**Selenium Training - Session 68**

..

**Jenkins, Git and GitHub**

* Explain the process of working on a project in group - View [here](https://drive.google.com/file/d/10bwMmLbnOEaJvfirwHBsaiEpEA6zOfdb/view?usp=sharing)
  + Check-ins the code to Central Repository Server
  + Checkouts to the local machines
  + Modify or fix the code and check-ins as version 2 and so on.
  + Updates the local code with the latest code in Central Repository Server
* Best practice to follow while working in a real time project
* As everyone is working on the same code and making the changes, there is a problems of build failures and will be very difficult to identify when the build failure occurred and after who modified the code and etc.
* To overcome the above problem, we have to use the combination of **Jenkins, Git and GitHub** for achieving continuous integration, collaboration and version controlling.

**Selenium Training - Session 69**

**Jenkins, Git and GitHub  (Continued)**

* **GitHub**
  + Create a public account on GitHub
  + GitHub is a code hosting platform for version control and collaboration.
  + Create a repository say 'DemoProject' to store the code on the server instead of in local.
* **Git**
  + Git is a version control system using which **we can version**, **upload**, **maintain** and **download the code** from GitHub
  + Install 32 bit version of Git
* **Jenkins**
  + Download Jenkins and install
  + On browsing localhost:8080, all the plug-ins required by Jenkins will be installed
  + Create a free style project in Jenkins and explain the building process
    - Source code option set to **none** will build the code from a local machine
    - Demonstrate building process using Windows command > echo %time%
    - Select 'Build Now' option to build
    - Go to Console output to view the results
    - Can build periodically
      * \* \* \* \* \* (Every Minute)
      * 0-0 5 \* \* \*  (Every day at 5 AM)
      * 0-0 0 \* \* \* (Every day at 12 midnight)
      * 0-0 5 \* \* 3 (Every Wednesday 5 AM)
      * 0-0 H/6 \* \* \* (Every 6th hour)
      * H/5 \* \* \* \* (Every 5 mins)
      * 0-59/2 \* \* \* \* (Every 2 mins)
  + We can invoke ANT > build.xml on Local machine using Jenkins by providing the complete build path and targets like run, compile etc.
  + Enable the Auto refresh in Jenkins

**Selenium Training - Session 70**

**Jenkins, Git and GitHub  (Continued)**

* Explain how the Jenkins, Git and GitHub are going to work together - view [here](https://drive.google.com/file/d/11TjgDJW-JwkKg5SwXjKEiuTtZ_quPZcM/view?usp=sharing)
* Configure the GitHub to work with Jenkins
  + Create a new repository in GitHub
  + Go to a folder in your machine where want to store the GitHub repository code
  + Open Git Bash from inside the above navigated folder
  + Type **pwd** command to confirm whether you are at the right path
  + Take the clone of the existing repository on GitHub using the below command:
    - Example: **git clone**[**https://github.com/seleniumbyarun/ProjectXYZ.git**](https://github.com/seleniumbyarun/ProjectXYZ.git)
    - Cloning will begin and an empty repo folder with a .git folder from GitHub will be copied to your local machine
  + Type the below two commands to provide a name and email ids to the activities that you are going to perform on GitHub
    - **git config --global user.name "Arun Motoori"**
    - **git config --global user.email "arun.motoori@gmail.com"**
  + Go inside the cloned folder on your local machine by using the below command
    - **cd ProjectABC**
    - Type **pwd** to confirm you are in the correct path
  + Type **git status** command and you will get nothing to commit
  + Now copy the Project files you want to upload to GitHub central repository
  + Again type **git status** command and you will this time get the details that can be committed
  + Type **git add .** to include all the files for committing
  + Commit the changes to confirm by using the command **git commit -m "MyFirstProjectCommit"**
  + Push the changes to the server by using the command **git push -u origin master**
  + After the above process is done, refresh the GitHub Project Repo page to see the Copied files in the repository
  + Go to Jenkins and create a new project
  + Configure the project by selecting the 'Git' option under Source Code Management and paste the URL of the GitHub Project
    - Example URL: <https://github.com/seleniumbyarun/ProjectABC>
  + Under Build section, select 'Invoke ANT' option and just provide the 'build.xml' into Build File field and 'run' target into the 'Targets' field and Save the Jenkins Project configurations.
  + Go the Dashboard project and build the Project using 'Build Now' option
  + View the results in the Console of Jenkins Project.
  + Install TestNG Results Plugin and configure the existing Jenkins Project by selecting 'Post Build' configuration option to 'Publish TestNG Reports'
  + Build the project again and view the generated TestNG results

**Selenium Training - Session 71**

**Jenkins, Git and GitHub  (Continued)**

What is **GitHub**?

GitHub is one of the source code management systems.

Steps to upload the local code to GitHub:

1. Create a new folder in your local machine

2. Create any folder with your desired project name in the above created folder

3. Launch git-bash inside the project folder and run 'git init' command

4. Copy the local code folders into the above created project folder

5. In gitbash, execute git status command

6. Execute git add . command

7. Execute git status

8. Execute git commit -m "first commit"

9. Execute git remote add origin sshurl

10. Generate ssh key by executing the command 'ssh-keygen -t rsa -b 4096 -C "arun.motoori@gmail.com"

11. Add the generated key to github account

12. Execute ssh -T git@github.com to verify the establishment of connection between your local machine and github

13. Execute git push origin master and refresh the GitHub repository to see the reflected code

Steps to execute code from github:

1. Do some modification say adding a small comment in one of the files and push the changes from local machine to github repository

2. Install Maven Integration plugin, create a new Maven Project in Jenkins and configure it to 'git', Provide Repository URL, mention clean compile install and set Post build actions to TestNG reports and Build the project

3. Intentionally fail the test by modifying the CRM locator option in HomePage.java and push the code to GitHub and Build it using Jenkins

4. Revert back the faulty code by fixing the mistake in HomePage.java and push the code to GitHub and build it using Jenkins

**Selenium Training - Session 72**

**Jenkins, Git and GitHub  (Continued)**

Git Commands:

1) For finding the git version

* git --version

2) Tell Git who you are :

* git config --global user.name "Arun Motoori"
* git config --global user.email "arun@qafox.com"
* git config --list (All the configurations with their values will be listed)

3) Get help with any git command

* git config --**help**
* git add **--help**
* git commit **--help**

4) Create a new local repository and initialize it to a git repository:

* git init

3) Checkout a repository

* git clone clonesshurl

4) Find the local code changes:

* git status

5) Understand where you are ?

* Are you at working directory, Staging area or remote repository

6) Add Files to staging area

* git add abc.txt
* git add .
* git add -A

7) Reset the added files back to working directory

* git reset
* git reset abc.txt

6) Commit the changes

* git commit -m "first commit"

7) Adding the github repostory location

* git remote add origin clonesshurl

8) Generating the ssh key (Only required to establish secured connection between local machine and github)

* ssh-keygen -t rsa

7)Validating the proper establishment of secured connection

* ssh -T git@github.com

8)Send changes to the master brach

* git push origin master

9) Update the local code with the changes done at remote repository

* git pull
* git pull origin master

10)For forcefully reverting the previous commit

* git reset --hard
* git pull

11) For clearing the changes in local

* git stash

14) Tracking the commits

* git log

15) View information about remote repository

* git remote -v

16) Finding the changes made

* git diff

17) To know different branches in the current repository

* git branch
* git branch -a (Shows all the local and remote branches)

18) Creating a new branch

* git branch branchname

19) Changing the branch

* git checkout branchname

Do some changes to local code and push to new branch

Check the changes in the github.com

20) Switch back to master branch

* git checkout master

21) To find whether all the branches have been merged to master

* git branch --merged

22)Merge the branch code to master branch

* git merge branchname

Now push the merged changes to the remote master branch

* git push origin master

Now recheck whether all the branches have been merged to master

* git branch --merged

23) Delete the branch

* git branch -d branchname

Now check the available branches

* git branch
* git branch -a

24) Delete the branch from remote repository

* git push origin --delete branchname

**Selenium Training - Session 73**

**Jenkins, Git and GitHub  (Continued)**

Jenkins Continued:

What is Jenkins?

Jenkins is a continuous integration tool which:

1) keeps on polling the code in GitHub,

2) immediately builds/runs the latest code or schedule the builds/runs every 1 hour etc.

3) And mails all the team members if the build fails.

Do you have to install Jenkins at your workplace?

No need to install Jenkins on your personal machines. At your workplace, you will be directly provided with the Jenkins URL (say http://jenkins.companyname.com) and your username and password to log in to Jenkins.

Email Notifications:

-- Allow your gmail to access less secured apps by enabling the option at the below url

    https://www.google.com/settings/security/lesssecureapps

-- Manage Jenkins > Configure System > Email Notification Settings

        SMTP server : smtp.gmail.com

        Default user e-mail suffix : @gmail.com

        Use SMTP Authentication: select this checkbox

        User Name:arun.motoori@gmail.com

        Password: your gmail password

        USE SSL: select this check-box

        SMTP PORT: 465

        Charset: UTF-8

        Test Configuration: Select this checkbox and provide any gmail address and test

-- Configure Project for Email settings (Project > Configure > Build Settings)

    Provide email ids as arun.motoori@gmail.com, arrampriyanka2009@gmail.com

    Observe that when the build fails, Jenkins will automatically send the emails to the provided email addresses.

    Click on the link provied in email to findout the root cause of failure.

-- Schedule the build to be executed every minute ( \* \* \* \* \*)

    Best Practices:

        -- Build will be generally scheduled at the night say 11 PM or 1 AM etc.

        -- Execute the Jenkins builds multiple times say 10 to 15 times to check whether the build is constantly stable or not.

-- Schedule the build to executed for every code change in GitHub

        -- Select 'Poll SCM' option and type \* \* \* \* \* in the Schedule text area field (This will poll the code in GitHub for every minute to check any changes to the code)

        -- If any changes are made, the Jenkins will build/run the code

        -- In real time, we choose to schedule the build periodically and also select to build when something changes in github

**Selenium Training - Session 75**

**Live Project**

Follow the below steps to start working on the real time project:

1.Download the Existing Framework from here - [Click here](https://drive.google.com/file/d/1t4tsUhMYSCBGqhIr2PXoHYrRRmPtYgpC/view?usp=sharing)

2.Extract the zip file and copy the project folder along with its items in any folder location

3.Create a GitHub Repository say DemoQAFoxLiveProject (Include ReadMe file)

4.Clone the GitHub Repo into your local machine

5.Copy the files available in the Framework which we have downloaded in step 1

6.Import the Project into Eclipse IDE and resolve the errors if any

7.Open GitBash inside the cloned Project folder

8.Upload the updated code/files to GitHub repository

* git status
* git add .
* git commit -m "Uploading Initial automation framework"
* git push origin master

9.Delete any existing feature files and Create Register.feature file and first scenario under src/test/resources > Features folder - [View Scenario here](https://drive.google.com/file/d/1h5AxP_YFtIphwMeYZNI-VN3KlMpw21ki/view?usp=sharing)

10.Create Register.java file under src/test/java > stepdef package and implement all the steps in first scenario

* First run the feature file without implementation to get the high level implementation in output
* Remove the errors and unnecessary comments/code which is auto-generated
* Write the code for opening the Application URL in the browser and Understand
* Copy the Hooks under the step-def package - [Download Hooks class here](https://drive.google.com/file/d/1Ly4NCtJOwSljGijw1owf8aBt7rGGIN6H/view?usp=sharing)
* Understand the methods in Hooks class
* Remove the Hooks related methods from Base Class
* Close and open the Register.feature file and observe that I am getting an error
* To overcome this error, I will remove Cucumber-eclipse add-on from Eclipse IDE and in place of it I will install Cucumber JVM eclipse plugin from <https://marketplace.eclipse.org/content/cucumber-jvm-eclipse-plugin>
* Update the Runner class to execute the so far implemented scenario - [View Runner Class here](https://drive.google.com/file/d/1QD4Sj_fHA-4OcGnpqNMI1XopJ4WujAjV/view?usp=sharing)
  + @RunWith(Cucumber.**class**)
  + @CucumberOptions(features={"classpath:FeatureFiles/Register.feature" },

glue={"classpath:com.tutorialsninja.automation.stepdef"},

plugin={"html:target/cucumber\_html\_report"},

tags={"@Register", "@One"})

* Execute the Runner class using JUnit and check the result
* If everything goes well, upload the updated code to GitHub

**Selenium Training - Session 76**

**Live Project (Continued)**

Follow the below steps to continue:

1.Continue automating the remaining steps in the first scenario

* Implement**I navigate to Account Registration page** step
  + Elements.click(HeadersPage.myAccountLink) & Elements.click(HeadersPage.register)
    - Create HeadersSection java class file under src/main/java > pages package
    - Create the WebElement for myAccountLink and register - [View here](https://drive.google.com/file/d/1BX8TmwjJteXc3ADwZrx6mQ75Bk47hFoH/view?usp=sharing)
    - Add static keyword for all the methods inside the Elements.java class
    - Create a constructor in the HeadersPage to initialize the WebElements - [View here](https://drive.google.com/file/d/1aMfT4D5r2_z2KaP06BGb6gfvW3r_93Et/view?usp=sharing)
    - Create an object for HeadersSection in Register java file
* Implement **I provide all the below valid details**step
  + Create RegisterPage java class file under src/main/java > pages package
  + Create the WebElement for all the scenario required fields in the RegisterPage - [View here](https://drive.google.com/file/d/1m88OMiFuHggtoYSP1LN5IMJiNZpTBrD2/view?usp=sharing)
  + Write the code to enter the required details into the required fields - [View here](https://drive.google.com/file/d/1Rw-sETKNpmqDr7VMnSQFSOgZcJapJpt7/view?usp=sharing)
  + Create a constructor to initialize the WebElements in RegisterPage
  + Create an object for RegisterPage in Register java file to call the constructor
  + Create a reusable method in RegisterPage for the code implemented under this step - [View here](https://drive.google.com/file/d/1u-osCYTTcugfRudIzlFeCYdzybK7ODqG/view?usp=sharing)
  + Call the reusable method from the RegisterPage current step - [View here](https://drive.google.com/file/d/126JcbT1mQUa4smMWjqwC64HNpg1sJT1X/view?usp=sharing)
* Implement **I select the Privacy Policy** step
  + Create the WebElement for the Privacy Policy checkbox option in the RegisterPage
  + Elements.click(RegisterPage.privacyPolicy)
* Implement **I click on Continue button** step
  + Create the WebElement for the Continue Button in the RegisterPage
  + Elements.click(RegisterPage.continueButton)
* Implement **I should see that the User Account has successfully created** step
  + Create AccountSuccessPage java class file under src/main/java > pages package
  + Create the WebElement for Success breadcrumb in the AccountSuccessPage
  + Create a constructor to initialize the WebElements in AccountSuccessPage
  + Create an object for AccountSuccessPage in Register java file to call the constructor
  + Assert.assertTrue(Elements.isDisplayed(AccountSuccessPage.successBreadcrumb));
* Execute the first scenario for the first time and observe that the scenario gets passed
* Re-execute the same scenario for the second time and observe that the scenario gets failed
  + View the reports to identify the problem
* To overcome this problem, append times-tamp while entering the email address as shown below:
  + Elements.typeTextIfElementPresent(email,System.currentTimeMillis()+map.get("Email"));
  + Execute the script multiple times and observe that the scenario will get passed every-time

..**Selenium Training - Session 77**

**Live Project (Continued)**

Follow the below steps to continue:

1.Create second scenario in the Register feature file - [view here](https://drive.google.com/file/d/1TJkPnsEv5qmEofKHJ8y4IlIopP-Yvbag/view?usp=sharing)

2.Create step definitions for the new steps in the existing Register java file

* Implement**I launch the application** step
  + No need to Implement as this is already implemented as part of first scenario
* Implement **I navigate to Account Registration page** step
  + No need to Implement as this is already implemented as part of first scenario
* Implement**I click on Continue button** step
  + No need to Implement as this is already implemented as part of first scenario
* Implement **I should see that the User Account is not created** step
  + Execute the second scenario in the feature file using Runner Class without implementing this step
  + Copy and use the suggested syntax for implementation of this step in Register java class
  + Start implementing this step - [View here](https://drive.google.com/file/d/1aMSixuDxRB8mek9pTHAKDaKTx9ul2A8I/view?usp=sharing)
    - Remove the unnecessary code
    - Create WebElement for register breadcrumb in RegisterPage java class
    - Implement the step
* Implement **I should see the error messages informing the user to fill the mandatory fields** step
  + Create WebElements for warning messages
  + Implement the step - [View here](https://drive.google.com/file/d/1wU5xNgzdmw8a4-hg-w7dmyqYN7r2DHlC/view?usp=sharing)
* Execute the second scenario

3.Create third scenario in the Register feature file - [view here](https://drive.google.com/file/d/19OGZT0TijdWa2M1dm9ylmFxm_piUWj6O/view?usp=sharing)

4.Create step definitions for the new steps in the existing Register java file

* Execute the third scenario in the feature file using Runner Class without implementing this step
* Copy and use the suggested syntax for implementation of this step in Register java class
* Implement the new step: **I subscribe to Newsletter**- [View here](https://drive.google.com/file/d/1aMSixuDxRB8mek9pTHAKDaKTx9ul2A8I/view?usp=sharing)
  + Remove the unnecessary code
  + Create WebElement for yes radio option of Newsletter
  + Implement the step
* Execute the third scenario

5.Create fourth scenario in the Register feature file - [view here](https://drive.google.com/file/d/1HuOa5kw3a6L9RAWF26t_fpZMq0u1-f1_/view?usp=sharing)

6.Create step definitions for the new steps in the existing Register java file

* Execute the fourth scenario in the feature file using Runner Class without implementing this step
* Copy and use the suggested syntax for implementation of this step in Register java class
* Implement the new step: **I provide the below duplicate details into the fields**
  + Remove the unnecessary code
  + Create another reusable method say enterDuplicateDetails having the similar code of enterAllDetails - View here - [View here](https://drive.google.com/file/d/1EqpEsJt341m77T3Ee0J2lxI1_LMoIwXq/view?usp=sharing)
  + Optimize the two reusable methods to one and change the parameter in calling methods - [View here](https://drive.google.com/file/d/1_arOfis-Sgy1z8N18SN1BvjH_gOvqwRX/view?usp=sharing)
  + Call the optimized reusable method by passing the "duplicate" as argument
* Implement the new step:
  + Remove the unnecessary code
  + Implement the step - [View here](https://drive.google.com/file/d/1g_akezFQ0vPGdEgowqcjnvK2MDX8_m7K/view?usp=sharing)
* Execute the fourth scenario

7.Execute all the scenarios in the register feature file individually

8.Execute all the scenarios in the register feature file in a single go

* Modify the tags attribute in Runner class

9.Upload the updated local code to GitHub

* Commit Message - Completed automating all the scenarios in the Register feature file

..**Selenium Training - Session 79**

**Live Project (Continued)**

Follow the below steps to continue:

1.Create Search.feature file and first scenario under src/test/resources > Features - [View here](https://drive.google.com/file/d/170mTBeSZmd2T3MNQCA3iKj7V-4hTGYXx/view?usp=sharing)

2.Create Search.java file under src/test/java > stepdef package and implement all the steps in first scenario

* First run the feature file without implementation using Runner Class to get the high level implementation in output
* Remove the errors and unnecessary comments/code which is auto-generated
* Implement**I search for a product "Samsung SyncMaster 941BW"** step
  + Create the WebElement for searchBoxField and searchButton in HeadersSection class
  + Create an object for HeadersSection in Search java file
  + Implement the step in Search class - [View here](https://drive.google.com/file/d/1Nu-tOfUxQyUb1NT3AXobKk3OPnBi3zD0/view?usp=sharing)
* Implement**I should see the product in the search results** step
  + Create SearchResultsPage java class under src/main/java > pages package
  + Create WebElement for Samsung SyncMaster 941BW search result in SearchResultsPage
  + Create a constructor in SearchResultsPage to initialize the WebElement
  + Implement the step in Search class - [View here](https://drive.google.com/file/d/1wqRX9IrP_zQorsQzH9JbLj7UwAGfMZr9/view?usp=sharing)
    - Create an object for SearchResultsPage in Search class
* Execute the first scenario of Search.feature file

3.Create second scenario under src/test/resources > Features - [View here](https://drive.google.com/file/d/1nxedumSJeNP25ZJK5MqabjRtRV0f7xSj/view?usp=sharing)

* Create step definitions for the new steps in the existing Login java file
  + Execute the second scenario in the Search feature file using Runner Class without implementing this step
  + Copy and use the suggested syntax for implementation of this step in Search java class
  + Implement the new step: **I should see the page displaying the message "Your shopping cart is empty!"**
    - Remove the unnecessary code
    - Implement the step - [View here](https://drive.google.com/file/d/1fPyBP62sEwsMXH_hIWDNjFfwxfC25-h-/view?usp=sharing)
      * Create a WebElement for no results message in SearchResultsPage
* Execute the second scenario of Search.feature file

4.Execute all the scenarios in the Search feature file individually

7.Execute all the scenarios in the Search feature file in a single go

* Modify the tags attribute in Runner class

7.Execute all the scenarios in all register, login and search feature files in a single go

* features={"classpath:FeatureFiles/Login.feature","classpath:FeatureFiles/Register.feature","classpath:FeatureFiles/Search.feature" }

8.Upload the updated local code to GitHub

* Commit Message - Completed automating all the scenarios in the Login feature file

**Selenium Training - Session 81**

**Live Project (Continued)**

Follow the below steps to continue:

1.We already have Jenkins installed in our machine

2.Create a new job in Jenkins and configure it to execute the so far created 11 scripts using the code hosted in gitHub repository

3.Build the job and observe that we get some errors

4.To investigate the error go to the code in the local repository and run the pom.xml file using maven and observe that the same errors will be displayed

5.To overcome this problem in local, rename the Runner.java to TestRunner.java and execute the pom.xml file in local repository using maven and observe that all the scripts are executed properly

6.Upload the changes in above step to github repository

7.Executed the updated code in github repository using Jenkins and observe that the errors didn't get resolved

8.Add the below properties to the pom.xml file before the dependencies tags in the local repository and upload the updated pom.xml file to github repository

* **<properties> <maven.compiler.source>1.8</maven.compiler.source> <maven.compiler.target>1.8</maven.compiler.target> </properties>**

9.Build the job to see that the execution is going on but most of the scenarios getting failed as they run in headless mode

9.To overcome the headless problem, we have to launch the Jenkins using the war file

* Go to the services, change the Jenkins service startup type to Manual and stop the service
* Go to C driver > Programs files (x86) > Jenkins folder and install the jenkins.war in the folder using the below command from command prompt:
  + java -jar jenkins.jar --httpPort=8090
  + Once the Jenkins is started and running, access http://localhost:8090
  + Install the required plugins and create user account details and login
  + Create a maven project in jenkins and configure it to run the code in github repository
  + Observer that all the scenarios will now be executed in real mode instead of headless mode

---- End of live project -------