the merge, ensure that the source branch is deleted to keep the repository clean. Only active branches should exist in the repository.

Release Merges

When a release branch is ready to be merged into `master` (or another main branch), it is imperative that the release branch is tested thoroughly before creating the merge request.

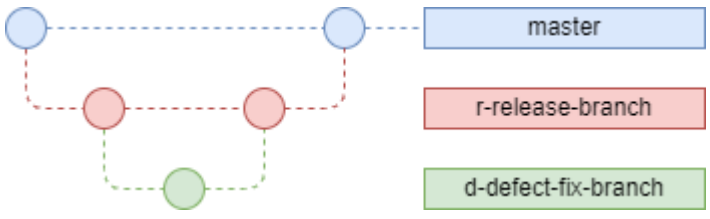
In most cases, if the release is a versioned release, then a customer-validated UAT phase may be necessary. Read more about Versioning for more information on versioned releases.

Tagging

A release should always be tagged at the merge commit in the short form (ex: `r220220`).

Reverting Changes

Reverting committed changes should be avoided as much as possible. Instead, additional code changes should be made on a newly created branch which can then be merged back into the source branch.



The main goal is to maintain commit history as much as possible.

In the event that reverting changes is absolutely necessary, document the changes thoroughly and descriptively and use `git revert` rather than `git reset`.