**DAY 5 – GITHUB, Apache Log4j, Questions**

**Topics Covered**

1. Selenium Waits
2. Logs
3. Uploading code in github
4. TestNG framework questions clarification
5. Cucumber framework question clarification
6. Java questions clarification
7. Selenium questions clarification.

## Implicit, Explicit and Fluent Waits in Selenium

Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an "**ElementNotVisibleException**" exception. Using Waits, we can resolve this problem.

Let's consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some **"ExpectedConditions** "(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an "**ElementNotVisibleException**".

**Selenium Web Driver Waits**

1. Implicit Wait
2. Explicit Wait

## Implicit Wait

Selenium Web Driver has borrowed the idea of implicit waits from Watir.

The implicit wait will tell to the web driver to wait for certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, web driver will wait for that time before throwing an exception.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

To declare implicit wait:

**Syntax**:

driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);

package guru.test99;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

public class AppTest {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

driver = new ChromeDriver();

driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS) ;

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

driver.manage().window().maximize() ;

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.equals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

//close browser

driver.close();

}

}

**Explanation of Code**

In the above example,

**Consider Following Code:**

driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS) ;

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

## Explicit Wait

The explicit wait is used to tell the Web Driver to wait for certain conditions (**Expected Conditions**) or the maximum time exceeded before throwing an "**ElementNotVisibleException**" exception.

The explicit wait is an intelligent kind of wait, but it can be applied only for specified elements. Explicit wait gives better options than that of an implicit wait as it will wait for dynamically loaded Ajax elements.

Once we declare explicit wait we have to use "**ExpectedCondtions**" or we can configure how frequently we want to check the condition using **Fluent Wait**. These days while implementing we are using **Thread.Sleep()**generally it is not recommended to use

In the below example, we are creating reference wait for "**WebDriverWait**" class and instantiating using "**WebDriver**" reference, and we are giving a maximum time frame of 20 seconds.

**Syntax:**

WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);

package guru.test99;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.annotations.Test;

public class AppTest2 {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

driver = new ChromeDriver();

WebDriverWait wait=new WebDriverWait(driver, 20);

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

driver.manage().window().maximize() ;

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.contentEquals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

WebElement guru99seleniumlink;

guru99seleniumlink= wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath( "/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

}

}

**Explanation of Code**

**Consider Following Code:**

WebElement guru99seleniumlink;

guru99seleniumlink = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

In the above example, wait for the amount of time defined in the "**WebDriverWait**" class or the "**ExpectedConditions**" to occur whichever occurs first.

The above[Java](https://www.guru99.com/java-tutorial.html)code states that we are waiting for an element for the time frame of 20 seconds as defined in the "**WebDriverWait**" class on the webpage until the "**ExpectedConditions**" are met and the condition is "**visibilityofElementLocated**".

The following are the Expected Conditions that can be used in Explicit Wait

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. elementToBeSelected()
5. frameToBeAvaliableAndSwitchToIt()
6. invisibilityOfTheElementLocated()
7. invisibilityOfElementWithText()
8. presenceOfAllElementsLocatedBy()
9. presenceOfElementLocated()
10. textToBePresentInElement()
11. textToBePresentInElementLocated()
12. textToBePresentInElementValue()
13. titleIs()
14. titleContains()
15. visibilityOf()
16. visibilityOfAllElements()
17. visibilityOfAllElementsLocatedBy()
18. visibilityOfElementLocated()

## Fluent Wait

The fluent wait is used to tell the web driver to wait for a condition, as well as the **frequency**with which we want to check the condition before throwing an "ElementNotVisibleException" exception.

**Frequency:**Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time

Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

**Syntax:**

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

Above code is deprecated in Selenium v3.11 and above. You need to use

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(Duration.ofSeconds(SECONDS))

.pollingEvery(Duration.ofSeconds(SECONDS))

.ignoring(Exception.class);

package guru.test99;

import org.testng.annotations.Test;

import java.util.NoSuchElementException;

import java.util.concurrent.TimeUnit;

import java.util.function.Function;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.FluentWait;

import org.openqa.selenium.support.ui.Wait;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.annotations.Test;

public class AppTest3 {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

driver = new ChromeDriver();

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

driver.manage().window().maximize() ;

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.contentEquals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

WebElement clickseleniumlink = wait.until(new Function<WebDriver, WebElement>(){

public WebElement apply(WebDriver driver ) {

return driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

}

});

//click on the selenium link

clickseleniumlink.click();

//close~ browser

driver.close() ;

}

}

**Explanation of Code**

**Consider Following Code:**

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring "**NoSuchElementException**"

**Consider Following Code:**

public WebElement apply(WebDriver driver) {

return driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

We have created a new function to identify the Web Element on the page. (Ex: Here Web Element is nothing but the selenium link on the webpage).

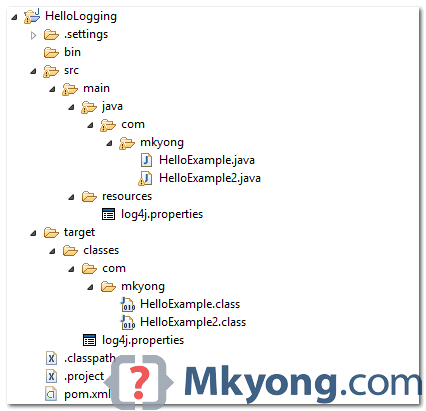
Frequency is set to 5 seconds and the maximum time is set to 30 seconds. Thus this means that it will check for the element on the web page at every 5 seconds for the maximum time of 30 seconds. If the element is located within this time frame it will perform the operations else it will throw an"**ElementNotVisibleException**"

# Log4j hello world example

In this tutorial, we will show you how to use the classic log4j 1.2.x to log a debug or error message in a Java application.

## 1. Project Directory

Review the final project structure, a standard Maven style Java project.



## 2. Get Log4j

Declares the following dependencies :

pom.xml

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

Copy

For non-Maven user, visit [log4j official page](http://logging.apache.org/log4j/1.2/), download the jar and put it in the project library path manually.

## 3. log4j.properties

Create a log4j.properties file and put it into the resources folder. Refer to the step #1 above.

log4j.properties

# Root logger option

log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

# Redirect log messages to a log file, support file rolling.

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=C:\\log4j-application.log

log4j.appender.file.MaxFileSize=5MB

log4j.appender.file.MaxBackupIndex=10

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

Copy

**Note**  
To understand the symbols in the ConversionPattern, please refer to this [log4j PatternLayout guide](http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html).

Let break it down :

1. %d{yyyy-MM-dd HH:mm:ss} = Date and time format, refer to [SimpleDateFormat](http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html) JavaDoc.
2. %-5p = The logging priority, like DEBUG or ERROR. The -5 is optional, for the pretty print format.
3. %c{1} = The logging name we set via getLogger(), refer to [log4j PatternLayout guide](http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html).
4. %L = The line number from where the logging request.
5. %m%n = The message to log and line break.

Log message examples :

2014-07-02 20:52:39 DEBUG className:200 - This is debug message

2014-07-02 20:52:39 DEBUG className:201 - This is debug message2

Copy

## 4. Demo – How to log a Message?

To log a message, first, create a final static logger and define a name for the logger, normally, we use the full package class name.

final static Logger logger = Logger.getLogger(classname.class);

Copy

Then, logs messages with different priorities, for example, debug, info, warn, error and fatal. Normally, you just need to use debug or error.

//logs a debug message

if(logger.isDebugEnabled()){

logger.debug("This is debug");

}

//logs an error message with parameter

logger.error("This is error : " + parameter);

//logs an exception thrown from somewhere

logger.error("This is error", exception);

Copy

4.1 Example : Logger is set to **debug** priority.

log4j.properties

log4j.rootLogger=DEBUG, stdout

#...

Copy

HelloExample.java

package com.mkyong;

import org.apache.log4j.Logger;

public class HelloExample{

final static Logger logger = Logger.getLogger(HelloExample.class);

public static void main(String[] args) {

HelloExample obj = new HelloExample();

obj.runMe("abcd");

}

private void runMe(String parameter){

if(logger.isDebugEnabled()){

logger.debug("This is debug : " + parameter);

}

if(logger.isInfoEnabled()){

logger.info("This is info : " + parameter);

}

logger.warn("This is warn : " + parameter);

logger.error("This is error : " + parameter);

logger.fatal("This is fatal : " + parameter);

}

}

Copy

Output

2014-07-02 20:52:39 DEBUG HelloExample:19 - This is debug : abcd

2014-07-02 20:52:39 INFO HelloExample:23 - This is info : abcd

2014-07-02 20:52:39 WARN HelloExample:26 - This is warn : abcd

2014-07-02 20:52:39 ERROR HelloExample:27 - This is error : abcd

2014-07-02 20:52:39 FATAL HelloExample:28 - This is fatal : abcd

Copy

4.2 Example – Logger is set to **error** priority.

log4j.properties

log4j.rootLogger=error, stdout

#...

Copy

Run the HelloExample again, you will get the following output

2014-07-02 20:56:02 ERROR HelloExample:27 - This is error : mkyong

2014-07-02 20:56:02 FATAL HelloExample:28 - This is fatal : mkyong

Copy

Review the log4j’s Priority class.

Priority.java

package org.apache.log4j;

public class Priority {

public final static int OFF\_INT = Integer.MAX\_VALUE;

public final static int FATAL\_INT = 50000;

public final static int ERROR\_INT = 40000;

public final static int WARN\_INT = 30000;

public final static int INFO\_INT = 20000;

public final static int DEBUG\_INT = 10000;

//public final static int FINE\_INT = DEBUG\_INT;

public final static int ALL\_INT = Integer.MIN\_VALUE;

Copy

If priority is defined in log4j.properties, only the same or above priority message will be logged.

## 5. Demo – How to log an Exception

An example to show you how to use log4j to log an exception.

HelloExample2.java

package com.mkyong;

import org.apache.log4j.Logger;

public class HelloExample2{

final static Logger logger = Logger.getLogger(HelloExample2.class);

public static void main(String[] args) {

HelloExample2 obj = new HelloExample2();

try{

obj.divide();

}catch(ArithmeticException ex){

logger.error("Sorry, something wrong!", ex);

}

}

private void divide(){

int i = 10 /0;

}

}

Copy

Output

2014-07-02 21:03:10 ERROR HelloExample2:16 - Sorry, something wrong!

java.lang.ArithmeticException: / by zero

at com.mkyong.HelloExample2.divide(HelloExample2.java:24)

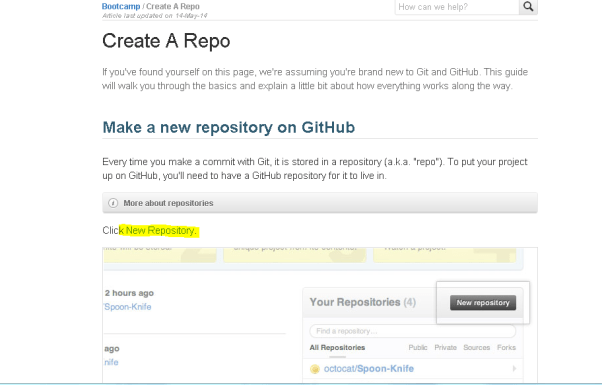
at com.mkyong.HelloExample2.main(HelloExample2.jav

## Upload Selenium Script to GITHUB using Eclipse

Step 1- First of all Create an account in github with your valid email and other information.

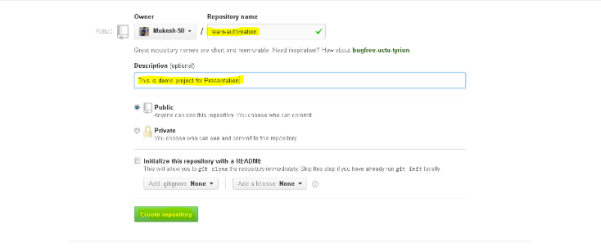
Step 2- Now login to [github](https://github.com/) account using credential then navigate Home page.

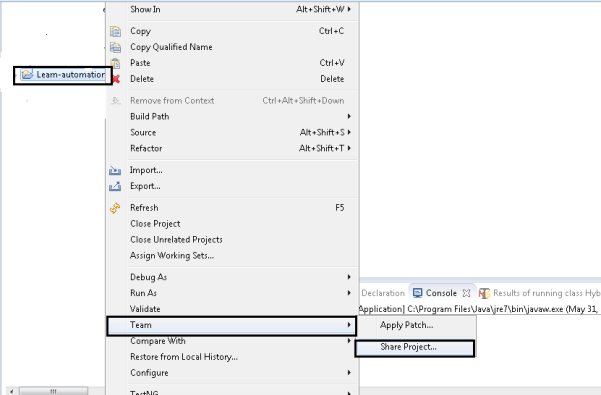
Step 3- Click on Create repository.  
[](https://i0.wp.com/learn-automation.com/wp-content/uploads/2015/03/200.png)

Step 4- Click on New Repository.  
[](https://i0.wp.com/learn-automation.com/wp-content/uploads/2015/03/201.png)

Step 5- Specify the name of the repository, description and click on create repository.

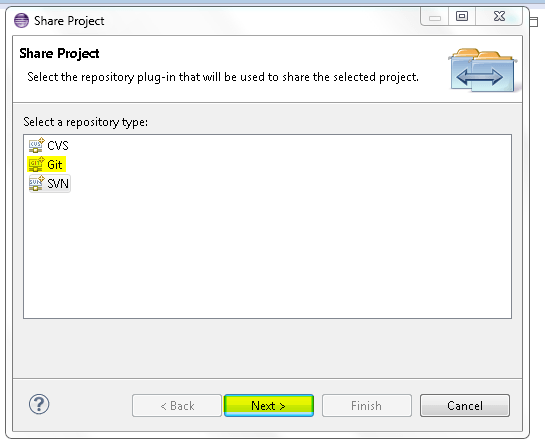
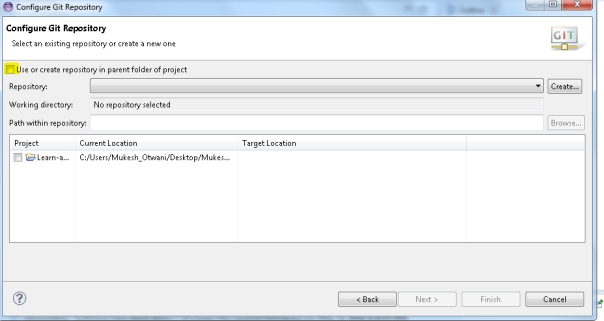
***Note- Github has public and private repository feature, so if you are using public repository then anybody on Github can see your code.***

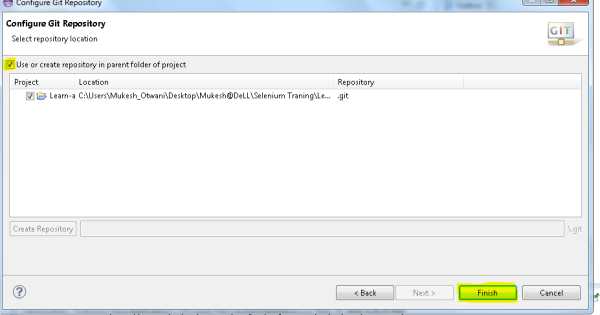
In case private repository, only who is added as a member in this repository can see this repo and can make the changes but for private repository we have to pay some amount.  
[](https://i0.wp.com/learn-automation.com/wp-content/uploads/2015/03/202.png)  
Step 6- Now copy the URL of your repository.  
  
Step 7- Now open Eclipse  and  Select project which we want to upload on github then  right click and  Go to team section and  Select share project.

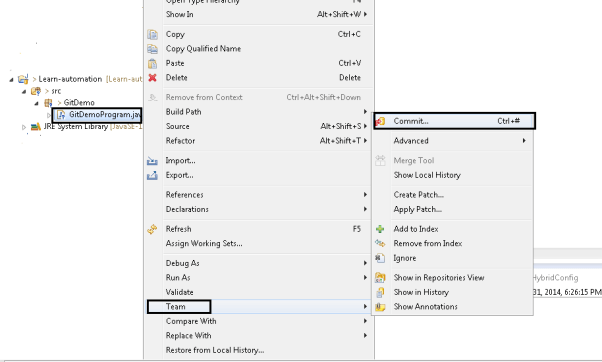
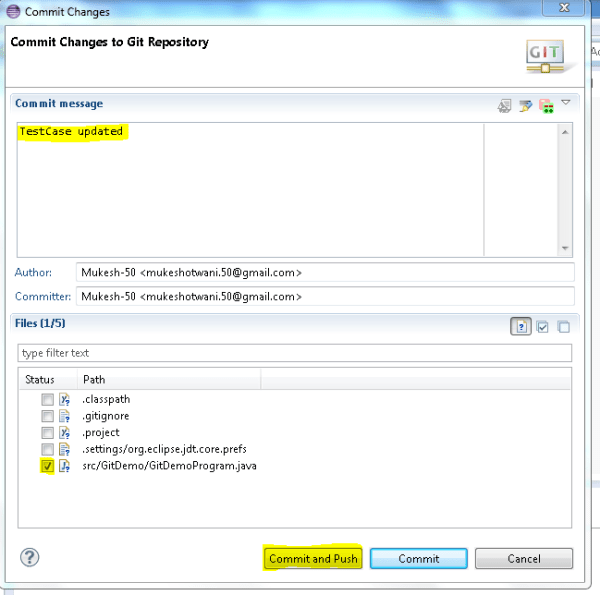
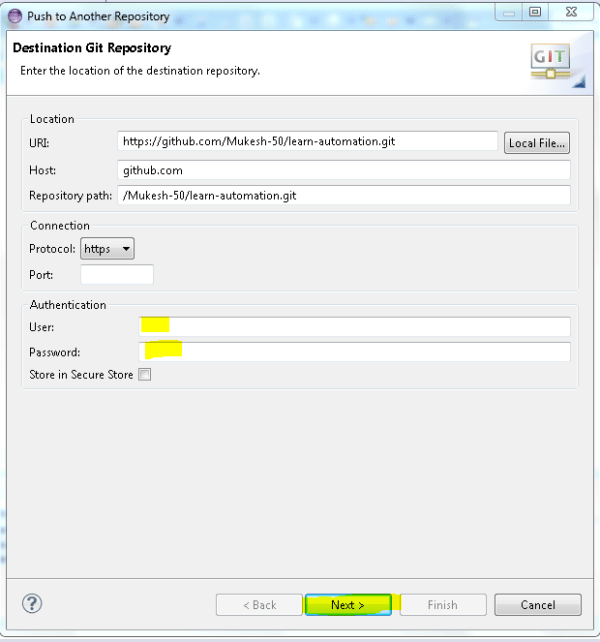
  
Step 8- Here you will get some other repository type as well.

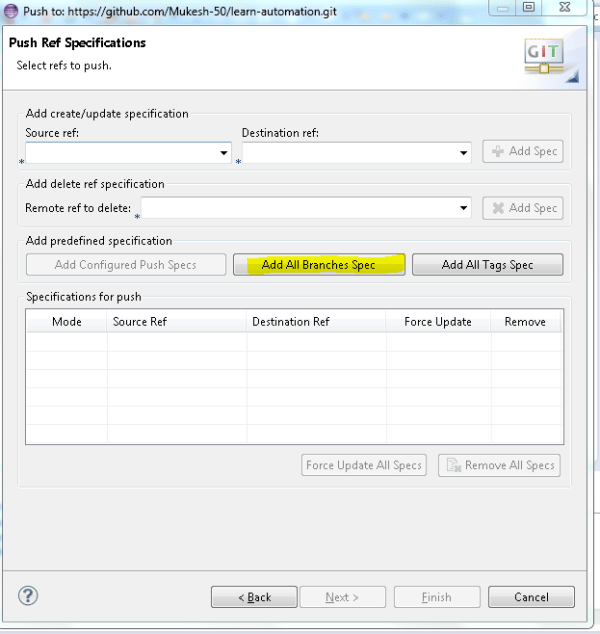
Select Git and complete the steps.

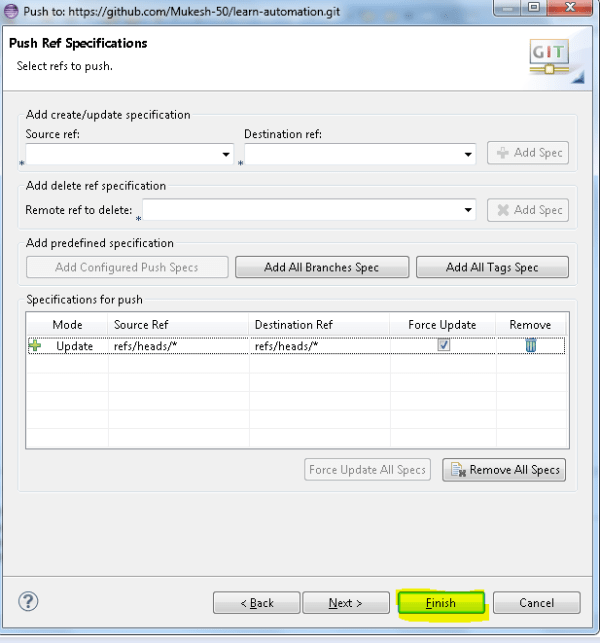
 Select Git and complete the steps.

[](https://i1.wp.com/learn-automation.com/wp-content/uploads/2015/03/205.png)  
[](https://i1.wp.com/learn-automation.com/wp-content/uploads/2015/03/206.png)

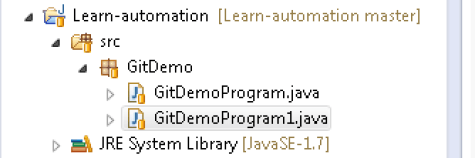
[](https://i2.wp.com/learn-automation.com/wp-content/uploads/2015/03/207.png)

Step 9-  Whenever we are creating new test cases or we are modifying our old test cases then we need to do checkout so that we can have updated copy on Github.  
Now select Test case > right click > team > select commit.  
[](https://i0.wp.com/learn-automation.com/wp-content/uploads/2015/03/208.png)  
Step 10- Here we have to specify comment and Select the file which need to update and click on Commit and Push.  
[](https://i0.wp.com/learn-automation.com/wp-content/uploads/2015/03/209.png)  
Step 11- Enter valid github credentials and complete the steps.  
[](https://i2.wp.com/learn-automation.com/wp-content/uploads/2015/03/210.png)

[](https://i1.wp.com/learn-automation.com/wp-content/uploads/2015/03/211.png)

[](https://i1.wp.com/learn-automation.com/wp-content/uploads/2015/03/212.png)

Click on finish and now your project is up to date with Git Repository.

[](https://i2.wp.com/learn-automation.com/wp-content/uploads/2015/03/213.png)

If you open Github and navigate to your repository just do a refresh and your full code will be updated.

**References –**

1. Maven Selenium Cucumber setup - <https://www.axelerant.com/resources/team-blog/setup-for-selenium-with-cucumber-using-maven>
2. Log4J with Selenium - <https://www.guru99.com/tutorial-on-log4j-and-logexpert-with-selenium.html>
3. Log4j with Maven - <https://www.mkyong.com/logging/log4j-hello-world-example/>
4. Github integration with Eclipse - <http://learn-automation.com/github-integration-eclipse-with-selenium/>