



D Y PATIL

— RAMRAO ADIK
INSTITUTE OF
TECHNOLOGY —

NAVI MUMBAI

Department of Computer Engineering

Lab Manual

Second Year Semester-VI

Subject: DevOps

Even Semester

Institutional Vision and Mission

Our Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems faced by our country and world at large.

Our Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

Our Quality Policy

ज्ञानधीनं जगत् सर्वम् ।
Knowledge is supreme.

Our Quality Policy

It is our earnest endeavour to produce high quality engineering professionals who are innovative and inspiring, thought and action leaders, competent to solve problems faced by society, nation and world at large by striving towards very high standards in learning, teaching and training methodologies.

Our Motto: If it is not of quality, it is NOT RAIT!

Departmental Vision and Mission

Vision

To impart higher and quality education in Computer Science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

To mobilize the resources and equip the department with men and materials of excellence to provide knowledge in the thrust areas of Computer Science and Engineering. To provide learning ambiance and exposure to the latest tools and technologies and the platforms to work on research and live projects. To provide the diverse platforms of sports, technical, co-curricular and extracurricular activities for the overall development of student with ethical attitude. To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service. To collaborate with IITs, reputed universities and industries for the technical and overall upliftment of students for continuing learning and entrepreneurship.

Departmental Program Educational Objectives (PEOs)

1. Learn and Integrate

To provide Computer Engineering students with a strong foundation in the mathematical, scientific and engineering fundamentals necessary to formulate, solve and analyze engineering problems and to prepare them for graduate studies.

2. Think and Create

To develop an ability to analyze the requirements of the software and hardware, understand the technical specifications, create a model, design, implement and verify a computing system to meet specified requirements while considering real-world constraints to solve real world problems.

3. Broad Base

To provide broad education necessary to understand the science of computer engineering and the impact of it in a global and social context.

4. Techno-leader

To provide exposure to emerging cutting edge technologies, adequate training & opportunities to work as teams on multidisciplinary projects with effective communication skills and leadership qualities.

5. Practice citizenship

To provide knowledge of professional and ethical responsibility and to contribute to society through active engagement with professional societies, schools, civic organizations or other community activities.

6. Clarify Purpose and Perspective

To provide strong in-depth education through electives and to promote student awareness on the life-long learning to adapt to innovation and change, and to be successful in their professional work or graduate studies.

Departmental Program Outcomes (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3 : Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 : Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11 : Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 : Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes: PSO

PSO1: To build competencies towards problem solving with an ability to understand, identify, analyze and design the problem, implement and validate the solution including both hardware and software.

PSO2: To build appreciation and knowledge acquiring of current computer techniques with an ability to use skills and tools necessary for computing practice.

PSO3: To be able to match the industry requirements in the area of computer science and engineering. To equip skills to adopt and imbibe new technologies.

Index

Sr. No.	Contents	Page No.
1.	List of Experiments	8
2.	Course Objective, Course Outcome & Experiment Plan	9
3.	CO-PO,CO-PSO Mapping	11
4.	Study and Evaluation Scheme	14
5.	Experiment No. 1	15
6.	Experiment No. 2	
7.	Experiment No. 3	
8.	Experiment No. 4	
9.	Experiment No. 5	
10.	Experiment No. 6	
11.	Experiment No. 7	
12.	Experiment No. 8	
13.	Experiment No. 9	
14.	Experiment No. 10	

List of Experiments

Sr. No.	Experiments Name
1	Git installation and basic Git Commands Perform installation of Git Bash and Explore usage of basic Git Commands. a. Git installation b. Git configuration c. Starting A Project d. Day-To-Day Work e. Git branching model f. Review your work
2	Explore Git Commands to a. Tagging known commits b. Reverting changes. c. Synchronizing repositories. d. Reverting Changes
3	Installation of Jenkins To install and configure Jenkins to setup a build Job.
4	Continuous Integration To build the pipeline of jobs in Jenkins, create a pipeline script to Test.
5	Test Software Applications To Setup and Run Selenium Tests in Jenkins Using Maven
6	Containerization with Docker To understand Docker Architecture, install docker and execute docker commands to manage and interact with containers.
7	To learn Dockerfile instructions, build an image for sample web application on Docker Engine
8	To install and configure Pull based Software Configuration Management and provisioning tools using Ansible/Chef/Puppet
9	To perform Software Configuration management using Ansible/Chef/Puppet
10	Case Study Comparative study of Software Development Life Cycle – Waterfall Cycle vs Agile vs DevOps.

Course Objective, Course Outcome&Experiment Plan

Course Objective:

1	To understand DevOps practices which aims to simplify Software Development Life
2	To be aware of different Version Control tools like GIT, CVS or Mercurial
3	To Integrate and deploy tools like Jenkins and Maven, which is used to build, test and deploy applications in DevOps environment
4	To be familiarized with selenium tool, which is used for continuous testing of applications deployed.
5	To use Docker to Build, ship and manage applications using containerization
6	To understand the concept of Infrastructure as a code and install and configure Ansible tool.

Course Outcomes:

CO1	To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.
CO2	To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub.
CO3	To understand the importance of Jenkins to Build and deploy Software Applications on server environment.
CO4	Understand the importance of Selenium and Jenkins to test Software Applications.
CO5	To understand concept of containerization and Analyze the Containerization of OS images and deployment of applications over Docker.
CO6	To Synthesize software configuration and provisioning using Ansible.

Experiment Plan:

Module No.	Week No.	Experiments Name	Course Outcome	Weightage
1	W1	Git installation and basic Git Commands Perform installation of Git Bash and Explore usage of basic Git Commands. a. Git installation b. Git configuration c. Starting A Project d. Day-To-Day Work e. Git branching model f. Review your work	CO2	05
2	W2	Explore Git Commands to a. Tagging known commits b. Reverting changes. c. Synchronizing repositories. d. Reverting Changes	CO2	05
3	W3	Installation of Jenkins To install and configure Jenkins to setup a build Job.	CO3	05
4	W4, W5	Continuous Integration To build the pipeline of jobs in Jenkins, create a pipeline script to Test.	CO3	05
5	W6	Test Software Applications To Setup and Run Selenium Tests in Jenkins Using Maven	CO4	10
6	W7, W8	Containerization with Docker To understand Docker Architecture, install docker and execute docker commands to manage and interact with containers.	CO5	05
7	W9	To learn Dockerfile instructions, build an image for sample web application on Docker Engine	CO5	05
8	W10	To install and configure Pull based Software Configuration Management and provisioning tools using Ansible/Chef/Puppet	CO6	05
9	W11	To perform Software Configuration management using Ansible/Chef/Puppet	CO6	05
10	W12	Case Study Comparative study of Software Development Life Cycle – Waterfall Cycle vs Agile vs DevOps.	CO1	10

Mapping of COs with POs:

Subject Weight	Course Outcomes (Weightage-100%)		Program Outcomes											
			PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
PRACTICAL 100%	CO1	To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.	1	1	1	1	1				1		2	2
	CO2	To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub.	1	1	1		1	1			2	2		1
	CO3	To understand the importance of Jenkins to Build and deploy Software Applications on server environment.					2				2	2	2	2
	CO4	Understand the importance of Selenium and Jenkins to test Software Applications.					2				2	2	2	2
	CO5	To understand concept of containerization and Analyze the Containerization of OS images and deployment of applications over Docker.	1				2				2	1	2	2
	CO6	To Synthesize software configuration and provisioning using Ansible.					2				2	2	2	2

Mapping of Course outcomes with Program Specific Outcomes:

Course Outcomes		Contribution to Program Specific outcomes		
		PSO1	PSO2	PSO3
CO1	To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.	2	2	2
CO2	To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub.	2	2	2
CO3	To understand the importance of Jenkins to Build and deploy Software Applications on server environment.	2	2	2
CO4	Understand the importance of Selenium and Jenkins to test Software Applications.	2	2	2
CO5	To understand concept of containerization and Analyze the Containerization of OS images and deployment of applications over Docker.	2	2	2
CO6	To Synthesize software configuration and provisioning using Ansible.	2	2	2

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
CESL601	Skill Based Lab IV – DevOPs	Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
			02+02*	-	-	02	-	02

Course Code	Course Name	Examination Scheme		
CESL601	Skill Based Lab IV – DevOPs	Term Work	Practical & Oral	Total
		25	25	50

Term Work:

1. Term work should consist of at least 10 experiments on above content.
2. The final certification and acceptance of term work ensures that satisfactory performance of laboratory work and minimum passing marks in term work.
3. Term Work: 25 Marks (Total) = 10 Marks (Experiments)+ 5 Marks (Mini Project)+ 5 Marks (Assignments)+ 5 Marks (Theory + Practical Attendance).

Practical/Experiments:

1. The final certification and acceptance of term work ensures that satisfactory performance of laboratory work and minimum passing marks in term work.
2. Practical exam will be based on the above syllabus.

Skill Based Lab IV – DevOPs

Experiment No. : 1

**Git installation and basic Git
Commands**

Experiment No. 1

- **Aim:** Perform installation of Git Bash and explore usage of basic Git Commands.
 - a. Git installation
 - b. Git configuration
 - c. Starting A Project
 - d. Day-To-Day Work
 - e. Git branching model
 - f. Review your work
- **Objectives:** From this experiment, the student will be able
 - To understand DevOps practices which aims to simplify Software Development Life.
 - To be aware of different Version Control tools like GIT, CVS or Mercurial
- **Outcomes:** The learner will be able to
 - To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.
 - To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub.
- **Hardware / Software Required :** Linux / Windows Operating System
- **Theory:**

What is version control?

How version control helps high performing development and DevOps teams prosper

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the

frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.

Git is the best choice for most software teams today.

a. Git Install

You can download Git for free from the following website: <https://www.git-scm.com/>

Git for Windows stand-alone installer

Download the latest Git for Windows installer from <https://gitforwindows.org/>

When you've successfully started the installer, you should see the **Git Setup** wizard screen. Follow the **Next** and **Finish** prompts to complete the installation. The default options are pretty sensible for most users.

Using Git with Command Line

To start using Git, we are first going to open up our Command shell.

For Windows, you can use Git bash, which comes included in Git for Windows. For Mac and Linux you can use the built-in terminal.

The first thing we need to do, is to check if Git is properly installed:

Example

```
git --version
```

```
git version 2.30.2.windows.1
```

If Git is installed, it should show something like **git version X.Y**

b. Configure Git

Run the following commands to configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name "Emma Paris" $ git config --  
global user.email "eparis@atlassian.com"
```

Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

Example

```
git config --global user.name "pujagit"
```

```
git config --global user.email "padiya.puja@rait.ac.in"
```


Change the user name and e-mail address to your own. You will probably also want to use this when registering to GitHub later on.

Note: Use `global` to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove `global`

c. Creating Git Folder (Starting a project)

Now, let's create a new folder for our project:

Example

```
mkdirmyproject
```

```
cdmyproject
```

`mkdir` makes a new directory.

`cd` changes the current working directory.

Now that we are in the correct directory. We can start by initializing Git!

Note: If you already have a folder/directory you would like to use for Git:

Navigate to it in command line, or open it in your file explorer, right-click and select "Git Bash here"

Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

Example

```
gitinit
```

Initialized empty Git repository in /Users/user/myproject/.git/

You just created your first Git Repository!

Note: Git now knows that it should watch the folder you initiated it on.

Git creates a hidden folder to keep track of changes.

Note: Create a new local repository. If [project name] is provided, Git will create a new directory name [project name] and will initialize a repository inside it. If [project name] is not provided, then a new repository is initialized in the current directory.

```
$ git clone [project url]
```

Downloads a project with the entire history from the remote repository.

d. Day-To-Day Work

`$ git status`

Displays the status of your working directory. Options include new, staged, and modified files. It will retrieve branch name, current commit identifier, and changes pending commit.

`$ git add [file]`

Add a file to the staging area. Use in place of the full file path to add all changed files from the current directory down into the directory tree

`$ git diff [file]`

Show changes between working directory and staging area.

`$ git diff --staged [file]`

Shows any changes between the staging area and the repository.

`$ git checkout -- [file]`

Discard changes in working directory. This operation is unrecoverable.

`$ git reset [file]`

Revert your repository to a previous known working state.

`$ git commit`

Create a new commit from changes added to the staging area. The commit must have a message!

`$ git rm [file]`

Remove file from working directory and staging area.

e. Git branching model

`$ git branch [-a]`

List all local branches in repository. With -a: show all branches (with remote).

`$ git branch [branch_name]`

Create new branch, referencing the current HEAD.

`$ git checkout [-b][branch_name]`

Switch working directory to the specified branch. With -b: Git will create the specified branch if it does not exist.

`$ git merge [from name]`

Join specified [from name] branch into your current branch (the one you are on currently).

`$ git branch -d [name]`

Remove selected branch, if it is already merged into any other. -D instead of -d forces deletion.

f. Review your work

`$ git log [-n count]`

List commit history of current branch. -n count limits list to last n commits.

`$ git log --oneline --graph --decorate`

An overview with reference labels and history graph. One commit per line.

`$ git log ref..`

List commits that are present on the current branch and not merged into ref. A ref can be a branch name or a tag name.

`$ git log ..ref`

List commit that are present on ref and not merged into current branch.

`$ git reflog`

List operations (e.g. checkouts or commits) made on local repository

- **Conclusion and Discussion:**

Students are supposed to write your own conclusion

- **Viva Questions:**

- What is version control?
- What is staging area?
- What do mean by repository?
- What care is to be taken when merging two branches?

- **References:**

- Jon Loeliger and Matthew McCullough, Version Control with Git, O'Reilly Media, Second Edition, 2012.
- Dennis Hutten, Git: Learn Version Control with Git A Step-By-Step Ultimate Beginners Guide.
- Scott Chacon and Ben Straub, Pro Git_ Everything You Need to Know About Git, apress, Second Edition

Skill Based Lab IV – DevOPs

Experiment No.: 2

**Create and fork repositories in
GitHub**

Experiment No. 2

1. Aim: Create and fork repositories in GitHub

- a. Creating a GitHub account
- b. Synchronizing repositories.
- c. Tagging known commits
- d. Reverting changes

2. Objectives: From this experiment, the student will be able

- To understand DevOps practices which aims to simplify Software Development Life.
- To be aware of different Version Control tools like GIT, CVS or Mercurial

3. Outcomes: The learner will be able to

- To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.
- To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub.

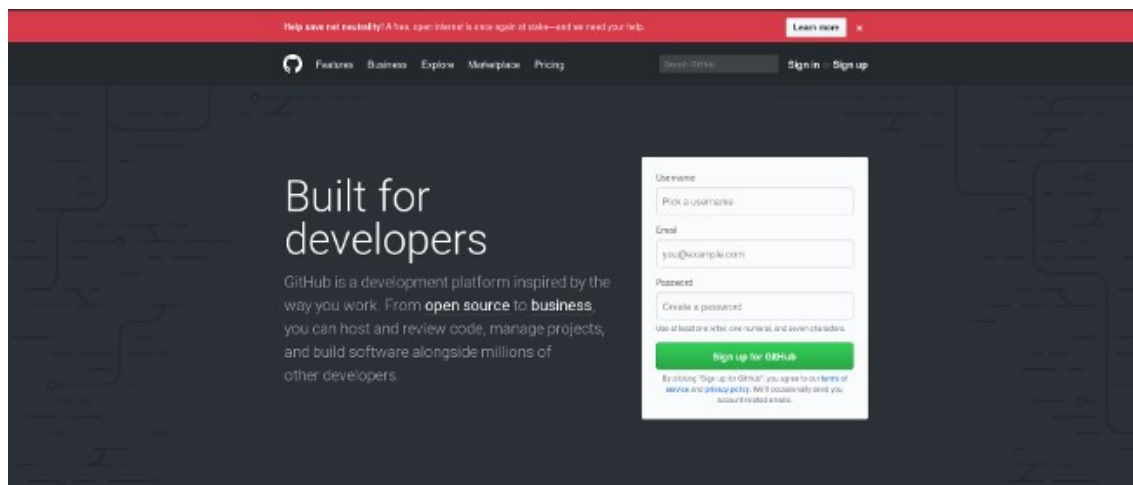
4. Hardware / Software Required : Linux / Windows Operating System

5. Theory:

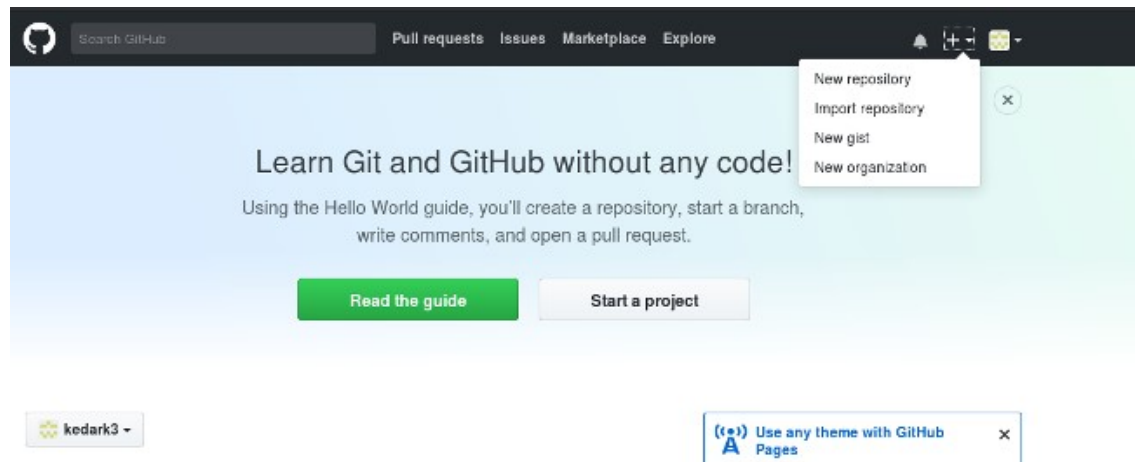
a. Creating a GitHub account

1. Step 1: Create a GitHub account

The easiest way to get started is to create an account on [GitHub.com](https://github.com) (it's free).

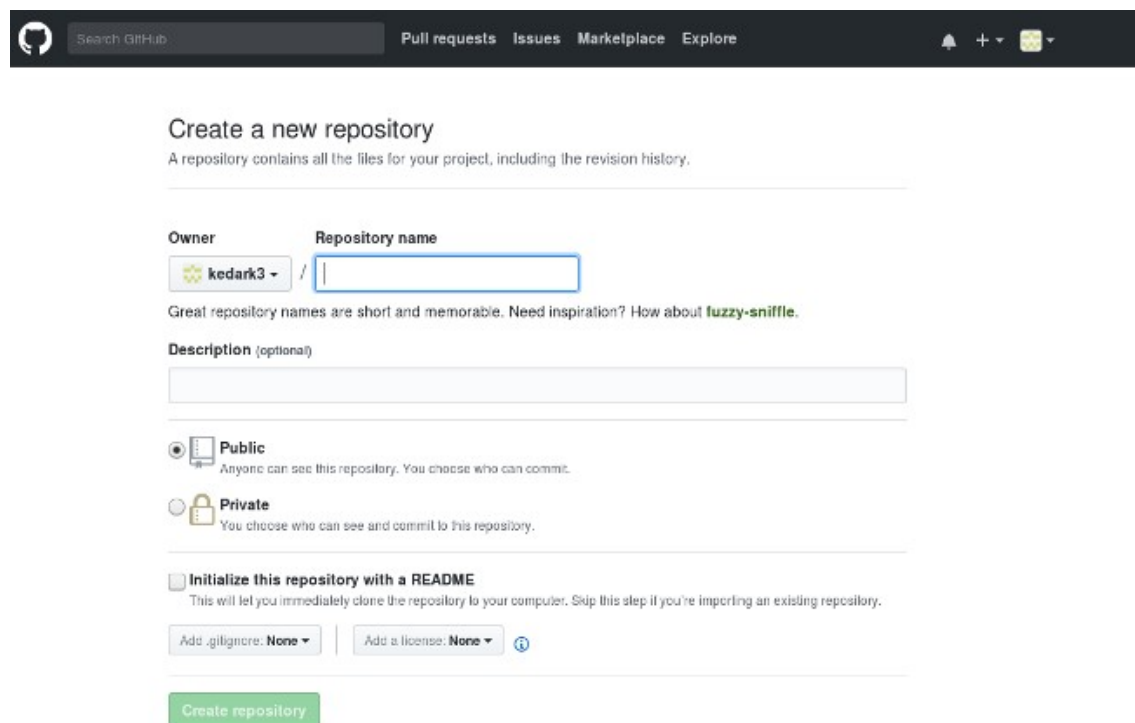


Pick a username (e.g., octocat123), enter your email address and a password, and click **Sign up for GitHub**. Once you are in, it will look something like this:



2. Step 2: Create a new repository

A repository is like a place or a container where something is stored; in this case we're creating a Git repository to store code. To create a new repository, select **New Repository** from the + sign dropdown menu (you can see I've selected it in the upper-right corner in the image above).

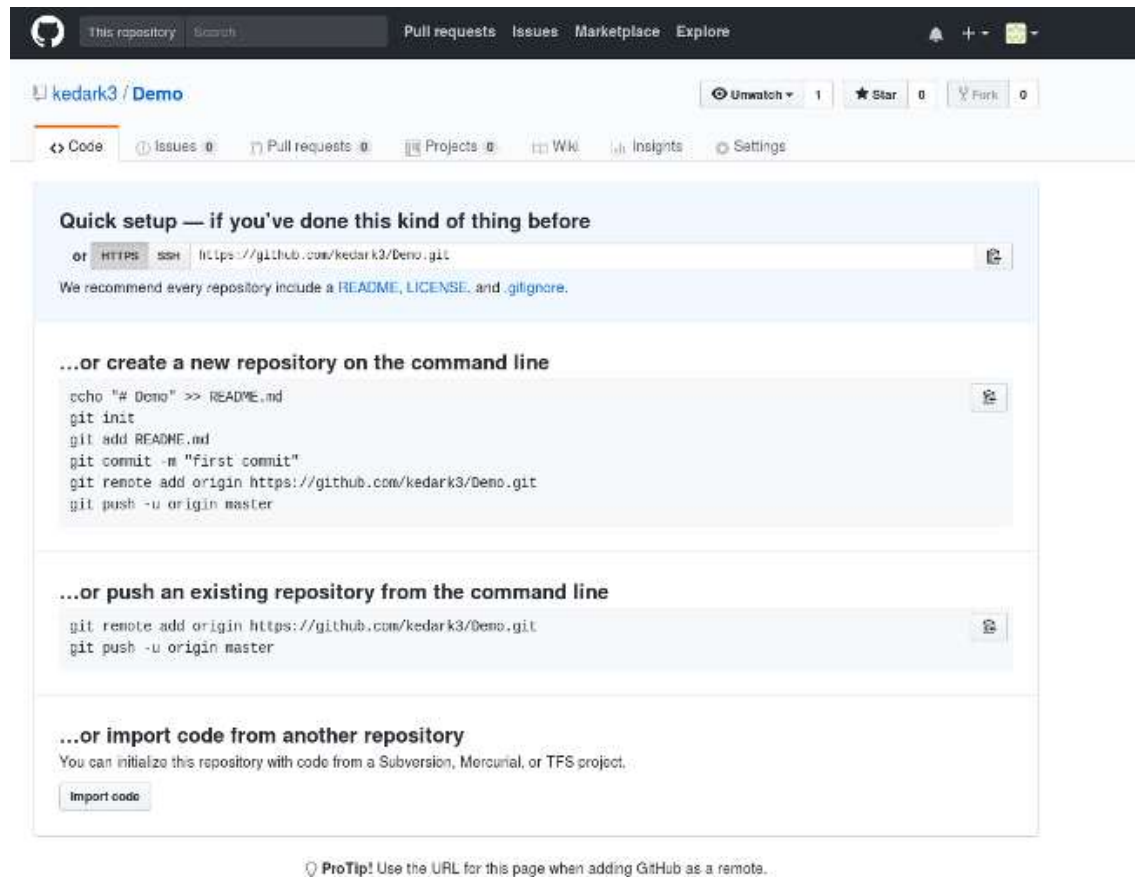


Enter a name for your repository (e.g., "Demo") and click **Create Repository**. Don't worry about changing any other options on this page.

Congratulations! You have set up your first repo on GitHub.com.

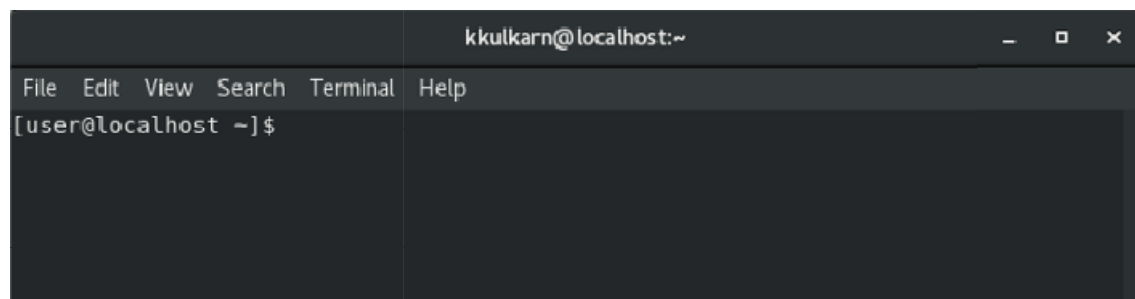
3. Step 3: Create a file

Once your repo is created, it will look like this:



Don't panic, it's simpler than it looks. Stay with me. Look at the section that starts "...or create a new repository on the command line," and ignore the rest for now.

Open the *Terminal* program on your computer.



Type `git` and hit **Enter**. If it says `command bash: git: command not found`, then [install Git](#) with the command for your Linux operating system or distribution. Check the installation by typing `git` and hitting **Enter**; if it's installed, you should see a bunch of information about how you can use the command.

In the terminal, type:

`mkdir Demo`

This command will create a directory (or folder) named *Demo*.

Change your terminal to the *Demo* directory with the command:

`cd Demo`

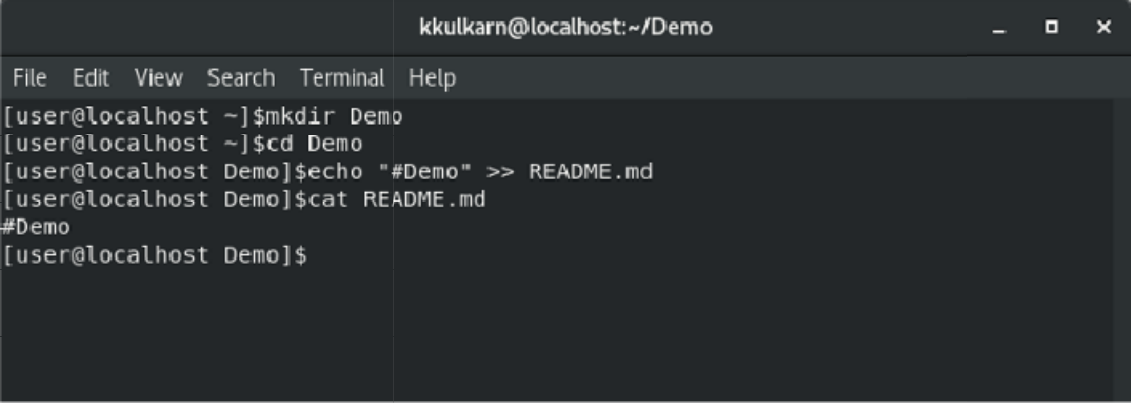
Then enter:

`echo "#Demo" >> README.md`

This creates a file named `README.md` and writes `#Demo` in it. To check that the file was created successfully, enter:

`cat README.md`

This will show you what is inside the `README.md` file, if the file was created correctly. Your terminal will look like this:



```
kkulkarn@localhost: ~/Demo
File Edit View Search Terminal Help
[user@localhost ~]$ mkdir Demo
[user@localhost ~]$ cd Demo
[user@localhost Demo]$ echo "#Demo" >> README.md
[user@localhost Demo]$ cat README.md
#Demo
[user@localhost Demo]$
```

To tell your computer that *Demo* is a directory managed by the Git program, enter:

`git init`

Then, to tell the Git program you care about this file and want to track any changes from this point forward, enter:

`git add README.md`

4. Step 4: Make a commit

So far you've created a file and told Git about it, and now it's time to create a *commit*. Commit can be thought of as a milestone. Every time you accomplish some work, you can write a Git commit to store that version of your file, so you can go back later and see what it looked like at that point in time. Whenever you make a change to your file, you create a new version of that file, different from the previous one.

To make a commit, enter:


```
git commit -m "first commit"
```

That's it! You just created a Git commit and included a message that says *first commit*. You must always write a message in commit; it not only helps you identify a commit, but it also enables you to understand what you did with the file at that point. So tomorrow, if you add a new piece of code in your file, you can write a commit message that says, *Added new code*, and when you come back in a month to look at your commit history or Git log (the list of commits), you will know what you changed in the files.

b. Synchronizing repositories

Step: Connect your GitHub repo with your computer

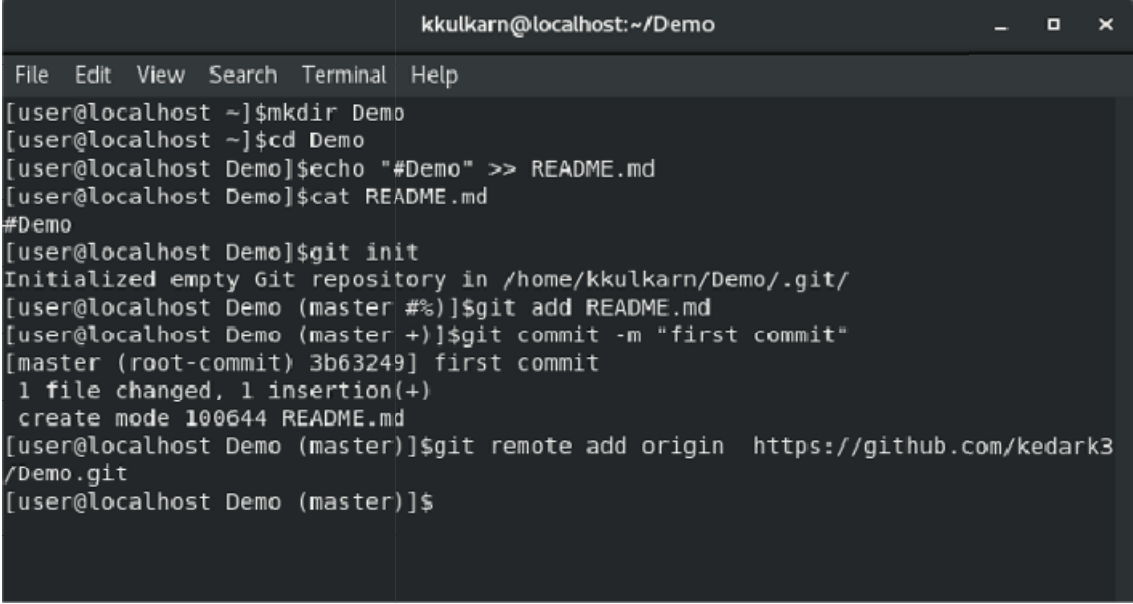
Now, it's time to connect your computer to GitHub with the command:

```
git remote add origin https://github.com/<your_username>/Demo.git
```

Let's look at this command step by step. We are telling Git to add a remote called origin with the address `https://github.com/<your_username>/Demo.git` (i.e., the URL of your Git repo on GitHub.com). This allows you to interact with your Git repository on GitHub.com by typing origin instead of the full URL and Git will know where to send your code.

Why origin? Well, you can name it anything else if you'd like.

Now we have connected our local copy of the *Demo* repository to its remote counterpart on GitHub.com. Your terminal looks like this:



```
kkulkarn@localhost:~/Demo
File Edit View Search Terminal Help
[user@localhost ~]$mkdir Demo
[user@localhost ~]$cd Demo
[user@localhost Demo]$echo "#Demo" >> README.md
[user@localhost Demo]$cat README.md
#Demo
[user@localhost Demo]$git init
Initialized empty Git repository in /home/kkulkarn/Demo/.git/
[user@localhost Demo (master #%)]$git add README.md
[user@localhost Demo (master +)]$git commit -m "first commit"
[master (root-commit) 3b63249] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
[user@localhost Demo (master)]$git remote add origin https://github.com/kedark3/Demo.git
[user@localhost Demo (master)]$
```

Now that we have added the remote, we can push our code (i.e., upload our README.md file) to GitHub.com.

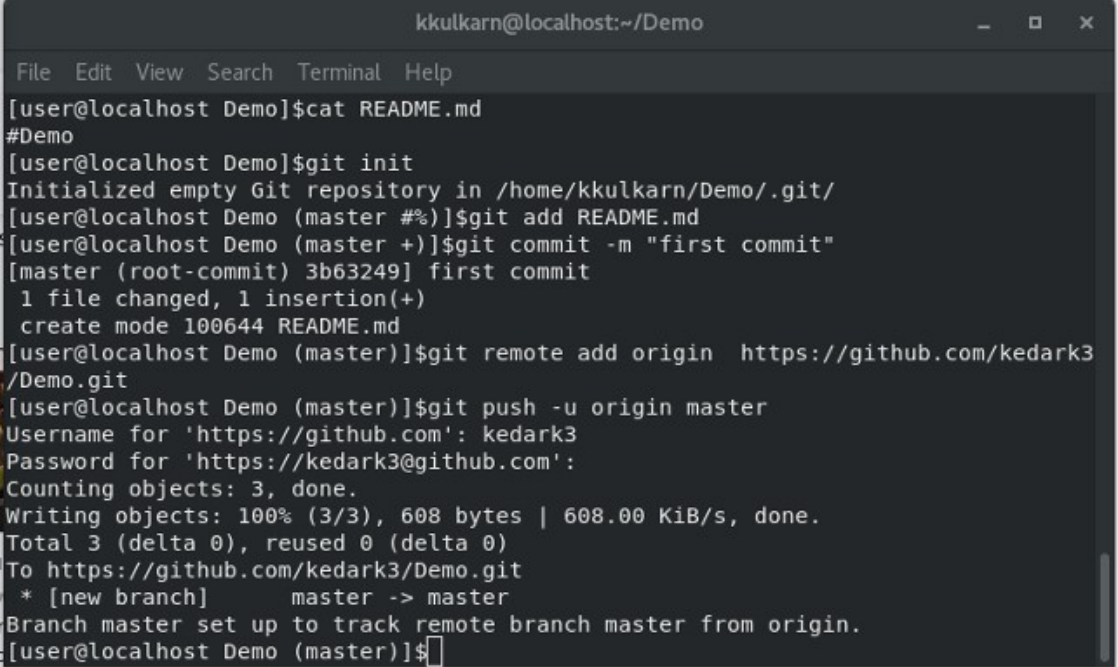
```
$ git push [--tags] [remote]
```

Push local changes to the remote. Use --tags to push tags.

```
$ git push -u [remote] [branch]
```

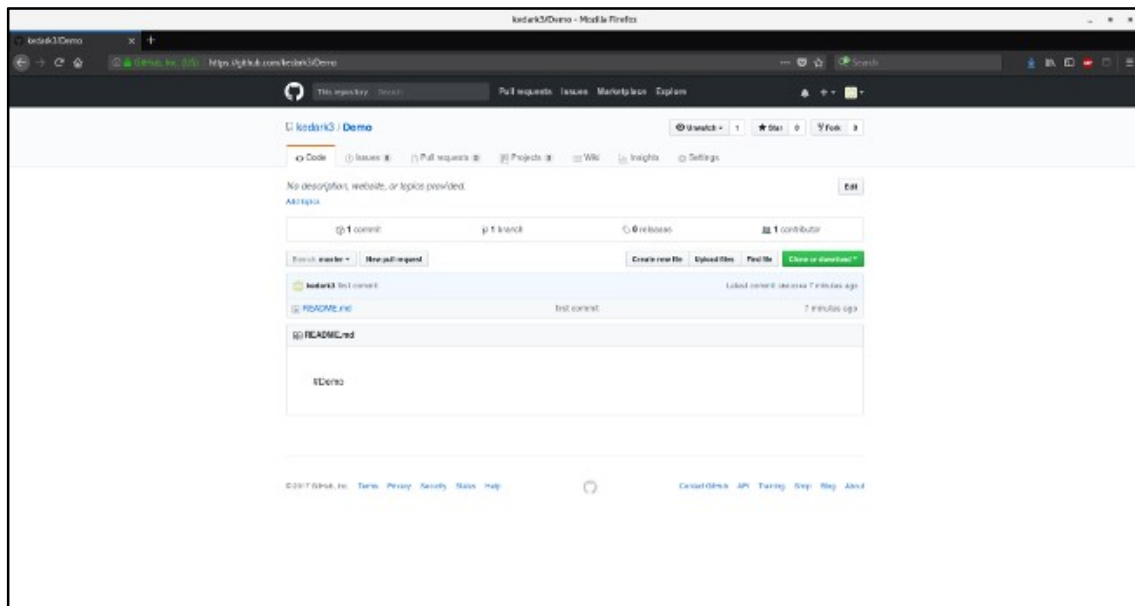
Push local branch to remote repository. Set its copy as an upstream.

Once you are done, your terminal will look like this:



```
kkulkarn@localhost:~/Demo
File Edit View Search Terminal Help
[user@localhost Demo]$cat README.md
#Demo
[user@localhost Demo]$git init
Initialized empty Git repository in /home/kkulkarn/Demo/.git/
[user@localhost Demo (master #%)]$git add README.md
[user@localhost Demo (master +)]$git commit -m "first commit"
[master (root-commit) 3b63249] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
[user@localhost Demo (master)]$git remote add origin https://github.com/kedark3/Demo.git
[user@localhost Demo (master)]$git push -u origin master
Username for 'https://github.com': kedark3
Password for 'https://kedark3@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 608 bytes | 608.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/kedark3/Demo.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
[user@localhost Demo (master)]$
```

And if you go to https://github.com/<your_username>/Demo you will see something like this:



That's it! You have created your first GitHub repo, connected it to your computer, and pushed (or uploaded) a file from your computer to your repository called *Demo* on GitHub.com.

```
$ git fetch [remote]
```

Fetch changes from the remote, but not update tracking branches.

```
$ git fetch --prune [remote]
```

Delete remote Refs that were removed from the remote repository.

```
$ git pull [remote]
```

Fetch changes from the remote and merge current branch with its upstream.

c. Tagging known commits

```
$ git tag
```

List all tags.

```
$ git tag [name] [commit sha]
```

Create a tag reference named name for current commit. Add commit sha to tag a specific commit instead of current one.

```
$ git tag -a [name] [commit sha]
```

Create a tag object named name for current commit.

```
$ git tag -d [name]
```

Remove a tag from local repository

d. Reverting changes

```
$ git reset [--hard] [target reference]
```

Switches the current branch to the target reference, leaving a difference as an uncommitted change. When --hard is used, all changes are discarded.

```
$ git revert [commit sha]
```

Create a new commit, reverting changes from the specified commit. It generates an inversion of changes.

6. Conclusion and Discussion:

Students are supposed to write your own conclusion

7. Viva Questions:

- What is difference between git and github?
- How do you push your project into remote repository?
- Is it possible to revert changes after commit? Is so, how?

8. References:

- Jon Loeliger and Matthew McCullough, Version Control with Git, O'Reilly Media, Second Edition, 2012.
- Dennis Hutten, Git: Learn Version Control with Git A Step-By-Step Ultimate Beginners Guide.
- Scott Chacon and Ben Straub, Pro Git_ Everything You Need to Know About Git, apress, Second Edition.

Skill Based Lab IV – DevOPs

Experiment No.: 3

Installation of Jenkins

**To install and configure Jenkins to setup
a build Job.**

Experiment No. 3

1. **Aim:** To install and configure Jenkins to setup a build Job.
2. **Objectives:** From this experiment, the student will be able
 - To install and build job in Jenkins for deploying application DevOps environment.
3. **Outcomes: The learner will be able to**
 - To understand the importance of Jenkins to Build and deploy Software Applications on server environment.
4. **Hardware / Software Required: Linux / Windows Operating System**
5. **Theory:**

Jenkins is a Java-based open-source automation platform with built-in continuous integration plugins. Jenkins is used to continuously build and test your software projects, making it simpler for developers to incorporate changes to the project and for users to get a new build. By interacting with a wide range of testing and deployment tools, it also enables you to release your software continually.

Organizations can use Jenkins to automate and speed up the software development process. Jenkins combines all stages of the development lifecycle, including build, document, test, package, stage, deploy, static analysis, and many others.

Jenkins achieves Continuous Integration with the help of plugins. Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven 2 project, Amazon EC2, HTML publisher etc.

Advantages of Jenkins include:

- It is an open-source tool with great community support.
- It is easy to install.
- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share it with the community.
- It is free of cost.
- It is built with Java and hence, it is portable to all the major platforms.

Jenkins Features:

- **Continuous Integration and Continuous Delivery:** As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.

- **Easy configuration**
Jenkins can be easily set up and configured via its web interface, which includes on-the-fly error checks and built-in help.
- **Easy installation**
Jenkins is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Linux, macOS and other Unix-like operating systems.
- **Plugins**
With hundreds of plugins in the Update Center, Jenkins integrates with practically every tool in the continuous integration and continuous delivery toolchain.
- **Extensible**
Jenkins can be extended via its plugin architecture, providing nearly infinite possibilities for what Jenkins can do.
- **Distributed**
Jenkins can easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster.

Steps for Installation of Jenkins:

1. **Installation of JAVA:** Download JDK 11/17 and choose windows 32-bit or 64-bit according to your system configuration. Click on "accept the license agreement." Set the Path for the Environmental Variable for JDK:
 - Go to System Properties. Under the "Advanced" tab, select "Environment Variables."
 - Under system variables, select "new." Then copy the path of the JDK folder and paste it in the corresponding value field. Similarly, do this for JRE.
 - Under system variables, set up a bin folder for JDK in PATH variables.
 - Go to command prompt and type the following to check if Java has been successfully installed:

`C:\Users\Aditi>java -version`
2. Browse to the official Jenkins download page. Under the Downloading Jenkins section is a list of installers for the long-term support (LTS) version of Jenkins. Click the Windows link to begin the download.
Once the download is complete, run the **jenkins.msi** installation file.

Downloading Jenkins

Jenkins is distributed as WAR files, native packages, installers, and Docker images. Follow these installation steps:

1. Before downloading, please take a moment to review the [Hardware and Software requirements](#) section of the User Handbook.
2. Select one of the packages below and follow the download instructions.
3. Once a Jenkins package has been downloaded, proceed to the [Installing Jenkins](#) section of the User Handbook.
4. You may also want to verify the package you downloaded. [Learn more about verifying Jenkins downloads.](#)

Download Jenkins 2.303.3 LTS for:

Generic Java package (.war) SHA-256: 8a0ee7307755b3f31a050faa945f7a3991abdb43d941c7294cac890c1e27
Docker
Ubuntu/Debian
CentOS/Fedora/Red Hat
Windows
openSUSE
FreeBSD
Gentoo
macOS
OpenBSD

Download Jenkins 2.319 for:

Generic Java package (.war) SHA-256: 50e9c818cda1bdf3ba7e2a1e590f027a888bd527d5bfc2daea944de351c7105
Docker
Ubuntu/Debian
CentOS/Fedora/Red Hat
Windows
openSUSE
Arch Linux
FreeBSD
Gentoo
macOS

3. The setup wizard starts. Click **Next** to proceed.



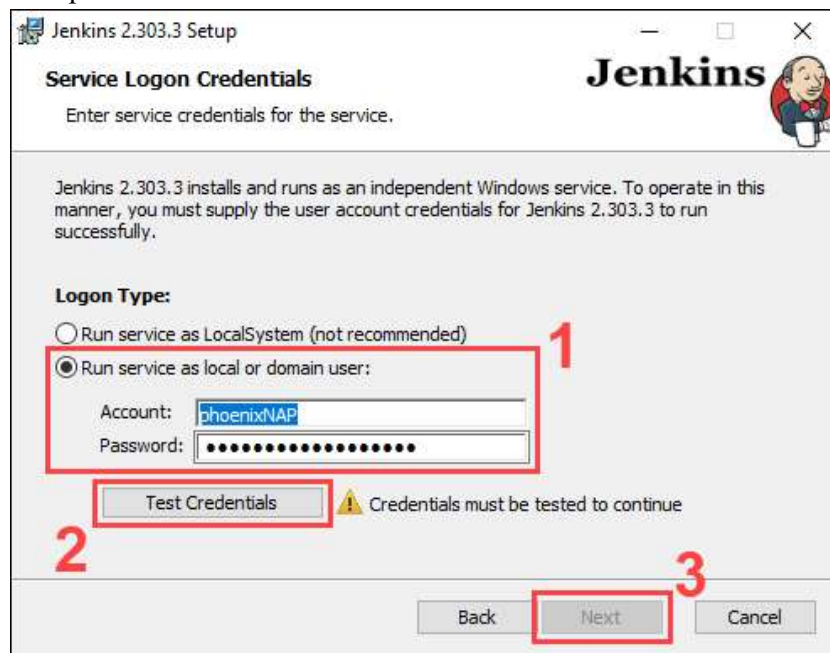
4. Select the install destination folder and click **Next** to continue.



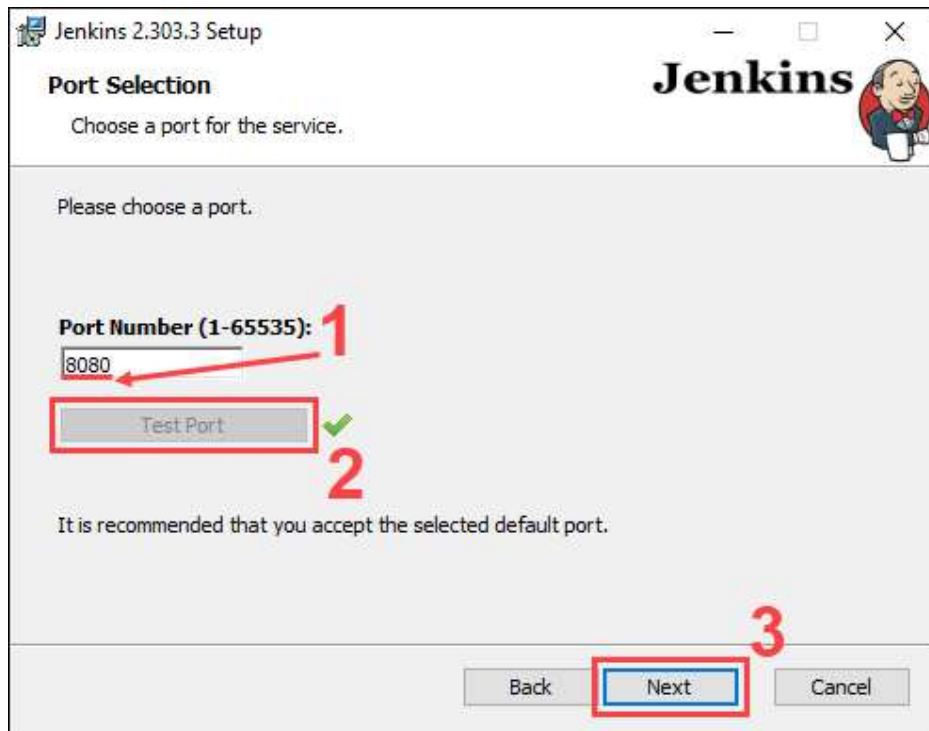
5. Under the **Run service as a local or domain user** option, enter the domain username and password for the user account you want to run Jenkins with. Click **Test Credentials** to verify the login data, then click **Next** to proceed.

Or

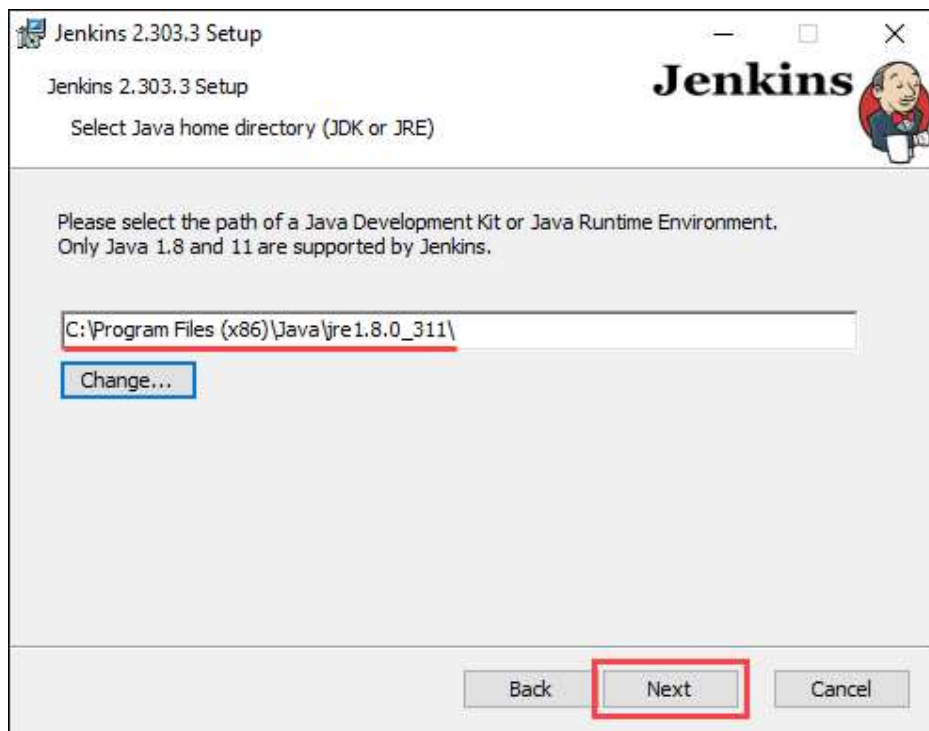
Using the **Run service as LocalSystem** option doesn't require you to enter user login data. Instead, it grants Jenkins full access to your system and services. Note that this is a less secure option and is thus not recommended.



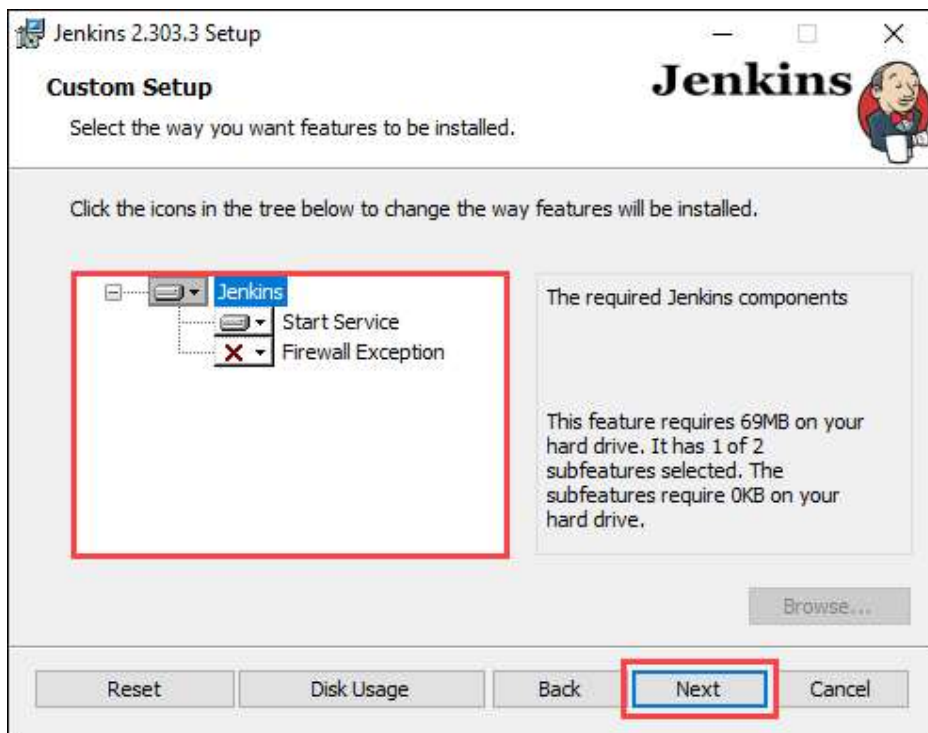
6. Enter the port number you want Jenkins to run on. Click **Test Port** to check if the selected port is available, then click **Next** to continue.



7. Select the directory where Java is installed on your system and click **Next** to proceed.



8. Select the features you want to install with Jenkins and click **Next** to continue.



9. Click **Install** to start the installation process.



10. Once the installation is complete, click **Finish** to exit the install wizard.



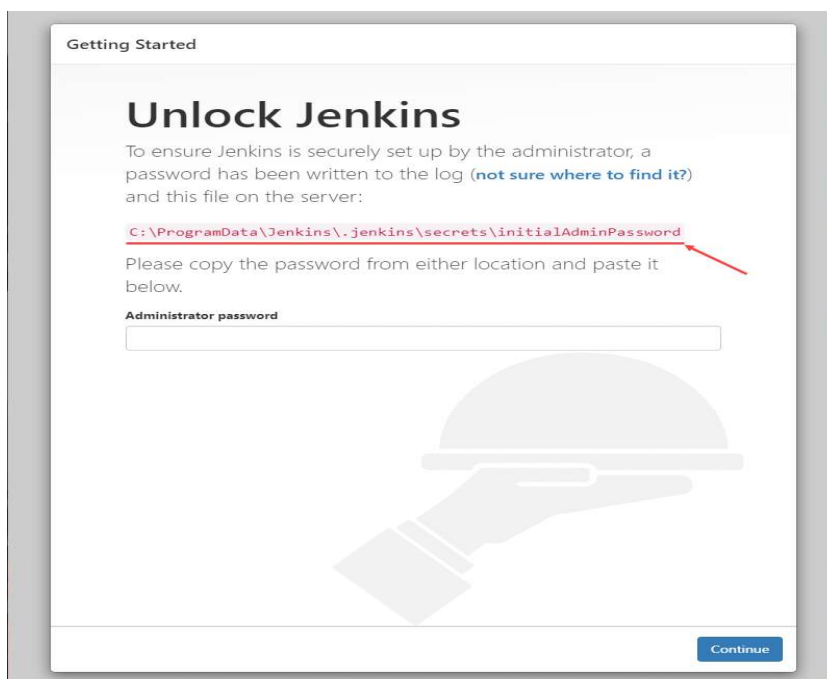
Unblock Jenkins

After completing the installation process, you have to unblock Jenkins before you can customize and start using it.

1. In your web browser, navigate to the port number you selected during the installation using the following address:

`http://localhost:[port number]`

2. Navigate to the location on your system specified by the Unblock Jenkins page.

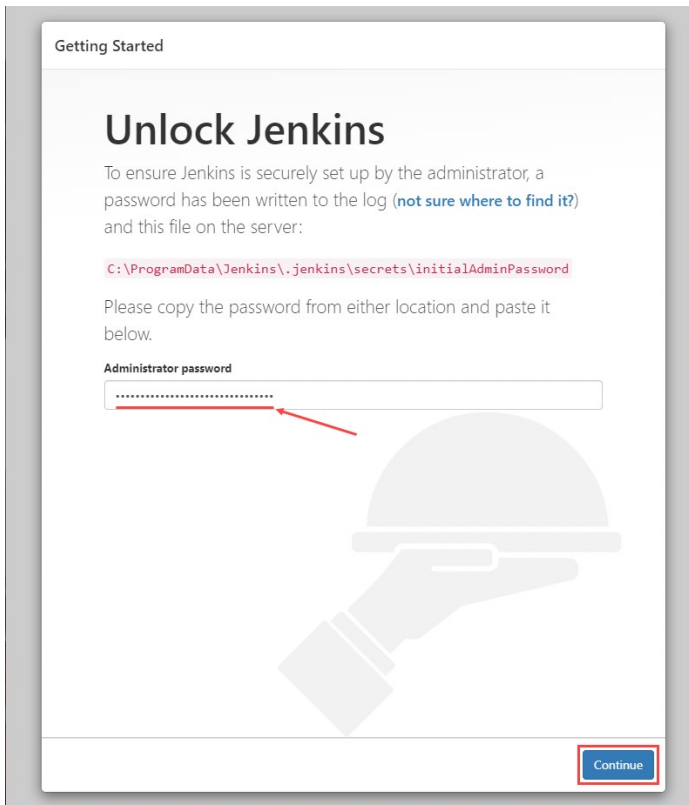


3. Open the **initialAdminPassword** file using a text editor such as Notepad.

4. Copy the password from the **initialAdminPassword** file.



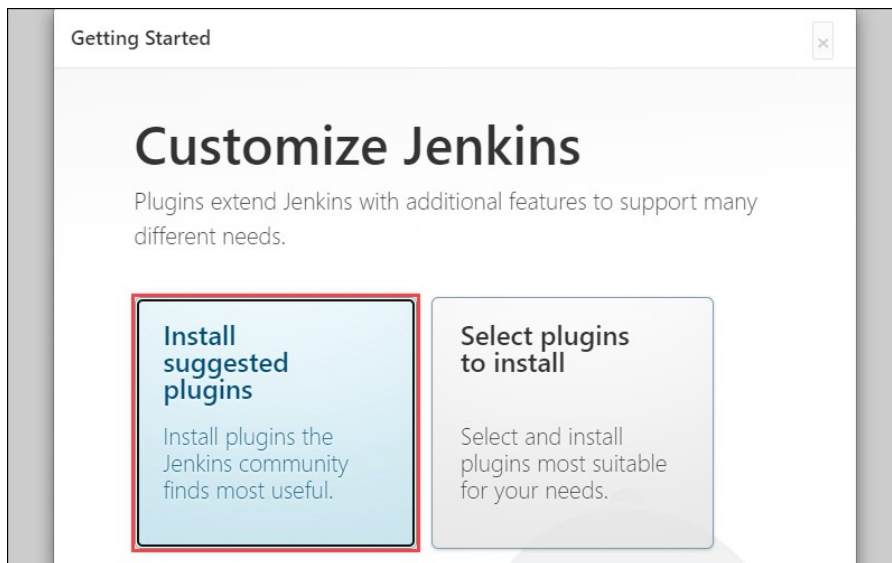
5. Paste the password in the **Administrator password** field on the Unblock Jenkins page and click **Continue** to proceed.



Customize Jenkins

Once you unlock Jenkins, customize and prepare the Jenkins environment.

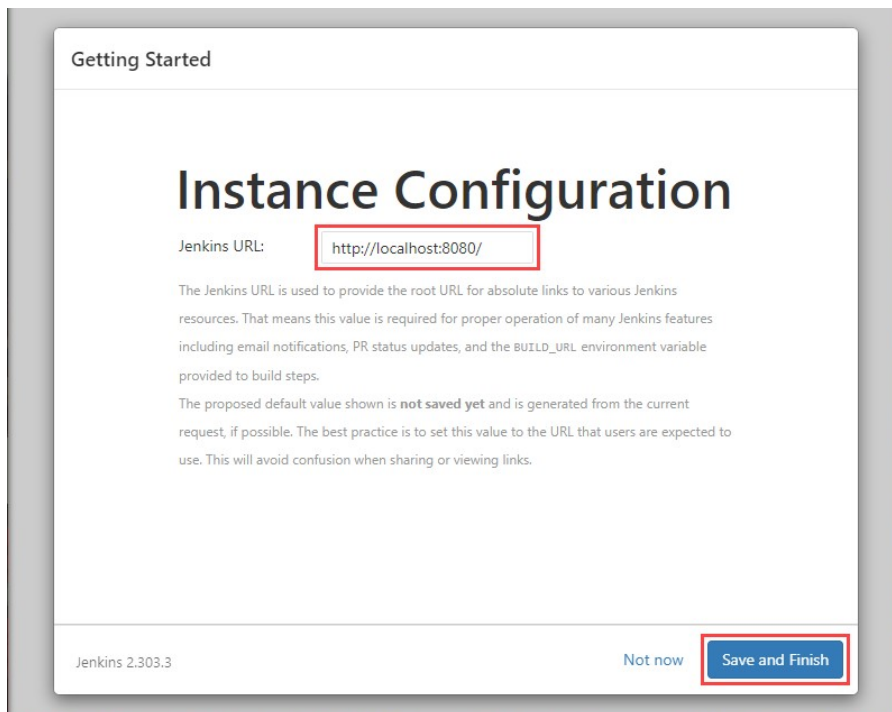
1. Click the **Install suggested plugins** button to have Jenkins automatically install the most frequently used plugins.



2. After Jenkins finishes installing the plugins, enter the required information on the **Create First Admin User** page. Click **Save and Continue** to proceed.

A screenshot of the Jenkins 'Getting Started' window. The title is 'Create First Admin User'. Below it, there are five input fields: 'Username:', 'Password:', 'Confirm password:', 'Full name:', and 'E-mail address:'. A red box highlights these five input fields. At the bottom right, there is a blue button labeled 'Save and Continue' (highlighted with a red box). At the bottom left, it says 'Jenkins 2.303.3'. In the center bottom, there is a link that says 'Skip and continue as admin'.

3. On the **Instance Configuration** page, confirm the port number you want Jenkins to use and click **Save and Finish** to finish the initial customization.

The image shows the 'Getting Started' section of the Jenkins Instance Configuration page. The title 'Instance Configuration' is prominently displayed. Below it, the 'Jenkins URL' field is pre-filled with 'http://localhost:8080/' and is highlighted with a red box. A paragraph explains that the Jenkins URL is used for absolute links to various resources. Another paragraph notes that the proposed default value is 'not saved yet' and is generated from the current request. At the bottom, there are two buttons: 'Not now' and 'Save and Finish', with the latter highlighted by a red box. The version 'Jenkins 2.303.3' is shown in the bottom left corner.

Getting Started

Instance Configuration

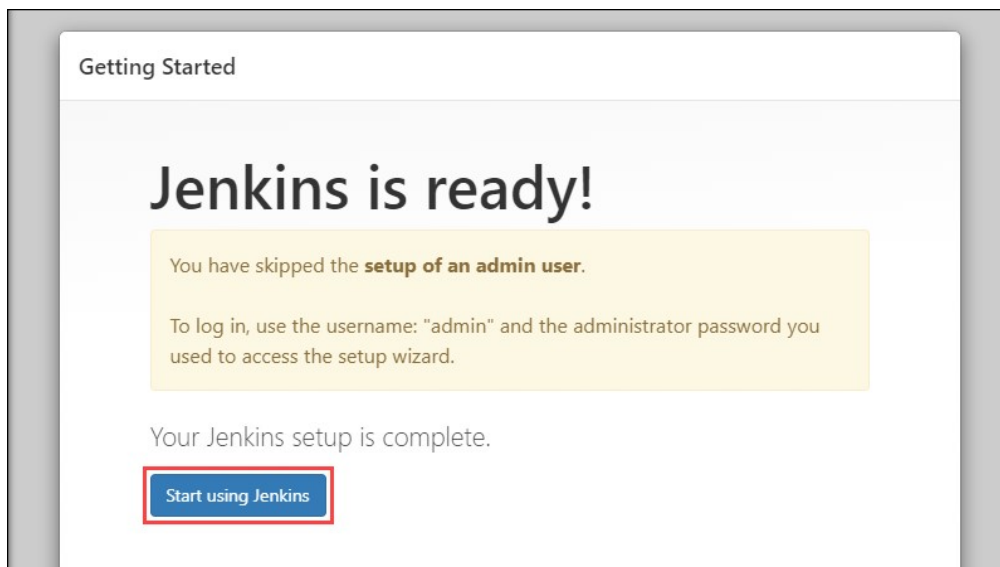
Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.303.3 [Not now](#) [Save and Finish](#)

4. Click the **Start using Jenkins** button to move to the Jenkins dashboard.

The image shows the 'Getting Started' section of the Jenkins 'Jenkins is ready!' page. The title 'Jenkins is ready!' is prominently displayed. Below it, a yellow box contains a warning: 'You have skipped the setup of an admin user.' and instructions to log in with the username 'admin' and the administrator password. Below this, it states 'Your Jenkins setup is complete.' and features a 'Start using Jenkins' button, which is highlighted with a red box.

Getting Started

Jenkins is ready!

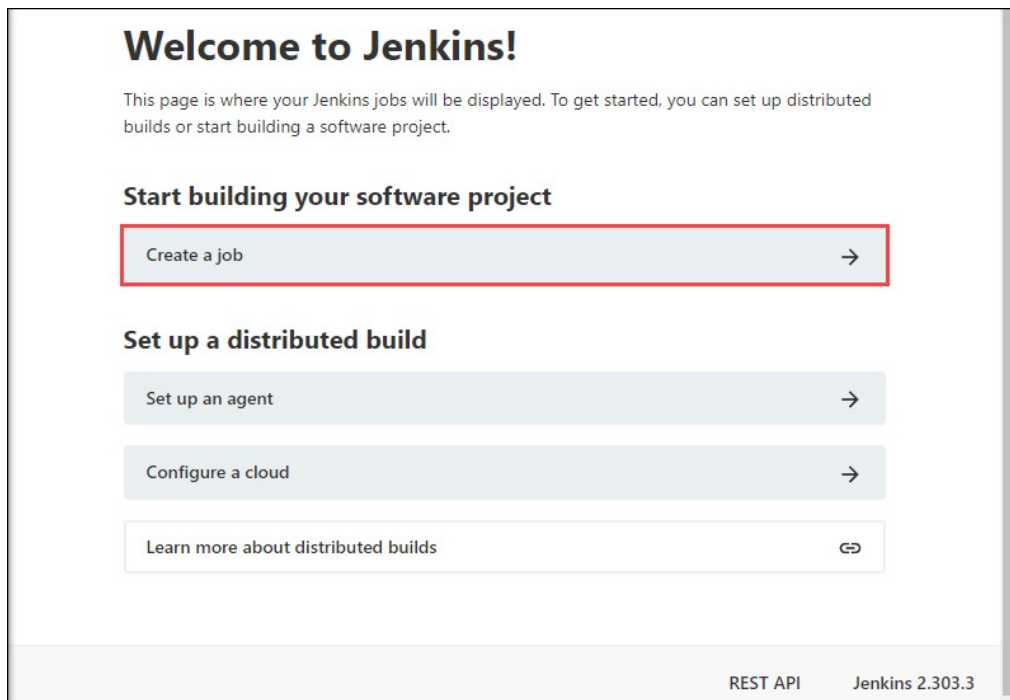
You have skipped the **setup of an admin user**.

To log in, use the username: "admin" and the administrator password you used to access the setup wizard.

Your Jenkins setup is complete.

[Start using Jenkins](#)

5. Using the Jenkins dashboard, click **Create a job** to build your first Jenkins software project.



Students should build a job and execute in Jenkins environment.

6. Conclusion and Discussion:

Students are supposed to write your own conclusion

7. Viva Questions:

- What is Continuous Integration?
- What is CI/CD?
- Which tools can be plugged with Jenkins?

8. References:

- Jenkins: The Definitive Guide: Continuous Integration for the Masses, by John Ferguson Smart Published by O'Reilly Media
- Jenkins 2: Up and Running Authored by Brent Laster Published by O'Reilly Media