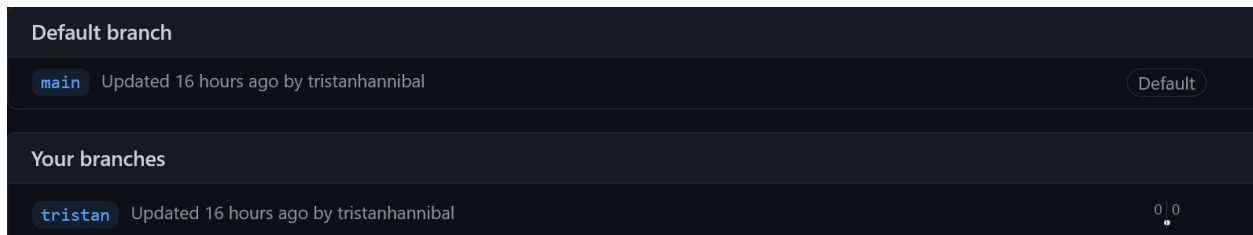


Group B

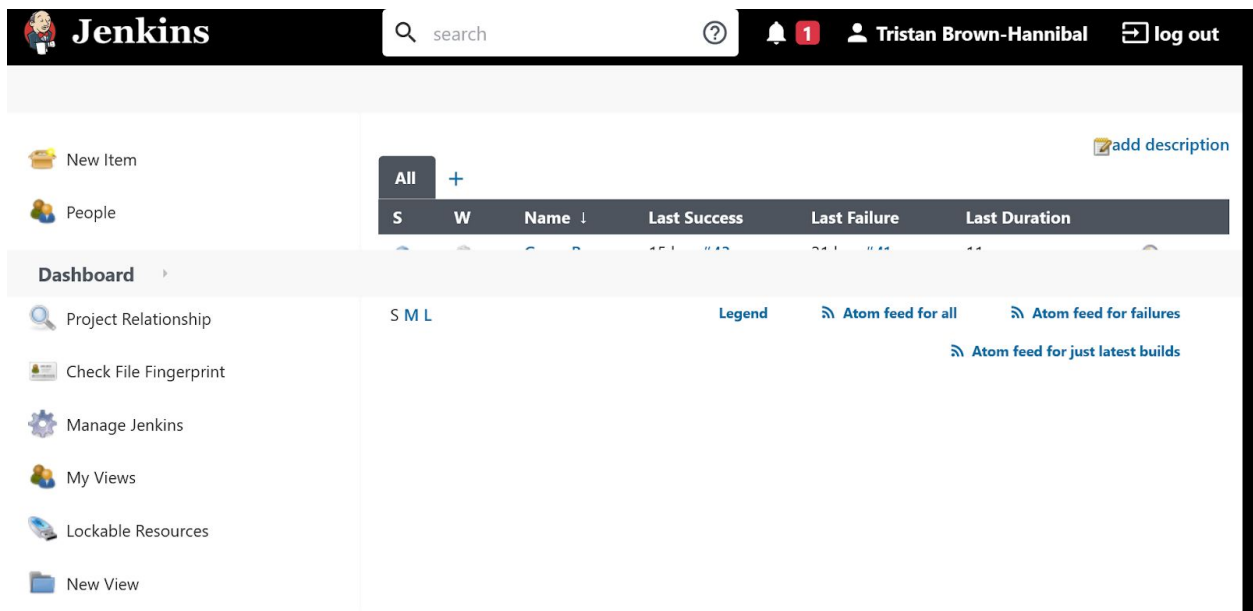
How we set up Jenkins.

Firstly, we create a new GitHub branch in which we will push our untested code.



This branch is where the individual's work is placed, before being merged back with the main.

Then using docker we will run Jenkins, create an account and login.



From here, we created a new item, a Freestyle project.

Enter an item name

Group B

» Required field



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Then we configured it as such:
We link the project to our GitHub

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Description

[Plain text] [Preview](#)

☐ Discard old builds ?

☒ GitHub project ?

Project url

[Advanced...](#)

And Indicate we want to use Git, while providing valid credentials for the project to use.

Source Code Management

☐ None
☒ Git

Repositories

Repository URL

Credentials

Next, the individual chooses which branch they want to build and test (The one where they uploaded their own work)

Branches to build

Branch Specifier (blank for 'any')

Additional behaviour is required to merge, we indicate the process we want

Additional Behaviours

Merge before build

Name of repository

Branch to merge to

Merge strategy

Fast-forward mode

We tried to get webhooks working, but since we are on localhost, we couldn't figure out how to set up localhost and GitHub webhooks properly. But if this was a real server we could set up webhooks to allow an automatic build and test after committing to a GitHub repo

Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☒ GitHub hook trigger for GITScm polling
- ☐ Poll SCM



Then we have to configure the tools Jenkins uses to build and test our project. Since none of us knew how to use Maven, and we used Gradle in the lab, we used Gradle as the build tool.

We get Jenkins to download the version of Gradle it needs, and used it to run our build.gradle file

A screenshot of the Jenkins configuration page for the 'Invoke Gradle script' build step. The 'Invoke Gradle' radio button is selected. The 'Gradle Version' dropdown menu is set to 'GRADLE_HOME'. The 'Tasks' text input field contains 'clean test'. The 'Build File' text input field contains 'ENSE375/RiskMeter/build.gradle'. There are help icons (question marks) for each of these fields. Below the 'Build File' field, there is a note: 'Specify Gradle build file to run. Also, some environment variables are available to the build script'.

Our build.gradle file is:

```
plugins {
    id 'java'
}

repositories {
    mavenCentral()
}

sourceSets {
    main {
        java {
            srcDirs = ['src/main/java/com/uregina']
            outputDir = file('bin')
        }
    }
}
```

```

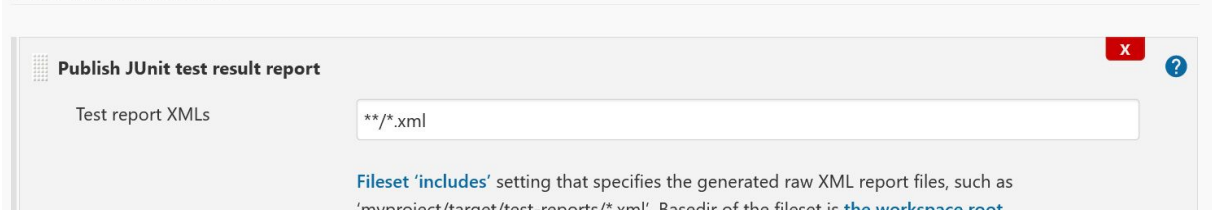
    }
    test {
        java {
            srcDirs = ['src/test/java/com/uregina/app']
            outputDir = file('bin')
        }
    }
}
dependencies {
    testCompile("org.junit.jupiter:junit-jupiter-api:5.2.0")
    testRuntime("org.junit.jupiter:junit-jupiter-engine:5.2.0")
}
dependencies {
    testCompile("junit:junit:4.12")
    testRuntime("org.junit.vintage:junit-vintage-engine:5.2.0")
}
test {
    useJUnitPlatform()
    testLogging {
        events "passed", "skipped", "failed"
    }
    reports {
        junitXml.enabled = true
    }
}
}

```

This file ensures output of XML, and uses either Junit5 or Junit4.

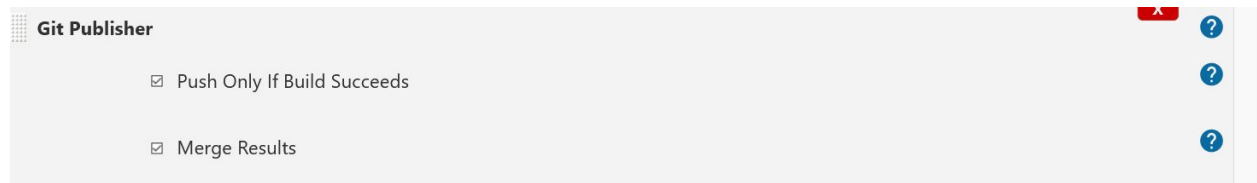
After this build and test, we make sure Jenkins uses the XML files and publishes the test results.

Post-build Actions

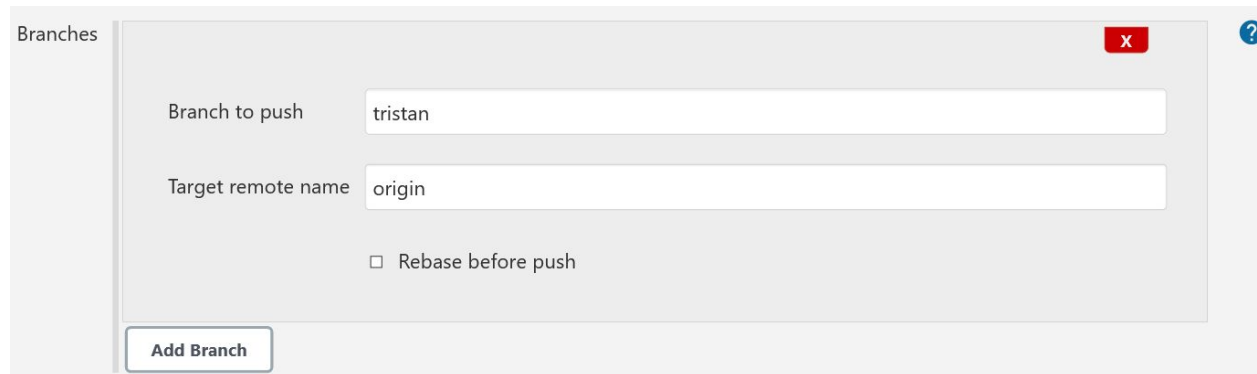


The screenshot shows the 'Post-build Actions' section in Jenkins. A card titled 'Publish JUnit test result report' is visible. It has a red 'X' icon and a blue question mark icon in the top right corner. Below the title, there is a field labeled 'Test report XMLs' with the value '**/*.xml'. Below this field, there is a blue link that says 'Fileset \'includes\' setting that specifies the generated raw XML report files, such as \'mynproject/target/test-reports/*.xml\' Basedir of the fileset is [the workspace root](#)'.

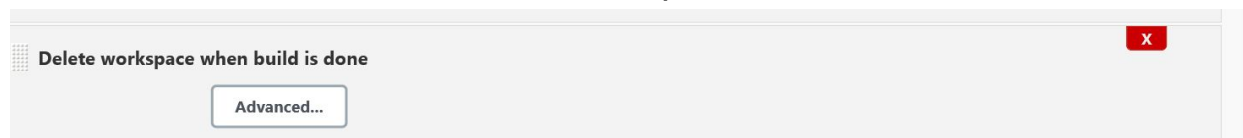
We only want the branch to be merged with main if all the tests pass.



Here we indicate the branch and push target



Once we are done, we delete the workspace so it is clean.



This setup allows for easy building, testing and merging if successful.

Let us work through an example of our workflow


How we implemented Jenkins


Example


First we push our changes to our own branch of our GitHub.


```
PS C:\Users\Tristan Hannibal\Desktop\375\ENSE375-groupB> git add .\README.md
PS C:\Users\Tristan Hannibal\Desktop\375\ENSE375-groupB> git commit -m "update readme"
[tristan e0e75ed] update readme
 1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Tristan Hannibal\Desktop\375\ENSE375-groupB> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 286 bytes | 286.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/JonBarVargas/ENSE375-groupB
   ec1badd..e0e75ed  tristan -> tristan
```


From there we can go to Jenkins, and manually build. This is an unneeded step if we had GitHub webhooks. Pressing 'Build Now' we can see Jenkins begin the process.


 Workspace


 Build Now


 Configure


 Delete Project


 GitHub Hook Log

 GitHub

 Rename


 Workspace

 Recent Char


 Latest Test f


Permalinks

- [Last build \(#43\),](#)
- [Last stable build](#)
- [Last successful b](#)
- [Last failed build](#)
- [Last unsuccessfu](#)
- [Last completed l](#)

 **Build History** trend ^

X

 **#44** [Mar 12, 2021 8:18 PM](#)



We can then view the details of the Jenkins build. Such as commit history and the test results.



Build #44 (Mar 12, 2021 8:18:21 PM)

 [add description](#)



Changes

1. update readme ([commit: e0e75ed](#)) ([details](#) / [githubweb](#))



Started by user [Tristan Brown-Hannibal](#)



Revision: e0e75edfc76b5341b39a5c83f702c059e9675e8b

- origin/tristan
- origin/main



Test Result (no failures)

Going into the test results we can view the XML Gradle outputs in Jenkins itself, seeing which tests pass and fail.

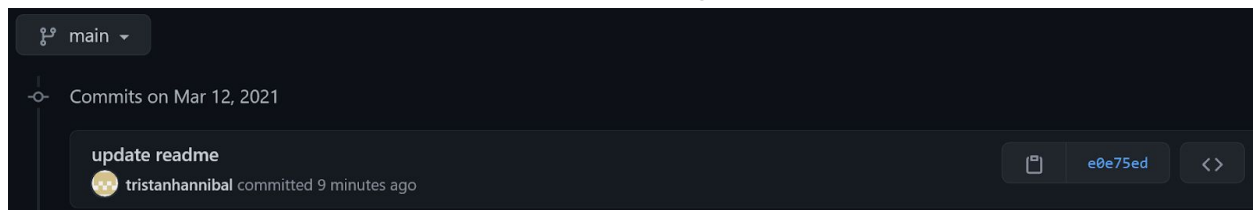
All Tests

Class	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
AppTest	5 ms	0		0		1		1	
PatientHistogramTest	1 ms	0		0		7		7	
PatientListTest	1 ms	0		0		7		7	
PatientTest	0 ms	0		0		1		1	
PostalCodeTest	2 ms	0		0		10		10	
RiskCodeMapTest	1 ms	0		0		1		1	
SampleTest	1 ms	0		0		2		2	

All Tests

Test name	Duration	Status
addPatientToARegion_InvalidHorizontal_False	1 ms	Passed
addPatientToARegion_InvalidVertical_False	0 ms	Passed
addPatientToARegion_Valid_True	0 ms	Passed
deleteAPatient_Invalid_False	0 ms	Passed
deleteAPatient_Valid_True	0 ms	Passed
getPatientsCountInRegion_True	0 ms	Passed
getPatientsCountInRegion_invalidOutOfRange_ThrowException	0 ms	Passed

Since all of our tests passed, Jenkins merged our branch into the main.



If the tests were to fail, nothing would've been merged to main. This process allows for easy collaboration of work, while still ensuring a high quality of code.