SLK

**Basic and advanced concepts of Git and GitHub.**

GIT & GitHub

**Contents**

[How to Install Git on Windows 4](#_Toc57881634)

[What is Git? 4](#_Toc57881635)

[Why Git? 4](#_Toc57881636)

[Staging & Commits 5](#_Toc57881637)

[**Create a Repository for a Blank (New) Project:** 6](#_Toc57881638)

[**To add files to the repository** 6](#_Toc57881639)

[**To list all the untracked files** 6](#_Toc57881640)

[**Create a Repository and Directory Together** 6](#_Toc57881641)

[**To create a file** 6](#_Toc57881642)

[**To add file to git** 6](#_Toc57881643)

[**To Add more Files** 6](#_Toc57881644)

[**To Removing Files** 6](#_Toc57881645)

[**To Undo Add** 6](#_Toc57881646)

[**To commit the changes** 6](#_Toc57881647)

[**Git commit -m** 6](#_Toc57881648)

[**Git -am option for already staged files** 6](#_Toc57881649)

[**Git Clone** 6](#_Toc57881650)

[**Git Clone Branch** 6](#_Toc57881651)

[**Git Stash** 6](#_Toc57881652)

[**Git Ignore** 6](#_Toc57881653)

[**To Initializing or Create a Repository** 6](#_Toc57881654)

[**Git Index** 6](#_Toc57881655)

[**Git Show Head** 6](#_Toc57881656)

[**Git Origin Master** 6](#_Toc57881657)

[**Git Remote** 6](#_Toc57881658)

[**Git Remote Add** 7](#_Toc57881659)

[**Fetching and Pulling Remote Branch** 7](#_Toc57881660)

[**Pushing to Remote Branch** 7](#_Toc57881661)

[**Git Remove Remote** 7](#_Toc57881662)

[**Git Remote Rename** 7](#_Toc57881663)

[**Git Show Remote** 7](#_Toc57881664)

[**Git Change Remote (Changing a Remote's URL)** 7](#_Toc57881665)

[Branching & Merging: 7](#_Toc57881666)

[Git Merge and Merge Conflict: 8](#_Toc57881667)

[Git Merge Conflict: 8](#_Toc57881668)

[Resolve Conflict: 8](#_Toc57881669)

[Git Rebase: 8](#_Toc57881670)

[Rebase Branch 8](#_Toc57881671)

[Git Interactive Rebase 8](#_Toc57881672)

[Git interactive rebase command 8](#_Toc57881673)

[Git Squash: 8](#_Toc57881674)

[Undoing changes 9](#_Toc57881675)

[Operations on Git Checkout 9](#_Toc57881676)

[1. Checkout Branch 9](#_Toc57881677)

[2. Create and Switch Branch 9](#_Toc57881678)

[3. Checkout Remote Branch 9](#_Toc57881679)

[Git Revert 9](#_Toc57881680)

[Git Revert to Previous Commit 9](#_Toc57881681)

[Git Revert Merge 9](#_Toc57881682)

[Git Reset 9](#_Toc57881683)

[Git Rm 9](#_Toc57881684)

[Git Rm Cached 9](#_Toc57881685)

[Undo the Git Rm Command 9](#_Toc57881686)

[Git Cherry-pick 9](#_Toc57881687)

[Inspecting changes 9](#_Toc57881688)

[Basic Git log 9](#_Toc57881689)

[Git Log Stat 10](#_Toc57881690)

[Git log P or Patch 10](#_Toc57881691)

[Git Log Graph 10](#_Toc57881692)

[Filtering the Commit History 10](#_Toc57881693)

[Git Diff 10](#_Toc57881694)

[Git Diff Branches 10](#_Toc57881695)

[Git Status 10](#_Toc57881696)

[Collaborating 10](#_Toc57881697)

[Git Pull / Pull Request 10](#_Toc57881698)

[Difference between pull and push 11](#_Toc57881699)

[Git Pull Remote Branch 11](#_Toc57881700)

[Git Force Pull 11](#_Toc57881701)

[Git Pull Origin Master 11](#_Toc57881702)

[Git Push 11](#_Toc57881703)

[Git Push Tags 11](#_Toc57881704)

[Git Push Origin Master: 11](#_Toc57881705)

[Git Force Push: 11](#_Toc57881706)

[How to Safe Force Push Repository: 11](#_Toc57881707)

[Delete a Remote Branch: 11](#_Toc57881708)

## How to Install Git on Windows

To use Git, you have to install it on your computer. Even if you have already installed Git, it's probably a good idea to upgrade it to the latest version. Click here to download

<https://git-scm.com/download/win>

## What is Git?

|  |
| --- |
| **Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.  Git is foundation of many services like **GitHub** and **GitLab**, but we can use Git without using any other Git services. Git can be used **privately** and **publicly**. |

## Why Git?

|  |
| --- |
| We have discussed many features and benefits of Git that demonstrate the undoubtedly Git as the leading version control system. Now, we will discuss some other points about why should we choose Git.  Why Git   * **Git Integrity** Git is **developed** to ensure the **security** and **integrity** of content being version controlled. It uses checksum during transit or tampering with the file system to confirm that information is not lost. Internally it creates a checksum value from the contents of the file and then verifies it when transmitting or storing data. * **Trendy Version Control System** Git is the most widely used version control system. It has maximum projects among all the version control systems. Due to its amazing workflow and features, it is a preferred choice of developers. * **Everything is Local** Almost All operations of Git can be performed locally; this is a significant reason for the use of Git. We will not have to ensure internet connectivity. * **Collaborate to Public Projects** There are many public projects available on the GitHub. We can collaborate on those projects and show our creativity to the world. Many developers are collaborating on public projects. The collaboration allows us to stand with experienced developers and learn a lot from them; thus, it takes our programming skills to the next level. * **Impress Recruiters** We can impress recruiters by mentioning the Git and GitHub on our resume. Send your GitHub profile link to the HR of the organization you want to join. Show your skills and influence them through your work. It increases the chances of getting hired. |

## Staging & Commits

|  |
| --- |
| **Create a Repository for a Blank (New) Project:**  1. $ git init  **To add files to the repository**  1. $ git add <file name>  **To list all the untracked files**  1. $ git status  **Create a Repository and Directory Together**  1. $ git init <NewDirectory name>  **To create a file**  1. $ touch newfile.txt  **To add file to git**  1. $ git add <file name>  **To Add more Files**  1. $ git add -A  **To Removing Files**  1. $ git rm <filename>  **To Undo Add**  1. $ git reset <filename>  **To commit the changes**  1. $ git commit  **Git commit -m**  1. $ git commit -m "Commit message"  **Git -am option for already staged files**  1. $ git commit -am "Commit message."  **Git Clone**  1. $ git clone <repository URL>  **Git Clone Branch**  1. $ git clone -b <Branch name><Repository URL>  **Git Stash** **git stash** when you want to record the current state of the working directory and the index, but want to go back to a clean working directory   1. $ git stash save "<Stashing Message>"  **Git Ignore**  1. $ git add .gitignore 2. $ git ls-files --ignore --exclude-standard  **To Initializing or Create a Repository**  1. $ git init <repository name> 2. $ git add <filename>  **Git Index**  1. $ git status  **Git Show Head**  1. $ git show HEAD  **Git Origin Master**  1. $ git clone <repository URL> 2. $ git push origin master 3. $ git pull origin master  **Git Remote**  1. $ git remote  **Git Remote Add**  1. $ git remote add <short name><remote URL>  **Fetching and Pulling Remote Branch**  1. $ git fetch <remoteURL> 2. $ git clone<remoteURL> 3. $ git pull <remoteURL>  **Pushing to Remote Branch**  1. $ git push <remote><branch> 2. $ git push origin master  **Git Remove Remote**  1. $ git remote rm <destination>  **Git Remote Rename**  1. $ git remote rename <old name><new name>  **Git Show Remote**  1. $ git remote show <remote>  **Git Change Remote (Changing a Remote's URL)** $ git remote set-url <remote name><new URL> |

## Branching & Merging:

|  |
| --- |
| **Git Master Branch**   1. $ git checkout -b <sample> 2. $ git branch <branchname>   **List Branch**   1. $ git branch --list   **Delete Branch**   1. $ git branch -d<branch name>   **Delete a Remote Branch**   1. $ git push origin -delete <branch name>   **Switch Branch**   1. $ git checkout<branch name>   **Switch from master Branch**   1. $ git checkout <branch name>   **Switch to master branch**   1. $ git branch -m master 2. $ git checkout -b master   **Rename Branch**   1. $ git branch -m <old branch name><new branch name>   **Merge Branch**  $ git merge <branch name> |

## Git Merge and Merge Conflict:

|  |
| --- |
| **Scenario1: To merge the specified commit to currently active branch:**   1. $ git merge commit   **Scenario2: To merge commits into the master branch:**   1. $ git log 2. $ git checkout master 3. $ git merge master   **Scenario 3: Git merge branch.**   1. $ git merge <branch name>  Git Merge Conflict: **Consider a example When two branches are trying to merge, and both are edited at the same time and in the same file and output is shown as below**  **[rejected] failed to push some refs to <remote URL>**  **error message like merge conflict in <filename>.** Resolve Conflict:  1. $ git mergetool 2. $ git rebase --continue  Git Rebase:  1. $ git rebase <branch name> 2. $ git status 3. $ git rebase --continue 4. $ git rebase --skip  Rebase Branch  1. $ git checkout <branch Name> 2. $ git rebase master  Git Interactive Rebase  1. $ git rebase -i  Git interactive rebase command **Pick (-p):**  Pick stands here that the commit is included. Order of the commits depends upon the order of the pick commands during rebase. If you do not want to add a commit, you have to delete the entire line.  **Reword (-r):**  The reword is quite similar to pick command. The reword option paused the rebase process and provides a chance to alter the commit message. It does not affect any changes made by the commit.  **Edit (-e):**  The edit option allows for amending the commit. The amending means, commits can be added or changed entirely. We can also make additional commits before rebase continue command. It allows us to split a large commit into the smaller commit; moreover, we can remove erroneous changes made in a commit.  **Squash (-s):**  The squash option allows you to combine two or more commits into a single commit. It also allows us to write a new commit message for describing the changes.  **Fixup (-f):**  It is quite similar to the squash command. It discarded the message of the commit to be merged. The older commit message is used to describe both changes.  **Exec (-x):**  The exec option allows you to run arbitrary shell commands against a commit.  **Break (-b):**  The break option stops the rebasing at just position. It will continue rebasing later with '**git rebase --continue**' command.  **Drop (-d):**  The drop option is used to remove the commit.  **Label (-l):**  The label option is used to mark the current head position with a name.  **Reset (-t):**  The reset option is used to reset head to a label. Git Squash: **Step1: Check the commit history**   1. $ git log --oneline   **Step 2: Choose the commits to squash.**   1. $ git rebase -i HEAD ~3   **Step 3: Push the squashed commit**   1. $ git push origin master |

## Undoing changes

|  |
| --- |
| Operations on Git Checkout**Checkout Branch** $ git branch  <branch name>  $ git checkout <branch name> **Create and Switch Branch** $ git checkout -b <branch name> **Checkout Remote Branch** $ git fetch –all  $ git checkout <remote branch> Git Revert $ git revert  $ git revert -e <commit-ish>  $ git revert -n <commit-ish> Git Revert to Previous Commit $ git log  $ git revert < recent commit-ish> Git Revert Merge $ git log  $ git revert <commit reference> -m 1 Git Reset $ git reset --hard  $ git reset Git Rm $ git rm <filename>  $ git commit -m "commit message" Git Rm Cached $ git rm --cached <filename> Undo the Git Rm Command $ git reset HEAD Git Cherry-pick **Scenerio1: Accidently make a commit in a wrong branch.**  $ git cherry-pick <commit id> |

## Inspecting changes

|  |
| --- |
| Basic Git log  1. $ git log 2. $ git log –oneline  Git Log Stat  1. $ git log --stat  Git log P or Patch  1. $ git log --patch  Git Log Graph  1. $ git log –graph 2. $ git log --graph –oneline  Filtering the Commit History  1. $ git log --after="yy-mm-dd" 2. git log --after="21 days ago"  Git Diff **Scenerio1: Track the changes that have not been staged.**   1. $ git diff   **Scenerio2: Track the changes that have staged but not committed:**   1. $ git status 2. $ git add < filename> 3. $ git diff --staged   **Scenerio3: Track the changes after committing a file:**   1. $ git diff HEAD 2. $ git diff <commit1-sha> <commit2-sha>   **Scenario4: Track the changes between two commits:**   1. $ git log 2. $ git log -p --follow -- filename  Git Diff Branches  1. $ git diff <branch 1> < branch 2>  Git Status **Status when Working Tree is cleaned**  $ git status  **Status when an existing file is modified**  $ echo "Text"> < filename>  **Status when a file is deleted**  $ git rm < filename> |

## Collaborating

|  |
| --- |
| **Scenario 1: To fetch the remote repository:**  $ git fetch<repository Url>  **Scenario 2: To fetch a specific branch:**  $ git fetch <branch URL><branchname>  **Scenario 3: To fetch all the branches simultaneously:**  $ git fetch -all  **Scenario 4: To synchronize the local repository:**  $ git fetch origin Git Pull / Pull Request  1. $ git pull <option> [<repository URL><refspec>...]  Difference between pull and push  * git fetch: Get the latest changes from origin (no merge) * git pull = git fetch + git merge  Git Pull Remote Branch  1. $ git pull <remote branch URL>  Git Force Pull  1. $ git fetch -all 2. $ git reset -hard <remote>/<branchname> 3. $ git reset-hard master  Git Pull Origin Master  1. $ git pull <options><remote>/<branchname> 2. $ git pull origin master  Git Push  1. $ git push <option> [<remoteURL><branchname><refspec>...]  Git Push Tags **<repository>:** The repository is the destination of a push operation. It can be either a URL or the name of a remote repository.  **<refspec>:** It specifies the destination ref to update source object.  **--all:** The word "all" stands for all branches. It pushes all branches.  **--prune:** It removes the remote branches that do not have a local counterpart. Means, if you have a remote branch say demo, if this branch does not exist locally, then it will be removed.  **--mirror:** It is used to mirror the repository to the remote. Updated or Newly created local refs will be pushed to the remote end. It can be force updated on the remote end. The deleted refs will be removed from the remote end.  **--dry-run:** Dry run tests the commands. It does all this except originally update the repository.  **--tags:** It pushes all local tags.  **--delete:** It deletes the specified branch.  **-u:** It creates an upstream tracking connection. It is very useful if you are going to push the branch for the first time. Git Push Origin Master:  * 1. $ git push origin master   2. $ git status   3. $ git add <filename>   4. $ git commit -m "added a new image to project."   5. $ git push origin master  Git Force Push:  1. $ git push <remote><branch> -f 2. $git push <remote> -f 3. $ git push -f  How to Safe Force Push Repository: $ git push <remote><branch> --force-with-lease Delete a Remote Branch: $ git push origin -delete edited |

## Installation of Plugins in Jenkins

Jenkins comes with a pretty basic setup, so you will need to install the required plugins to enable respective third-party application support.

GitHub is a web-based repository of code which plays a major role in DevOps. It provides a common platform for multiple developers working on the same code/project to upload and retrieve updated code, thereby facilitating continuous integration.

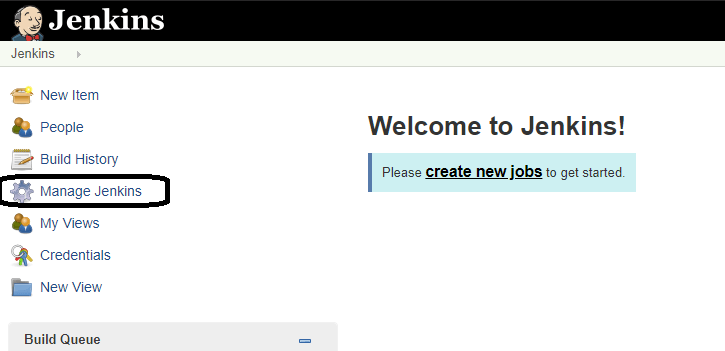
Jenkins needs to have GitHub plugin installed to be able to pull code from the GitHub repository.

You need not install a GitHub plugin if you have already installed the Git plugin in response to the prompt during the Jenkins' installation setup. But if not, here is how you install GitHub plugins in [Jenkins](https://www.guru99.com/jenkin-continuous-integration.html) and pull code from a GitHub repository.

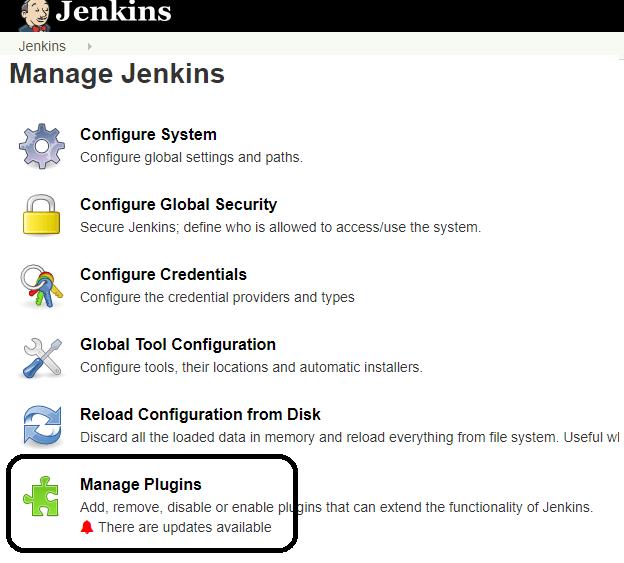
## How to Install Git Plugin in Jenkins

Following is a step by step process on how to Install Git plugin in Jenkins:

**Step 1:**Click on the **Manage Jenkins** button on your Jenkins dashboard:

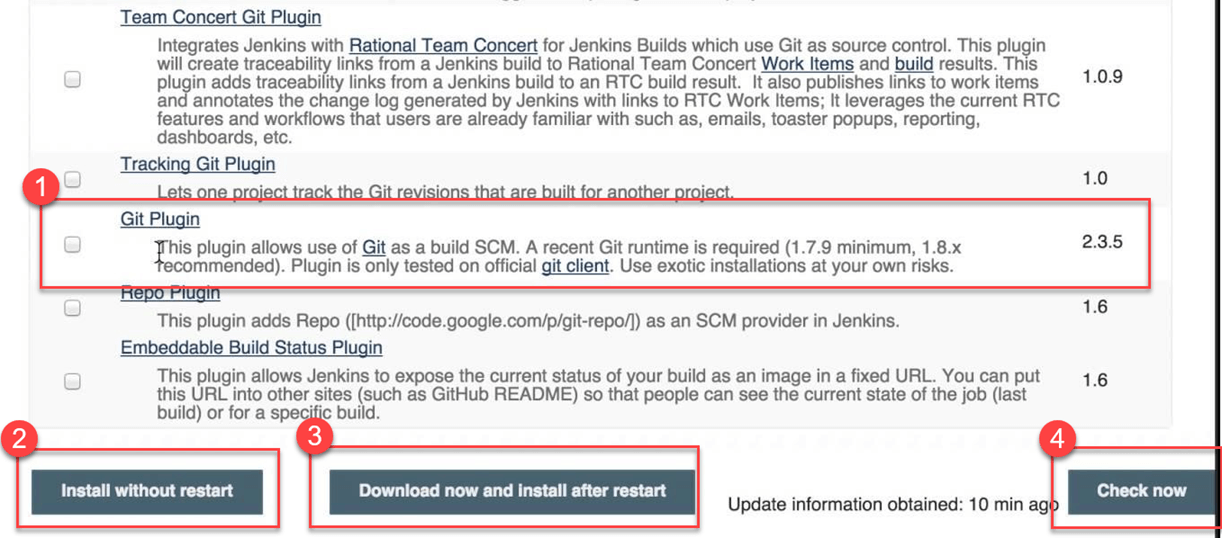
[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH2.png)

**Step 2:**Click on **Manage Plugins**:

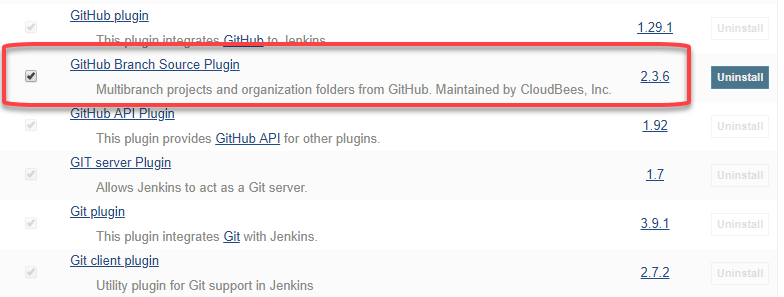
[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH3.png)

**Step 3:**In the Plugins Page

1. Select the GIT 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 restart**button. In which plugin is installed after restart
4. You will be shown a "No updates available" message if you already have the Git plugin installed.

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH4.png)

**Step 4:** Once the plugins have been installed, go to **Manage Jenkins**on your Jenkins dashboard. You will see your plugins listed among the rest.

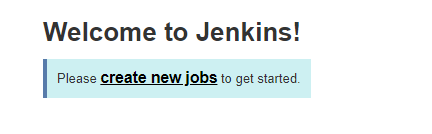
[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH5.png)

## How to Integrate Jenkins With GitHub

We shall now discuss the process of integrating Jenkins and GitHub a Windows system:

**Step 1)**Create a new job in Jenkins, open the Jenkins dashboard with your Jenkins URL. For example, <http://localhost:8080/>

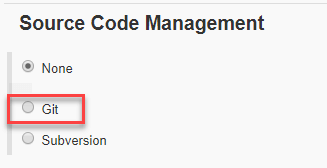
Click on **create new jobs**:

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH6.png)

**Step 2)**Enter the item name, select job type and click **OK**. We shall create a Freestyle project as an example.

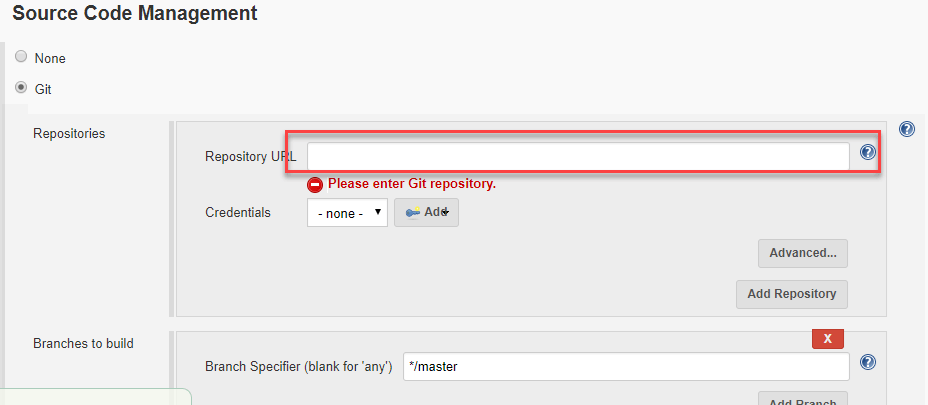
**Step 3)**Once you click **OK,**the page will be redirected to its project form. Here you will need to enter the project information:

**Step 4)**You will see a **Git**option under **Source Code Management**if your Git plugin has been installed in Jenkins:

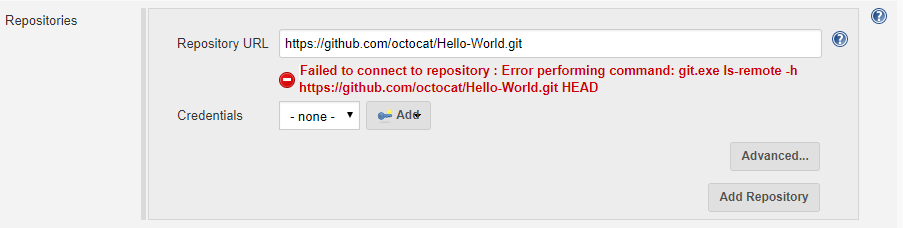
[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH9.png)

NOTE: If the **Git**option does not appear, try re-installing the plugins, followed by a restart and a re-login into your Jenkins dashboard.You will now be able to see the **Git**option as mentioned above.

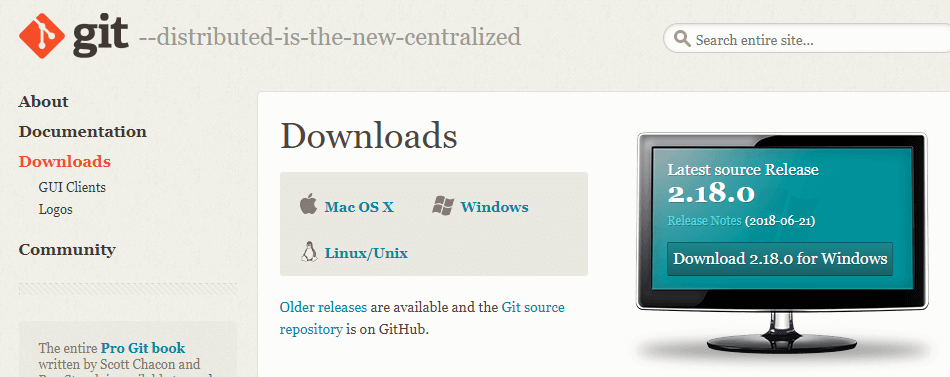
**Step 5)**Enter the Git repository URL to pull the code from GitHub.

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH10.png)

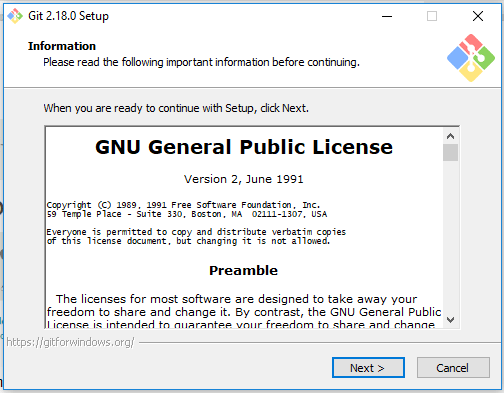
**Step 6)**You might get an error message the first time you enter the repository URL. For example:

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH11.png)

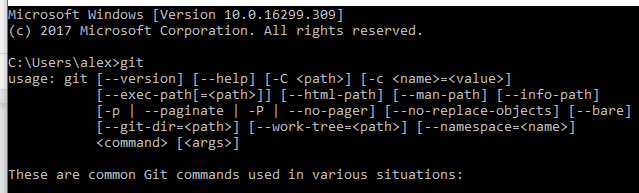
This happens if you do not have Gitinstalled in your local machine. To install Git in your local machine, go to <https://git-scm.com/downloads>

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH12.png)

Download the appropriate Git file for your Operating System, in this case, Windows, and install it onto your local machine running Jenkins. Complete the onscreen instructions to install GIT.

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH13.png)

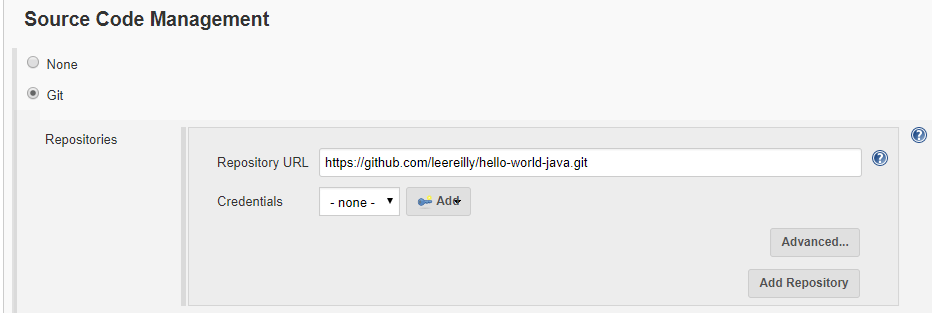
**Step 7)**You can execute Git repositories in your Jenkins once Git has been installed on your machine. To check ifithas been successfully installed onto your system, open your **command prompt,** type "Git"and press enter. You should see different options come up for Git:

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH14.png)

This means that Git has been installed in your system.

Note: If you have GIT already installed in your system, just add git.exe path in Global Tool Configuration.

**Step 8)**Once you have everything in place, try adding the Git URL into Jenkins. You will not see any error messages for Jenkins Git integration:

[](https://www.guru99.com/images/1/091318_0440_JenkinsGitH15.png)

Git is now properly configured on your system.