Welcome to a demo of how to implement version control of your R projects with Microsoft Azure DevOps, via interaction with RStudio.

Before you can implement version control in the particular manner that we will go over today, you will need R, RStudio, and Git installed on your local computer. You can reference our RClub’s resources for more information on that, as well as more detailed instructions for version control with DevOps and RStudio.

I already have the software installed so the next step is to set up and log in to my work-based DevOps account.

You can do a quick web search for Microsoft Azure DevOps and navigate to the page you see here.

Then click on the “Start free” button to set up your account. You’ll use your existing Microsoft Office account information for set up.

Since I already have an account, I’ll switch to that profile now, which I actually have bookmarked and am already logged in.

After logging in, I’m taken to my account’s home screen, where you can see multiple projects that I have set up. You can click on these tiles to access the projects.

We’ll start the demo process, though, by creating a new project or repository.

Keep in mind that when starting a new project that will be version controlled, it is easiest to start the process from DevOps (as opposed to starting in R and then trying to implement version control afterwards).

To start a project in DevOps, we’ll click on the “+New Project” button in the top right corner.

Provide a name and short description. Note that names can NOT have spaces.

We’ll call this one “RClub-Test” and put a short description here. Then we can choose whether it’s Public or Private. Until we get the OK from our OIT department, we’ll keep this one Private. But know that you can still share it with individuals even if it is Private.

Then we click the “Create” button in the bottom right corner.

When your project is created, you’ll automatically be directed to its empty page.

We’ll now start a repository, or master folder to house our test project. To start a repository, click on either the Repos tab on the left side of the screen, or the Repos button in the middle of the empty page.

You’ll notice other features that are great for collaborative work, but today we’ll focus on repositories only.

So we’ll click Repos and you’ll be directed to this page, asking to add some code.

Since we’re starting a brand new project, we want to Clone to our computer, the first option here. You can also Import a repository from GitHub, but today we are starting a brand new one.

So we’re going to copy this link and then move to RStudio.

So let’s open RStudio and start a new Project. I’m going to do this via File > New Project > Version Control.

In this pop-up window I want to select “Version Control”.

In the next pop-up I want to select “Git”, since that is the program I would like to use.

This prompts me to fill in the Repository URL. This is what we copied from the DevOps project, so we paste that here.

Notice it automatically populates the Project directory name.

Then you can choose what directory (on your computer) you want to work in. It will automatically create a folder with the project name so you do not need to create a folder first.

I’m going to place mine on my desktop.

And now I click “Create Project”

You’ll get a popup to select your Microsoft Office account. You might get a warning that says your logon failed and/or that you cloned an empty repository – both are OK as long as you are still signed into your DevOps account

When your project opens in RStudio, you’ll know it’s associated with version control when you can see the Git icon and Git tab.

Now let’s make our first commit, or change, to this repository. Let’s start a new script and just populate it with some text. Then we’ll save it.

You should now see this saved script in the Files pane and in the Git tab of the Environment pane – you may have to hit the “refresh” button in the Git tab in order to see the new script show up here. And this can take a little while so be patient, especially when scripts or files are large.

To make a commit, select the files from the Git tab that you want to commit. Check the box next to each file you want to make a commit or change for.

When they have a check mark and their status is now changed to a green with an “A” or alternatively a blue “M” for subsequent commits, click the Commit button in the Git tab.

Now you’ll get a pop up where you can enter your short commit message that let’s you know in general what has changed.

Click commit. You should see a command prompt window open and eventually you will see the results of your commit.

Click close when it’s done.

Now we need to push that commit to DevOps. Click the Push button and you should get another command prompt. You might also get a Microsoft office account prompt window, and even a warning that says the logon failed.

In my experience, it still works as long as a second message appears afterward with the project URL.

You can confirm that your commit and push were successful by navigating back to DevOps and refreshing the page.

Now instead of a URL page for the repository, you should see the files that have been added.

Every time you want to commit and push changes, you follow the same steps in RStudio from selecting the files you want to update and then committing and pushing.

So that’s how you create a repository, commit, and push.

We’ll follow a similar workflow for when we make changes within our project. We will go back into RStudio and make a change.

Let’s make a change to our script.

Save it. Now we’ll see that the script appears in our Git tab as a modified file. This means the local version is different than the online version of the project. We can repeat the commit and push process to update our DevOps project file and get everything in sync.

Once the commit and push are complete from RStudio, we’ll go back to DevOps.

In DevOps, we’ll refresh the page and you should be able to see the new commit. Let’s explore some more DevOps features now that you have multiple commits….

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I have navigated back to my DevOps account’s home page by clicking on my name.

Here you can see our new repository as a project tile. Now we can explore the project since we have populated it with files.

So I’ll click on the tile and I’m taken to the project’s home page. Here you’ll see the description we gave our project, the coding language present, as well as some Project Stats. By default, it shows the last 7 days, but you can change that to 1, 7 or 30 days.

You’ll also notice the members and here it is just one since I haven’t added anyone else. If you want others to have access to your project and collaborate, you can click on the “Invite” button at the top right. Here you will be prompted for email addresses and they’ll be sent an email saying they’ve been invited to this “team” or “project”.

We might also want to look at the history or different versions of our project. We can do that through the “Repos” link on the left side of the page.

When hovering over “Repos”, we get an additional menu. If we click on “Files” we can view the different files in our repository. We can also look at the history of our commits by either clicking on the History tab here, or by the “Commits” link on the left side of the page.

Either of these options will give you the commit history. If you want to see the exact changes that were made, you can click on any of the commits.

Here I’ll click on our “Second commit” and you can see the line by line changes to our script. They are highlighted in either green (for additions) or red (for deletions).

By highlighting, you can see exactly what was added or deleted.

You can do this for any commit throughout the project.

So those are the basics for creating a repository, committing, and pushing changes, and some introduction to navigating DevOps.

Hopefully this will get you started on your version control journey and you can explore and test more on your own. I only covered the basics for more solo-programming, but online version control is a great resource for collaborative work too.

Thanks for watching!