Bitbucket Workflow for Tech Team

Contents

Last updated on: 11/19/2025

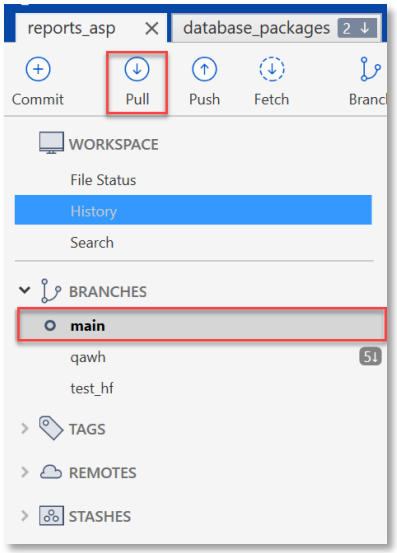
Workflow Summary

1. Start with main by checking out main branch.
2. Do a pull to sync local with remote.
3. Create a new branch and check out the new branch.
4. Stage and commit the changes in the new branch to local repo.
5. Push the local repo (with the new branch) to remote.
6. Create a pull request to merge the new branch into QAWH.
7. QA analyst reviews the changes in the new branch. When done, merge the changes to QAWH.
8. Create a pull request to merge the new branch into main.
9. QA analyst approves the pull request and merges the changes into main.
10. Delete the new branch from remote.

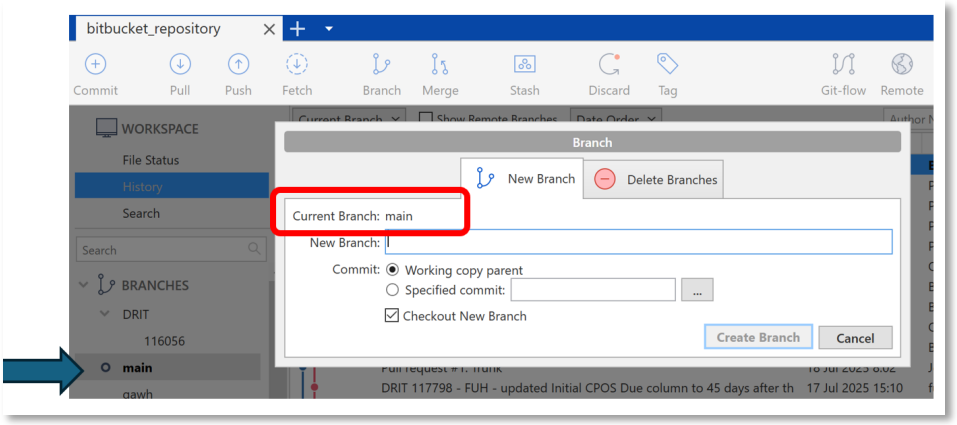
Note: Never merge qawh into main directly through pull requests because qawh contains a lot of changes in testing that should not be in production branch.

Workflow In-depth

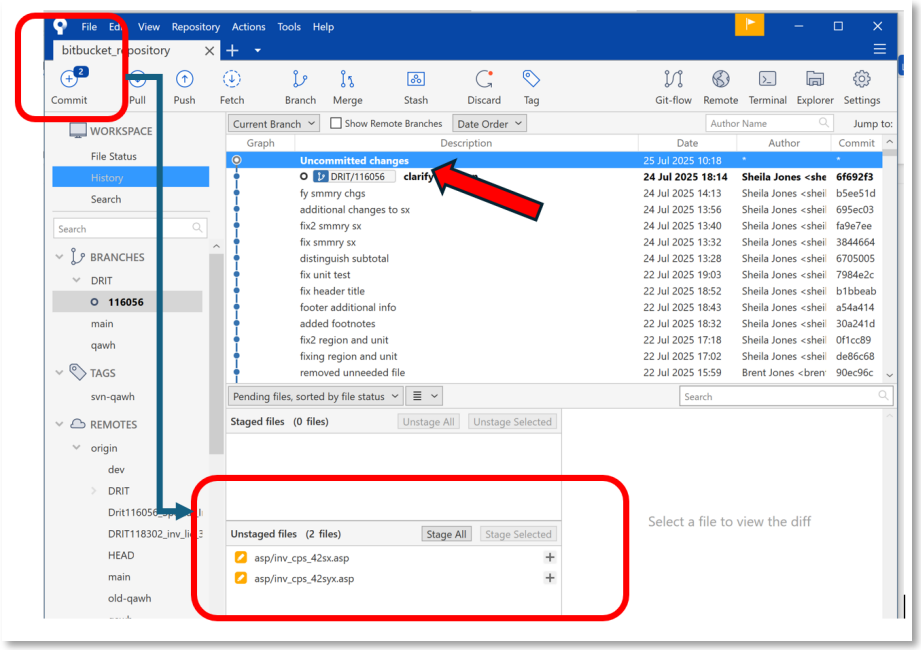
1. Double click on 'main' under **Branches**. This will make the main branch active (active branch is **bold**).
2. Then click on **Pull** in the tool bar to sync up with remote. **5↓** to the right of the branch name means that your local branch is 5 commits behind remote. Once you click on pull to pull changes from remote, it will merge in remote changes into your local branch. **5↓** should disappear and your local branch is now in sync with remote.
    a. However, if you see **5↑**, which means your local qawh/main branch is 5 commits ahead of remote, which should not happen. But if you see it, please refer to the section 'Local qawh/main branch ahead of remote'.



3. Click on Branch icon on the tool bar. Type in your desired branch name and make sure **checkout new branch** is selected and Current Branch is main. Checking out just means that you want to make that new branch active.

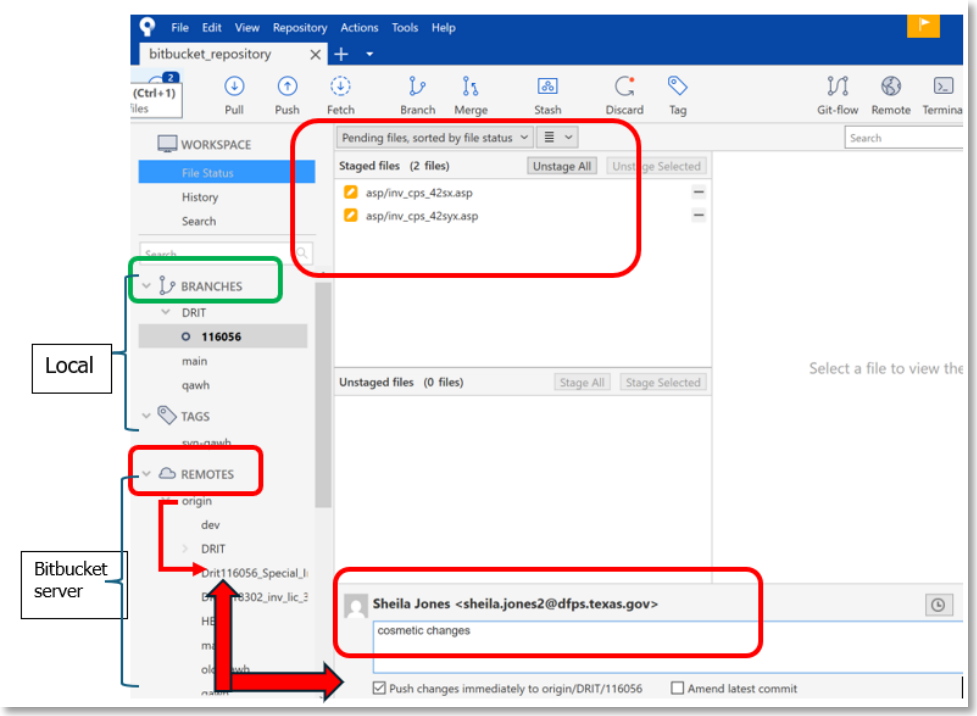


4. Now that your new branch is active, you can edit the code in your favorite code editor and test it in the appropriate environment. As soon as you make an edit, it is reflected in Sourcetree as '**Uncommitted changes**'. To stage these changes, Click the plus signs next to each file you are ready to stage. They will be moved to the "Staged files" area.



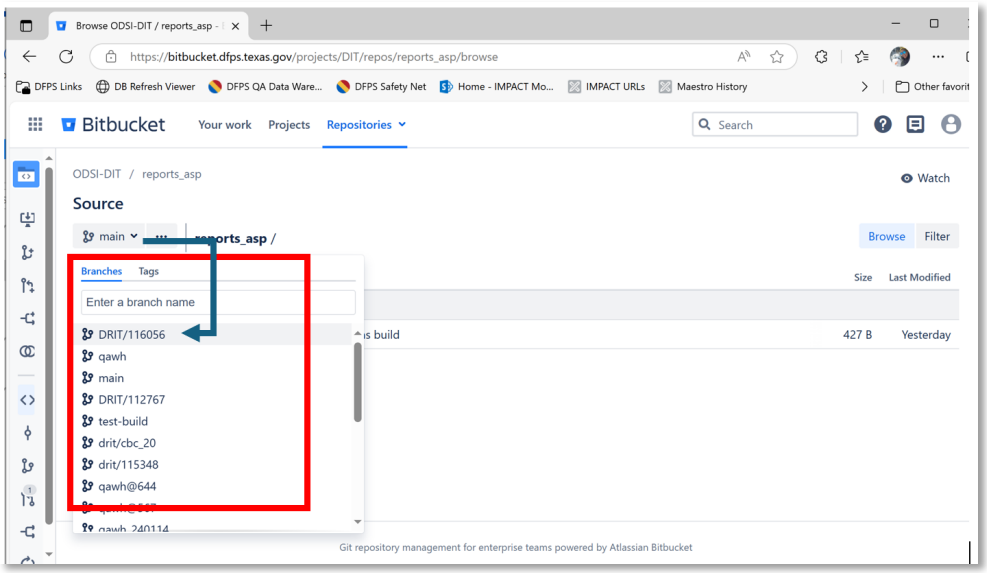
5. Once the changes are staged, click the "Commit" icon on the tool bar. Enter the comments in the box at the bottom of screen. If you want to push the changes

immediately to remote, make sure the checkbox ‘Push changes immediately to origin/DRIT/xxx’ is checked.

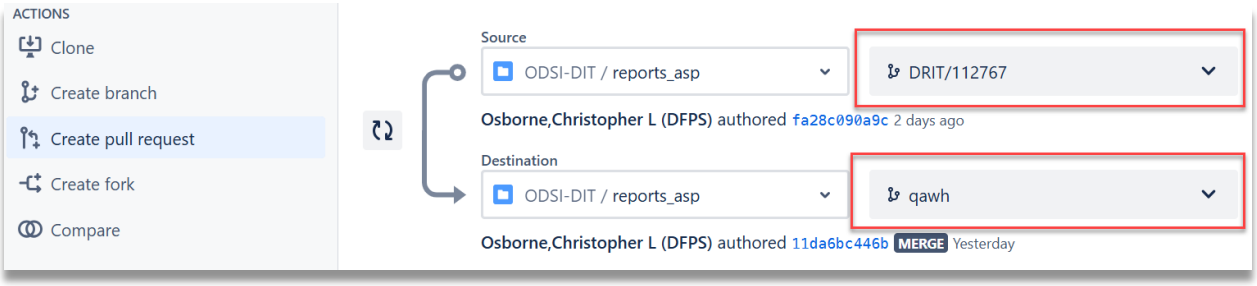


Alternatively, you can opt to push the changes from multiple commits in one step later. If that’s the case, uncheck the checkbox. Then continue to make commits without checking that box. When you are ready to push all your changes to remote, click on ‘Push’ on the tool bar.

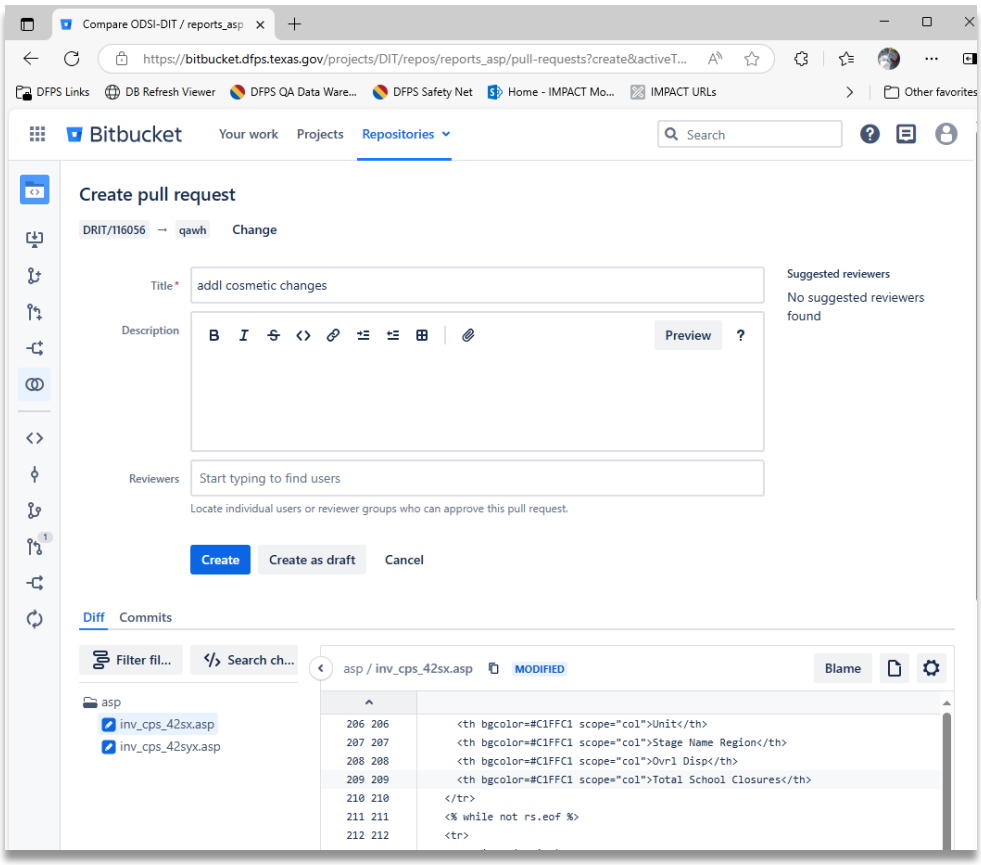
6. Go to the repo on bitbucket. Under “Source” it will most likely will show “main”. Click the dropdown to select your new Branch.



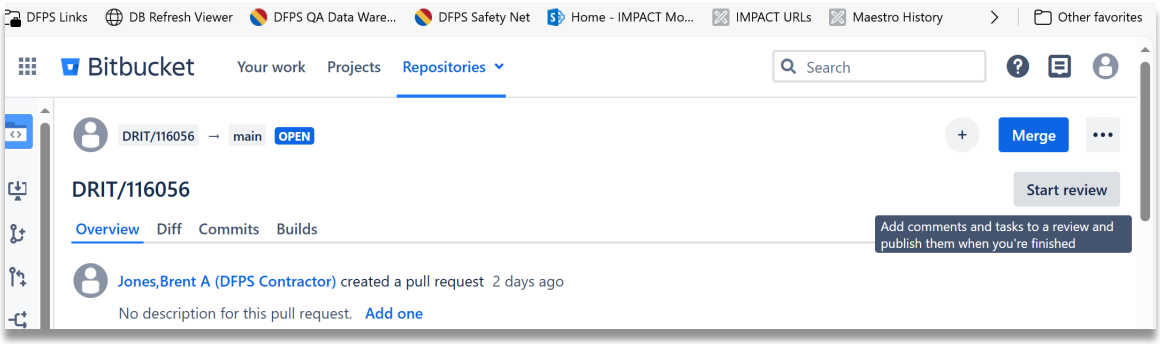
Click on **create a pull request** and select the source (your branch) and destination (qawh branch).



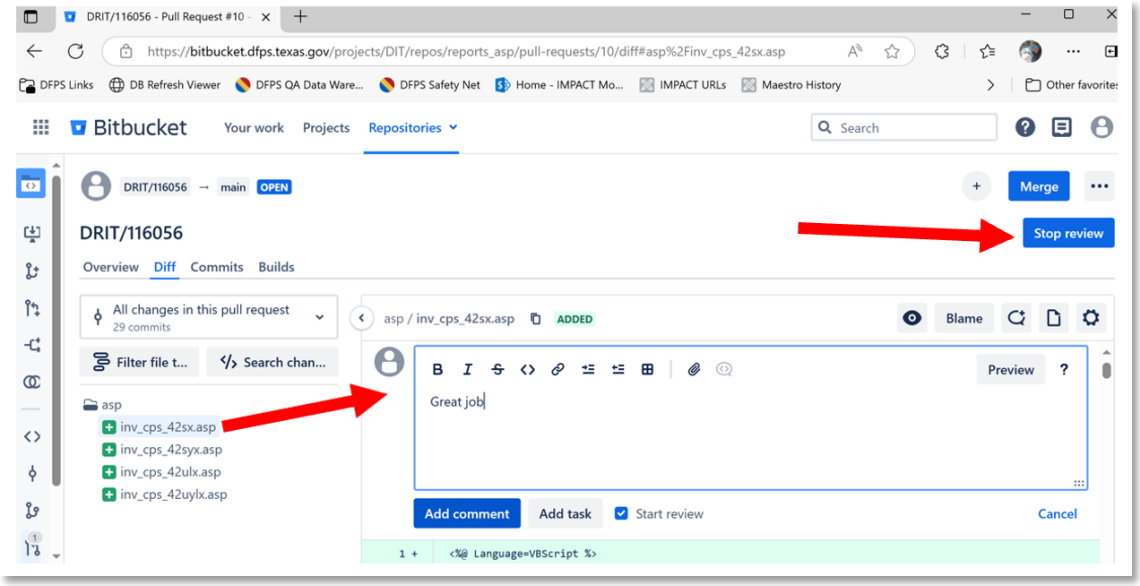
Enter the title and description for the pull request. Make sure that you add in the description any information that may be helpful for the QA so that he or she can hit the ground running. Type in the reviewer and then click Create. The QA will receive an email notification about the pull request.



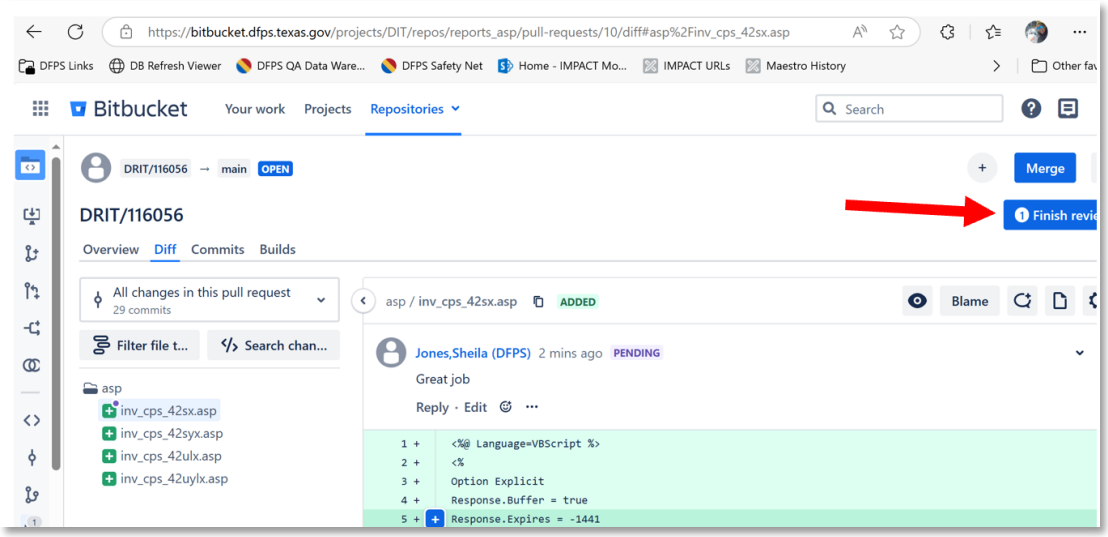
7. Once the reviewer logs into bitbucket, they can click **Start Review**



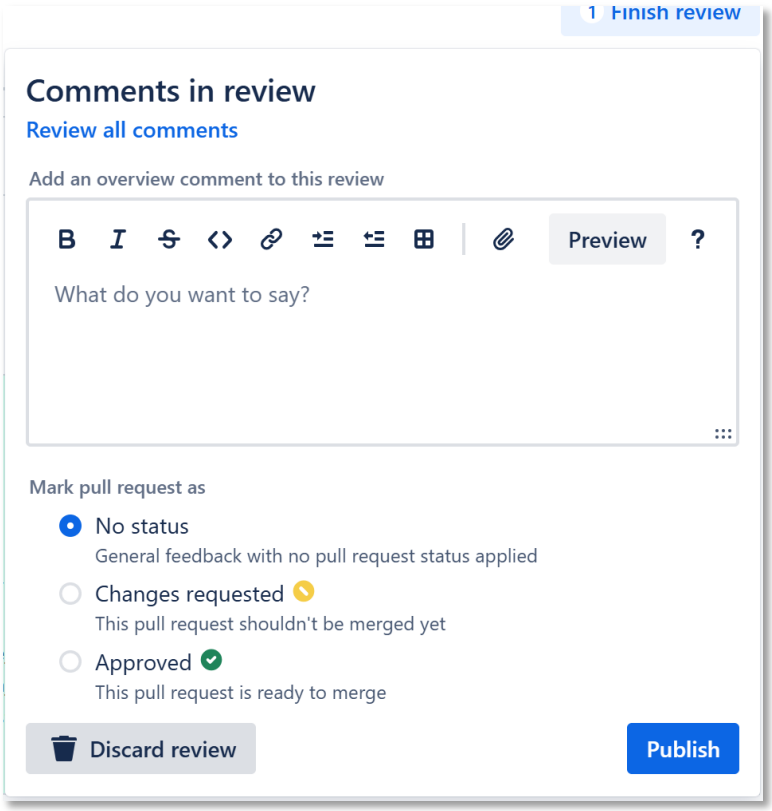
The reviewer can click **Diff** to see the changes and add any comments or tasks on the code they are reviewing. In fact, the reviewer can add comments or tasks at specific locations of the code by clicking on the plus sign with a blue background ( ) next to changed code. When the reviewer is done, they click **Stop review**.



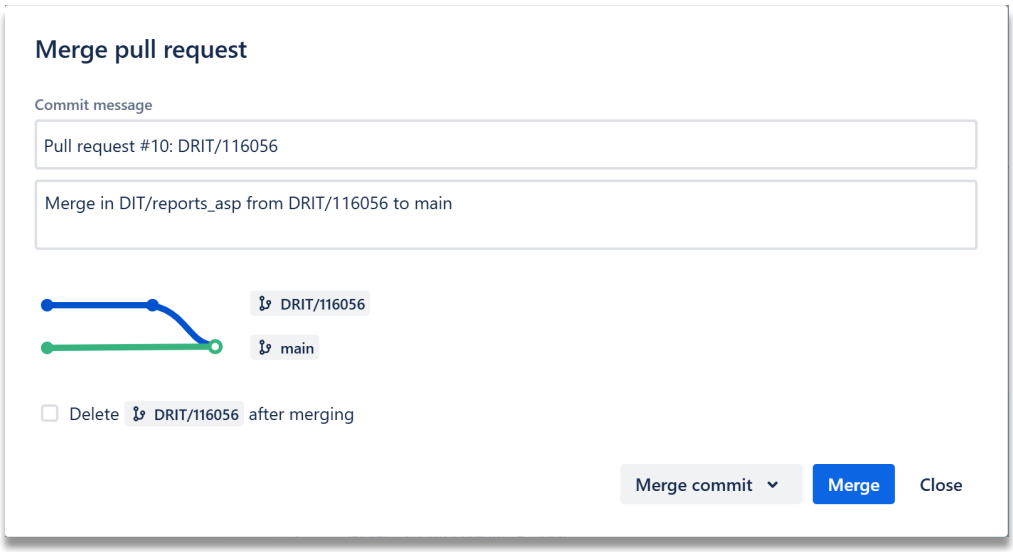
Then the Reviewer can click **Finish Review.**



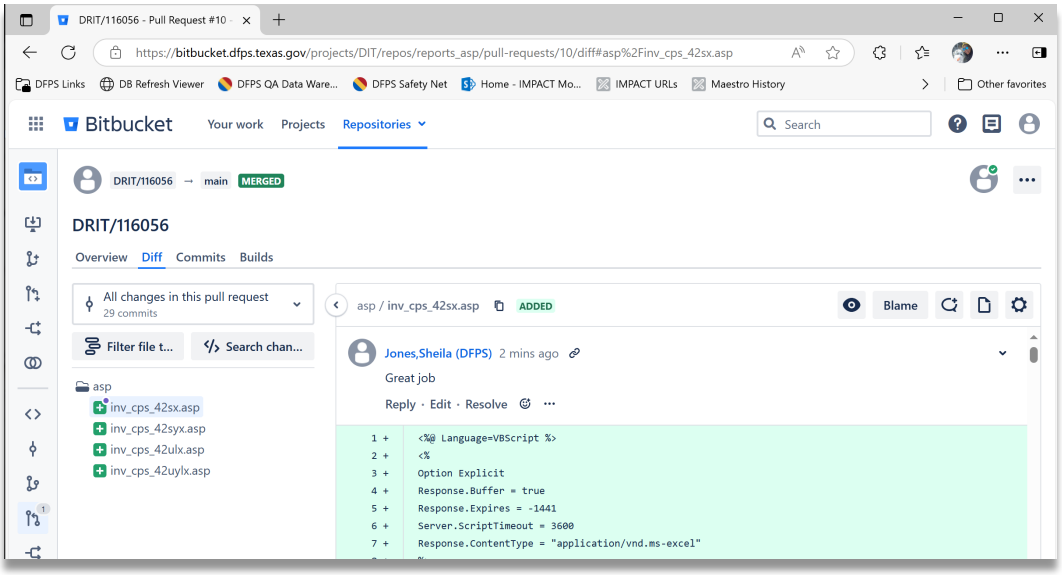
The reviewer can enter any overall comments. When done, the reviewer can choose **Approve or Changes requested.** Then click **Publish.**



If no changes are needed, the QA can now click **Merge** to get the Merge window. click **Merge.**



Now the screen indicates the branch was merged (you see ‘main’ next to merged due to the screenshot used, but it should be ‘qawh’ not main for this step. Just remember you need to create a pull request to merge into qawh before you do the same for main).



8. Repeat step seven to create a pull request to merge your branch to the main branch.

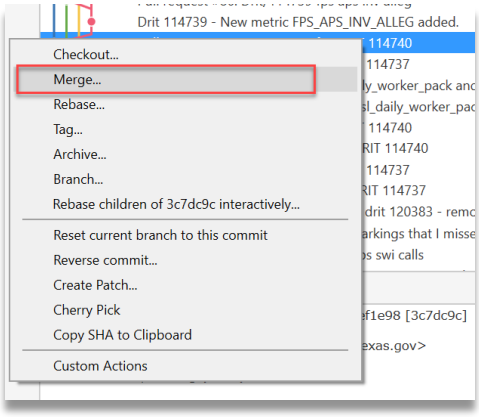
Merge Conflicts

The above workflow works for most of our DRITs. However, it may not work for OIDBA DRITs that require us to modify shared files (e.g., oidba_driver procedure, oidba_pkg). In that situation, you will likely encounter merge conflict when you try to merge your DRIT branch into qawh or main branch. The conflict occurs because git does not know which branch’s edits to retain (see example below).

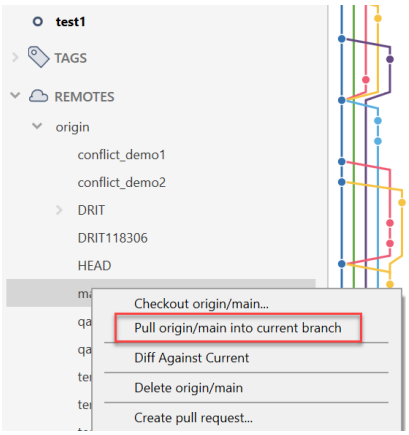
```
Starting point - File contains:
Line 1: Hello
Line 2: World

You do this (on your branch):
Line 1: Hello
Line 2: Beautiful World  ← You changed this

Another analyst does this (on their branch, at the same time):
Line 1: Hello
Line 2: Amazing World    ← They changed this
```

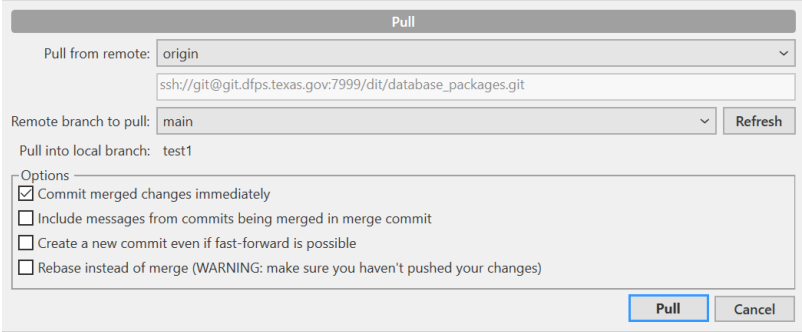
To prevent this from occurring, you can merge in commits that conflict with yours and resolve the conflicts locally before pushing to bitbucket. Here is how you do it:

1. Make sure your target branch (qawh or main) is in sync with remote by double clicking on the branch and then click on pull.
2. Create a temporary branch based on the target branch. For instance, if you need to merge into qawh, name it 'merge-qawh-drit#'. **Remember to create a different temporary branch for each merge. Do not merge a temporary branch based on qawh into main.**
3. Right click on the branch you need to merge into your target branch, then select 'merge'. This will attempt to merge your DRIT branch into the target branch.

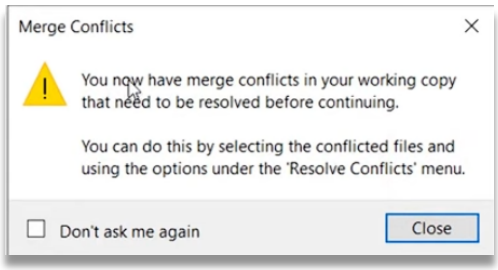


- Alternatively, if you want to avoid creating a temporary branch when merging into main, you can do this: First, make sure that your DRIT branch is active, then go to REMOTES/origin and right-click on main to select 'pull original/main into current branch'. Then click 'Pull'. This will merge main into your DRIT branch.

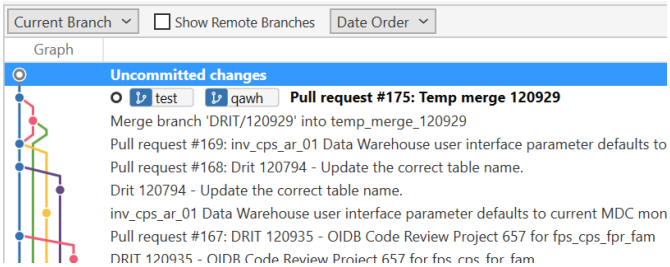




4. If there is a conflict, you will see the conflict message when you click ok to confirm merge.

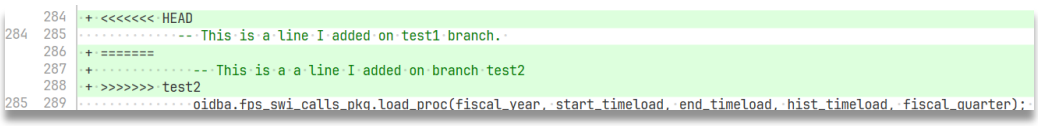


Click ‘close’ to remove the merge conflict message. Click on the temporary branch again and you will see ‘Uncommitted Changes’ in the commit history graph.



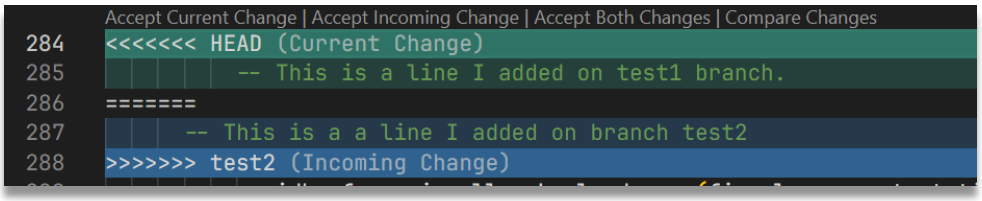
Under ‘unstaged files’, you will see the file with conflict. Notice the warning icon to the left of the file name.

In the file preview window, you see something like this. Open it in visual studio code or notepad++.

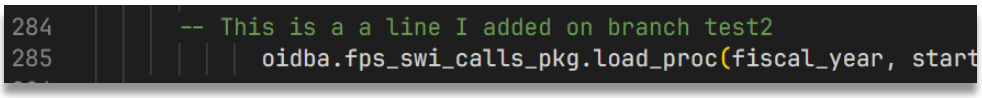


In this example, anything between **<<<<<<< HEAD** and `**=======**` is from your current branch (the merge branch). Anything between `**=======**` and **>>>>>>> test2** is from another branch named test2 (called incoming branch, i.e., the drit branch you want to merge into the merge branch).

5. Open the file with conflict in an editor and delete all the markings and resolve the conflict manually by **1)** removing both lines, **2)** keeping both lines, or **3)** keeping one of the lines. Either option would resolve the conflict, but option 2 is appropriate for our workflow. If you use Visual Studio Code, it comes with a nifty tool to allow you to choose one of these options and resolve the conflict for you automatically once selected.



I clicked on 'Accept Incoming Change'. This is what it looks like after that.

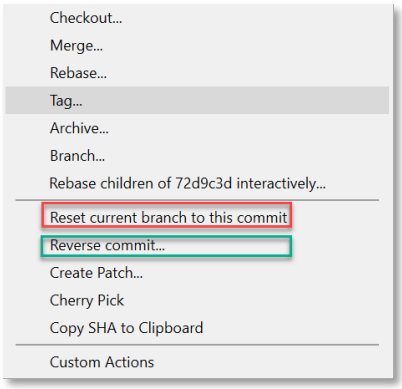


6. Commit the changes into the temporary branch. Now you can push the temporary branch to bitbucket. Then create a pull request to merge from the temporary branch to qawh or main. Don't forget to check `delete the branch after merge` since it is no longer needed.

In short, merge conflicts occur because two branches modify the same area of a common file and git doesn't know edits from which branch to keep. To avoid conflicts during pull requests, merge in commits that conflict with yours and manually resolve them locally before pushing back to bitbucket.

Guidelines

1. Make changes to source code from your git repo only.
2. For quick testing of asp files in qawh, copy and paste code from your git repo to qawh reports folder. When you are done testing, create a pull request to merge it into qawh branch.
3. Make changes to production code through bitbucket workflow only.
4. Do not delete existing commits that are already on qawh or main. Instead, make changes in new commits.
    a. When you right-click on a commit in Sourcetree, you will see actions allowed on that commit. Avoid '**reset current branch to this commit'** unless you know what you are doing (highlight in red). Use '**reverse commit'** if necessary to remove changes in previous commits (highlight in green).



5. Do not edit source code in bitbucket directly because it may introduce merge conflicts. Instead, make changes in your local repo and push it to the remote and merge it into the remote through a pull request.
6. Always pull from remote to get the latest code before you create your drit branch.
7. Always start with the main branch and create your drit branch based on the main branch.
8. Stick to the workflow as it is less likely to introduce complications.
9. Delete your drit branch from the remote once it has been merged into the main branch. For the drit branch on your local, you can keep it for reference or follow-up work.
10. When it comes to committing, here are some recommended practices:
    a. Commit when you complete a logical unit of work.
    b. Avoid commits that are either too small (too many commits) or too large (too few commits). As a rule of thumb, it doesn't hurt to commit once for every hour of focused work.

c. A commit message is composed of a subject and body (optional). For simple work, a subject is sufficient. For complex work, consider including a longer message in the body. Subject and body should be separated by a blank line.
d. Keep the subject within 50 characters and each line in the body within 72 characters.
e. Explain what and why, not how.
f. Use imperative mood, e.g., add, delete, remove, refactor
g. Be very specific and avoid vague terms such as 'fix this', 'change the code', 'update the code'.

Commit Message Examples

Simple work:

```
Resolve the error in sql that pulls in wrong cases in asp report
Add additional codes in sql to account for edge cases
```

Complex work:

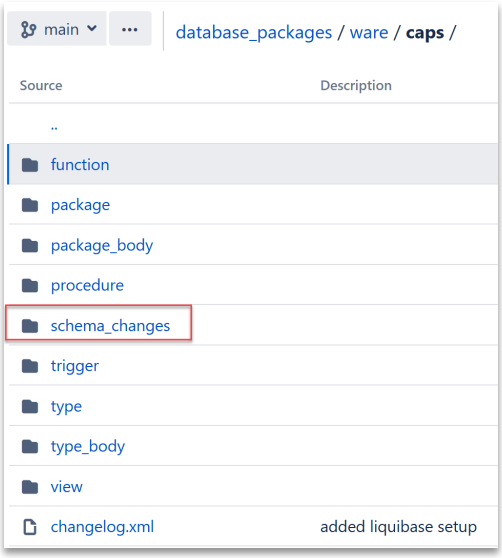
```
Revise completed investigation logic

This commit revised the logic that includes or excludes an investigation from
completed investigations. Excludes rapid closures and closed-to-merge cases.
Include any other completed investigations within a given timeload.
```

Branch Naming

1. DRIT - 'drit/#####'
2. MDC – 'mdc/mmyyyy'
3. PPM – 'ppm/####'
4. Other – 'other/####'

Database Schema Changes

If you need to make changes to database schema, e.g., create table, drop/change existing table, and so on, you should put your SQL scripts in the schema_changes directory within the respective schema directory. In this screenshot, you see schema_changes directory within caps schema directory.



Your SQL scripts should adhere to these guidelines (see 001_baseline.sql):

1. The first line should be : -- liquibase formatted sql
2. For each schema change, put it under a separate changeset with a unique identifier. In the screenshot below, each SQL statement follows a changeset notation: '-- changeset user_name: identifier' . **Make sure the identifier is unique**.

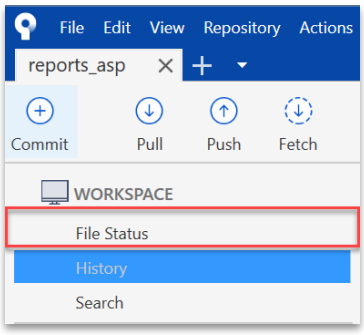
```
--changeset jonesba:001
CREATE TABLE liquibase_test (
    user_id       number PRIMARY KEY,
    username      VARCHAR2(50) NOT NULL,
    email         VARCHAR2(100) NOT NULL,
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

--changeset jonesba:002
CREATE INDEX idx_users_username ON liquibase_test(username);

--changeset jonesba:003
drop table liquibase_test;
```

3. Each changeset should contain a single operation, e.g., create a table, drop a table, create an index, etc. Do not try to put multiple operations under one changeset.
4. **For subsequent changes, add new changesets rather than revise existing changesets**.
5. For more complex changes, consider separating them out into multiple SQL scripts and putting the scripts in a subdirectory in the schema_changes directory.
6. Name your sql scripts in the following format:
   yyyymmdd_drit_#####_objectname.sql
   You can add additional identifier after object name if necessary.

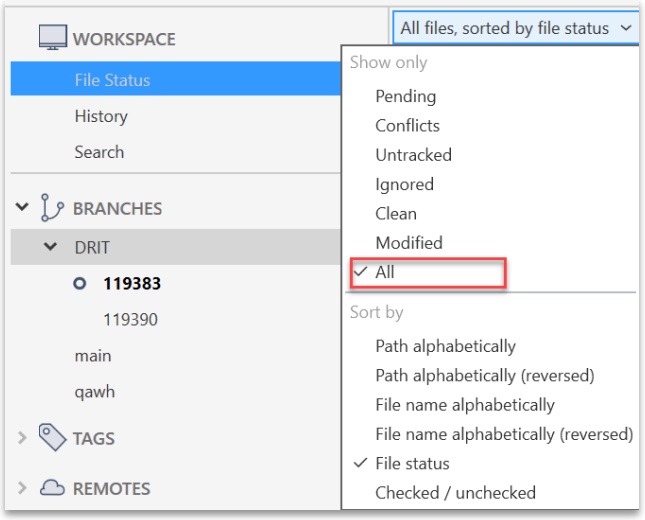
Display Commit History

If you need to see all the commits that touched a specific file:

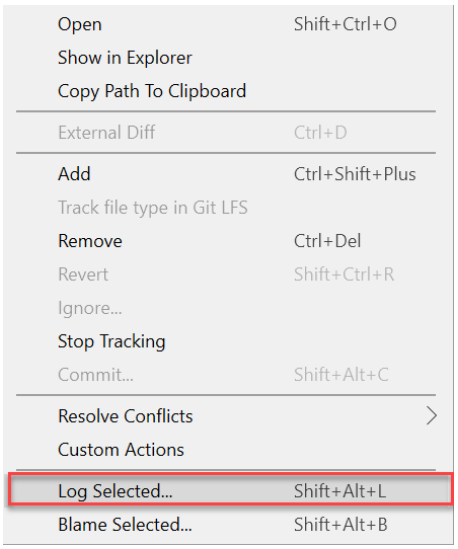
1. Check out the target branch by double clicking on the branch name (branch name turns bold).
2. Under workspace, select 'File Status'.



3. Select 'All' in the dropdown to the right of Workspace.



4. Navigate to the file you want and right-click on it and then select
    a. 'Log Selected', if you want to see all the commits that touched that file.
    b. 'Blame Selected', if you want to see line-by-line history of who changed what in the file.



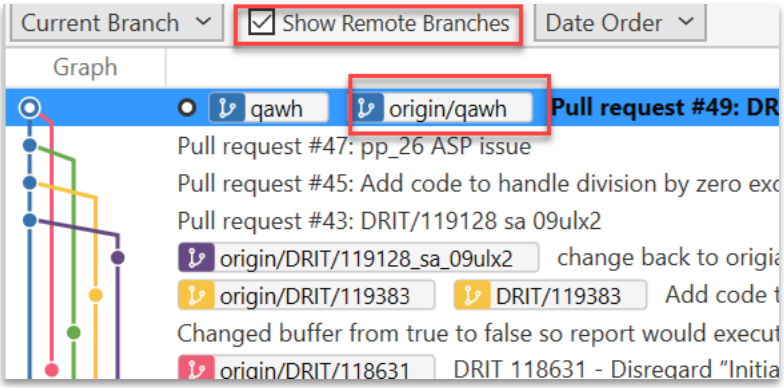
Local qawh/main branch ahead of remote

If you see **5↑** next to the qawh/main branch, it indicates that your local branch is 5 commits ahead of remote. **This should not happen as we do not directly work on our local qawh or main branch.** We merge in our changes to remote qawh/main through a pull request and then sync up our local with remote by clicking on 'pull' to merge in those changes from remote.

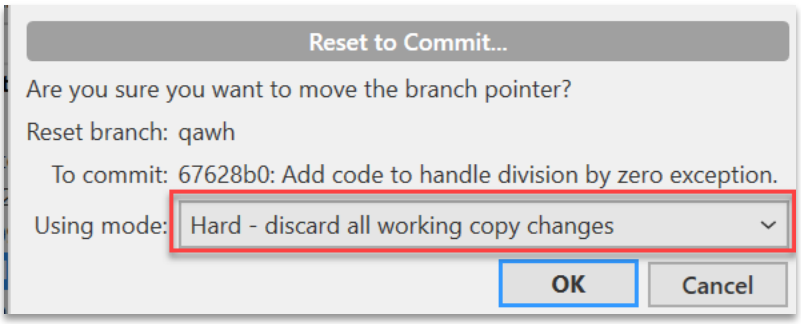
If you see 5↑ next to qawh/main branch, it is likely that you did something wrong, e.g., accidentally committed changes on the qawh or main branch rather than on your drit branch.

If that's the case, here is one way to make the number go away:

1. Check out qawh/main by double clicking on the branch.
2. Click on 'Fetch' in the toolbar to get the latest commits from remote (fetch is a safe operation as it doesn't merge changes from remote).
3. Above commit history window, click on show remote branches.



4. Then right click on the commit that has the label 'origin/qawh' to select 'reset current branch to this commit' and then select 'Hard – discard all working copy changes' for using mode and click OK.



What this reset does is to match your local qawh with remote qawh by discarding any commits that are on your local but not on remote. After this, 5↑ should disappear. Be aware reset-hard is a destructive operation that cannot be reversed. So, make sure that you have backed up any changes that would be discarded by this step.