

## 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:

```
for variable in item1 item2 item3
do
    # Commands to execute
done
```

- **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 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:

```
while [ condition ]
do
    # Commands to execute
done
```

## 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:

```
#!/bin/bash

for var1 in chandan cgoyal lala
do
    sleep 1

    echo "looping-----Start"
    echo
    echo $var1
    echo
    echo "-----"
done
```

#### Output:

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

```
looping-----Start
chandan
-----

looping-----Start
cgoyal
-----

looping-----Start
lala
-----
```

## Example of While Loop:

**Bash Script:** Input and Output

**Input (Script):**

```
#!/bin/bash

i=1

while [[ $i -le 10 ]]; do
    sleep 1
    echo "value is $i";
    ((i++))
done
```

**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:**

```
value is 1
value is 2
value is 3
value is 4
value is 5
value is 6
value is 7
value is 8
value is 9
value is 10
```

## 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.

## 1. Make a new repository in GitHub.


### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

---


*Required fields are marked with an asterisk (\*).*

**Owner \***

 Cgoyal-Developer ▾

**Repository name \***

my-scripting-files


 my-scripting-files is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-robot](#) ?


**Description** (optional)

Hey buddy !! It's my first repo

---

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

---

**Initialize this repository with:**

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more about READMEs.](#)

**Add .gitignore**

.gitignore template: **None** ▾


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

**Choose a license**

License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

---

 You are creating a public repository in your personal account.

---

Create repository

2. **Open Git Bash and check the Git version:**

git --version

```
smart@Lenovo MINGW64 ~  
$ git --version  
git version 2.46.2.windows.1
```

3. **Create a new folder in your system related to the repository.**

```
smart@Lenovo MINGW64 ~  
$ cd E:/  
  
smart@Lenovo MINGW64 /e  
$ mkdir my-scripts-repo  
  
smart@Lenovo MINGW64 /e  
$ cd my-scripts-repo/  
  
smart@Lenovo MINGW64 /e/my-scripts-repo  
$
```

```
smart@Lenovo MINGW64 /e/my-scripts-repo  
$ git config --global user.name "Cgoyal-Developer"  
  
smart@Lenovo MINGW64 /e/my-scripts-repo  
$ git config --global user.email "smartchandan141@gmail.com"  
  
smart@Lenovo MINGW64 /e/my-scripts-repo  
$ |
```

4. **Set up global configuration:**

git config --list

```
smart@Lenovo MINGW64 /e/my-scripts-repo  
$ git config --list  
diff.astextplain.textconv=astextplain  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
http.sslbackend=openssl  
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt  
core.autocrlf=true  
core.fscache=true  
core.symlinks=false  
pull.rebase=false  
credential.helper=manager  
credential.https://dev.azure.com.usehttppath=true  
init.defaultbranch=master  
user.name=Cgoyal-Developer  
user.email=smartchandan141@gmail.com
```

5. **Initialize a new Git repository:**

git init

```
smart@Lenovo MINGW64 /e/my-scripts-repo
$ git init
Initialized empty Git repository in E:/my-scripts-repo/.git/
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$
```

## 6. Create files in your local machine.

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ touch Deploy_a_html_website
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ vi my-php-script
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ ls
Deploy_a_html_website  my-php-script
```

## 7. Check the status of your repository:

git status

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Deploy_a_html_website
        my-php-script

nothing added to commit but untracked files present (use "git add" to track)
```

## 8. Add files to the staging area:

git add <file-name>/< . >

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git add my-php-script
warning: in the working copy of 'my-php-script', LF will be replaced by CRLF the next time Git touches it
```

## 9. Check the status again:

git status



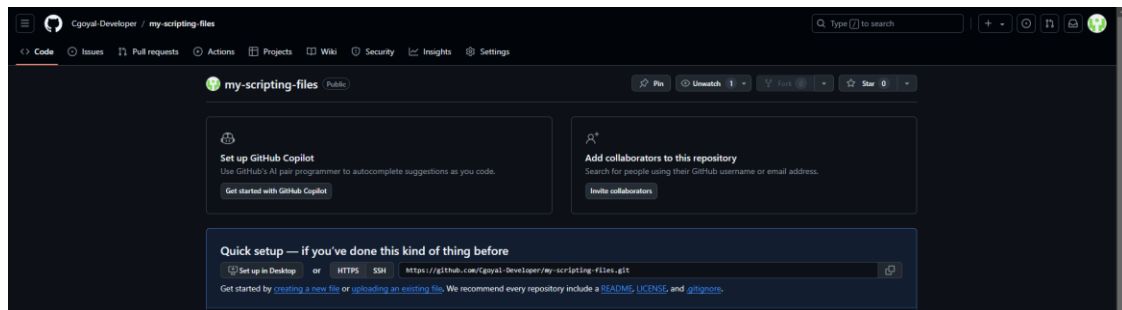
```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   my-php-script

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Deploy_a_html_website
```

## 10. Copy the repository HTTPS link from GitHub.



## 11. Add the remote repository:

git remote add origin <repo-link>

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git remote add origin https://github.com/Cgoyal-Developer/my-scripting-files.git
```

## 12. Commit the changes with a message:

git commit -m "<message>"

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git commit -m "This is scripts files!"
[master (root-commit) 2a9db95] This is scripts files!
2 files changed, 1 insertion(+)
create mode 100644 Deploy_a_html_website
create mode 100644 my-php-script
```

## 13. Check the status:

git status

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

#### 14. Push changes to the remote repository:

`git push -u origin <branch-name>`

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 298 bytes | 298.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Cgoyal-Developer/my-scripting-files.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

#### 15. Add a new file to the repository.

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ touch one

smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git add .

smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git commit -m "one-file"
[master d947de2] one-file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 one

smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 259 bytes | 259.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Cgoyal-Developer/my-scripting-files.git
 2a9db95..d947de2 master -> master
```

#### 16. Modify or make changes to the file.

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ ls
Deploy_a_html_website my-php-script one

smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ vi my-php-script

smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git add .
warning: in the working copy of 'my-php-script', LF will be replaced by CRLF the next time Git touches it

smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git commit -m "-my-script-"
[master 65b0f3f] -my-script-
1 file changed, 1 insertion(+), 1 deletion(-)

smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 319 bytes | 319.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Cgoyal-Developer/my-scripting-files.git
 d947de2..65b0f3f master -> master
```

## 17. View logs:

git log

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git log
commit 65b0f3fb661dca947ef9db87239b09783dfe8a4d (HEAD -> master, origin/master)
Author: Cgoyal-Developer <smartchandan141@gmail.com>
Date: Thu Nov 21 10:11:33 2024 +0530

    -my-script-

commit d947de24568592efc32d2f1462ebc3f91b479ced
Author: Cgoyal-Developer <smartchandan141@gmail.com>
Date: Thu Nov 21 10:07:40 2024 +0530

    one-file
```

git log --oneline

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ git log --oneline
65b0f3f (HEAD -> master, origin/master) -my-script-
d947de2 one-file
2a9db95 This is scripts files!
```

## 18. Check the repository configuration:

cat .git/config

```
smart@Lenovo MINGW64 /e/my-scripts-repo (master)
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true
[remote "origin"]
    url = https://github.com/Cgoyal-Developer/my-scripting-files.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
```

## 19. 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

```
smart@Lenovo MINGW64 ~
$ ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/smart/.ssh/id_ed25519):
Enter passphrase for "/c/Users/smart/.ssh/id_ed25519" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/smart/.ssh/id_ed25519
Your public key has been saved in /c/Users/smart/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:d1DrWiG/NedD+d0cdB8Us0SFzvbIfQSu+ka1OL1a/H4 smart@Lenovo
The key's randomart image is:
+--[ED25519 256]--+
|      . . * o |
|      . . + + |
|      o o + + |
|      = . B o |
|      S . =Bo@o|
|      . =*o*=@|
|      . o + o +B|
|      . . o . E|
|      . + . o . |
+-----[SHA256]-----+
```

#### 2. View SSH keys:

ls .ssh/

```
smart@Lenovo MINGW64 ~
$ ls .ssh/
id_ed25519  id_ed25519.pub
```

#### 3. Display the public key:

cat .ssh/id\_ed25519.pub

```
smart@Lenovo MINGW64 ~
$ cat .ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKkzG39PhEQ8DbHhG8vmXfKUKr0n4XQXko9WkywDhAf0 smart@Lenovo
```

#### 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.

```

smart@Lenovo MINGW64 ~
$ cd e:/

smart@Lenovo MINGW64 /e
$ mkdir ssh-git-repo

smart@Lenovo MINGW64 /e
$ cd ssh-git-repo/

smart@Lenovo MINGW64 /e/ssh-git-repo
$ git init
Initialized empty Git repository in E:/ssh-git-repo/.git/

```

## 6. View all files including hidden ones:

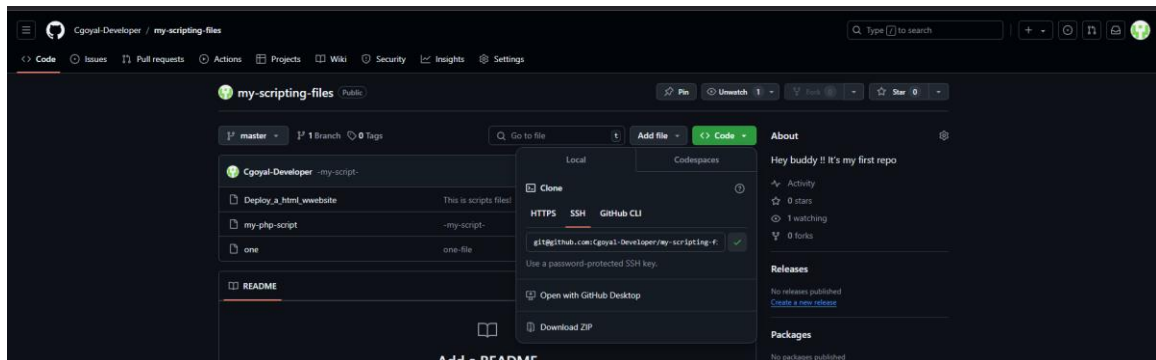
ls -a

```

smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ ls -a
./ ../ .git/

```

## 7. Copy the SSH link of the repository.



```

smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ git remote add origin git@github.com:Cgoyal-Developer/my-scripting-files.git

smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hey-dear

nothing added to commit but untracked files present (use "git add" to track)

smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ git add .
warning: in the working copy of 'hey-dear', LF will be replaced by CRLF the next time Git touches it

smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ git commit -m "ssh"
[master (root-commit) bf26884] ssh
1 file changed, 1 insertion(+)
create mode 100644 hey-dear

smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ git push -u origin master --force
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 222 bytes | 222.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Cgoyal-Developer/my-scripting-files.git
+ 65b0f3f..bf26884 master -> master (forced update)
branch 'master' set up to track 'origin/master'.

```

## 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>`

```
smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ git branch chandan

smart@Lenovo MINGW64 /e/ssh-git-repo (master)
$ git branch
chandan
* master
```

**2) Switch to the new branch:**

`git switch <branch-name>`

**3) Push the branch to the remote repository:**

`git push origin <new-branch-name>`

```
smart@Lenovo MINGW64 /e/ssh-git-repo (chandan)
$ git push origin chandan
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (2/2), 228 bytes | 228.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'chandan' on GitHub by visiting:
remote:   https://github.com/Cgoyal-Developer/my-scripting-files/pull/new/chandan
remote:
To github.com:Cgoyal-Developer/my-scripting-files.git
 * [new branch]      chandan -> chandan

smart@Lenovo MINGW64 /e/ssh-git-repo (chandan)
$ |
```

**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?

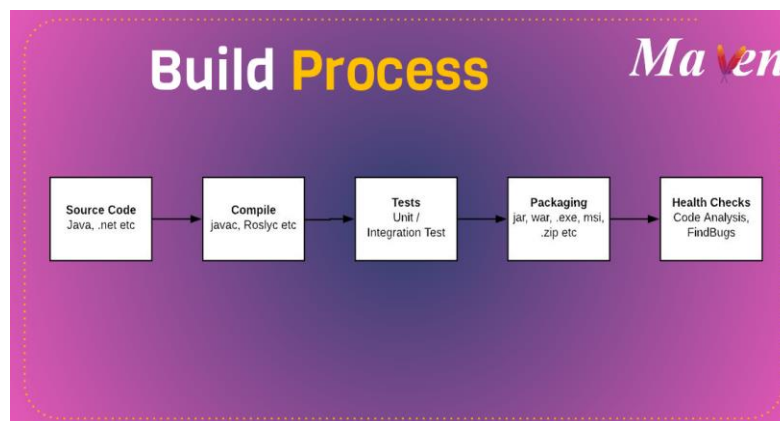
Maven simplifies and automates the software build process, managing dependencies, standardizing workflows, and ensuring efficient multi-project handling.

### 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.

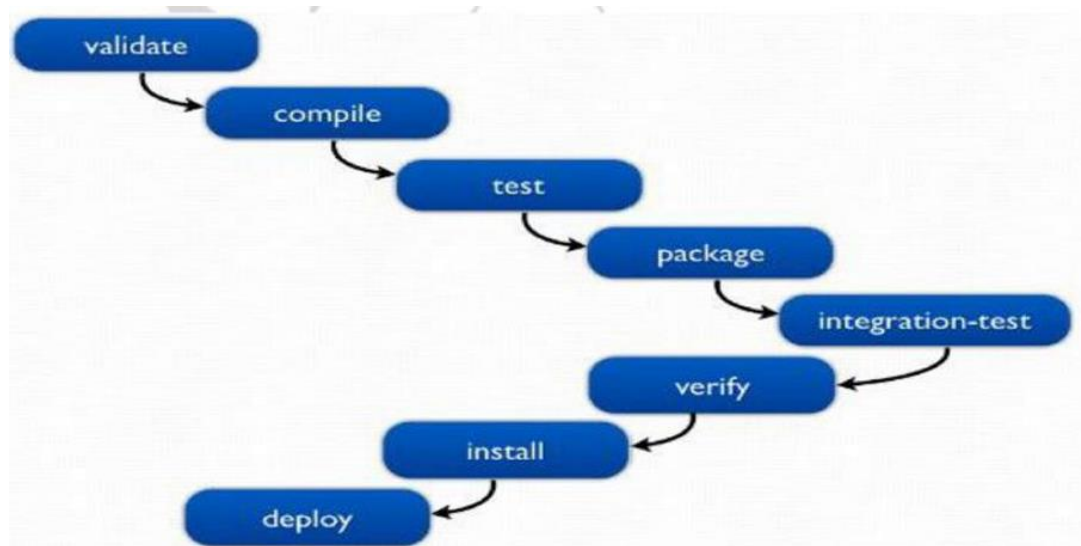




## Maven Phases:

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.

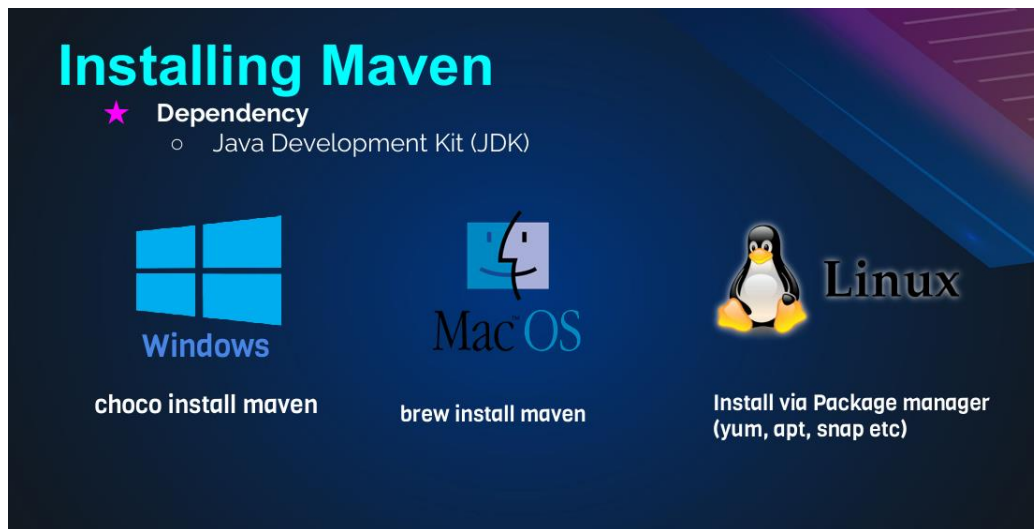


## Installing Maven

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

### 1) Update the System Packages

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

```
sudo -i  
apt-get update -y
```

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

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

```
java -version  
apt install openjdk-8-jdk -y  
java -version
```

### 3) Install Apache Maven

Maven is a build automation tool used for Java projects.

**Commands:**

```
apt install maven -y
mvn -version
```

#### 4) Install Git and Clone the Repository

Git is required to download the project repository.

##### Commands:

```
apt install git -y
git clone https://github.com/devopshydclub/vprofile-repo.git
ls
```

#### 5) Navigate to the Project Directory

Change into the directory where the project files are stored.

##### Commands:

```
cd vprofile-repo/
ls
```

#### 6) Run Tests

Use Maven to run the tests in the project.

##### Commands:

```
mvn test
ls
ls target/
```

##### 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.

## 7) Build the Project

Compile and package the project into a distributable format.

### Commands:

```
mvn install  
ls target/
```

### 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.

## 8) Clean the Build

Remove the target directory to clean the build environment.

### Commands:

```
rm -rf target  
ls  
(or)  
mvn clean  
ls
```

## 9) Validate Maven Project

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

```
mvn validate
```

## 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`

```
vagrant@ubuntu-jammy:~$ sudo -i
```

2. **Update packages:** `apt-get update -y`

```
root@ubuntu-jammy:~# apt-get update -y
Hit:1 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
```

3. **Install Java:** `apt-get install openjdk-21-jdk`

```
root@ubuntu-jammy:~# apt-get install openjdk-21-jdk -y
```

4. **Verify Java:** `java -version`

```
root@ubuntu-jammy:~# 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.`

```
root@ubuntu-jammy:~# sudo apt-get install jenkins
```

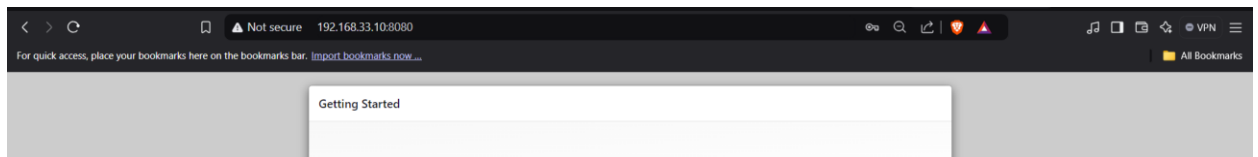
9. **Check Status:** `service Jenkins status`

```
root@ubuntu-jammy:~# service jenkins status
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled;
   Active: active (running) since Mon 2024-11-25 06:54:56 UTC; 1
 Main PID: 6149 (java)
    Tasks: 37 (limit: 1102)
   Memory: 300.2M
      CPU: 16.327s
   CGroup: /system.slice/jenkins.service
           └─6149 /usr/bin/java -Djava.awt.headless=true -jar /u
```

10. Check IP Addr: ip a s

```
root@ubuntu-jammy:~# ip a s
```

11. Copy ip & paste it browser with port no 8080 : See Jenkins Interface



12. View password: cat /var/lib/jenkins/secrets/initialAdminPassword

```
root@ubuntu-jammy:~# cat /var/lib/jenkins/secrets/initialAdminPassword
f13a8b9e4a854f9bb28e5206a91c658d
```



### 13. Paste it & Continue

Getting Started

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

### 14. Select Plugins to install

Getting Started

## Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

#### Install suggested plugins

Install plugins the Jenkins community finds most useful.

#### Select plugins to install

Select and install plugins most suitable for your needs.

## 15. Select what you want to install and click to install

Getting Started

Organization and Administration

Build Features

Build Tools

Build Analysis and Reporting

Pipelines and Continuous Delivery

Source Code Management

Distributed Builds

User Management and Security

Notifications and Publishing

Appearance

Languages

All | None | Suggested

BuildParameter).

☐ Parameterized Trigger

This plugin lets you trigger new builds when your build has completed, with various ways of specifying parameters for the new build.

25 >

☐ Copy Artifact

Adds a build step to copy artifacts from another project.

2 >

### Source Code Management (2/10)

☐ Bitbucket

Integrates with BitBucket

52 >

☐ ClearCase

This plugin makes it possible to retrieve files from a ClearCase SCM using a configspec.

16 >

☐ CVS

Integrates Jenkins with CVS version control system using a modified version of the Netbeans cvsclient.

9 >

☒ Git

This plugin integrates Git with Jenkins.

23 >

☐ Git Parameter

Adds ability to choose branches, tags or revisions from git repositories configured in project.

29 >

☒ GitHub

This plugin integrates GitHub to Jenkins.

34 >

☐ GitLab

This plugin allows GitLab to trigger Jenkins builds and display their results in the GitLab UI.

45 >

☐ P4

Perforce Client plugin for the Jenkins SCM provider. The plugin includes extension points for: Perforce Password and Ticket Credentials storeWorkspace management for static, manual, template and streamAction point for Review

35 >

Selected (21/52)

Back

Install

Jenkins 2.479.1

## 16. Create First Admin User

Getting Started

### Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins 2.479.1 [Skip and continue as admin](#) [Save and Continue](#)

## 17. Save & Finish (if u want to change your url then change it)

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 2.479.1 [Not now](#) [Save and Finish](#)

## 18. Click Start using Jenkins

