Development guide for Mantid-fork@ORNL

Overview

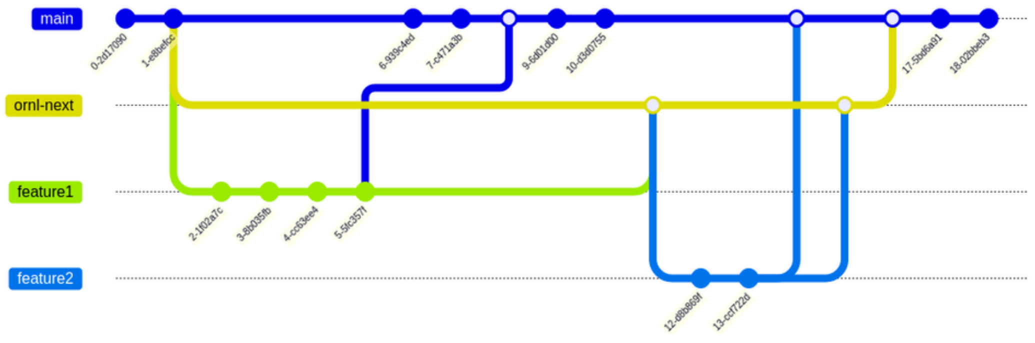
The purpose of the ORNL fork of mantid is to decrease the time between work being done and it being deployed to users, without overly increasing the amount of testing required. This is achieved by having a fork of mantid that includes only changes made (or approved) by ORNL staff. The fork will be built/tested fortnightly and released as a stable release once CIS agree that it is ready. At a high level, the workflow is

1. Developers do work based off of the `ornl-next` branch that gets reviewed/approved in a PR targeting `main`, then brought into `ornl-next` via a separate PR.
2. Every fortnight a release candidate is tagged on `ornl-qa`.
3. CIS have a week to test and either accept or reject the release candidate.
4. If the release candidate is accepted, a full release is tagged on `ornl-qa`. Rejected release candidates get bug fixes on `ornl-next` and wait until the next release candidate.
5. To prevent divergence with upstream mantid development, `release-next` branch of mantid is merged into `ornl-next` every night. In practice, this means no changes until mantid’s code freeze in preparation for their next release.

Links

- ornl-next: GitHub - mantidproject/mantid at ornl-next
- ornl-qa: GitHub - mantidproject/mantid at ornl-qa
- ornl-stable: GitHub - mantidproject/mantid at ornl
- Anaconda channel: Package repository for mantid-ornl :: Anaconda.org
- Jenkin’s package build pipeline: https://builds.mantidproject.org/job/build_packages_from_branch/build

Workflow guideline for developers



For the general developer in ordinary times, the workflow is

1. Update local checkout or `ornl-next` and base new work off of that branch
2. Make changes
3. Create a PR targeting `main` which should follow mantid's normal git workflow except for which branch the work was started from.
4. Create a **second PR** targeting `ornl-next`. One should take care that only intended changes are included in the branch. This may require a second branch being created if conflicts with main arose during development. See below for tips on how to resolve such issues. This PR should not get labels or milestone set and should contain references to PRs into `main` that it is targeting. **NOTE:** it is recommended to go through the entire process for the PR into main including getting the PR merged, before creating the second PR into `ornl-next`.
5. The PR targeting `ornl-next` can normally be marked "auto-merge" immediately. The developer should check back to verify that all requirements passed and that the merge did actually happen. There is no review on the second PR because the PR into `main` already had that validation.

This technique will generate sibling PRs, one into `main` and one into `ornl-next`.

When everything is going well, the two PRs can be created from a single branch. Example PR into main and PR into ornl-next. The only real pitfall of this approach is that github will annotate *all* PRs with the build status for the commit that last posted.

For EWM: Both branches should be linked in the description of the EWM item. The final task on an EWM item should not be closed until all PRs are merged into their respective branches.

For branch into `main`: have a link back to the EWM item for reference.

For branch into `ornl-next`: text similar to "This is a version of #36474 into ornl-next" is sufficient.

Hints for merge conflicts

For more interesting things, the branch for the PR into `ornl-next` will need to be created independently of the one into `main`. Example PR into main and PR into ornl-next. In this case, cppcheck was updated and the configuration was changed. These changes were copied over into a new branch based off of `ornl-next` and the PR was created. There are three main ways to copy changes from other branches that can be employed:

- `git checkout origin/<branchname> <list of files>` will get the current state of `<list of files>` from `<branchname>`. Results of this will still need to be committed.
- `git cherry-pick <refspec>` will attempt to apply individual changesets to the current branch. By default this will not pull in merge commits, which probably shouldn't be copied over anyhow.
- Hand-edit files to represent the changes from another branch. This is necessary when the changes are within a context that is different between branches. Normally this solution indicates that the branches have diverged and more changes need to be pulled from `ornl-next`.

Workflow guideline for computational instrument scientist (CIS)

In general, the development team will keep track of which channel and version of mantid is necessary for the software being deployed. However, one can view what version of mantid is installed in an environment by activating the environment, by name, using `nsd-app-wrap.sh <conda-env-name>` then running

```
conda list | grep mantid
```

The result will give the version number and the channel that the packages were installed from.

Separately, the following conda environments will be modified to point at the ornl fork

- `mantid-dev` conda environment (invoked as `mantidworkbenchnightly`) will be `mantid-ornl/label/nightly`. In principle, this will be updated every night.

- `mantid-qa` conda environment (invoked as `mantidworkbenchqa`) will be `mantid-ornl/label/rc`. In principle, this will be updated every fortnight.
- `mantid` conda environment (invoked as `mantidworkbench`) will be `mantid-ornl/label/main` which can be specified simply as `mantid`. This will be updated one week after an accepted release candidate.

Workflow guideline for devops

With this plan, the main issue that will arise for deployment is that the wrong anaconda channel will be pointed at. It is very likely that changing channels will get the desired version from anaconda. Remember that the order the channels are listed in will change dependency resolution. The two channels are

- mantid :: Anaconda.org - mantid's conda channel
- mantid-ornl :: Anaconda.org - conda channel for the ORNL fork of mantid

Publishing conda packages to channel "mantid-ornl"

By default, the branch `ornl-next` is automatically built and published to `mantid-ornl` channel at a nightly basis. When there are no changes to `ornl-next`, nothing will be run. However, the QA branch, `ornl-qa` and stable branch `ornl` requires manual built after a release candidate or a formal release is confirmed. The instructions below only apply to `ornl-qa` and `ornl`.

Creating a release candidate

Publishing from branch `ornl-qa` requires the creation of a release candidate tag. The convention is to append suffix `rcX`, where `X` is a number (e.g `v6.8.0.2rc2` for the second release candidate to future version `v6.8.0.2`). Following with this example, the following git commands will appropriately create the tag:

```
git fetch --all --prune --prune-tags
git switch ornl-next
git rebase -v origin/ornl-next
git merge --no-edit origin/ornl-qa
git push origin ornl-next
git switch ornl-qa
git rebase -v origin/ornl-qa
git merge --ff-only origin/ornl-next
git tag v6.8.0.2rc2
git push origin --tags ornl-qa
```

Creating a tweak release

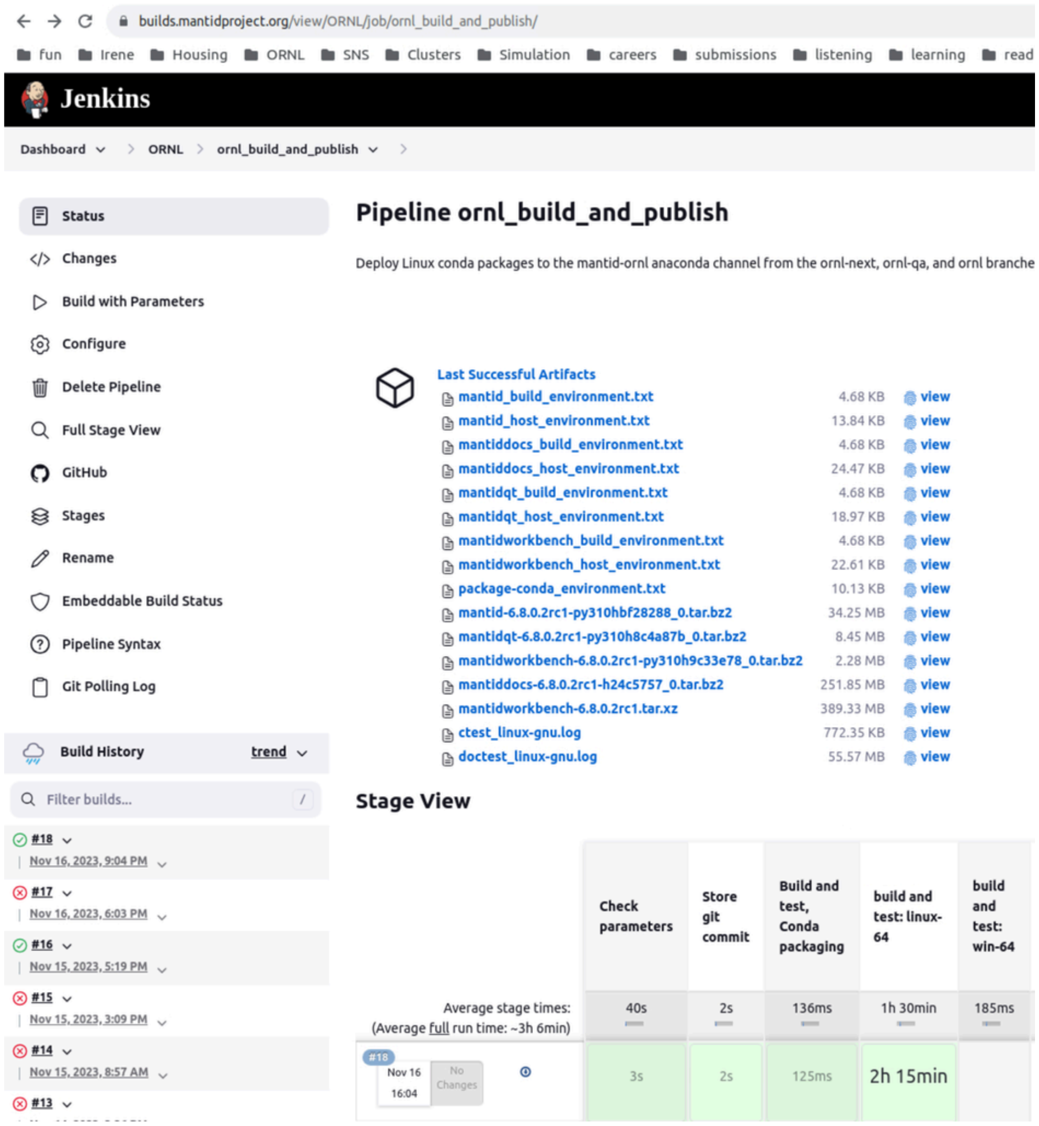
Publishing from branch `ornl` requires the creation of a tweak release, so called because we increase the tweak digit (**main**.*minor*.**patch**.*tweak*). For instance, version `v6.8.0.2` has tweak number *2*). Following with this example, the following git commands will appropriately create the tag:

```
git fetch --all --prune --prune-tags
git switch ornl-qa
git rebase -v origin/ornl-qa
git merge --no-edit origin/ornl
git push origin ornl-qa
git switch ornl
git rebase -v origin/ornl
git merge --ff-only origin/ornl-qa
git tag v6.8.0.2
git push origin --tags ornl
```

Creating and publishing conda packages

Creating and publishing conda packages from `ornl-next`, a release candidate tag, or a tweak release tag requires identical steps and almost identical parameter values.

Go to the `ornl_build_and_publish` page on Jenkins (here) and login



Pipeline ornl_build_and_publish

Deploy Linux conda packages to the mantid-ornl anaconda channel from the ornl-next, ornl-qa, and ornl branche

Last Successful Artifacts

mantid_build_environment.txt	4.68 KB	view
mantid_host_environment.txt	13.84 KB	view
mantiddocs_build_environment.txt	4.68 KB	view
mantiddocs_host_environment.txt	24.47 KB	view
mantidqt_build_environment.txt	4.68 KB	view
mantidqt_host_environment.txt	18.97 KB	view
mantidworkbench_build_environment.txt	4.68 KB	view
mantidworkbench_host_environment.txt	22.61 KB	view
package-conda_environment.txt	10.13 KB	view
mantid-6.8.0.2rc1-py310hbf28288_0.tar.bz2	34.25 MB	view
mantidqt-6.8.0.2rc1-py310h8c4a87b_0.tar.bz2	8.45 MB	view
mantidworkbench-6.8.0.2rc1-py310h9c33e78_0.tar.bz2	2.28 MB	view
mantiddocs-6.8.0.2rc1-h24c5757_0.tar.bz2	251.85 MB	view
mantidworkbench-6.8.0.2rc1.tar.xz	389.33 MB	view
ctest_linux-gnu.log	772.35 KB	view
doctest_linux-gnu.log	55.57 MB	view

Stage View

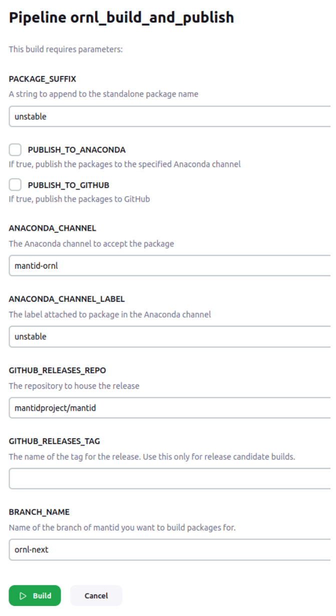
	Check parameters	Store git commit	Build and test, Conda packaging	build and test: linux-64	build and test: win-64
Average stage times: (Average full run time: ~3h 6min)	40s	2s	136ms	1h 30min	185ms
#18 Nov 16 16:04 No Changes	3s	2s	125ms	2h 15min	

Click option `Build with Parameters` (if not visible, contact to @Peter Peterson for access) to kick off the build configuration page (henceforth, any variable that isn't mentioned should stay with its default value).

- Select a package suffix via `PACKAGE_SUFFIC`.
  - Please use the default `unstable` for pipeline testing.
  - For actual publication, change it to empty string `""`
- Check `PUBLISH_TO_ANACONDA` when building `ornl-next`, `ornl-qa`, and `ornl`.
- **DO NOT** check `PUBLISH_TO_GITHUB`.
- Set `ANACONDA_CHANNEL` to `mantid-ornl`.
- Change `ANACONDA_CHANNEL_LABEL` to
  - if publishing from branch `ornl-next` → `nightly`
  - if publishing from a release candidate tag (e.g. `v6.8.0.2rc2`) or from branch `ornl-qa` -> `rc`
  - if publishing from tweak release tag (e.g. `v6.8.0.2`) or from branch `ornl` → `main`

    - `all other branches` → `unstable`
- Leave both `GITHUB_RELEASES_REPO` to its default state (`mantidproject/mantid`).
- Set `GITHUB_RELEASES_TAG` to:
    - if publishing from branch `ornl-next` → default value (empty string)
    - if publishing from a release candidate tag (e.g. `v6.8.0.2rc2`) → `v6.8.0.2rc2`
    - if publishing from a tweak release tag (e.g. `v6.8.0.2`) → `v6.8.0.2`
- Select the branch via `BRANCH_NAME`
- Click `Build` and watch to pipeline go through

After the pipeline completes successfully, check that the packages uploaded to mantid-ornl :: Anaconda.org



Pipeline ornl_build_and_publish

This build requires parameters:

PACKAGE_SUFFIX
A string to append to the standalone package name
unstable

☐ PUBLISH_TO_ANACONDA
If true, publish the packages to the specified Anaconda channel

☐ PUBLISH_TO_GITHUB
If true, publish the packages to GitHub

ANACONDA_CHANNEL
The Anaconda channel to accept the package
mantid-ornl

ANACONDA_CHANNEL_LABEL
The label attached to package in the Anaconda channel
unstable

GITHUB_RELEASES_REPO
The repository to house the release
mantidproject/mantid

GITHUB_RELEASES_TAG
The name of the tag for the release. Use this only for release candidate builds.

BRANCH_NAME
Name of the branch of mantid you want to build packages for.
ornl-next

Default view of `Build_with_parameters` under `ornl_build_and_publish`

git_ornl_fork_w...mermaid
02 Jan 2024, 07:28 PM

- Once the package is published to Anaconda, make sure to use a clean Conda environment to install the package from `mantid-ornl` channel to verify that it works on ORNL's analysis machines.
    - For instance, you can use `conda install -c conda-forge -c "mantid-ornl/label/nightly" mantidworkbench` to install the nightly version of `mantidworkench` via conda on analysis.
    - Make sure you have enough disk space in your Home directory (use `snsquota` to check your allocated disk space).

Generally speaking, it takes roughly about **3 hours** to finish all steps, and you should be able to find the packages on `mantid-ornl` Anaconda channel after all steps are complete.

If you are redirected back to the build page or see an error stating missing permission, please contact @Peter Peterson to acquire the necessary permission on Jenkins. Make sure to have your Github handle ready as we are using Github to authenticate on Mantid's Jenkins service platform.

Sometimes the build will fail for some random un-related reasons. In such situations, simply re-submit the build request on Jenkins and wait for the pipeline to finish. If the issue persists, either contact @Peter Peterson for a quick troubleshooting, or post the error messages on the Mantid's slack channel (#jenkin is a good place to start).