This is the beginning of your QA Automation journey! From here on, you will be challenged to do your best as a future tester.

In the following days, you'll be learning about Continuous Deployment and Integration.

Let's dive right in!

GIT Intro, Terms and Mechanics

In this lesson, we'll be covering the basics of Git. Why are we taking the time to learn this in the first place?

For starters, this tool allows us to answer the following question: **How can I have my code on a centralized repository for everyone to access it?** This is when Git comes in. Take a look at the following video and exercises to pace your way through this content.

https://youtu.be/CUPc76NFMa0 (video)

Exercise

Preconditions

1. Read the following document about Version control: **What is version control**.
2. Read the following document about Git: **What is Git**
3. Read the following document about why should we use git: **Why Git**
4. Follow the instructions here: **Install Git**, to install git on your computer.

Practical Exercise

You have to create a centralized repository, which you should have access to from your local workspace, to achieve this, create your own repository on **github.com** and clone it into your local workspace.

When you have your local and remote repository, let's start making some changes!. Create a file in your local repository and add a text file called test.txt. Write into the created file, the "Hello world" text. Save the file, create a version with these changes, and upload the version to the remote repository. After doing this process, check the remote repository to guarantee that there is a new version of the project, containing the test.txt file containing the "Hello World" text.

Take the time to dig through official documentation that we will be offering throughout this course to get the most out of your practice.

https://www.atlassian.com/git/tutorials/git-ssh https://www.atlassian.com/git/tutorials/export-git-archive

Git Commands

By now you should be getting the big picture of Git's main purpose: collaborate and organize coding processes.

How can I create, move and interact between branches and commits? Let's take a look at the following video:

https://youtu.be/0lCmQ0ZCa74

Git Commands Practice.

For this practice, we are going to perform different actions between branches. Enter the following link **https://learngitbranching.js.org/?locale=en_US** and **complete all the levels from the "Main" section** (it might be labeled Principal depending on the language). The levels to complete are all available in the following categories:

1. Introduction Sequence
2. Ramping Up

3. Moving Work Around
4. A Mixed Bag
5. Advanced Topics

Git cheat sheet | Atlassian Git Tutorial

Git Advanced

Perfect! Now you know how to manage branches on Git. Please discuss with your teammates about how you managed to solve these exercises, and take a moment to troubleshoot any difficulties you had in the process.

Now it's time to take a look at other commands that are very useful when collaborating with others on your coding project.

https://youtu.be/AaLX7KqZNcA

Advanced Git Practice.

Part I

For this practice, we are going to perform advanced actions between branches. Enter the following link: **https://learngitbranching.js.org/?locale=en_US** and **complete all the levels from the "Remote" section.** You can find the levels you need to solve in the following categories:

1. Push & Pull -- Git Remotes!
2. To Origin And Beyond -- Advanced Git Remotes!

Part II

Now, we are going to practice the advanced git concepts on a real repository.

1. Start by creating a new public repository on github.com adding only a README file by default. Make sure to clone the repository on your local workspace.

2. On your local repository, create a git ignore file, to exclude IDE files and compiled code. On **a new branch called base_calculator,** created from the main branch; create a Java project for a Calculator and **create a class called Calculator in the folder src and commit.**

Now we are going to work with changes on multiple branches. Create a new branch from base_calculator called advanced_calculator. Then create the following methods on the respective branches:

3. On base_calculator add a method for sum and subtract and commit

4. On advanced_calculator add a method for multiplication and division and commit

We are now ready to merge the branches on the remote repository, so make sure to push all your updated branches into the remote. Start by **merging advanced_calculator into base_calculator using a pull request**, solve any conflicts if present. After this, **merge the updated base_calculator branch into the main branch via pull request and check:**

1. No files from IDE or compiled code should be on the main branch
2. The java project for the calculator, including all the created methods (sum, subtract, multiplication and division) should be in the main branch.

---

https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts https://www.atlassian.com/git/tutorials/using-branches/merge-strategy

CI Concepts

Great! You've blazed past Git, and we are sure you will be using it throughout the remaining of your tester career.

Now we'll focus con Continuous Integration and Deployment. Take a look at the key concepts of this subject in the following video:

https://youtu.be/cwtFypZAOrg

Practice

By learning the basics about CI/CD you start gathering tools to answer the following question: **How may I execute specific actions every time a new code is integrated to the pre-existent one?**

Let's try out what you've seen so far:

Start by reading the following document about CI / CD: **https://medium.com/globant/basics-of-cicd-7a5aed1eda5a** and the guide to install Jenkins **https://www.jenkins.io/doc/pipeline/tour/getting-started/**

In this practice, we are going to develop a simple Continuous Integration workflow and use Jenkins, so make sure to install its latest version on your local workspace.

Now we need to create a coding project.

1. Start by creating a new repository on GitHub and develop on its main branch a simple java app which prints the message "A commit has been made."

2. Make sure to also add a README.md file that provides a brief explanation of the application and the steps required to run it locally.

Now that your repository is done, we are ready to implement CI into it.

Create a new project in Jenkins, link it with your created repository, and configure the project to perform the following actions each time a commit is made to the remote repository main branch:

1. Clone the repository and fetch the latest version of the source code.
2. Compile the application.
3. Execute the application.

Now we can test our CI Project!

1. On the main branch, create an additional class with any name and any attributes.

2. Commit and push your changes.

3. Check if the Jenkins action is triggered and prints “A commit has been made” message on the console output.

Delivery and Deployment

Great! Now that you've practiced with CI, it's time to learn about delivery and deployment.

This will grant you the information you need to answer this question: **How can I define a flow to automatically test an application after its deployment on a certain environment?**

Check out how it's done:

https://youtu.be/69YW4jp9LJs

Hands on! Get ready for your next practice.

Let’s start by reading **the following document** about CD.

The following exercise involves designing a continuous delivery flow for a basic web application. **You need to write a step-by-step description of how you would carry out the process,** including the main stages and tools you would use.

1. **Project Description:** Imagine you have a web application for e-commerce that allows users to purchase products. Briefly describe the main components of the application, such as the user interface, the backend server, and the database.
2. **Version Control:** Explain how you would use a version control system, such as Git, to manage the application's source code. Describe how you would create a repository, work with branches, and merge changes.
3. **Automated Testing:** Mention how you would establish automated tests to ensure the quality of the application. You can mention tools like Selenium for UI testing and unit testing frameworks like JUnit.

4. **Build and Packaging:** Explain how you would configure a build system, such as Maven or Gradle, to compile the source code, manage dependencies, and generate an executable package of the application.
5. **Automated Deployment:** Describe how you would automate the process of deploying the application to a production environment. You can mention tools like Jenkins, Travis CI, or GitLab CI/CD to orchestrate the stages of build, testing, and deployment.
6. **Monitoring and Feedback:** Explain how you would set up monitoring and logging tools to gather insights about the application's performance and behavior in production.

Complimentary material

https://codefresh.io/learn/ci-cd/

Jenkins Concepts

Let's recap: you have installed Jenkins and taken the first few steps on CI and CD. But, what else can you achieve with Jenkins?

Find this out in the next video:

https://youtu.be/yLRQvMqlXHo

Practice

Your job is to **create a scheduled job on Jenkins** to print a sequence of numbers, printing one number each minute.

Example:

After 1 minute, print: 1

After 2 minutes, print: 2

After 3 minutes, print: 3

To achieve this, use the following guide about scheduling jobs: **https://qaautomation.expert/2022/12/05/how-to-schedule-a-jenkins-job/**

You may also check Jenkins' official documentation: https://www.jenkins.io/doc/book/getting-started/

Jenkins Automation Integration

We are getting closer to the good part! Now it's time to learn how Jenkins can help in your automation practices.

Get a head start by watching this clip:

https://youtu.be/_pDB9_-CSgw

Practice

The goal now is to implement a solution using Jenkins to test the code we are developing.

1. For this specific purpose, create a java application for a calculator, containing methods that add, subtract, divide and multiply two parameterized numbers. Make sure to have this application uploaded to a GitHub repository.

Now we need to create a testing module for our application.

2. Create a module for unit testing that contains functions to test all the calculator methods, and remember to always upload your changes to the remote repository.

At this point, we are ready to create a Jenkins project to test our software.

3. Follow this tutorial **https://www.browserstack.com/guide/jenkins-for-test-automation** to implement the application testing on a new Jenkins job. Configure the job to execute the unit tests for add, subtract, divide and multiply functions.

Lastly, check if the job is working correctly.

4. Manually execute the job and check if the execution was successful and the outputs are what were expected. Finish by modifying the add function to subtract, and check if the add unit tests are failing because of the non-working sum function.

Additional resources

https://www.jenkins.io/doc/developer/testing/

Pipeline Creation and Set Up

What are pipelines? Why should we use them? Are they necessary?

Take a look at the answers to these questions on the following video:

https://youtu.be/f9D42M9qTQg

Practice

In this practice, we'll build a simple Jenkins pipeline to execute tasks in parallel.

Let's start by creating a Java application on a GitHub repository that executes de following processes 15 times:

1. Print the message "Hello, I am <NAME> and I am printing the number <X>"
    1. NAME must be a parameter required to run the application
    2. X is the index of the iteration
2. Wait a random time (between 1 and 5 seconds) to repeat the process.

If your code is working correctly, the printed text on the console should be the following (assuming "John" is sent as the name parameter):

1. Hello, I am John and I am printing the number 1
2. (random time between 1 and 5 seconds passes)
3. Hello, I am John and I am printing the number 2
4. (random time between 1 and 5 seconds passes)
5. Hello, I am John and I am printing the number 3
6. (random time between 1 and 5 seconds passes)
7. Hello, I am John and I am printing the number 4
8. (repeats until counting to 15)

Now it's time to configure our local Jenkins to have 10 different agents. Once this is completed, we need to create a pipeline with the following characteristics:

1. The pipeline must have 5 different parameters, which will be the different names on the java application.
2. The pipeline must execute in parallel, 5 scripts on a different agent each
3. Each script on their respective agent should clone the newly created repository and its application.
4. Each script should execute the created application on its respective agent, sending the respective Jenkins name parameter to the application. (The first Jenkins parameter for the first agent script, the second Jenkins parameter for the second agent script, and so on).

After configuring the pipeline, we need to execute it, setting the 5 different name parameters, and check:

1. 5 different agents should be occupied on the executions.
2. On the pipeline's console, you should see different messages of people counting at different rhythms.
3. After the 5 agents have counted to 15, the execution should be successfully finished.

Need a hand? Browse the following resources to get unstuck!

Continuous Integration with Jenkins

https://www.jenkins.io/pipeline/getting-started-pipelines/

https://www.jenkins.io/doc/pipeline/tour/hello-world/