# Lab 1: Working with Repositories

Goal: Learn to work with Git and Repositories.

**Exercise 0: Create a default repository for you.**

Login to your Azure DevOps environment. Go to Project.

Create a new repository, choose a name that is unique for you. This repository will be used throughout the days.

**Exercise 1:**

Step 1: Download and install git (skip if already done)

If you have not yet installed git on your machine, download and configure git.

[Git - Installing Git (git-scm.com)](https://www.git-scm.com/book/en/v2/Getting-Started-Installing-Git)

Type “git help -a” in your terminal and press enter. You will be presented with a list of available git commands.

**Exercise 2: Setup authentication to Azure DevOps Server & clone an existing repository using SSH.**

In this step we are going to create an SSH key which you will use to authenticate to Azure DevOps and download the existing repository, which has been created in exercise 1.

For this we are going to use the steps mentioned in.

[Use SSH key authentication - Azure Repos | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/repos/git/use-ssh-keys-to-authenticate?view=azure-devops)

Keep in mind, you can clone the repository to a directory of choice on your machine.

**Exercise 3: Your first commit**

In this exercise, you will learn how to change code in a Git repository.

When you make changes to your files, Git will record the changes in the local repository. You can select the changes that you want to commit by staging the changes. Commits are always made against your local Git repository, so you don’t have to worry about the commit being perfect or ready to share with others.

You can make more commits as you continue to work and push the changes to the server when they are ready to be shared.

What’s in a commit?

Git commits consist of the following:

* The file(s) changed in the commit. Git keeps the contents of all file changes in your repo in the commits. This keeps it fast and allows intelligent merging.
* A reference to the parent commit(s). Git manages your code history using these references.
* A message describing the commit. You give this message to Git when you create the commit. It’s a good idea to keep this message descriptive, but to the point.

Commit change to existing file

Open your editor of choice and open the README.md file, that has been created upon creating the repository.

Change the text to whatever you like. Then save your changes locally.

Verify your changes

Go back to your terminal and navigate to the root directory of the repository.

Enter <git status> and press enter.

Notice that your terminal will show a message like the following.

A screen shot of a computer

Description automatically generated

A couple of things are important to understand from the output.

1. The branch you are currently working on (Top)
2. The modified files (middle part)
3. Instructions on how to add changes to a commit (bottom)

Add your changes to your commit

To commit your local changes to the repository of the server. We need to create a commit.

As instructed type <git add> and press enter.

Read the feedback and try and find a way to add your local changes to a commit.

Once you’ve been successful, enter <git status> again and verify your changes.

A screen shot of a computer code

Description automatically generated

Create the commit and push to server

Now that changes have been staged, you will create a commit.

Type <git commit -m “Your message”> but replace “Your message” with your own commit message.

Now that the commit has been created, push the changes to server by performing.

<git push>.

Now head over to the commits section of Azure DevOps and observe your commit.

A screenshot of a computer

Description automatically generated

**Exercise 4: Protect your repository.**

Now we’ve successfully made our first commit to the main branch. We realize that pushing directly to the main branch can potentially be dangerous.

Pipelines are often configured to deploy to production environments upon changes to the main branch. Therefor changes made to the main branch can potentially trigger a deployment to production. A way of preventing mistakes is by protecting your main branch.

Protect your main branch according to the following requirements:

* All changes to the main branch require a reviewer

Use the following documentation to find instructions.

[Branch policies - Azure Repos | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/repos/git/branch-policies-overview?view=azure-devops)

Once complete, show your configuration to the coach before you head to the next Exercise.

Before moving on to exercise 5, feel free to remove branch protection for now. In this case it is allowed because we are not working on production applications.

**Exercise 5: Working with branches**

After learning how to protect our branch, it is time to start working with branches.

For this exercise we are going to create a new branch on which we are going to create a PowerShell “hello world” script.

Go back to your command line, and from the directory of the repository create a new branch.

You can do so by running the command: <git checkout -b hello-world>.

Create a PowerShell script which writes “Hello world” to console.

Now proceed to perform the following actions, use the documentation listed in the urls below if you are stuck:

* Commit and push your changes to the server on your new ‘hello-world’ branch.
* Create a pull request to merge to main
* Approve the pull request
* Merge pull request
* Bring your local machine to the latest server status by running <git pull>

Make sure to keep the change history of the main branch condense.

When pushing a new local branch to the server you will receive an error message, try to see if you can fix it.

[Create a pull request to review and merge code - Azure Repos | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/repos/git/pull-requests?view=azure-devops&tabs=browser)

[Merge strategies and squash merge - Azure Repos | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/repos/git/merging-with-squash?view=azure-devops)

**Exercise 6: Enable auto setup of remote branches**

As you’ve noticed, you need to create an upstream origin for your new branch.

Git supports functionality that automatically does this for you in the future. In this exercise we will set that up.

Use <git help config> to find instructions this will open a web page in your browser with documentation. Search the page for:

push.autoSetupRemote.

Once you’ve finished, find a way to list the config of autoSetupRemote. And validate that the value is “true”.

[Git - git-config Documentation (git-scm.com)](https://git-scm.com/docs/git-config)

**Exercise 7: Working with hot-fixes**

We will now simulate a situation where you are working on a feature in the sprint and without completing the feature, are being confronted with a production incident which needs your immediate attention.

During sprint planning you’ve been given the task to modify the README file, in the Getting started section, describe what is required to run the PowerShell script created in Exercise 5.

Create a new branch, named: getting-started

* Change README.MD
* Add and commit the changes. **Do not push**

Now you have not yet finished your work and have not pushed to the server yet. Now a production incident occurs. The Hello-World script in production is causing problems. We need to change it ASAP. Remember production code is part of the main branch.

Solve the problem by performing the following actions.

* Checkout main and pull the latest version <git pull>
* Create a new branch, called hot-fix-hello-world
* Update the PowerShell script and add a new line at the bottom with <exit>
* Commit, push and create a pull request
* Approve the pull request and merge to main

Notice that you didn’t have to set the upstream origin thanks to configuration of autoSetupRemote in exercise 6.

You have now created a new branch and fixed the problem. Now you can go continue your sprint work, which was left unfinished on the <getting-started> branch.

* Checkout <getting-started>
* Make a few changes
* When finished create a commit, push, pull request and
* Merge to main.

At the end of this exercise you have merged your hot-fix changes to main. The only branch in your repository is the main branch.

**Exercise 8: Restore a branch**

Azure DevOps allows for the removal of branches. In this exercise we are going to experiment with this.

Actions to perform.

* Create a new branch from the Azure DevOps web interface
* Delete the branch in the web interface
* Restore the branch in the web interface

To be able to perform the actions, reference the documentation below.

[Create a new Git branch from the web - Azure Repos | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/repos/git/create-branch?view=azure-devops&tabs=browser)

[Delete a Git branch - Azure Repos | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/repos/git/delete-git-branch?view=azure-devops&tabs=Browser)

[Restore a deleted branch in your Git repo - Azure Repos | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/repos/git/restore-deleted-branch?view=azure-devops)

**Exercise 9: Clean up local branches**

By now we have merged multiple branches to our main branch. It’s a good practice to delete local branches which have already been merged.

Find a way to:

* List all local branches
* Delete the already merged branches from your machine

**Exercise 10: Solve merge conflicts**

Merge conflicts occur when 2 branches are merging that contain conflicting content.

We are now going to create a merge conflict on purpose, and then resolve it.

For this exercise create 2 new branches based on main.

* Branch-A
* Branch-B

In Branch A modify the text in README.md under # Build and Test.

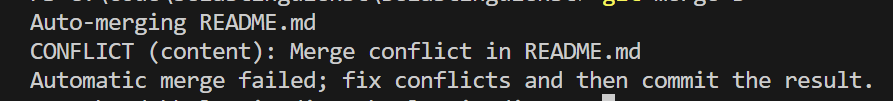
Now perform the following steps:

* Checkout A
* Modify the text
* Commit your changes

Then switch to branch B, you will see your changes to the file will disappear.

In Branch-B modify the same text, to a different value. Use the same actions as the ones for branch-A, to modify the text and commit your changes.

Then proceed to merge branch B with branch A. Use the command <git merge A>.



You will receive the above message. Now type <git status>.

This will provide you a bit more information.

A screen shot of a computer

Description automatically generated

Now go ahead and fix your merge conflict. Hint: Using an editor such as VS Code can make resolving merge conflicts more easy.

Choose which version you would like to keep by clicking one of the options within the highlighted area in the below snippet.

A screenshot of a computer program

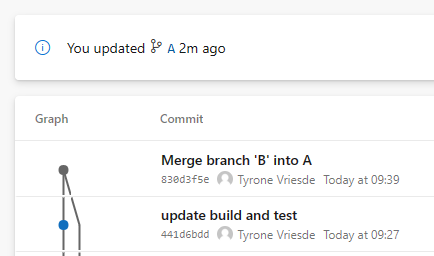
Description automatically generated

Figure : VS Code Merge conflict

Now perform the following steps:

* Now save the file.
* Use <git add <file>> in this case <git add README.md> to add the unmerged path to a commit.
* Now go ahead and commit the merge. <git commit>.
* You will now have the option to edit comments related to the merge.
* Push changes to the server

In Azure DevOps, head over to Repos>Files>History. You will notice that the graph shows the merge action.



Create a pull request and merge your changes to main and remove local branches A and B.

**Exercise 11: Link a work item to your PR**

For this exercise you will link a work item to your pull request. Linking work items to pull requests is a good practice to improve traceability across your DevOps project.

Perform the following steps.

1. Create a new work item within your project, you can decide on the content
2. Create a pull request
3. Link the pull request to the work item in the pull request overview screen

Observe the linked work item like in the screenshot below.

A screenshot of a computer

Description automatically generated

Linking of work items can be configured to be mandatory for pull requests by using branch policies.

Keep the pull request open, we will use it in the next exercise.

**Exercise 12: Managing pull requests**

In the overview tab of the pull request you can see all the key information related to a pull request.

Click on the files tab to review the changes.

A screenshot of a computer

Description automatically generated

If you don’t see the code side by side, click on the “Inline” button.

Now use the comment functionality to add a comment.

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated

After finishing your comment, go back to the overview screen. Notice the comment to be shown there. Team members can respond to these comments.

Note, it is possible to enforce the resolution of comments before completing pull requests.

Notice the overview on how many comments are yet to be resolved.

A close up of a text

Description automatically generated

From the PR overview screen, go ahead and resolve the comment, after which you merge to main.

Clean up the local branch as well.