Restructure of SDS REPO

Contents

[SDS REPO 3](#_Toc63251991)

[Migration Steps 3](#_Toc63251993)

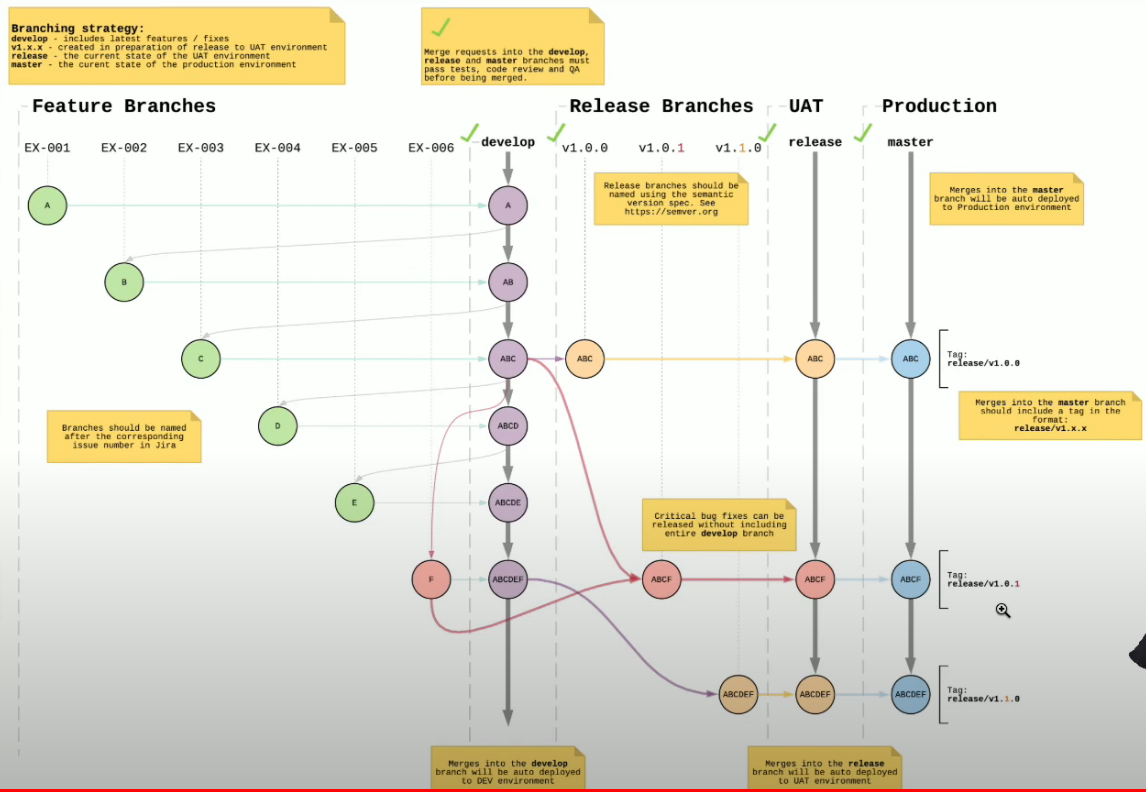
[Automated Build Process 15](#_Toc63251994)

[RELEASE BRANCH NAMING 15](#_Toc63251995)

[Additional REPO settings that I would suggest we have 16](#_Toc63251996)

# SDS REPO

# As part of the development process we want to adopt a development workflow that will help structure the release process, help maintain development standards and help support the automated build process and help us going forward. The core of the workflow is shown below but we have a few changes which are as follows.

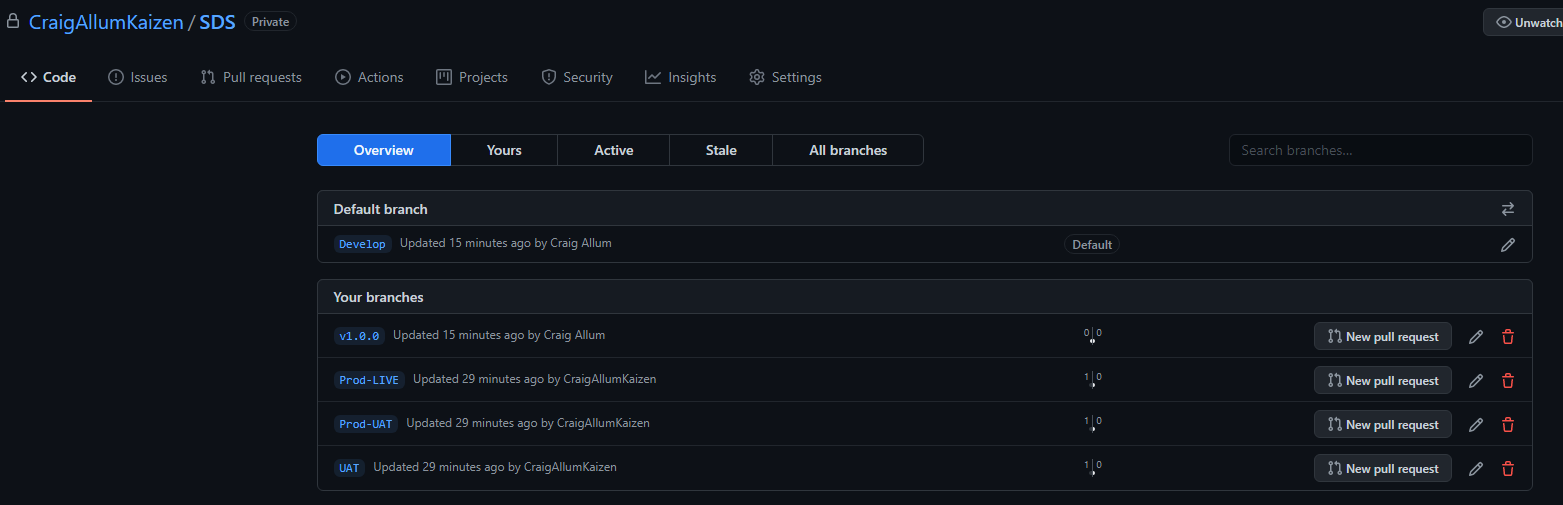


When merging from UAT branch we will be merging back into the Master trunk where we will be tagging the release. We will then be merging the release from master into the clients UAT and then from that client production system, so ignore production column in above flow.

Where the ticks are which occur when merging to other branches we will introduce peer review checks and auto test.

# Migration Steps

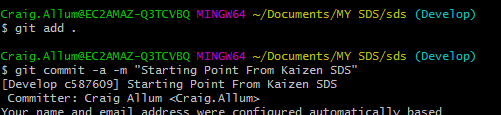
To demonstrate the theory I have set this up on a local REPO and work though an example to show how it works. First step is to create the Following REPO Structure in GitHub in New REPO



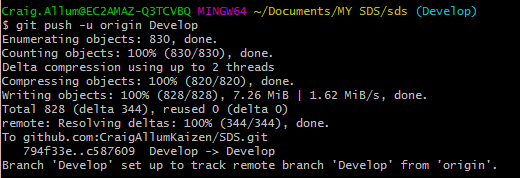
Now Clone the new SDS repo to local machine

Now clone old SDS repo to the local machine and copy all the objects to the new SDS repo. NOTE, I change the setting so the default branch is DEVELOP.

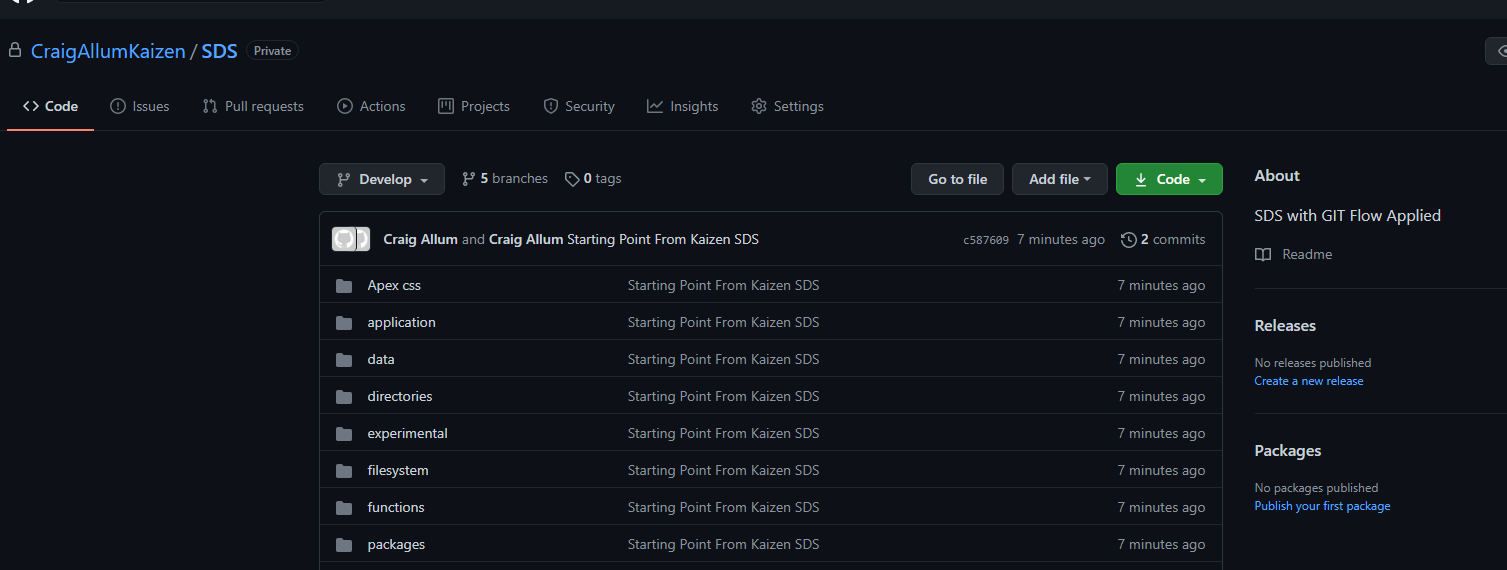
Stage and Commit the changes on the new SDS repo



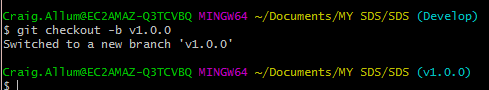
Push the changes up to the new repo



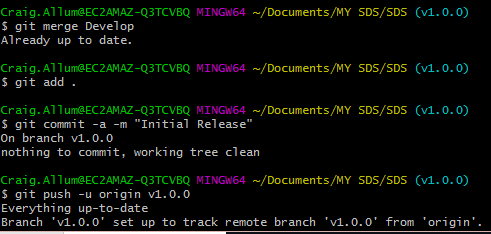
Check Git Hub and the Develop branch should reflect the old SDS main branch.



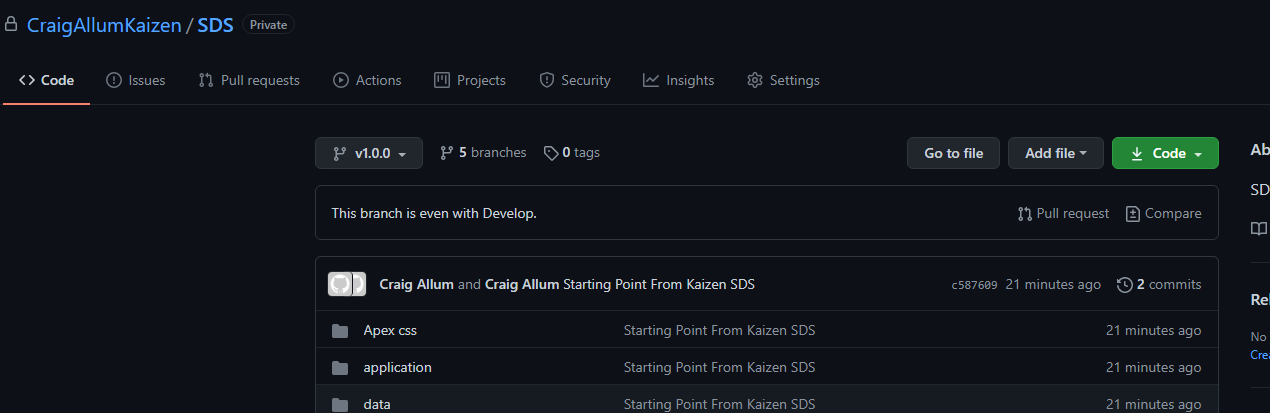
Now we need to make the changes in develop flow through the workflow so we have a baseline.



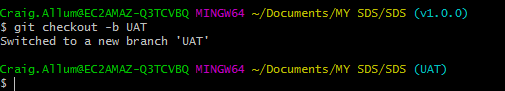
Now Merge the Changes in Develop to v1.0.0

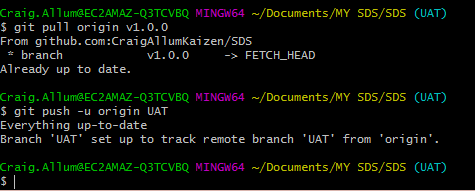


Now check Git, Develop and v1.0.0 should be in line with each other.

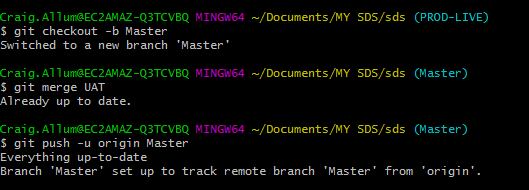


Now move v1.0.0 to UAT

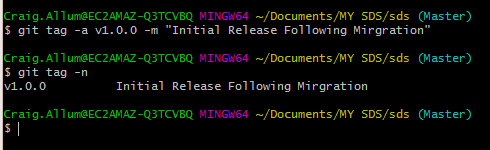




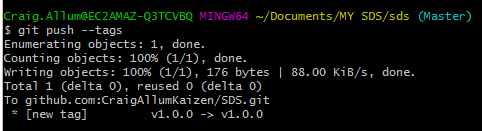
Assuming we have passed UAT we merge back into master and generate a Release tag.



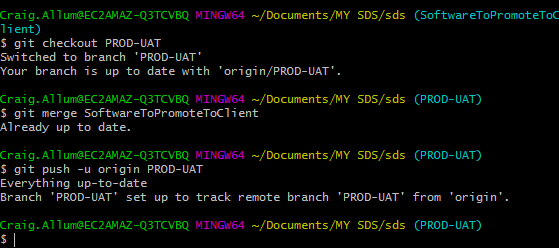
Now we Tag it



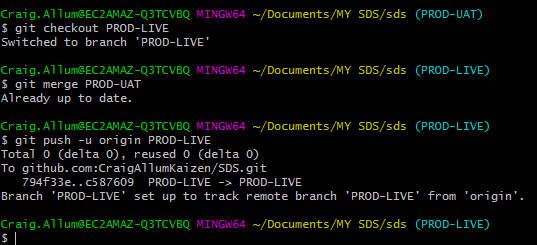
Push the tags



Having generated the Software release and update our master branch, we now move the code from UAT or ( Master checkout for the tag) and move the code over to PROD-UAT. In My example I just took the cut from master once we had tagged it up. The branch SoftwareToPromoteToClient reflects v1.0.0 in this case so I merge that into PROD-UAT.



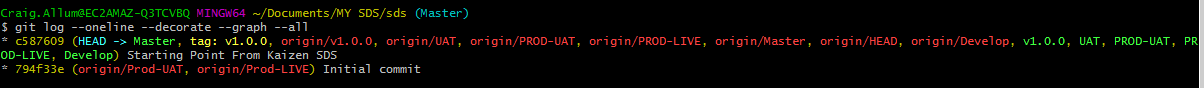
The Client does there thing and then we promote PROD-UAT to PROD-LIVE in the same way



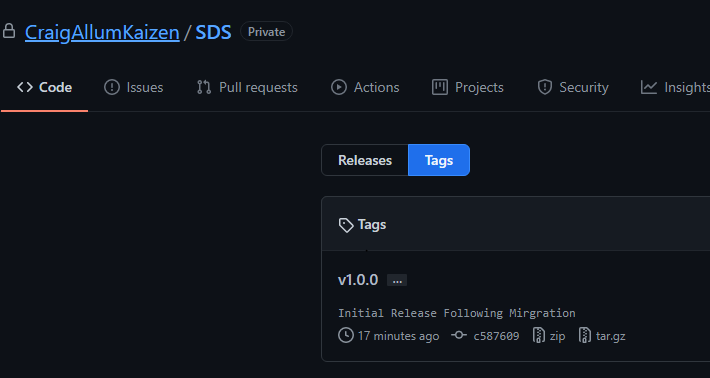
Once live is updated we bin off the branch we used to make sure we had the correct version

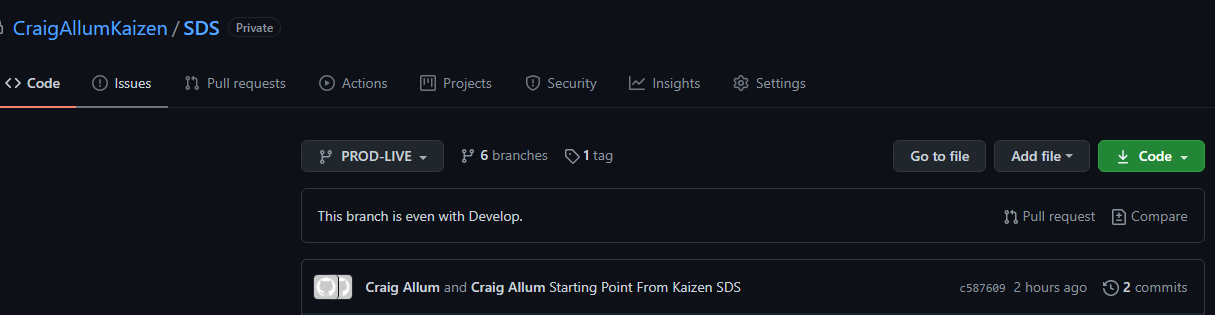


Now to check it looks all ok.



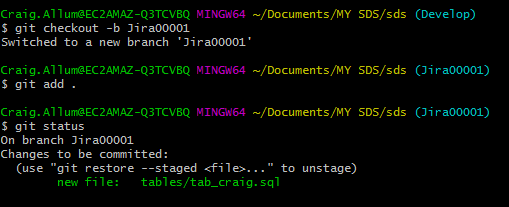
Check GitHub and we will see that we have all our branches in line, Master is showing the tag.

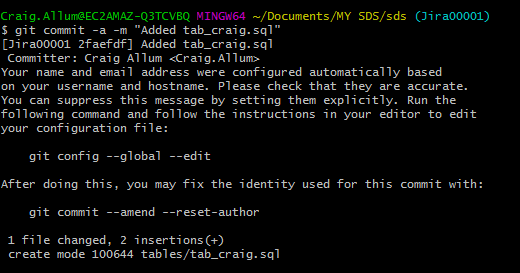


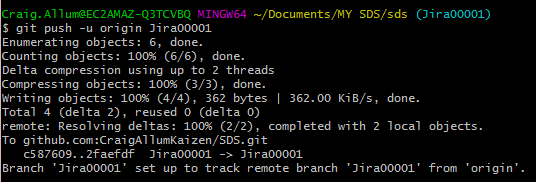


Now that is the basic setup of the new REPO.

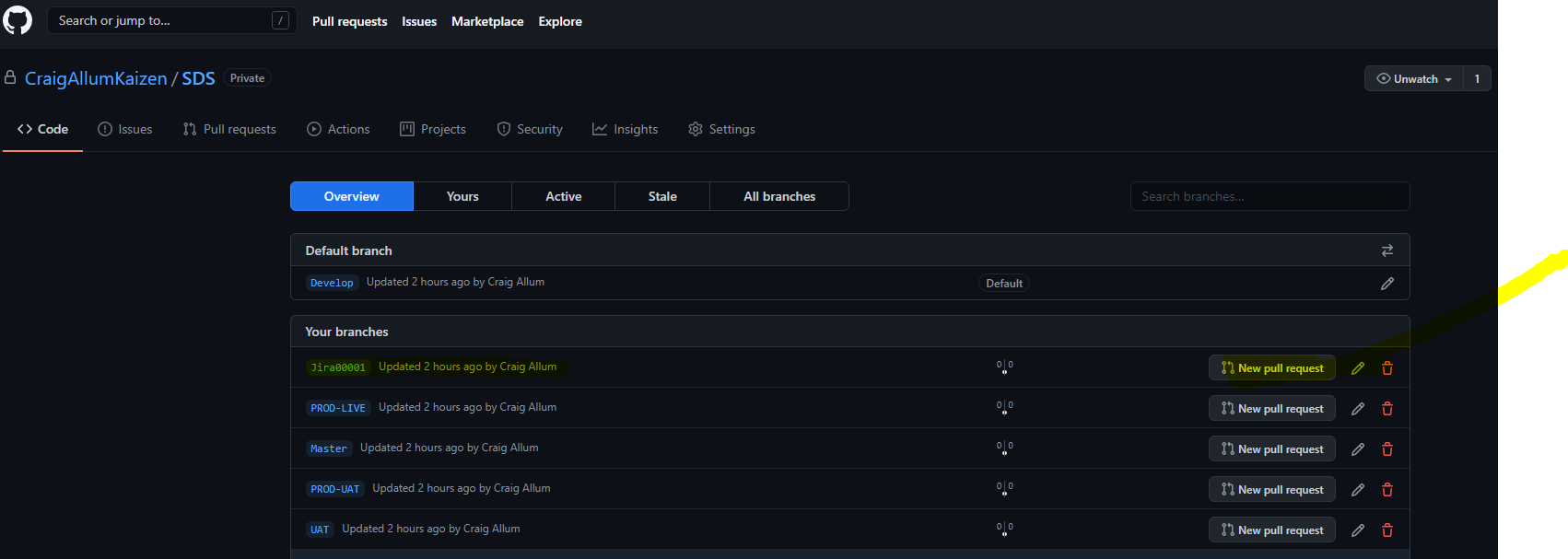
Now when we do development we branch from DEVELOP and create what we refer to as a feature branch, this branch should be named in line with the Jira task. Once on the Jira task branch do the change, stage it and the commit it.

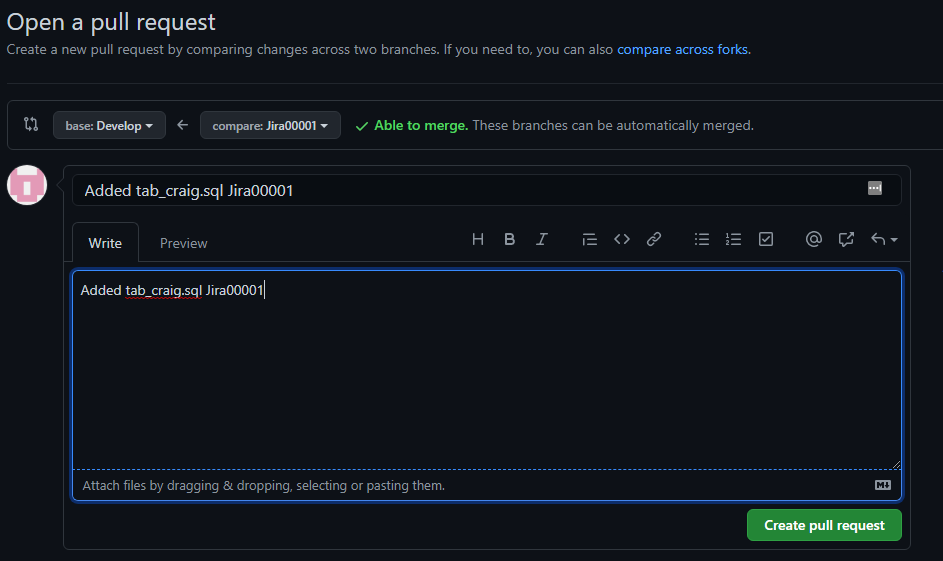




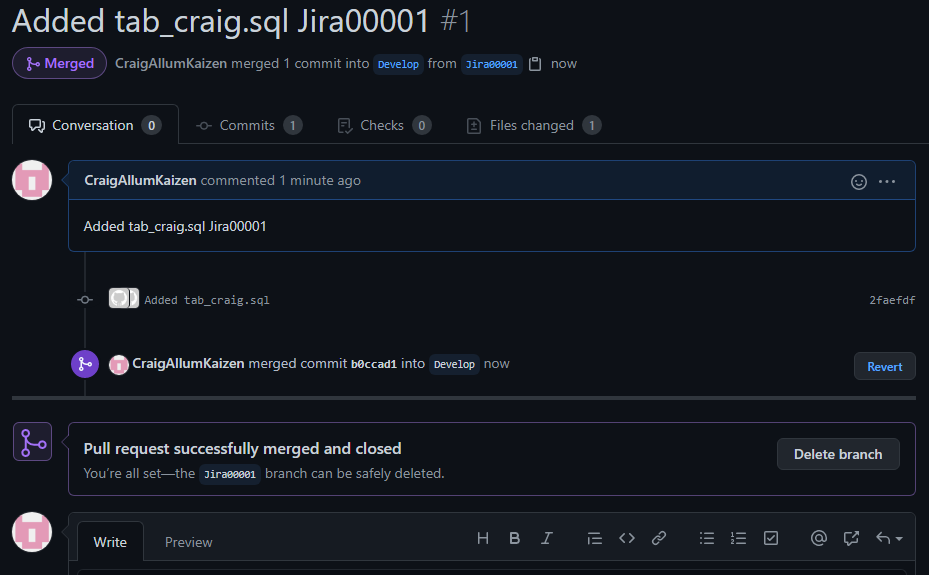


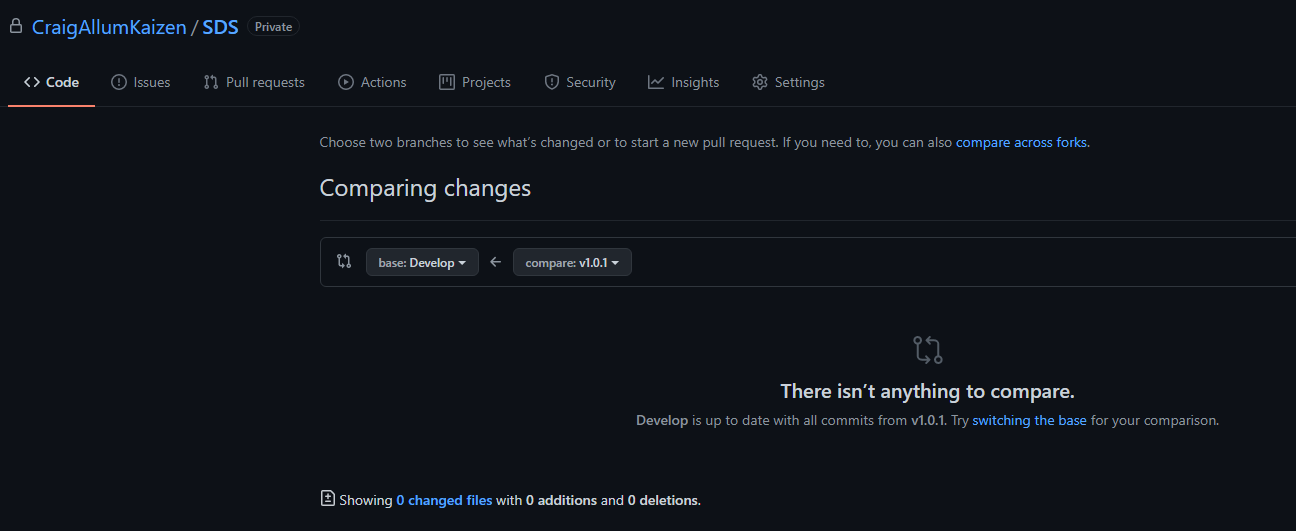
Now the User would request a Pull Request

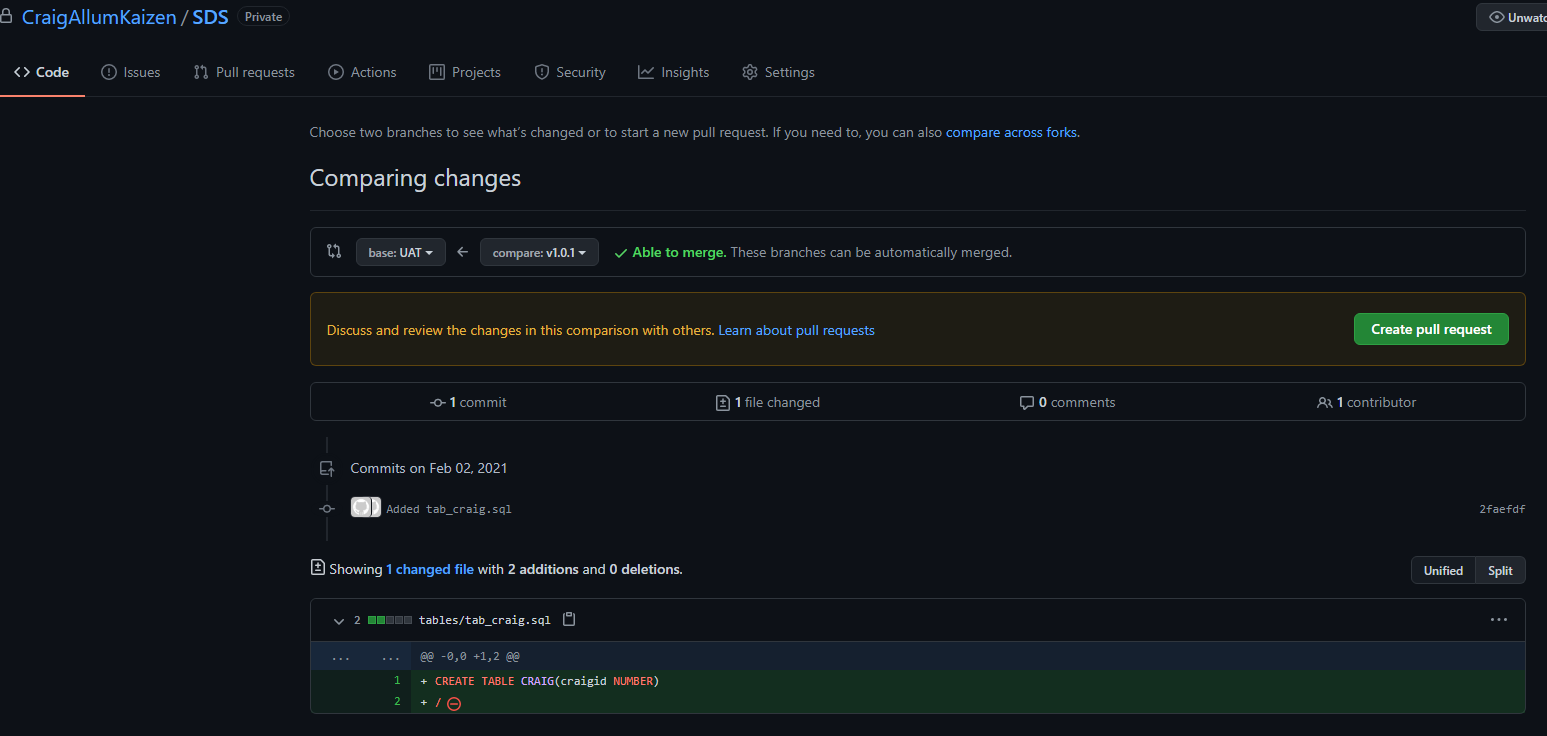




Note the BASE should be Develop and Compare should be the Jira branch you created. In this demo I am merging my own pull requests this will not be the case when we do the mirgiration as we will enforce the fact a developer cannot merge there own pull request.



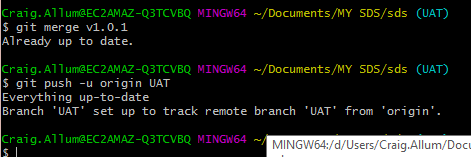




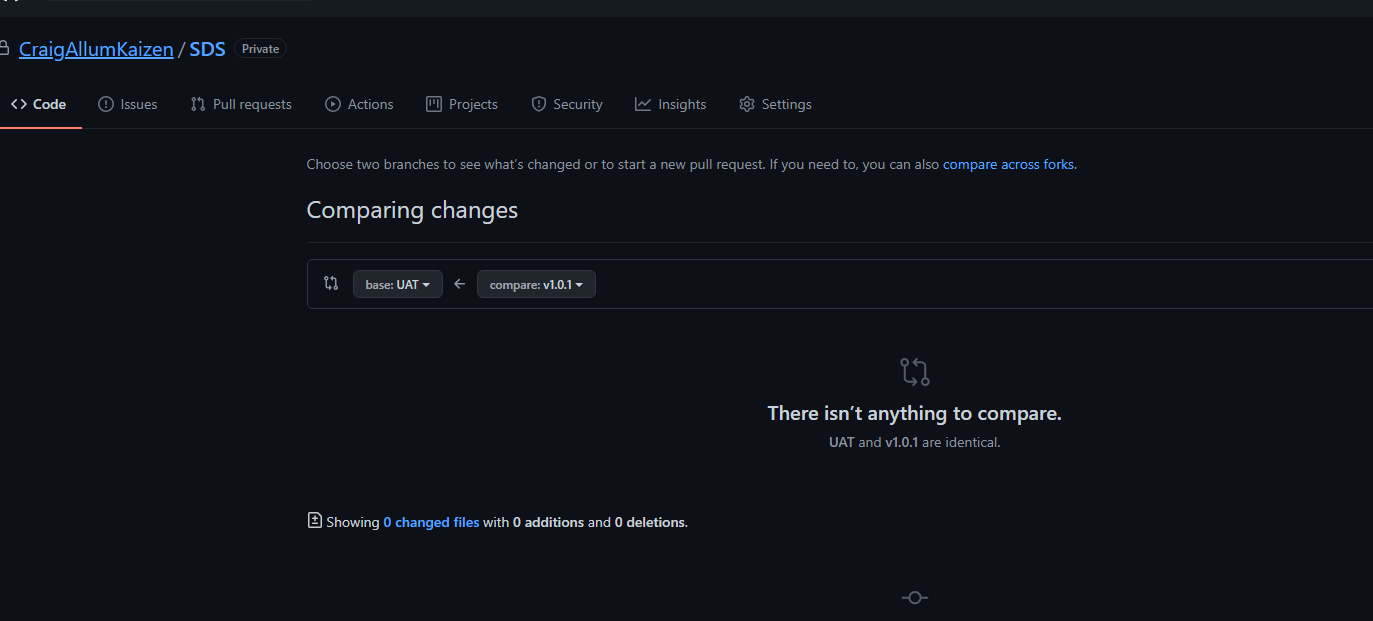
Now the Code is merged into Develop and we no longer need the Jira00001 branch so delete it.

Then all the workflow explained above would kick in, a release would be created at some point and a release branch would be created. That branch would be merged into UAT and if ok it will be merged into Master and Tagged. Then it would be merged in the PROD\_UAT and PROD\_LIVE as the client needs.

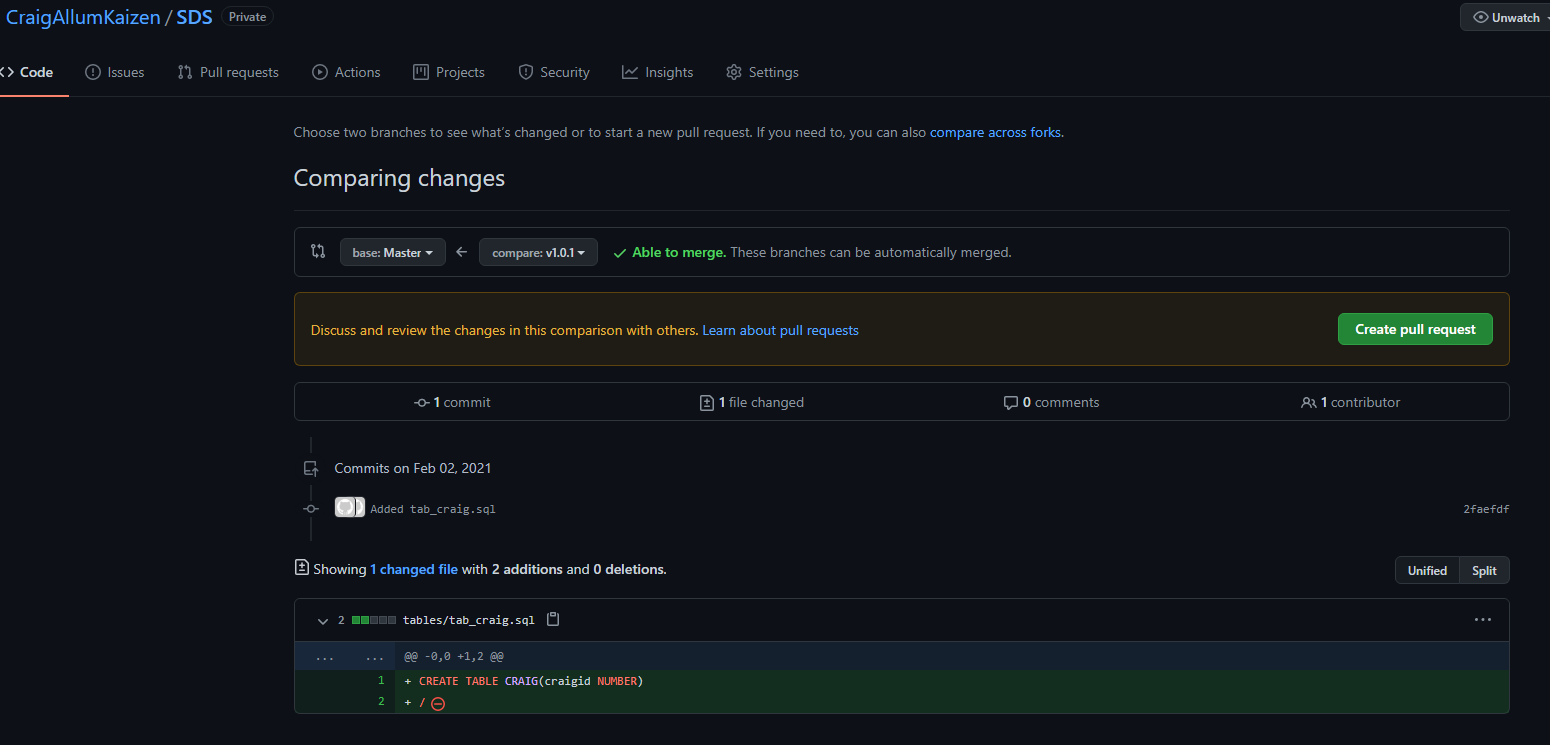
Just to recap lets pull the code through the process to Master and see what it looks like.



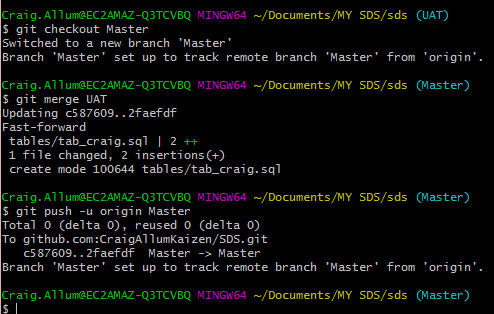
So when I compare UAT and Develop we should see no difference

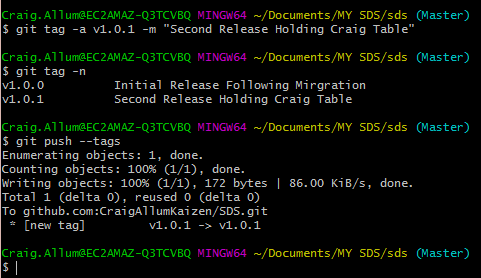


But changes are not showing in MASTER as expected.

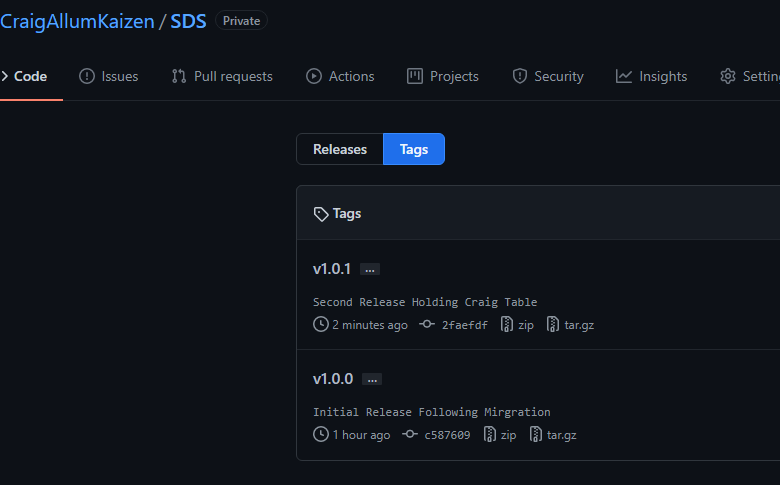


Now merge UAT into Master and then generate a release tag

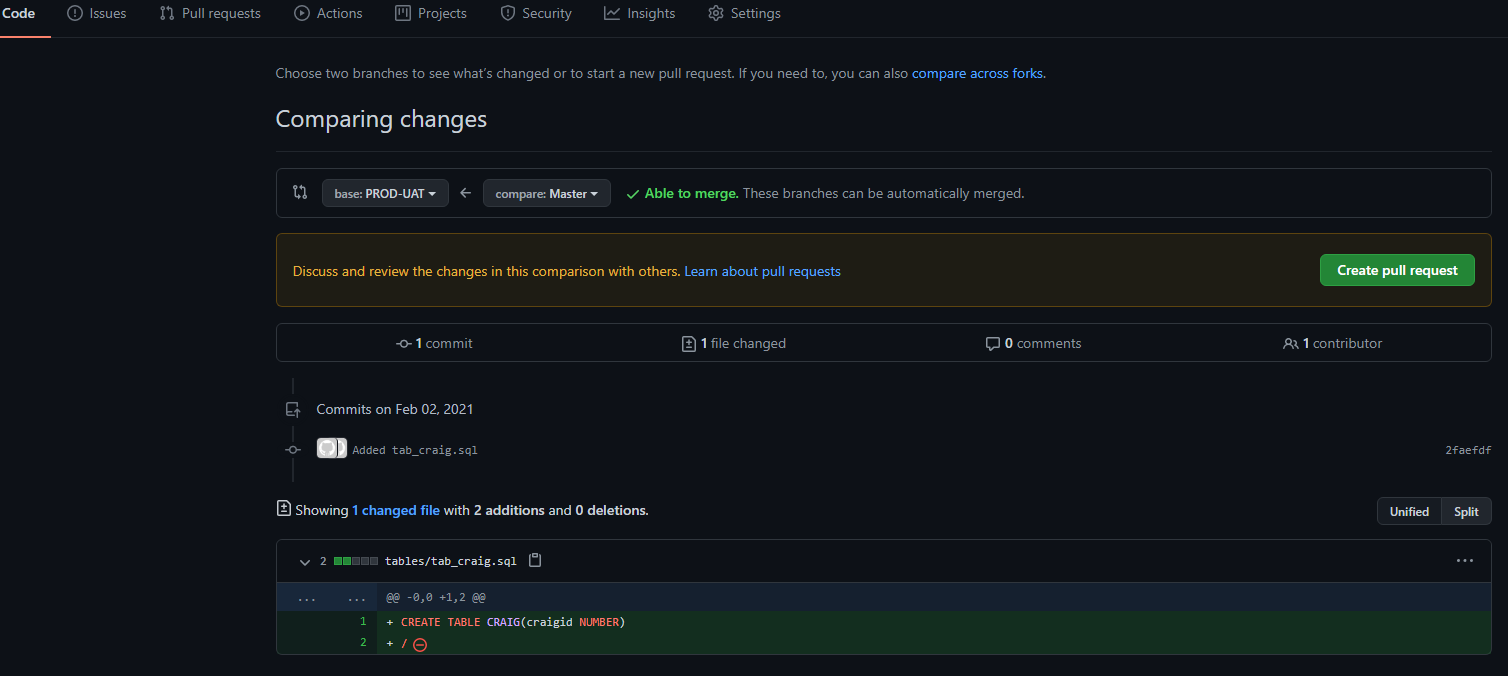




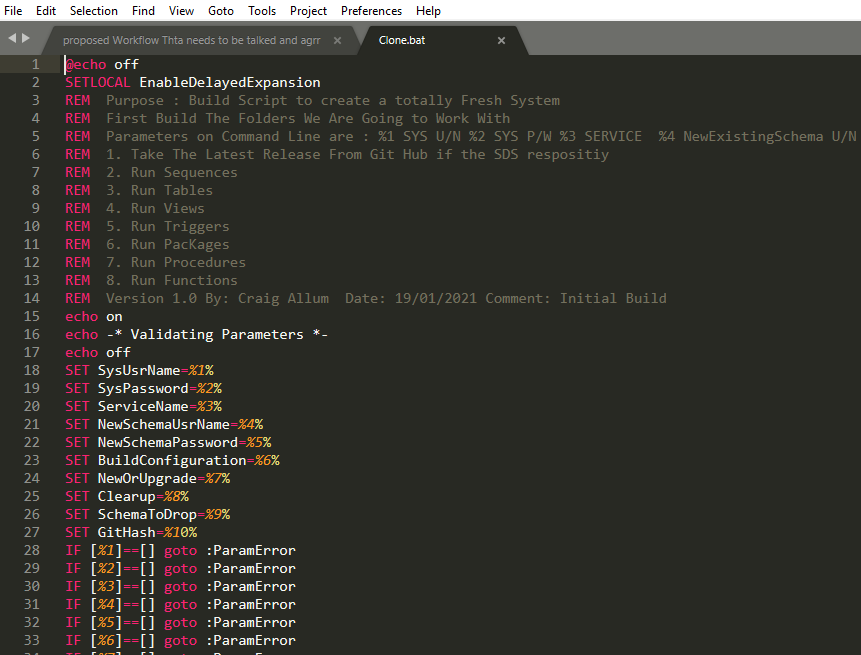
Check the release is looking as expected.



And when comparing changes it is clear I have stopped and not pushing it into the PROD-UAT branch



# Automated Build Process



Having get this workflow in place we know that we can clone the REPO and then using the version tag we can pick up all the relevant software for build version we want and then we can work our way thought them objects for that version and apply them to a target system.

To read about this process in more detail see AuotmatedBuildProcess.docx in REPO

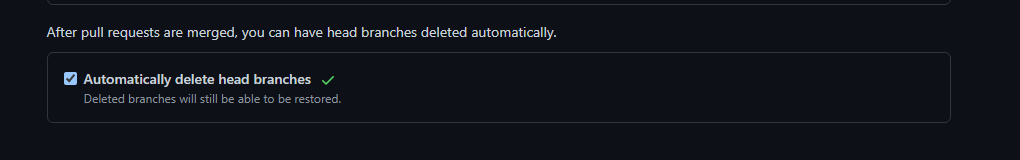
# Release branch Naming

When naming these branches we are going to be following Semantic versioning. It is quite easy to understand :

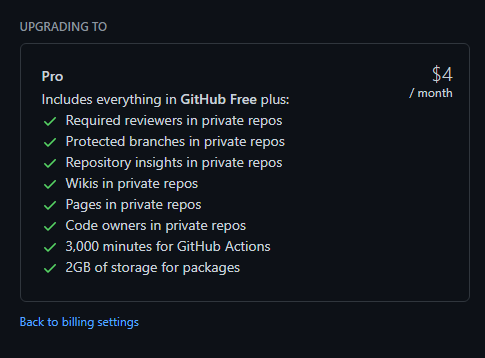
v<major>.<minor>.<patch>

* **major**: is a version number where you introduced breaking modifications (modifications of your new version are NOT compatible with previous versions);
* **minor**: is a version number that is compatible with previous versions;
* **patch**: is an increment for a bug fix or a patch fix done on your software.

# Additional REPO settings that I would suggest we have





Also I would like to get into the Branch Options

But I don’t have access to these, do you?