# Salesforce DX Workshop

MN Developer Group Meetup 3/2

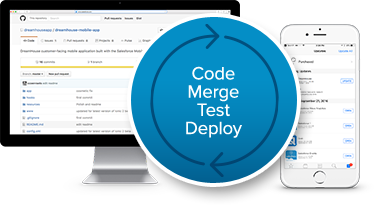


Figure - Photo Credit: <http://developer.salesforce.com>

Author: James Loghry (Demand Chain)

Date: 2/22/2018

# Introduction

During this workshop, you’ll learn how to interact with both source control and Salesforce DX.

Table of Contents

[Salesforce DX Workshop 1](#_Toc515017365)

[Introduction 2](#_Toc515017366)

[Pre Workshop Homework 3](#_Toc515017367)

[Equipment 3](#_Toc515017368)

[Install Git 3](#_Toc515017369)

[Windows 3](#_Toc515017370)

[Mac 3](#_Toc515017371)

[Other System? 3](#_Toc515017372)

[Install Salesforce DX 3](#_Toc515017373)

[Sign up for a developer hub org 4](#_Toc515017374)

[Step 1: Setting up the Source Control environment 5](#_Toc515017375)

[Fork the Repository 5](#_Toc515017376)

[Clone the repository 5](#_Toc515017377)

[Step 2: Connecting to the Dev Hub 7](#_Toc515017378)

[Step 3: Fixing the First Issue 8](#_Toc515017379)

[Create a Scratch Org and Populate With Your GitHub Repository 8](#_Toc515017380)

[Make the Fix 8](#_Toc515017381)

[Step 4: Fixing the Second Issue 14](#_Toc515017382)

[Step 5: The Pull Requests 18](#_Toc515017383)

[Step 6: Push it. Push it Real Good! 19](#_Toc515017384)

[Step 7: Extra Credit: Create a Continuous Delivery System with Bitbucket Pipelines and Salesforce DX 20](#_Toc515017385)

[Create and Configure the Connected App 20](#_Toc515017386)

[Create a Bitbucket account and Set Up Pipelines 20](#_Toc515017387)

[Kicking off the Pipeline 24](#_Toc515017388)

[Additional Tools and Resources 26](#_Toc515017389)

[Salesforce DX Documentation 26](#_Toc515017390)

[Trailhead Modules 26](#_Toc515017391)

[Salesforce DX Related Tools 26](#_Toc515017392)

[Salesforce DX Around the Community 26](#_Toc515017393)

# Pre-Workshop Homework

Before you arrive or get started with the workshop, please take time to complete the following steps.

## Equipment

Make sure you bring a laptop to the workshop, so you can follow along. You should be able to install software (like the Git and DX command line interfaces or CLIs) on the laptop. If you’re unable to follow along during the workshop, you may take this print out home with you and follow along later.

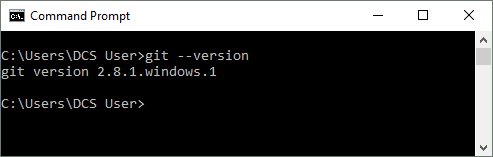
## Install Git

Install the Git command line.

### Windows

Download and install the file: <http://git-scm.com/download/win>

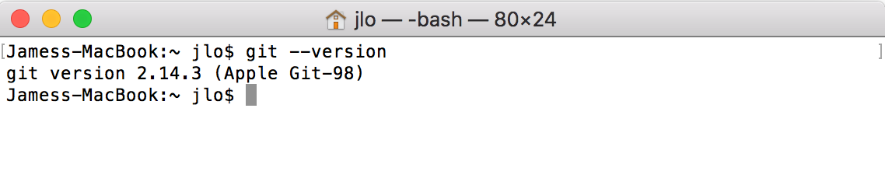
Note: Not sure if you already have git? Open the Windows command prompt and type git --version. If You see git version <version>, you’re in business.



### Mac

Download and install the file: <http://git-scm.com/download/mac>

Note: Not sure if you already have git? Open the Terminal and type git --version. If You see git version <version>, you’re in business.



### Other System?

Follow the instructions here: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git> if you are installing git on a different system.

## Install Salesforce DX

1. Install Salesforce DX, by choosing your operating system (should be preselected) and following any on screen instructions: <https://developer.salesforce.com/tools/sfdxcli>
2. Open the command prompt (Windows) or Terminal (Mac) and type SFDX update to update it to the latest version.

## Sign up for a developer hub org

If/when you or your company decide to use Salesforce DX for your build environment / developer experience, you can turn the production org into a “Dev Hub” org.

For this workshop however, you will provision your own dev hub org, so you don’t interfere with production.

Click here to request a developer hub: <https://developer.salesforce.com/promotions/orgs/dx-signup>

Let’s get started!

# Step 1: Setting up the Source Control environment

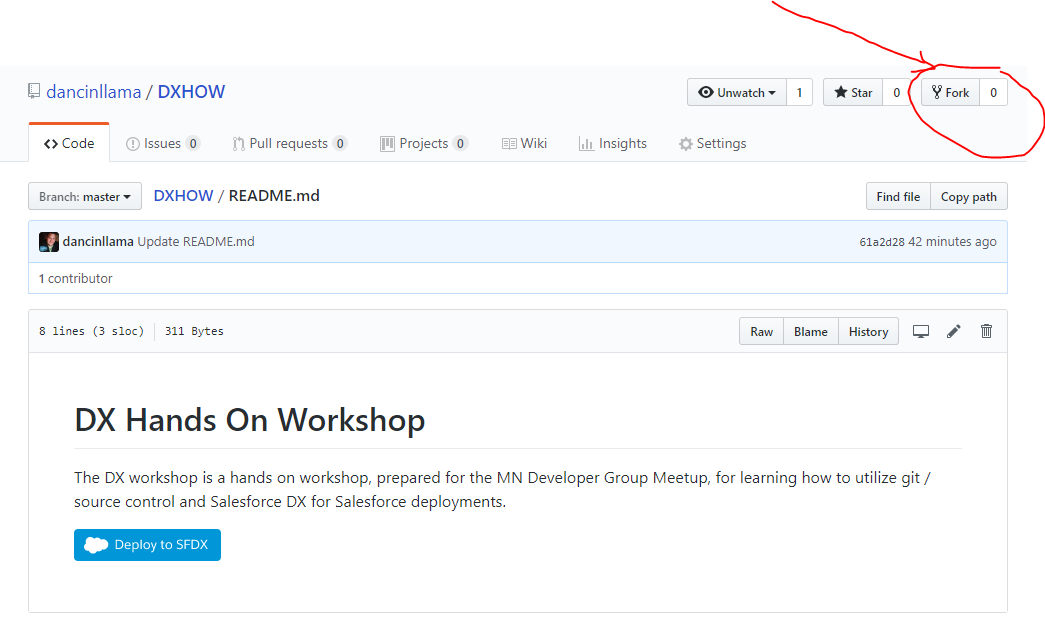
For this workshop, we have two production issues that need to be solved during the same sprint. For the first issue, a developer or admin needs to add a list view called “All Accounts”. For the second issue, a tester found a typo on the lightning home page that we need to address and fix.

As the lead developer, you’ll start with a source control repository, and by forking (or creating a copy of an existing repository), you’ll work on each issue in separate scratch orgs. Once the fixes are completed, you’ll promote them using Salesforce DX.

## Fork the Repository

Your first step is to fork the production.

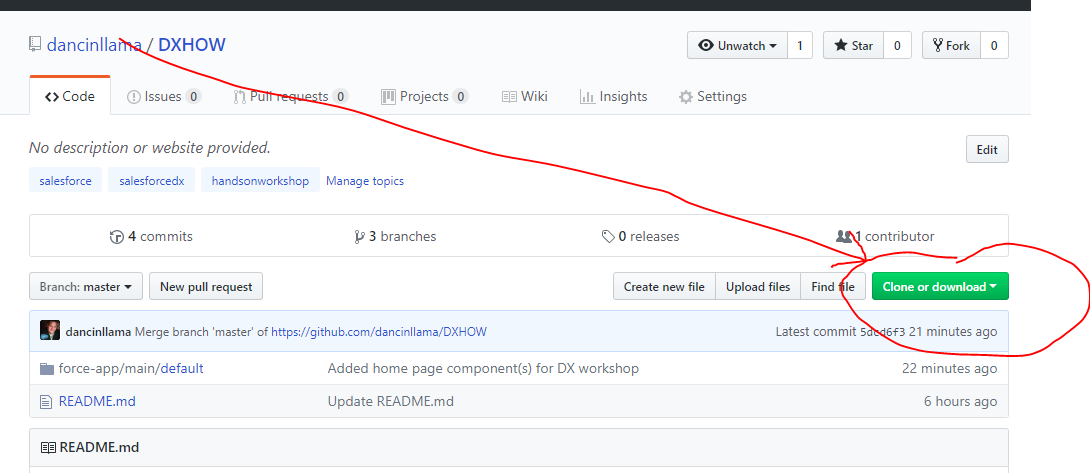
1. In a browser, open <https://github.com/dancinllama/DXHOW>
2. Log into your GitHub account that you created as part of the pre-work.
3. Click on the “Fork” button in the upper right-hand corner.
4. If prompted, verify your e-mail address first by clicking the Verify link sent your GitHub account’s e-mail address.



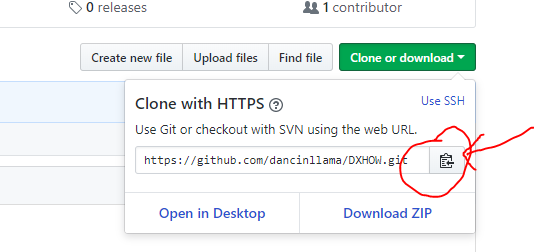
## Clone the repository

When you clone the repository, you’re downloading the code from the repository to your laptop. After forking the repository, create two branches, one for each issue we need to work.

1. Click the “Clone or download” button in the top right of the screen.



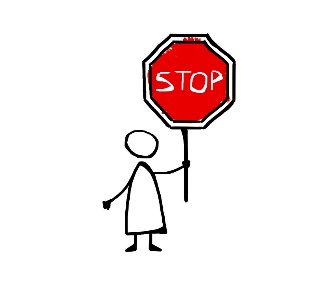
1. Copy the URL (https:// …git) (Or Click the Clipboard icon)



1. Open your command prompt (Windows) or Terminal (Mac).
2. Change to the directory where you want to store this project, for instance C:\Users\<username>\workspace.
3. Use the following command to copy the files to your workspace:

git clone <URL from step 4 goes here>

1. Enter your username and password if needed.
2. Use “dir” (Windows) or “ls” (Mac) to see if your folder now contains a directory called “DXHOW”



# Step 2: Connecting to the Dev Hub

In the pre-work, you already signed up for a dev hub trial org (good for 30 days). Before creating the scratch orgs to fix the issues though, you’ll need to set up salesforce dx to use your dev hub.

1. Open the command prompt (Windows) or Terminal (Mac) and issue the following command, to authenticate with your dev hub. The -d take denotes that is the default dev hub to use, and the -a flag denotes a shorthand or alias that we can use going forward.

Note: You can change the default later if needed.

sfdx force:auth:web:login --setdefaultdevhubusername -a DevHub

# Step 3: Fixing the First Issue

Now that your local environment is set up and the project has been cloned to your desktop, it’s time to get to work. First,

## Create a Scratch Org and Populate with Your GitHub Repository

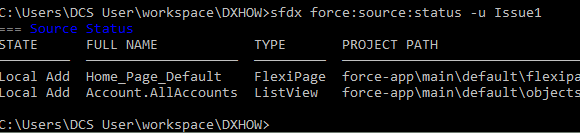
1. Create a scratch org. This command will create a scratch org with the alias of “Issue1”, which we will use later.

sfdx force:org:create -f ./config/project-scratch-def.json -a Issue1

1. First, let’s view the status of our local workspace and scratch org:

sfdx force:source:status -u Issue1

Because you downloaded code from GitHub, and have NOT synched those changes to your dev hub or scratch org, you should see two changes listed:

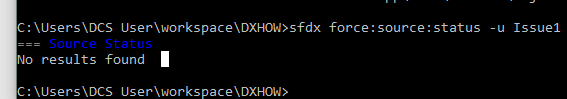


1. Since we’re working with a fresh dev hub and a new scratch org without any information, let’s go ahead and push the code we downloaded from GitHub to your scratch org. (Please note, we’re NOT pushing anything to source control just yet)

sfdx force:source:push -u Issue1

1. Reissue the status command. Having pushed all changes to the scratch org, you should see “No results found”, since there are no differences between your local workspace and the scratch org.

sfdx force:source:status -u Issue1



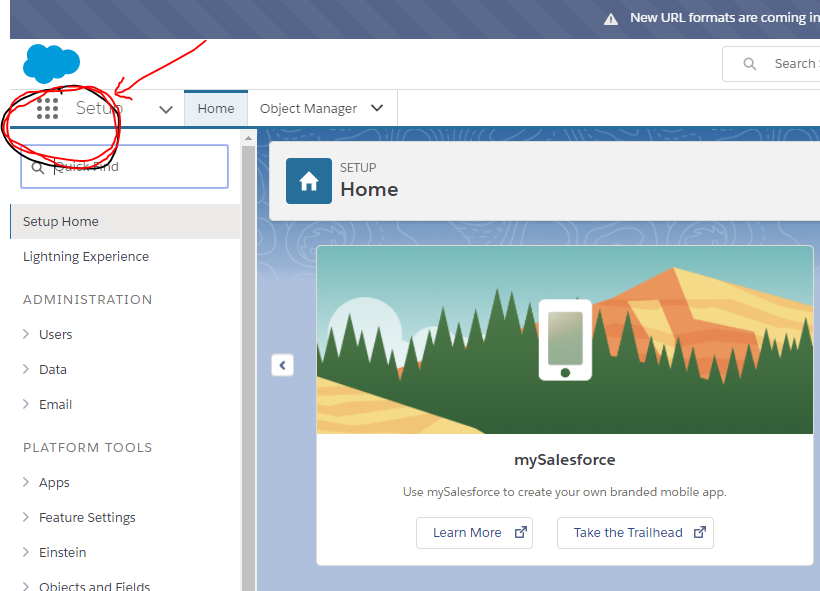
## Make the Fix

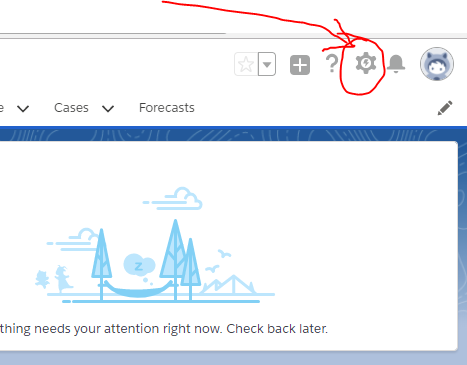
Congratulations! We’re getting close to resolving the first issue. We’ve created a scratch org with code from our application’s source control repository. Next, it’s time to fix a typo one of our admins made! We’re going to open our scratch org and fix the typo in a component on the home page.

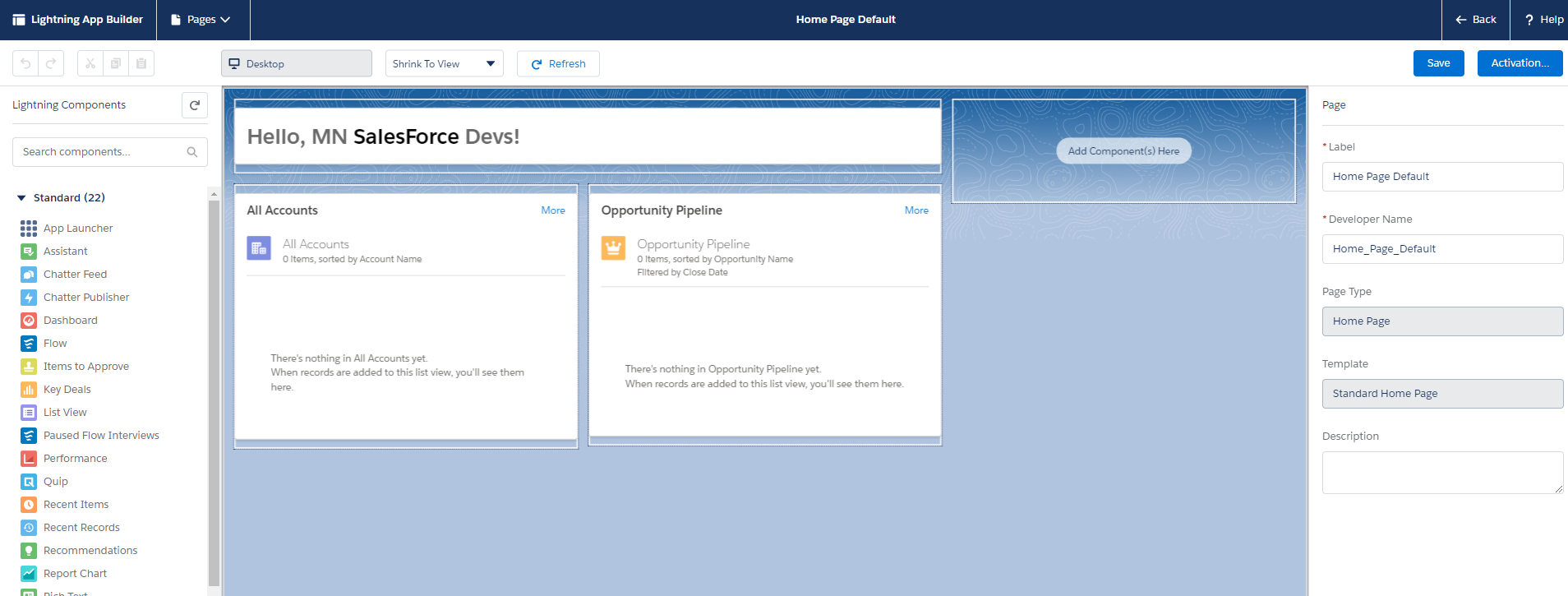
1. Open the scratch org. The below command should open the Salesforce Setup screen (in Lightning Experience) in your default browser.

sfdx force:org:open -u Issue1

1. Click the application switcher (shown below) and switch to the Sales application.



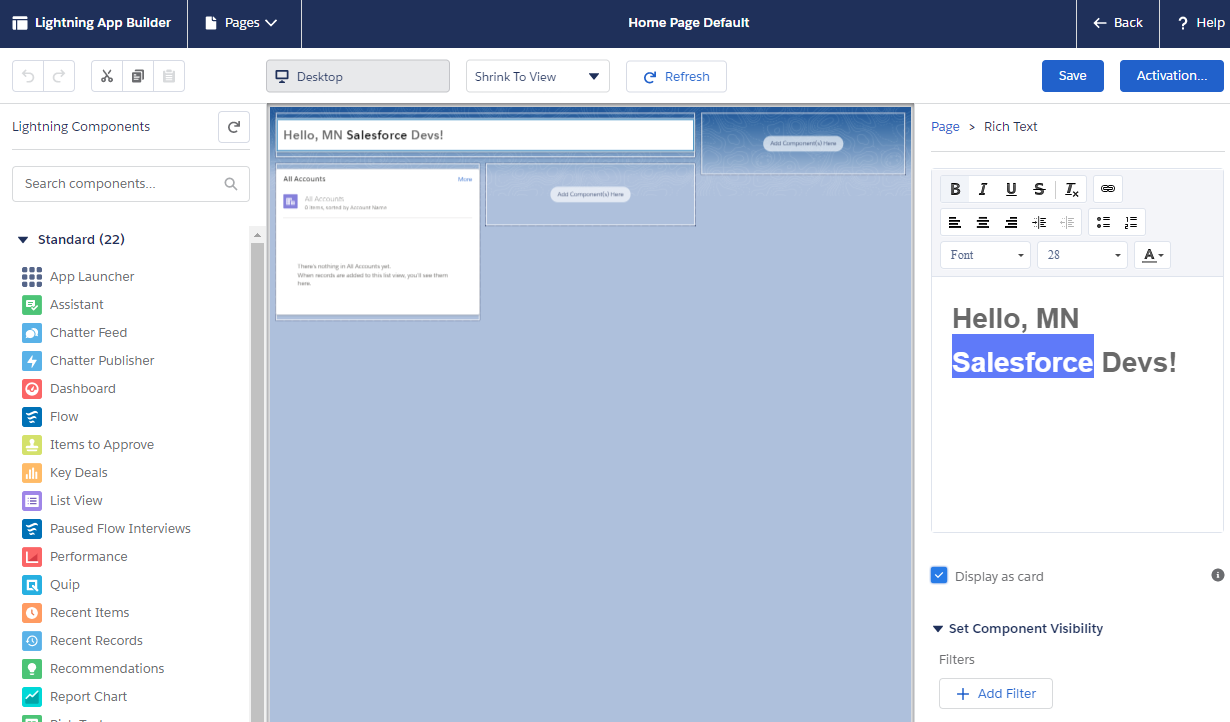
1. Now you should be on the Sales Home page. Click on the widget icon in the top right corner (shown below) and click “Edit Page”.  
   E
2. If you don’t already see a component called “Hello, MN SalesForce Devs”, click the “Page” button on the top bar and select “Home Default”.



1. All true Salesforce fans know that you can’t get away with incorrectly spelling or capitalizing Salesforce. Let’s go ahead change the component to fix the capitalization error, and change SalesForce to Salesforce.

Click on the component. Edit the component’s value on the right-hand side of the Lightning App Builder to correct the capitalization.

1. When you’re ready, click the save button at the top.

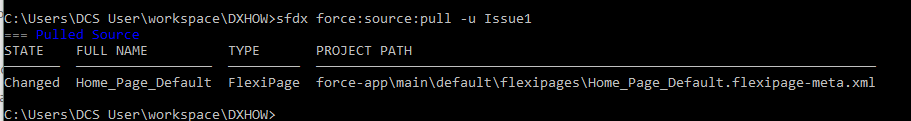


1. If asked, then choose to activate the page and assign it as the org default.
2. Now that you’ve made your change, it’s time to check your change into source control. However, first you’ll need to make sure your local repository is up to date. First, let’s change to the Issue1 branch. (Note: this branch was already created and should have come across when you forked the original repository at the beginning of the workshop). Issue the following git command:

git checkout –track origin/Issue1

1. Next, you’ll want to pull the code from your scratch org down to your local work station. Issue the following SFDX command:

sfdx force:source:pull -u Issue1



1. Now that the code has been downloaded / synched to your local workstation, it’s time to check that change in to source control! Issue the following commands below.

git add force-app\main\default\flexipages\Home\_Page\_Default.flexipage-meta.xml

git commit -m “Fixed a typo in the Home Page Component”

git push

You might say, “Huh??? What?? Is that English?”

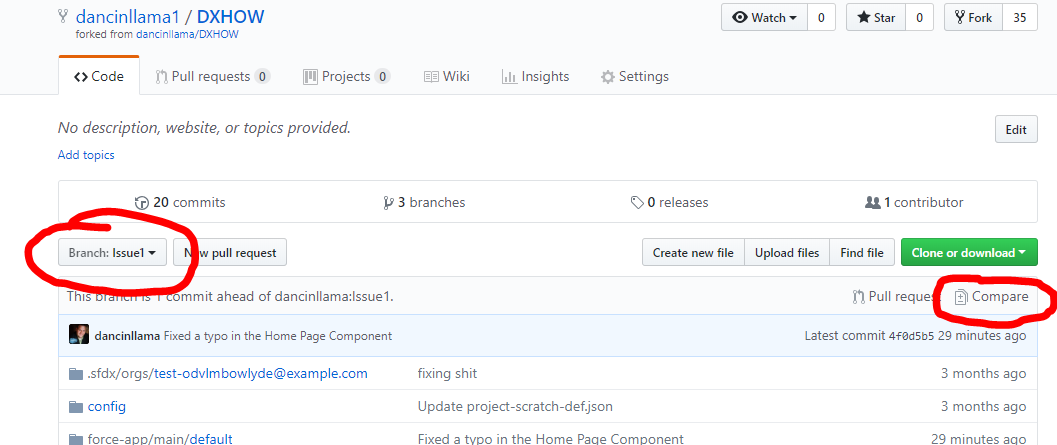
Recap

What you’re doing is the following: When you cloned the DX repository at the start of this workshop, you cloned a repository with three branches (Master, Issue1, and Issue2). The commands you just entered switched to the Issue1 branch. Next, you added the Home Page to a list of files to be committed (called an index). Last, you committed those files to the Issue1 branch, with a descriptive note.

1. The last step is to request that your Home Page changes are included in the deploy to production at the end of your current sprint. This is done through a git feature called a Pull Request.

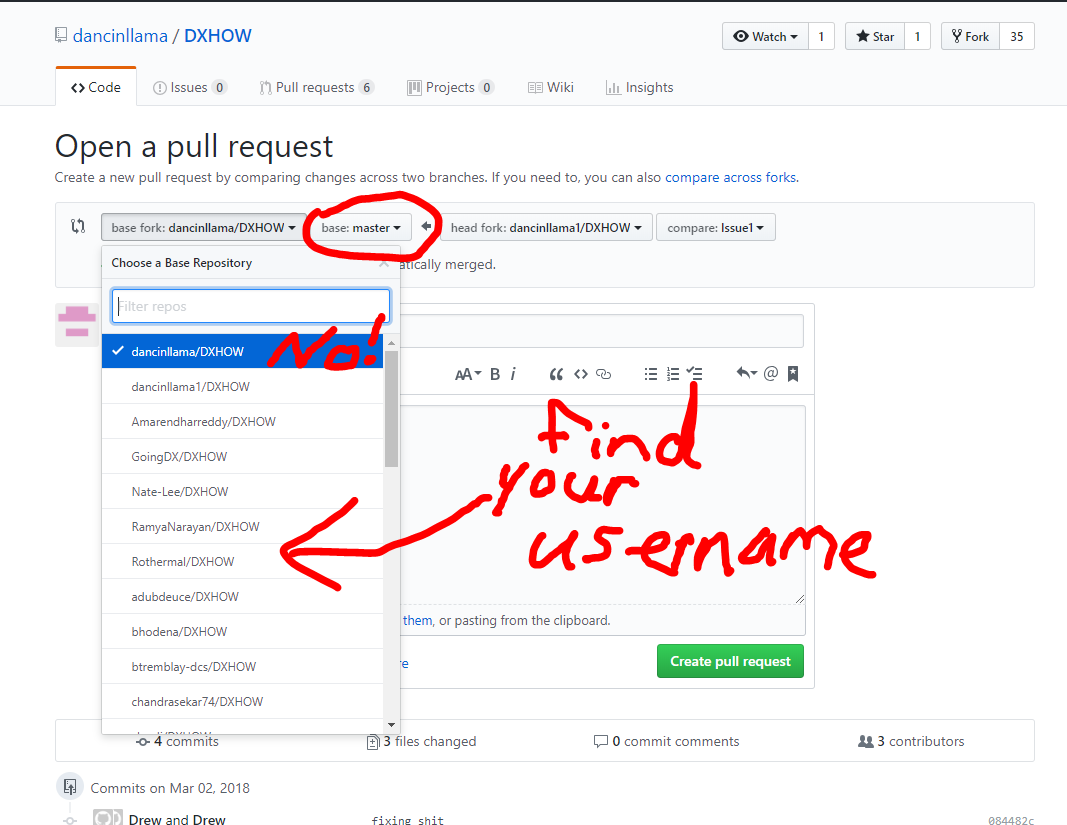
Go ahead and open your GitHub repository in the browser (e.g. https://github.com/<username>/DXHOW), if you don’t already have it open. You might have to log into GitHub again. Also, make sure you’re on the “Issue 1” branch (see circled image below).

1. Click the “Compare” button.



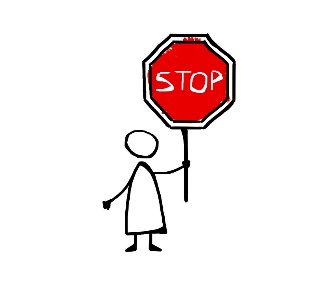
#### **WARNING!**

The Pull Request may default to “dancinllama/DXHOW”, because that is how you forked the repository. You must change this to YOUR repository! Also, make sure that “master” is selected as the “Base”. Switching to <your username> and the master base, will merge the changes from your “Issue1” branch into your “master” branch.



1. Click the “Create Pull Request” button to create the PR. (Your build lead will oversee reviewing the pull request)

Congratulations! You’ve identified an issue, created a scratch org specifically to tackle the issue, and then checked it into source control, and now it’s ready for a deploy. Let’s pause here, and then fix the 2nd issue, and the last item in our sprint before our big deployment.



# Step 4: Fixing the Second Issue

This step is very similar to the previous step. The point of this step is to show that you can modularize your work in separate scratch orgs. For instance, you could have two separate developers working on similar fixes and push those fixes to production at the same time.

If you’re struggling, and feeling a bit behind by this point, you can skip to step 5. However, if you’re feeling ambitious, you’re in the right place.

1. Create the second scratch org. In command prompt (Windows) or Terminal (Mac OS):

sfdx force:org:create -f ./config/project-scratch-def.json -a Issue2

1. Push the code you downloaded from GitHub to your second scratch org.

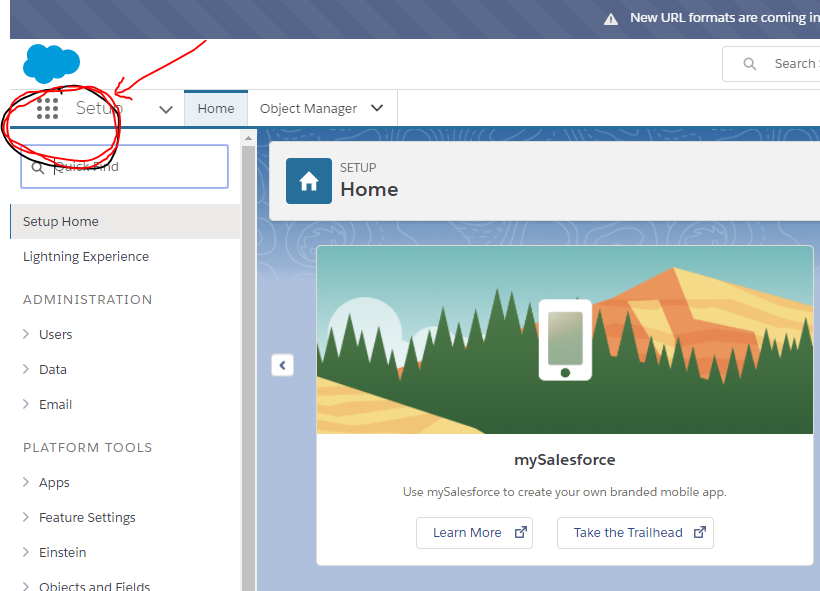
git checkout --track origin/Issue2

sfdx force:source:push -u Issue2

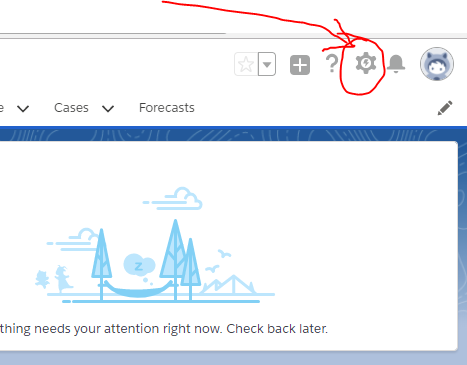
1. Open the second scratch org to make your change:

sfdx force:org:open -u Issue2

1. Click the application switcher (shown below) and switch to the Sales application.



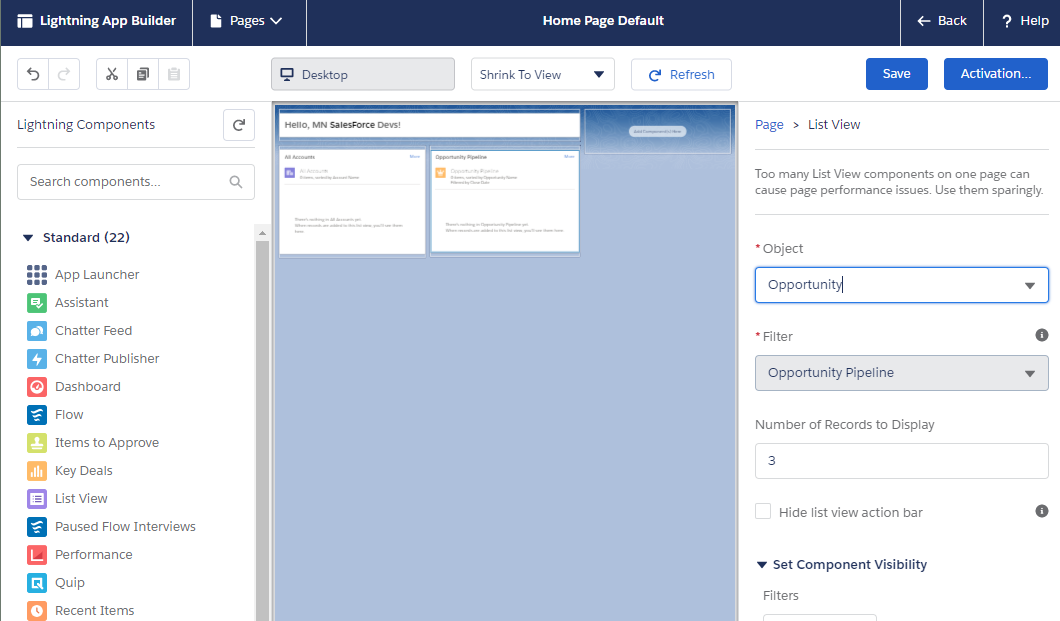
1. Now you should be on the Sales Home Page. Click on the Widget icon, in the top right corner and click “Edit Page”



1. If you don’t already see a component called “Hello, MN SalesForce Devs”, click the “Page” button on the top bar and select “Home Default”.

#### Note: Don’t worry about the SalesForce mis capitalization! Remember, our other developer (aka you a few moments ago), fixed this while working in a different scratch org.

1. Your stakeholder(s) would like to add another list view to this page. In addition to accounts, they would also like to see opportunities. Drag the list view component onto the home page and select the Opportunity list view.



1. Click the save button at the top. Activate the Home Page.
2. Now, back in your CLI / command prompt, pull the code back to your local workstation:

sfdx force:source:pull -u Issue2

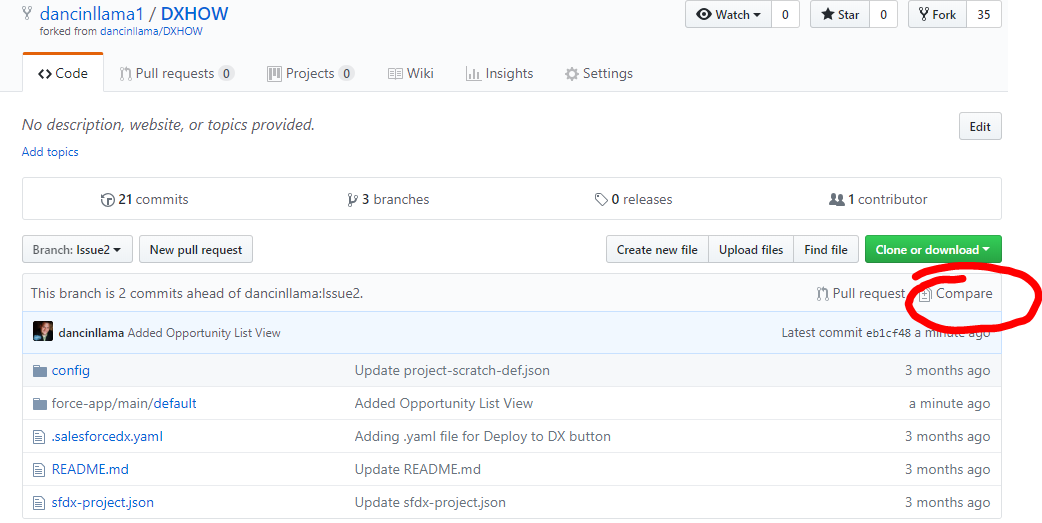
1. Now commit the files to the “Issue2” branch in GitHub:

git add force-app\main\default\flexipages\Home\_Page\_Default.flexipage-meta.xml

git commit -m “Added Opportunity List View”

git push

1. Almost done with the second issue! Time to issue a second pull request. Open your GitHub repository in the browser, if you don’t already have it open. You might have to log into GitHub again.
2. Click the “Compare” button.



#### **WARNING!**

The Pull Request may default to “dancinllama/DXHOW”, because that is how you forked the repository. You must change this to YOUR repository! Also, make sure that “master” is selected as the “Base”. Switching to <your username> and the master base, will merge the changes from your “Issue2” branch into your “master” branch. Check the screen shot in Issue1, for more information.

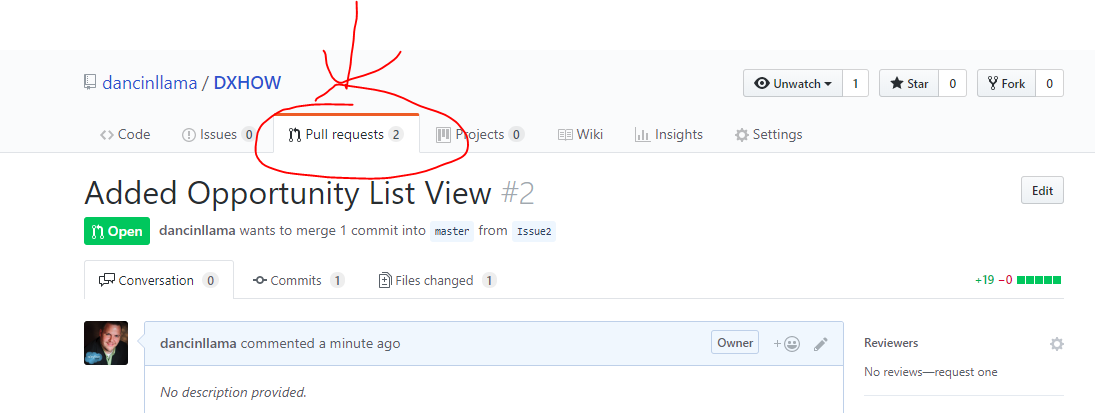
1. On the compare screen, click the “Create Pull Request” button to create the PR. Your build lead is now in charge of reviewing the pull requests.

Congratulations! You just fixed a second issue, with a second scratch org. Well done!

# Step 5: The Pull Requests

Well done, you’ve fixed one if not both issues, submitted pull requests, and are ready for the build lead to promote your fixes to production. Now, we’re going to switch roles from the lead developer to the build lead and promote those changes.

1. Open your GitHub repository. If it’s already open, make sure to refresh the page.
2. Near the top, you should see a “Pull Requests” tab with either a 1 or 2 indicating the pull requests that are waiting your review. Click on the Pull Request tab.



1. Click the Merge Button on the first pull request. (You can leave the defaults as is)
2. Click the Commit Button on the first pull request.
3. Refresh the page and you should see the available Pull Requests change.
4. Merge and Commit the second pull request.

#### **Warning!**

You may encounter a merge conflict when committing the 2nd Pull Request. This is because we are updating the same file across two different commits, and the 2nd Pull Request doesn’t have the change from the first pull request in place.

If this happens, you’ll see a “Resolve Conflicts” button appear instead of a “Merge pull request” button. This should open the GitHub web editor, where you can remove the “SalesForce Devs” line from the Issue2 branch. Once you remove the offending lines and resolve the conflict, you should be able to commit the pull request.

Congratulations, our master branch (which resembles our production org), is ready to push!

# Step 6: Push it. Push it Real Good!

We’re headed back to the command line to deploy your changes from source control to your production org / dev hub.

1. Open the command prompt (Windows) or Terminal (Mac OS).
2. Go to your project’s directory.
3. Switch to the master branch, where the production code resides, in GitHub:

git checkout master

1. Pull the latest code down to your local workstation:

git pull

1. Now we’re going to push the code up to the dev hub. Developer Hubs are different than scratch orgs, and unfortunately do not support the sfdx force:source:push command that we’ve been using throughout much of this workshop.

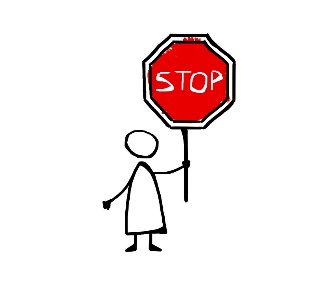
Instead, we’ll have to convert the DX project to the metadata API format, before we use the metadata API to deploy our project.

First, we must convert the source to the Metadata API format. The command below takes our DX project that is contained in the force-app directory (specified with the -r flag), and converts it to a metadata API structure, before storing it in the “deploy” directory, denoted with the -d flag.

sfdx force:source:convert -r force-app -d deploy

1. Now, go ahead and deploy the Metadata API structured files to your developer hub:

sfdx force:mdapi:deploy -d deploy -u DevHub



Congratulations! You’ve completed the workshop! By now, you should have a decent idea of how source control and Salesforce DX can work in tandem to provide a reliable, repeatable deployment process.

That’s right, you can use DX to help with repeatable deployment processes, in addition to other items like automating unit tests. In fact, DX works well with Continuous Integration (CI) or Continuous Delivery (CD) systems like Jenkins, Travis CI, Bitbucket pipelines, or Heroku Pipelines. If you’re anxious to continue learning about DX, check out the extra credit below, where you’ll explore running DX deployments through Bitbucket Pipelines.

# Step 7: Extra Credit: Create a Continuous Delivery System with Bitbucket Pipelines and Salesforce DX

For extra credit, we’re going to look at setting up automated builds with Bitbucket Pipelines and Salesforce DX. Automated builds allow teams to use source control commits to automatically kick off items such as automated tests, scratch org provisioning, or deploying code to production.

For this step, we will kick off an automated deploy to your dev hub as soon as items are pushed to a repository, this time, hosted in another source control cloud called Bitbucket.

## Create and Configure the Connected App

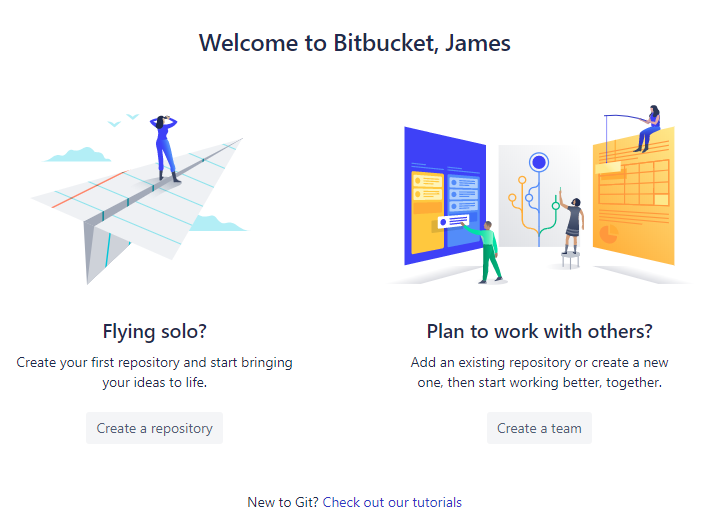
For Pipelines to authenticate with your dev hub, you’ll need to create a connected app and use the JWT auth feature of Salesforce DX. The JWT authentication is documented fairly well here: <https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_auth_jwt_flow.htm>

1. Create the certificate by following these instructions: <https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_auth_key_and_cert.htm>
2. Create and configure the connected app in your dev hub by following these instructions: <https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_auth_connected_app.htm>

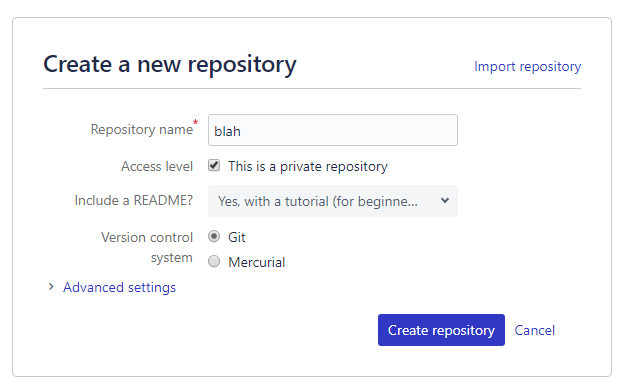
## Create a Bitbucket account and Set Up Pipelines

Bitbucket is a cloud-based git service, much like GitHub, which differs slightly in pricing and features. Bitbucket, unlike GitHub, provides a feature called Pipelines, which allows automated deployments and scripting whenever changes are committed and pushed to repositories.

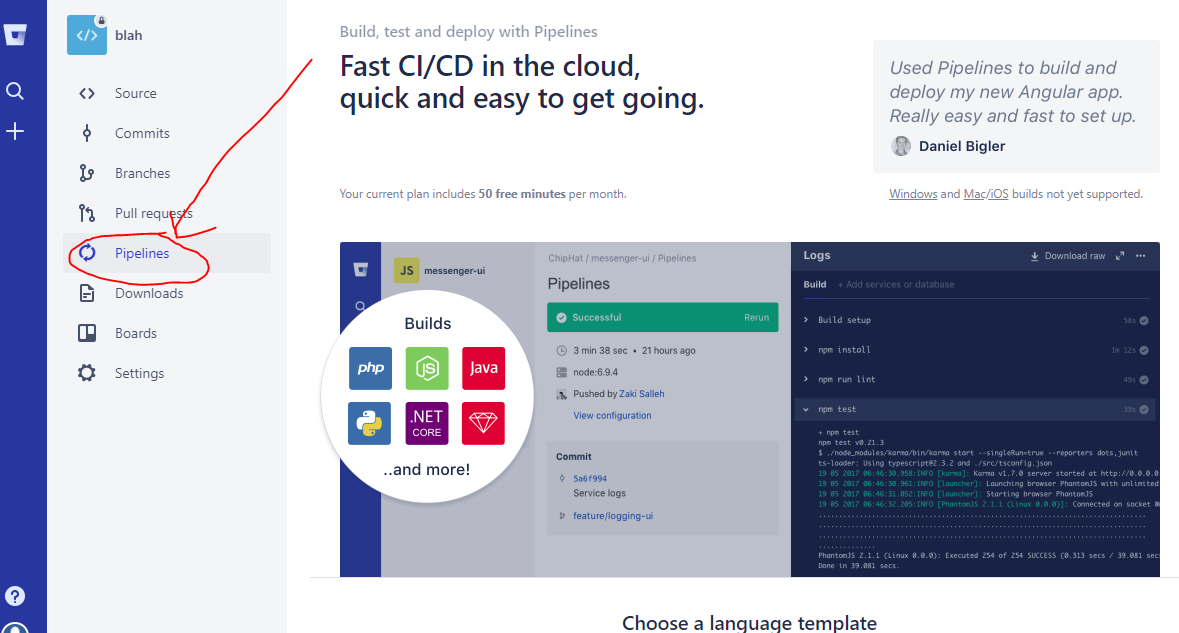
1. Go to <https://bitbucket.org/account/signup/> and sign up for a new bitbucket account (for free).
2. When you get to the welcome screen (shown below), clock the “Create a repository” button under “Flying solo?”



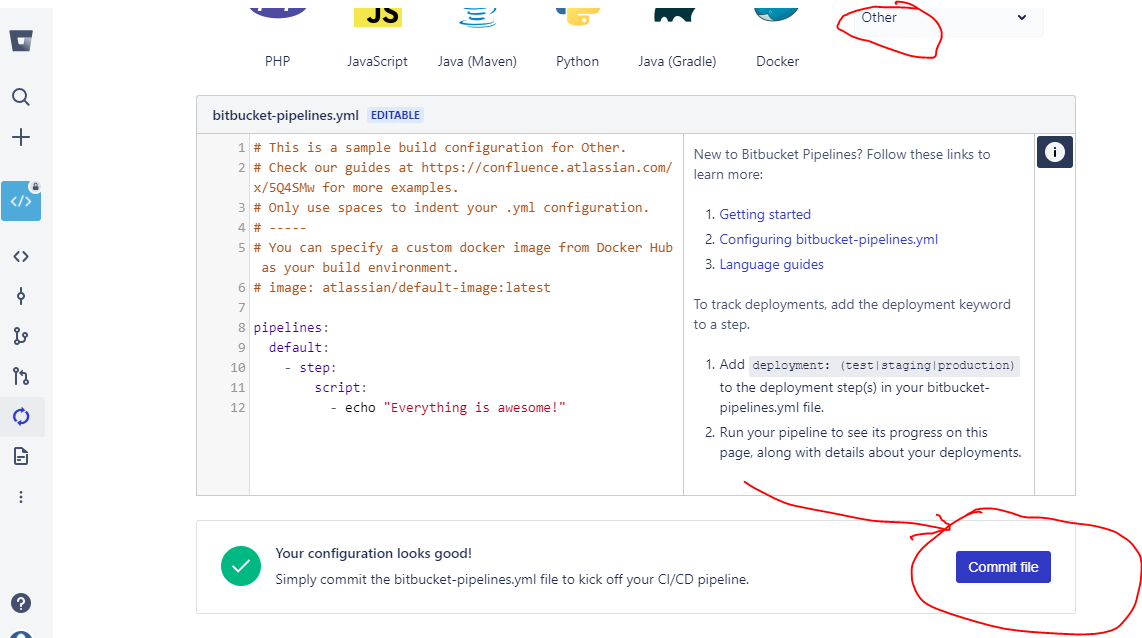
1. Specify a name for your repository, leave the rest of the defaults (make sure Git is selected), and click “Create repository”:



1. Next, select the “Pipelines” option from the bar on the left.



1. Scroll down to the “Choose a language template” option and select “Other” from the drop down.
2. You should see a message saying, “Your configuration looks good!”. Click the “Commit File” button.



If you see a blue screen, indicating that your pipeline is running or has finished running, then you’re on the right track!

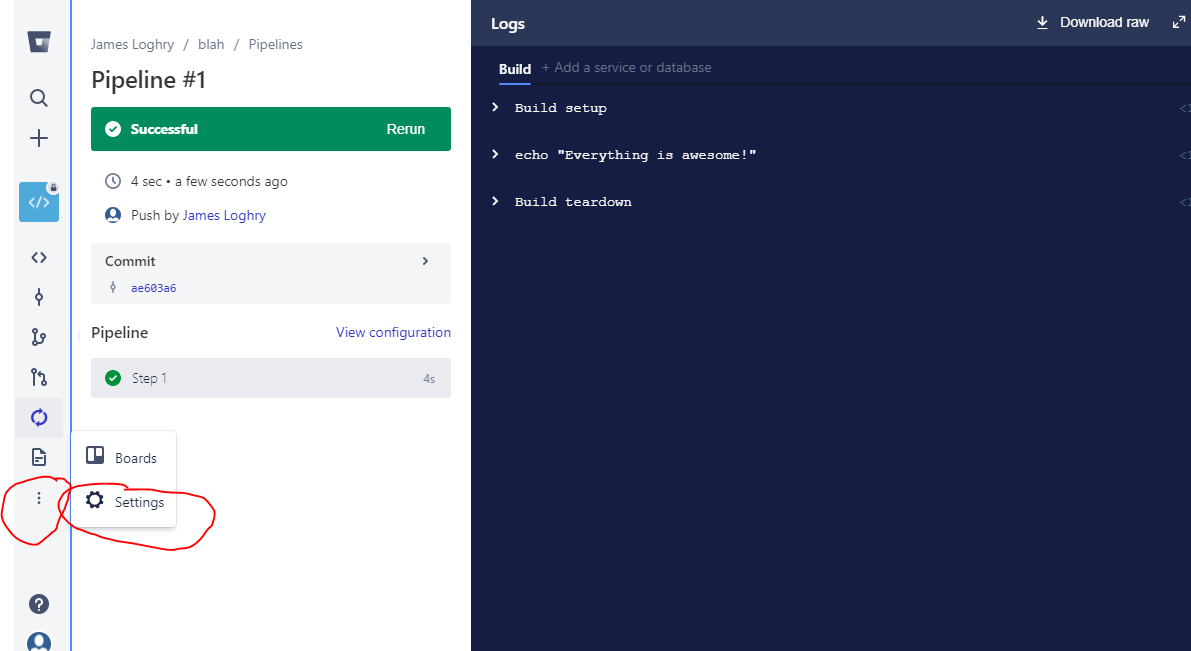
1. For the next few steps, we’ll set up three environment variables that Pipelines will use to for SFDX related pushes. These steps are based on a tutorial Kevin O’Hara wrote here: <https://github.com/kevinohara80/salesforcedx-circleci-demo>.

For the first environment variable, you’ll write the server certificate to a hex string, to be picked up by pipelines. To do this, you’ll utilize a hex dump utility called xxd. Open your command prompt (Windows) or Terminal (Mac OS), and type xxd -version to see if you have it installed. If not, then find xxd and install it on your system.

From the directory where you created your server key (in step 1 above), issue the command:

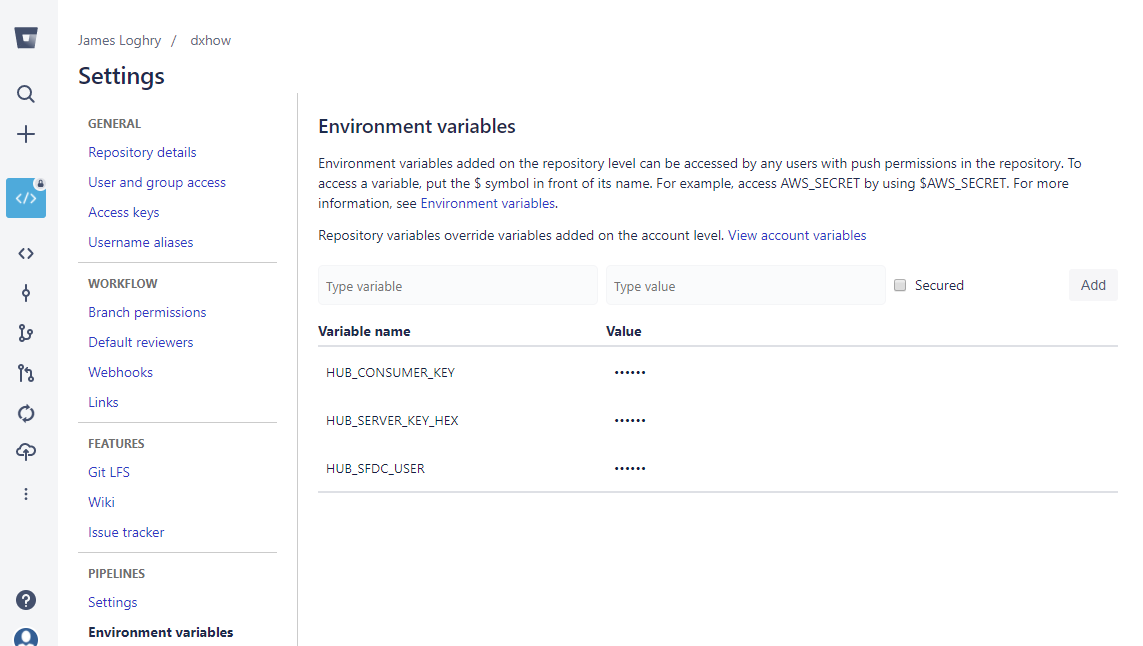
xxd -p server.key >> server.key.hex

1. Open the server.key.hex file and copy the contents to your clip board.
2. Back in bitbucket, click the ellipses (vertical …) icon, and click “Settings”



1. Scroll down and on the left side of the screen, click on “Environment Variables”
2. Add a variable with the name HUB\_SERVER\_KEY\_HEX copy the contents of the server.key.hex file into the value and save the environment variable.
3. Create a second environment variable called HUB\_SFDC\_USER with the username of your dev hub org.
4. Back in your dev org, locate the consumer key of your connected app, and copy the consumer key into a third environment variable called HUB\_CONSUMER\_KEY.

You should now have three environment variables that look like this:

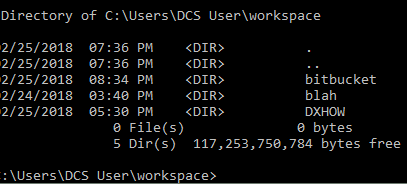


Congratulations! Your environment is set up for pipelines. Wait a second, where’s the code? We’ll get to that in a second, but first, let’s talk about the yml file. Bitbucket pipelines, Circle CI, TravisCI and other CD or CI systems typically use a yml file that determines what scripts are run when. In Bitbucket’s case, it uses a combination of a virtual image known as a docker image and a series of script statements to run commands as soon as a push happens in a repository.

Way back when you started this workshop, you cloned a repository from GitHub. That same repository contains a file called bitbucket-pipelines.yml. This yml file will tell bitbucket to use a docker image called dockerdx. The dockerdx image is an image that loads SalesforceDX into a virtual image for your building pleasure.

## Kicking off the Pipeline

Let’s go ahead and push the code to our new bitbucket repository. Find the root directory where you initially cloned the GitHub code from. The root directory will show the “DXHOW” directory like this:



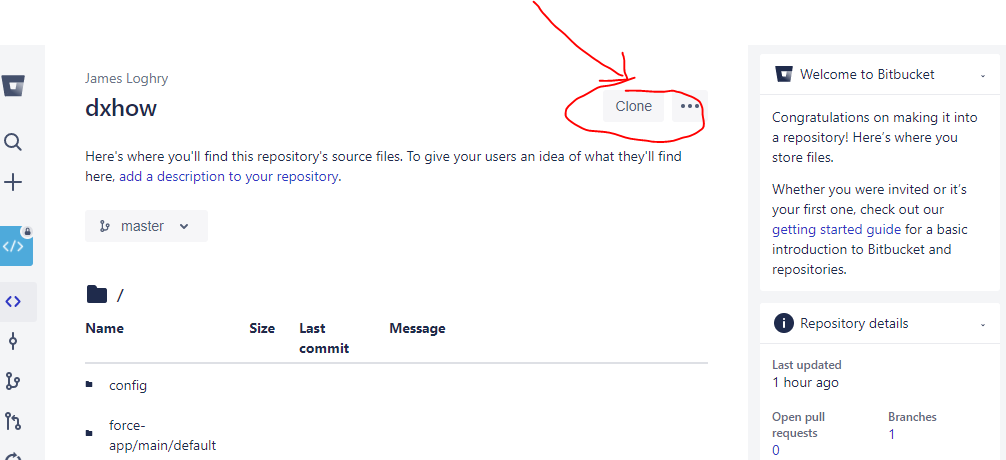
1. First, in the same root directory, create a sub directory called “bitbucket”

mkdir bitbucket

1. Change directories into the new bitbucket directory

cd bitbucket

1. In your browser, go to your repository page on <https://bitbucket.org>, and click the “Clone” button at the top right.



1. Copy the entire command (git clone <https://username@bitbucket.org>...), and run it in your command prompt (Windows) or Terminal (Mac OS)

##### **Warning!**

If you’re using this Bitbucket account for the first time, you may get a permission denied error. This is because you have not yet added an SSH key to your Bitbucket account. To generate and add an SSH key for Bitbucket, read the following documentation: <https://confluence.atlassian.com/bitbucket/set-up-an-ssh-key-728138079.html>

1. Next, copy the files you cloned from the GitHub repository into the new repository folder.

For example: cp -r ../DXHOW/\* ./<repository folder name>/

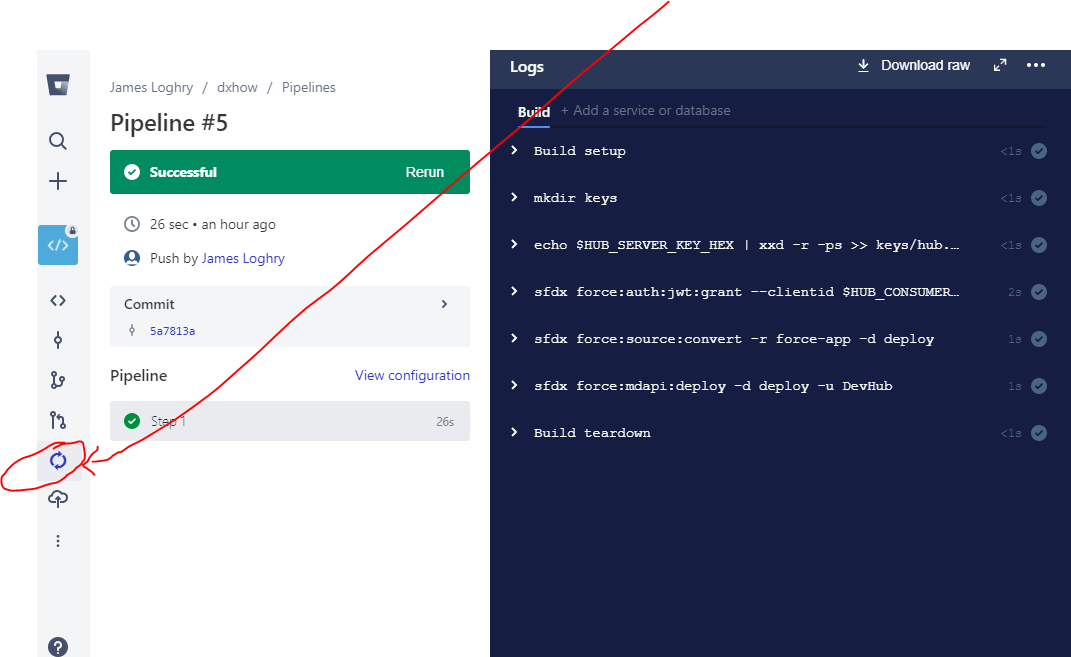
1. Change to the repository directory if needed.
2. Add all the files to a git index and push them to your master branch. This should kick off your pipeline. Run the following commands:

git add \*

git commit -m “Initial code commit. Hold on to your butts.”

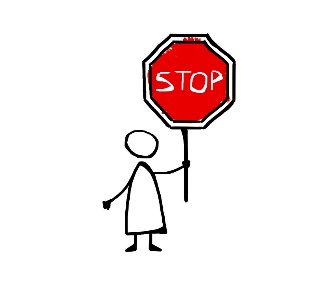
git push

1. Now, check your pipeline to see if was kicked off (and to see if it succeeded). From bitbucket.org, click the Pipelines icon.



If you see a green “successful” message, congratulations! You just kicked off a successful automated build using DX and Pipelines!

If you got an error though, don’t feel too bad. Try to see if you can fix the issue and recommit your changes.



# Additional Tools and Resources

With the advent of Salesforce DX, there are several tools that are popping up to make Salesforce Developer’s lives much easier, in terms of deployments. This section will go over some of those tools, as well as Trailhead modules and documentation you’ll want to check out around Salesforce DX.

## Salesforce DX Documentation

The one stop shop for all DX related documentation: <https://developer.salesforce.com/platform/dx>

Creating a scratch org: <https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_create.htm>

Scratch org shapes (how to define the editions and features when creating scratch orgs): <https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_def_file.htm>

## Trailhead Modules

Check out the following Trailmix for Salesforce DX related modules: <https://trailhead.salesforce.com/en/users/0055000000612i1AAA/trailmixes/salesforce-dx>

## Salesforce DX Related Tools

Deploy to DX – A GitHub button for quickly creating a scratch org from a GitHub repository: <https://deploy-to-sfdx.com/>

Visual Studio DX plugin (This is actually a review of the plugin, but goes through how to install and use the plugin): <https://bluecanvas.io/2017/09/13/salesforce-dx-visual-studio-code.html>

IntelliJ / Illuminated Cloud – IntelliJ is a paid IDE, and Illuminated Cloud is a Salesforce add on (similar to Mavens Mate and Sublime Text). Disclaimer: I haven’t used this but have heard nothing but good things. <http://www.illuminatedcloud.com/home/salesforcedx>

SFDX Falcon Template – A set of templates or org shapes, hosted on GitHub, geared toward ISVs / managed package developers: <https://github.com/sfdx-isv/sfdx-falcon-template>

## Salesforce DX Around the Community

Be sure to check out the following community related resources, to continue learning more about Salesforce DX:

Follow the MN Developer Group on the Success Community for upcoming events: <https://success.salesforce.com/_ui/core/chatter/groups/GroupProfilePage?g=0F9300000001p2f>

Follow the DX group on the Success Community for discussions regarding Salesforce DX: <https://success.salesforce.com/_ui/core/chatter/groups/GroupProfilePage?g=0F93A000000HTp1>