Exercise 1 : Write a procedure to Deploy Version Control System / Source Code Management, install git and create a GitHub account.

Aim:

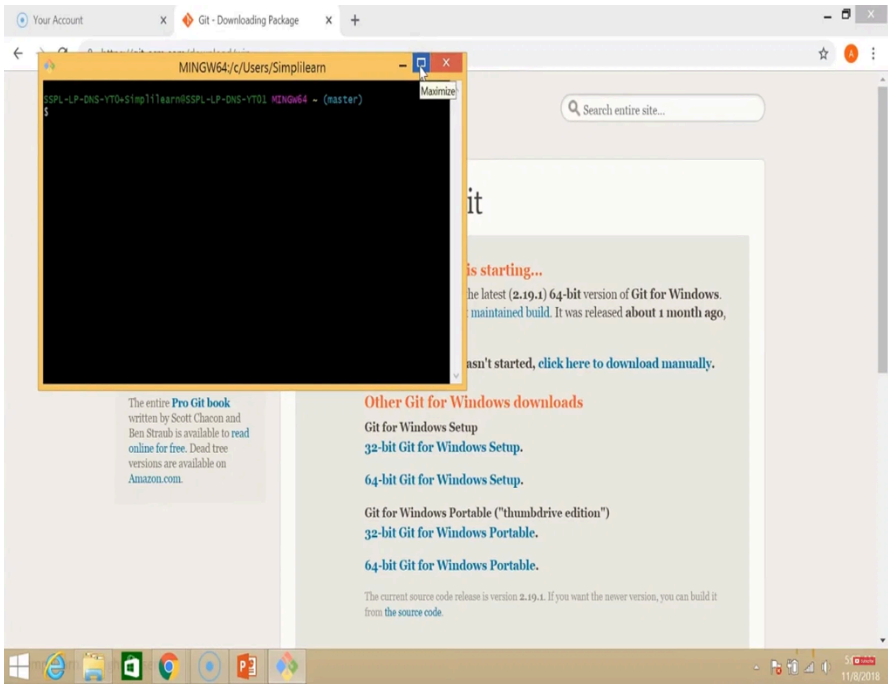
Procedure:

Steps For Git Installation on Windows

1. Download and install Git
2. Git bash interface
3. Basic Git commands
4. Create a local repository
5. Connect to the remote repository
6. Push the file to GitHub

Step 1:

Download the latest version of Git and choose the 64/32 bit version. After the file is downloaded, install it in the system. Once installed, select Launch the Git Bash, then click on finish. The Git Bash is now launched.



```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~ (master)
$
```

it

is starting...

he latest (2.19.1) 64-bit version of Git for Windows. maintained build. It was released about 1 month ago,

asn't started, click here to download manually.

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Other Git for Windows downloads

Git for Windows Setup
32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Git for Windows Portable ("thumbdrive edition")
32-bit Git for Windows Portable.

64-bit Git for Windows Portable.

The current source code release is version 2.19.1. If you want the newer version, you can build it from the source code.

Step 2:

Check the Git version:

```
$ git --version
```

Step 3:

For any help, use the following command:

```
$ git help config
```

This command will lead you to a browser of config commands. Basically, the help the command provides a manual from the help page for the command just following it (here, it's config).

Another way to use the same command is as follows:

```
$ git config --help
```

Step 4:

Create a local directory using the following command:

```
$ mkdir test

$ cd test
```

Step 5:

The next step is to initialize the directory:

```
$ git init
```

Step 6:

Go to the folder where “test” is created and create a text document named “demo.” Open “demo” and put any content, like “Hello MCA.” Save and close the file.

Step 7:

Enter the Git bash interface and type in the following command to check the status:

```
$ git status
```

Step 8:

Add the “demo” to the current directory using the following command:

```
$ git add demo.txt
```

Step 9:

Next, make a commit using the following command:

```
$ git commit -m “committing a text file”
```

Step 10:

Link the Git to a Github Account:

```
$ git config --global user.username
```

Note: simplilearn-github is the username on the Github account.

Step 11:

Open your Github account and create a new repository with the name "test_demo" and click on "Create repository." This is the remote repository. Next, copy the link of "test_demo."

Step 12:

Go back to Git bash and link the remote and local repository using the following command:

```
$ git remote add origin <link>
```

Here, <link> is the link copied in the previous step.

Step 13:

Push the local file onto the remote repository using the following command:

```
$ git push origin master
```

Step 14:

Move back to Github and click on "test_demo" and check if the local file "demo.txt" is pushed to this repository.

1. There are some experimental options available such as pseudo control Support or Built in file system monitor concerning your installed Git version.

How to Launch Git in Windows?

There are two methods to launch git in windows. One is launching git using a bash scripting shell with the help of the command line and another is launching git using a graphical user interface.

1. To launch git via bash scripting shell,
First, open the window and search for git bash and open it.

2. To launch git via graphical user interface(GUI), similarly, first open the window and search for git GUI and click on the application icon and open it.

Configure GitHub Credentials

You can configure your local GitHub installation with credentials by using the following commands. Also, don't forget to add your own GitHub credentials for username and email address.

1. git config –global user.n
ame "github_username"

2. git config –global user.e
mail "email_address"

Clone a GitHub Repository

1. Initially you need to click the options repository on GitHub.

2. Then in the top right corner, click the option clone or download where a small drop-down box will appear having a URL for cloning over HTTPS.

3. Then enter into your Powershell windows and write clone URL as:
git clone repository_url

4. On the other hand, you can clone a github repository with SSH URLs where first you need to generate an SSH key pair on your windows workstation as well as need to assign a public key to your GitHub account.

List Remote Repositories

1. Make a copy of the repository from GitHub for your working directory.

2. Ensure that the working directory should have the project name as
"cd git_project" and replace the project name from the downloaded repository.

3. If the above option doesn't work, you can list the content using "ls command" for the current directory, especially to check your exact number of spellings.

4. Besides, you can list the remote repository in the sub-directory as "git remote -v".

Output:

—--

Exercise 2: Write a procedure to perform various GIT operations on local and Remote repositories using GIT Cheat-Sheet

Aim:

Procedure:

1. Git configuration

- **Git config**

Get and set configuration variables that control all facets of how Git looks and operates.

Set the name:

```
$ git config --global user.name "User name"
```

Set the email:

```
$ git config --global user.email "gitcheatsheet@gmail.com"
```

Set the default editor:

```
$ git config --global core.editor Vim
```

Check the setting:

```
$ git config -list
```

- **Git alias**

Set up an alias for each command:

```
$ git config --global alias.co checkout
```

```
$ git config --global alias.br branch
```

```
$ git config --global alias.ci commit
```

```
$ git config --global alias.st status
```

2. Starting a project

- **Git init**

Create a local repository:

```
$ git init
```

- **Git clone**

Make a local copy of the server repository.

```
$ git clone
```

3. Local changes

- **Git add**

Add a file to staging (Index) area:

```
$ git add Filename
```

Add all files of a repo to staging (Index) area:

$ git add*

- **Git commit**

Record or snapshots the file permanently in the version history **with a message**.

$ git commit -m " Commit Message"

4. Track changes

- **Git diff**

Track the changes that have not been staged: $ git diff

Track the changes that have staged but not committed:

$ git diff --staged

Track the changes after committing a file:

$ git diff HEAD

Track the changes between two commits:

$ git diff Git Diff Branches:

$ git diff < branch 2>

- **Git status**

Display the state of the working directory and the staging area.

$ git status

- **Git show Shows objects:**

$ git show

5. Commit History

- **Git log**

Display the most recent commits and the status of the head:

$ git log

Display the output as one commit per line:

$ git log -oneline

Displays the files that have been modified:

$ git log -stat

Display the modified files with location:

$ git log -p

- **Git blame**

Display the modification on each line of a file:

$ git blame <file name>

6. Ignoring files

- **.gitignore**

Specify intentionally untracked files that Git should ignore. Create .gitignore:

$ touch .gitignore List the ignored files:

$ git ls-files -i --exclude-standard

7. Branching

- **Git branch Create branch:**

$ git branch List Branch:

$ git branch --list Delete a Branch:

$ git branch -d Delete a remote Branch:

$ git push origin -delete Rename Branch:

$ git branch -m

- **Git checkout**

Switch between branches in a repository.

Switch to a particular branch:

$ git checkout

Create a new branch and switch to it:

$ git checkout -b Checkout a Remote branch:

$ git checkout

- **Git stash**

Switch branches without committing the current branch. Stash current work:

$ git stash

Saving stashes with a message:

$ git stash save ""

Check the stored stashes:

$ git stash list

Re-apply the changes that you just stashed:

$ git stash apply

Track the stashes and their changes:

$ git stash show

Re-apply the previous commits:

$ git stash pop

Delete a most recent stash from the queue:

$ git stash drop

Delete all the available stashes at once:

$ git stash clear

Stash work on a separate branch:

$ git stash branch

- **Git cherry pic**

Apply the changes introduced by some existing commit:

$ git cherry-pick

8. Merging

- **Git merge**

Merge the branches:

$ git merge

Merge the specified commit to currently active branch:

$ git merge

- **Git rebase**

Apply a sequence of commits from distinct branches into a final commit.

$ git rebase

Continue the rebasing process:

$ git rebase -continue Abort the rebasing process:

$ git rebase --skip

- **Git interactive rebase**

Allow various operations like edit, rewrite, reorder, and more on existing commits.

$ git rebase -i

9. Remote

- **Git remote**

Check the configuration of the remote server:

$ git remote -v

Add a remote for the repository:

$ git remote add Fetch the data from the remote server:

$ git fetch

Remove a remote connection from the repository:

$ git remote rm

Rename remote server:

$ git remote rename

Show additional information about a particular remote:

$ git remote show

Change remote:

$ git remote set-url

- **Git origin master**

Push data to the remote server:

$ git push origin master Pull data from remote server:

$ git pull origin master

10. Pushing Updates

- **Git push**

Transfer the commits from your local repository to a remote server. Push data to the remote server:

$ git push origin master Force push data:

$ git push -f

Delete a remote branch by push command:

$ git push origin -delete edited

11. Pulling updates

- **Git pull**

Pull the data from the server:

$ git pull origin master

Pull a remote branch:

$ git pull

- **Git fetch**

Download branches and tags from one or more repositories. Fetch the remote repository:

$ git fetch< repository Url> Fetch a specific branch:

$ git fetch

Fetch all the branches simultaneously:

$ git fetch -all

Synchronize the local repository:

$ git fetch origin

12. Undo changes

- **Git revert**

Undo the changes:

$ git revert

Revert a particular commit:

$ git revert

- **Git reset**

Reset the changes:

$ git reset -hard

$ git reset -soft:

$ git reset --mixed

13. Removing files

- **Git rm**

Remove the files from the working tree and from the index:

$ git rm <file Name>

Remove files from the Git But keep the files in your local repository:

$ git rm –cached

Output:

—--

Exercise 3

Continuous Integration: install and configure Jenkins with Maven/Ant/Gradle to setup a build Job.

Aim:

Procedure:

1. Prerequisite

2. Implementation Steps

1. *Create a project for running the tests using Selenium WebDriver and TestNG*
2. *Create the Test Code*
3. *Start the Jenkins server*
4. *Log in to Jenkins UI*
5. *Download and Install Maven Plugin*
6. *Add the Maven Integration plugin*
7. *Restart Jenkins*
8. *Create a new project using the Maven project plugin*
9. *Build Management*
10. *Execute the tests*

3. Implementation Steps

Step 1: Create a project for running the tests using *Selenium WebDriver* and *TestNG*

Step 2: Create the Test Code

You can refer to this tutorial to get the test code – *Integration Of Jenkins With Selenium WebDriver***.**

Step 3: Start the Jenkins server

Open the browser and navigate to the localhost and the port in which Jenkins is running. http://localhost:8080/

Step 4: Log in to Jenkins UI

Provide username and password and click on ***Sign in.***

Step 5: Download and Install Maven Plugin

Click on the ***Manage Jenkins.***

Choose ***Manage Plugins.***

Step 6: Add the Maven Integration plugin

On the Plugins Page, go to the ***Available option***
1. Select the Maven Integration Plugin
2. Click on ***Install without restart.*** The plugin will take a few moments to finish downloading depending on your internet connection, and will be installed automatically.
3. You can also select the option ***Download now and Install after the restart*** button. In which plugin is installed after the restart
4. You will be shown a "***No updates available***" message if you already have the Maven plugin installed.

The plugin "***Maven Integration***" has been installed successfully.

Step 7: Restart Jenkins

Click on the checkbox "***Restart Jenkins when installation is complete when no jobs are running***".

The Jenkins is being restarted, It is about to restart. Again, log in to Jenkins UI.

Step 8: Create a new project using the Maven project plugin

1. *Give the Name of the project.*
2. *Click on the **Maven project.***
3. *Click on the **OK** button.*

In the **General** section, enter the project description in the Description box.

Step 9: Build Management

Go to the **Build** section of the new job.
§ In the **Root POM** textbox, enter the ***full path to pom.xml***
§ In the Goals and options section, enter "***clean test***"
Click on **Apply** and Save buttons.

We have created a new Maven project **"*SeleniumTestNG_MavenDemo*"** with the configuration to run the Selenium with TestNG Tests

Step 10: Execute the tests

Click on the ***Build Now*** link. Maven will build the project. It will then have TestNG execute the test cases.
To see the current status of the execution, click on the "***console output***".

OUTPUT:

—---

Exercise 4:

Build the pipeline of jobs using Maven / Gradle / Ant in Jenkins, create a pipeline script to Test and deploy an application over the tomcat server.

AIM:

Procedure:

- **Tomcat Configuration**
  - o Environment Configuration
  - o Update Roles and User credentials - Configuring Tomcat Security

- **Jenkins Configuration**
  - o Step1: Make Sure you have Git and Maven installed
  - o Step2: Install Deploy to Container Plugin.
  - o Step3: Create and Configure a Maven Job with Source Code Management (Github)
  - o Step4: Configure the Post-build Action and Specify the Tomcat Server Details
  - o Build Jenkins Job
  - o Testing the Application

Tomcat Configuration

· We presume that you have downloaded and started the tomcat server with no issues. Now we will advance to the next level which is making Tomcat ready to receive deployments and enabling TEXT based and GUI based management interface.

· All the tomcat application servers come with the manager application by default. If you look at the webapps directory of your downloaded tomcat server. You can see it.

· At the first time, you may not be able to access it cause the manager application needs some security elements to be configured prior to being accessed.

Environment Configuration

· Let us taken my tomcat7 setup as an example and I will paste the steps what I did to make it working.

· I have installed my Tomcat7 in CENTOS server at /apps/tomcat/tomcat7 directory and this is my CATALINA_HOME Now you need to find your CATALINA_HOME before continuing with the next steps.

Update Roles and User credentials - Configuring Tomcat Security

· Got to $CATALINA_HOME/conf directory and look for tomcat-users.xml file.

· Open the file in your favourite editor like VI or nano and add the following lines right before the last line

```
<user username="tomcatmanager" password="password" roles="manager-gui"/>

<user username="deployer" password="password" roles="manager-script"/>
```

· Here we are creating two usernames named tomcatmanager and deployer. here the deployer account would be used to deploy the WAR file over http.

· manager-gui based tomcatmanager user would be used to manage the manager web application at http://<HostName>:8080/manager

· Note*: If you are using Tomcat8+ version, Steps to enable manager application might be different. Refer the product version specific documentation if you get stuck.

· Now we are ready with the Tomcat Servlet Container aka Application server and it is ready to be connected from Jenkins.

Jenkins Configuration

· I presume that you have a Running Jenkins Server and a Administration Access

Step1: Make Sure you have Git and Maven installed

· In Jenkins UI, Goto Manage Jenkins -> Global Tool Configuration Section of Jenkins

· Note*: If you are not able to see a Manage Jenkins Option, You need to consult with your Administrator. It might be due to insufficient privileges.

Step2: Install Deploy to Container Plugin.

· Manage Jenkins -> Manage Plugins -> Available -> Deploy to Container Plugin

Note:* For the Next step we have selected a Maven Job as our Choice. you can also create a Free Style Project and use Gradle or Ant as your build tool .

Step3: Create and Configure a Maven Job with Source Code Management (Github)

· New Item -> Maven Project

· In the Configuration Section, Under Source Code Management Fill your Github/BeanStalk/Gitlab Repository URL

· For testing you can use one of our Sample Application Named Tomcat Maven App from our Github public Repository.

· You can use this GITHUB Repository

· Click on Add button displayed near the Credentials drop-down and enter the username and password of your SCM Repo and Once it is saved. It would be available on the Dropdown for you to select.

· Once you have selected the valid Credentials to wait for few seconds. If you see any error message in RED colour which means your connection to the SCM repository is unsuccessful. Check the URL or the credentials and retry.

Step4: Configure the Post-build Action and Specify the Tomcat Server Details

· Drag to the bottom and Go to the Post-build Actions section

· Click on Add post-build action button

· On the available options click on the Deploy war/ear to container

· Fill the required parameters for the plugin. Use the following Screen Shot as the reference

· Choose the Context Path in which the application should be installed. It would rename the WAR file before deploying to the server and thereby the application context root would be changed.

· Tomcat URL http://[Tomcat Server Host]:[Primary http port]/

Build Jenkins Job

· Execute the Job you have created by clicking on the Build Now button

· Console Output after the Successful build.

· At the last line you can see that the WAR file has been generated and deployed on the remote server.in our case, http://192.168.0.107:8081/

Testing the Application

· As the deployment is completed and the Jenkins Job ran Successfully without issues. (Or So I presume)

· Let us test our application.

· In my case, the URL should be as follows

- http://192.168.0.107:8081/TomcatMavenApp
- That's all. You have successfully configured Tomcat with Jenkins Continuous Deployment .

OUTPUT:

—---

Exercise 5:

Implement Jenkins Master-Slave Architecture and scale your Jenkins standalone implementation by implementing slave nodes.

Jenkins Master and Slave Architecture

- The Jenkins master acts to schedule the jobs, assign slaves, and send builds to slaves to execute the jobs.
- It will also monitor the slave state (offline or online) and get back the build result responses from slaves and the display build results on the console output. The workload of building jobs is delegated to multiple **slaves**.

Steps to Configure Jenkins Master and Slave Nodes

- Click on **Manage Jenkins** in the left corner on the Jenkins dashboard.
- Scroll down, Click on **Manage Nodes and clouds**.

Select **New Node** and enter the name of the node in the Node Name field.

Select **Permanent Agent** and click the **OK** button. Initially, you will get only one option, "Permanent Agent." Once you have one or more slaves you will get the "Copy Existing Node" option.
In the above screen shot, Parallel_Agent_01 was Created and currently it is in offline mode.

Click on configure, Provide the details.

1. **Name** -Parallel_Agent_01,
2. **Number of executors-** 5
3. **Remote root directory**– We have to provide Jenkins path
4. **Labels**-Parallel_Agent

5. **Launch method**-Launch agent by connecting it to the master

Node Properties Tab:

· Check Environment variables

o Provide the Java path

· Check Tool Locations:

o Provide the Git path and click on save button.

Click on Go to security configuration screen and change it. It will redirect to **Configure Global Security -> Agents ->**click on Fixed radio button port: 49187 and click on save Button. Go back to Nodes settings.

· We can see the above screen,

1. Click on Launch button, it will download the launch agent in your system.

2. Jenkins- slave.exe file should copy in the Jenkins folder which you installed in your system.

3. Double Click on jenkins – slave.exe.
4. Run the launch agent, click on run button and it will show connected.
5. In below screen shot, we can see the connected popup, click on file menu, select the install as service and click yes button . Once it is done, refresh the page.

· In above screenshot, we can see the Build executors. One is master and other is

Parallel_agent_01

o In Master node, we can see number of executors as 2.

o In Parallel_agent_01, we can see number of executors as 5.

· Go to build job -> configure

· In General tab, check on Restrict where this project can be run.

· In Label Expression, we have to select node name where we need to execute the build job.

· We can create the more number nodes as well.

OUTPUT
—---