**UHOne Facets**

**Git Feature Branching Strategy & Git Steps**

Version 1.0

12/13/2016

# 

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Section | Description of Changes | Authors |
| 11/16/2016 | 0.1 | All | Initial Strategy | Sandy Krishnan |
| 11/25/2016 | 0.2 | Git Steps | Added Git Steps | Himanshu S Srivastava |
| 12/13/2016 | 1.0 | All | Reviewed with Vince & minor updates | Sandy Krishnan |
| 03/06/2017 | 1.1 | All | Specific Release branch Names were removed based on Vince Input | Sandy Krishnan |

[Revision History 2](#_Toc476595255)

[1. Introduction 4](#_Toc476595256)

[a. Purpose 4](#_Toc476595257)

[b. Pre-requisite – Git Install 4](#_Toc476595258)

[c. Git Training/Reading 4](#_Toc476595259)

[2. UHOne Git URL & Secure Access 4](#_Toc476595260)

[a. UHOne Git URL 4](#_Toc476595261)

[b. Git Access 5](#_Toc476595262)

[1. Feature Branching Overview 6](#_Toc476595263)

[2. GitFlow ‘Feature Branching’ 6](#_Toc476595264)

[a. Various Branches in GitFlow ‘Feature Branching’ 7](#_Toc476595265)

[6. Various WorkFlows 9](#_Toc476595266)

[a. Developer’s WorkFlow 9](#_Toc476595267)

[b. Release Branch WorkFlow 9](#_Toc476595268)

[c. HotFix Branch WorkFlow 10](#_Toc476595269)

[7. WorkFlows – Detailed Git Steps 11](#_Toc476595270)

[a. Git Steps for Developer’s workflow 11](#_Toc476595271)

[b. Git Steps for Release’s workflow 15](#_Toc476595272)

[c. Git Steps for Hotfix’s workflow 19](#_Toc476595273)

[8. Misc Git Steps 20](#_Toc476595274)

[a. When and How to rebase.. blah, blah… 20](#_Toc476595275)

[b. When and How to do git reset blah blah 20](#_Toc476595276)

[c. When and How to check git status blah blah 20](#_Toc476595277)

[d. When and How to do git fetch blah blah 20](#_Toc476595278)

[9. Troubleshooting tips 20](#_Toc476595279)

[a. My build failed during the merge, but it worked fine on my branch! 20](#_Toc476595280)

[10. Git Help/Additional Websites 20](#_Toc476595281)

**UHOne – Git Install & Feature Branching Guide**

# Introduction

## Purpose

This document provides the guidance for installing Git and how to work with Git for the Developers. It also mainly explains about the Feature Branching Strategy and various workflows to be followed for different branch.

## Pre-requisite – Git Install

Client software can be requested from the App Store. Direct links to our software in the App Store can be found below. Always, check for latest version available in appstore.

* [Git 2.7.0](http://appstore.uhc.com/AppInfo/AppVersionId/12090) - The Git command line client.
* [TortoiseGit 1.8.16.0](http://appstore.uhc.com/AppInfo/AppVersionId/12092) – A graphical Git client that integrates with Windows Explorer. If you request this you will also receive the command line client.

## Git Training/Reading

To help understand the Feature Branching strategy better, it is a good idea to take the below training and read the article.

1. Go to [Learn Source](https://hrdirect.uhg.com/LearnSource) 🡪Take Courses

**Beginner Git (314045)  - Must take**

**Advanced Git (314265)  - Nice to take; more techniques are covered**

1. <https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>

# UHOne Git URL & Secure Access

## UHOne Git URL

**CodeHub Git Site**: <https://codehub.optum.com/uhonefacets>

**Dev Git Clone URL**: [https://<msid>@codehub.optum.com/uhonefacets/uhone-core.git](https://%3cmsid%3e@codehub.optum.com/uhonefacets/uhone-core.git)

**QE Git Clone URL**: https://<msid>@ codehub.optum.com/uhonefacets/uhone-atdd.git

## Git Access

#### Request access to an existing repository

If you found an existing CodeHub repository but don't have the appropriate access to it yet, you need to request access. There are six types of access in CodeHub.

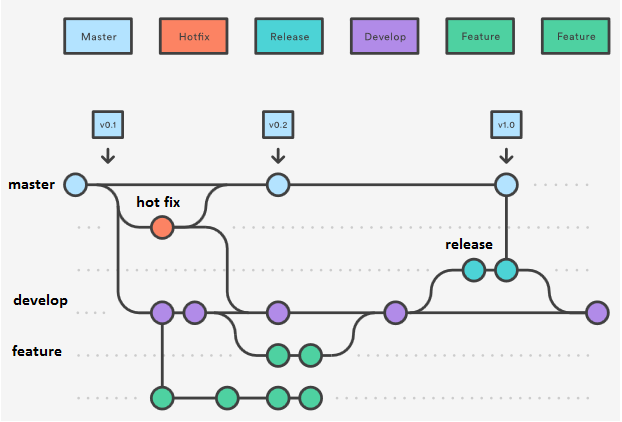
1. Admin users — CodeHub organization admin users have write access to all of the repositories within the organization. They also have the ability to change the list of admin users or make any other access or configuration changes for the organization's repositories.
2. Read users — Each repository can have a list of read users. These users have read access to the repository.
3. Write users — Each repository can have a list of write users. These users have both read and write access to the repository.
4. Read groups (UHO\_GIT\_RO) — Each repository can have a list of read groups. These are Windows groups whose members have read access to the repository. Windows group membership can be determined using the [NT Lookup](http://ntlookup.uhc.com/) site.
5. Write groups (UHO\_GIT\_RW) — Each repository can have a list of write groups. These are Windows groups whose members have both read and write access to the repository. Windows group membership can be determined using the [NT Lookup](https://codehub.optum.com/orgs) site.
6. Approver users (SEE BELOW) — Approvers are write access users who also receive email notifications about and approve Merge Requests. Approvers can be set at either the organization or repository level
7. Approver for Master Branch (**Repo Owner**) – BPD Team for FSG; UHOne Team for UHOne
   1. Approves merge request to Master
   2. Merge can either come from Release or HotFix Branch
8. Approver for Release Branch (**Release Branch Owner**) - Release Tech Lead –
   1. Creates merge request to Master after Production Deployment
      1. Confirms all the Released code is intact and tag is created by Jenkins with version# before merge request.
      2. If tag not present, he manually creates it for this Release.
   2. Merges to Develop after Production Deployment
   3. Approves merge request to Release Branch(if any merge comes from HotFix)
9. Approver for Hot Fix Branch (**HotFix Branch Owner**) - HotFix Tech Lead –
   1. Creates merge request to Master after Production Deployment
      1. Confirms all the production code is intact and tag is created by Jenkins with version# before merge request.
      2. If tag not present, he manually creates it for this Hotfix.
   2. Merges to Develop directly or Release Branch via Merge Requests after Production Deployment

# Feature Branching Overview

Feature branching is a branching methodology where each feature (or User Story) is isolated on its own branch in the VCS (Git or Subversion). There are many well written articles that explain the methodology in great detail including advantages, tradeoffs, and even its application to CI. For those interested reading more please refer to the [Comparing WorkFlows](https://www.atlassian.com/git/tutorials/comparing-workflows) website. **For UHOne**, we will follow **GitFlow** Workflow for Feature Branching/Merging technique (Refer <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>) . The purpose of this strategy is to educate developers on how we apply the feature branching methodology to UHOne using Git.

# 

# GitFlow ‘Feature Branching’



## Various Branches in GitFlow ‘Feature Branching’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Branch Name** | **Description** | **In terms of current FSG SVN** | **Rules of the Road** | **Continuous Integration** |
| master | Production Code – Stable Branch - **Centralized Repo** | 500.0.0.x | - Nobody commits (or pushes) directly to Master branch.  - Merging back to the master branch is done via [merge requests](https://codehub.optum.com/help/articles/merge-request-basics).   * Master is merged from the Release Branch and HotFix Branch * Merge Requestor - Release Tech Lead/Hot Fix Tech Lead * Merge Approver – Repo Owner * Before the Merge to Master happens, every release will be tagged with Release Number (automation with Jenkins during Release Process) | NO  - No build pipeline with Jenkins. |
| develop | Development Integration Code (branched off from Master) - **Centralized Repo** | 500.0.1.x / 500.0.2.x | - ‘develop’ Branch serves as the integration branch for Features  - It is branched off from Master initially and kept updated.  - The develop branch can get updates/merges from Feature or Release branch or hot fix branch. (No Merge Requests to Develop required)  - Jenkins will do builds on commit to this branch. | YES  - Build pipeline with Jenkins to build ‘SnapShot Releases’ and to run auto test cases on check-ins |
| feature | Short lived story specific branch (branched Off from Develop)   * **Local Repo (Optional Centralized Repo)** | No Specific Branch for Feature / User Stories | * All local development is done on a branch called Feature branch. * Feature Branch use ‘develop’ as their parent Branch and not branched off of Master * ‘Feature branch’ gets merged back to Develop Branch * Feature should never directly interact with Master branch * Feature branch need not be centralized, unless developer needs a remote backup or wants to share their code with other developers * If centralized, Feature Branch should be deleted after non usage of 14 days (for example)   **Branch Naming Rule** - Branch names need to start with either **US\*.** Branch names should not contain any characters other than A-Z, a-z, 0-9, - or \_.  **Example story branch**: US12345-membership\_enhancements | NO  - No build pipeline with Jenkins. Only used for developer collaboration and to test locally. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| release | Short Lived Dedicated Release Branch (branched off from Develop just before Release)   * **Short lived Centralized Repo** | No Specific Branch for Releases | * It is branched off from Develop just before Release * Release Branch is created once Develop has acquired enough features for a Release or when the Release Date is approaching. * When Release Branch is created, the next Release cycle starts (No Features are added at this point; only bug fix and release oriented tasks are worked in this branch) * Once Release Branch is ready to ship, the release gets merged into Master and Develop Branch * The only merge release branch should take from HotFix branch (if there was any production issues identified during the release). * Merging to Release branch is done via [merge requests](https://codehub.optum.com/help/articles/merge-request-basics)   + Release Branch is merged only from HotFix Branch (if there is any)   + Merge Requestor - Hot Fix Tech Lead   + Merge Approver – Release Tech Lead | YES  - Build pipeline with Jenkins to build final ‘Release Version’  - A tag will be created with appropriate version number when build&Deploy |
| Hot Fix | Short Lived – used for Production Defects.  (It is forked off of Master)   * **Short lived Centralized Repo** | 500.0.99.x | - “hotfix” branches are used to quickly patch production releases  - It is forked off of Master | YES (Not tied with Jenkins initially).  - Later, pipeline with Jenkins to build final ‘Release Version’  - A tag will be created with appropriate version number when build&Deploy |

# Various WorkFlows

## Developer’s WorkFlow

**Action: Feature🡪Develop Branch**

**Actors**: Feature developers

1. Developer creates local feature branch from the upstream branch ‘develop’
   * **For example, US12345-membership\_enhancements**
2. Developer codes his user story in his local feature branch.
3. Developer pushes the local feature branch to the remote codehub. (makes it visible to other developers) – **Optional/not required**
4. Developer keeps his local code synchronized very often with upstream changes that are committed by other developers (git pull origin develop)
5. Developer resolves conflict between develop and feature branch locally.
6. Developer thoroughly tests locally in local branch and does maven build and pushes the changes from local ‘feature branch’ to remote develop.
7. Developer deletes his Feature Branch (locally and in remote) after the merge to remote Develop (may be after a week or two) , **applicable only if present in remote**
8. Developer repeats these 7 steps for each user story/feature.
9. Jenkins ‘Build job’ kicks off the build on develop branch (on commits), run automated unit tests and creates a snapshot version of the artifact to be deployed. (Example; **UHOne\_MemberOutbound-1.0.0-SNAPSHOT.jar)**

## Release Branch WorkFlow

**Action: Develop Branch🡪Release Branch🡪Master & Release Branch🡪Develop Branch**

**Actors**: Release Team developers, Release Branch Owner (Release Tech Lead) and the Repo Owner

* 1. Release branch is the dedicated branch to prepare releases
  2. A dedicated Release Team and a Release Tech Lead is identified
  3. Release Branch is created **by the Release Tech Lead** once develop has acquired enough features for a Release or when the Release Date is approaching
     + When Release Branch is created forked from the develop branch, the next Release cycle starts (No Features are added at this point; only bug fix and release oriented tasks are worked in this branch)
  4. Release Team needs to be aware that a HotFix branch can get created if there are any production issues. If so, those changes are also merged in to Release Branch after the hotfix goes to production.
  5. If any merge requests comes from HotFix branch via [merge requests](https://codehub.optum.com/help/articles/merge-request-basics), Release Tech Lead verifies and approves them.
  6. The Release Team developers make the changes in his local release branch and push his/their code changes to remote Release.
  7. Jenkins ‘Build/Release’ job kicks off the build on the release branch (on commits), runs automated unit tests, create a release version of the artifact to be deployed. **(ex; UHOne\_MemberOutbound-1.0.0.jar)**
  8. Jenkins creates a tag **(ex; ReleaseName-Version#)** and saves the release version of the source code in the remote codehub repository.
  9. Once the code is deployed in Production, Release Tech Lead creates a tag for this Release (incase, Jenkins automation didn’t create one)
  10. **Release Tech Lead** then request the merge to master via [merge requests](https://codehub.optum.com/help/articles/merge-request-basics).
  11. The Repo Owner reviews the merge and performs the merge to ‘master’
  12. The Release Tech Lead also pushes the Release code changes to ‘develop’ branch
  13. After the merges are done to Master and Develop branch, the Release Tech Lead then deletes this release branch after couple of weeks.

## HotFix Branch WorkFlow

**Action: Master 🡪Hotfix Branch🡪Master & Hotfix Branch 🡪 Develop/Release Branch**

**Actors**: Hotfix Team developers and HotFix Branch Owner (HotFix Tech Lead) and the Repo Owner

* + 1. Hotfix branch is the dedicated branch to quickly patch production releases (break fix)
    2. A dedicated hotfix Team and/or a HotFix Tech Lead(possibly from SSMO) is identified
    3. Hotfix Branch is created off of master by the HotFix Tech Lead when a bug is identified by the end-user.
    4. The hotfix Team developers make the changes in his local hotfix branch and push his/their code changes to remote hotfix.
    5. Jenkins ‘Build/Release’ job kicks off the build on the hotfix branch (on commits), runs automated unit tests, create a release version of the artifact to be deployed. **(ex; UHOne\_MemberOutbound-1.0.1.jar)**
    6. Jenkins creates a tag **(ex FixName-Version#)** and saves the release version of the source code in the remote codehub repository.
    7. Once the code is deployed in Production, HotFix Tech Lead creates a tag for this Release (incase, Jenkins automation didn’t create one)
    8. Hotfix Tech Lead then request the merge to master via [merge requests](https://codehub.optum.com/help/articles/merge-request-basics).
    9. The Repo Owner reviews the merge and performs the merge to ‘master’
    10. The Hotfix Tech Lead also pushes his code changes to ‘develop’ branch directly (or to the Release Branch via merge requests, if one exists).
    11. After the merges are done to Master and Develop or Release branch, the hotfix Tech Lead then deletes this hotfix branch after couple of weeks.

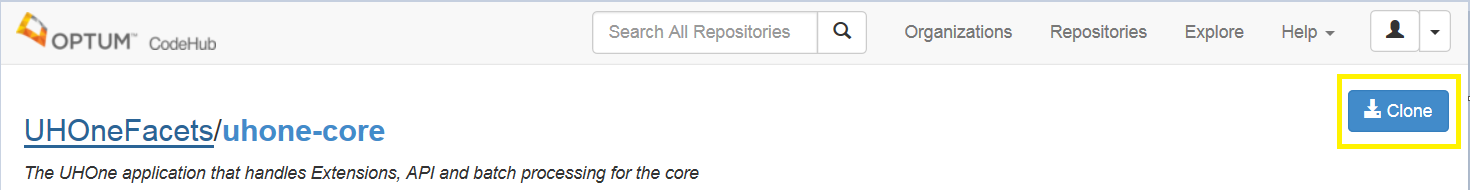
# WorkFlows – Detailed Git Steps

## Git Steps for Developer’s workflow

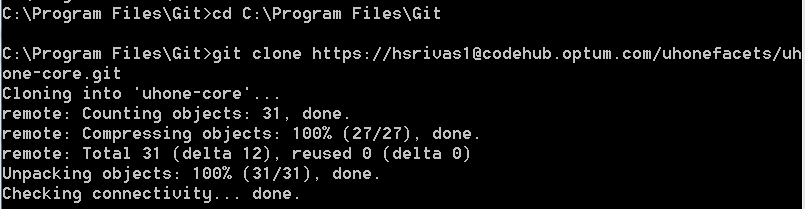
**Action: Feature**🡪**Develop Branch**

**Actors**: Feature developers

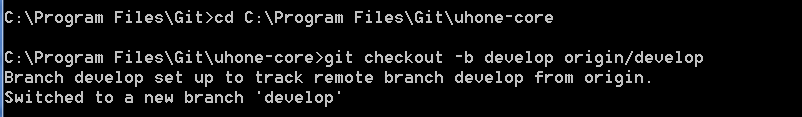
1. Developers needs to clone the central repository and create a tracking branch for develop
   * Open GIT CMD and go to a path where you want to create local repository.
   * Click on Clone button in CodeHub Repository page (https://codehub.optum.com/uhonefacets/uhone-core/tree/develop) to copy the clone URL link



* + Clone the central repository : **git clone –b develop** [**https://msid@codehub.optum.com/uhonefacets/uhone-core.git**](https://msid@codehub.optum.com/uhonefacets/uhone-core.git)

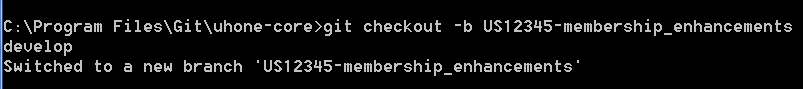


* + Go to your local repository and checkout to create a tracking branch for develop : **git checkout -b develop origin/develop**

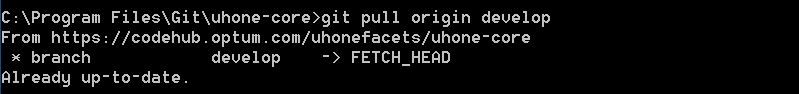


**Note:** This is one time activity to clone the repository in our local, follow step 2 for repo synchronization.

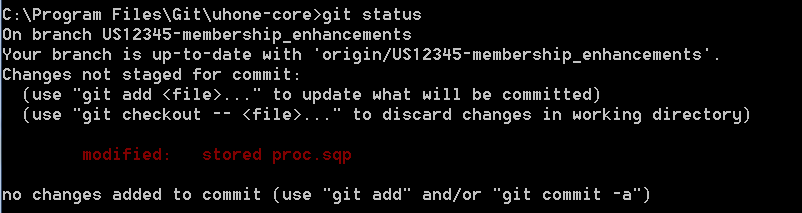
1. Once a feature started , Developer creates local feature branch from the upstream branch ‘develop’
   * **git checkout -b US12345-membership\_enhancements develop**



1. Developer codes his user story in his local feature branch.
2. Developer keeps his local code synchronized very often with upstream changes that are committed by other developers.
   * **git pull origin develop**
   * **git checkout develop**



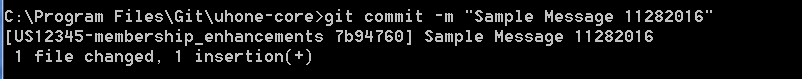
1. Developer resolves conflict between develop and feature branch locally.
2. Developer thoroughly tests locally and does maven build and pushes the changes from local ‘feature branch’ to the remote feature**(optional)**. Add commits to the feature branch in the usual fashion: edit, stage, commit.
   * **git status**

****

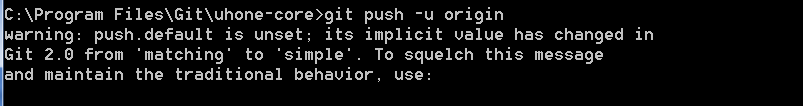
* + **git add "~path\stored proc.sqp"**



* + **git commit –m "commit message"**



1. Developer pushes the completed feature branch to the remote CodeHub **(Optional)**.
   * **git push –u origin <feature branch Name>**



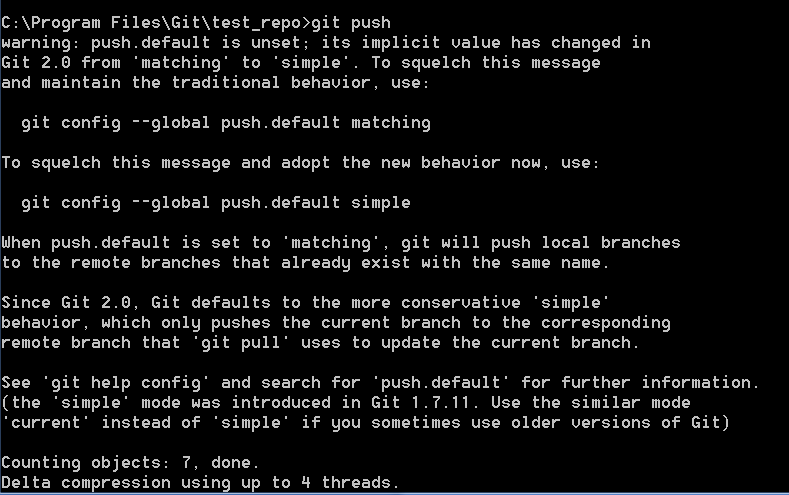
1. In order to merge the changes to ‘develop’ remote branch, developer needs to synchronized and checkout the latest changes from remote.
   * **git pull origin develop**
   * **git checkout develop**



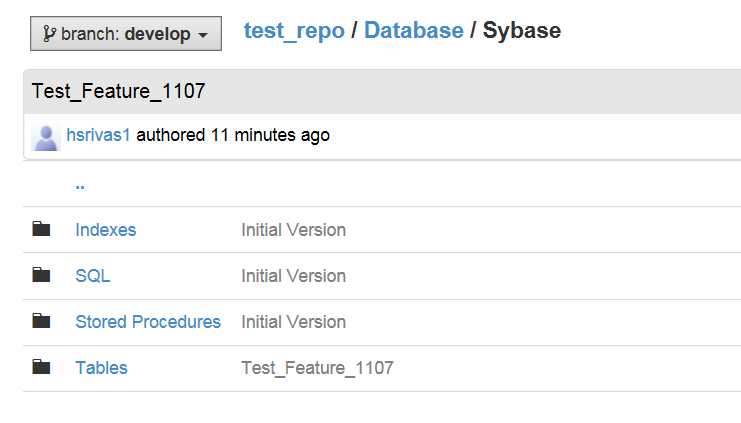
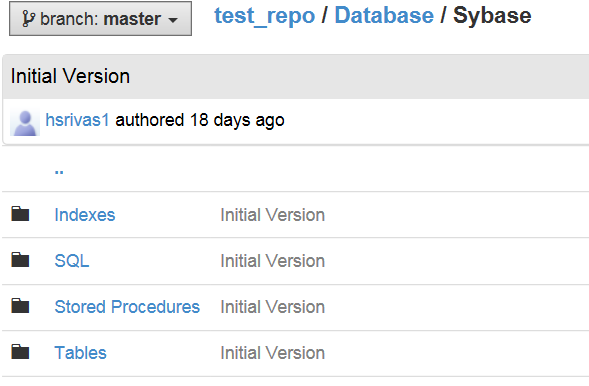
1. Developer performs the merge to ‘develop’ branch from feature branch.
   * **git merge <branch name>**



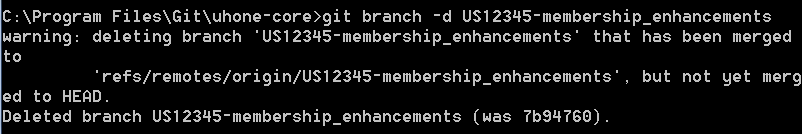
1. Developer pushes the changes to remote ‘develop’ branch.
   * **git push**



Once we push the latest feature to develop, a push mail notification will be sent to Developer’s DL. it will be reflected to Develop branch only , master will still not have any changes as below :

1. Developer deletes his Feature Branch (locally and in remote) after the merge to Develop (may be after a week or two)
   * To Delete the Branch locally use : **git branch –d <branch name>**



* + To Delete the Branch on Remote CodeHub, Click on View Branches and delete your feature branches.



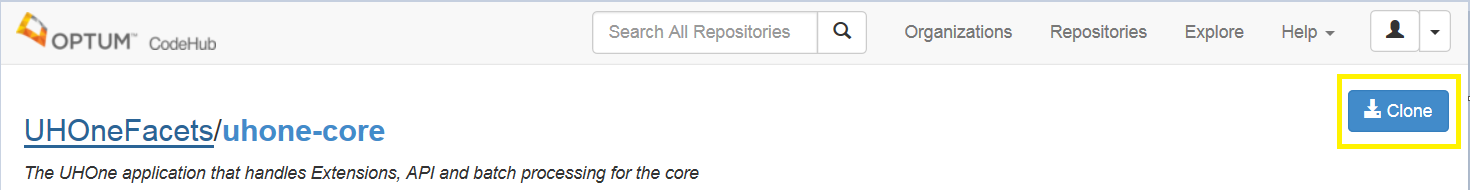
1. Developer repeats steps ii-xi for each user story/feature.

## Git Steps for Release’s workflow

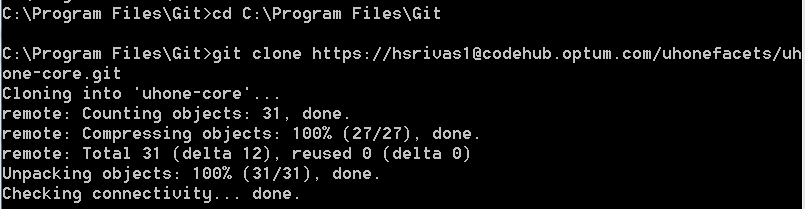
**Action: Develop Branch**🡪**Release Branch**🡪**Master & Release Branch**🡪**Develop Branch**

**Actors**: Release Team developers, Release Branch Owner (Release Tech Lead) and the Repo Owner

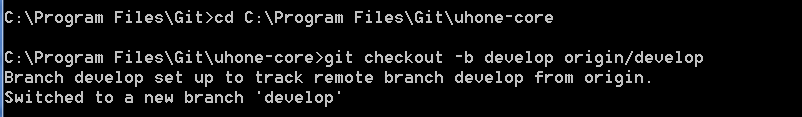
1. Release branch is the dedicated branch to prepare releases.
2. A dedicated Release Team and a Release Tech Lead is identified.
3. Release Branch is created **by the Release Tech Lead** once develop has acquired enough features for a Release or when the Release Date is approaching
   * + **For example, REL\_PI1\_Sep**
     + When Release Branch is created forked from the develop branch, the next Release cycle starts (No Features are added at this point; only bug fix and release oriented tasks are worked in this branch).
4. Release Team Developer/Tech lead needs to clone the central repository and create a tracking branch for develop
   * Open GIT CMD and go to a path where you want to create local repository.
   * Click on Clone button in CodeHub Repository page (https://codehub.optum.com/uhonefacets/uhone-core/tree/develop) to copy the clone URL link



* + Clone the central repository : **git clone –b develop** [**https://msid@codehub.optum.com/uhonefacets/uhone-core.git**](https://msid@codehub.optum.com/uhonefacets/uhone-core.git)

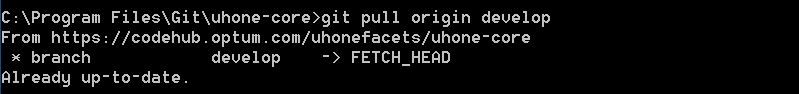


* + Go to your local repository and checkout to create a tracking branch for develop : **git checkout -b develop origin/develop**

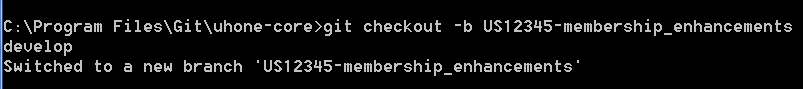


**Note:** This is one time activity to clone the repository in our local, follow step 2 for repo synchronization.

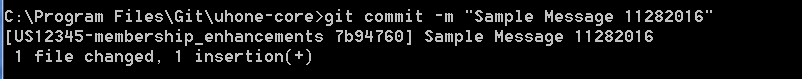
1. Release Team developer keeps his local code synchronized with upstream changes that are committed by other developers.
   * **git pull origin develop**
   * **git checkout develop**



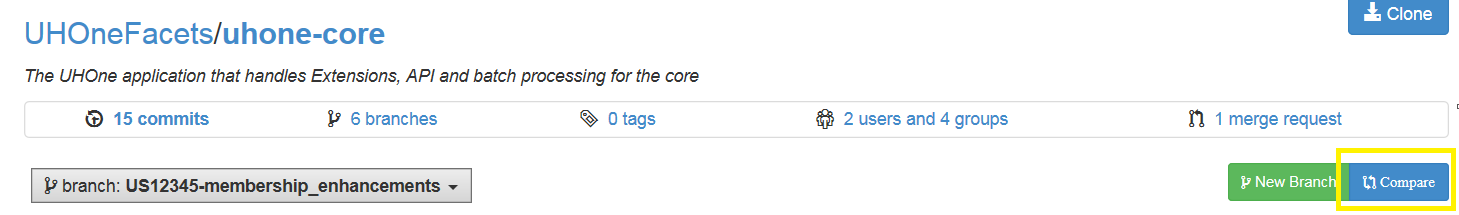
1. The Release Team developers create a local release branch and push his/their code changes to remote Release.
   * **git checkout -b REL\_PI1\_Nov develop**



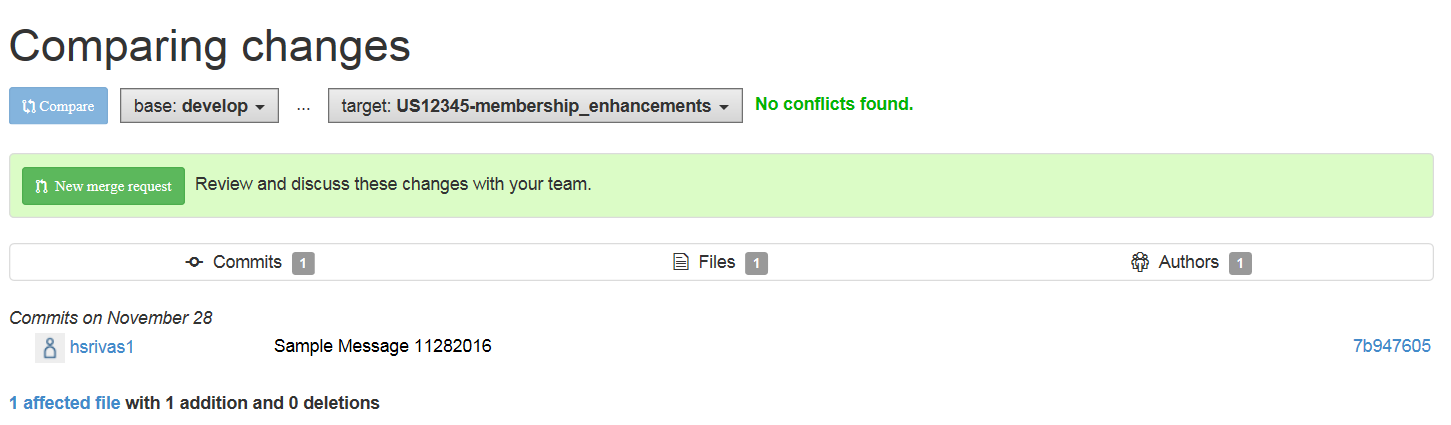
1. Release Team developer pushes the Release branch to the remote Release.
   * **git push –u origin <Release Branch Name>**



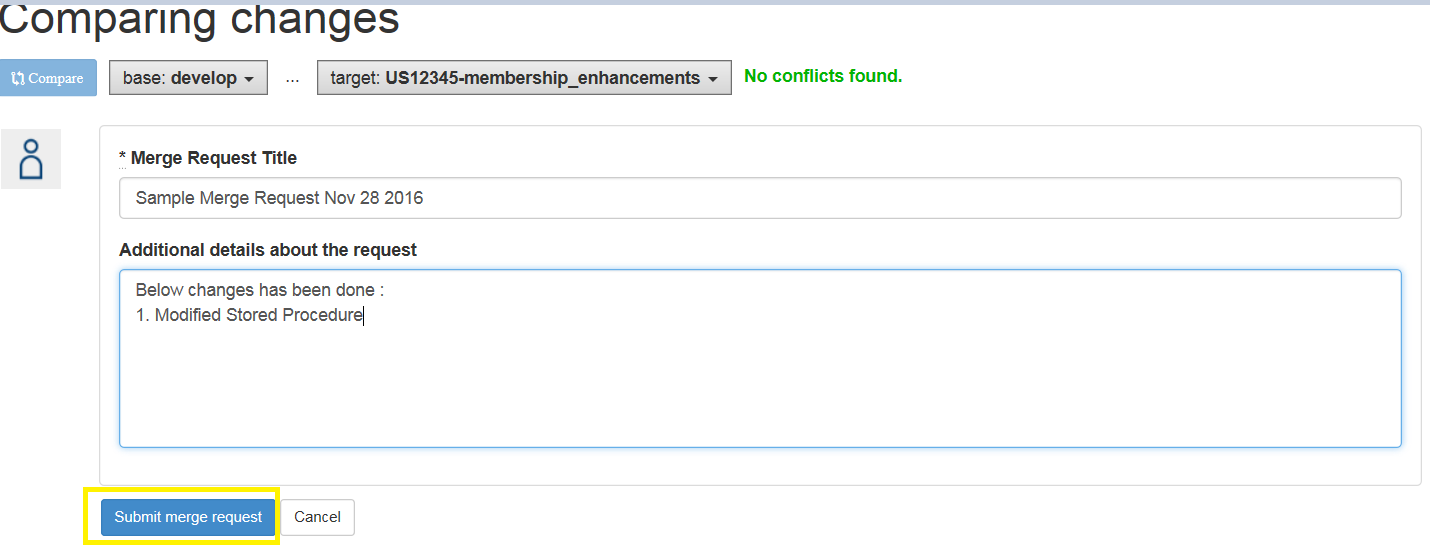
1. Jenkins ‘Build/Release’ job kicks off the build on the release branch (on commits), runs automated unit tests, create a release version of the artifact to be deployed. **(ex; UHOne\_MemberOutbound-1.0.0.jar)**
2. Jenkins creates a tag **(ex; ReleaseName-Version#)** and saves the release version of the source code in the remote CodeHub repository.
3. Once the code is deployed in Production, Release Tech Lead creates a tag for this Release (incase, Jenkins automation didn’t create one)
4. Release Team Developer/Tech Lead then request the merge to master via [merge requests](https://codehub.optum.com/help/articles/merge-request-basics).
   * Go To CodeHub Repository and click on Compare



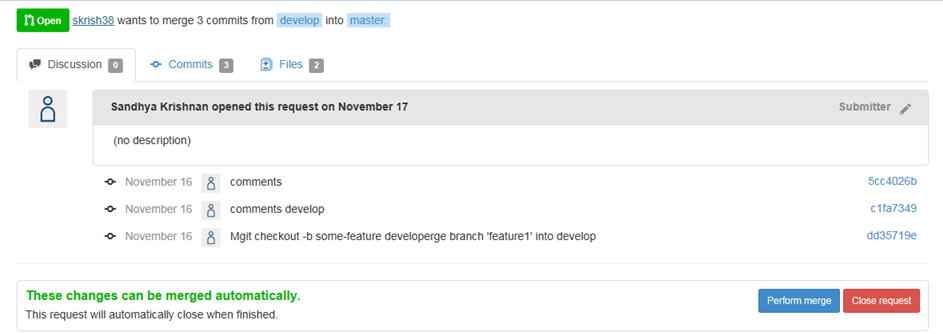
* + Select Master as Base and Release branch as target branch and click on new merge request



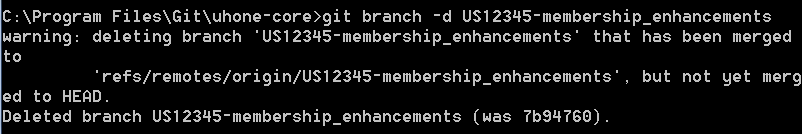
* + Add some Title and details about the merge request and submit , a notification mail will also be received (by the requestor and also by the approver) once request has been submitted.

` 

1. The Repo Owner - the Approver, reviews the merge and performs the merge to ‘master’



1. The Release Tech Lead also pushes the Release code changes to ‘develop’ branch
   * **git checkout develop**
   * **git merge release-0.1**
   * **git push**
2. After the merges are done to Master and Develop branch, the Release Tech Lead then deletes this release branch after couple of weeks.
   * To Delete the Branch locally use : **git branch –d <Release branch name>**



* + To Delete the Branch on Remote CodeHub, Click on View Branches and delete your feature branches.



## Git Steps for Hotfix’s workflow

**Action: Master 🡪Hotfix Branch🡪Master & Hotfix Branch 🡪 Develop/Release Branch**

**Actors**: Hotfix Team developers and HotFix Branch Owner(HotFix Tech Lead) and the Repo Owner

# Misc Git Steps

## When and How to rebase.. blah, blah…

## When and How to do git reset blah blah

## When and How to check git status blah blah

## When and How to do git fetch blah blah

# Troubleshooting tips

## My build failed during the merge, but it worked fine on my branch!

The leading causes of this are:

* You tried to merge a branch build that isn't the latest.
* Somebody pushed to develop between the time the final merge step started and ended. It is safe to get the latest code from develop and do the final merge again.
* Your branch is out of date with ‘develop’ and there are conflicting changes during the merge. Merge develop into your branch, resolve the conflicts and push them back.
* Develop has some tests that your changes impacted. This is unlikely but possible. Merge develop into your branch and run your tests to reproduce the issue. Resolve, push, build and merge again.

# Git Help/Additional Websites

<https://www.atlassian.com/git/tutorials/syncing>

<http://blog.endpoint.com/2014/05/git-workflows-that-work.html>