Contents

[1. Git, Git Flow, and CodeCommit 2](#_Toc35509560)

[1.1 What is GIT? 2](#_Toc35509561)

[1.2 What is Git Flow? 3](#_Toc35509562)

[1.3 What is AWS CodeCommit? 3](#_Toc35509563)

[1.4 Why Git and Git Flow? 4](#_Toc35509564)

[2. Development environment setup (Windows) 4](#_Toc35509565)

[2.1 GIT Setup 4](#_Toc35509566)

[2.2 VS Code Setup 4](#_Toc35509567)

[2.3 Clone Your First GIT Repository 5](#_Toc35509568)

[2.4 Clone QKit Repository 11](#_Toc35509569)

[2.5 Initialize “git flow” 11](#_Toc35509570)

[3. Procedure for implementing a feature branch 13](#_Toc35509571)

[3.1 Create new feature branch 13](#_Toc35509572)

[3.2 Merge into the develop branch 19](#_Toc35509573)

[3.3 Build QALK\_ALPHA via Jenkins 20](#_Toc35509574)

[4. Procedure for releasing QKit 21](#_Toc35509575)

[4.1 Create a release branch for BETA 21](#_Toc35509576)

[4.2 Merge into the master branch and develop branch 22](#_Toc35509577)

[4.3 Kick-off build for BETA Jenkins build 23](#_Toc35509578)

[4.4 Retag PROD using BETA changeset 24](#_Toc35509579)

[5. Procedure for implementing hot fix 26](#_Toc35509580)

[5.1 Create a hot fix branch from master 26](#_Toc35509581)

[5.2 Merge into the master branch and develop branch 27](#_Toc35509582)

[6. GIT Conflict Resolution 29](#_Toc35509583)

[6.1 Mock up 29](#_Toc35509584)

[6.2 Strategy 1: Merge feature branch into develop branch (GIT Merge) 32](#_Toc35509585)

[6.3 Strategy 2: Merge develop branch into feature branch (GIT Merge - Recommend) 33](#_Toc35509586)

[6.4 Strategy 3: Rebase feature branch onto develop branch (GIT Rebase) 35](#_Toc35509587)

[7. Other GIT Features 37](#_Toc35509588)

[7.1 Diff 37](#_Toc35509589)

[7.2 Undo 41](#_Toc35509590)

[8. More 43](#_Toc35509591)

[8.1 Conflicts when multiple developers working on same branch 43](#_Toc35509592)

[9. References 45](#_Toc35509593)

# Git, Git Flow, and CodeCommit

## What is GIT?

**GIT is a “Control System”**

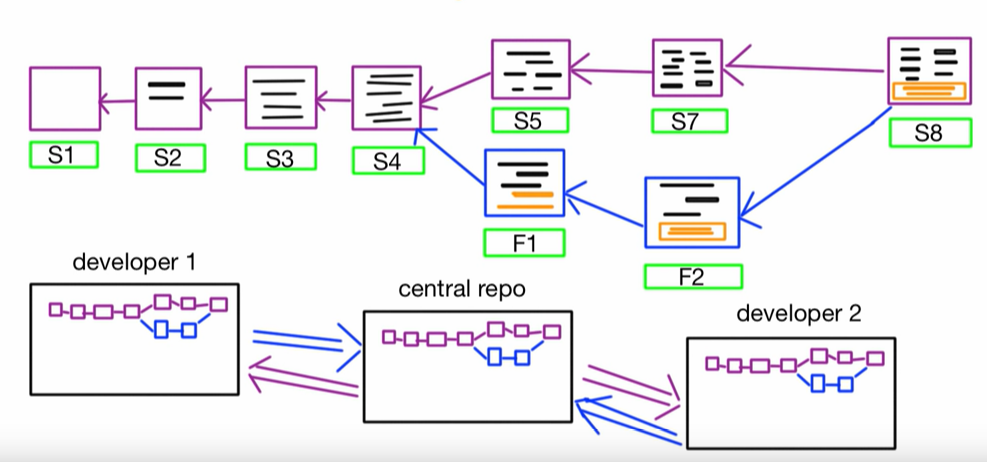
This basically means that Git is a content tracker. So Git can be used to store content — it is mostly used to store code due to the other features it provides.

**GIT is a “Version Control System”**

The code which is stored in Git keeps changing as more code is added. Also, many developers can add code in parallel. So Version Control System helps in handling this by maintaining a history of what changes have happened.

**GIT is a “Distributed Version Control System”**

Git has a **remote** repository which is stored in a server and a **local** repository which is stored in the computer of each developer. This means that the code is not just stored in a central server, but the full copy of the code is present in all the developers’ computers. Git is a Distributed Version Control System since the code is present in every developer’s computer.



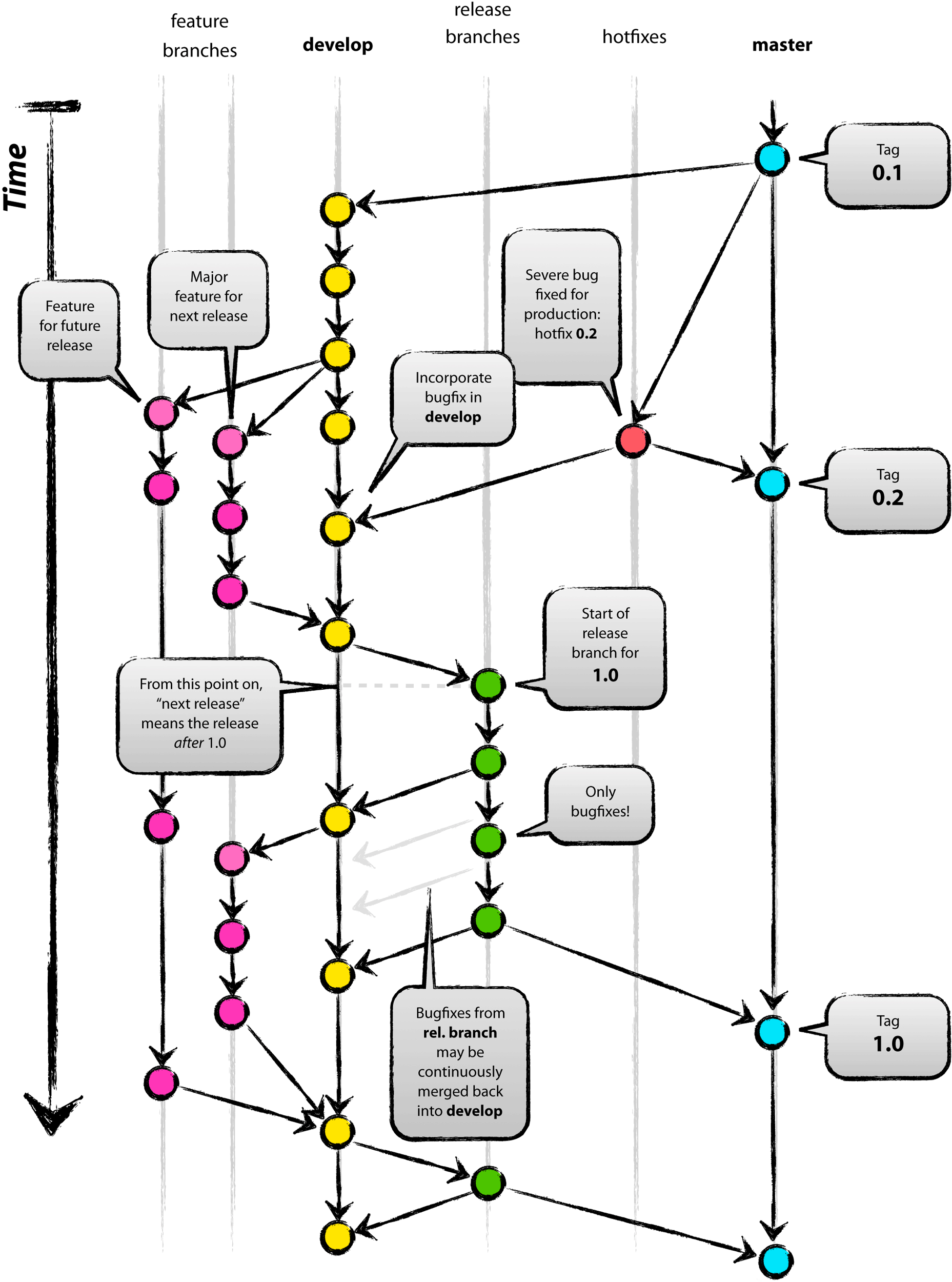
Git provides a set of command line. For example,

* $git add. Usage: git add [file] ...
* $git commit. Usage: git commit -m “[ Type in the commit message]” ...
* $git diff. Usage: git diff. ...
* $git reset. Usage: git reset [file] ...
* $git log. Usage: git log. ...
* $git branch. Usage: git branch. ...
* $git checkout. Usage: git checkout [branch name] ...
* $git push. Usage: git push [variable name] master.
* …

This document will cover most of the popular commands with examples; it will also intrudoce how to use GUI to execute those commands.

## What is Git Flow?

**GIT Flow** branching model is a git branching and release management workflow that helps developers keep track of features, hotfixes and releases in bigger software projects. This workflow has lot of commands to type and remember, so there’s also the git-flow library of git **subcommands** to help automate some parts of the flow to make working with it easier.



## What is AWS CodeCommit?

AWS CodeCommit is a fully-managed [source control](https://aws.amazon.com/devops/source-control/) service that hosts secure Git-based repositories. It makes it easy for teams to collaborate on code in a secure and highly scalable ecosystem. CodeCommit eliminates the need to operate your own source control system or worry about scaling its infrastructure. You can use CodeCommit to securely store anything from source code to binaries, and it works seamlessly with your existing Git tools.

## Why Git and Git Flow?

1. VS Code has integrated source control and includes Git support in-the-box. There are hundreds for Git plugins, but only 4 TFS plugins;
2. Use one workspace to switch among Git branches is easier than switching among multiple TFS workspaces; Merge changes from one Git branch to another branch is easier than merging changes from one TFS workspace to another TFS workspace.
3. Git Flow makes developer speed up the process with familiar branch structure. Single command to do multiple things at a time. Switching branches is easy. Keep repository & process clean and tidy.

# Development environment setup (Windows)

To play GIT on Aflac Windows 10, you need to change some Windows settings; install GIT and VS Code; Clone your first GIT Repo; etc.

## Installation Instructions

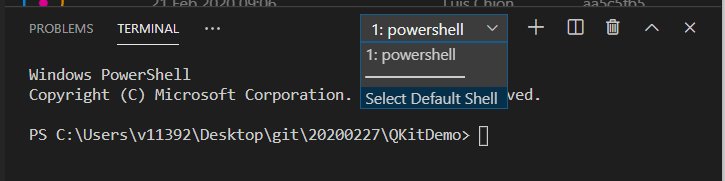
1. Follow instructions in Section Git Setup
2. Follow instructions in Section VS Code Setup.
3. Follow instructions in Section Clone QKit Repository

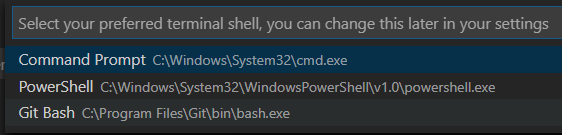
## GIT Setup

Download git from <https://git-scm.com/> and install it with default options.

## VS Code Setup

1. Download VS Code from <https://code.visualstudio.com/> and install it with default options.
2. Open VS Code; Select View > Terminal or use shortcut “Ctrl + ~” to open terminal; Change the “Default Shell” from “Powershell” to “Command Prompt”;

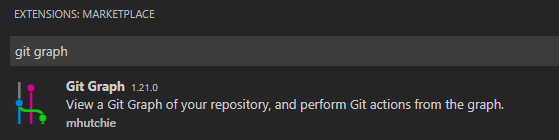




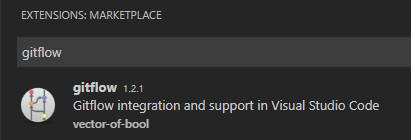
Click the “+” button in terminal, you will see a “cmd” shell is opened.

1. Open VS Code; Switch to “Extensions” view  from the left menu.

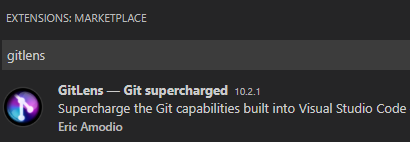
Search and install plugins “Git Graph”, “gitflow”, and “GitLens”.



<https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph>

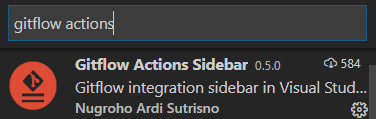


<https://marketplace.visualstudio.com/items?itemName=vector-of-bool.gitflow>



<https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens>

1. (Optional) Search and install plugin “Gitflow Actions Sidebar”



<https://marketplace.visualstudio.com/items?itemName=ardisaurus.gitflow-actions-sidebar>

Recently added the plugin to this document (v1.1). It provides a GUI way to start/finish feature/release/hotfix branch. If you know how to use “gitflow” plugin, you will not have problem to use “Gitflow Actions Sidebar” plugin. This plugin is built on top of the source code of “gitflow” plugin. No sure if the author will maintain it in the future.

e. Search and install other additional plugins : “Python”, “Python Test Explorer for Visual Studio Code”, “Test Explorer UI”

## Clone Your First GIT Repository

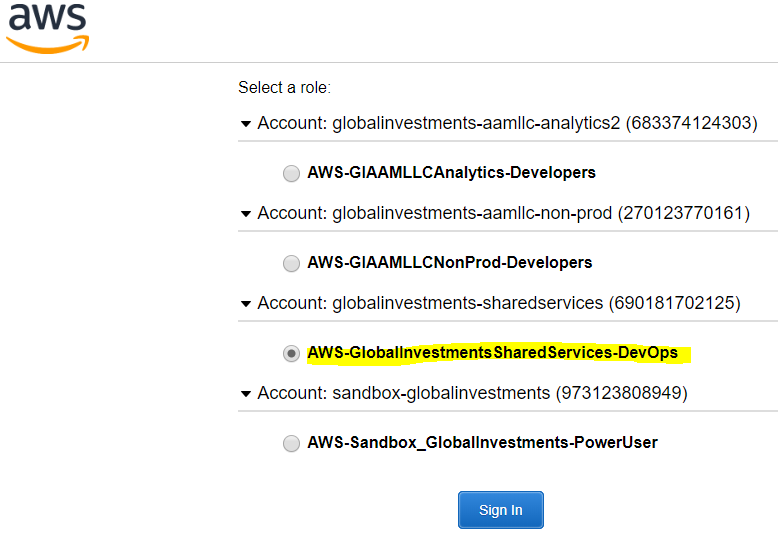
Let us clone GIT Repository from AWS CodeCommit.

1. Login AWS console with SSO through following URL: (If you do not have access to AWS console, please go ahead with step b)

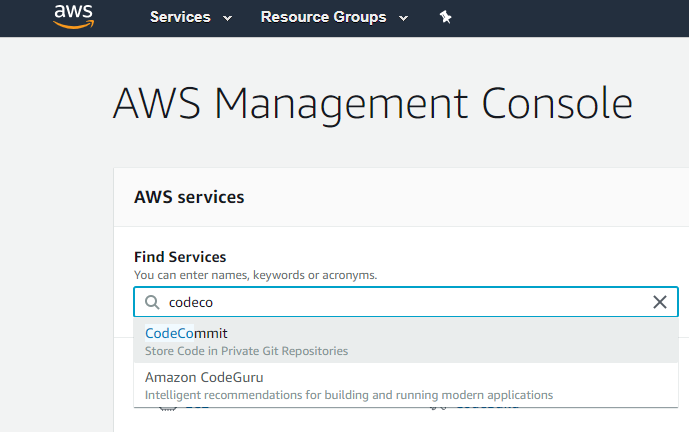
<https://fim.aflac.com/idp/startSSO.ping?PartnerSpId=urn:amazon:webservices>

(Please copy the URL to chrome browser)

Select “Shared Services” account; Click “Sign In”.

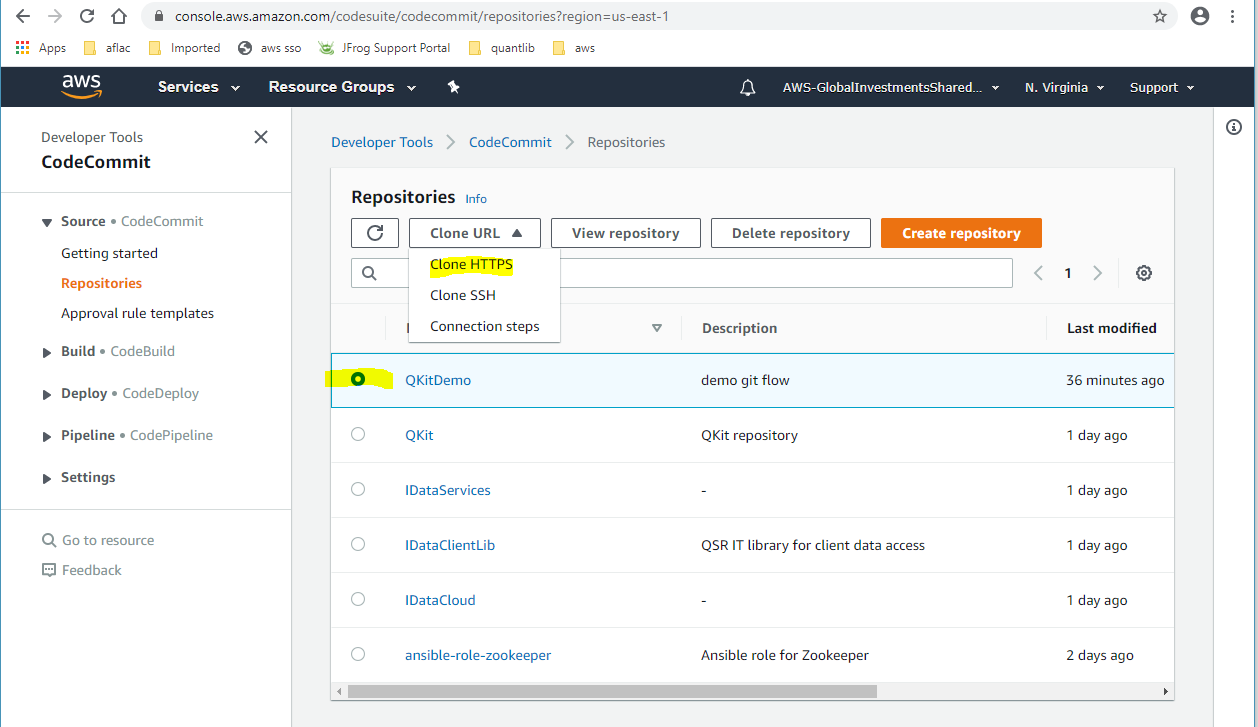


Search “CodeCommit”.



Select “QKitDemo” repository; Click “Clone URL” > “Clone HTTPS” to copy repo URL.

(e.g. <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/QKitDemo> )



b. Open a “Command Prompt” from the folder where you want to place the GIT repo.

(e.g. C:\Users\v11392\Desktop\git\demo)

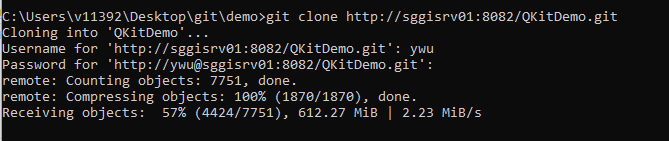
Run commands:

$git config --global http.sslVerify false

$git config --global fetch.prune true

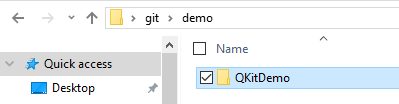
$git clone <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/QKitDemo>

It will ask you to input username and password to process the “git clone”.

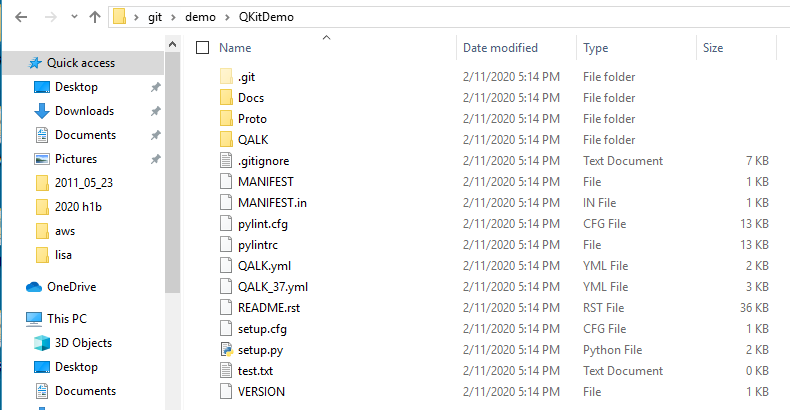


It will take a while because it is downloading repository which contains full git history from CodeCommit (1+ GB).

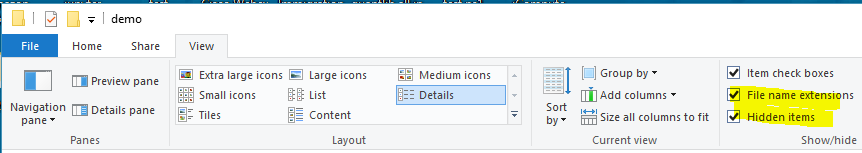
You will get a “QKitDemo” folder once “git clone” is done.



Open “QKitDemo” folder, you should see:



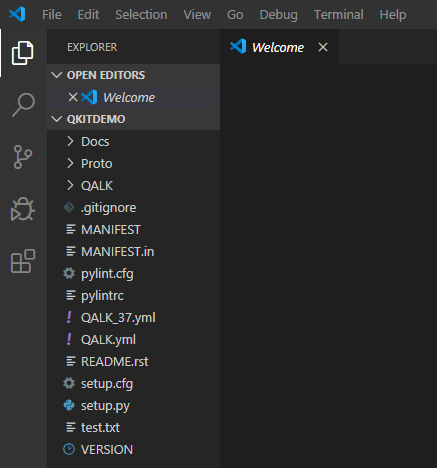
If you cannot see the **.git** folder, please change the folder view options



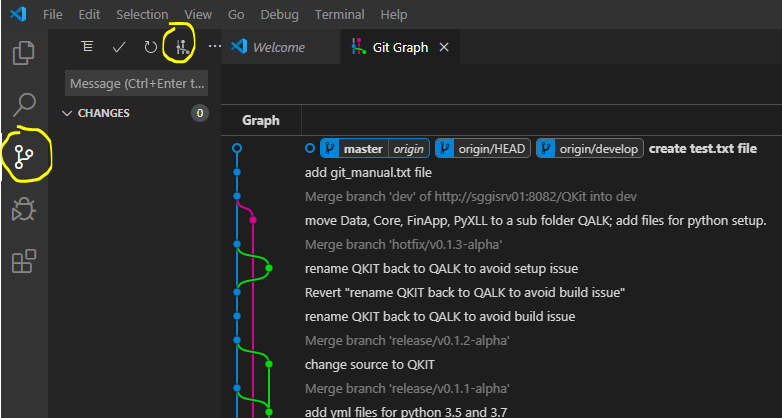
The **.git** folder contains all the information that is necessary for your project in version control and all the information about commits, remote repository address, etc. All of them are present in this folder. It also contains a log that stores your commit history so that you can roll back to history.

The **.gitignore** file is a text file that tells GIT which files or folders to ignore in a project.

Open the “QKitDemo” folder in VS Code.

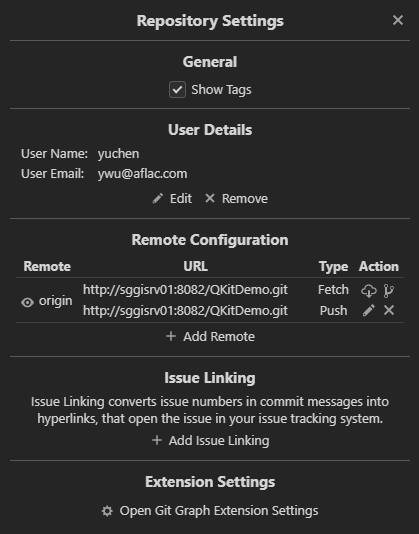


Switch to “Source Control” view; Open “git graph” view.



(You may see any error message when open “git graph” view. That is because “git clone” only fetched “default branch” from remote repository. If the “default branch” is not “master” branch, “git graph” will show an error message. To fix it, you can open VS Code terminal with shortcut “Ctrl + ~”, and then run command “$git checkout master”.)

Click  button (on top right of “git graph” view) to check repository settings.



If your username or email does not look good, please change it.

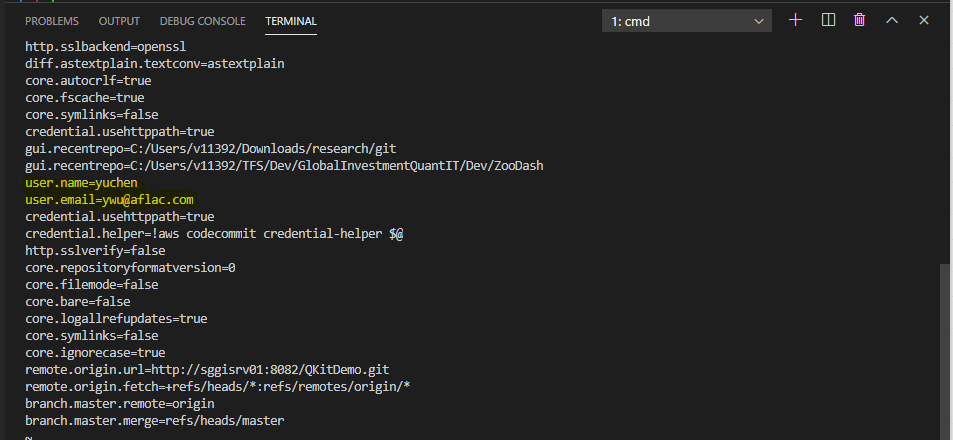
You can do it via “Edit” button. But if you are a GIT beginner, please use GIT commands as much as possible.

Press “Ctrl + ~” to open the terminal in VS Code

Run commands:

$git config --list

You can see your current user.name and user.email.



Run commands:

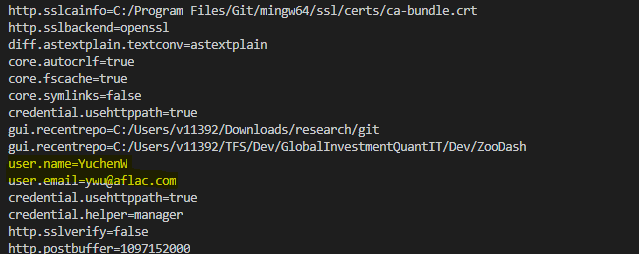
$git config --global user.name YuchenW

$git config --global user.email [ywu@aflac.com](mailto:ywu@aflac.com)

$git config --list

(You can use ‘--local’ instead of ‘--global’ if you want a particular settings for current repository)

You will see the name and email are updated.



Click  button again, you will see the updated “User Details”.

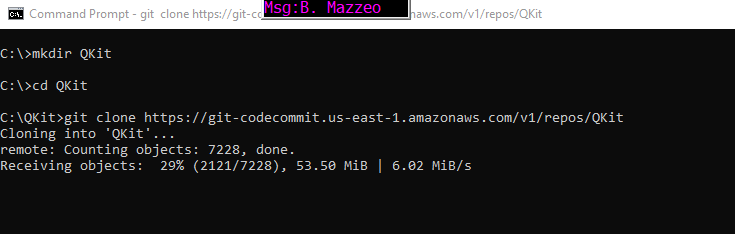
(It is import you provide accurate user name and email. They will be tagged in your future commits, and are not changeable once you submitted commits.)

## Clone QKit Repository

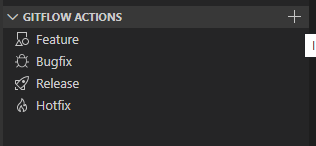
* Open command prompt
* Type:
  + mkdir C:\Repos
  + cd C:\Repos
  + git config --global http.sslVerify false
  + git config --global fetch.prune true
  + git clone <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/QKit>

(Use AWS credentials if a message box pops up. If you don’t have credentials, ask QSR IT team.)

* + cd QKit
  + git checkout master



* go to VS Code and click on “GITFLOW ACTIONS” to initialize repository (see section 2.6 “Initialize gitflow for details)



* type the following lines in the command prompt:
  + git config --global user.name <user name : Yuchen Wu>
  + git config --global user.email < Aflac email : [ywu@aflac.com](mailto:ywu@aflac.com)>
* make sure that you create a directory .vscode under C:\Repos\QKit\QALK
* add launch.json and settings.json from http://sggisrv02:8085/documents/ to .vscode folder
* close and open VS code. Load folder C:\Repos\QKit\QALK
* You need to change your environment variable. Go to MyPC and right click on Properties. Then, Advanced System Settings > Environment Variables > System Variables. Change the following environment variable:

Variable name : QALK\_ALPHA\_LCL\_WRKSPC

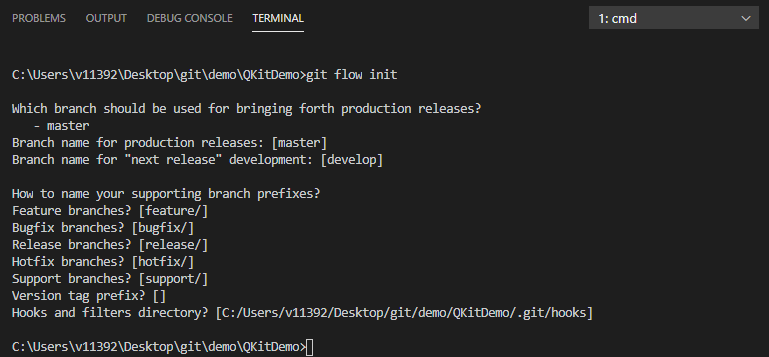
Variable value : C:\Repos\QKit\QALK

## Initialize “git flow”

Run commands in VS Code terminal:

$git checkout master

$git flow init

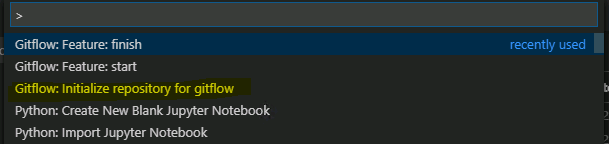


Press “<enter>” button multiple times to use the default configuration.

Equivalent action using “gitflow” plugin:

Open “Command Palette” with shortcut “Ctrl + Shift + p”;

Type ‘gitflow’, then select

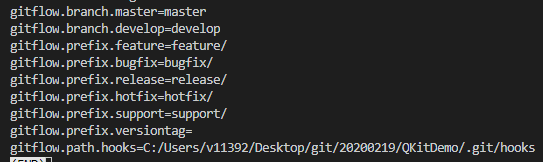


It will ask you to specify “production release branch”, “next release branch”, etc. You can press “<enter>” button to use the default options.

Run commands:

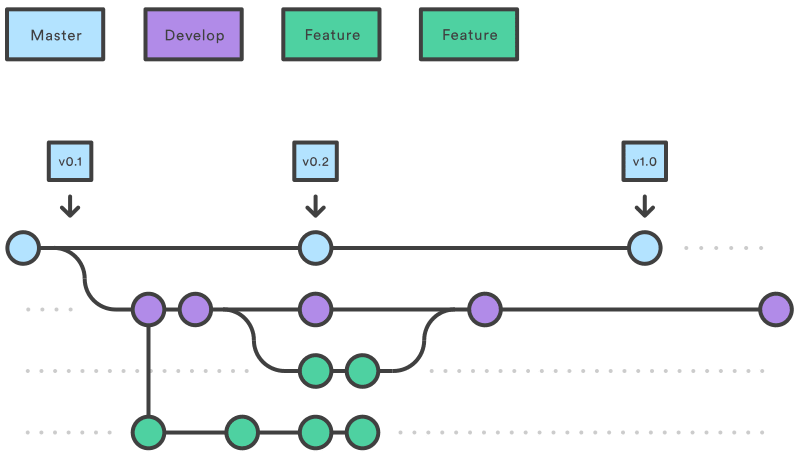
$git config --list

You will see some new configurations.



# Procedure for implementing a feature branch

Feature branches are used to develop new features for the upcoming or a distant future release. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop or discarded.



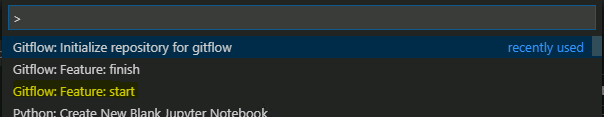
## Create new feature branch

Run commands:

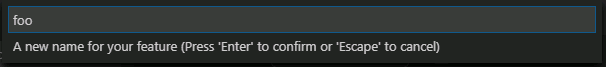
$git flow feature start foo

Equivalent action using “gitflow” plugin:

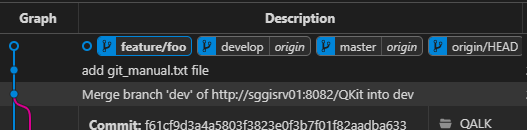
Open ‘Command Palette’; type ‘gitflow’; then select



Type ‘foo’, so the new branch name will be “feature/foo”.



You will see a new branch named “feature/foo” in “git graph”.

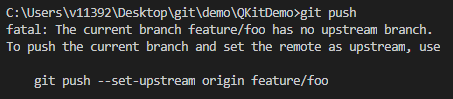


There is no “origin” behind the feature branch name, so it is a local branch.

Run commands:

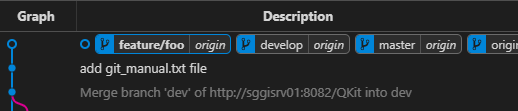
$git push

You will see a fatal error, because you did not specify the upstream remote branch.



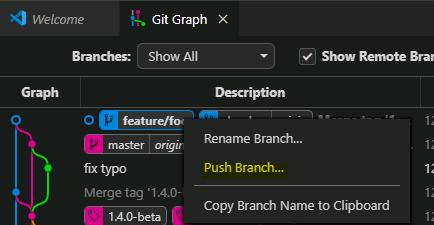
Run commands:

$git push --set-upstream origin feature/foo

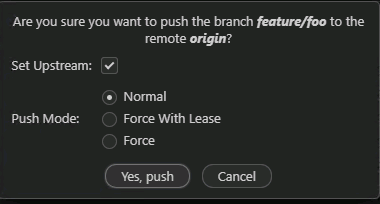


Equivalent actions with “git graph” plugin to do ‘git push’:

Right click ‘feature/foo’ from ‘git graph’, click ‘Push branch…’



You will see a confirmation popup; use the default options; click “Yes, push”

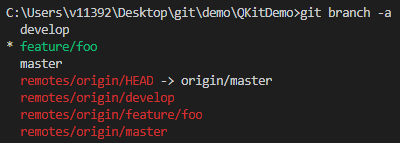


You can see “origin” in the new feature branch name once push is done.

Run commands:

$git branch --all

You will see all local branches and remote branches.

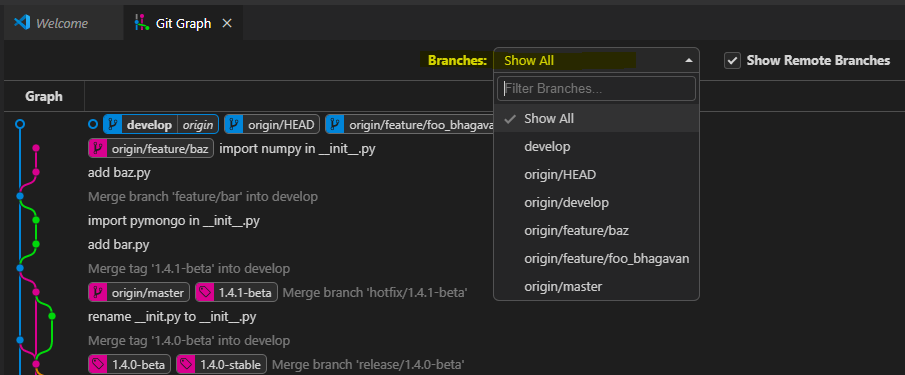


“feature/foo” branch begins with \*, which means you are currently on that branch.

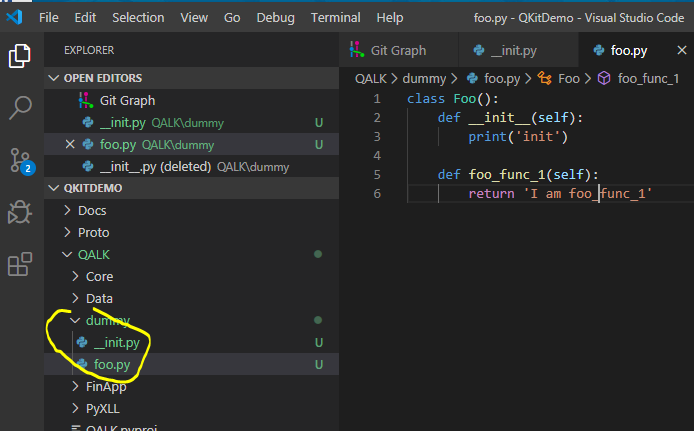
Alternative ways to view all branches:

Use “git graph” plugin.

Click the dropdown list which is in the middle of “Git Graph” view.

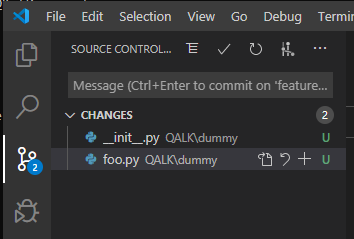


Add some files to source code.



Switch to “Source control” view.

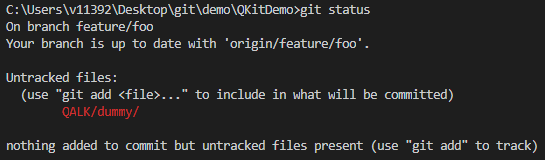
You will see 2 changes.



Those changes are “untracked files”.

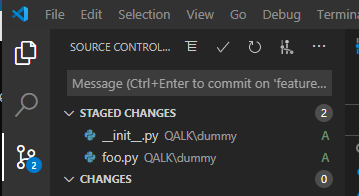
Run commands:

$git status



Run commands:

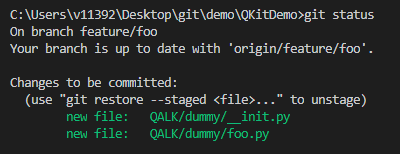
$git add .



Changes are staged but not committed yet.

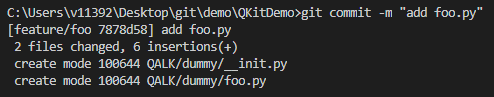
Run commands:

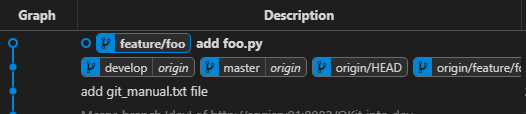
$git status



Run commands:

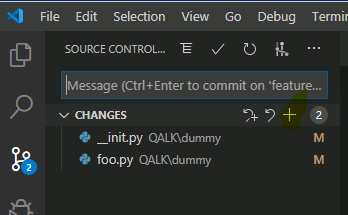
$git commit -m "add foo.py"



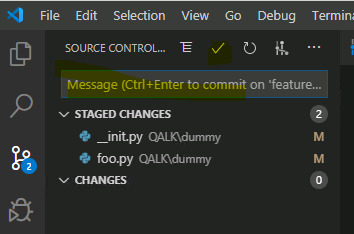


Equivalent actions using VS Code “Source Control”

Click “+” button near “CHANGES” to stage files.

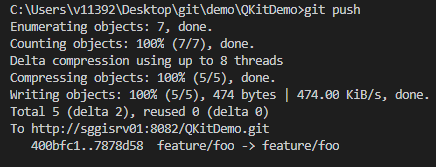


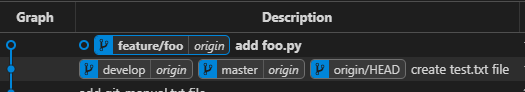
Input message; Click “tick” to commit changes.



Run commands:

$git push





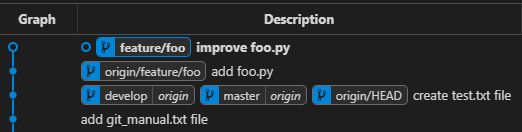
Change foo.py again.

Run commands:

$git add .

$git commit -m “improve foo.py”

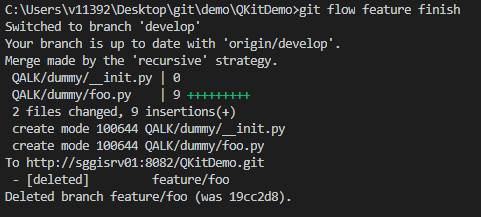
$git push

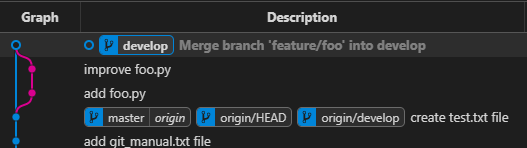


## Merge into the develop branch

Run commands:

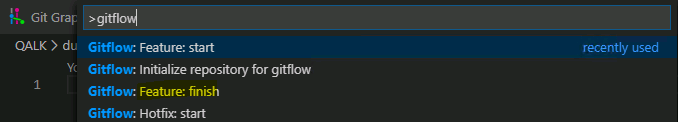
$git flow feature finish





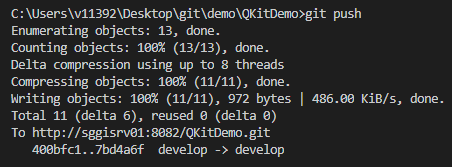
Equivalent action using “gitflow” plugin:

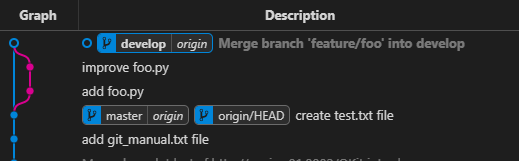
Open ‘Command Palette’; type ‘gitflow’; then select



Run commands:

$git push





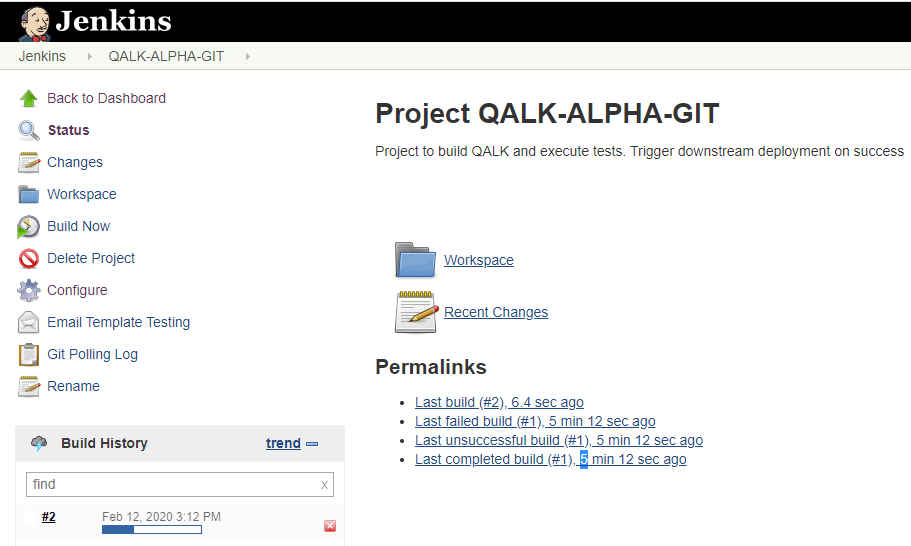
## Build QALK\_ALPHA via Jenkins

Whenever there is new commit in remote “develop” branch, QALK\_ALPHA Jenkins build will automatically start in 5 minutes.

To check Jenkins build

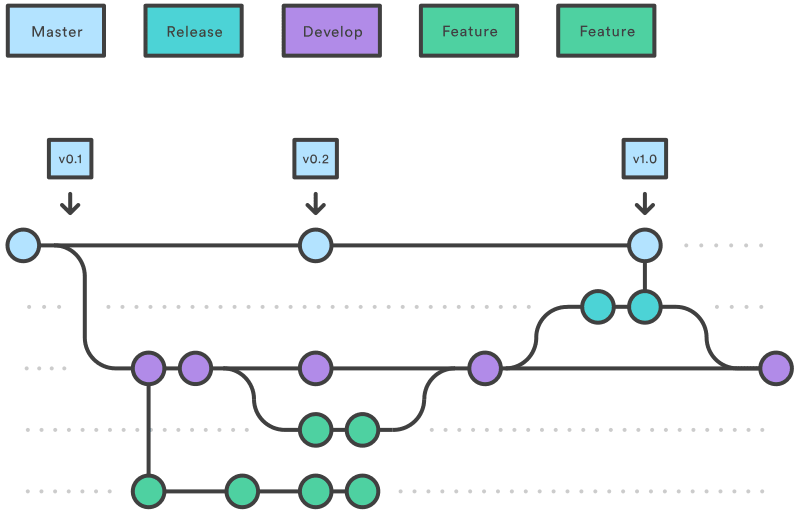
Login Jenkins <http://sggisrv01:8080/> with your E#/V#.

You will see ‘QALK-ALPHA-GIT’ job automatically start in 5 minutes. Please go through the job configuration.



# Procedure for releasing QKit

Release branches support preparation of a new production release. They allow preparing meta-data for a release (version number, build dates, etc.). By doing all of this work on a release branch, the develop branch is cleared to receive features for the next big release.



## Create a release branch for BETA

Run commands: (update develop branch; create release branch; push release branch to remote)

$git checkout develop

$git pull

$git flow release start 1.4.0-beta

$git branch

$git push --set-upstream origin release/1.4.0-beta

Update VERSION file: increase version to 1.4.0

You can also add changelog to readme file if any.

You should not do any other code changes in release branch.

Run commands: (stage changes; commit changes; push branch)

$git add .

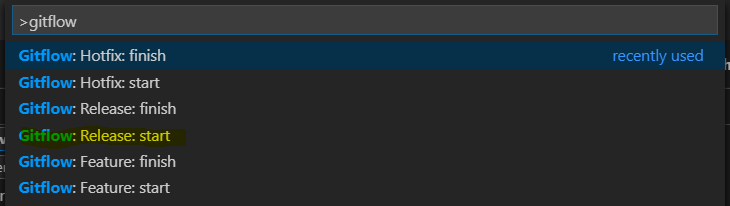
$git commit -m "increase version to 1.4.0-beta"

$git push

Equivalent action using “gitflow” plugin:

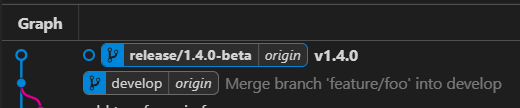
Checkout and pull develop branch via “git graph”;

Open ‘Command Palette’; type ‘gitflow’; then select



Change VERSION file;

Push release branch;



## Merge into the master branch and develop branch

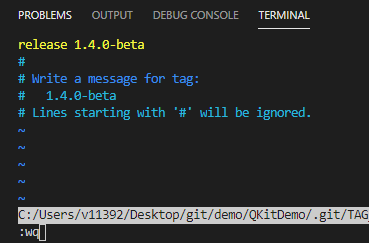
Run commands: (merge release branch to master and develop; close release branch; add tag to master commit)

$git flow release finish

It will open a Vi editor.

Add message: “<esc>” -> “i” -> type a message for the tag;

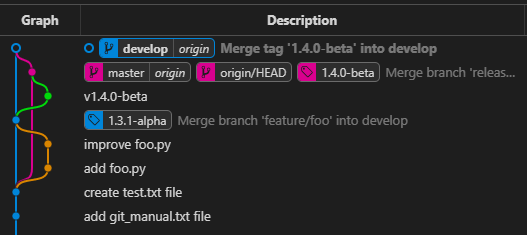
Close Vi editor: “<esc>” -> “:” -> “wq”



Run commands: (push all branches to remote; push all tags)

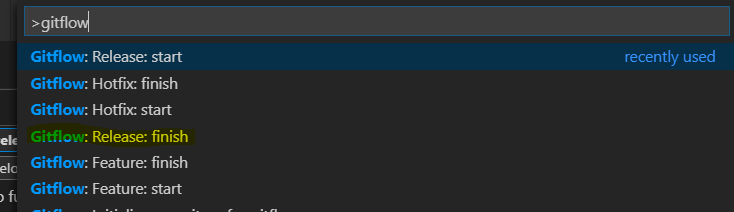
$git push --all

$git push --tags

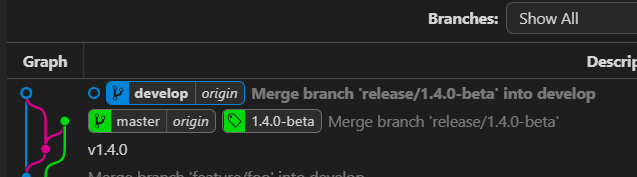


Equivalent action using “gitflow” plugin:

Open ‘Command Palette’; type ‘gitflow’; then select



It will display a dialog for you to input release message. Input a message and press <enter>.



The release branch has been merged into develop branch and master branch. Tag 1.4.0-beta was created. Branches and tag were also pushed to remote.

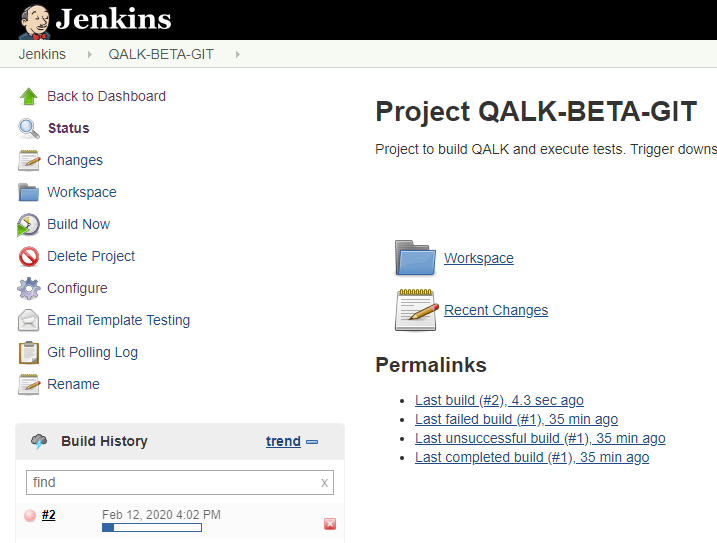
“git graph” does not tell if tag is pushed or not. You can manually push the tag in case it was not pushed.

## Kick-off build for BETA Jenkins build

Different from ALPHA build, BETA build and STABLE are triggered by “tag”. If “tag” name comes with “beta”, BETA build will start. If tag name comes with “stable”, STABLE build will start.

Login Jenkins <http://sggisrv01:8080/>

You will see ‘QALK-BETA-GIT’ job automatically start in 5 minutes after you finish 4.2 (push tag).



## Retag PROD using BETA changeset

You may want to build QALK\_STABLE based on the changeset tagged by BETA.

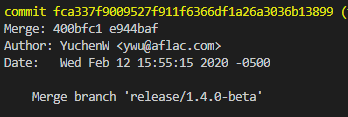
In this case, we will add a “1.4.0-stable” tag to the commit which has been tagged with “1.4.0-beta”.

Tag has to be attached on an existing commit, so you need to find the commit ID first.

Run commands:

$git log master -5

Find the commit ID you want to add tag



Run commands:

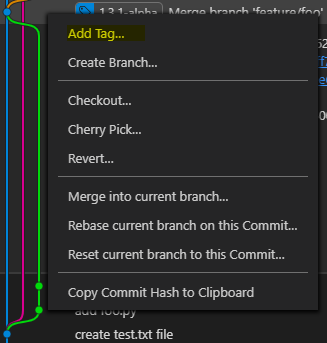
$git tag -a 1.4.0-stable fca337f

$git push origin 1.4.0-stable

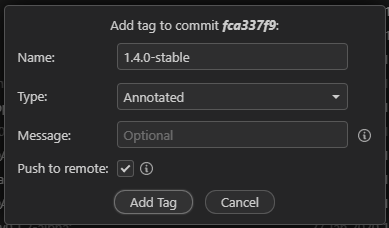
Use VS code plugins (‘git graph” and “gitlens”) is much easier to implement same actions.

For “git graph”: (**Recommend**)

Right click the commit where has been tagged with “1.4.0-beta”.

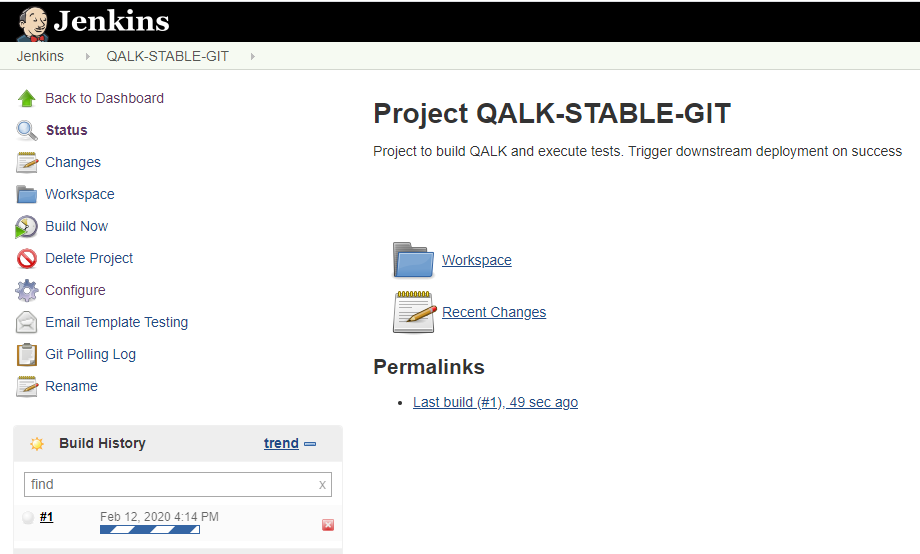


Click “Add tag …” option.



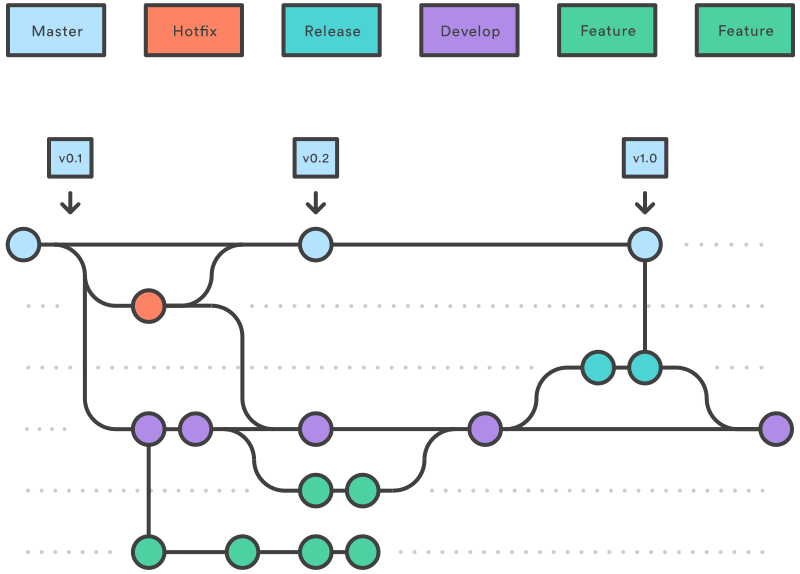
Provide tag name (consistent with BETA tag): 1.4.0-stable; Type: “Annotated”; Select “Push to remote”.

“QALK-STABLE-GIT” job will automatically start in 5 minutes once you pushed tag “1.4.0-stable”.



# Procedure for implementing hot fix

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version. It uses for maintenance or quick patches after release.



## Create a hot fix branch from master

Run commands:

$git checkout master  
$git pull  
$git flow hotfix start 1.4.1-beta  
$git push --set-upstream origin hotfix/1.4.1-beta

Rename \_\_init.py to \_\_init\_\_.py in QALK/dummy.

Change VERSION file to 1.4.1

Run commands:

$git add .  
$git commit -m "rename \_\_init.py to \_\_init\_\_.py"  
$git push

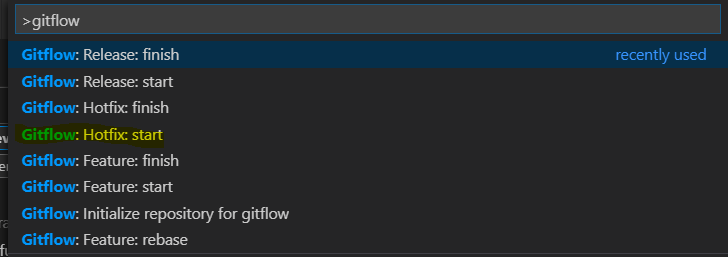
## Merge into the master branch and develop branch

Run commands:

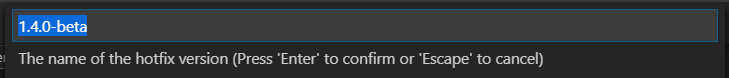
$git flow hotfix finish  
$git push origin --all  
$git push origin --tags

Equivalent action using “gitflow” plugin:

Open ‘Command Palette’; type ‘gitflow’; then select



It will display a dialog for you to input hotfix version. Change it to “1.4.1-beta” and press <enter>.

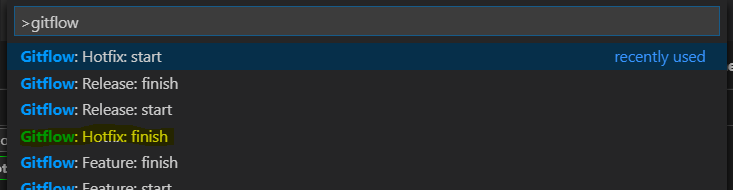




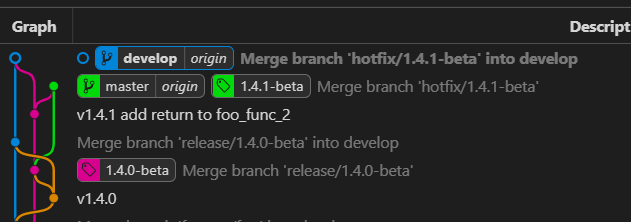
Make a change to file and commit;

Push hotfix branch to remote;

Open ‘Command Palette’; type ‘gitflow’; then select



It will display a dialog for you to input hotfix message. Input a message and press <enter>.



The hotfix branch has been merged into develop branch and master branch. Tag 1.4.1-beta was created. Branches and tag were also pushed to remote.

“git graph” does not tell if tag is pushed or not. You can manually push the tag in case it was not pushed.

# GIT Conflict Resolution

## Mock up

Mock up two developers (AAA and BBB) modified same file and have to merge from feature branch to develop branch.

1. **Mockeup Two Developers**

Open two “CMD” consoles from the folder you want to place repos

Console 1:

Run commands in console 1:

$git clone <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/QKitDemo> devlA

$cd devlA

$code .

Console 2:

Run commands in console 2:

$git clone <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/QKitDemo> devlB

$cd devlB

$code .

You got two VS Code IDEs.

For VS Code - devlA:

Open a terminal with hotcut “Ctrl + ~”.

Run commands:

$git config --local user.name AAA

$git config --local user.email AAA@email.com

$git checkout master

$git flow init -f

For VS Code - devlB:

Open a terminal with hotcut “Ctrl + ~”.

Run commands:

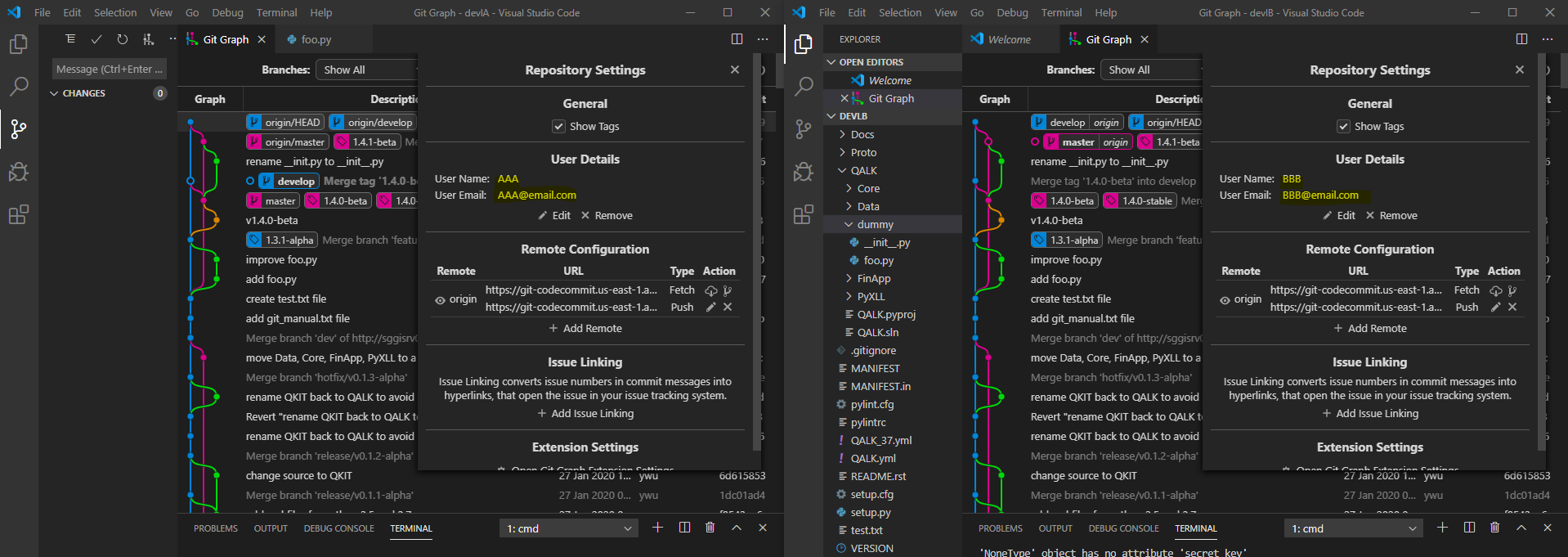
$git config --local user.name BBB

$git config --local user.email [BBB@email.com](mailto:BBB@email.com)

$git checkout master

$git flow init -f

Now, you got two developers (AAA and BBB).



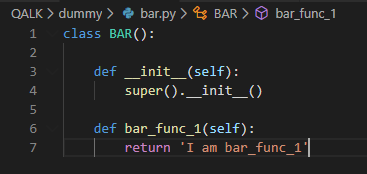
1. **Create conflicts**

Developer AAA:

Run commands:

$git flow feature start bar

Create bar.py in QALK/dummy

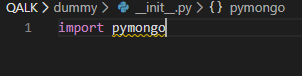


Run commands:

$git add QALK/dummy/bar.py

$git commit -m "add bar.py"

Modify \_\_init\_\_.py in QALK/dummy



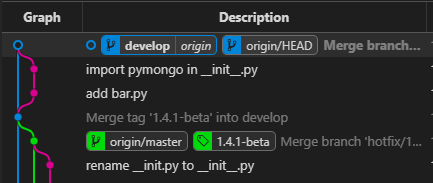
Run commands:

$git add QALK/dummy/\_\_init\_\_.py

$git commit -m "import pymongo in \_\_init\_\_.py"

$git flow feature finish

$git push

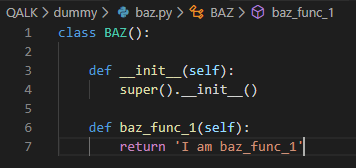


Developer BBB:

Run commands:

$git flow feature start baz

Create baz.py in QALK/dummy

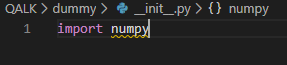


Run commands:

$git add QALK/dummy/baz.py

$git commit -m "add baz.py"

Modify \_\_init\_\_.py in QALK/dummy (Develop AAA has changed \_\_init\_\_.py, there will be a conflict)

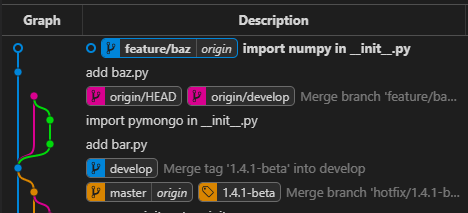


Run commands:

$git add QALK/dummy/\_\_init\_\_.py

$git commit -m "import numpy in \_\_init\_\_.py"

$git fetch --all



You can see your local develop branch is behind origin/develop.

Run commands:

$git checkout develop

$git pull

$git checkout feature/baz

$git flow feature finish

There are multiple strategy to deal with conflicts, please see next 3 sections.

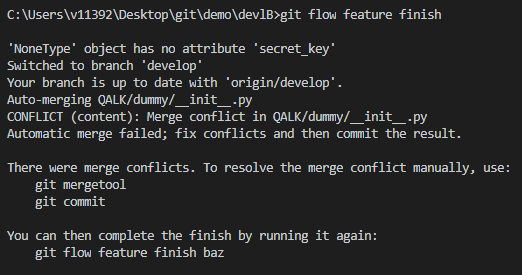
## Strategy 1: Merge feature branch into develop branch (GIT Merge)

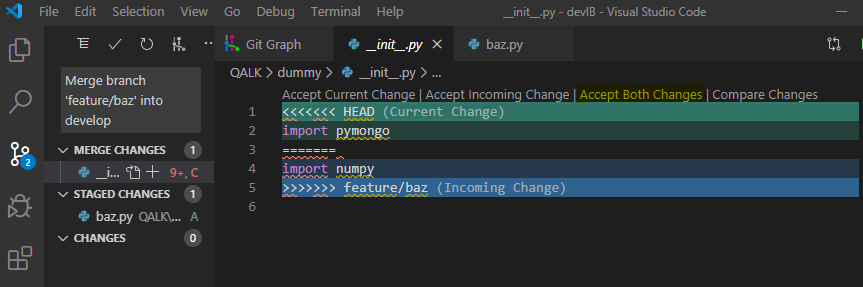
If BBB finished the changes in feature branch, he can merge the feature branch into develop branch.

Developer BBB:

Run commands:

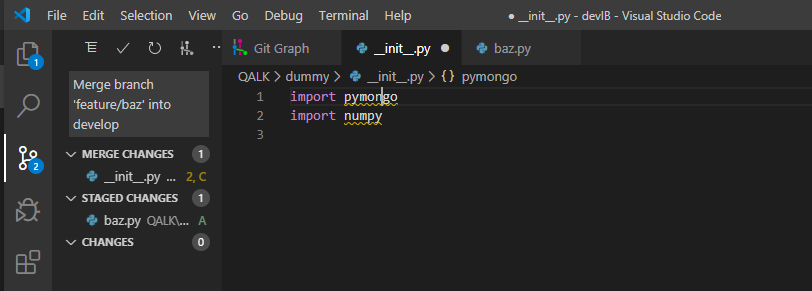
$git flow feature finish





You will see the merge changes in “Source Control” view.

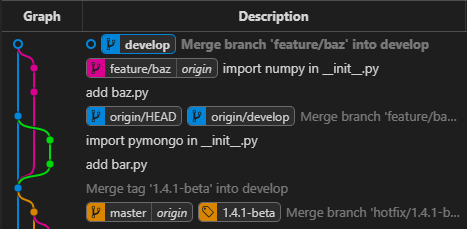
Click “Accept Both Changes” and save the file.



Run commands:

$git add .

Click  to commit changes.



Run commands:

$git push

$git branch -d feature/baz

## Strategy 2: Merge develop branch into feature branch (GIT Merge - Recommend)

If BBB wants to continue developing feature branch on top of the latest changes from develop branch, he can merge the latest develop branch into feature branch.

Developer BBB:

Run Commands:

$git checkout feature/foo

$git merge develop

Resolve the conflict.

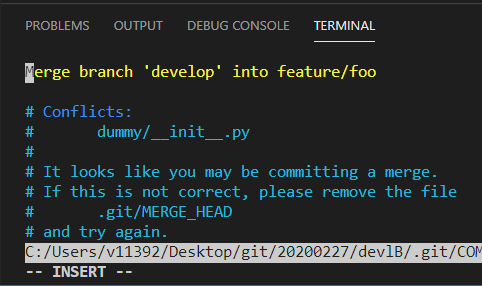
Run commands:

$git add .

$git merge --continue

A Vi Editor will display in terminal with default merge message.

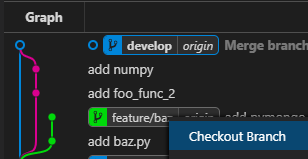
Use “<esc>” -> “:” -> “wq” to save the merge message.



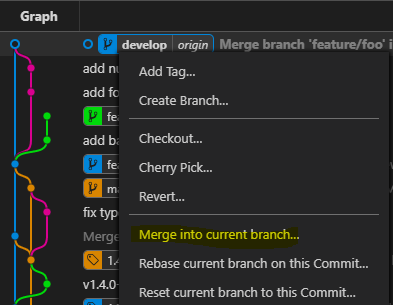
Then you can continue developing feature/baz. Or use “git flow feature finish” to merge feature branch into develop branch.

Equivalent actions using “git graph”

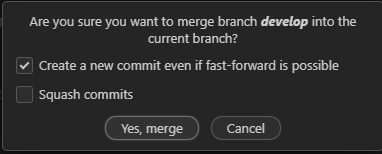
Right click the feature branch you are working on; Select “Checkout Branch”.



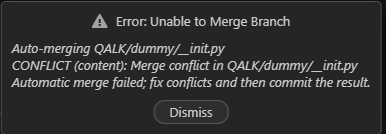
Right click the “develop | origin” branch; Select “Merge into current branch”.



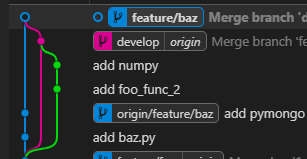
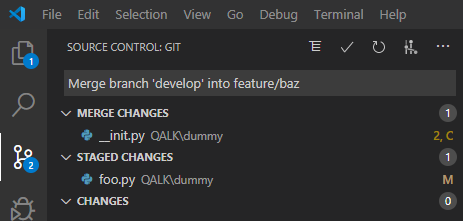
You will see a popup about how you want to commit; Use the default strategy; Click “Yes, merge”.



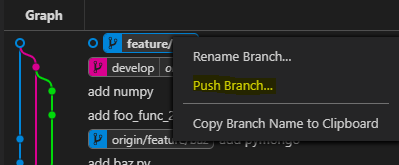
You will see “Error” due to the conflict. Click “Dismiss” to start merge.



You will see something like below images. Double click files under “MERGE CHANGES” to resolve conflicts. Once you are done, click the “+” button near “MERGE CHANGES” to stage files. Then you can click “Tick” button near “SOURCE CONTROL: GIT” to commit.



Push the feature branch to remote again.



## Strategy 3: Rebase feature branch onto develop branch (GIT Rebase)

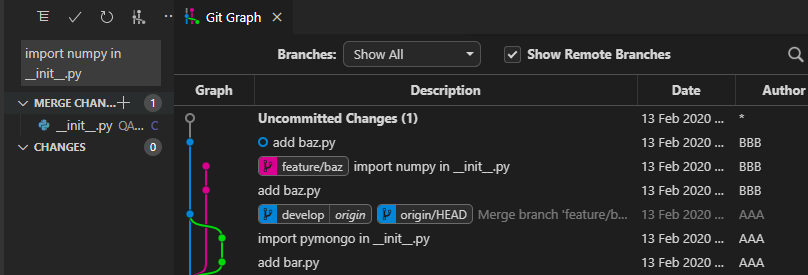
If BBB wants to move feature branch to start from the latest commit from develop branch.

Developer BBB:

Run Commands:

$git checkout feature/baz

$git rebase develop



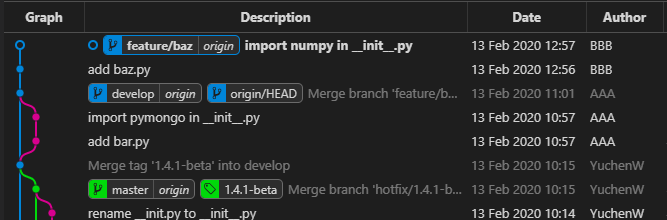
Resolve the conflict.

Run commands:

$git rebase --continue

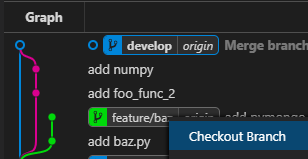
A Vi Editor will display in terminal with default merge message.

Use “<esc>” -> “:” -> “wq” to save the merge message.

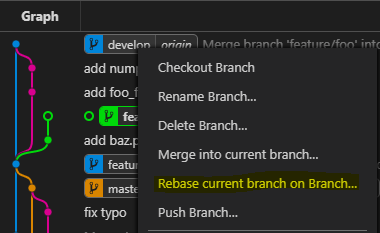


Equivalent actions using “git graph”:

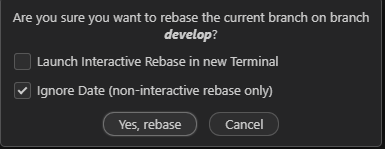
Right click the feature branch you are working on; Select “Checkout Branch”.



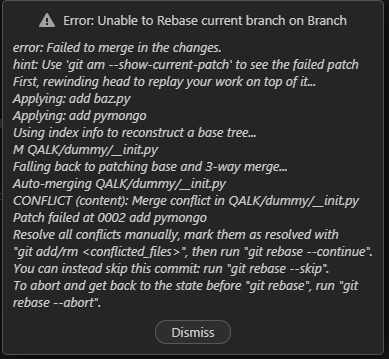
Right click the “develop | origin” branch; Select “Rebase current branch on Branch…”.



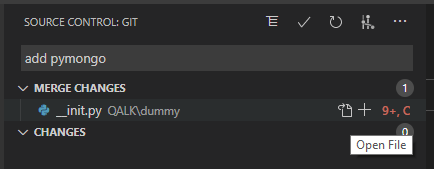
You will see popup; Use the default options; Click “Yes, rebase”.



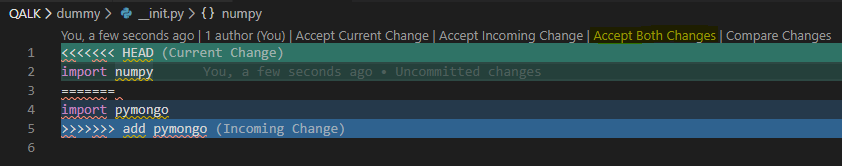
You will see “Error” dialog due to the conflict; Click “Dismiss” to start merge;



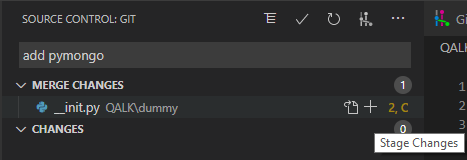
You will see “MERGE CHANGES” in “SOURCE CONTROL” view.



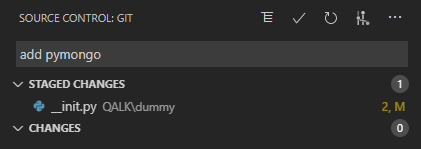
Double click the files under “MERGE CHANGES” and resolve conflicts.



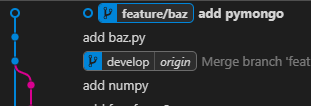
Click “+” button near “MERGE CHANGES” to stage changes.



Click “Tick” button near “SOURCE CONTROL: GIT” to commit.



Push the feature branch to remote again.



# Other GIT Features

There are some other useful git features like reset, diff, stash, log, reflog, show, etc. This chapter will introduce how to implement diff and undo actions.

## Diff

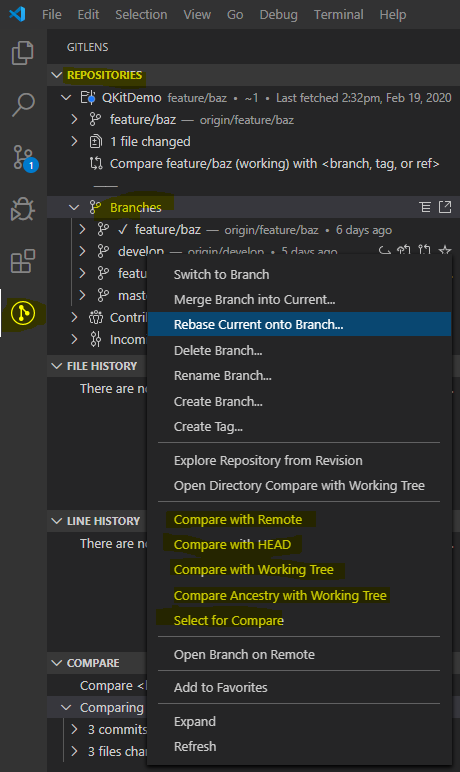
You may want to view commit history, and check what has been changed between files, commits, branches, etc.

“GitLens” plugin provides cool feature to implement “git diff” easily.

**Compare Branches:**

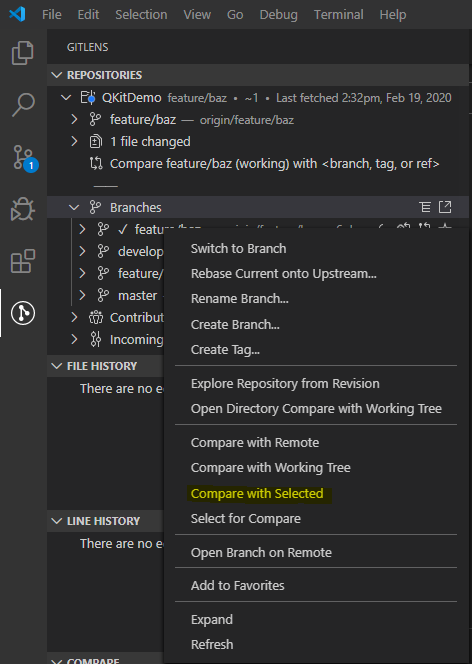
Switch to “GitLens” view; Expand “REPOSITORIES” -> “Branches”. (If you do not see all branches, please checkout to your local first.)

Right click the branch which you want it to be “source” branch for comparison.

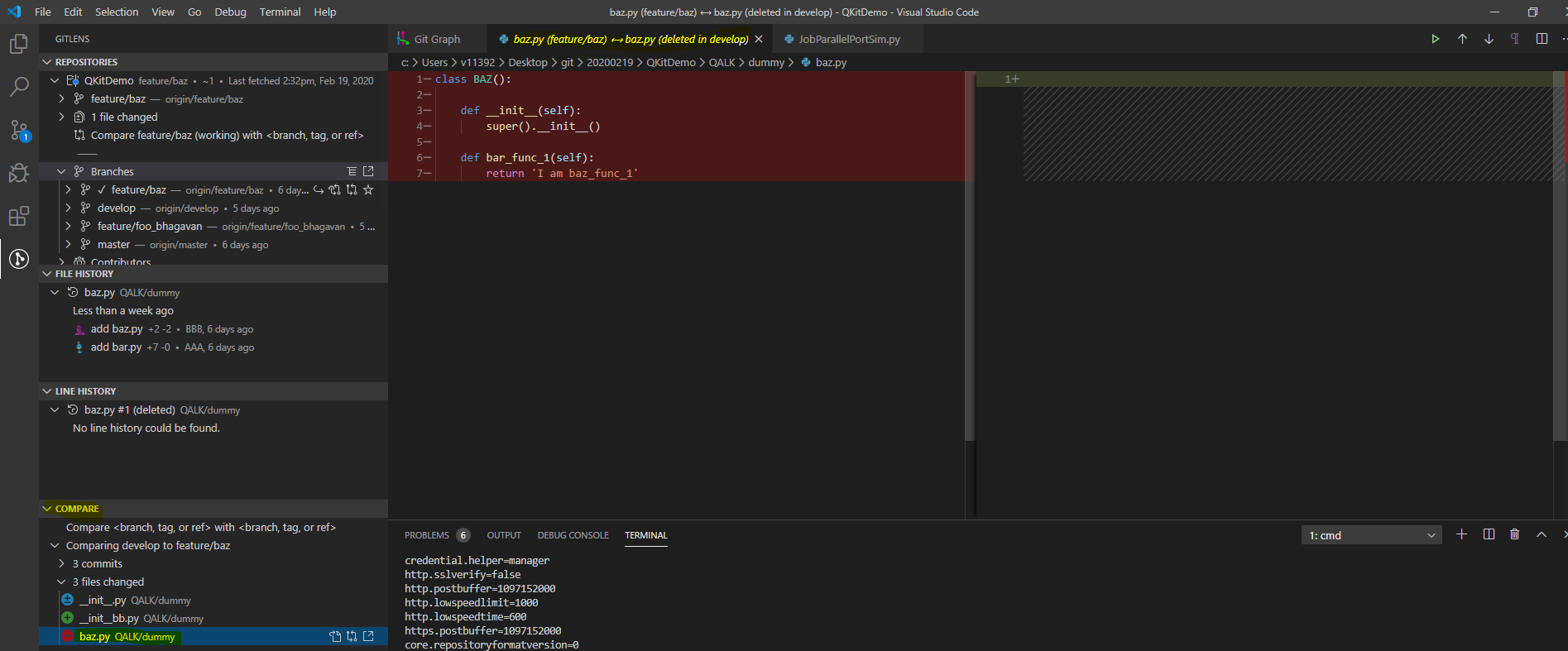


There are 5 compare options. Let us try “Select for Compare”.

Right click another branch which you want to use as “target” branch. For example, “feature/baz” branch.



Click “Compare with Selected” option. You will see all differences in “COMPARE” section.



Double click a file, you will see the difference.

**Compare Commits:**

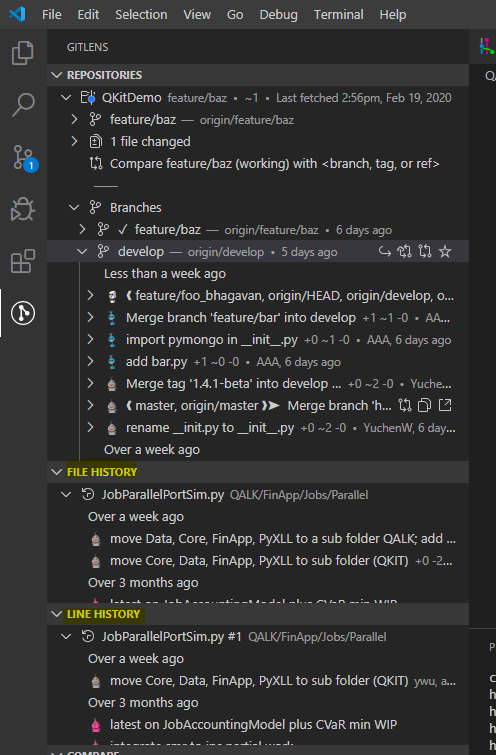
Similar to “Compare Branches”, you can pick commits from “REPOSITORIES” -> “Branches”, and perform comparison.

**Compare Files:**

Open a file from “Explorer” view. For example, “JobParallelPortSim.py”.

Switch to “GitLens” view.

You will see lots of commits in “FILE HISTORY” and “LINE HISTORY”.



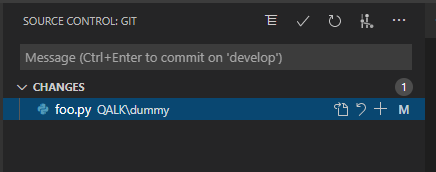
Similar to “Compare Branches”, you can pick files for comparison.

## Undo

There are two terms to implement “undo” action.

* Revert (Rollback files, create a new commit, will not change existing commit history)
* Reset (Point HEAD to old commit, throw away newer commits)

**For “uncommitted changes” (unstaged or staged:**



**Approach 1:**

Run commands:

$git checkout -- QALK/dummy/foo.py

**Approach 2:**

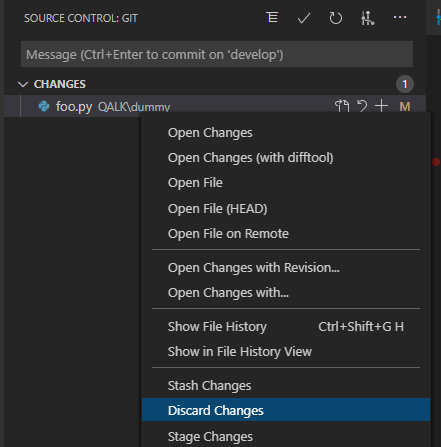
Run commands:

$git stash

$git stash drop

**Approach 3 (GUI):**

Right click the file, select “Discard changes” option. If file is staged, do unstage first then discard.



**For committed changes:**

**Approach 1:**

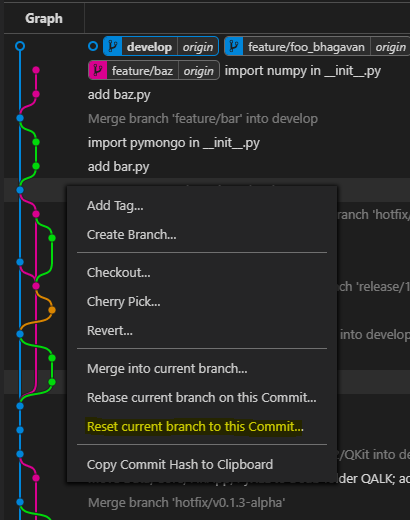
Run commands:

$git reset --hard HEAD~1

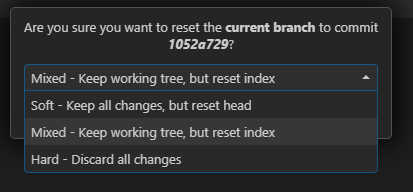
**Approach 2 (GUI):**

Open “git graph” view.

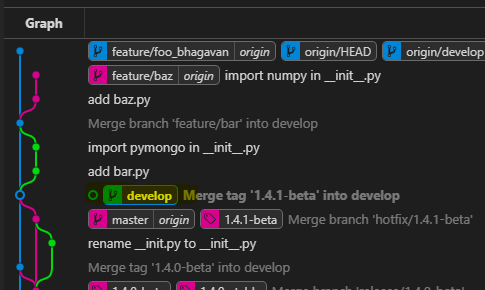
Right click the commit where you want to reset HEAD back to.



Select “Reset current branch to this Commit …” option. You will see a dialog with a drop-down list.



Let’s pick “Hard-Discard all changes” here.



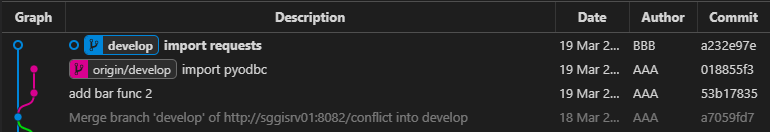
Your local HEAD has been reset. You can push it to remote. (Reset on “develop” branch is not a good practice, it may impact other developers’ work. You should try to avoid.)

# More

## Conflicts when multiple developers working on same branch

Conflicts may happen when multiple developers modified same source code.

For example,



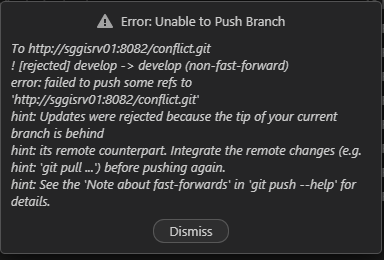
Developer AAA and BBB are both working on “develop” branch on top of commit id “a7059fd7”.

Developer AAA modified base.py (import pyodbc), then he committed and pushed develop branch to remote repository.

Developer BBB also modified base.py (import requests), then he committed changes and is going to push.

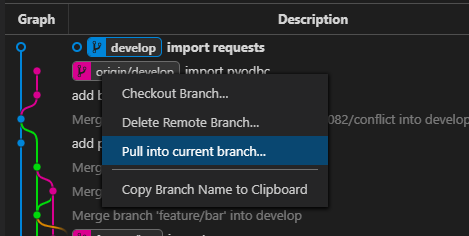
As you can image, there will be a conflict on base.py.

If BBB pushes the develop branch, there will be an error message

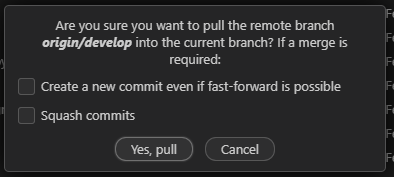


So BBB has to resolve the conflict before push.

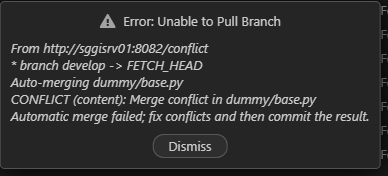
Right click the remote develop branch (origin/develop), and select “Pull into current branch”.



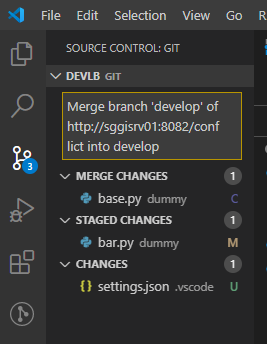
Click “Yes, pull” with default options.



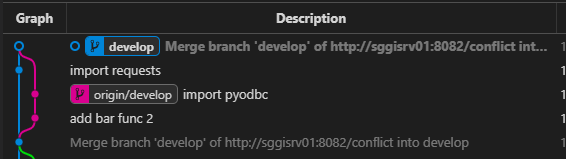
An error message will popup, and it tried to perform “automatic merge”, but failed.



Click “Dismiss” to perform “merge” manually. You will see pending changes in “Source Control” view.



Resolve the conflicts and commit the change.



Do not forget to push the “develop” branch.

# References

GIT: <https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/>

GIT Object Model: <http://shafiul.github.io/gitbook/1_the_git_object_model.html>

GIT Index: <https://hackernoon.com/understanding-git-index-4821a0765cf>

GIT Flow: <https://jeffkreeftmeijer.com/git-flow/>

GIT vs GIT Flow: <https://gist.github.com/JamesMGreene/cdd0ac49f90c987e45ac>

GIT Rebase vs Merge: <https://www.atlassian.com/git/tutorials/merging-vs-rebasing>

GIT Stash: <https://www.atlassian.com/git/tutorials/saving-changes/git-stash>

GIT Diff: <https://www.atlassian.com/git/tutorials/saving-changes/git-diff>

GIT Undo: <https://www.atlassian.com/git/tutorials/undoing-changes>

GIT Reset vs Revert: <https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting>

GIT Cherry Pick: <https://www.farreachinc.com/blog/far-reach/2017/03/16/git-cherry-pick-why-and-how>

GIT Habits: <https://datagoodie.com/blog/git-simple-tutorial-explanation-LEVEL-4/>