**18-Nov-2024**

**Internship Day - 81 Report:**

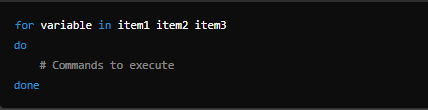
# ****Understanding Loops in Bash Scripting****

Loops are fundamental constructs in programming that allow repetitive execution of a block of code. They are essential for automating tasks, processing data, and managing iterative operations efficiently. In Bash scripting, loops help execute commands multiple times without manual intervention, enhancing script functionality and flexibility.

## ****1. Introduction to**** for ****Loops in Bash****

The for loop in Bash iterates over a list of items, executing a set of commands for each item in the list. This is particularly useful when performing repetitive tasks such as processing files, handling user inputs, or automating system operations.

### ****Syntax of**** for ****Loop:****

****

* **variable**: A placeholder for the current item in the iteration.
* **item1 item2 item3**: The list of items to iterate over.
* **do ... done**: The block of commands to execute for each item.

## ****2. Introduction to**** for ****While in Bash****

The while loop in Bash scripting is a control flow statement that allows code to be executed repeatedly as long as a given condition evaluates to true. It is commonly used for tasks where the number of iterations is not predetermined, making it highly flexible.

### ****Syntax of**** While ****Loop:****

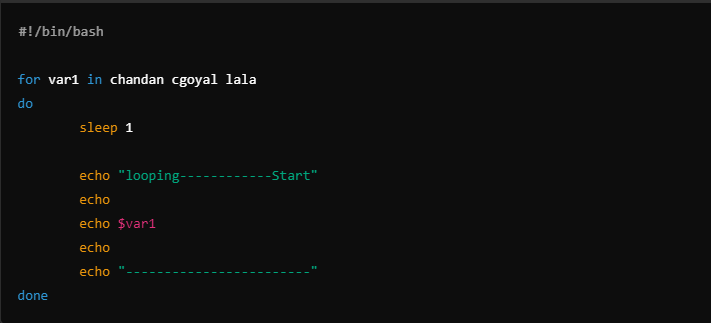
### 

# ****Example of For Loop:** **Bash Script:** Input and Output**

## ****Input:****

The uploaded Bash script iterates over a list of names and outputs them with formatting and a delay. Here is the script content:

**Script:**

****

## ****Output:**** When the script is executed, it produces the following formatted output:

****

# ****Example of While Loop: Bash Script:** Input and Output**

**Input (Script):**

****

### Output:

When you run this script, it will:

1. Start with i=1.
2. Continue the loop as long as i is less than or equal to 10.
3. Print the value of i with a delay of 1 second between each iteration.

### Example Output:

****

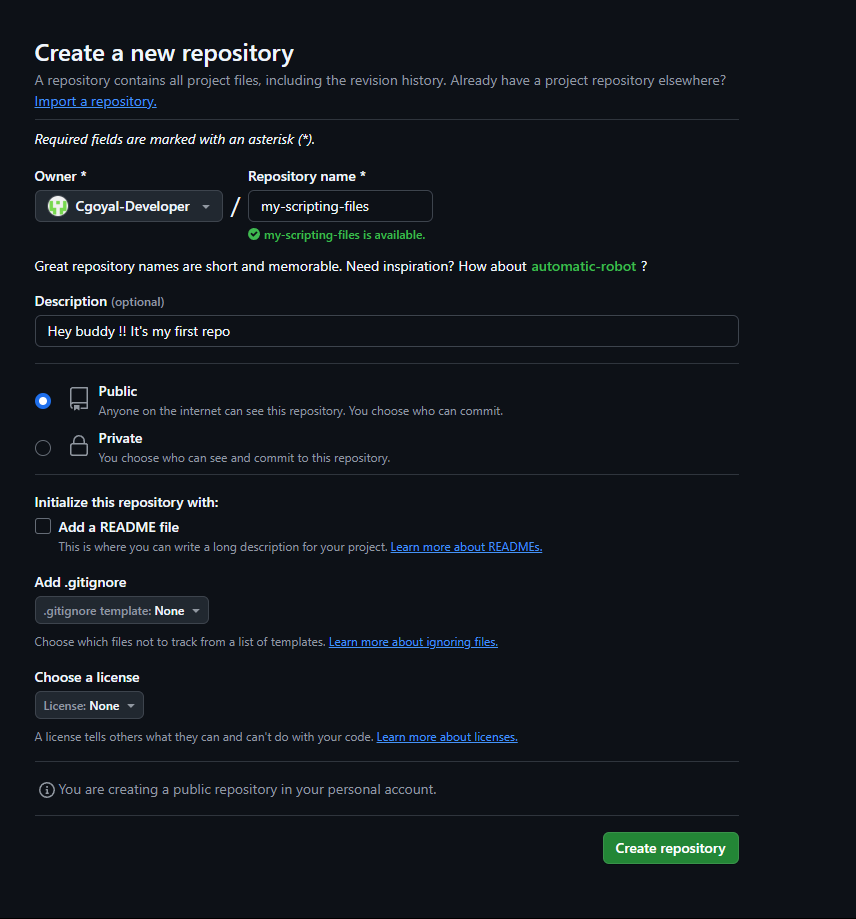
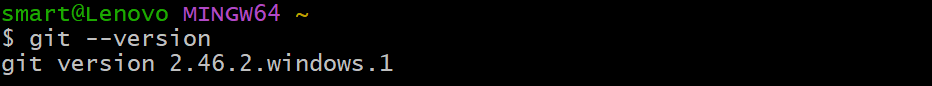
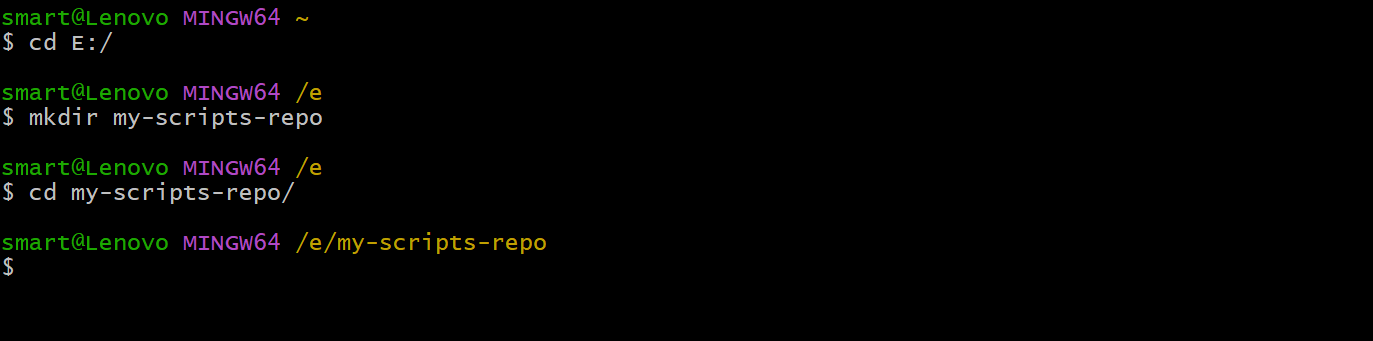
**19-Nov-2024**

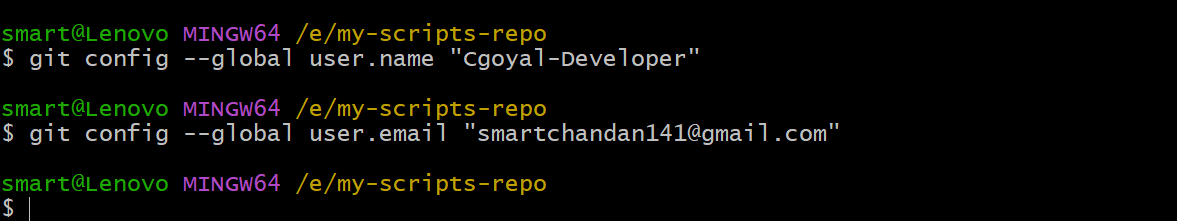
**Internship Day - 82 Report:**

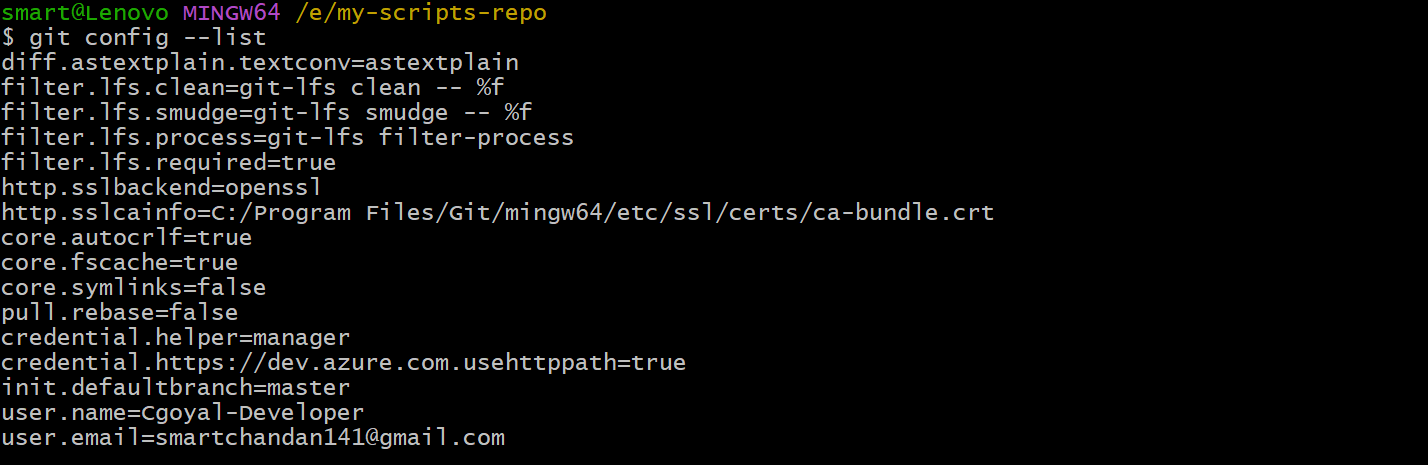
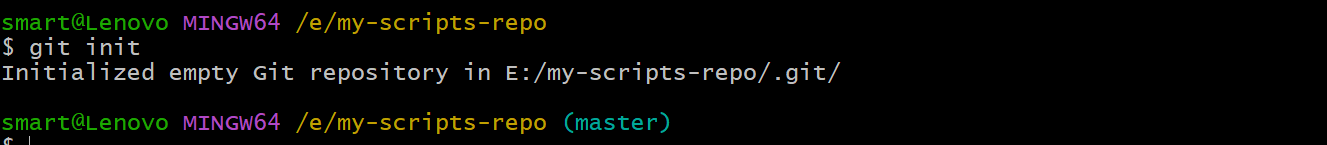
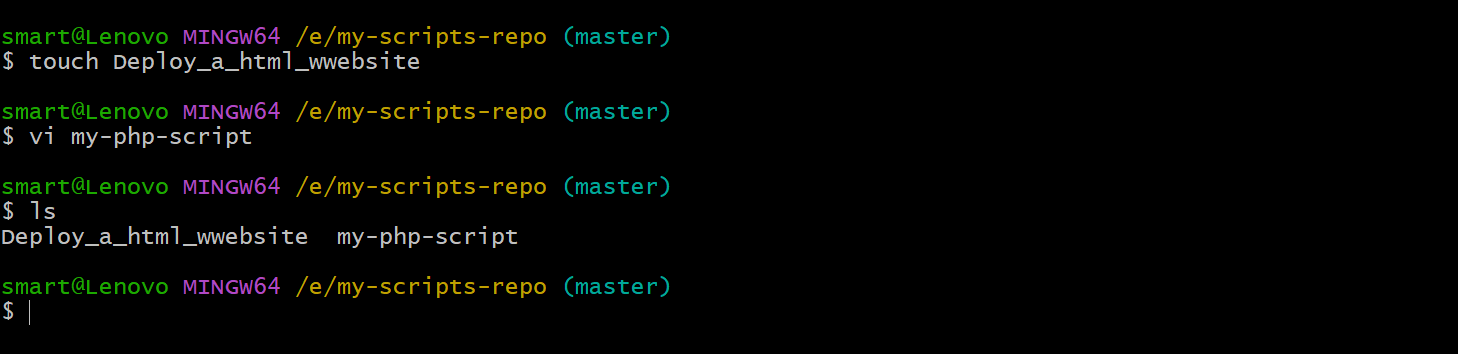
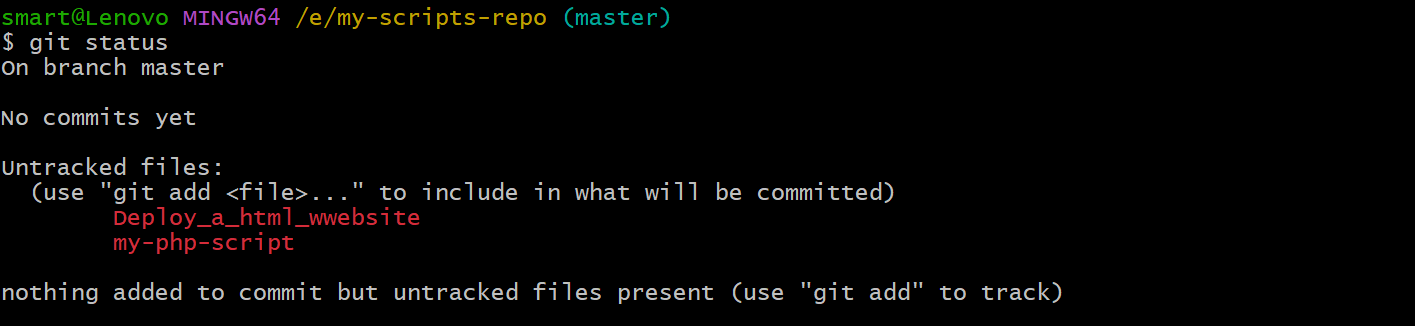
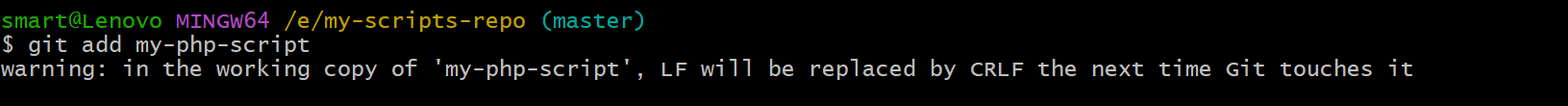
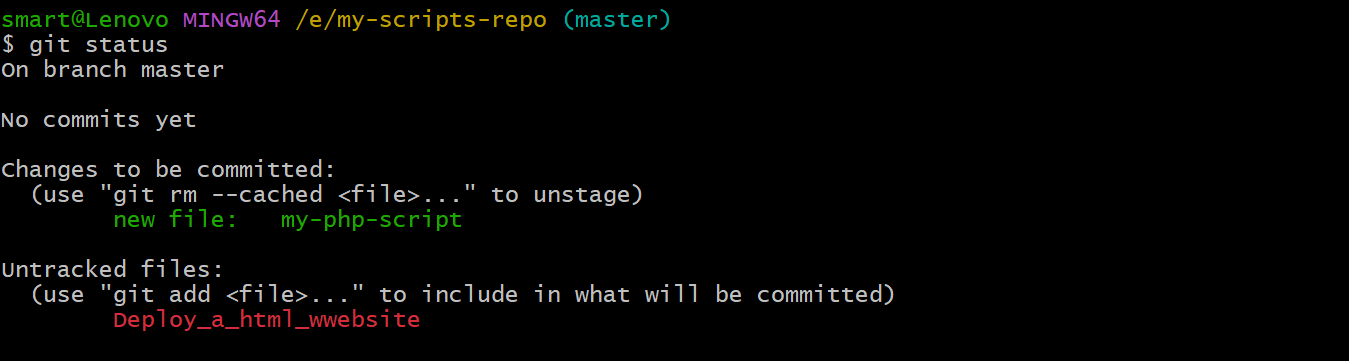
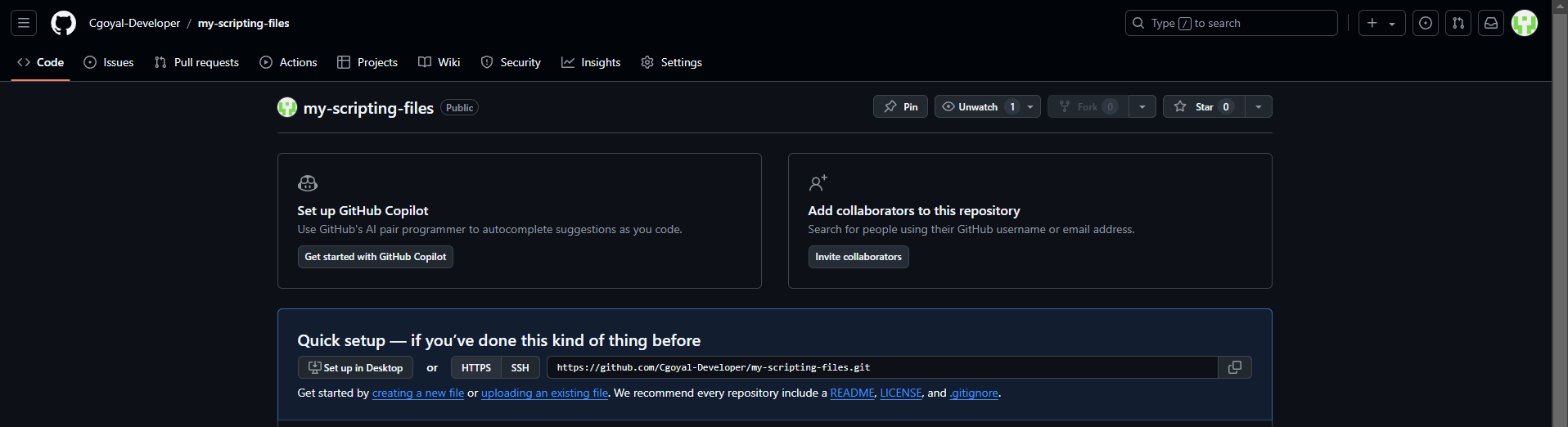
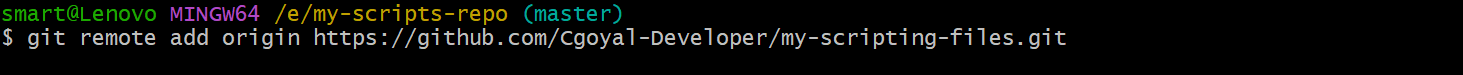
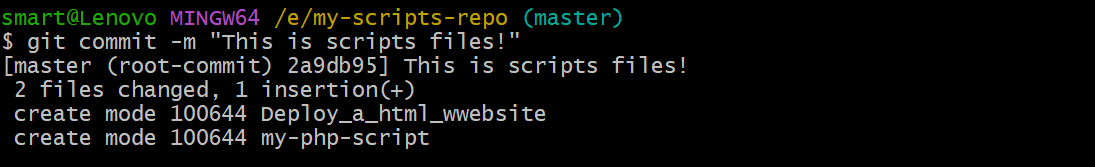
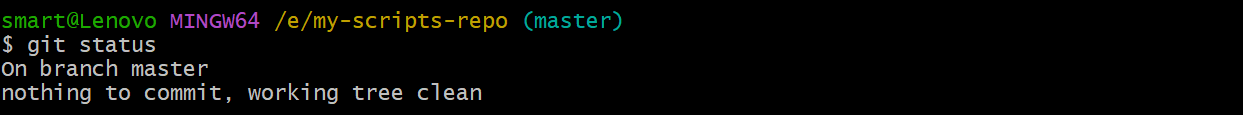
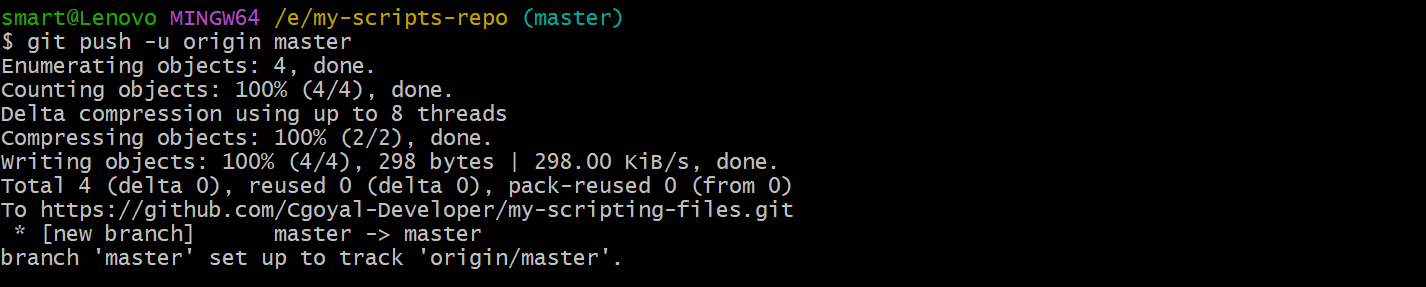
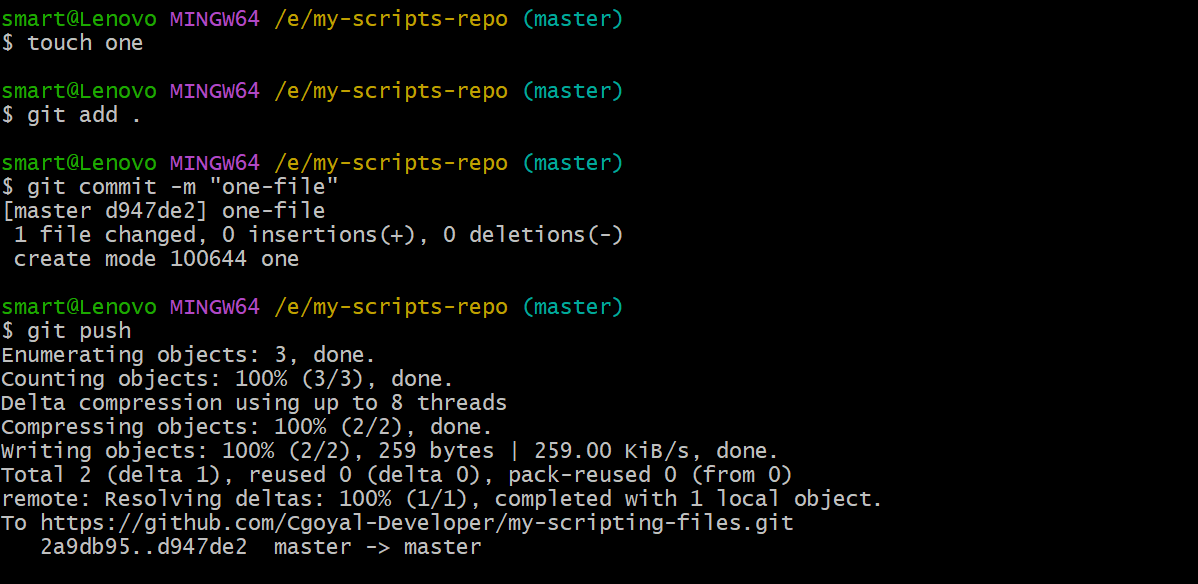
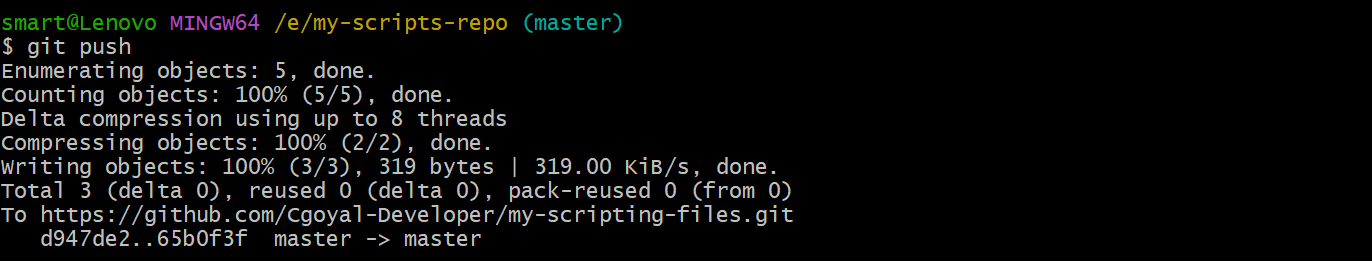
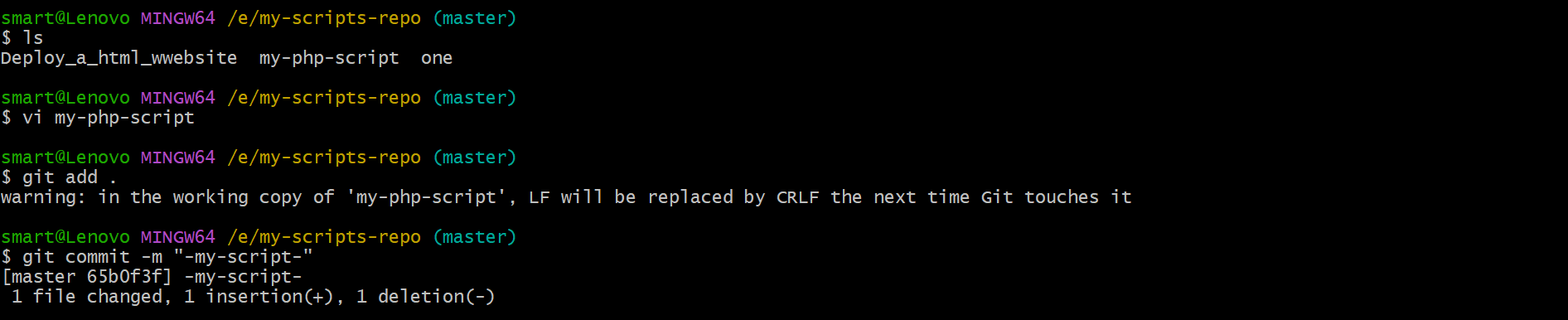
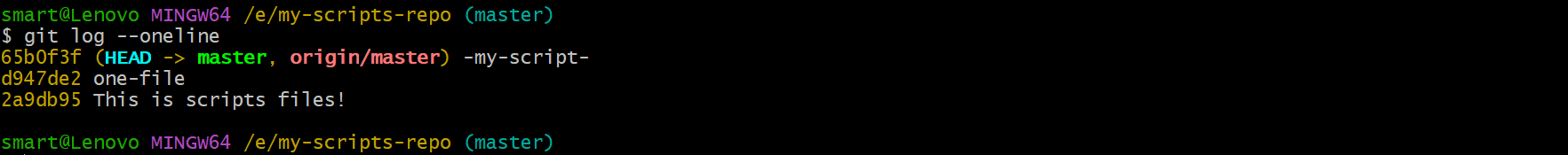
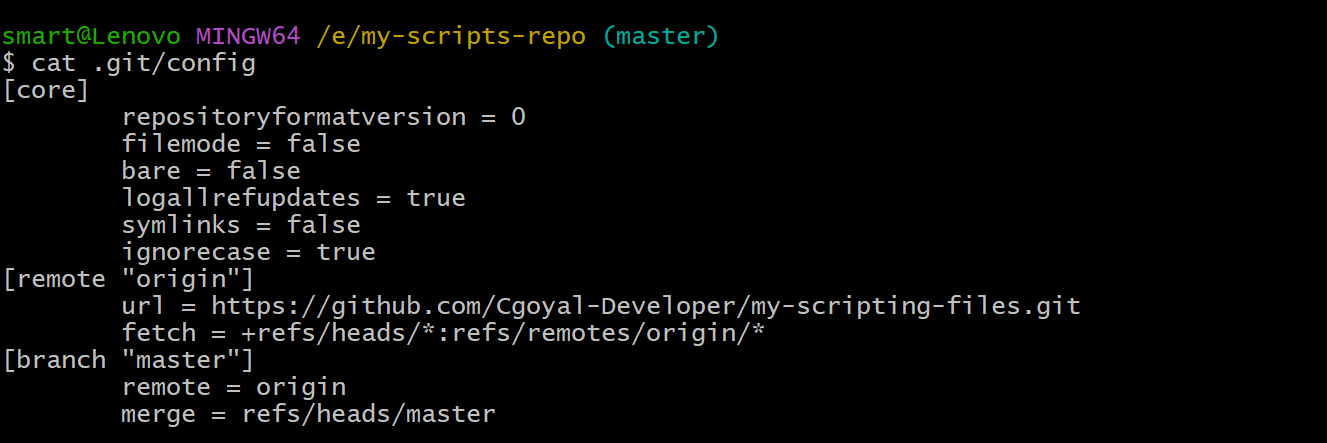
### ****Introduction to GitHub****

GitHub is a web-based platform that facilitates version control and collaborative software development. Built on top of Git, a distributed version control system, GitHub provides a host of tools for managing projects, tracking changes, and collaborating effectively.

### ****Key Features of GitHub****

1. **Version Control:**
   * Tracks changes in your code over time.
   * Allows reverting to previous versions if needed.
2. **Repositories (Repos):**
   * A repository is a storage space for your project files.
   * Can be public (accessible to everyone) or private (restricted access).
3. **Collaboration:**
   * Multiple developers can work together on the same project.
   * Features like pull requests, forks, and issues enhance teamwork.
4. **Pull Requests:**
   * Used to propose changes to a codebase.
   * Enables discussion, code review, and testing before merging changes.
5. **Issues and Bug Tracking:**
   * A feature for reporting and tracking bugs or suggesting improvements.
6. **Branching and Merging:**
   * Allows developers to work on different features or fixes without affecting the main codebase.
   * Changes can be merged back into the main branch after review.
7. **Actions:**
   * Automates workflows such as CI/CD (Continuous Integration and Continuous Deployment).
8. **Markdown Support:**
   * Provides formatting for project documentation, README files, and comments.
9. **Community and Open Source:**
   * A vast library of open-source projects to contribute to or use as inspiration.
10. **Integrations:**
    * Supports integration with tools like Slack, Jira, and third-party CI/CD systems.
11. **Make a new repository in GitHub.**
12. **Open Git Bash and check the Git version:**  
    git –version  
      
    
13. **Create a new folder in your system related to the repository.**

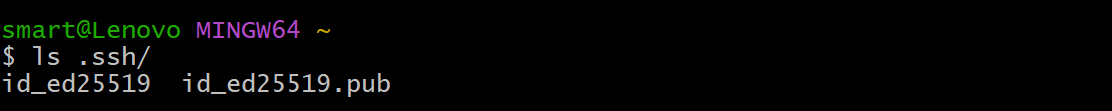
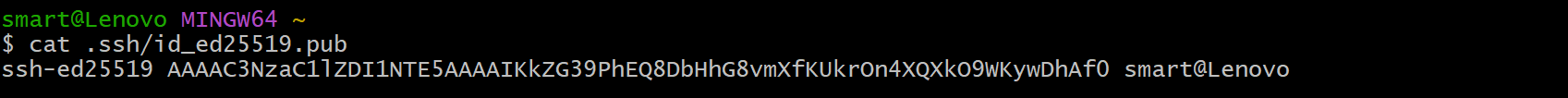
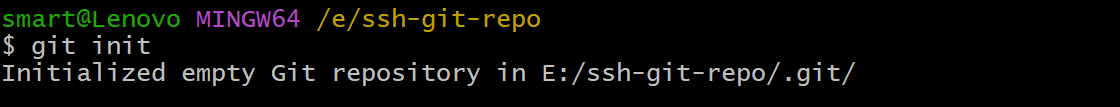
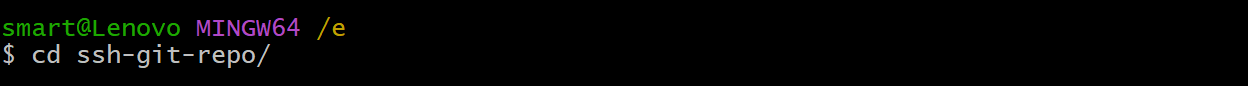
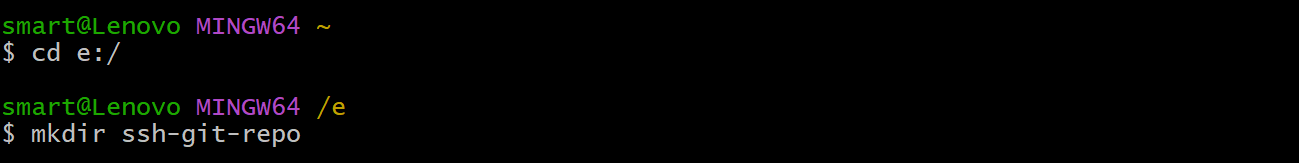
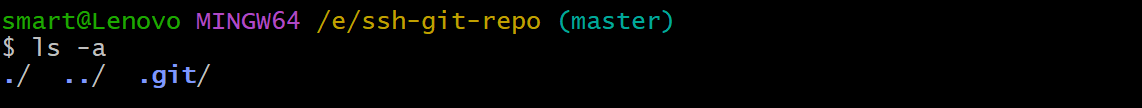
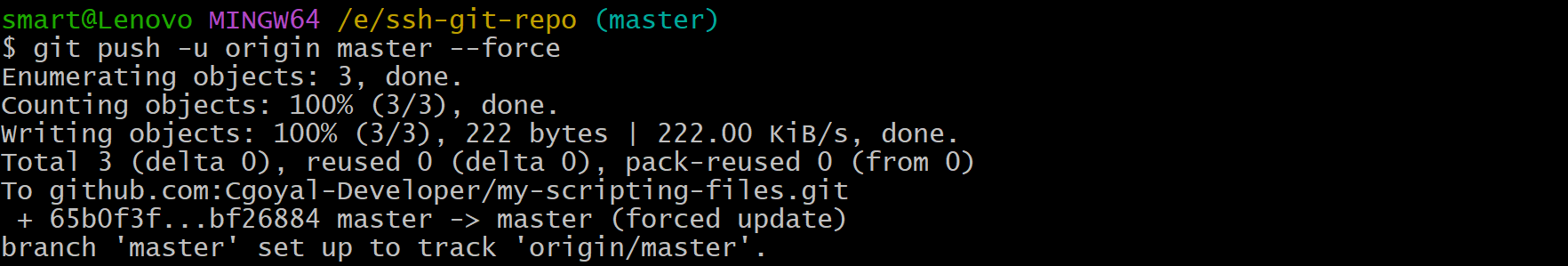
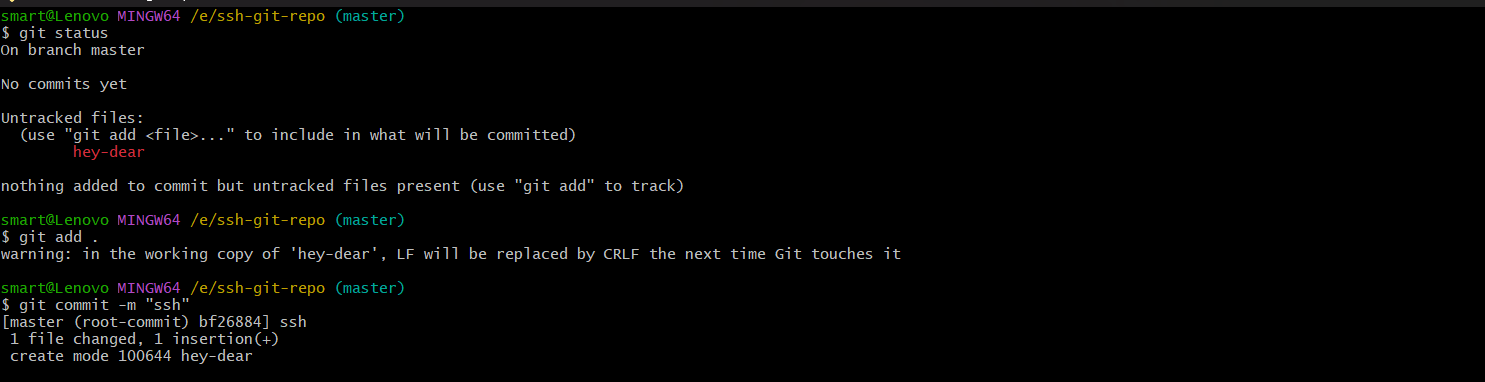
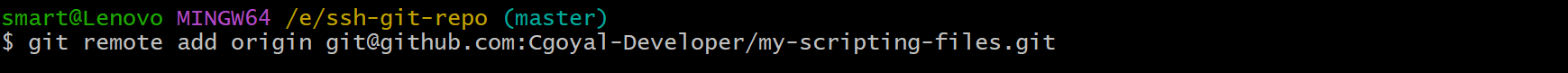
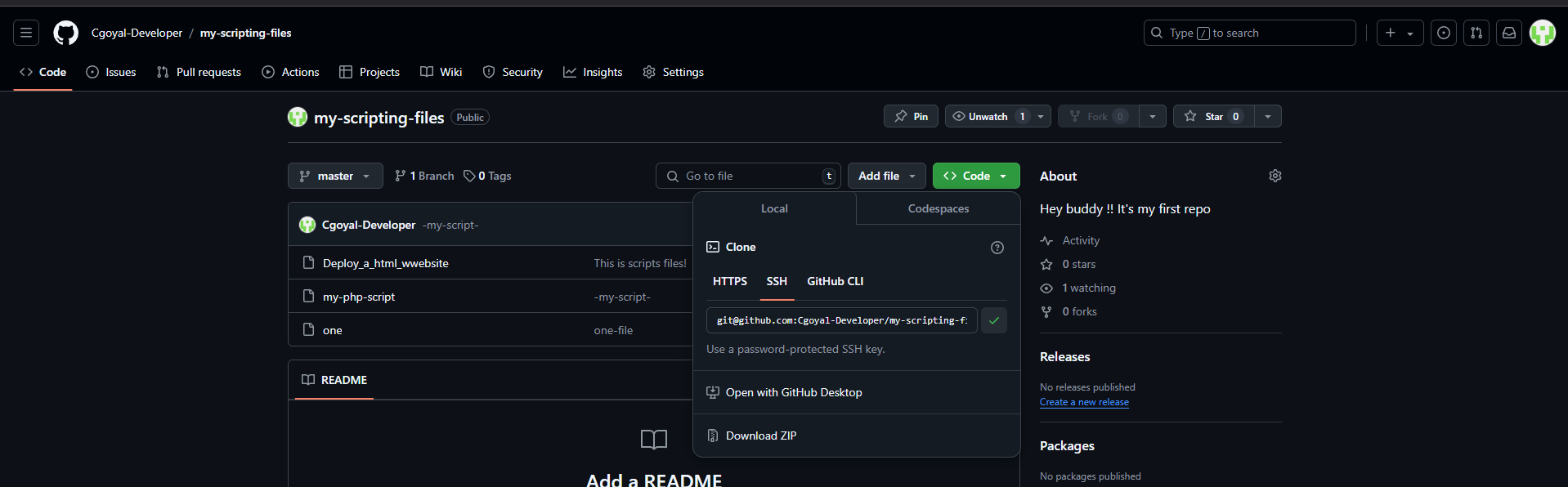


1. **Set up global configuration:**  
   git config –list  
     
   
2. **Initialize a new Git repository:**  
   git init  
     
   
3. **Create files in your local machine.**
4. **Check the status of your repository:**  
   git status  
     
   
5. **Add files to the staging area:**  
   git add <file-name>/< . >  
     
   
6. **Check the status again:**  
   git status  
     
   
7. **Copy the repository HTTPS link from GitHub.**
8. **Add the remote repository:**  
   git remote add origin <repo-link>  
     
   
9. **Commit the changes with a message:**  
   git commit -m "<message>"  
     
   
10. **Check the status:**  
    git status  
      
    
11. **Push changes to the remote repository:**  
    git push -u origin <branch-name>  
      
    
12. **Add a new file to the repository.**
13. **Modify or make changes to the file.**
14. **View logs:**  
    git log  
      
      
      
    git log –oneline  
      
    
15. **Check the repository configuration:**  
    cat .git/config  
      
    
16. **Clone a repository:**  
     1. Create a folder:  
     mkdir clone-files  
     2. Clone the repository:  
     git clone <git-hub-link>

**20-Nov-2024**

**Internship Day - 83 Report:**

**SSH and Branching:**

1. **Generate an SSH key:**  
   ssh-keygen  
   ****
2. **View SSH keys:**  
   ls .ssh/  
     
   
3. **Display the public key:**  
   cat .ssh/id\_ed25519.pub  
     
   
4. **Copy the key and add it to GitHub:**  
    1. Go to the repository settings in GitHub.  
    2. Click on 'SSH and GPG keys'.  
    3. Add a new SSH key with a title and paste the key.
5. **Create a folder for the SSH repository.**
6. **View all files including hidden ones:**  
   ls –a  
     
   
7. **Copy the SSH link of the repository.**

**Branches:**

### ****Introduction to Branches in GitHub****

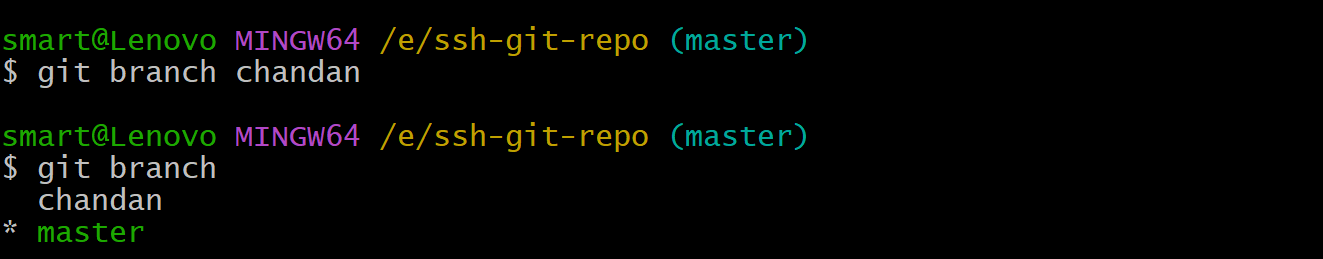
In GitHub (and Git), **branches** are a core feature used to manage different versions of a codebase. A branch represents an independent line of development within a repository. They allow developers to work on new features, bug fixes, or experiments without affecting the main codebase.

### ****Key Concepts of Branches****

1. **Default Branch**:
   * Typically named main or master.
   * Represents the "production-ready" version of the code.
   * It’s the branch that other branches are usually merged into.
2. **Feature Development**:
   * Create branches for individual tasks or features, e.g., feature/login-page or bugfix/error-404.
   * Helps in isolating work and avoiding conflicts with other developers.
3. **Branching Workflow**:
   * Developers can switch between branches to work on different features simultaneously.
   * Each branch maintains its history of changes.
4. **Collaboration**:
   * Multiple team members can work on their own branches and later merge them into the main branch.
   * Encourages parallel development.

### ****Why Use Branches?****

* **Isolation**: Prevents unfinished code from affecting the stable version.
* **Collaboration**: Teams can work on different features independently.
* **Code Review**: Pull requests from branches allow for better code review processes.
* **Experimentation**: Test new ideas without disrupting the main codebase.

1. **Create a new branch:**  
   git branch <new-branch-name>  
   ****
2. **Switch to the new branch:**  
   git switch <branch-name>
3. **Push the branch to the remote repository:**  
   git push origin <new-branch-name>  
     
   ****
4. **Add an untracked file:**  
    1. Create a file using touch.  
    2. Stage the file:  
    git add .  
    3. View status:  
    git status  
    4. Unstage the file:  
    git restore --staged <file-name>  
    5. View status again:  
    git status
5. **Merge the new branch into the master branch.**
6. **Unmerge the new branch from the master branch.**
7. **Add a file and merge changes:**  
    **Use the command:**  
    git reset --hard <commit-id>

**21-Nov-2024**

**Internship Day - 84 Report:**

**Introduction to Maven Build Tool**

Maven is a tool designed to help with building software. It simplifies the process of turning source code into working programs. It automates tasks like compiling code, running tests, and packaging the program into a format that can be distributed.

Maven is emphasized as a build tool specifically designed for Java projects. It helps automate tasks such as compiling code, running tests, and creating packages (like JAR, WAR files).

**Why Do We Need Maven?**

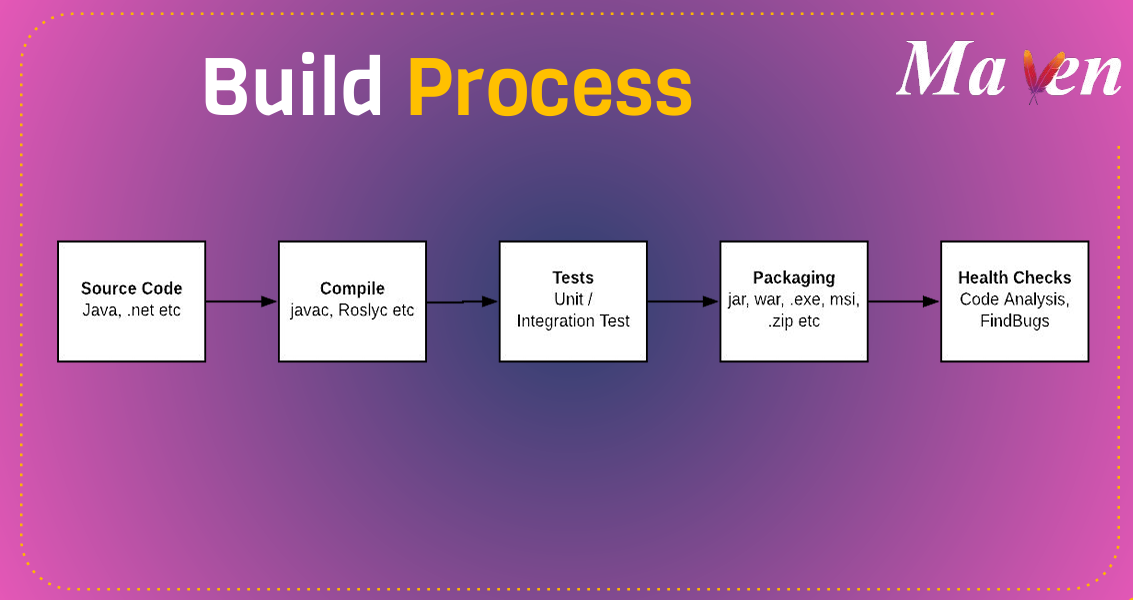
Maven simplifies and automates the software build process, managing dependencies, standardizing workflows, and ensuring efficient multi-project handling.

### **Overview of Build Tools**

This page describes different tools used for automating the software build process:

* **Maven:** Primarily for Java projects, it uses an XML file to manage the build process.
* **Ant:** Another tool for Java with XML configuration.
* **MsBuild:** A Microsoft tool for building applications.
* **Gradle:** Uses Groovy-based DSL (Domain-Specific Language) for configuration.
* **NANT:** A build tool for the Windows .NET platform.
* **Make:** Commonly used for creating executable programs from source code.

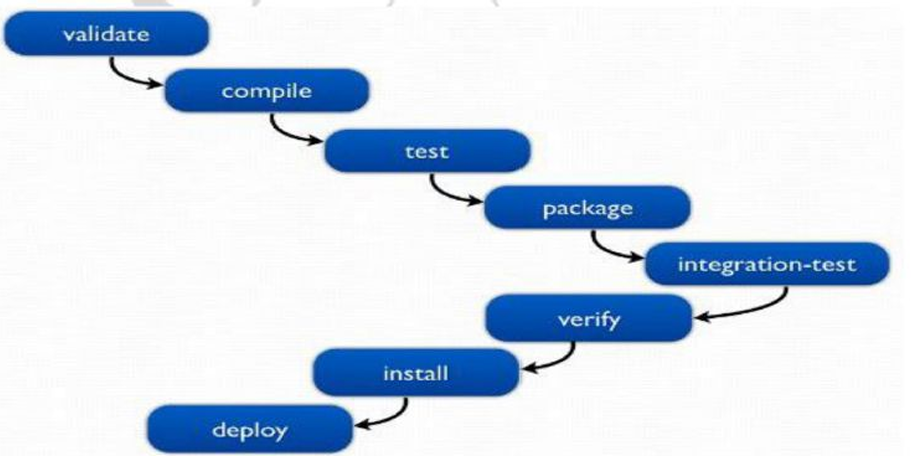
These tools make software development easier and more consistent by automating repetitive tasks.



**Maven Phases:**

Maven organizes its tasks into a series of steps called "phases." Each phase accomplishes a specific goal in the software build process:

1. **validate:** Ensures the project has all necessary information and is correct.
2. **compile:** Converts the source code into executable code.
3. **test:** Runs unit tests to check the code without deploying it.
4. **package:** Packages the compiled code into a distributable format (e.g., JAR).
5. **verify:** Ensures integration test results meet quality standards.
6. **install:** Saves the package in a local repository for use in other projects.
7. **deploy:** Uploads the package to a remote repository for sharing with others.



**Installing Maven**

To use Maven, you need to:

1. Install it using a package manager (like Chocolatey, Homebrew, yum, or apt) based on your operating system.
2. Have the **Java Development Kit (JDK)** installed on your system, as Maven depends on Java.

The process is simple and ensures developers can start using Maven efficiently.



**Step-by-Step Guide to Install Maven via CLI**

### ****Update the System Packages****

Before installing any software, it's crucial to ensure the system packages are updated.



### ****Install Java Development Kit (JDK)****

Java is required for Maven to function as Maven runs on the Java Virtual Machine (JVM).



### ****Install Apache Maven****

Maven is a build automation tool used for Java projects.

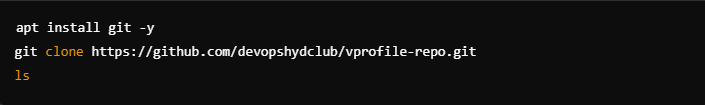
**Commands:**



### **Install Git and Clone the Repository**

Git is required to download the project repository.

**Commands:**



### ****Navigate to the Project Directory****

Change into the directory where the project files are stored.

**Commands:**



### ****Run Tests****

Use Maven to run the tests in the project.

**Commands:**



**Explanation:**

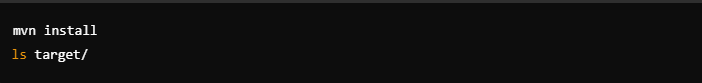
**mvn test:** Executes the tests defined in the project.

**ls target/:** Lists the contents of the target directory, where the test outputs are stored.

### ****Build the Project****

Compile and package the project into a distributable format.

**Commands:**



**Explanation:**

mvn install: Builds the project and generates a package (e.g., a JAR or WAR file).

ls target/: Confirms that the build artifacts are in the target directory.

### ****Clean the Build****

Remove the target directory to clean the build environment.

**Commands:**



1. **Validate Maven Project**

Validates that the project’s structure and dependencies are correct.



**22-Nov-2024**

**Internship Day - 85 Report:**

**What is Jenkins, Why Do We Need It, and an Explanation of Continuous Integration?**

**What is Jenkins?**

Jenkins is an **open-source automation server** widely used for **Continuous Integration (CI)** and **Continuous Delivery (CD)** in software development. It automates repetitive tasks involved in building, testing, and deploying applications, streamlining the software development process



**Why Do We Need Jenkins?**

1. **Automates Build and Deployment**:
   * Jenkins automates the process of compiling code, running tests, and deploying applications, saving time and reducing manual errors.
2. **Supports Continuous Integration**:
   * Ensures that code changes are continuously merged and tested, maintaining a stable codebase.
3. **Integration with Tools**:
   * Jenkins integrates with various tools like Git, Maven, Docker, Kubernetes, and others to manage the entire software lifecycle.
4. **Cross-Platform**:
   * Runs on Windows, macOS, Linux, and other platforms, providing flexibility for different environments.
5. **Extensible**:
   * Offers a wide variety of plugins (over 1,500) to customize and extend its functionality.
6. **Scalable**:
   * Can distribute builds across multiple machines to handle large-scale projects efficiently.

**Key Features of Jenkins**

1. **Pipeline as Code**:
   * Use **Jenkinsfiles** to define build pipelines as code, enabling better version control.
2. **Distributed Builds**:
   * Execute builds across multiple nodes, improving performance.
3. **Integration with SCM**:
   * Works seamlessly with Source Code Management (SCM) tools like Git, GitHub, or Bitbucket.
4. **Automated Testing**:
   * Runs automated tests to detect and fix issues early.
5. **Dashboard and Notifications**:
   * Provides a web-based dashboard for monitoring builds and integrates with communication tools for alerts.

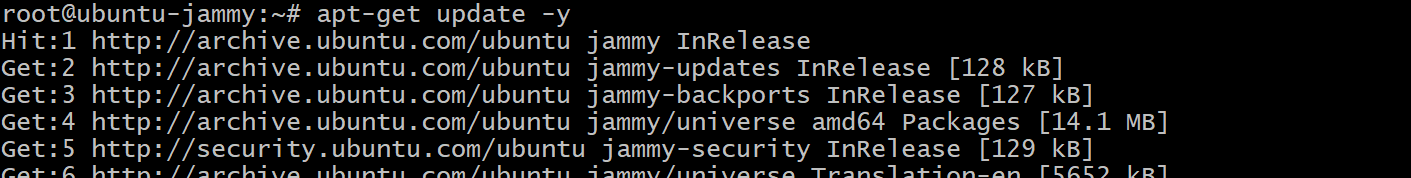
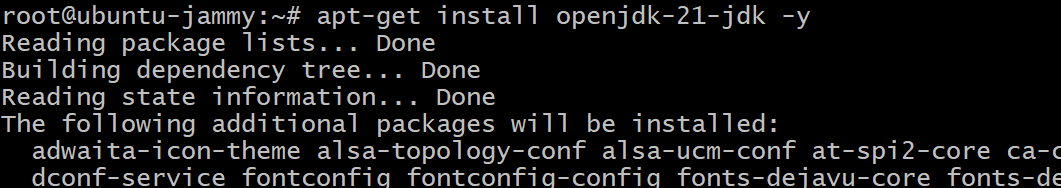
**What is Continuous Integration (CI)?**

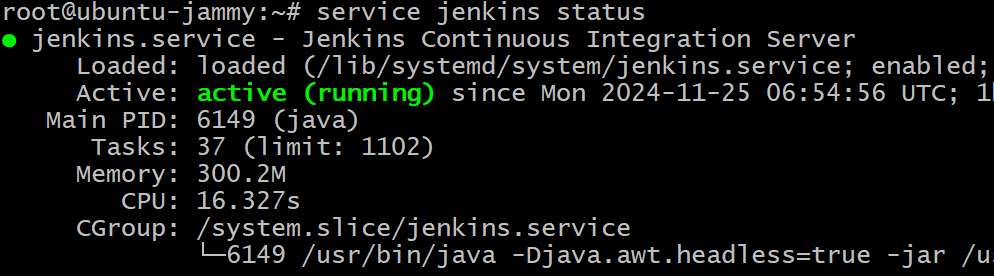
**Continuous Integration (CI)** is a software development practice where developers integrate code into a shared repository frequently, often multiple times a day. Automated builds and tests are triggered with every integration to detect issues early.

**Why is Continuous Integration Important?**

1. **Early Bug Detection**:
   * Frequent integrations catch bugs earlier in the development cycle, reducing costs and effort.
2. **Stable Codebase**:
   * Automated testing ensures that the main branch remains stable and deployable.
3. **Faster Development Cycles**:
   * Developers can focus on writing code instead of spending time on manual testing and   
     debugging.
4. **Improved Collaboration**:
   * Encourages teamwork by providing a centralized repository for all code changes.

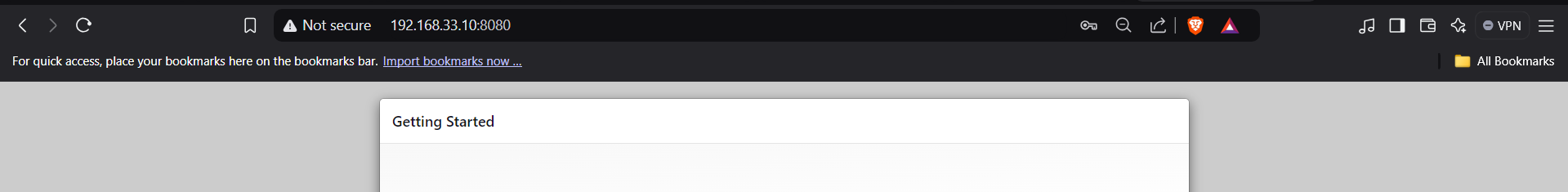
**Installation of Jenkins:**

1. **Switch to root:** sudo –i  
     
   
2. **Update packages:** apt-get update –y  
     
   
3. **Install Java:** apt-get install openjdk-21-jdk  
     
   
4. **Verify Java:** java –version  
     
   
5. **Download Jenkins GPG key:** sudo wget -O /usr/share/keyrings/jenkins-keyring.asc <https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key>
6. **Add Jenkins repo:** echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
7. **Update packages:** sudo apt-get update
8. **Install Jenkins:** sudo apt-get install jenkins.  
     
   
9. **Check Status: service Jenkins status**

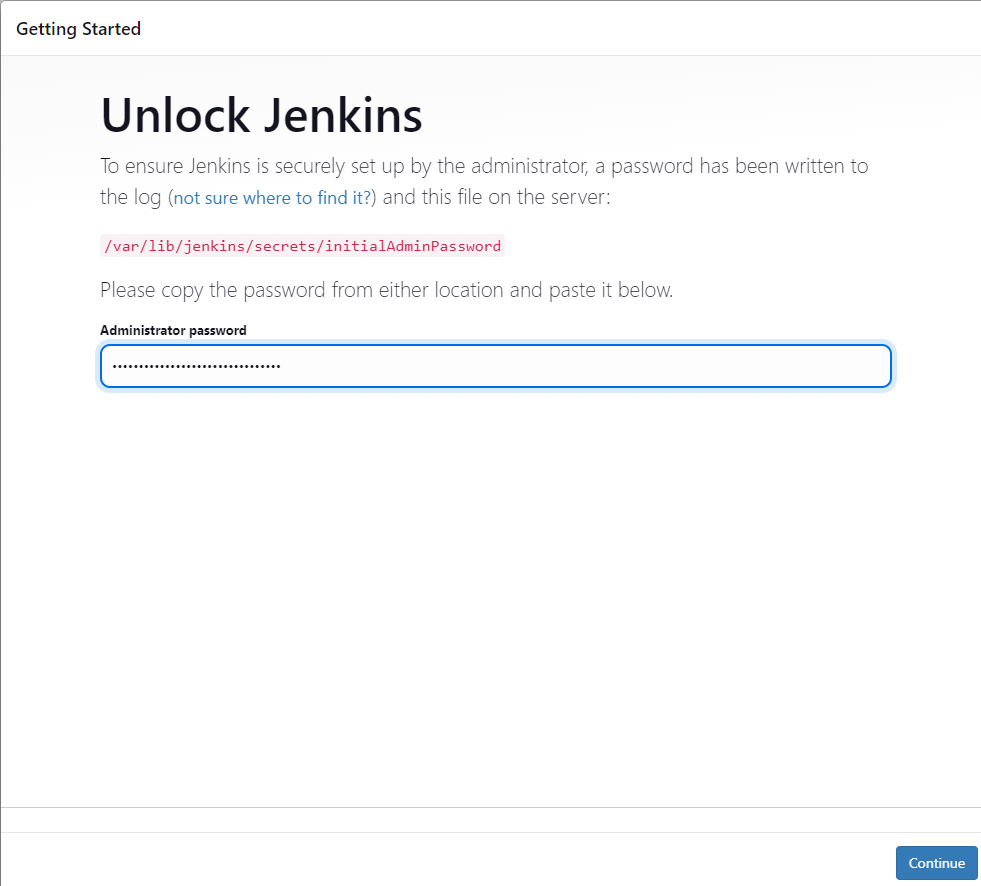


1. **Check IP Addr:** ip a s

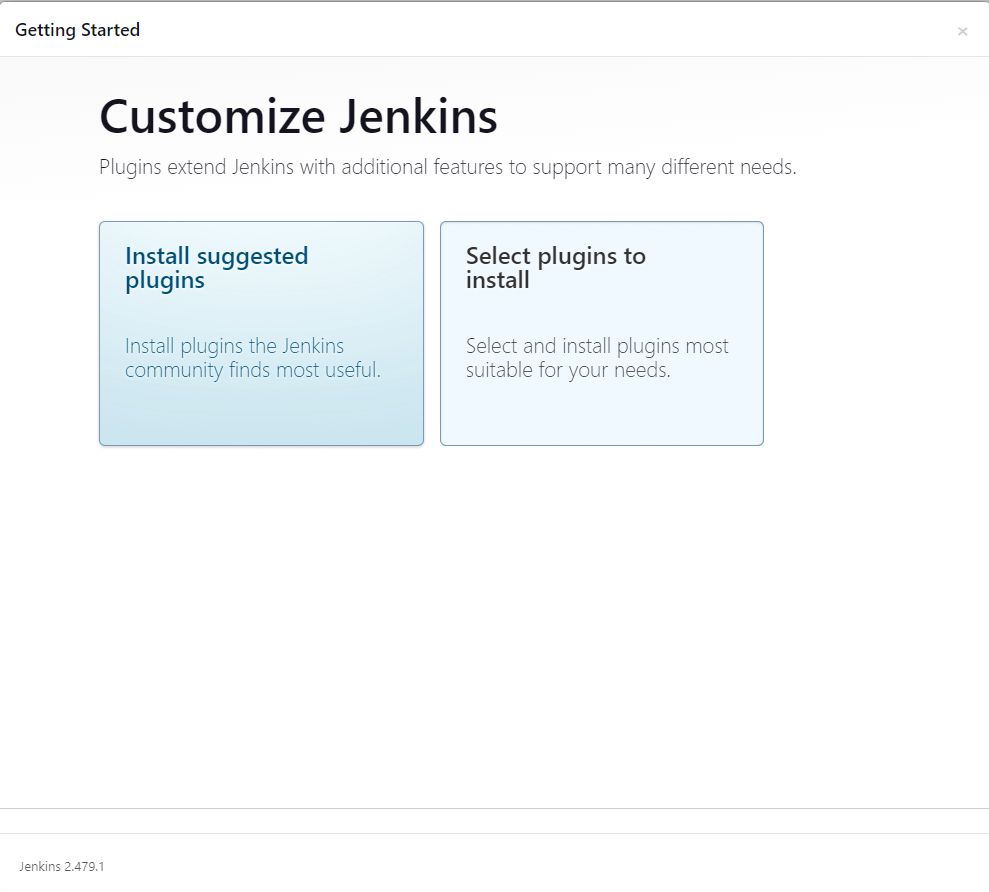


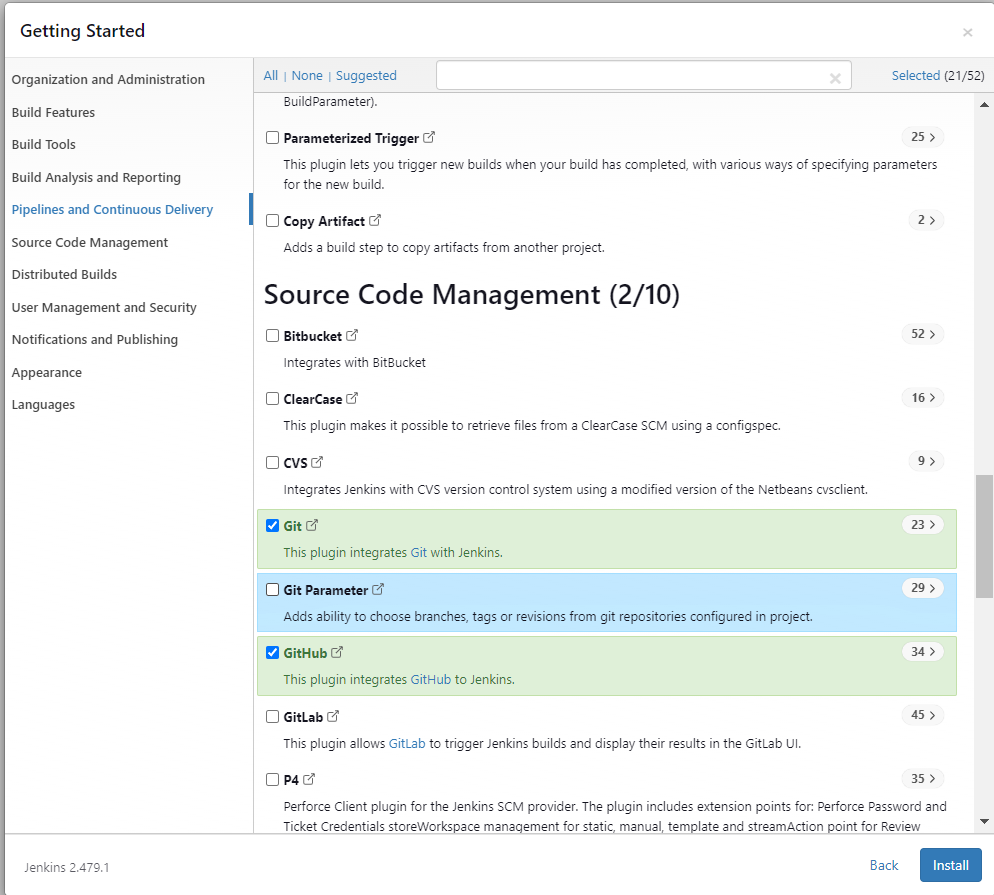
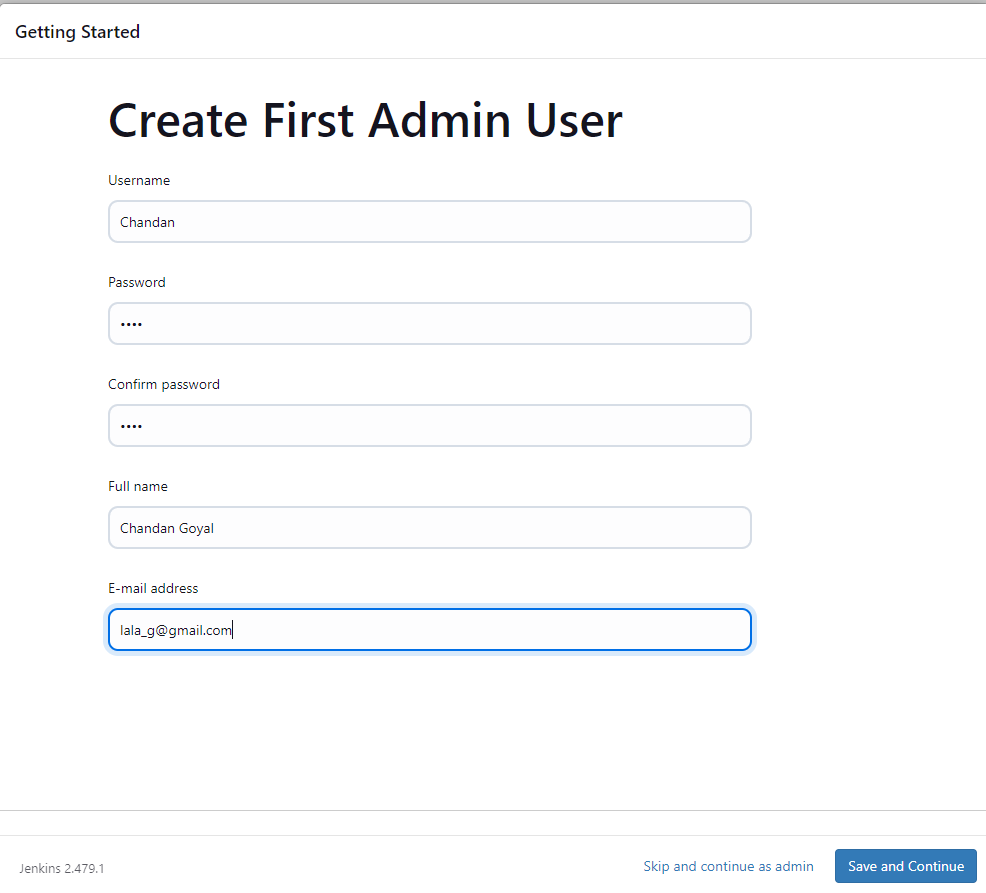
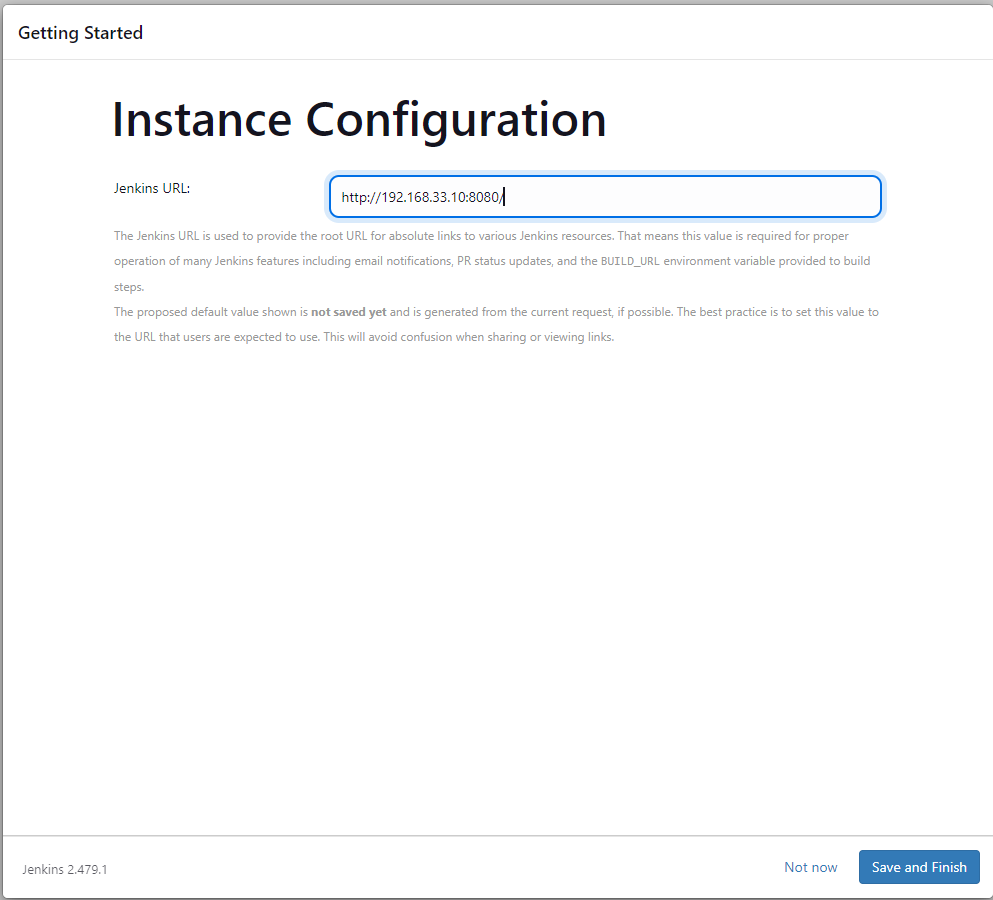
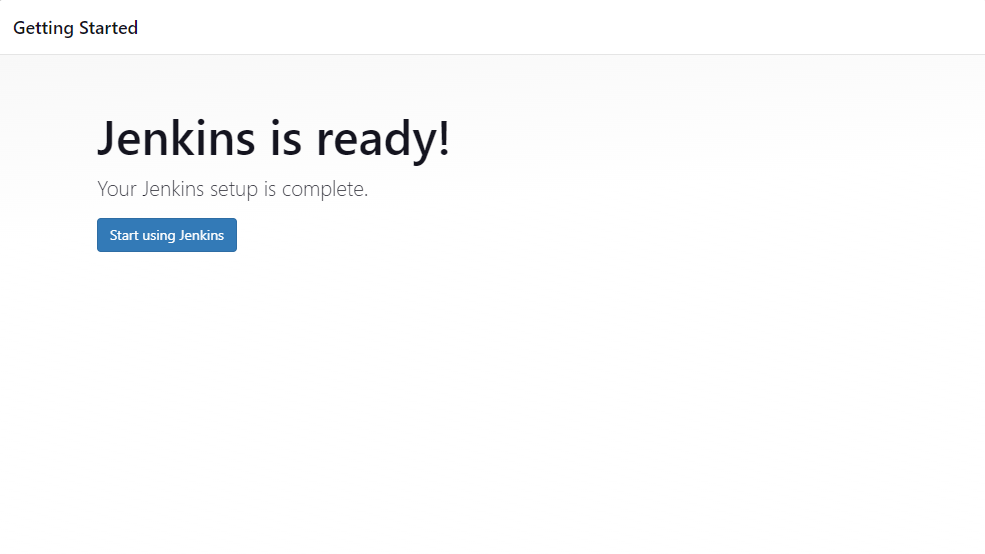
1. **Copy ip & paste it browser with part no 8080 :** See Jenkins Interface
2. **View password:** cat /var/lib/jenkins/secrets/initialAdminPassword  
     
   

1. **Paste it & Continue**



1. **Select Plugins to install**



1. **Select what you want to install and click to install**
2. **Create First Admin User**
3. **Save & Finish (if u want to change your url then change it)  
     
   **
4. **Click Start using Jenkins  
     
   **