

GIT TUTORIAL

PL/SQL Developer 14.0.0.1961

Git Extensions 3.3.1.7897

View Changes Using GIT extension in PL/SQL (Right-Click on Files -> Git Extensions -> View Changes):

The screenshot shows the 'Diff' window in PL/SQL Developer. The top section displays the commit history for the 'master' branch. A yellow arrow points to the commit 'Transferring files once created' by 'alanars' 21 hours ago. A red arrow points to the commit 'New periods have been added' by 'origin/master' 2 days ago. The bottom section shows the diff for the file 'Packages/B2_Kronos_API.pck', highlighting changes between the current state and the commit 'New periods have been added'.

Commit	Author	Time	Hash
Transferring files once created	alanars	21 hours ago	8cb2615
Fixed a warning & removed comments	alanars	1 day ago	14d5df5
Missed one filename cleanup	alanars	1 day ago	54c8dcf
Fixed spaces in filenames	alanars	1 day ago	edb495a
Following code standards	alanars	1 day ago	845d058
Added a pay end date check	alanars	1 day ago	8829079
origin/master New periods have been added	alanars	2 days ago	5ea47b7
Removed Raise Error added Out Msg	alanars	2 days ago	419f49f
Make sure pay period is correct	alanars	2 days ago	9c44341
7 changes by Barry	alanars	2 days ago	9a70b64
Only one more instance of Env Id left	alanars	6 days ago	2de0e87

```
diff --git a/Packages/B2_Kronos_API.pck b/Packages/B2_Kronos_API.pck
index b9dbfd6..00bfb58 100644
--- a/Packages/B2_Kronos_API.pck
+++ b/Packages/B2_Kronos_API.pck
@@ -945,6 +945,9 @@ CREATE OR REPLACE PACKAGE BODY B2ag.B2_Kronos_Api AS
 945 945      l_Pay_Start      DATE;
 946 946      l_Pay_End        DATE;
 947 947      l_Full_Period    NUMBER;
 948 +      l_Time_Archive_Name VARCHAR2(100);
 949 +      l_Wo_Archive_Name VARCHAR2(100);
 950 +      l_Archive_Datetime DATE := SYSDATE;
 948 951  BEGIN
 949 952
 950 953      -- validate pay end date is valid Pay Period
@@ -1016,7 +1019,7 @@ CREATE OR REPLACE PACKAGE BODY B2ag.B2_Kronos_Api AS
 1016 1019      LOOP
 1017 1020          l_Unapproved_Count := l_Unapproved_Count + 1;
 1018 1021          IF l_Unapproved_Count = 1 THEN
 1019 -      Csv_Rec := B2_App_Util.Create_Csv_File('Mainsaver_Time_' || To_Char(l_Pay_End, B2v.Isodt) ||
 1022 +      Csv_Rec := B2_App_Util.Create_Csv_File('Mainsaver_Time_' || To_Char(In_Pay_Period_End_Date, B2v.Isodt) ||
 1020 1023          '_UNAPPROVED_TIME_' || REPLACE(In_Extract_Choice, ' ', '_') ||
 1021 1024          '.csv');
 1022 1025      --write header columns record
@@ -1045,7 +1048,7 @@ CREATE OR REPLACE PACKAGE BODY B2ag.B2_Kronos_Api AS
 1045 1048          B2v.Linefeed || 'Mainsaver timekeepers have been notified.';
 1046 1049
 1047 1050      ELSE
 1048 -      l_Time_Filename := 'Mainsaver_Time_' || To_Char(l_Pay_End, B2v.Isodt) || '-' ||
 1051 +      l_Time_Filename := 'Mainsaver_Time_' || To_Char(In_Pay_Period_End_Date, B2v.Isodt) || '-' ||
 1049 1052      REPLACE(In_Extract_Choice, ' ', '_') || '.csv';
```

Notice how *origin/master* (which is the repository in the cloud) is behind 6 commits? Why? Master is the current branch. We can think of master as the trunk or root of the source tree...

This beautifully shows how we've been changing code and by adding meaningful commit comments we get a deep understanding and can always come back to reference if we forget.

Let's look at our repo online (<https://bitbucket.org/scpafinsys/development/src/master/>):

bitbucket.org/scpafinsys/development/commits/

SCPAFinSys / Billing Aggregator / Development

Commits

Search commits

All branches

Author	Commit	Message	Date
Alan Arsiniega	5ea47b7	New periods have been added	3 days ago
Alan Arsiniega	419f49f	Removed Raise Error added Out Msg	3 days ago
Alan Arsiniega	9c44341	Make sure pay period is correct	3 days ago
Alan Arsiniega	9a70b64	7 changes by Barry	3 days ago
Alan Arsiniega	2de0e87	Only one more instance of Env Id left	7 days ago
Alan Arsiniega	77808df	Added chk02 and chk03 constraints	2020-05-14
Alan Arsiniega	37c23f6	Created 60/71 steps to allow some spacing	2020-05-13
Alan Arsiniega	f8c4129	Retry extract if issues (no materialized views)	2020-05-13
Alan Arsiniega	4ab38e4	Remove environment param for FTP and FTP PATH	2020-05-11
Alan Arsiniega	abb9baf	Forgot to add my comment to package header	2020-04-24
Alan Arsiniega	f82413d	Added comment rm -4 since that was for testing	2020-04-24
Alan Arsiniega	df5ccec	Modified per Barry to keep consistent with get_X/I	2020-04-24

So, our *origin/master* has all the code from our last *push*. Our *local master* on our desktop is about 6 *commits* ahead. Let's look and see if we have any pending changes we haven't committed to our local master before we *push*.

Stage code, commit, and push (Right-Click on Files -> Git Extensions -> Commit)

Commit to master (P:\finance\BillingAggregator\Source\Development\)

Working directory changes

Filter files using a regular expression...

1st choose all your files and stage below

Unstage

Stage

Commit message

Commit templates

Create branch

Options

Commit

Commit & push

Amend Commit

Reset all changes

Reset unstaged changes

This window is the diff tool, we saw it in the View Changes extension as well, here because I've highlighted B2_Kronos_API we see the differences

This is the Staging window, we can either stage a bunch of file/code changes here and commit them all with one message or split them up

2nd enter the commit message it will apply to ALL the files in the staging window once the commit or commit & push button are pressed

Committer alanars <aarsiniega@scspa.com>

master → origin/master

Staged 1/61

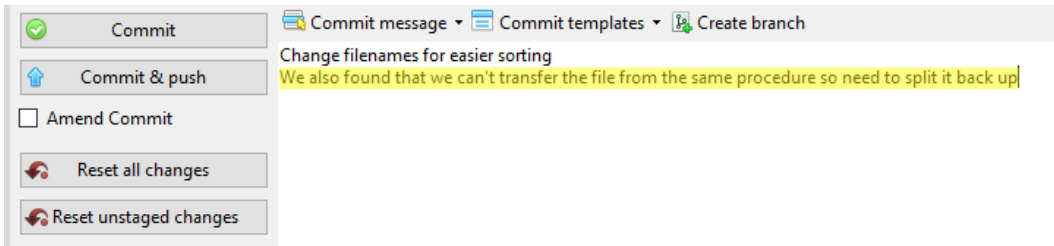
Ln 0

Col 0

Figure 1 There's actually a WHOLE lot of changes here that are asking to be committed... Everything in the 1st box.

Here is a good *commit* message. After this I was ready to *push* my *local master* to *origin/master***.

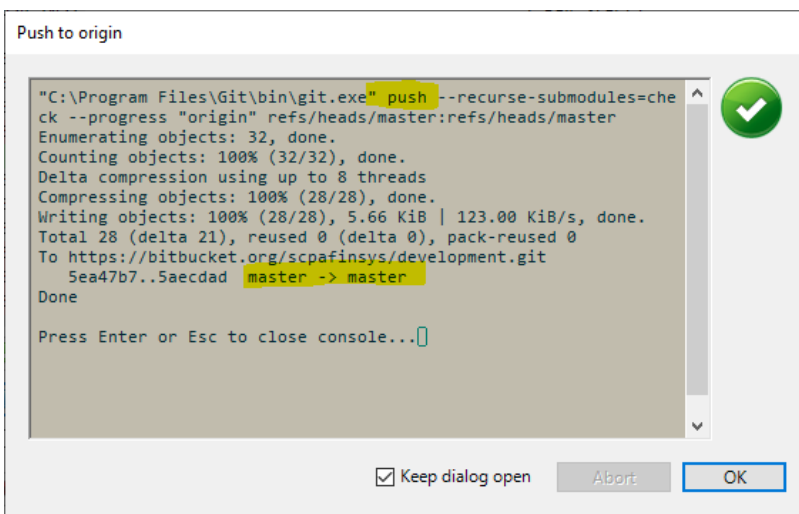
Had I wanted to work on the code more I would have just clicked *Commit* instead of *Commit & push*.



1st window tells us what the commit did (notice the commit cmd being issued):

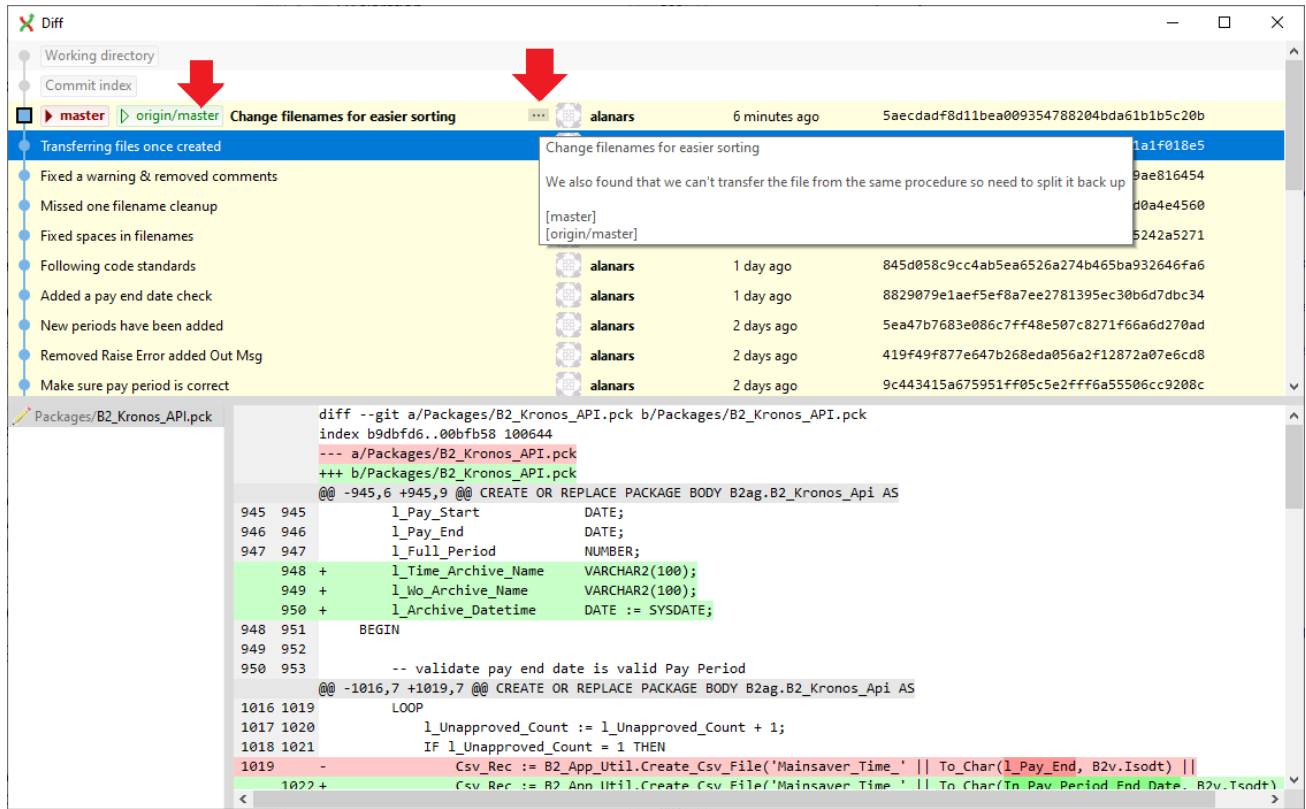


Once we click OK we get the 2nd window (because we chose Commit & Push) which shows us the result of the push:



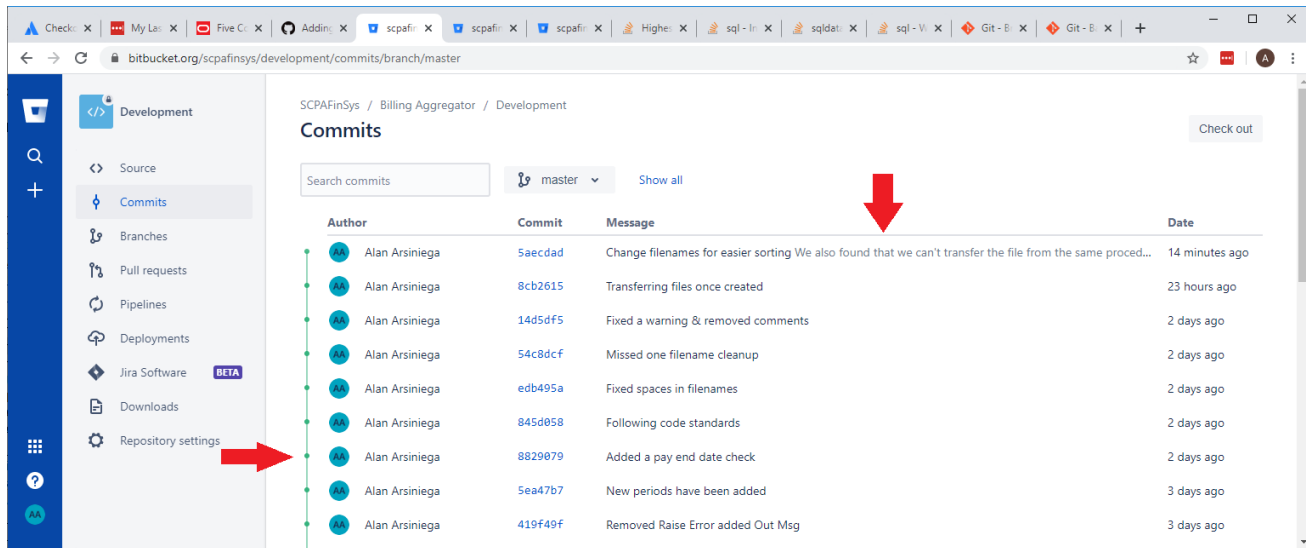
**(Notice that *master* is our branch, *local* is our PC or wherever the local code resides and *origin* is the cloud/bitbucket or wherever our main source code rests)

Let's view our changes:



First, we see that our master branch is caught up with origin/master. Also, by adding extra comments to the commit message we hover our cursor over the ellipsis (...) and see the full message.

Let's check origin/master (bitbucket):



That push caught up origin with ALL the previous commits that had been done on that branch (which happens to be master). Also, the extended commit message is visible just need to hover cursor over to see pop-up window.

Possible Hardships working with Master

The problems with working on solely master are numerous. A cursory google search rendered 3 articles:

Don't Mess with the Master: Working with Branches in Git and GitHub

(<https://thenewstack.io/dont-mess-with-the-master-working-with-branches-in-git-and-github/>)

Now it is time to start actually working with GitHub (and git) the way they are meant to be used: making changes in the project safely off to one side, and merging them back into the original project once they have proved to be correct. Or at least not disastrous.

Git-what-issues-arise-from-working-directly-on-master

(<https://softwareengineering.stackexchange.com/questions/335654/git-what-issues-arise-from-working-directly-on-master>)

If another developer starts work for a new feature from master, she starts with a potentially broken state. This slows down development. Different features/bugfixes are not isolated, so that the complexity of all ongoing development tasks is combined in one branch. This increases the amount of communication necessary between all developers. *new features* need their own development branch that can be deployed to a test environment before it is pushed to production. Otherwise, you're in a perpetual state of half-completed features. You can't deploy half-completed features to production, so if you're working directly on the master branch, everyone else must wait for you to finish your feature before anyone else's changes can go to production, including bug fixes.

Reasons-for-not-working-on-the-master-branch-in-git

(<https://stackoverflow.com/questions/5713563/reasons-for-not-working-on-the-master-branch-in-git>)

The master branch should represent the stable history of your code. use branches to experiment with new features, implement them, and when they have matured enough you can merge them back to master.

Using independent branches for features means that each new feature can be tested and deployed independently of the others.

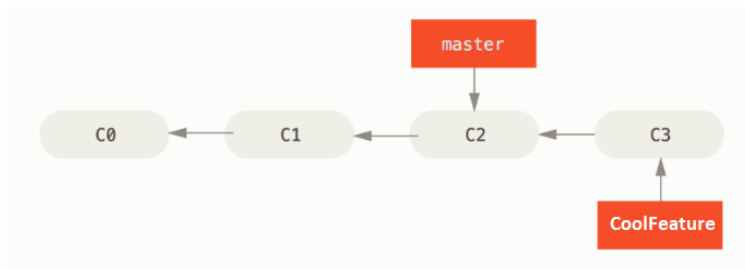
Luckily, branching amounts to learning two extra commands or just using the pl/sql extension further.

Branches in our Repository (Right-Click on Files -> Git Extensions -> Open Repository)

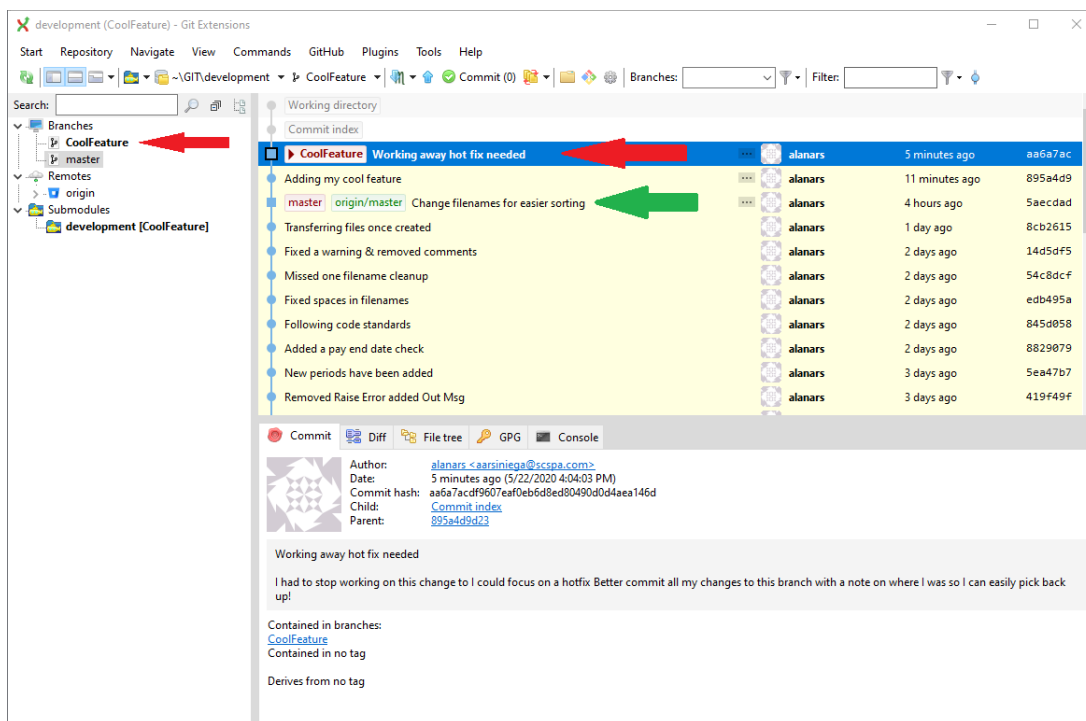
GIT has mastered working with branches to organize our work, whether it's a hotfix, a feature, or some lesser important bug fix – we can easily switch in and out of branches to logically break up the work. This eliminates the need of manually keeping track of which code changes are developed and tested to get pushed and which ones haven't and need to be commented out, for example.

We will create a branch, create and switch into another branch, and then merge to master from the completed branch (<https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>)

Some of this gets very repetitive so let's assume we are at this point (from the link above):



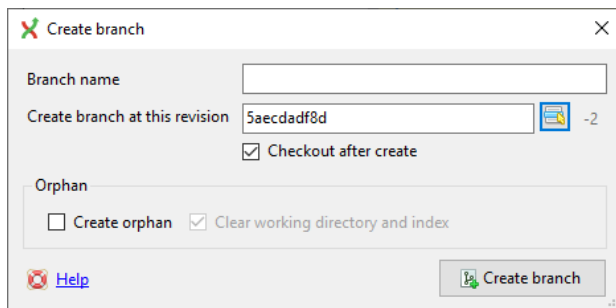
We open the repository (Right-Click on Files -> Git Extensions -> Open Repository) and we have:



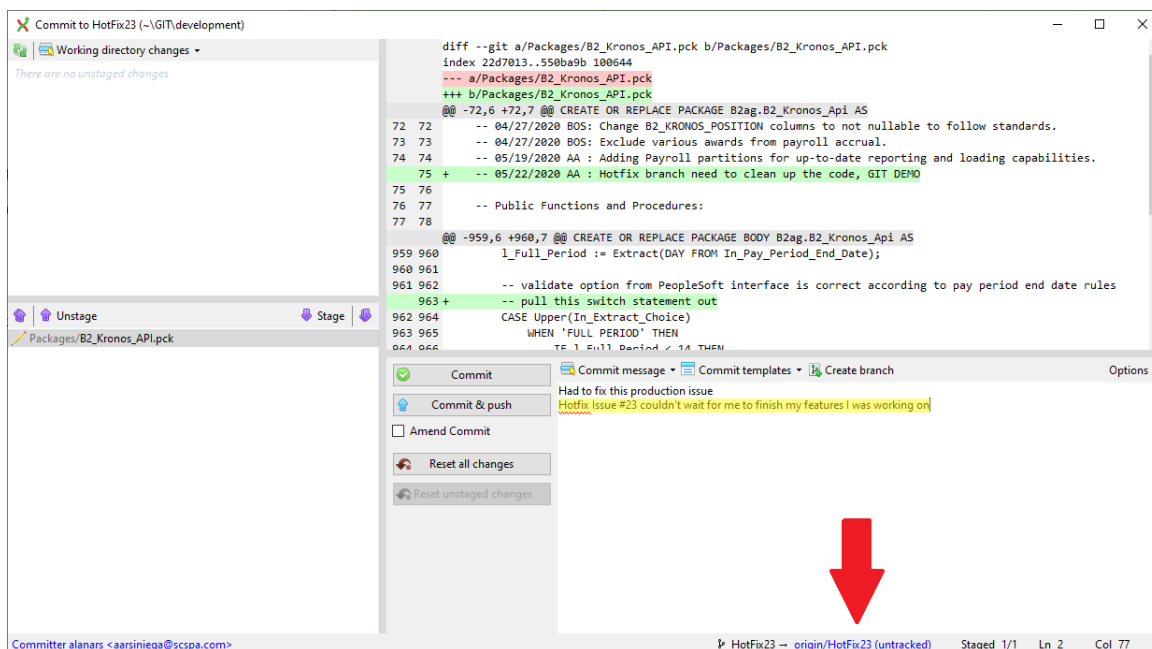
Notice how master is just another branch (left arrow). We've added our CoolFeature branch that we were diligently working on (right red arrow) when the request for a Hotfix comes rushing in. We will create that hotfix branch despite already being two commits ahead of master on our CoolFeature branch, as you can see above. Because our ticket system (like Jira) has an ID of 23, we can use that as part of the branch name.

Right-Click anywhere on the master, origin/master row (green arrow) to create a branch and choose 'Create new branch here... Ctrl+B' from that point. It makes sense to branch from there and not keep adding on to our current

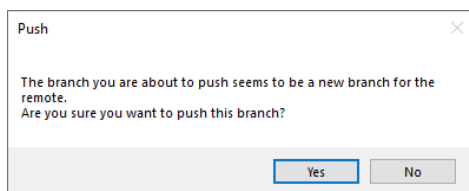
CoolFeature branch because we are still in the middle of development on that branch so we can just split from a point in time when master was in the ideal state.



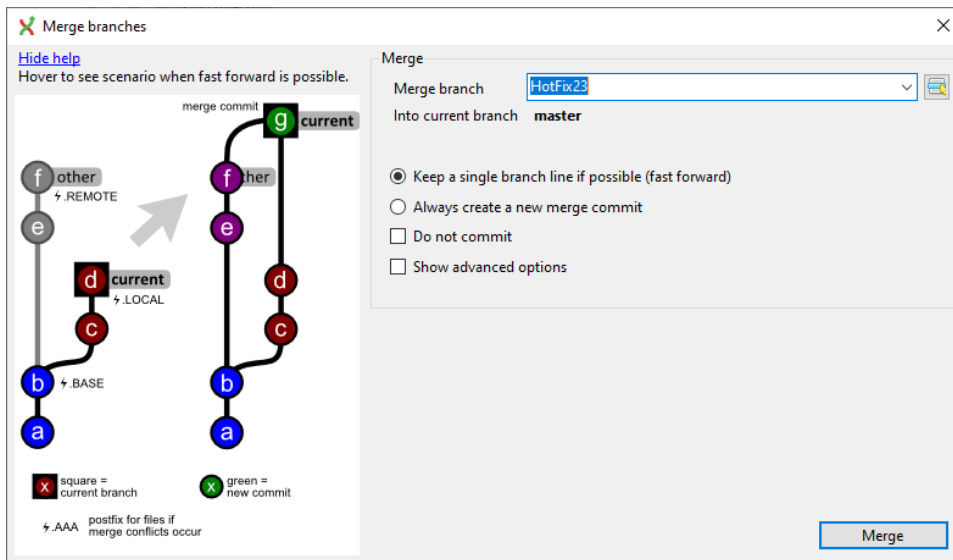
Let's call this branch HotFix23 and leave "Checkout after create" box checked this automatically check out this branch for you so that your local repo is now on this branch (it will show as bold in the Repository, left arrow above.) We will make a change to our code in the HotFix23 branch and commit the changes (**Right-Click on Files -> Git Extensions -> Commit**):



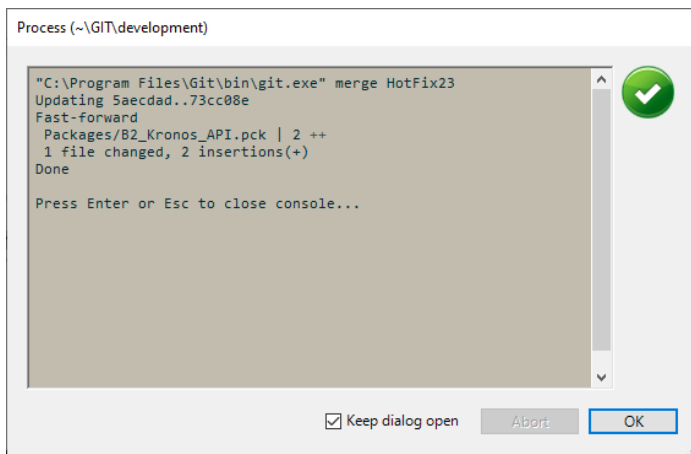
Way on the bottom we see origin/HotFix23 (untracked). This means the branch exists locally but not in *origin*, we can change that by choosing to Commit & Push and you would get this message to confirm:



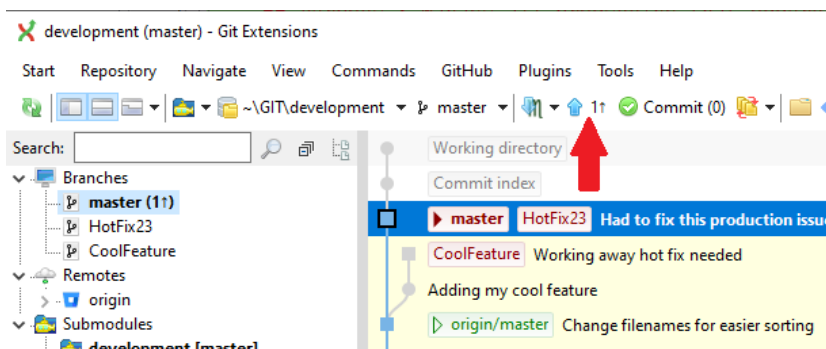
Let's choose Commit and am we now ready to merge this HotFix23 branch. Open the Repository, right-click on "master" underneath "Branches" and choose Checkout. A window pop us letting us know we've switched to branch "master." Hit OK and the font on "master" turns to **bold**. Now, choose the correct branch we want to *merge* into master, in this case HotFix23, right-click on it and choose *Merge*. Now we should see this window, with default settings:



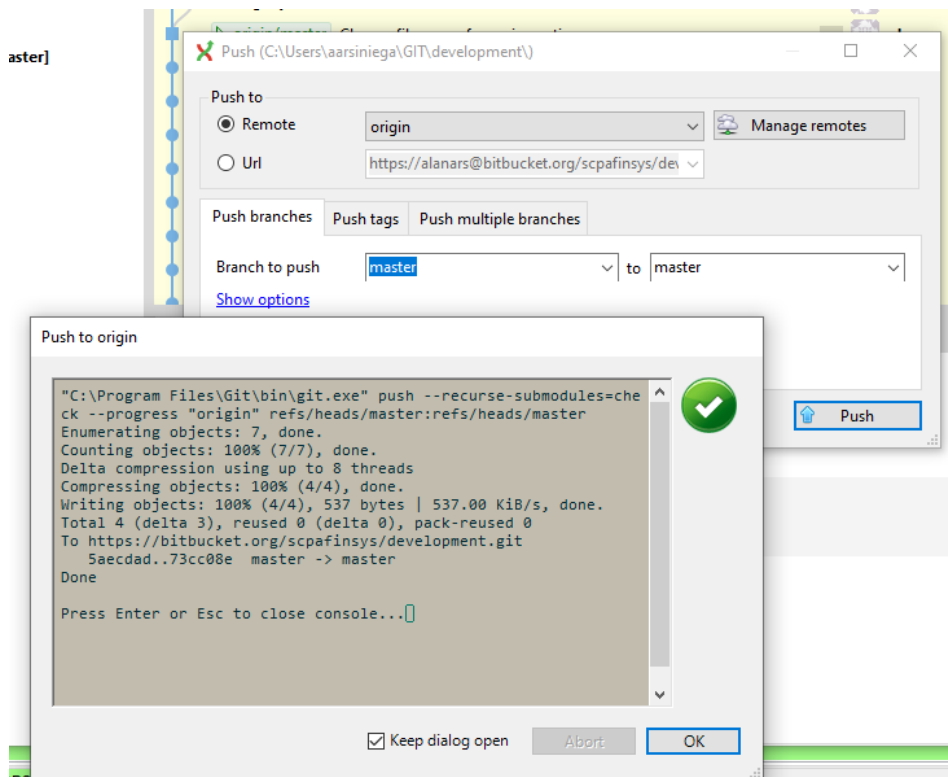
Pay close attention we have the right Merge branch and it's going into the right current branch, in this case **master**. Click merge, now we see:



Success! We have one small thing to do before we run this code in Production. So far, we are working on our local branch. We need to get these changes to the origin/master in the Cloud so they're visible by all. Still in the repository we notice a blue arrow with a (1) beside it:



I think this is saying we have a major merge we need to push up to origin/master since our local/master branch is now ahead in commits**. So, let's go ahead and click on the blue arrow pointing towards the cloud.



To Quickly Recap full lifecycle of GIT and code changes:

- Create the CoolFeature branch – (Open Repo, right click on master row to Create branch)
- Checkout the branch – (branch turns bold in the Repo if you leave “Checkout after create” box checked)
- Code – commit (ensure commits go to right BRANCH, publish the branch to the origin if untracked)
- Create hotfix branch from master (Open Repo)
PL/SQL Developer will warn you the timestamps changed. Always RELOAD or we might commit our coolfeature code into our hotfix branch.
- Code and commit (note code differences when we commit) until hotfix is resolved
Note new branch name – push to publish branch on BitBucket is optional)
- To merge our hotfix into the master branch
 - 1) Checkout branch we want to merge into (will show as bold)
 - 2) Right click & merge on branch you want to merge (hotfix23)
 - 3) Make sure you’re FROM branch is correct and TO is master and click MERGE.
- At this point we’re in master branch and should Push this new feature to our origin/master and resolve any conflicts before we run the code in production! Click on blue arrow pointing up to do so.

** If there were any conflicts we would need to resolve them since we are working locally but our origin/master is in the cloud and unprotected so someone could have pushed some commits to it without branching and now we need to resolve those commits if they're conflicting. Most of the times resolving conflicts isn't that difficult, please refer to the [git-scm](#) link above for an example.

For this reason, many shops will choose to protect their Master branch and require Pull Requests with a review required or several reviews and maybe even automation to fire off several tests before being able to merge into Master. In some cases, there can be a couple main branches that get pull requests instead adding another layer of protection. Every organization should implement the best strategy that adds the most value to their internal business processes and all their customers.