**The Basics**

Below, the Git flow to manage creating, publishing, merging, and closing branches:

![A picture containing sky, map

Description automatically generated]()

The branches are described in the following table:

| **Branch** | **Consideration** |
| --- | --- |
| prod (Production runs off of prod branch) | - Never edit prod directly - Hotfixes are forked off of prod - Releases branches are merged into prod |
| hotfix | - Forked off of prod - Merged back into prod and master when closed |
| master | - Feature, epics and release branches are forked off of master - Never edit master directly - master is never merged directly back into prod - This branch must be kept as clean as possible - only code reviewed and approved features/epics should be merged |
| releases | - Forked off of master - Merged into prod when closed |
| epics/ | - Forked off of master - Merged into master when closed - Use interactive rebase to merge into master |
| feature/ | - Forked off of epics (or master if is an isolated task) - Feature branches must be rebased (interactive) off of epics/ and code reviewed (pull request) before they can be merged back in |

**Workflow**

1. **Create a feature branch**

* First you need to make sure master (or epic branch) is up to date by doing a pull (fetch / merge).
* Then create a feature branch off of master

git checkout master  
git fetch origin master  
git reset --hard origin/master  
git checkout -b feature/[ticket-number]-[feature-name]

*Note: If the feature is a epic feature, the source branch is the epic branch.*

1. **Make code changes locally in feature branch**

At this point to have created a feature branch locally and can start making code changes. Once you are satisfied with your code, you can commit it locally. Please include the ticket number at the start of your commit messages:

git commit -m "[ticket-number] Description of commit"

1. **As needed (daily procedure): Interactive rebase from master (or epic branch) before to publish**

Before publishing your branch for code review and as otherwise needed, merge any changes from master (or epic) into your feature branch to make sure it is up to date. The rebase command will remove your commits, merge changes from master, and then reapply your commits on top of the merged code. If there are any merge conflicts, you will need to address them before moving on.

You can rebase as many times as needed. It is easier to rebase before your feature branch is published.

git checkout master  
git fetch origin master  
git reset --hard origin/master  
git checkout feature/[ticket-number]-[feature-name]  
git rebase master

1. **Publish feature branch**

Publishing your feature branch will create a remote version of your branch. This allows other users to see/checkout your branch so that they can test and code review it.

git push origin feature/[ticket-number]-[feature-name]

1. **Before review and merging to master (or epic): rebase from master and squash commits**

Once the feature is complete, you can then merge (only using a pull request) it back into master (or epic branch).

git checkout master  
git fetch origin master  
git reset --hard origin/master  
git checkout feature/[ticket-number]-[feature-name]  
git rebase -i master #pick or reword the first line, fixup the rest  
git push -f origin feature/[ticket-number]-[feature-name]

1. **Merge to master (or epic)**

Once your branch is rebased, you're ready to merge your finished code only using a [Pull Request](https://docs.microsoft.com/en-us/azure/devops/repos/git/pull-requests-overview?view=azure-devops) .

1. **Create release branch**

When all features for a specific release are in state Done, we can generate a release branch for that sprint.

git checkout master  
git fetch origin master  
git reset --hard origin/master  
git checkout -b release/v[number-of-version]  
git push origin release/v[number-of-version]

1. **Merge to prod**

git checkout prod  
git fetch origin prod  
git reset --hard origin/prod  
git merge release/v[number-of-version]