**VA GitHub Enterprise**

**VDIF Repository**



**Department of Veterans Affairs**

**October 2018**

**Version 1.0**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/01/2018 | 1.0 | Initial | D. Wayland, R. Lauro |
|  |  |  |  |

# Introduction

In an effort to better coordinate and coalesce custom code and configurations for the HealthShare platform currently under development for the VHA, the VDIF (Veterans Data Integration and Federation) Enterprise Platform (formerly HSEP – HealthShare Enterprise Platform), the decision was made to utilize the GitHub platform to maintain source control on the various projects underway. This encompasses teams working on various aspects of the HealthShare integration (see below for a list of the components being worked on by these teams.

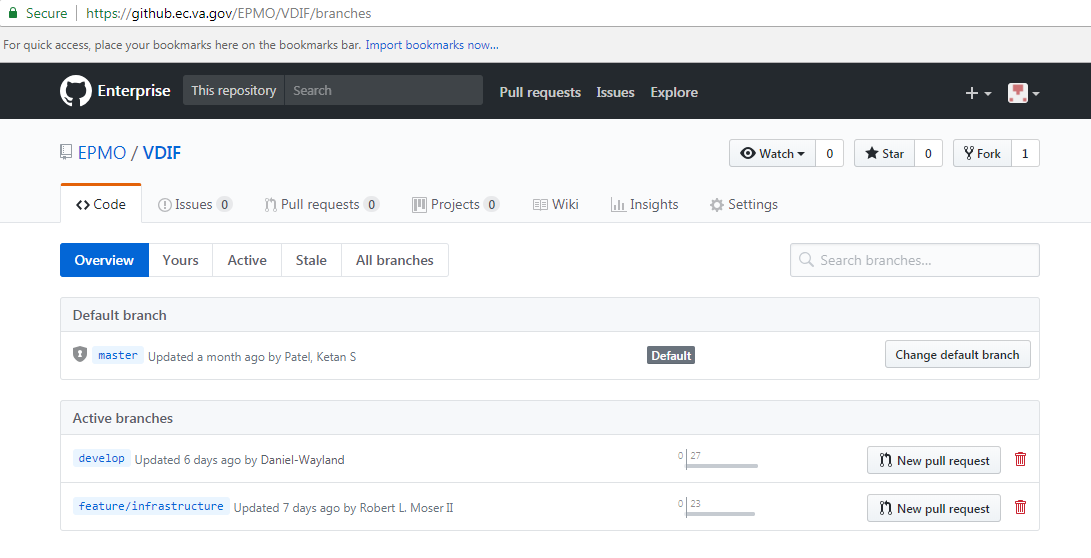
In current state, many teams have already been working with source control through the Rational platform, but are all under separate, unique Rational projects. The GitHub platform approach is intended to ensure that all code and configuration changes made under the VDIF umbrella are managed in a single repository to help ensure teams do not encounter any conflicts with such changes, should another team be working within the same area of the HealthShare platform.

The repository has been stood up on the EPMO GitHub and can be found [here](https://github.ec.va.gov/EPMO/VDIF). The site uses SAML for login,

# Teams Included in VDIF Repository

|  |  |
| --- | --- |
| **HSEP Components** | **Team POC/Admins** |
| CCDA | Andrew Carter/Dan Wayland |
| SDA | Ken Leonard/Ron DiMiceli |
| VIE | Ken Leonard/Jessica Denman |
| Edge Gateways (ECR) | Lori Warren/Rick Ross |
| Registry - MVI | Roger Metcalf/Rick Ross |
| Registry - Terminology | Steve Owens/Jason Jones |
| Enrollement Services Reengineering (ESR) | Jason Jones/Rick Ross |
| Audit Repository | ?/Rick Ross |
| x12 (EDI) Enterprise | ?/Rick Ross |
| x12 - PCAC | ?/Rick Ross |
| FHIR – AMPL GUI | ?/Rick Ross |
| FHIR - COMS | ?/Rick Ross |
| Referral and Authorization |  |
| GitHub – Expert/Admin | Yousef |

# Repository Information



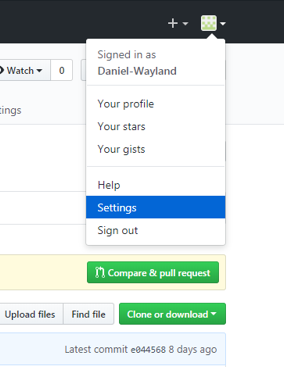
The above image shows the main page for the VDIF repository on the browser platform. From here, you have access to browse the branches and sub-folders as needed to find the artifacts you may need to update, or find locations for where new artifacts need to be added. In subsequent sections of this document, you will find instructions for cloning this repository to your local system along with instructions for following the GitFlow approach the team determined to be the most effective for handling changes to this repository.

# Cloning – Person Token Setup

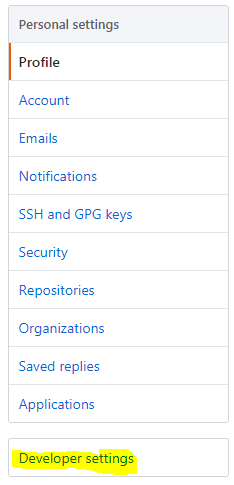
If you don’t have a password yet, you get it by going to the VA GitHub Enterprise VDIF Repository

<https://github.ec.va.gov/EPMO/VDIF>

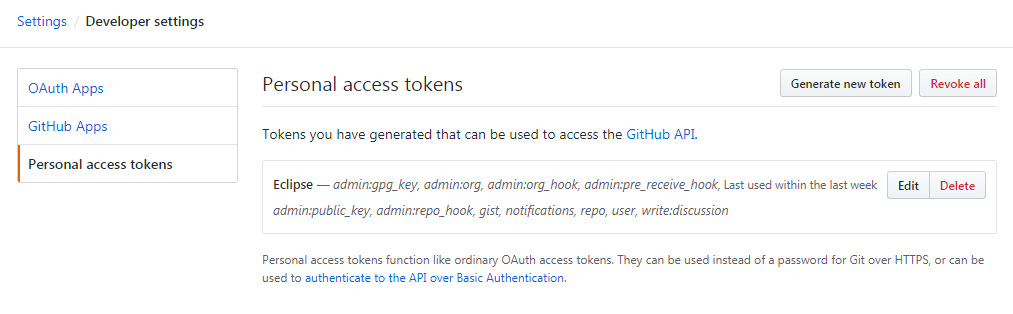
And navigating to your user settings



And then Developer Settings



And then Personal Access Tokens -> Generate New Token



And then check everything except the admin stuff or the delete repository stuff and generate your token.

You shouldn’t be able to get the ability to do the admin or delete repo stuff, but why try?

Your token will be a very long string. Save that somewhere. That’s your password

Also note your username from this website. It’s typically firstname-lastname. E.g. John-Castle

# Cloning - Git Command Line Utility

While there are various methods and approaches available for cloning and managing the repository on your local system, the Git Command Line Utility is the suggested utility to use for creating local development branches and eventually committing these changes to the repository. The steps for all this will be discussed later, but this section will focus on installing and configuring the command line utility.

## Download and Install

Download the Get git command line from here:

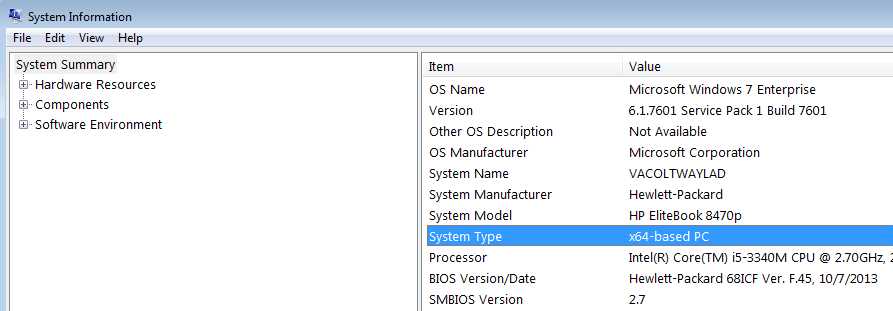
<https://git-scm.com/download/win>

The portable one will not require admin privileges on your VA laptop



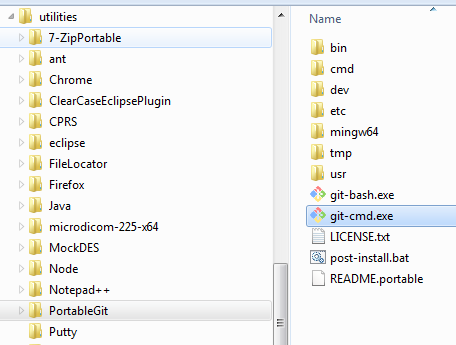
My laptop is 64-bit, but get the right one for your laptop’s chipset (64 bit is typical these days)

You can verify that your workstation is 64-bit in the System Information tool



Unzip the portable command line client to a folder of your choosing:

e.g.



The one you will run that will be like the Windows DOS command prompt will be git-cmd.exe

You can do everything with the command line client. Eclipse’s EGit plugin works as well, but I’m going to the Git Flow bits don’t work reliably in Eclipse, so we’re going to recommend command line for now.

## Local Repository Setup

Create a directory for your copy of the VDIF Repository

mkdir VDIF\_Repository

and move your command prompt into that directory:

cd VDIF\_Repository

Issue the following command to let GitHub retain client self signed certificate:

git config --global http.sslBackend schannel

Replace ${user} with your username and ${token} with your access token, and run the clone command (ignore word wrap)::

git clone -b develop [https://${user}:${token}@github.ec.va.gov/EPMO/VDIF.git](mailto:https://$%7buser%7d:$%7btoken%7d@github.ec.va.gov/EPMO/VDIF.git)

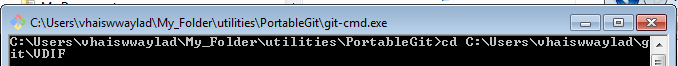
If you get an error regarding the certificates such as a self signed certificate being in the repository chain error, remember to issue the command above.

A work around, although not recommended, is to add “-c http.sslVerify=false” to the git command like so (ignore word wrap):

git -c http.sslVerify=false clone -b develop [https://${user}:${token}@github.ec.va.gov/EPMO/VDIF.git](mailto:https://$%7buser%7d:$%7btoken%7d@github.ec.va.gov/EPMO/VDIF.git)

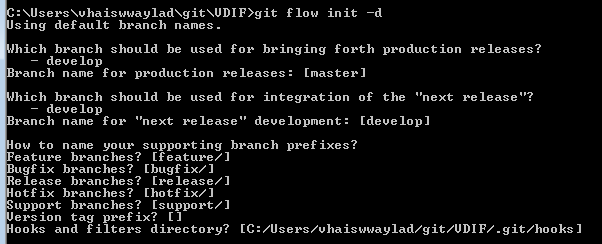
## Setting up for Git Flow

Then, for Git Flow, you need to cd your Git command line into your local clone of the repositories’ VDIF directory



And then run:

git flow init -d



You only need to do this **once** when you clone the repository (would only need to do it again if you ever made a new clone)

# Updating and Committing – Git Command Line Utility

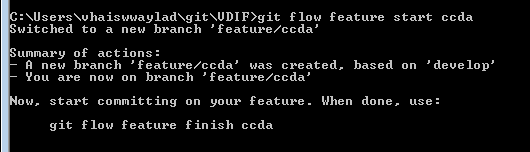
## Create New Local Branch

Git Flow is a standard branching strategy that the community recommended we use. If developers follow these steps and use the Git Flow commands, then all of the branching, merging, and moving around of the code between the branches is fairly automatic. This serves as a great example of incorporating governance into the tooling by making it easy to do the right thing.

**Step 1 -** Replace ${featureName} below with a name that you can make up for the feature or issue that you’re working on, and run the following command:

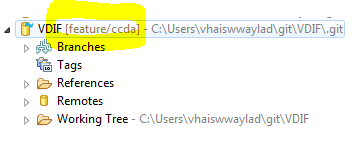
git flow feature start ${featureName}

This will create a branch for you to do your work on. Here’s an example with a feature name of ccda



\*\*\* Note –

If you’re using Eclipse, too, then your eclipse plugin should reflect the change to the new branch:



\*\*\* end note

**Step 2 -** Make your changes / edits using any editor you want.

Then add to index via the command:

git add -A

Put in a comment describing this commit and run this command to commit the work:

Git commit -m "${comment}"

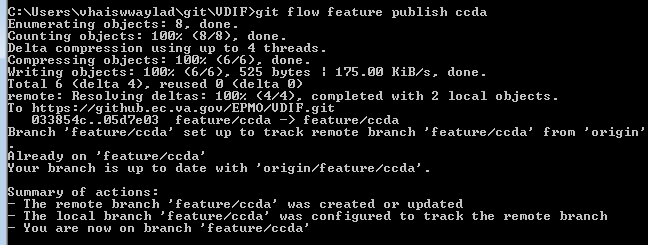
At this point, your changes are just on your local copy of your feature branch.

Do the above edit, add to index, and commit steps as many times as you want/need to work out your stuff on your local branch.

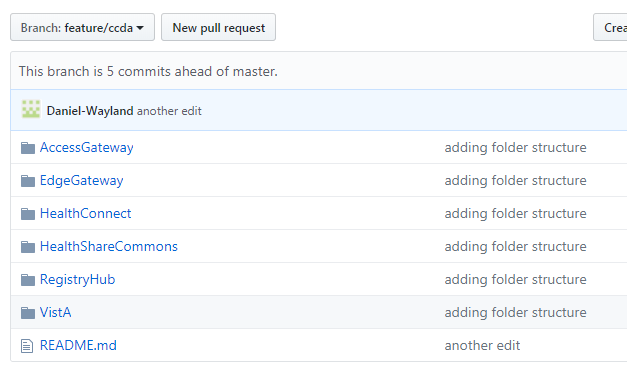
## Incremental Commits/Testing

When you’re done and you’re ready for your teams internal testing, then make sure you’ve committed everything and then do:

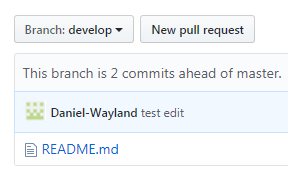
git flow feature publish ${featureName}

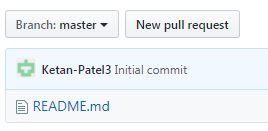


Now your completed work is on your feature branch in the remote server repository



But not on any of the other branches

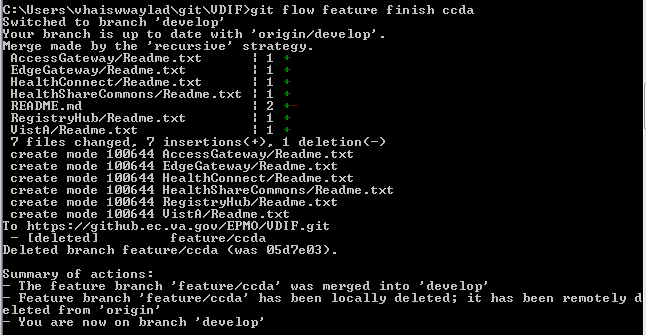




## Committing Changes to Develop Branch

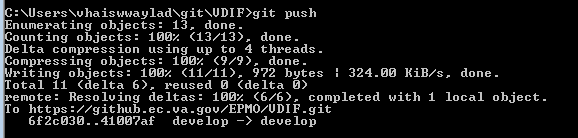
When your team’s testing shows that it’s good to go and you’re ready to share with other teams / for integration testing / SQA, you run:

git flow feature finish ${featureName}

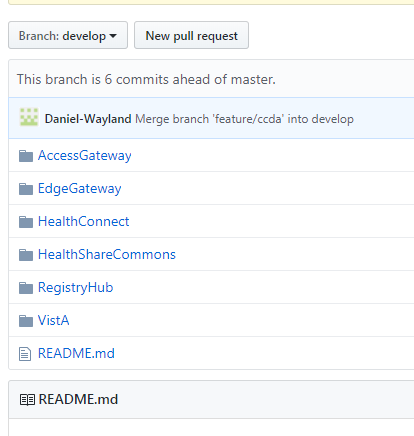


Now it’s on your local copy of the develop branch, so just one more step – and so just one more command, run:

git push



Now your changes are on the remote server’s develop branch that the other team’s branches and the release branches are pulling from and pushing to.

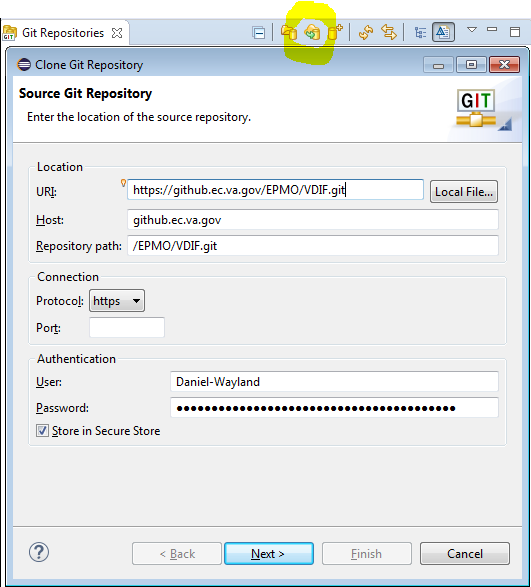


# Eclipse Git Client Setup

Eclipse comes with a Git client called EGit that works really well and is really convenient for some of these regular non-git-flow operations like clone, add to index, and commit.

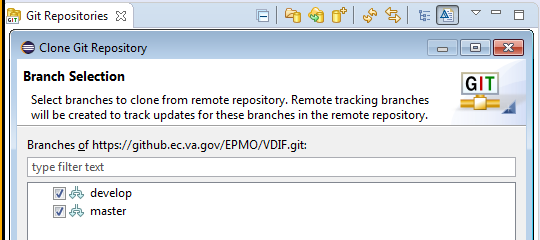
## Eclipse Clone

To clone the repository with Eclipse, you switch to the git perspective, hit the button highlighted below.

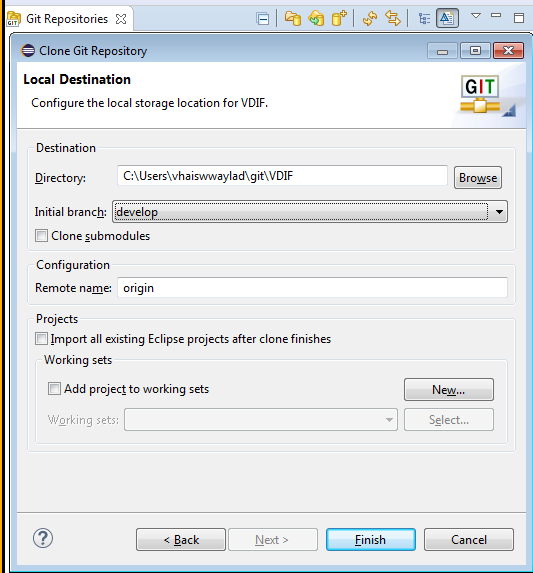


In this dialog you only have to paste in the URL into the URI field and most of the form will fill out automatically. Then add your username and password (token – see above command line clone instructions for how to obtain your token).

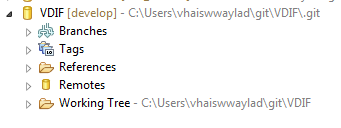
On the next page, make sure master and develop are both checked… you won’t be able to change master, but you do need to be able to see it / read it for gitflow to work



On the next page, make sure the Initial Branch is set to develop



And click finish, you should see something like this when it works

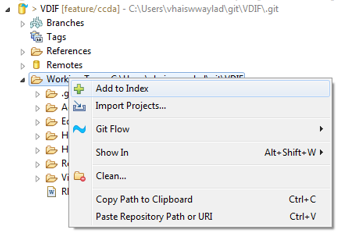


## Eclipse Add to Index and Commit

Make your changes / edits using Eclipse Atelier or any editor that you like.

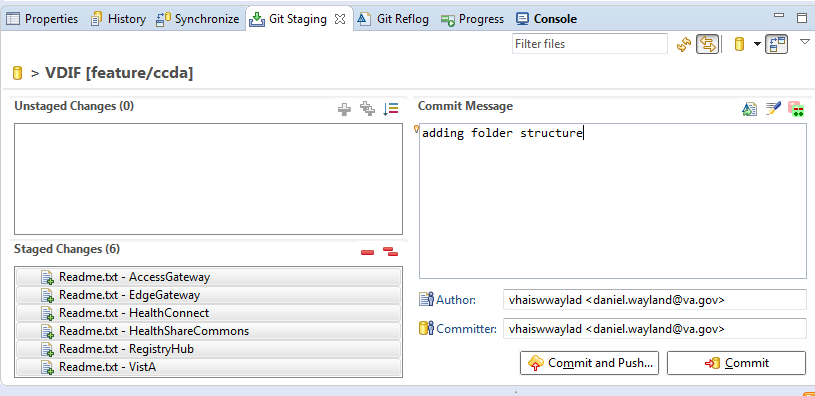
Regardless of the editor used, here’s how to add and commit the changes with Eclipse

Right click on your working tree and choose add to index:



Make sure all Unstaged Changes are Staged Changes (the ++ button will move any that are unstaged, but the above should have staged them all).

Put in a comment describing this commit and hit “commit”



At this point, our changes are just on our local copy of our feature branch just like if you were using the command line.

Do the above edit, “add to index,” and “commit” as many times as you want/need to work out your stuff on your local feature branch.

## Eclipse Git Flow Plugin

**We recommend using the command line for all of the git flow commands. Despite the fact that the screenshots above show an integrated Eclipse Git Flow plugin, that plugin has proven unreliable.**

**For now, the Git command line remains the best way to work the Git Flow way with VA’s GitHub Enterprise.**

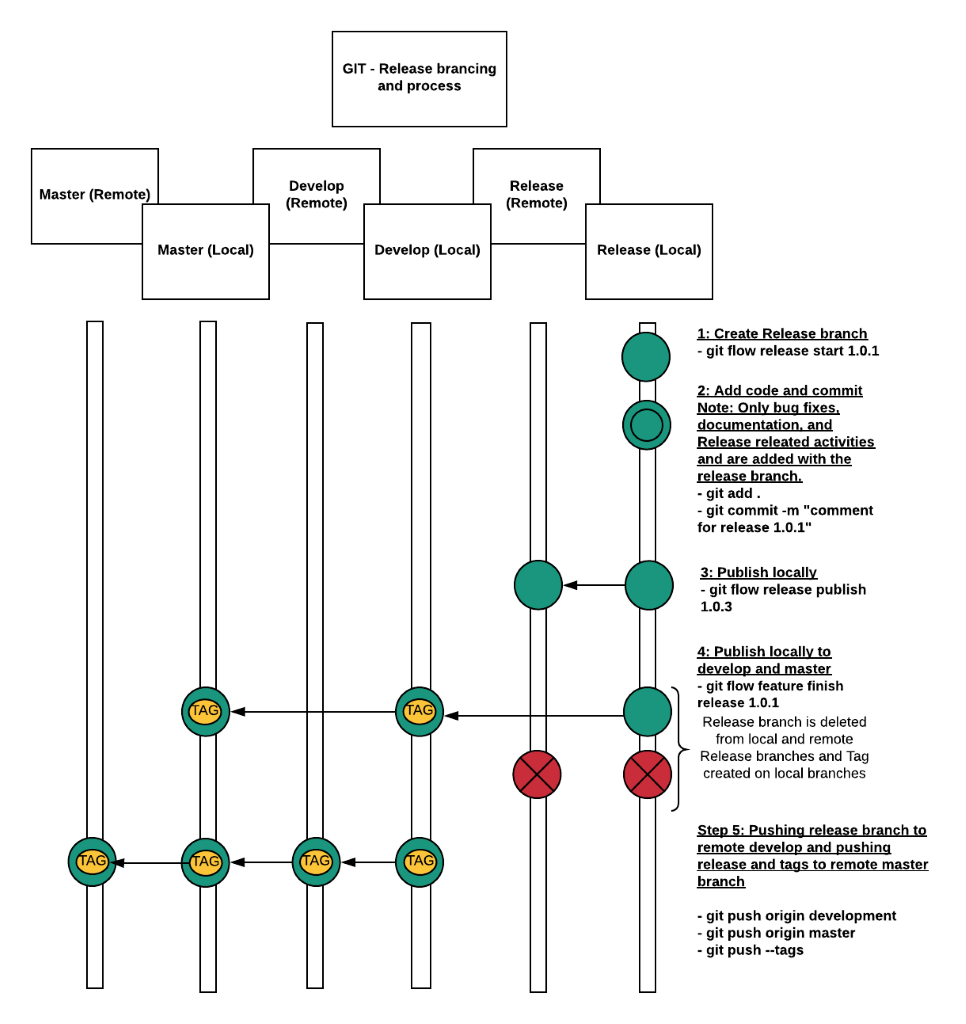
# Release Process - Incremental/Full release

Looks like as things get done and are working on develop, to do a release in git flow, we’re supposed to use git flow to start a release branch.

It ends up being a lot like a feature branch, but the main difference is that when you finish and push it, it not only put it up to develop and master, but also tags it.

Those tags make it where we’re always be able to pull up the version 1.0 source of the initial HealthShare version even years later when there’s no longer a copy of that running anywhere or any branch where that’s the latest on the branch.

There’s going to be some stuff to figure out about controlling that release branch… I think the github / CM admins should probably make it and control it and not let anything but very controlled changes get made on it.



# Release Process – HotFix release

And then, if something really bad happens in production, here are the git flow commands / process for a hot fix. The main difference here is that the branch will be made off the master (current production version) and when finished it will become the new master / production version and the change will be merged into Develop (which is presumably already beyond production a bit).

And it will also be tagged since it’s a release. Same reasoning as above.

