Git: Best Practises

|  |  |
| --- | --- |
| Author: | Richard Stevenson |
| Document Name: | Git Kata |
| Document Location: | \\lukpc210\tbd |
|  |
| Date Created: | 24/03/2017 |
| Date Last Modified: | 24/03/2017 |
| Status: | Draft |
| Confidentiality: | All Lhasa Users (including contractors) |
| Circulation: | <Detail circulation list> |
| Document Version: | 0.2 |

Contents

[1. Overview 2](#_Toc478374754)

[2. Setup 3](#_Toc478374755)

[2.1. Download and install git 3](#_Toc478374756)

[2.2. Set your user name and user email 3](#_Toc478374757)

[3. Command Line Kata 4](#_Toc478374758)

[3.1. Clone the repository 4](#_Toc478374759)

[3.2. Create main line of development branch 5](#_Toc478374760)

[3.3. Create branches to work from 8](#_Toc478374761)

[3.4. Do task 1 10](#_Toc478374762)

[3.5. Merge first piece of work 17](#_Toc478374763)

[3.6. Do task 2 22](#_Toc478374764)

[3.7. Tidy up 28](#_Toc478374765)

[4. Eclipse Plugin Kata 29](#_Toc478374766)

[5. Appendix 30](#_Toc478374767)

# Overview

The purpose of this document is to provide people who have not used git before a repeatable exercise to practice its use. This will focus on how we use git within Lhasa. There are currently two ways to achieve the exact same outcome, using the command line and using the eclipse git plugin.

There are many other tools that are available to achieve the same goals, if you have a particular preference then feel free to add your own version of the Kata to this document. Intellij would be a good example of that.

Some parts of the workflow are common, where we use Bitbucket for certain operations. Once you have done this kata you should understand the following concepts:

* Git is a distributed source control system, we have LOCAL and REMOTE repositories
* What fetch, pull, push, merge and rebase are
* The concepts of commit objects
* How to commit code
* How to merge code back into the main line of development.

# Setup

If you have not used git before there are a couple of things that you need to do in order to set your machine up. These only have to be done once.

# Download and install git

Install the latest version of git onto your machine, you should be able to follow the instructions on the git website, <https://git-scm.com/>

# Set your user name and user email

Open Git Bash on your machine

Set your user name

$ git config --global user.name "Firstname Surname"

Confirm that you have set your name correctly with the following command.

$ git config --global user.name

Set your email address

$ git config --global user.email "your\_email@example.com"

Confirm that you have set your email address correctly with the following command.

$ git config --global user.email

# Command Line Kata

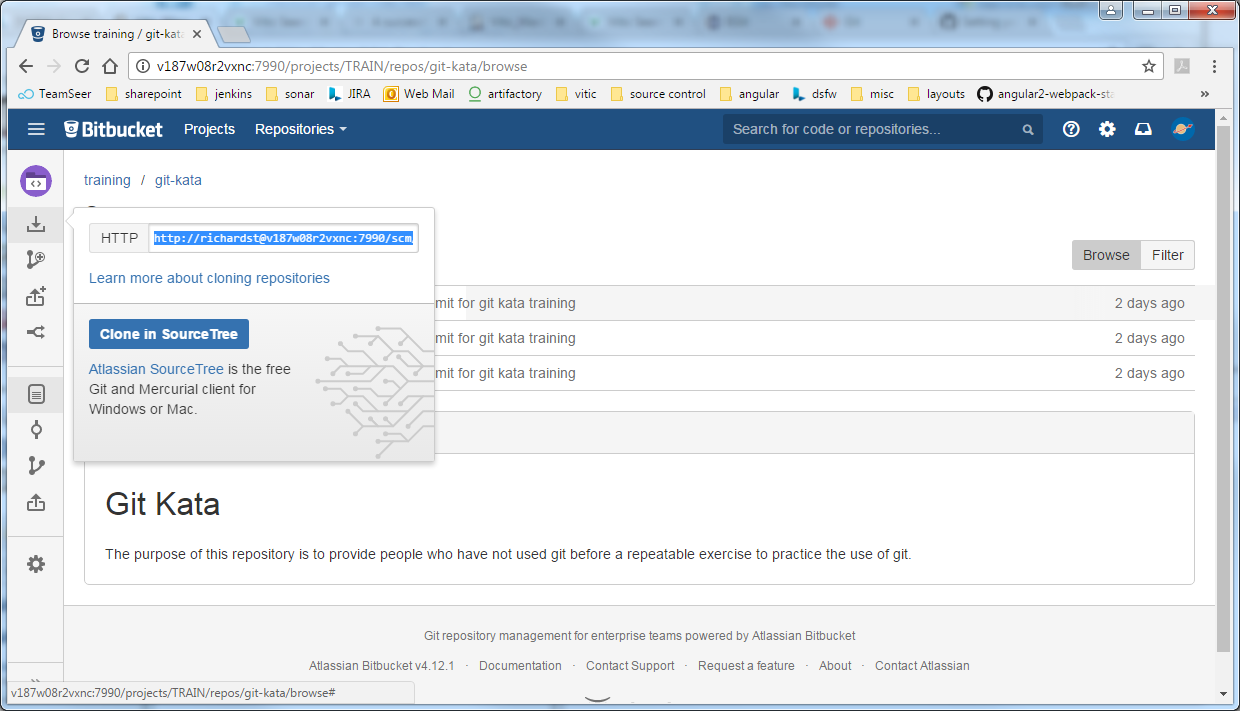
Using the command line might seem onerous but it gives a very simple view of what is happening without adding complexity of options that you don’t need. For each step you will need to open Git Bash on your machine.

# Clone the repository

This section simply copies the code to your local machine. We copy from what we will term the “REMOTE” repository to your “LOCAL” version of the repository. Git is a distributed source control system which means that you have a copy of the whole repository on your machine.

Navigate to the git kata repository on Bitbucket, click on the clone repository icon and copy the url.

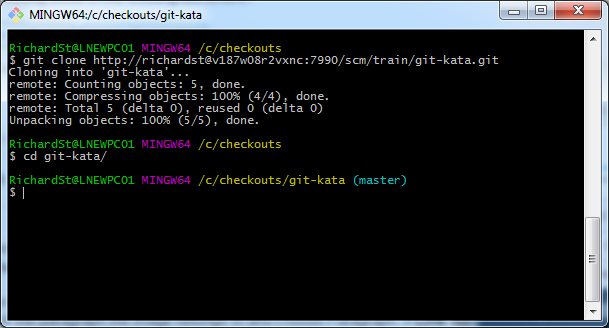
<http://v187w08r2vxnc:7990/projects/TRAIN/repos/git-kata/browse>



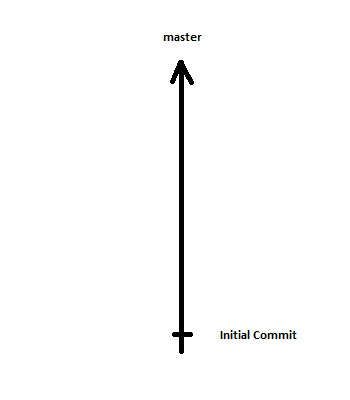
In git bash navigate to a folder you want your checkouts to be in and type the following:

$ git clone <url>

This will copy the whole repository from Bitbucket, the REMOTE, to your LOCAL machine. Now change directory into the git-kata folder and you should see something like the picture below. Note that Git Bash has some nice features, it shows you which branch you are on in light blue, by default we are on the master branch.



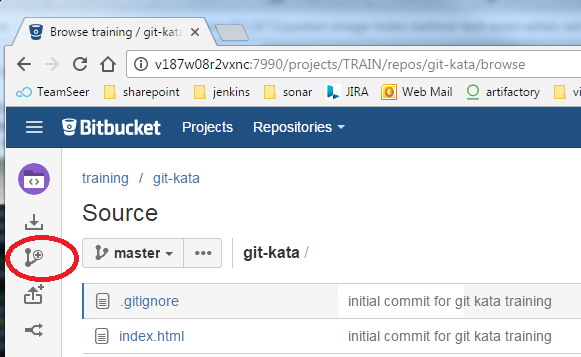
Our repository tree looks very simple.



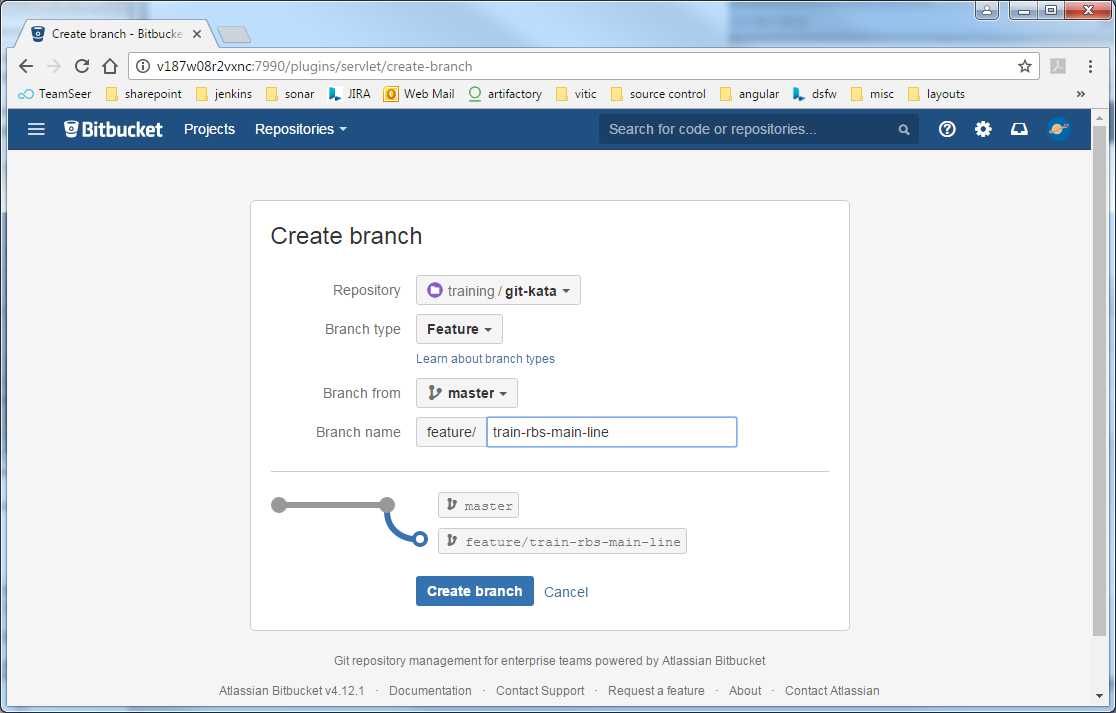
# Create main line of development branch

When we are working in git it is common to have a branch that is long running where a development team does all their work. Here we will create that main line of development branch from master, you should not ever commit directly to this main line of development branch or to master, we will do this later using the merge option provided by Bitbucket.

Navigate to the git kata repository in Bitbucket, and click on the create branch option.



And create a branch of the form feature/train-<initials>-main-line, click the create branch button



Now we get to learn about the REMOTE and LOCAL. We have just created our long running branch but this only exists on the REMOTE. If we search for our branch on our LOCAL we will not find it yet, so let us do that. Type the following into git bash:

$ git branch –a | grep <initials>-main-line

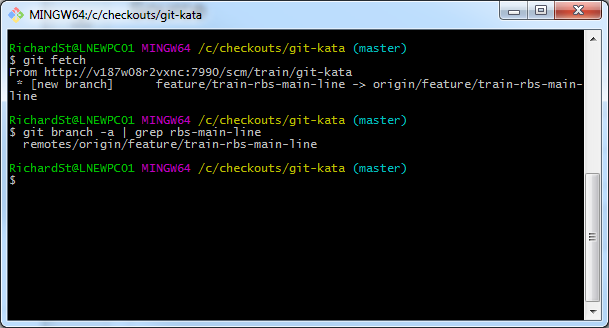
And note that there are no matches.

Now we will “fetch” the all latest things from the REMOTE that we don’t have on our LOCAL. This will not only copy the code that we don’t have locally but will update the remote tracking branches and pointers on those tracking branches, but it won’t update our checked out branches (more of that later).

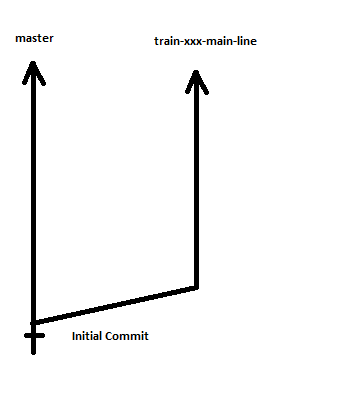
Type the following into git bash:

$ git fetch

Now search again for your branch, and you should find it in your LOCAL.



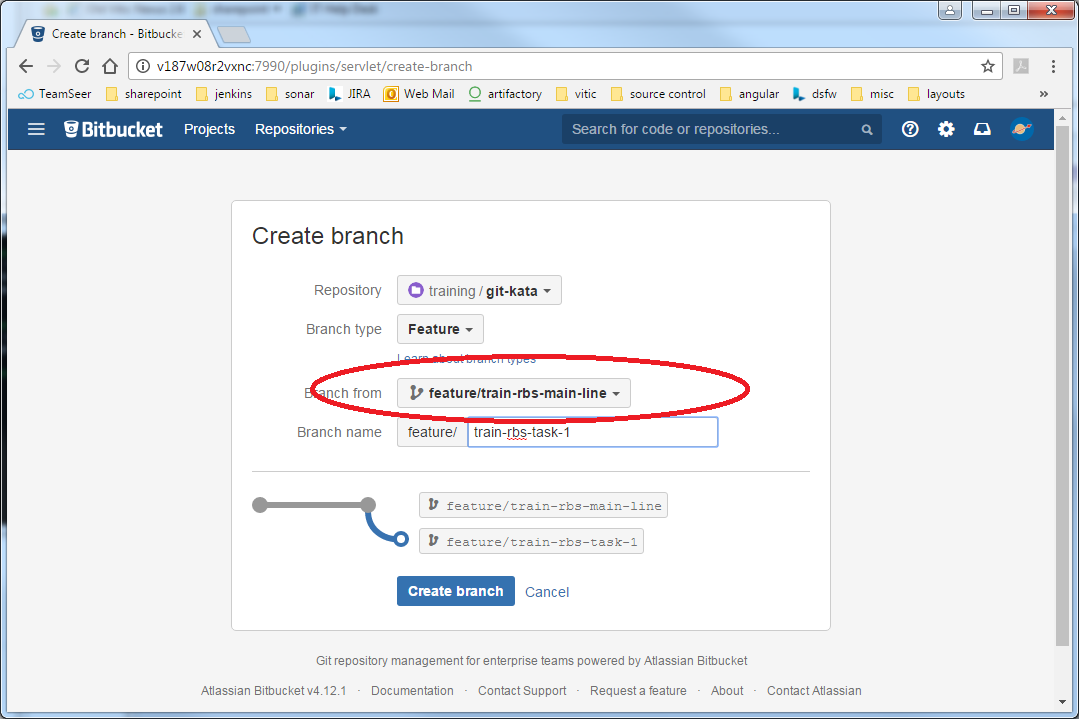
Now our git tree looks like the following on both the REMOTE and the LOCAL:



# Create branches to work from

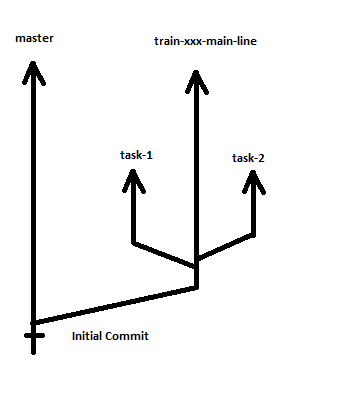
Now we actually want to make some changes, to do this we will create two branches from our main line of development. These allow us to code in isolation from other developers and have our code reviewed before merging our code to the main line of development.

Go back to the create branch screen in Bitbucket and create a branch from your main line of development of the form feature/train-<initials>-task-1, note it important to check where you are creating your branch from



Now create another branch in the same way, this time of the form feature/train-<initials>-task-2

On the REMOTE our git tree now looks like:



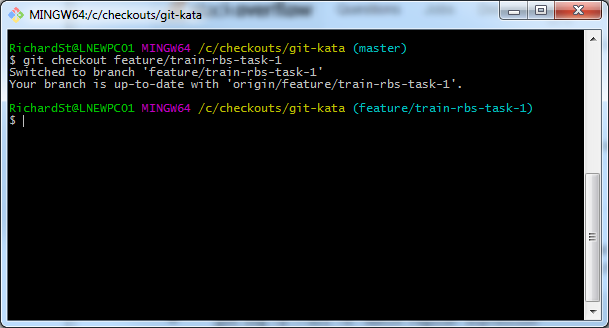
So far so unremarkable, we have not yet done any work. So let us get ready to do that. “fetch” the latest changes from the REMOTE as you did before and check that you can see these new branches in your LOCAL repository. Now your LOCAL is up to date with you REMOTE and the git trees will look the same on both.

# Do task 1

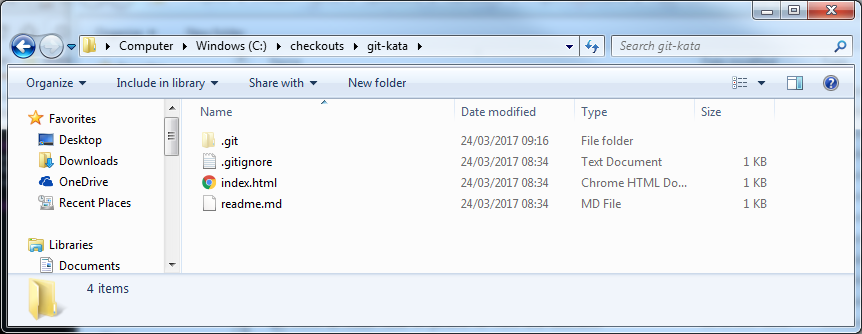
Here we will learn how to “commit” code and how to “push” those changes to the remote machine.

Checkout your task-1 branch. In Git Bash type:

$ git checkout feature/train-<initials>-task-1



If we look at the files in the directory we should see the following:



* .git is the git folder that represent the repository
* .gitignore is the file that tells git what not to commit, things like build files and IDE specific files
* index.html is the file that we will alter and the readme.md has some simple information about our project

Open index.html with your favourite editor and add another paragraph to the body so that it looks like this.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Git Kata</title>

</head>

<body>

<p>This is the git kata index file</p>

<p>Today I am learning about git</p>

</body>

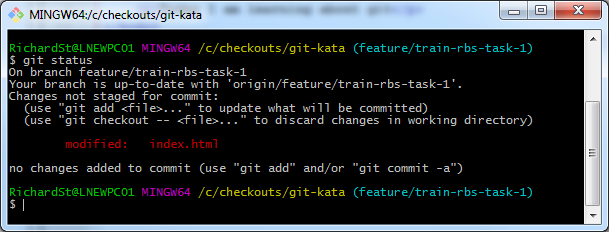
</html>

To see that we have made changes type the following into Git Bash

$ git status

It’s a command that is very useful as it shows the state of the branch and gives us helpful hints about what we should do.

Here we can see the file we have modified in red.



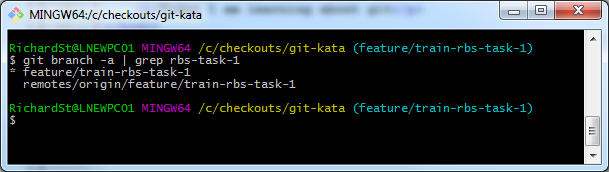
Let’s see what changes we have made, type the following into Git Bash

$ git diff

This shows us the changes that we have made.

The command git status also tells us that our checked out branch is up to date with the branch that is tracking the remote version of the branch (now that we have checked out our branch there are effectively two versions of it on our LOCAL, our checked out one where we do our work and one that tracks the REMOTE version) type the following into Git Bash to see the two branches

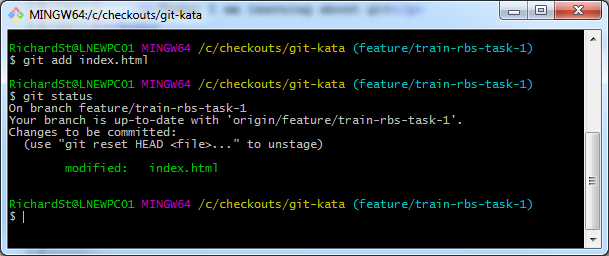
$ git branch –a | grep <initials>-task-1



As the status command suggested we might want to add the file we changed to the staging area to commit it. Do as it suggested and type the following to add the file.

$ git add index.html

Now check the status again, we can see the files ready to commit, in the staging area, in green.



At this point let us make another change to the file, we can say we are learning about git in the command line.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Git Kata</title>

</head>

<body>

<p>This is the git kata index file</p>

<p>Today I am learning about git in the command line</p>

</body>

</html>

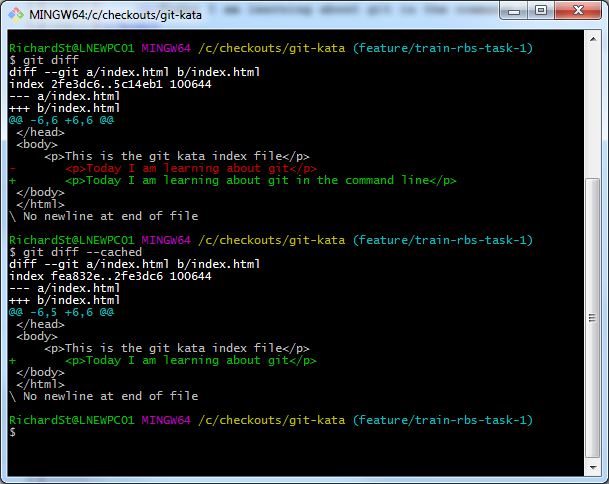
Do a git status. Now we see the same file in the staging area and in the files not staged. What is going on?

We have told git that we are happy with the initial version so this has been copied to the staging area ready to commit, but the working version has also been changed.

To see the differences show the changes to the un-staged file as before.

When you have done that type

$ git diff -–cached



We can see the first one holds our current changes and the second one shows the changes we made the first time around, when we moved the changes to the staged area.

Let us say we were happy with the first attempt, to get rid of the current changes type

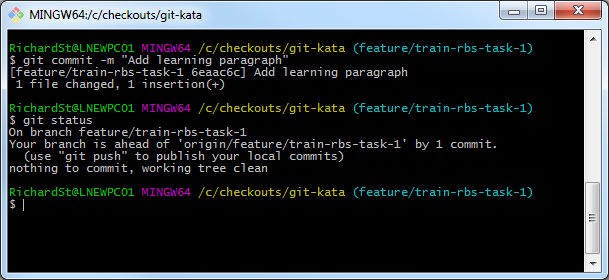
$ git checkout -- index.html

Run the status command to see that we only have changes in the staged area (green)

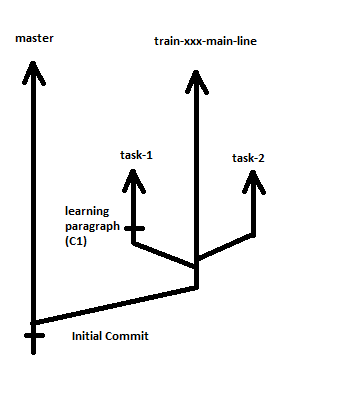
Now let us commit, for brevity we will add the commit message in the commit command.

$ git commit –m “Add learning paragraph”

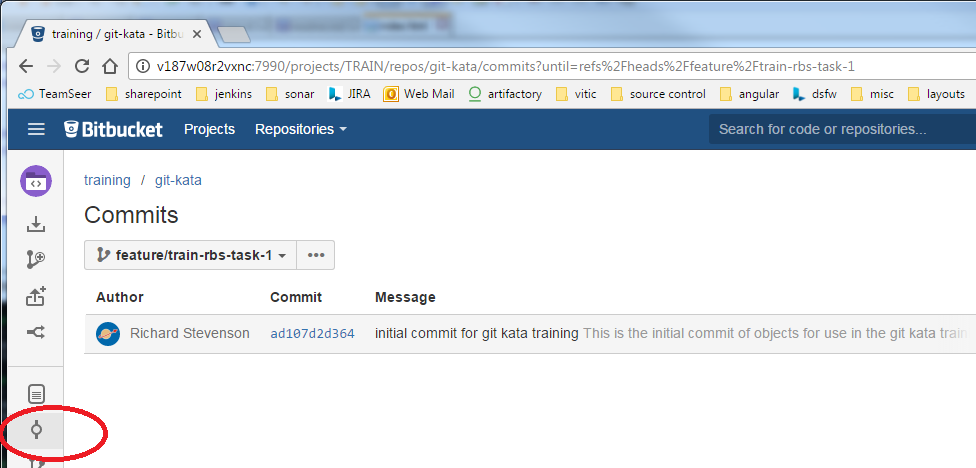
Now look at the status, it tells us we are now 1 commit ahead of the origin, because it has the branch tracking the REMOTE it knows that we are one commit ahead.



Let’s explore where we are. Our LOCAL tree looks like this.



We can view the remote commits on our branch to confirm that this commit does not exist on our REMOTE. Navigate to your branch on Bitbucket to check.



You should only see the initial commit that was made when setting up the repository.

We can view the commits on our LOCAL. Type the following into Git Bash

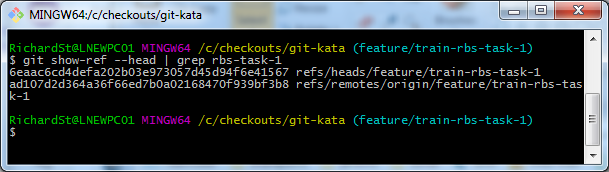
$ git log

There should be 2 commits, the original and the one we just made. The long hashcode is the way that git marks each commit object, usually you just need the first 6 characters of this to identify the commit.

To see the difference between our checked out version and the remote tracking branch type the following

$ git show-ref –-head | grep <intials>-task-1

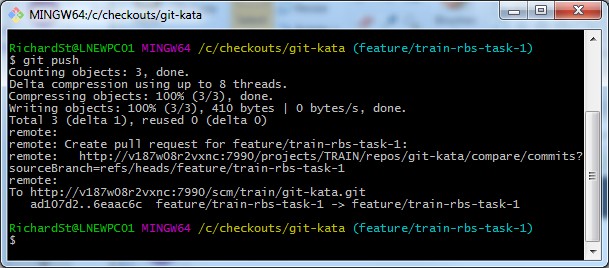
Here we can see what commit objects are at the HEAD of these branches. The REMOTE has a different commit object at its HEAD than the LOCAL branch, as we would expect as we have not yet told git to send those new changes to the REMOTE.



We now want to push our changes to the REMOTE

Type the following into Git Bash

$ git push

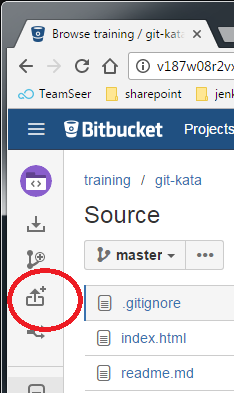


If we now go and view the commits on Bitbucket for our branch we should see our commit object, we can also explore the files to see our changes.

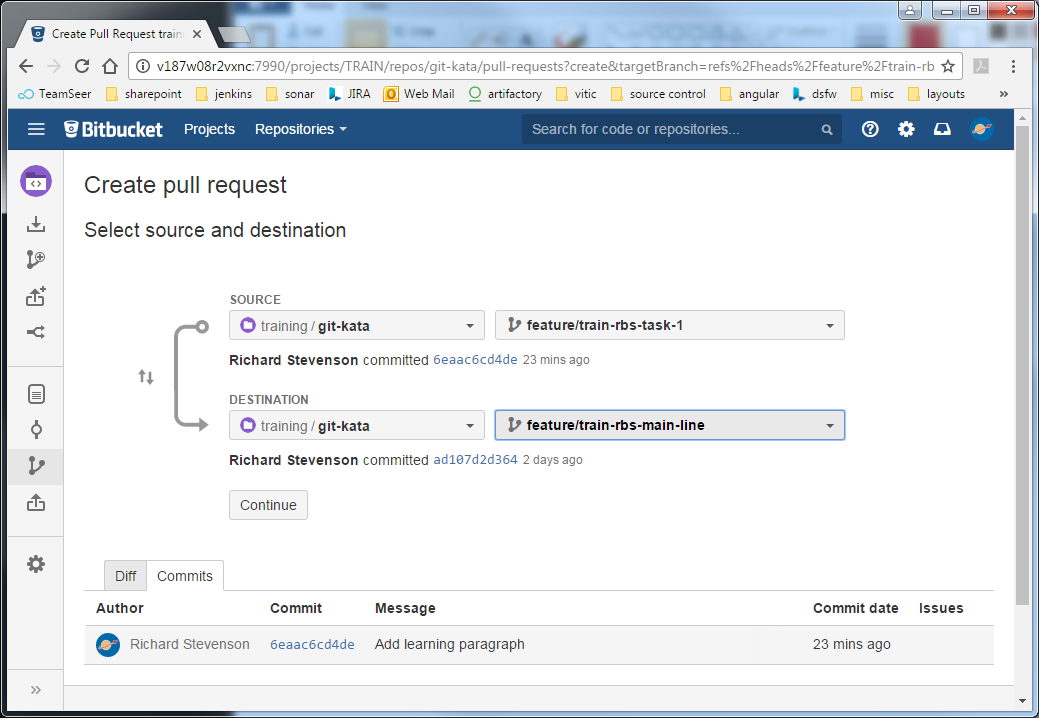
# Merge first piece of work

The standard development workflow of code reviews does not form part of this document. However as stated before we should not merge directly into the main line of development or master. We do this using a pull request. Here we will omit someone actually viewing and approving that pull request.

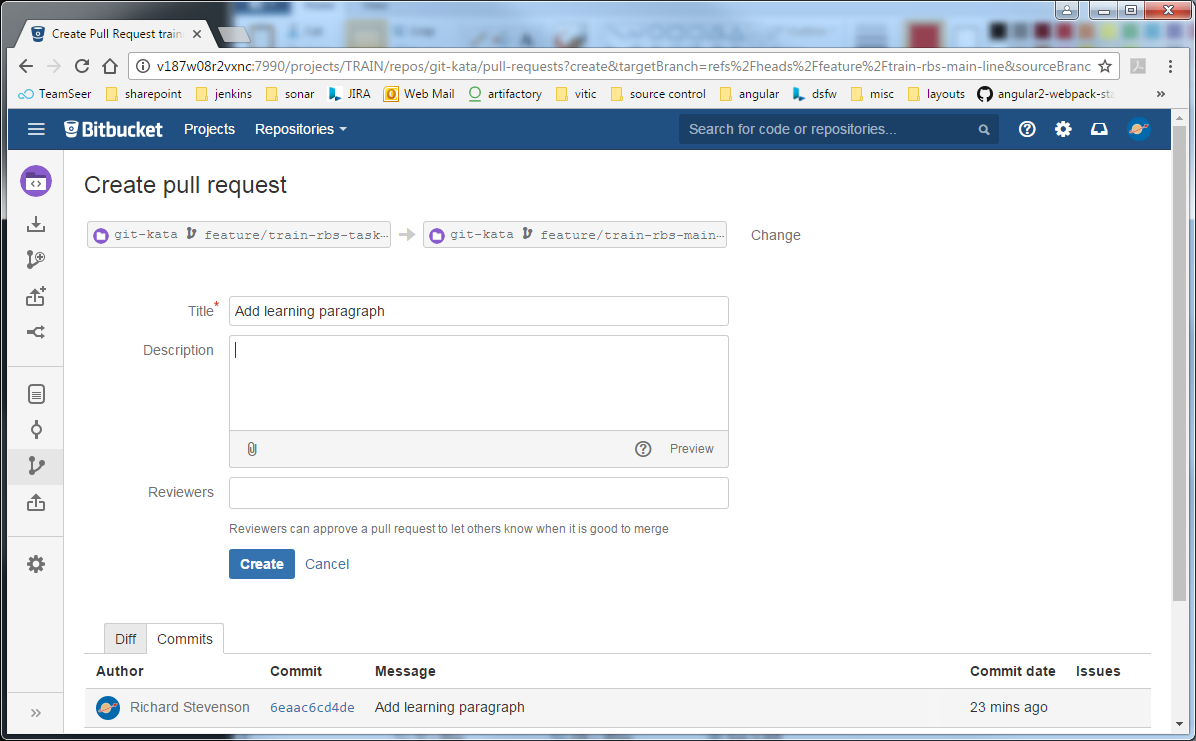
Navigate to Bitbucket and click on the create pull request icon



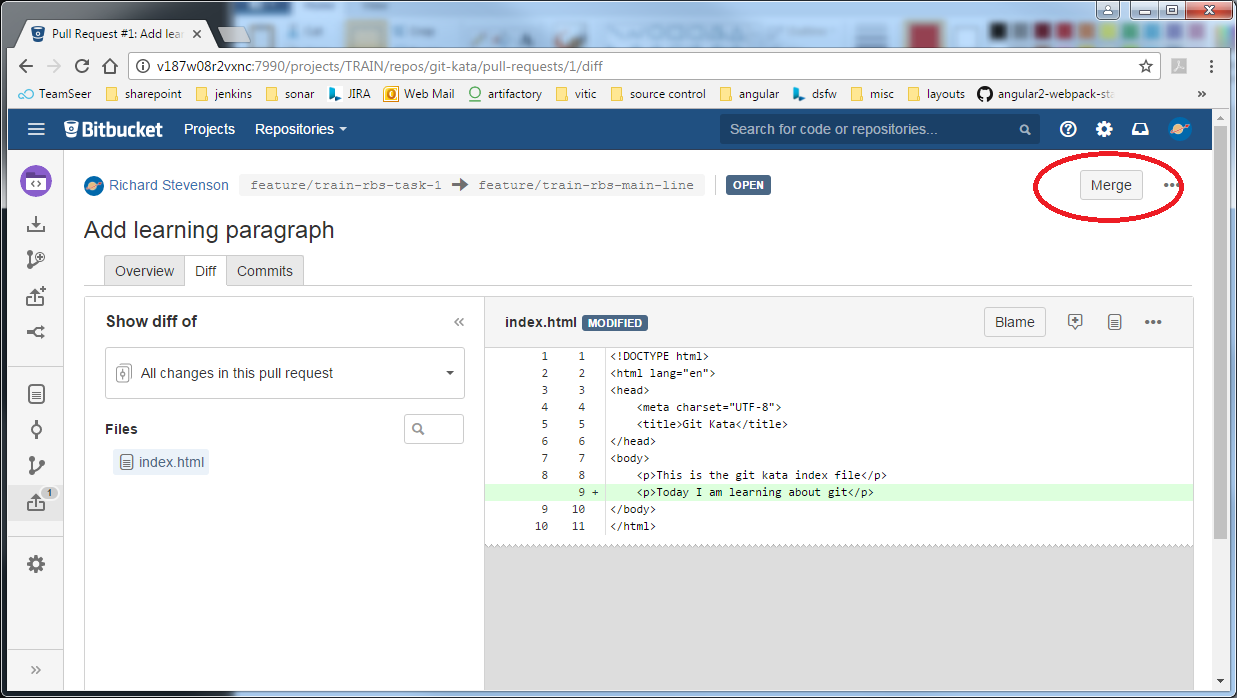
Now fill in the source of your pull request (which is your task branch) and your destination branch (which is your long running mainline of development).



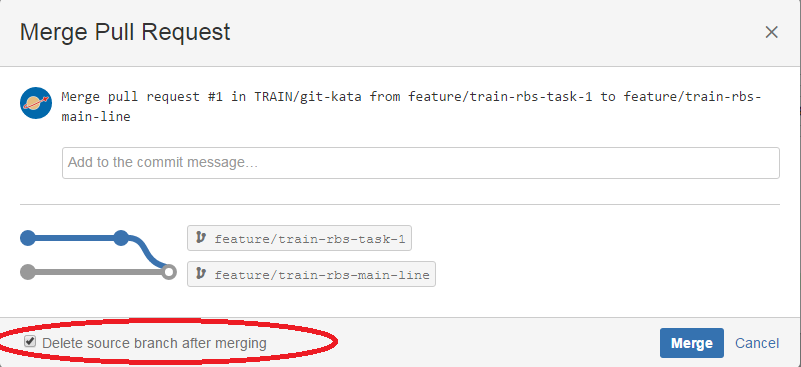
Click “Continue” and then “Create” on the next page



This will take you to the pull request. Here you can view the changes that you have made, make comments, check that you are merging from the correct task branch to the correct target branch etc. For the purposes of this document we simply want to click on the “Merge” button.

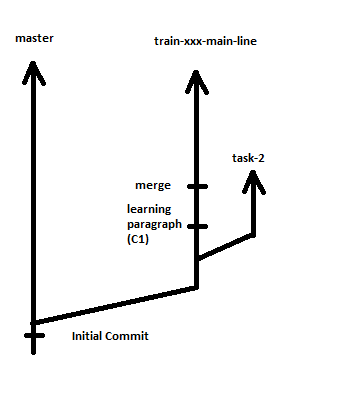


Make sure you tidy up after yourself by deleting the task branch, you can check the checkbox on the dialog to do just that.



Let’s see where we are now.

The REMOTE git tree now looks like this.

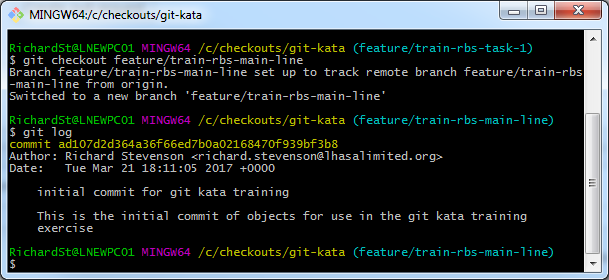


Let’s check our LOCAL. The main line development branch does not have the commit C1 or the merge commit in it.

Checkout the main-line branch locally and view the history

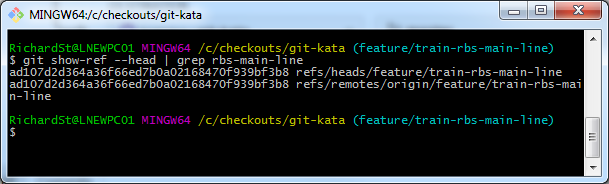
$ git checkout feature/train-<intials>-main-line

$ git log



Let’s see this in the remote tracking branch

$ git show-ref –-head | grep <intials>-main-line

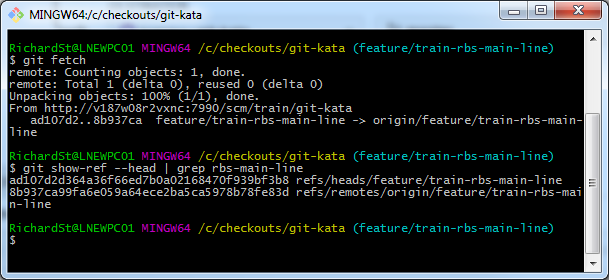


This shows that both the LOCAL versions are pointing at the initial commit on master.

This next bit is quite fundamental to how git works. Let’s fetch everything from the REMOTE as we did earlier, then see where those main line branches point to

$ git fetch

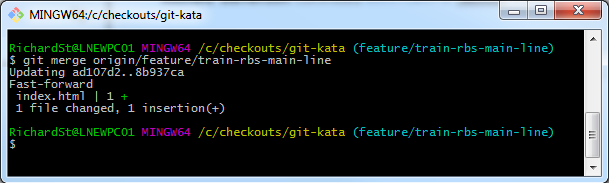
$ git show-ref –-head | grep <intials>-main-line



We now can see that the remote tracking branch points where we would expect, to our merged changes but our checkout out branch still only has the initial commit that exists on master, you can do a git log to see that

We now do a “merge” to bring the changes from the tracking branch to our checked out version. Type the following into Git Bash

$ git merge origin/feature/train-<initials>-main-line



Now we can use show-ref and log commands to see that our local checked out version is where we expect it to be, at the same reference as the REMOTE and the remote tracking branch.

Note that here we could have done a git pull which does a git fetch and git merge all in one command

# Do task 2

You should now be able to do the following without much help:

1. Checkout train-<initials>-task-2 branch on your local machine
2. Change the title of the page

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Git Kata - Tutorial</title>

</head>

<body>

<p>This is the git kata index file</p>

</body>

</html>

1. Add this change to staged area
2. Commit the change with the following message “Change Title”

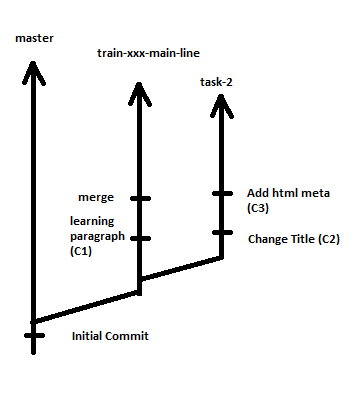
Now we decide that we need to do some more work for this task so we make another change and commit that.

1. So add the following meta information into the page, on the line below the current meta tag

<meta name="keywords" content="git">

1. Add this change to the staged area
2. Commit the change with the following message “Add html meta”

We now have a LOCAL git tree that looks like this:



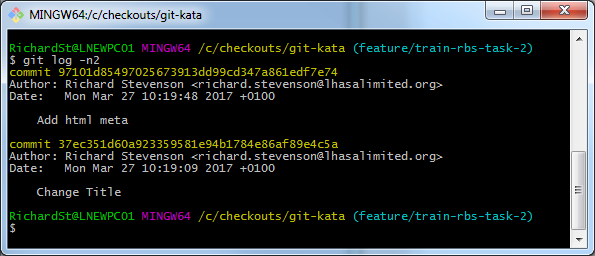
Now we want to merge our changes on branch task-2 into the main line of development. To do this neatly we need to do two things:

* Squash our changes
  + This makes the git tree neater when viewing the history, one commit per task
  + It makes the rebase easier, as we only have to replay one commit object over other changes (and fix merge conflicts for one commit object)
* Rebase with the main line of development
  + Makes sure we run our code with any other code that has been committed since we took our branch
  + Makes the git tree neater

So let’s squash our two commits. Before we do let’s note down the two commit object references that we have for C2 and C3. Type the following:

$ git log –n2

And write down the first 6 characters of the commit hash for each of the commits, in the case below 97101d and 37ec35.

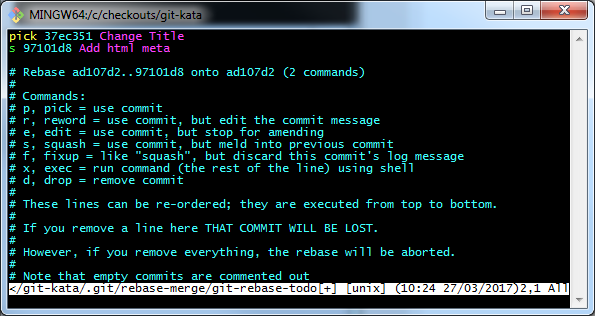


Now to squash the two commits together. Type the following into Git Bash, where 2 is the number of commit objects we want to squash

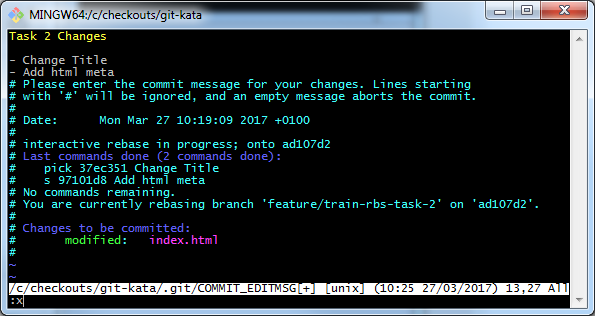
$ git rebase –i HEAD~2

We then follow the interactive guide by selecting which commits we want to keep (pick) and which we want to squash (s). This is a vi editor, so if you are unfamiliar with vi follow these commands very carefully.

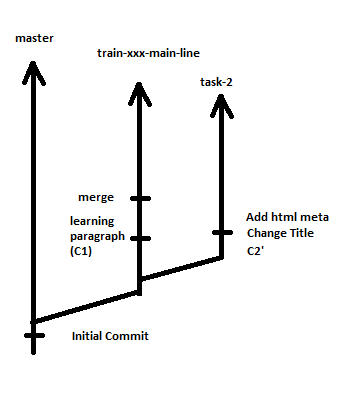
* Use arrow keys to move the cursor to the beginning of the line of the second pick option
* Press the ‘x’ key 3 times to delete 3 characters
* Press the ‘r’ key followed by ‘s’ to replace the remaining character of the work pick with an s
* Press ‘:’ followed by ‘x’ followed by ‘return’ to save and quit
* On the next page simply do ‘:’ followed by ‘x’ and ‘return’ to save and quit, if you are familiar with vi then feel free to edit this commit comment so that it makes more sense.



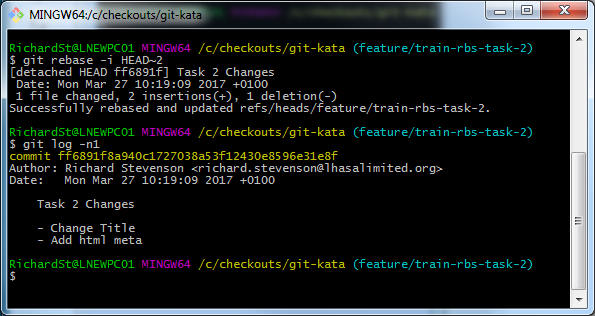
We then amend the git commit message to something that is more representative



Now we have one commit on our branch and our LOCAL git tree looks like this.



It is really important to note that this commit object C2’ is a different commit object to the one that was there before. Do another git log and check the commit objects hash code to see that it is indeed a different commit object. Also write the first 6 characters down again.



In this case ff6891.

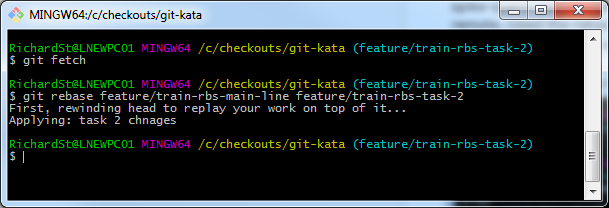
Now we want to rebase from our main line of development, this will effectively mean that we have the other changes from task one on our branch.

First we will fetch all changes, this makes sure that the remote tracking braches are up to date with the REMOTE:

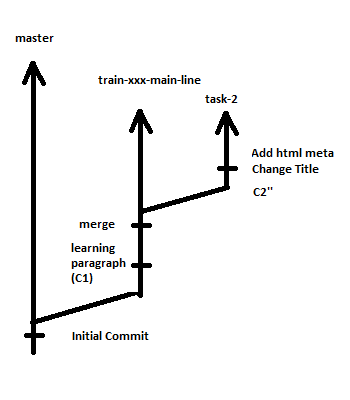
$ git fetch

Now we will rebase from the main line

$ git rebase feature/train-<initials>-main-line feature/train-<initials>-task-2



Now our local git tree looks like this



Note it is important to understand that C2’’ is a different commit object than C2 and C2’. Do a git log and compare git commit hash codes to confirm this

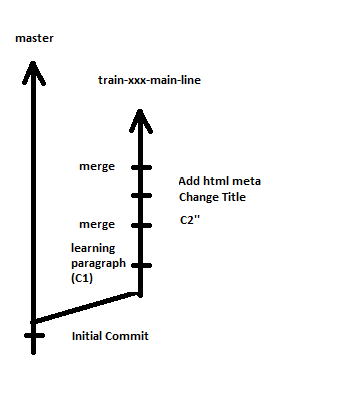
Now we can run our tests, or eyeball the html file to see if it has all our changes in it.

We can now do what we did before:

* Push our changes to the remote
* Raise a merge request
* Merge the changes into the main line of development (deleting our task-2 branch)

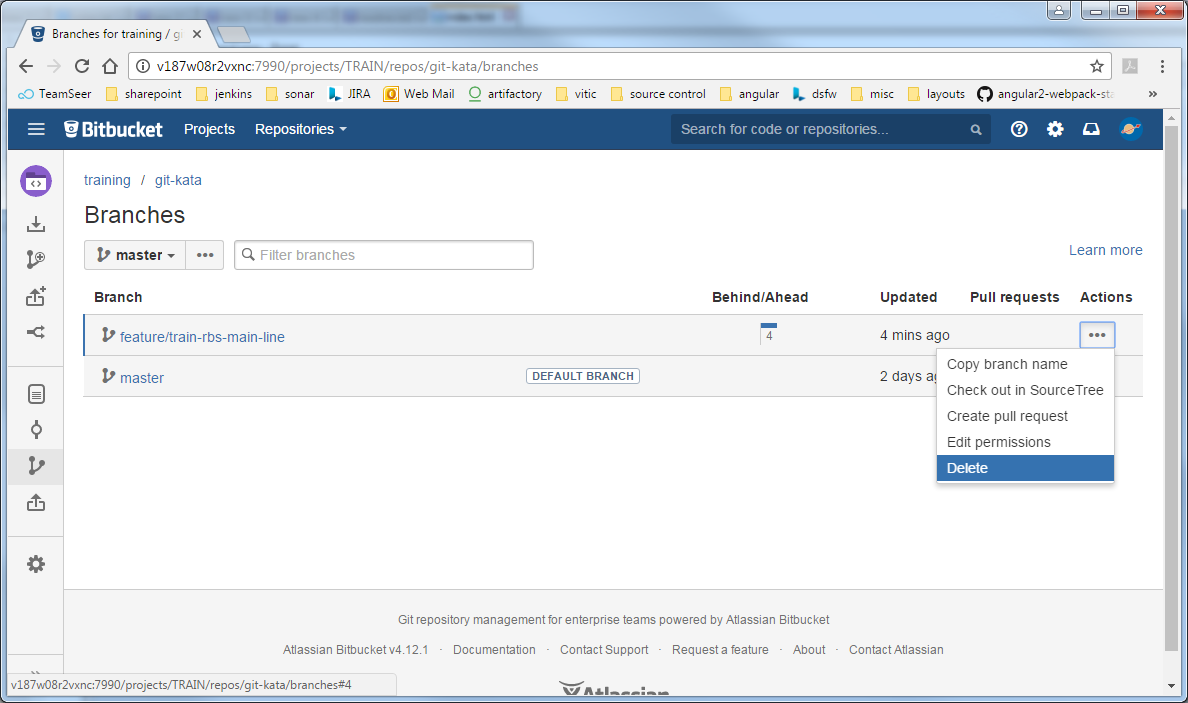
You can then check on the REMOTE in Bitbucket that all the commits are where they are supposed to be.

The git tree now looks like this:



# Tidy up

To leave the repository in the same state as it was before, please delete your main line branch. Do this buy navigating to the git-kata repository on Bitbucket and select the delete option.



# Eclipse Plugin Kata

Using the eclipse plugin gives you a nicer graphical experience with some added bells and whistles that you don’t need.

# Appendix

Egit – the eclipse git plugin <http://www.eclipse.org/egit/>