Table of Contents

1. Introduction

1.1 Summary

The main purpose of this document is to provide a source code set of versioning tools and associated training to use those tools

1.2 Objectives

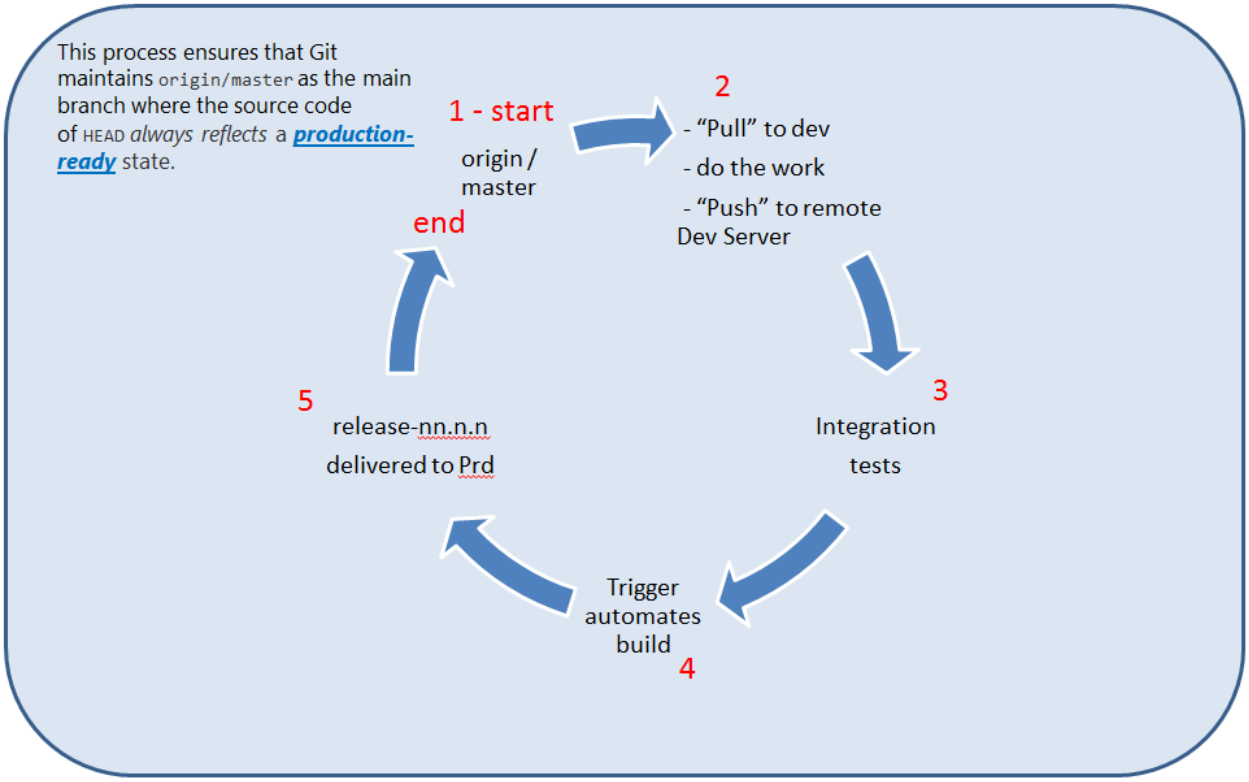
- Allow open source developers to install Git, configure Git on their workstations in a standard way, and use Git with the approved IDE's for Big Data development
- Maintain official versions of code on: http://sa0tfs101:8080/tfs/ACSC/Big%20Data/_git

1.3 Scope

This document specifically addresses open source code for the "R" and Python languages, but can be expanded to include libraries for:

- ❑ R libraries
- ❑ Python libraries
- ❑ Java/Scala libraries
- ❑ Others if needed

2 Overview of the Git Process



1) All code starts in (1) the origin / master branch
2) Developers "Pull" existing objects to their workstations
    a. Do their work
    b. Then "Push" their work to Remote dev branch
3) The code now resides on the MLS Dev server where Integration Tests are run
4) Leads kick off a trigger
5) and the tested code is moved to release-nn.n.n on the Production server
6) End – After stabilization, the code is merged back into the origin / master branch

Then the cycle can start over at step 1.

3 General Setup

3.1 Using AAA On Premises TFS Git

- Download and install “R” from the AAA authorized location below.
- Download and install “RStudio” from the AAA authorized location below.
- **Download and install** Git from this AAA authorized location:

\\sa0code100.ace.aaaclubnet.com\install\Server Software\Windows R Client

You must have admin privileges to install under the C:\Program Files (x86)\Git.
This is the preferred location. If you do not have Admin privileges, get an admin to install this software or choose the default location:

C:\Users\<yourUserName>\AppData\Local\Programs\Git

Use the rest of the default settings during installation of Git.

3.2 Configuring Git on Your Workstation

Setting your Git username for ***every*** repository (globally) on your computer.

Task	Example/Task
Create a directory for all your open source code or use an existing directory (Git root directory)	Example: “C:\R_Repository”
Open a DOS or Windows command prompt	From the Start button type cmd: Start -> cmd or press the Windows key + R - type cmd Type: **cd ** <enter> #start at root **mkdir \R_Repository** <enter>
Goto Git root directory where you want you repository to begin	Example: C:\R_Repository -> **Type: cd \R_Repository** <enter>
Initialize Git from the command line	git init
Set username for GLOBAL git	`git config --global user.name "John Doe"`
Set email for GLOBAL git	`git config --global user.email doe.john@aaa-calif.com`
Set the Git push default style	`git config --global push.default simple`
Remove any existing remote repos	`git remote rm origin`

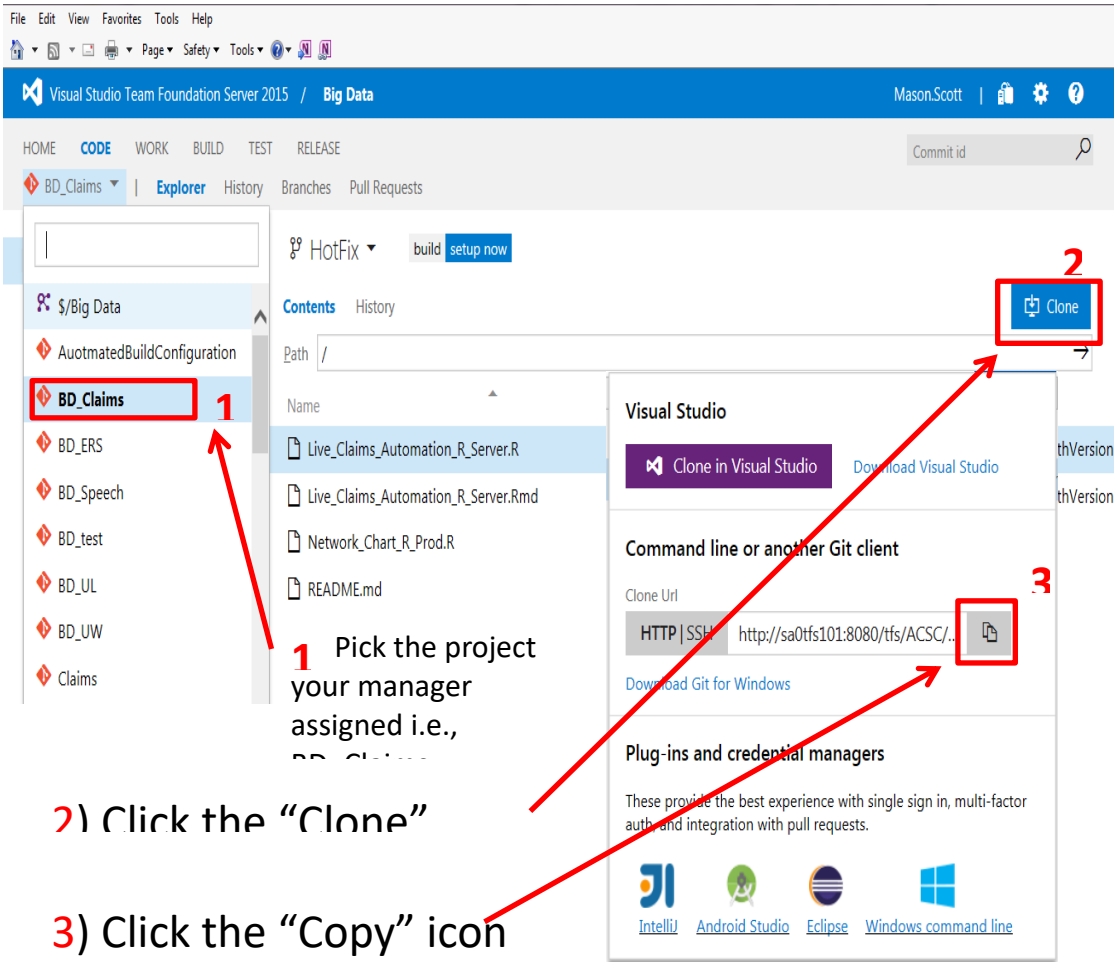
3.3 Find your team's Git On Premises TFS Git Remote Repo

Go to: http://sa0tfs101:8080/tfs/ACSC/Big%20Data/_git

From the Dropdown at the top left - Select your Team's Git Repo:

- BD_Claims
- BD_ERS
- BD_Mkt
- BD_Screen
- BD_UL
- BD_UW

1. Pick your assigned project from the dropdown list as shown in **(1)** below



The url of your remote repository is now in your clipboard.

3.4 Always Start RStudio by Opening the "R Project" File

Always Start RStudio by Opening the R Project File. The information below will show you how to create a new R Project under source control.

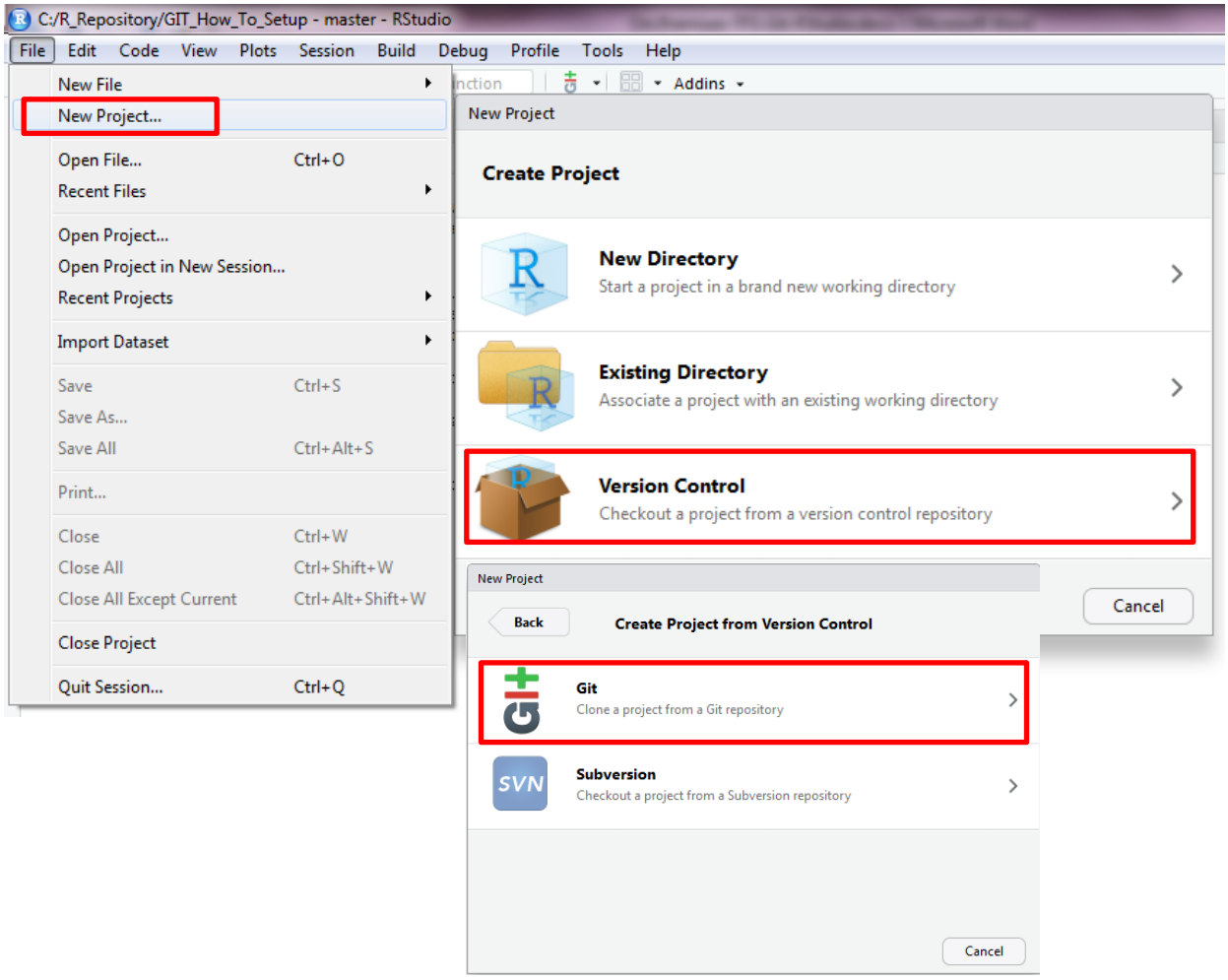
Never go to File Explorer and open a *.R file.

Opening an *.R file from File Explorer with not launch the R project and GIT version control will not work without a R Project.

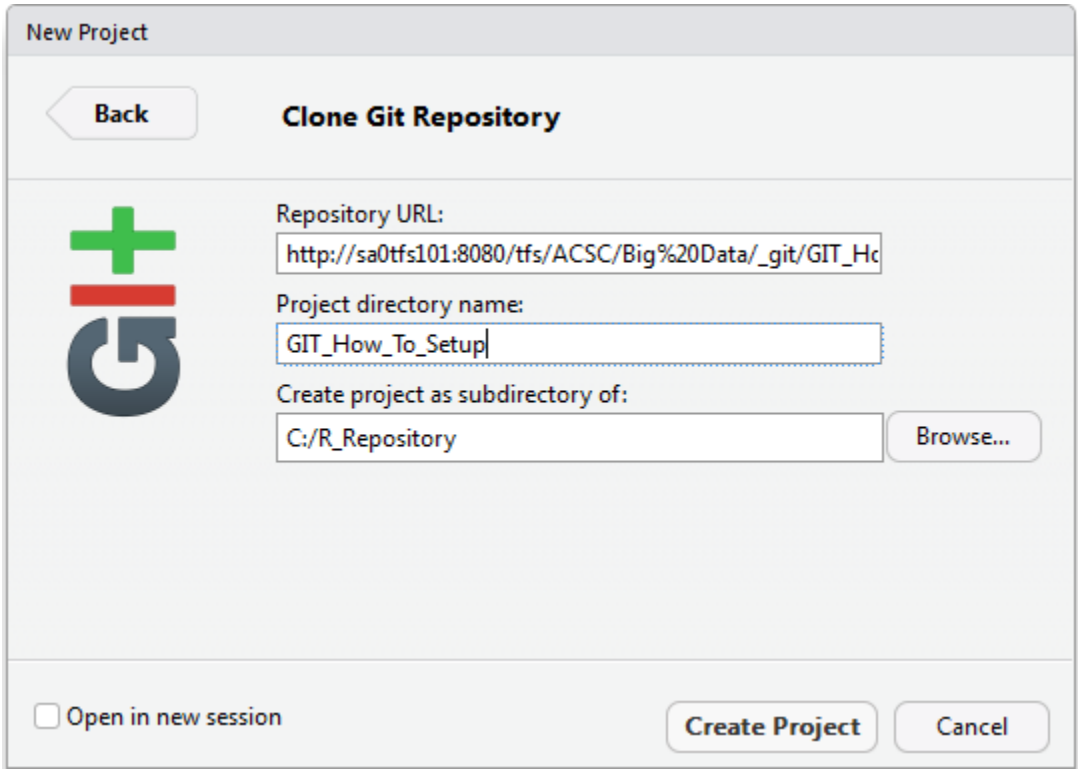
3.5 Set up New Project - Get the Checked-In Code From the Remote Repository

To get a copy of the source code, you clone a RStudio Git repository. Cloning creates both a copy of the source code for you to work with and all the version control information so Git can manage the source code. In order to use GIT with RStudio you have to create a new RStudio Project.

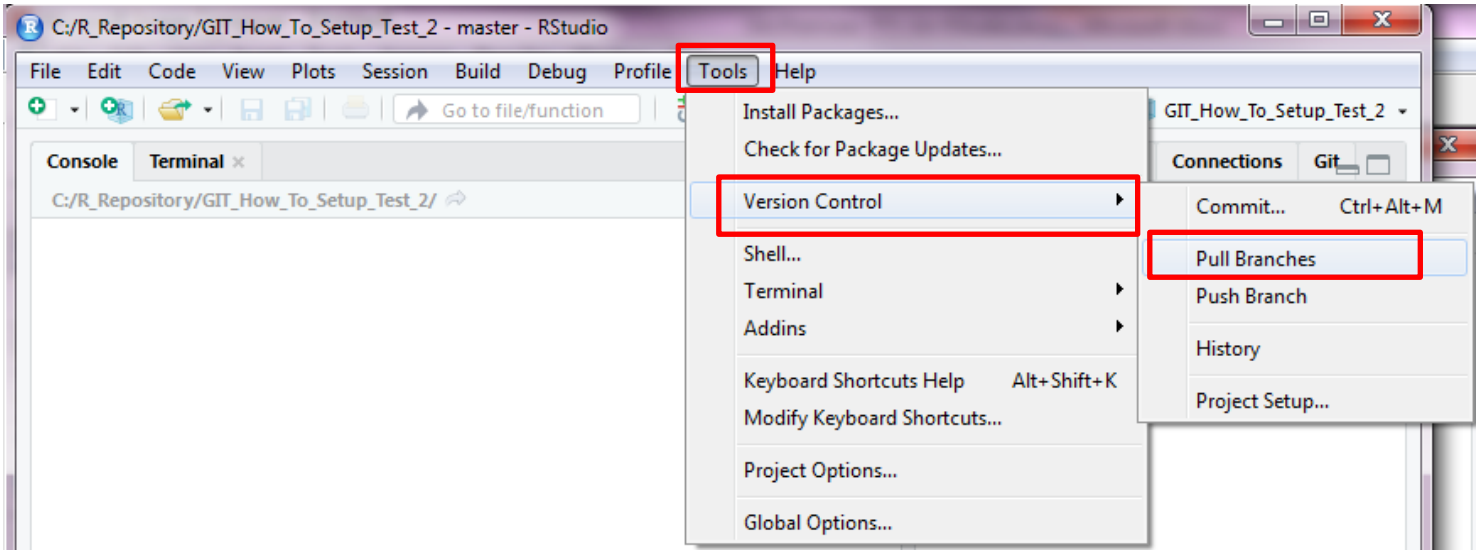
1. Open RStudio
2. Create a new RStudio project (this is required for Git)
3. Select the "Version Control" option as shown below:



4. Paste in the url of the TFS / Git repo you copied to your clipboard above into the Repository URL: box shown below.



5. The system will automatically populate the directory name for you
6. Select the "Open in new session" checkbox – click the "Create Project" button
7. From the "Tools" menu pull the remote's Branches into your local repository as shown below:



The files from the AAA remote repository will now show in your "Files" tab in RStudio.

3.6 Always Start Your Development By Getting Code From 'master'

From the Windows command line <Windows key R – cmd> or Run cmd
Type:

cd \R_Respository\BD_ERS <BD_Claims> <BD_ERS>, etc.
git branch # to view the branches you already have locally

```
git checkout -b dev master #change to local 'dev' branch with code from master
```

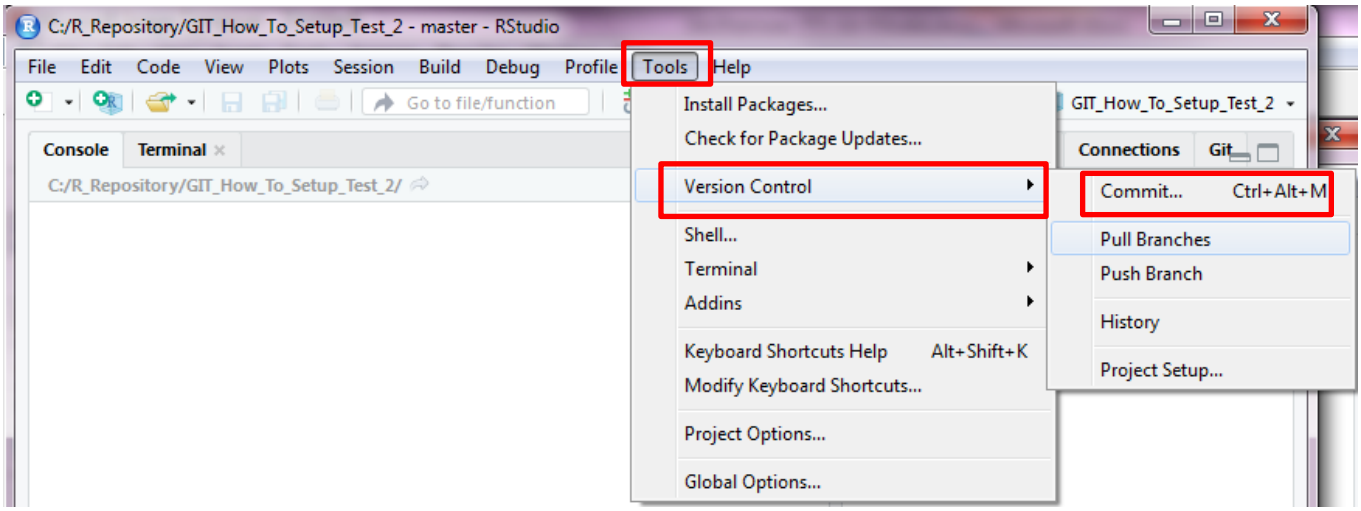
This will create a local dev branch and pull all files in 'master' branch to your local 'dev' branch.

3.7 Edit Your Code and 'Commit' as Often as You Wish

Edit RStudio "R" files (Code) that your manager has assigned to you.

Every time you want to save a version of your code on your **local** repository:

1. From the "Tools" select "Version Control", then "Commit" as shown below:

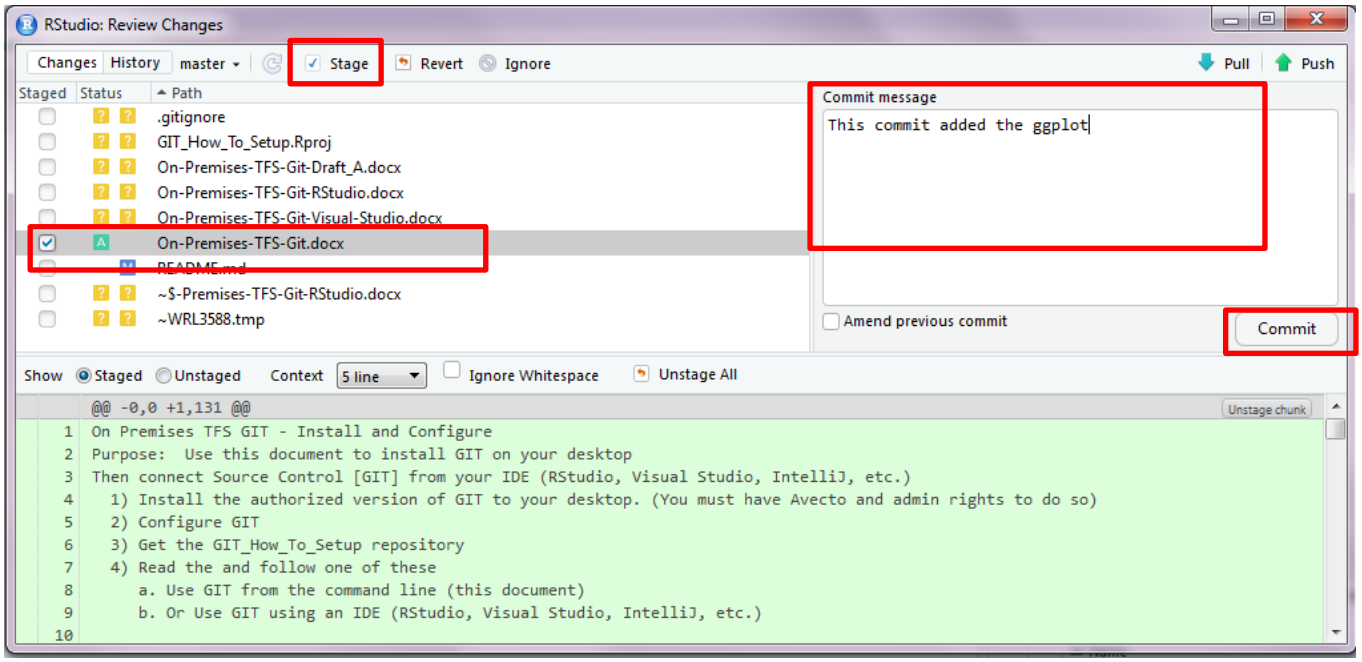


This will bring up a window to select which files you want to commit.

2. Select the "dev" branch to commit your code to your local "dev" branch
3. Select the checkbox next to the file(s) you want to commit

Always add a comment to each one of your commits to help others know what this version changed/added/deleted.

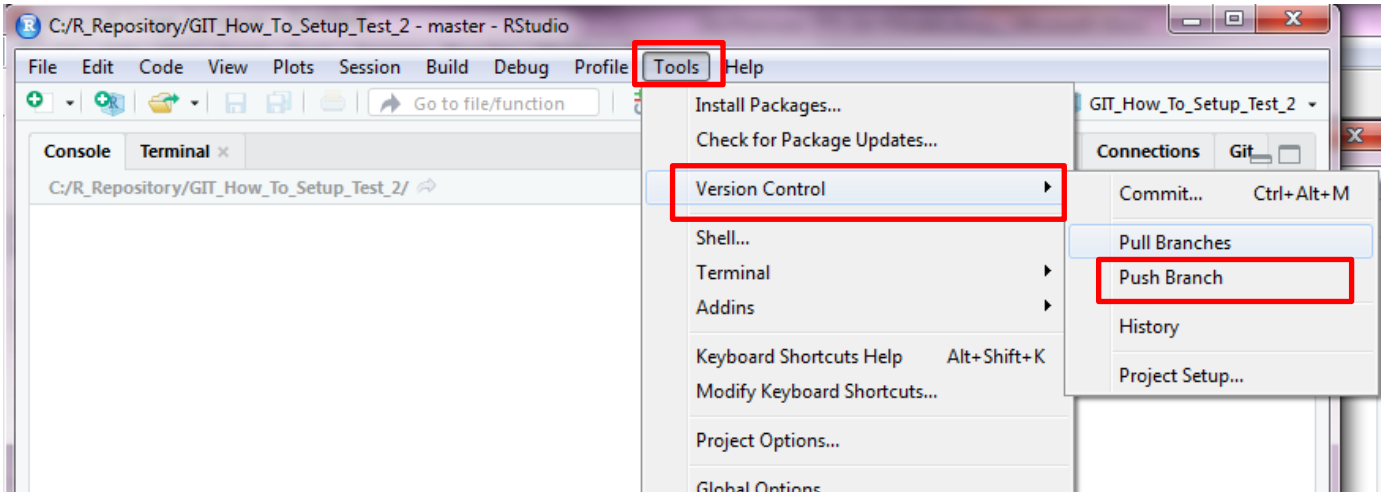
4. **Do Not check** the "Amend Previous Commit" button show below.
5. Type a message for this version to identify what changed
6. **Do not check** the "Amend previous commit" checkbox
7. Click the "Commit" button



When you are done unit testing your code, or on your team's scheduled check-in day (often every Thursday), check in your (Push) code and/or changes to your documents to the Remote Repo that the rest of your team has access to.

3.9 When Your Manager Approves Your Code to be Moved to the Dev Server - Share / Push Your Code to the Remote Repository

1. When your manager and/or Lead authorizes your code you may "Push" your code to the remote 'dev' branch. This will trigger some automation to move that code to the MLS Dev server.
2. After committing your code you can "Push" your code
3. From the "Tools" select "Version Control", then "Push Branch" as shown below:



This will push the files you have kept in all your commits to the remote server.

Your team will now be able to see your changes.

3.10 View History of Changes

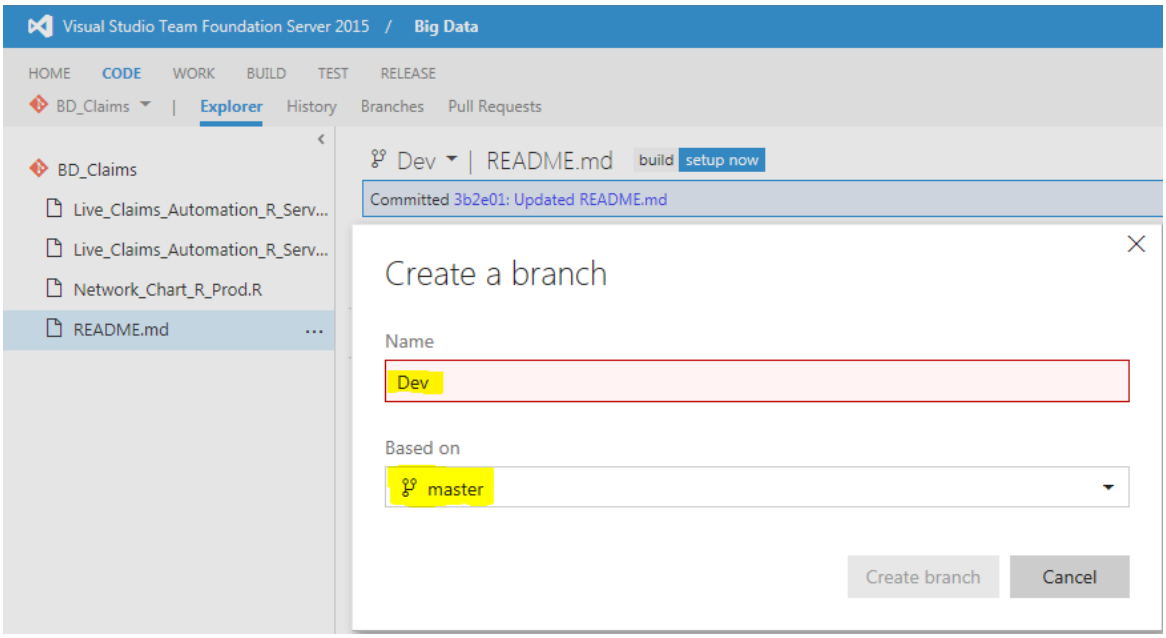
You can view the "History" of changes from the "History" menu item above.

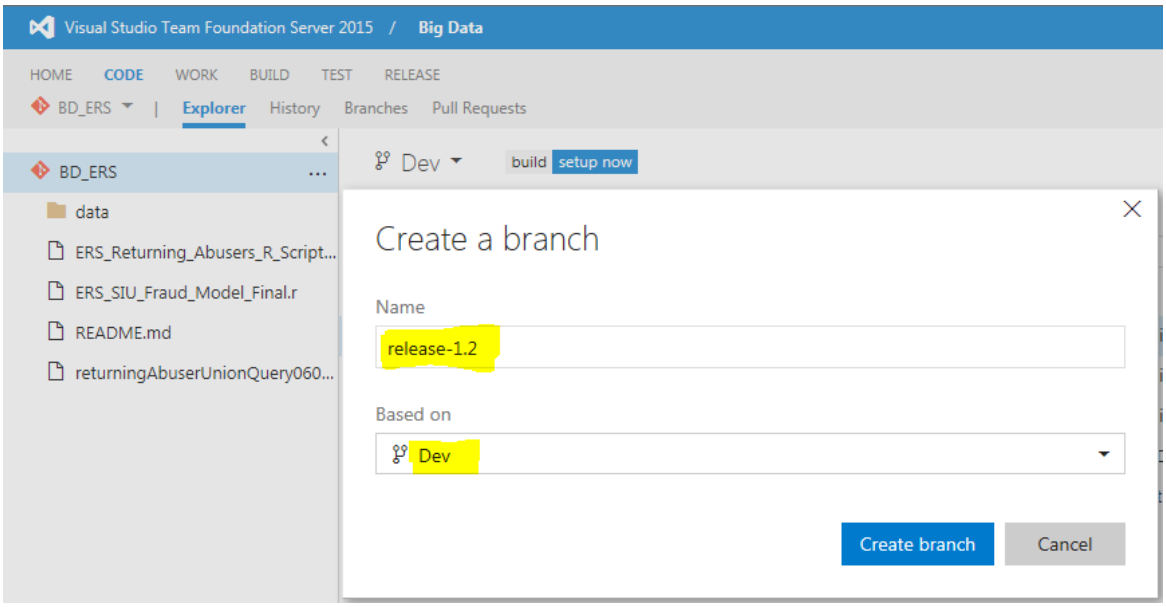
This is where you can view various branches and changes between 2 files.

3.11 Understanding Branches

Currently, AAA supports 3 Branches in Git

- master (always refers to 'origin' in Git terms)
- dev (always based on master as a AAA standard)
- *release-nn.n.n* (always based on dev as a AAA standard)





Developers: This means that when you begin development or update existing code you first 'pull' from 'master' master -> dev (local on your workstation)

Developers: You can commit as often as you wish to dev (local).

Developers: You do your unit testing on your workstation until it passes all user story testing steps.

Developers: then you 'push' your developed/changed code to 'dev' (remote) where other team members have access to it.

Deployment Manager: The Deployment Manager then gets notice to 'pull' the dev (remote) to dev (local on the MLS Dev server).

Next the Deployment Manager creates a 'release-nn.n.n' branch so that finished code in dev can move to release-nn.n. branch when code is put into production.

Team: Your code is tested on MLS Dev until it passes all QA activities.

Developers: Once your code has passed all tests:

- On your computer you 'Pull' from dev (remote) to dev (local) to pick up any changes made in the testing cycles on the Dev server above
- You're your team fill out an RFC to move the code to production

Deployment Manager: The Deployment Manager refers to the RFC and in the MLS Production machine 'pulls' the new changes to Production.

Developers: Validate your code.

How to handle 'hotfixes' on release-nn.n.n is still an open question. At a later time AAA may implement additional branches (See Appendixes below).

For finer tuned management of your Branches, Commits, etc. use Git command line, not RStudio.

See "Managing Branches" Section below.

3.12 Managing Branches (By Code Master/Version Manager)

The main branches[1]

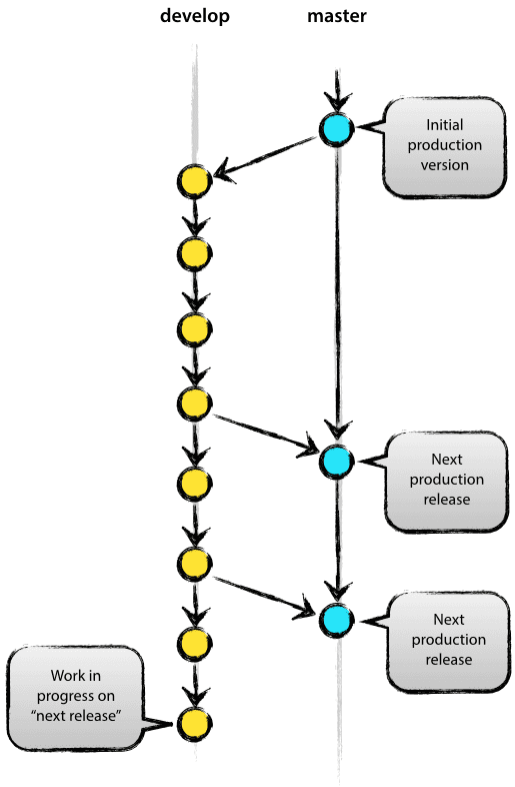
At the core, the development model is greatly inspired by existing models out there. The central repo holds two main branches with an infinite lifetime[1]:

- `master`
- `Dev`

The `master` branch at `origin` should be familiar to every Git user (this is the default).

Parallel to the `master` branch, we have created another branch called `Dev`.

[1] "A successful Git branching model" By Vincent Driessen https://nvie.com/posts/a-successful-git-branching-model/



Git considers `origin/master` to be the main branch where the source code of `HEAD` always reflects a ***production-ready*** state.

We consider `origin/dev` to be the main branch where the source code of `HEAD` **always** reflects a state with the latest delivered development changes for the next release. Some would call this the "integration branch". This is where any automatic nightly builds are built from.

When the source code in the `dev` branch reaches a stable point and is ready to be released, all of the changes should be merged back into `master` somehow and then tagged with a release number. How this is done in detail will be discussed further on.

Therefore, each time when changes are merged back into `master`, this is a new production release ***by definition***. We tend to be very strict at this, so that theoretically, we could use a Git hook script to automatically build and roll-out our software to our production servers every time there was a commit on `master`.

3.13 Start Development

Whenever your team has finished a release cycle the production tested and verified code should be in the 'master' branch.

Therefore, whenever you want to start development to change your code the first step should always be to get the verified code from 'master'.

Change your workstation branch to 'dev' from RStudio or the command line:

```
C:                          #change to C:\ drive
cd \R_Repository\UL         #change directories to where you keep your r code
git branch                  #view the branch(s) you have locally
git checkout -b dev master  #change to local 'dev' branch with code from master
# Begin development of your code
```

Use RStudio to 'commit', 'Push', and 'Pull' the code you are developing.

3.14 Creating a New Release Branch

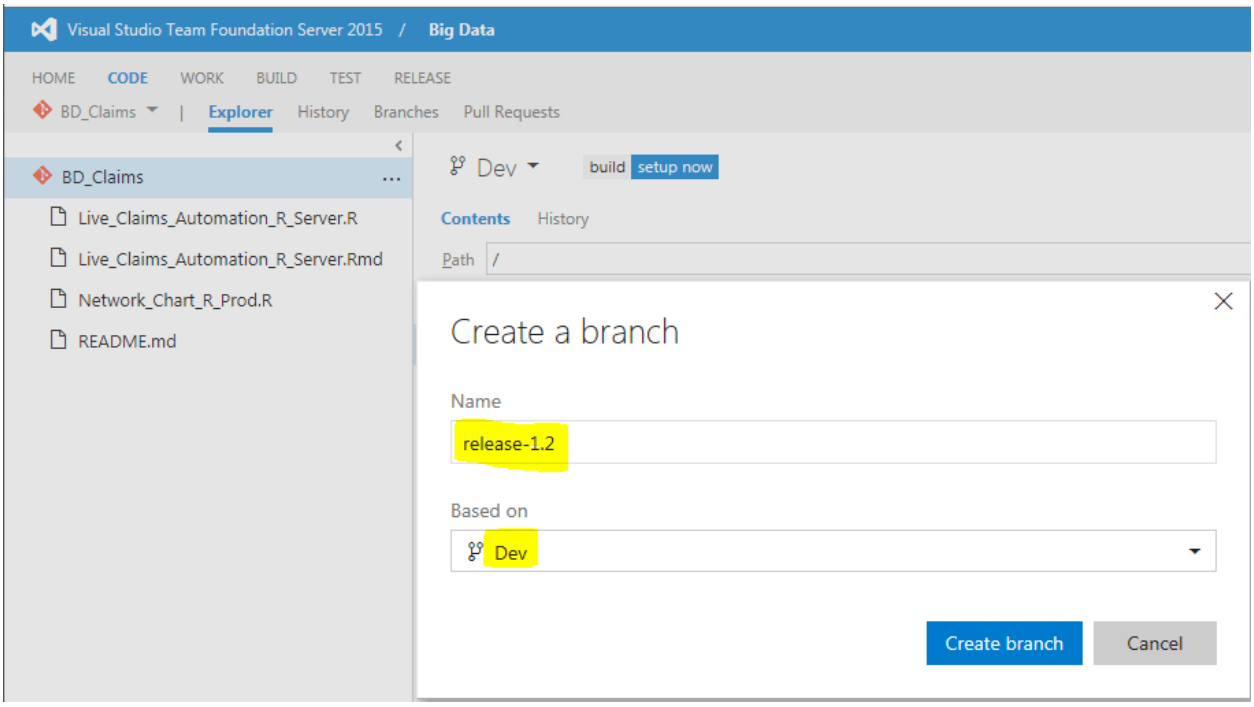
Only Leads will be creating and managing release-nnn.n branches.

Release branches will only exist on the Production Server and TFS.

Ultimately, release branches will be automated to trigger releases and movement of Dev code to the MLS Production Server.

Release branches are derived from the `dev` branch. For example, say version 1.1.5 is the current production release and we have a big release coming up.

The state of `dev` is ready for the "next release" and we have decided that this will become version 1.2 (rather than 1.1.6 or 2.0) = *release-1.2*. So we branch off and give the release branch a name reflecting the new version number:



You can accomplish creating the same from the command line as shown below. You can get to the command line using RStudio menu: Tools -> Shell

Leads will use Git from the command line:

```
# Create a new branch "release-1.2" derived from “dev”
#     and switch to the new branch "release-1.2"
$ git checkout -b release-1.2 Dev
# Files modified successfully, bump version to 1.2 (optional)
$ ./bump-version.sh 1.2
$ git commit -a -m "Bumped version number to 1.2"
[release-1.2 74d9424] Bumped version number to 1.2
1 files changed, 1 insertions(+), 1 deletions(-)
```

After creating a new branch and switching to it, we bump the version number. Here, `bump-version.sh` is a fictional shell script that changes some files in the working copy to reflect the new version. (This can of course be a manual change—the point being that ***some*** files change.) Then, the bumped version number is committed.

This new branch may exist there for a while, until the release may be rolled out definitely. During that time, bug fixes may be applied in this branch (rather

than on the `dev` branch). Adding large new features here is strictly prohibited. They must be merged into `Dev`, and therefore, wait for the next big release.

3.15 Finishing / Merging Branches

Finishing a release-nnn.n ***branch, (i.e., release-1.2)***

The Version Control manager should conduct the following steps are each stage of release.

When the state of the release-nnn.n branch is ready to become a real release - some actions need to be carried out.

First, the release-nnn.n branch is git br
d into `master` (remember, every commit on `master` is a new release ***by definition***).

Next, that commit on `master` must be tagged for easy future reference to this historical version.

Leads will use Git from the command line to merge branches:

```
# Switch to branch 'master'
$ git checkout master
# Now merge release-1.2 to master - recursively.
$ git merge --no-ff release-1.2
# (Summary of changes)
# Double check by pulling from ‘master’ (origin)
git pull origin
# Add a tag to easily identify releases by version number
$ git tag -a 1.2
```

The release to production is now done, and tagged for future reference.

The production code is back in the `master` branch. In the diagram at the top of this document we are back to step 1

3.15.1 Start the Development Cycle All Over Again – Get the code from the 'master' branch in Remote Back to Your 'dev' Branch

Finally, the changes made on the release-nnn.n branch need to be merged back into Dev, so that future releases also contain any bug fixes or *hotfixes* made in the release-nnn.n branch.

To keep the changes made in the release-nnn.n branch, we need to merge those back into Dev.

Using Git from the command line:

```
# From the Windows command line <Windows key+R cmd>
cd \R_Respository\BD_ERS      <BD_Claims> <BD_ERS>, etc.
git branch   # to view the branches you already have locally
git checkout –b dev master # change to local 'dev' branch with code
from master
```

The commands above will put you on your local dev branch and pull all files in 'master' branch to your local 'dev' branch so that you have the latest changes from your team.

You are now ready to start the development cycle again.

3.16 Types of Development

There are three general types of development:

- New Development
- Changes to existing code
- Hotfixes (emergency fixes to code in production)

New Development: follows the process above.

Cloning the Remote Git repo automatically does the first "Pull" for you.
The next step you should immediately do is to "Push" your first version of your code to the Remote Git repo.

If the new development is a significant amount of work, you can request a "feature_XXXXX" branch, for example, to be created. You would name the feature

whatever you want, but it should start with "`feature_`". You would "Pull" any existing code needed into the new feature branch, then your new code, and then follow the process outlined in Appendix B, C, and D below.

Changes to existing code: It is important to know the full lifecycle described above to make changes to code for the next release. After your code has completed the full lifecycle above it has moved from:

workstation –> dev Branch on Dev server -> release-nn.n -> Prd server -> remote `origin/master`

This process ensures that Git has `origin/master` as the main branch where the source code of `HEAD` *always reflects* a ***production-ready*** state.

Therefore you should always **`git checkout –b dev master`** from remote `origin/master` to local `dev` branch to pick up the latest changes for your new development.

See the merge process in the section: "Start the Development Cycle All Over Again" above.

Hotfixes: Hotfixes are minor teaks that need to be done from time-to-time when bugs are found in code in production *during* a release cycle.

When a new release goes into production there is a verification period and may consist of smoke tests and user acceptance. This verification period will last until the release has been determined to be stable.

During verification some hotfixes may need to be done to the production code. The MLS support admins will assist with the changes to the release-nn.n.n code until the release has been stabilized.

Once the release has been stabilized, the code in release-nn.n.n is merged into remote `origin/master`

release-nn.n -> Prd server -> remote `origin/master`

Some companies create a "hotfix-nn.n.n" branch for this purpose.

You can see the Hotfix process in Appendix C and D below.

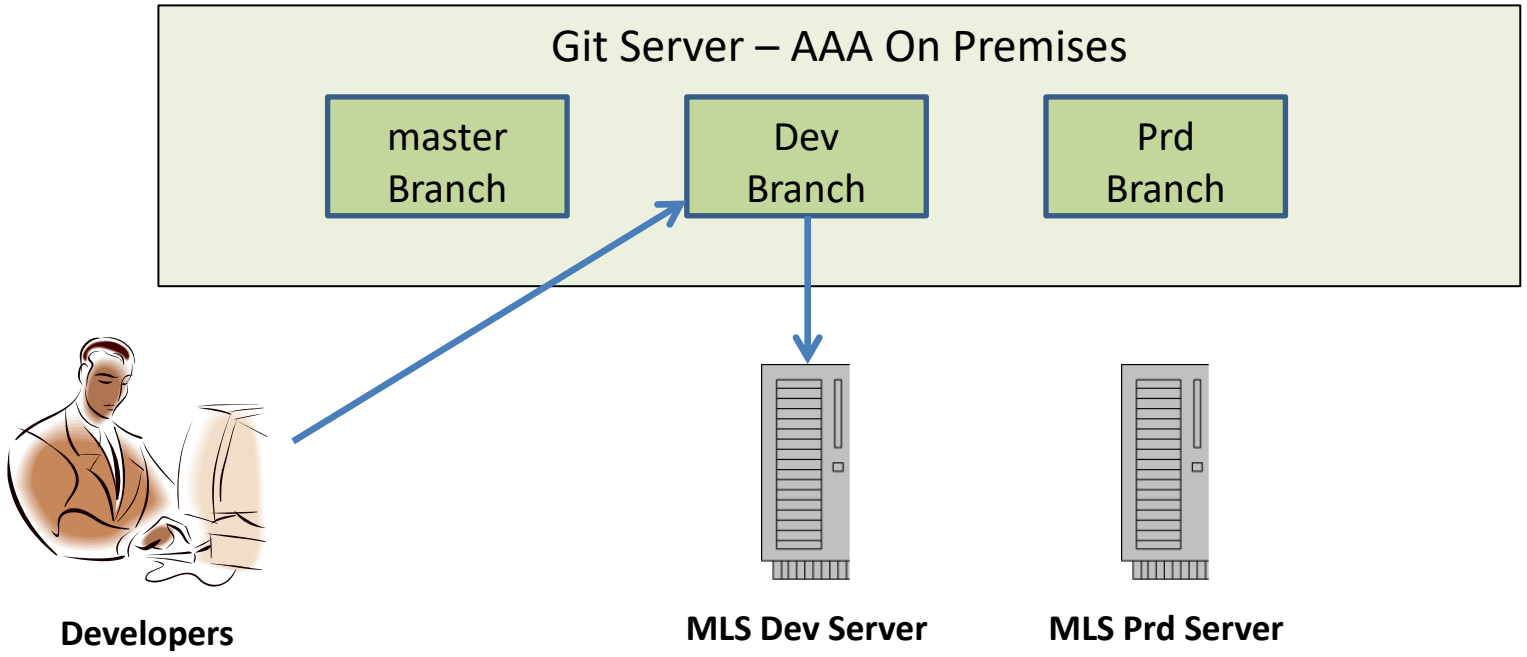
This process ensures that Git has `origin/master` as the main branch where the source code of `HEAD` *always reflects* a ***production-ready*** state.

All of the above scenarios will be covered in the Big Data Deployment Strategy as outlined here: Link, Document name: Big_Data_Deployment_Strategy.pptx

Appendix A

Administrative Control of MLS Dev Server with Version Control – Git

Open Source Code Migration to Dev



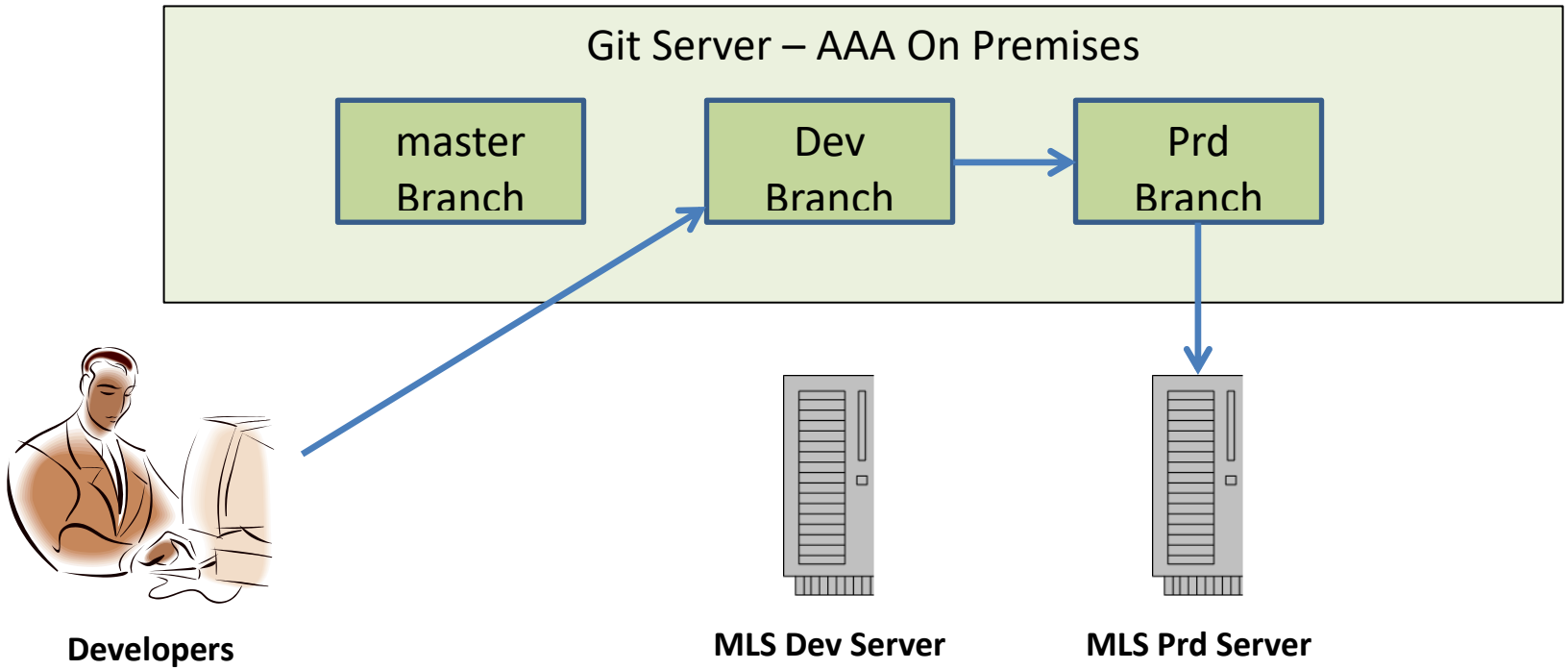
Development Team

1. Pushes code changes to Git **Dev** Branch
2. Updates control spreadsheet: R_scripts_scheduled_for_Production.xlsx
3. Updates officialListOfPackagesDev.txt with any new required libraries
4. Notifies Deployment Team a move to Dev is needed

Deployment Team

1. Ensures mini-library process runs
2. Using Git - pulls code into MLS Dev Server "**Dev**" (local) branch
3. Creates/updates SQL Server Agent Job(s)

Open Source Code Migration to Production



Development Team

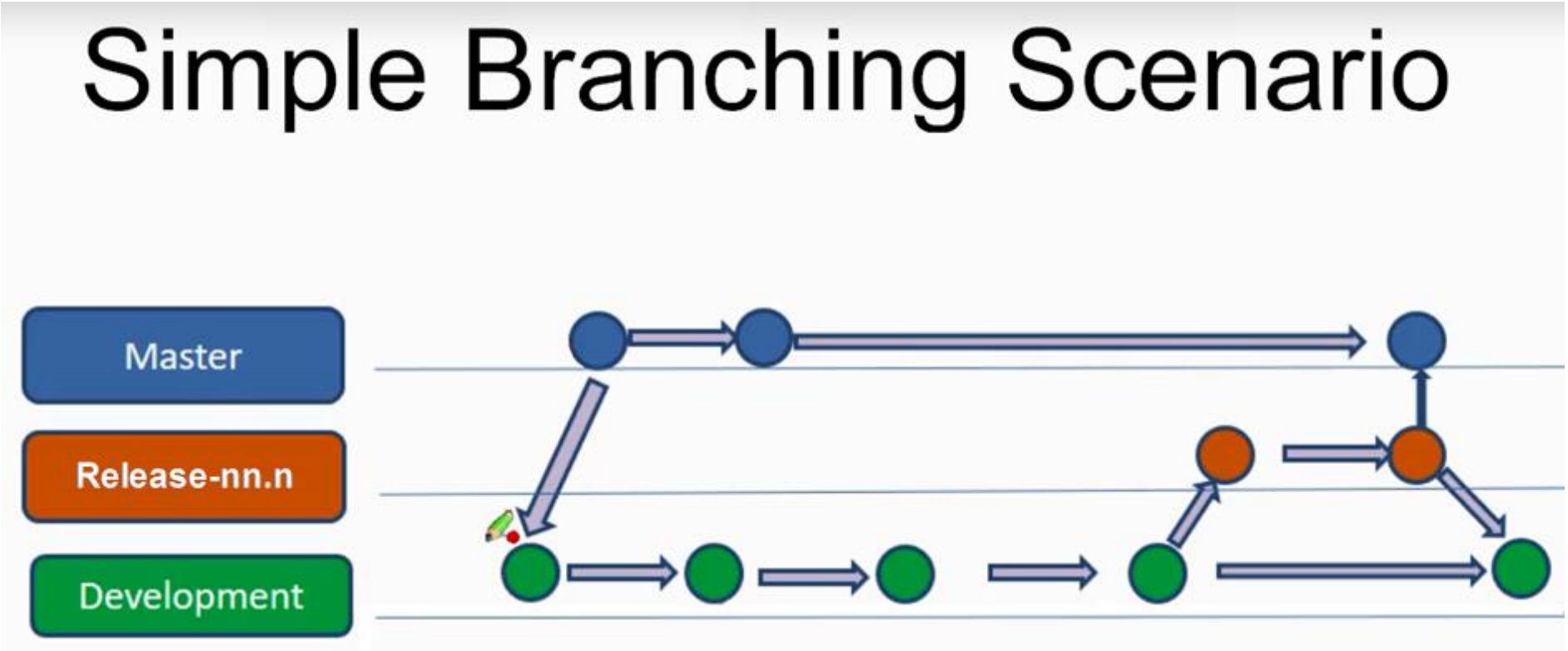
1. Pushes code changes to Git **Prd** Branch
2. Updates control spreadsheet: R_scripts_scheduled_for_Production.xlsx
3. Updates officialListOfPackagesPrd.txt with any new required libraries
4. Creates an RFC
5. Notifies Deployment Team a move to Prd is needed

Deployment Team

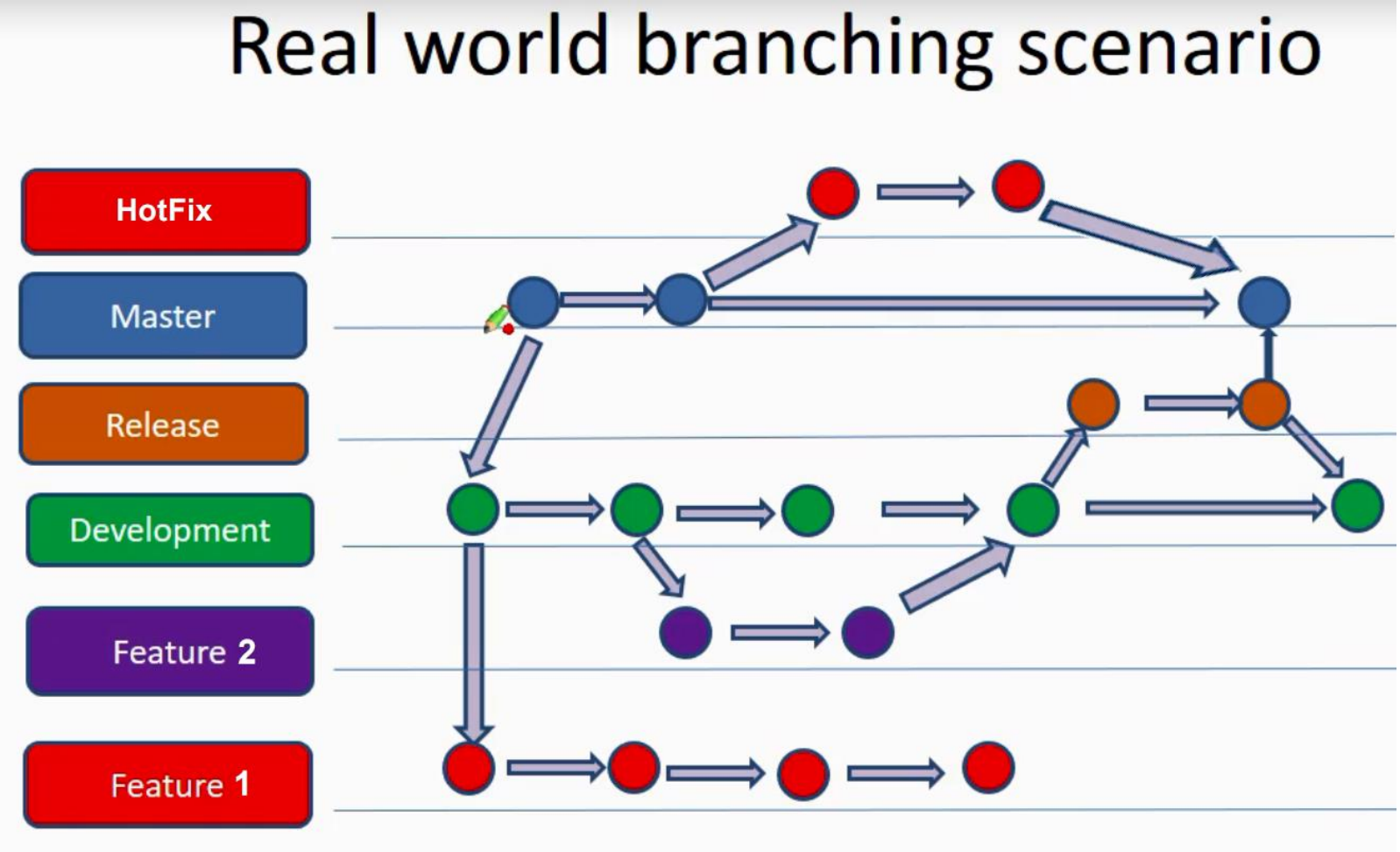
1. Ensures mini-library process runs
2. Using Git - Pulls code into MLS Prd Server "**Prd**" (local) branch
3. Creates/updates SQL Server Agent Job(s)

Appendix B

Simple Branching Scenario



Appendix C


Real world branching scenario
HotFix
Master
Release
Development
Feature 2
Feature 1

Appendix D

Description of Real Life Branching Strategy