How to contribute or backport a fix

This page outlines the detailed recipe of what to do with a fix. It covers backports to jdk11u and jdk17u. Steps may or may not be valid for other update lines as well; please check.

Also, see The OpenJDK Developers' Guide, which covers all aspects of contributing to OpenJDK. It especially covers Backports and how to Backout a Backport. Also, please read how to Request push approvals for fixes.

There are two general types of fixes:

- *backports:* most commonly, a change for OpenJDK 11 updates is a backport of a change made in a higher OpenJDK version. Start the recipe from Step 1.
- *new fixes:* rarely, there is a need for a net new change for OpenJDK 11 updates, e.g., because a fix would not apply to higher OpenJDK versions that are in maintenance. Start the recipe from Step 3.

Important: The whole process is driven by the backport requester/contributor. Nothing here assumes that somebody else would do the work. If you are not an OpenJDK Author, you don't have a JBS user account, so you'll have to ask for help for steps 5 and 6 (working with JBS to put appropriate metadata). The regular place to ask for help is the JDK Updates mailing list. Regular contributors would eventually gain the necessary privileges to avoid this overhead.

OpenJDK 11u uses SKARA (Git) for backporting fixes from later JDK releases since June 2021. The old Mercurial-based workflow is no longer described. Please reach out to the JDK Updates mailing list in case you need help.

To prepare for contributing to one of the repositories (<your>/jdk11u-dev, openjdk-bots/jdk17u-dev, <your>/jdk17u etc), enable Pre-submit testing for each of the github repositories!

Contribution recipe:

1. Check the original JBS issue on https://bugs.openjdk.org/
    a. Carefully check linked issues and whether follow-up fixes need to be brought with the backport. See below how to handle fixes depending on each other.
    b. If relevant issues prevent a clean backport, consider backporting those first (within reason).
    c. To avoid others picking up the same issue, add a preliminary "Fix request 11u|17u" (see step 6.) comment saying that you are working on this. Once you reach step 6.), edit the comment and add the required information.
2. Create the backport commit:
    - Option 1 - Use the /backport comment command on GitHub:
        a. Make sure GitHub Actions is enabled for you on the OpenJDK Bots jdk11u-dev repository resp. OpenJDK Bots jdk17u-dev repository.
        b. Open the link of the original commit in GitHub and issue the /backport command by adding the comment: "**`/backport jdk11u-dev`**" or "**`/backport jdk17u-dev`**"
        c. For a clean backport, the bot will provide you with a branch in his own repository and a link to create a pull request. If the backport needs manual resolving, it will provide you with instructions, similar to option 3.
    - Option 2 - Use SKARA CLI tooling:
        a. Create a branch for your backport, e.g.,"**`git checkout -b my-backport-branch master`**"
        b. **`"git backport --from https://github.com/openjdk/jdk <commit-sha>"`**. See the SKARA Wiki for more info.
        c. If necessary, resolve changes and follow the instructions.
    - Option 3 - Use plain Git to create the change:
        a. Create a branch for your backport, e.g.,"**`git checkout -b my-backport-branch master`**"
        b. **`"git fetch --no-tags https://github.com/openjdk/jdk <commit-sha>"`**
        c. **`"git cherry-pick --no-commit <commit-sha>"`**
        d. If necessary, resolve changes.
        e. **`"git commit -a -m "Backport <commit sha>"`**
3. Test the patch: testing is very important. Your backport is very close to the customer, with few safety nets. In comparison, patches for the mainline head release get much more cooking time before seeing broad adoption. Don't rely on maintainers doing testing for you! You should know your patch best and must make sure it works and does not introduce regressions.
    a. "tier1" tests should pass at all times. Use **`"make run-test TEST=tier1"`** to run. You should ***test both debug- and release*** builds. Don't just test one of them!
        i. If your patch changes platform-dependent code, test your patch on as many of those platforms as you have available. If you cannot test every platform but feel that tests are needed, clearly state so in the PR or in the Fix Request. Maintainers then will strive to fill the testing holes for you.
    b. "tier2" provides a larger coverage if you have the resources to run it. Use **`"make run-test TEST=tier2"`** to run
    c. Run tests from the area that the patch affects, use **`"make run-test TEST=<path-to-tests>"`** to run specific tests
    d. New regression tests that come with the patch should pass
    e. Enabling GitHub Actions for your personal fork of the jdk11u-dev repository before publishing your branch will provide you with builds and a tier 1 test run via GitHub Actions on many platforms. If tests fail, analyze why they are failing and share this information.
4. Create a pull request at https://github.com/openjdk/jdk11u-dev | https://github.com/openjdk/jdk17u-dev
    a. If you have created a backport via Option1, the /backport command, and the backport was clean, you can use the provided link to create a PR
    b. In all other cases, push the new branch to your fork of https://github.com/openjdk/jdk11u-dev | https://github.com/openjdk/jdk17u-dev
    c. and open a PR. You can do this in one step via the SKARA command "**`git pr create --publish`**". If it is a backport, make sure the title of the PR is "**`Backport <SHA hash of original commit>`**" to have the bots correctly recognize your change as a backport.
5. If your patch is not a clean backport (labeled as **`clean`** by the bots), get the change reviewed by some jdk-updates reviewer
    a. **Note:** the change review *is not* the approval, which you would get at the next step
    b. The PR message is automatically posted to the jdk-updates-dev mailing list. You might optionally cc the original mailing list or other OpenJDK mailing lists to get some more attention to your PR by using the **`/label`** command.

c. In case of a backport, state in the PR description what changes were needed and why: the difference against the original patch, motivations for doing things differently, etc... The description is addressed to the reviewers who assess whether the change is correct for the update release.

Example PR message

```
Hi,

This is a backport of JDK-8888888: My Hovercraft Is Full Of Eels

Original patch does not apply cleanly to 11u, because eels are all different sizes
and shapes. Notably, I had to change the com/antioch/holy/Grenade.cpp to avoid API
that only exists in 12+.

Testing: x86_64 build, affected tests, tier1

Thanks,
-Monty
```

6. Request and await approval for the fix (See below for how to handle non-public issues). Only do this after finishing all previous steps: the PR should have been reviewed and tested.
    i. Add a *"Fix Request <java version>"* comment to the JBS issue that explains: why the fix should be backported, gives a risk estimate of introducing new errors, explains the dependencies on other backports (if any), shows what testing was done to verify the backport, etc. The "Fix Request" comment should give maintainers all information they need to make an informed decision for inclusion into the update release. Then put the **`jdk11u-fix-request`** or the **`jdk17u-fix-request`** label on the JBS issue. Now the JBS issue will appear in the filters used by the maintainers. The maintainers might remove the label if the issue is not ready to be decided upon. Add the label again if all preconditions are fulfilled.
    ii. Wait for maintainer approval or rejection, which will manifest as either **`jdkXXu-fix-yes`** or **`jdkXXu-fix-no`** label on the issue.

Example Fix Request comment with RFR

```
Fix Request 11u|17u

Backporting this patch eliminates the critical eel overflow.
The risk is medium. It changes the critical component xyz, where little changes sometimes
have unexpected effects. But this only touches abc and not the primary functionality of
xyz. Fixing the issue overweights the risk.
Patch does not apply cleanly to 11u and requires adjustments.
Backport requires JDK-8423421 and JDK-8771177 to be applied first.
Included test passes. Ran tier1 and tier2 and a big application to rule out secondary
effects.
```

Example Fix Request comment without RFR

```
Fix Request 11u|17u

Backporting this patch eliminates the eel overflow.
Low risk as this only touches tests.
Patch applies cleanly to 11u.
Backport requires follow up issue JDK-8282288.
New test fails without the product patch, and passes with it. Tier1 and tier2 tests pass
with the patch and 8282288.
```

In case of a larger change or backport, you might not want to invest the work for steps 1-5. and only then find out that the change is not accepted. In this case, you can add the "Fix Request" comment and label in advance or address the maintainers for advice in other ways.

What to do with changes depending on each other?

1. Make a backport of the first change. The pull request gets numbered <ID>, and a branch pr/<ID> is created.
2. Backport the second change on top of branch pr/<ID>. In the second pull request, compare against this branch.

After pushing the first pull request, the second will be retargeted to master. See also the description in this Mail.

What if the change needs a CSR?

1. From the original JBS issue, create a backport JBS issue ("More""Create Backport"). Target the backport issue to **`11-pool`**. This issue will be resolved when the change is pushed.
2. From that **11-pool** JBS issue, create a new CSR and copy/paste the contents from the original CSR. If the CSR is a straight copy of the original CSR, say so clearly in the issue text, otherwise point out differences. The new CSR should also have version **11-pool**.
3. Run the CSR through its process to get it approved.

1. What if the change you want to downport is not public?
    a. Sometimes a change you want to downport is not public in JBS. This means you can see the change and its JBS ID in the Git repository, but you will not find the corresponding issue in JBS.
    b. In such a case, you have to create the corresponding backport issue manually, according to the following recipe:
    c. Create a new issue in JBS with type "**Bug**" (you can't directly create an issue of type "**Backport**")
    d. The new issue should have exactly the same summary like the original change. You can take this from the Git "**summary**" line by stripping the prefixed Bug ID.
    e. It's helpful if the bug description contains a link to the original commit.
    f. Affected version should be one of "**8**", "**11**", ...
    g. Under "**Linked issues**" choose "**backport of**" and enter the bug id of the change you want to downport into the "**Issue**" filed (in the format "**JDK-XXXXXXX**"). After adding this link, it will be not visible. You can only verify that you added it by looking at the histrory.
    h. Once you've filled out all the other fields, press "**Create**" and once the issue has been created, edit the issue "**Type**" and change it from "**Bug**" to "**Backport**".
    i. Continue as usual at step 2.

2. If and only if everything is approved, push the change. This is done using the /integrate command of SKARA on the backport pull request. If you are not an jdk-updates project committer you'll need a committer to sponsor the PR for you.