Guidelines for Committing

Guidelines for Committing Code to DSpace GitHub

All of the below guidelines are just *suggestions*. They are not meant to create additional "red tape" during the normal development process. If you find they are getting in the way, let us know, we can always change them! Development processes and best practices change over time, and these guidelines should be no different.

General Guidelines for Committing

1. **Use GitHub Issues to track your changes**
   - Ensure there is an issue describing the fix/improvement/feature in GitHub Issues (If this is a minor typo fix, or an obvious bug, you need not add an issue – use your judgment on whether this fix is worth tracking in an issue or not)
2. **Submit a Pull Request for your code changes**
   - First off, you should be doing the majority of your development in your own GitHub Fork. See Development with Git.
   - Generally speaking, code changes should come in via a Pull Request. This gives the rest of the Committers/Developers a chance to comment on the Pull Request.
   - If this is a very minor fix/change (typo / improved comments / obvious bug), you can bypass creating a Pull Request and commit it immediately. **This is an exception to the rule.**
3. **Code approval processes**
   - *General Rule of Thumb:* Committers are **trusted developers**. Use your best judgment on whether or not you feel a particular piece of code should be approved before committing/merging your Pull Request. The below guidelines are just suggestions and they may not apply to all scenarios:
   - If this is a small, obvious fix (typo / improved comments / obvious bug), just commit it immediately. This is the exception to all other rules – there's no reason to add a JIRA issue for every typo or obvious fix, and no reason to wait around for approval to commit it. 🙂
   - If this is a bug/issue fix, try to get approval for your Pull Request from at least one other Committer. If there is any doubt, feel free to run it by others or ask to have it added to the agenda of the next developers meeting.
   - If this is a new feature, generally speaking you need at least three Committers to approve (or bring it up in a developer meeting, if it's a larger change). The only reason we suggest to run it by more than one committer is that oftentimes others may be able to provide additional suggestions or configurations that may be worth investigating.
   - Please note that our Developer Voting Procedures actually state that at least three Committers should approve of all code changes, and none should disapprove. We tend to be lenient on minor code changes / bug fixes (as these tend to be non-controversial changes). But, new features should really get the approval of at least three Committers (and larger features should be discussed / voted on in a public forum, e.g. GitHub, Slack or email).
4. **Document your code early & often**
   - New features/improvements need to come with documentation, otherwise we won't be able to accept them. Keep this in mind while you are developing, and use the Wiki or GitHub to start your documentation early on (it'll make it easier on you in the long run).
5. **Once Approved, Merge your Pull Request in a timely manner**
   - Once your Pull Request is approved, you should work to merge it as soon as you can. This will avoid your Pull Request getting "stale" and ensure that any upcoming work can take it into account.
   - Before merging, please ensure that your code doesn't break any Unit or Integration Tests. This will save you a potential headache later, when our Continuous Integration system does a verification of all Unit Tests.
   - If the Pull Request was not created by a Committer, the Committer(s) who are most familiar with the work (and who may have been supporting/giving feedback) should merge the Pull Request.
   - Obviously, Committers can ask for someone else to take over the merge process as needed.

GitHub "Main Branch" Committing Rules

By "Main Branch" we are not only referring to DSpace/DSpace in GitHub. We're also referring to the Main branch of any other production-quality, out-of-the-box GitHub projects (e.g. DSpace/dspace-api-lang, etc.)

The GitHub Main branch is one of the few places that needs to remain well managed. As we all fork it and/or create Pull Requests from it, it can be detrimental to us all if incomplete or extremely buggy code is committed. Obviously, mistakes happen (and we've all made them), so don't worry if you accidentally commit/merge something you didn't mean to (just try to roll it back as soon as you notice it). Here are the few rules we try to follow when committing code to the Main branch:

1. **No broken builds on the Main Branch, ever.**
   - All commits / Pull Requests should be tested beforehand to ensure they will not "break" the Main Branch (this includes Unit Testing the code to ensure our Continuous Integration system doesn't report a unit test failure).
   - *A broken build is a matter of urgency.* Mistakes happen, but if you break the build, please make sure to fix it ASAP or temporarily rollback your changes until you can resolve the issues.
   - Obviously, there may be situations where temporarily breaking the build may be necessary. However, we should make all attempts to keep breakage to a minimum (or forewarn the Committer team if anticipated). Longer-term breakage (a few days or more) can negatively affect us all.
2. **No unapproved / incomplete features on the Main Branch.**
   - As mentioned, all your code / new feature development should take place in your own fork and submitted via a Pull Request once it is ready for feedback (see Development with Git). The only exception to this rule is trivial changes (e.g. typos / improved comments / obvious bug fixes).

- Your work should not be obviously missing functionality, or be only partially integrated, in a way that would make the system difficult to use in the near term or otherwise hamper other development efforts.
- If your code is known to be "incomplete" in any way, you must make this clear to the Committers and detail what work is still required.
- *Before your work can be merged, it must satisfy Rule #1 AND be approved for merging by the Committers* (See "code approval processes" note in the #General Guidelines for Committing above). During the "release window", the Release Team has the final say.
- Once something is merged to Main, we are effectively committed to it being in the next release.

3. **All new features must have documentation before merging into the Main Branch.**
    - New features/improvements should not be merged into the Main Branch until there is some minimal documentation (minimal documentation includes documenting all configuration options). Ideally, they should also come with usage documentation (i.e. examples of how to use the feature, how to configure the feature, etc.) This will help us all ensure that Documentation is ready by the time we get to the next release, and hopefully lessen the time that any one person has to spend cleaning up or rewriting documentation.

Guidelines for Security Fixes

1. Security vulnerabilities may be reported by users as detailed at DSpace Software Support Policy
    a. Once reported, further information may be gathered/discussed via email with the reporting user in order to verify the vulnerability
2. If a vulnerability is verified, Committers should use GitHub Security Advisories to draft an advisory & begin collaborating on a fix.
    a. Original reporter may be invited to collaborate on that advisory, see Adding a collaborator to a security advisory
    b. (Recommended) A private temporary fork repository may be created to develop the fix.
        i. *Note:* These private forks will NOT trigger our normal GitHub Actions/CI (as this is not yet supported by GitHub). So, **Unit Test & Integration Test impact will have to be verified manually by reviewers** (by running tests locally).
        ii. To collaborate in the private fork, you will need to either: clone that private repository, or add it as a new remote. The latter tends to be easier, and may be done as follows:

```
# Add the private repo as a new remote
git remote add DSpace-ghsa-[id] git@github.com:DSpace/DSpace-ghsa-[id].git
# Pull down that repo
git remote update
# Checkout the branch you are collaborating on
git checkout -b [local-branch-name] DSpace-ghsa-[id]/[remote-branch]
```

    c. Once the code is verified/approved, merge it. *NOTE:* This applies the changes directly to GitHub `main` which will make them public. So you may want to wait until the release is almost ready to do this.
    d. Submit the draft advisory to GiHub to request a CVE. The response should come back in a few hours or so, with an assigned CVE number (e.g. CVI-[year]-[num]). When assigned, a comment is added from "github-staff" with the details.
    e. When ready, publish the draft advisory. *NOTE:* This makes everything public, which means it usually should occur around the time of the release which fixes this vulnerability.