Development Processes

> **Stop!**
>
> First review Development Environment to setup your environment and get a working clone our main repository. if you already have an environment with branches from the Github repository, please reference Migration to CI Process.

> **Stop Again!**
>
> Before reviewing this documentation, familiarize yourself with the Branch and Release Workflow of the repository. This may look intimidating at first, but a developer's day to day interactions with the repository is actually rather limited and contained in their isolated development branches which will be discussed in this document. It's just good to know the big picture.

Development Process

A developer's first task is to make sure that the issue has been identified by their scrum master for a specific release and is in the current sprint. The associated JIRA ticket will clearly identify the targeted release in the "fixVersion" field. The targeted release determines how the branch is created for making your fixes. The developer is responsible for frequently rebasing from the targeted integration branch to ensure (s)he has the latest changes, which will help avoid merge conflicts or bugs. Here is an overview:

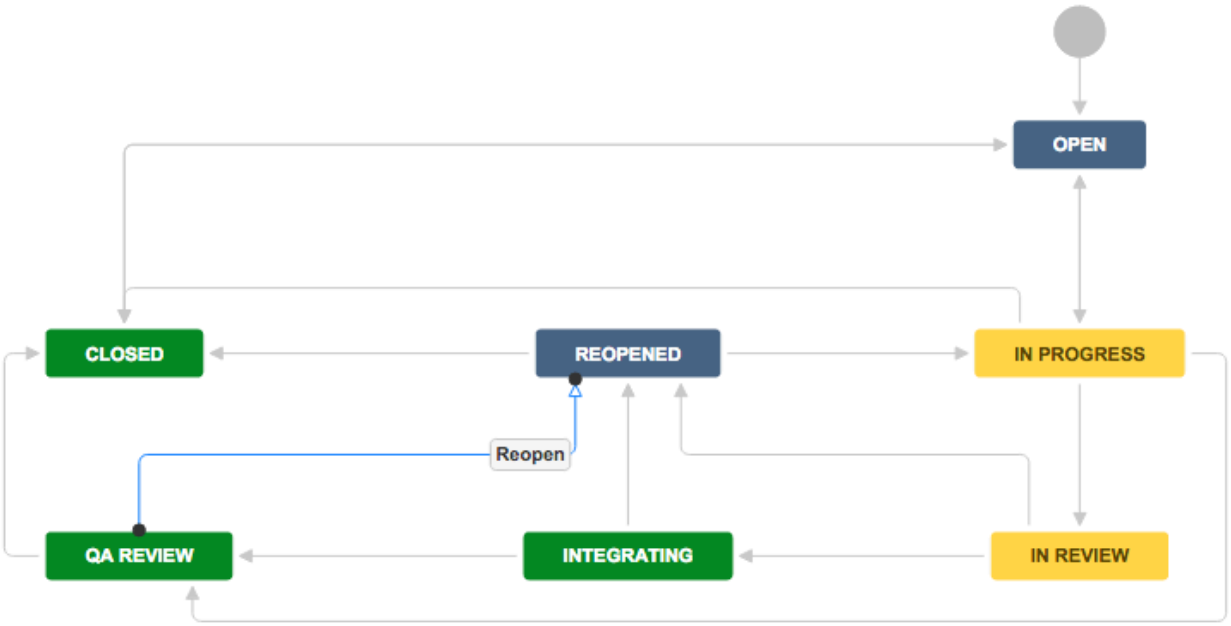
- **HotFix**: When the fixVersion is a HotFix (e.g. Joule 2.9 MP1 HF2), create a **fix/INT-X** branch off of the stable release branch (release/2.9+1).
- **Maintenance Pack**: When the fixVersion is a Maintenance Pack (e.g. Joule 2.9 MP2), create an **update/INT-X** branch off the current unreleased maintenance pack branch (e.g. release/2.9+2).
- **Major Release**: When the fixVersion is a Major Release (e.g. Joule 3.0), create a **feature/INT-X** branch off of **master**. Closer to the major release, master will be branched to a release branch (e.g. release/3.0) for release hardening.

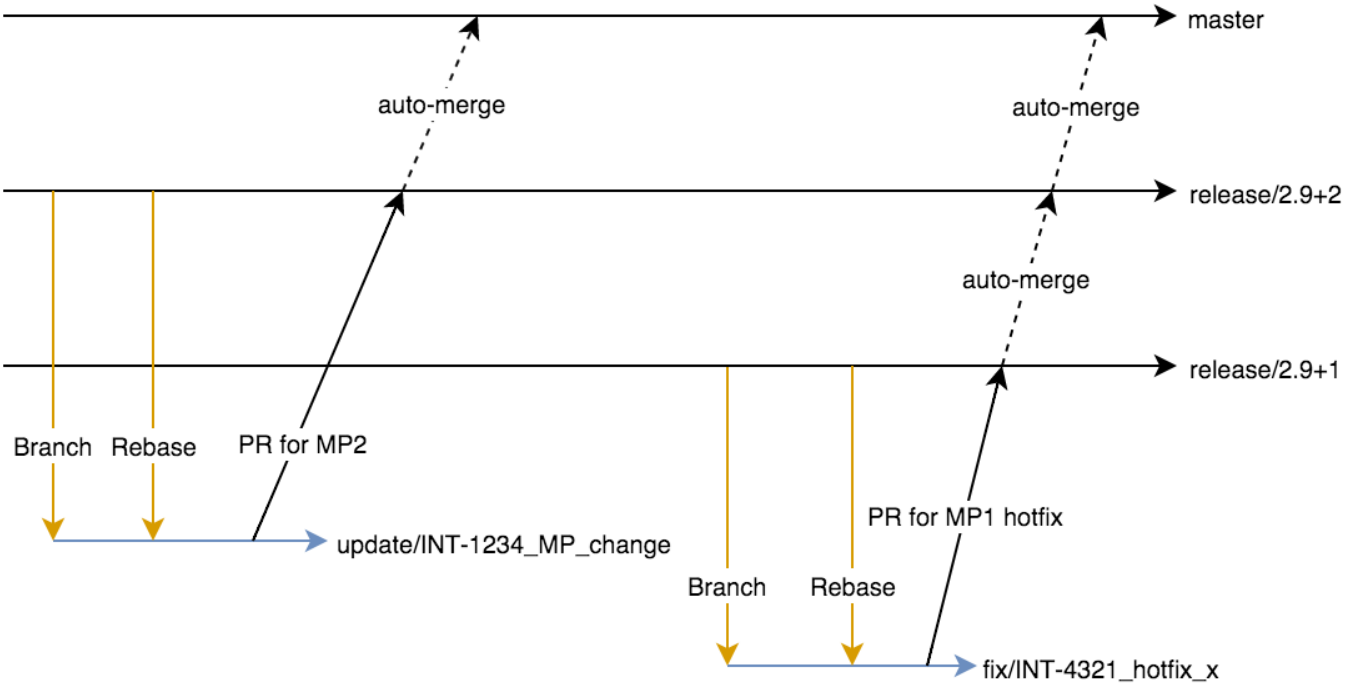
For the purposes of this documentation, a Maintenance Pack Fix will be demonstrated, though any variations in the other workflows will be documented. Here is a quick summary of what will be discussed:

1. **Open**
2. **In Progress**
    1. Best practice information on Continuous Integration Test Coverage and how we approach test driven development.
3. **In Review –** this state allows for the change to be reviewed in multiple ways. See further down about each review item, but quick overview is the following review items:
    1. Code Review
    2. QA readiness
    3. Prototype Review
4. **Integrating –** issues whose pull request have been approved. At this point tickets will have a "Resolution" set.
5. **QA Review –** this is integration testing and acceptance testing.

6. **Closed**
7. **Reopened –** has been sent back from In Review, QA Review, Integrating or Closed.

For the visually inclined, there are some workflow images that you can use to follow along:





JIRA Status: Open

This is the default status for newly created issues. This status signifies that work hasn't started. Before starting the issue, verify that the scrum master has set a fix version for the issue and that the issue has been added to the current sprint.

When you are ready to start working on this issue, flip the JIRA issue's status to **In Progress**.

Moodle Core and 3rd party Tickets

We work in an open source world, and sometimes you'll be working on a ticket whose solution relies on a different maintainer. These types of tickets are generally marked with a component of either "Moodle Core" or "3rd Party Plugins". If the ticket does not link to a corresponding patch or Moodle Tracker ticket, we'll want to ensure that the ticket has been reported to the external tracker as there's a possibility that there may already be a solution, or development in progress, by the maintainer. Developers should check the external tracking system if applicable, particularly when the ticket is for Moodle Core. In general, tickets that have been reported by the Defect Management team (i.e. have a Support URL) will already have checked the external tracker, although not every 3rd Party Plugin has a public tracking system. Tickets that have been reported by Quality Assurance, which will usually be housed under a generalized "Core X.Y Known Issues" ticket, will not have checked an external tracker.

JIRA Status: In Progress

Step 1: Branch Creation

Firstly, you must create a branch that'll contain your work for the issue. Depending on the fix version, the name of your branch may vary:

- **HotFix**: **fix/INT-X** branched off of **current production release branch** (EG: release/2.9+1)
- **Maintenance Pack**: **update/INT-X** branched off of **current unreleased release branch** (EG: release/2.9+2)
- **Major Release**: **feature/INT-X** branched off of **master**

The reason why the branches have different names is so that one can easily identify the purpose of the branch and what must be done with it.

Now, let's make the branch! Let's imagine the following:

- You're working on JIRA issue **INT-123**
- The issue has to do with adding **Feature Foo to Bar**
- The issue has a fix version for the next **Maintenance Pack**

Therefore, you would want to make an "update/INT-123" branch to store your work. Create that branch now:

```
# First, SSH into your virtual machine (optional if you have KOW
setup on your host computer)
> vagrant ssh
# If you want to know more about the checkout command
> kow help checkout
# Perform the actual checkout
> kow checkout INT-123
```

Due to how ticket integration and the build system works, you must make a new branch for every change. This means if your ticket is Reopened after already being integrated, then it is best to create a new branch name instead of re-using your old branch name which was already integrated. The recommended **kow checkout** command will do this for you as it randomizes the branch name on creation.

Step 2: Committing Changes to Your Branch

Before you continue...

- If you are making a *patch to core*, follow the step in the Submitting Patches to Core Moodle documentation. Once done, you can skip to *Step 3* unless you have more changes to make.
- If you are adding or updating a *3rd Party Plugin*, follow the step in the 3rd Party Plugin Management documentation. Once done, you can skip to *Step 3* unless you have more changes to make.

Let's say time has passed and you are ready to commit your changes:

```
# Do git status and diff to see your changes
> git status
> git diff
# Everything looks good, then add all changes to the index, this
readies them for commit
> git add .
# Make sure things still look OK
> git status
# Commit the changes
> git commit -m "INT-123: Added Foo to Bar"
```

Repeat the above steps until the work has been completed.

Step 3: Publishing Your Branch

Now your work is all done, it is time to publish your changes out to the repository so it can be reviewed. Do the following:

```
# This will push your currently checked out branch
> kow push
# As always, if you want to know more about this command, use help
> kow help push
# Pro tip: if you want to push your branch AND make a pull request,
do the following:
> kow push -p
```

Step 4: Filing Pull Requests for Your Branch

Once your branch has been published, you must file your pull request in Bitbucket for your branch. The pull request should be from your branch to the appropriate integration branch.

```
# You can create pull request for any branch with the following,
replacing YOUR_BRANCH_NAME with your branch, EG: update/INT-
123_FooBar
> kow pullrequest YOUR_BRANCH_NAME
```

Step 5: Change status to In Review

Once that has been completed, click "Start Review" button to put the JIRA ticket to **In Review** status. This is a good time to ensure all the fields in your JIRA ticket are up-to-date. The JIRA screen for moving to In Review has the most relevant fields displayed. For example, if the resolution to the ticket is actually different from how the ticket is described, then update the subject and description of the ticket. An example of this is when the bug is reported, it was actually misdiagnosed and the real problem was with something else.

Some things you may want to do before submitting for review

- Check PHPUnit and Behat tests pass locally.
- Run unit tests with code coverage. Ideally 100% code coverage is attained for the relevant files.
- Run inspection tools against your code - PHP mess detector, Moodle code style checker, etc.
- Manually check your diff and review it yourself.
- Familiarize yourself with the Continuous Build System and what it can do for you.
- If the branch contains a new plugin or plugins ensure that it can be control panel enabled/disabled. If it can be then add an OPS ticket so that it can be added to the control panel. See the code review checklist for a list of what should be included. If it can not be control panel enabled/disabled confer with another developer for possible solutions.
- If the pull request includes new or changed text on page (i.e. lang string changes) create a technical subtask called "Text on page" and assign it to Tam my Secord. In the description list the changes to the strings that are being made and attach a screenshot of where the text appears in the product. Tammy will review the changes and make comments/suggestions. When she has closed the subtask the parent ticket can be moved along in the workflow.

JIRA Status: In Review

In Review is when your change is reviewed by your team members in multiple ways. You may be assigned as the code reviewer or just voluntarily code review issues. Open pull requests can viewed in the Bitbucket repository and the Bitbucket UI should be used to add comments to the pull request. The following sub-sections describe the various reviews that take place in this status:

Code Review:

1. Must have two code reviewers approve the pull request. For now, Product Dev will be the second code reviewer for Client Dev.
2. Each code reviewer must complete a code review checklist. The checklist is just a list of reminders and sanity checks to perform during the code review.
3. Tasks can be created on pull requests in Bitbucket by the reviewer or the developer to track items that need to be fixed to pass code review.

QA readiness:

QA uses this state to confirm multiple things prior to the fix/feature being integrated. Once QA readiness is complete, QA adds the "qa-approved" label to the ticket. Things that QA will verify:

1. They can replicate an issue
2. Automated test coverage sufficiently covers all scenarios
3. Testing instructions are complete
4. Testing scenarios are complete

Prototype Review:

Prototype Review - this is an optional step that may be used to review a change before committing to an integration branch. Reasons for doing this:

1. Product Development may use it for prototyping or for early feedback on a feature. This is when a simple screen sharing doesn't work of course.
2. Product Development may use it for testing of risky or large changes. Thinking of sweeping changes or something like upgrading to latest version of Moodle and allow QA to smoke test it.
3. Client Development may use it for client UAT. A "clientdev-qa" prototype branch has been created for convenience, and Client Dev test servers can be auto-deployed from this branch. Note that this branch does not get the benefits of any auto-merging, so treat it as a feature branch that must be rebased.

If the code review fails, then the assigned developer sets the issue to **In Progress** status. The assignee of the issue will then follow the Development Process steps of the **In Progress** status.
If the code review passes, then the original developer clicks the "Ready to Integrate" button and selects the applicable Resolution to set the ticket to **Integrating** status. Warning: once you move your issue to **Integrating** status, it will be merged in the release branches, so only do this when you are truly ready.

JIRA Status: Integrating

You don't need to do anything when your issue is in this status. In this status, an automated tool will verify your issue and your pull request. If verification passes, then the tool will merge the pull request and move the issue to **QA Review**. If verification fails, then the tool will **Reopen** the issue and comment about why the issue could not be integrated. Please note that after your branch has been integrated, **it will be deleted**. You will need to **make a new branch** if QA finds problems with your changes.

The validation performed during integration is as follows:

- The issue must have an **open** pull request in the **Integration Pull Requests** field.
- The issue must be in the **Integrating** status.
- The issue must have only one active fix version.
- The issue must have a **qa-approved** label. This is added by *QA only*.
- For Hot Fixes: the issue must have a **hf-approved** label. This is added by *managers and integrators only*.
- For Hot Fixes: the pull request must not contain any modifications to **version.php** files.

- For Hot Fixes: the pull request must include **PHPUnit or Behat tests** to verify the fix. If tests are not possible (EG: core backport, 3rd party plugin update), this can be bypassed by *managers or integrators*.
- If the issue is in the Product Development project, then the issue must belong to an **active sprint**.
- The pull request must not be conflicted.
- The pull request must not have a merge veto. Merge vetoes can include not having a successful build or not having two approvers.
- The pull request must belong to the **DEV/moodle** project.
- The pull request must be merging into the correct integration branch based on the fix version.
- The pull request's assigned reviewers must all approve the pull request. This is required even if there are already two other approvers.
- The pull request cannot have more than **20** commits.

JIRA Status: QA Review

Once the automated integration tool integrates a pull request, it sets the JIRA ticket to **QA Review**. The site that has the code for QA review depends on the fix version of the issue:

- **HotFix or Maintenance Pack**: http://qa2-sandbox.mrooms.net/
- **Major Release** and **Phase 3**: http://qa2-next-sandbox.mrooms.net/

If a client is waiting to perform acceptance testing, then now is the time to allow them to do this. This will be on a test server auto-deployed from the current release branch.

If the QA review fails, then create QA Sub-Tasks for each failure. Then click "Reopen" set the issue to **Reopened** st atus.
If the QA review passes, then close the issue to set the ticket to **Closed** status.

JIRA Status: Closed

This means that the work has been completed for this issue. Only re-open the issue if something has gone wrong with the issue itself or if one of the steps was not completed. **Do not re-open the issue** if the issue has caused regression, instead, file a new issue.

Other Processes

Deleting your branches

Please see Deleting Stale Branches.

Switching from your current task to another

Sometimes when you are working on something, a bug or an emergency will come up and you have to switch tasks. There are a couple of ways to handle this:

- Commit your unfinished work (Easy, but might be undesirable in its current state)
- Use git stash (A GUI might be helpful for handling this):
  - http://book.git-scm.com/4_stashing.html
  - http://progit.org/book/ch6-3.html

Once your working branch has a clean status, you can re-start the development process for the new task. Once the new task has been completed, switch back to your old task by checking out your branch (git checkout branchname) and if you used git stash, then apply your stash. Then continue to work.