## *Institute's Vision*

To be an organization with potential for excellence in engineering and management for the advancement of society and human kind.

## *Institute's Mission*

To excel in academics, practical engineering, management and to commence research endeavors.

To prepare students for future opportunities.

To nurture students with social and ethical responsibilities.

## *Department's Vision*

To create IT graduates with ethical and employable skills.

## *Department's Mission*

To imbibe problem solving and analytical skills through teaching learning process.

To impart technical and managerial skills to meet the industry requirement.

To encourage ethical and value based education.

Excelssior's Education Society

# K. C. COLLEGE OF ENGINEERING AND MANAGEMENT STUDIES AND RESEARCH THANE (EAST).

## Certificate

This is to certify that Mr. / Ms. _____

of Semester _____ Branch _____ Roll No. _____

has performed and successfully completed all the practical's in the subject of

_____ for the academic

year 20____to 20____as prescribed by University of Mumbai.

DATE :-_____.

_____

Practical Incharge

_____

Internal Examiner

_____

Head of Department

_____

External Examiner

COLLEGE SEAL

## SYLLABUS

| Sr. No. | Module | Detailed Content | Hours | LO Mapping |
|---|---|---|---|---|
| 0 | Prerequisite | To Understand the Concept of DevOps with related technologies which are used to Code, Build, Test, Configure & Monitor the Software Applications. | 2 | --- |
| I | Build & Test Applications With Continuous Integration | To Install and Configure Jenkins to test, and deploy Java or Web Applications using Netbeans or eclipse. | 4 | ITL803.1<br>ITL803.2 |
| II | Version Control | To Perform Version Control on websites/ Softwares using different Version control tools like RCS/ CVS/GIT/Mercurial (Any two) | 4 | ITL803.1<br>ITL803.3 |
| III | Virtualization & Containerization | To Install and Configure Docker for creating Containers of different Operating System Images | 4 | ITL803.1<br>ITL803.4 |
| IV | Virtualization & Containerization | To Build, deploy and manage web or Java application on Docker | 4 | ITL803.1<br>ITL803.4 |
| V | Software Configuration Management | To install and configure Software Configuration Management using Chef/Puppet/Ansible or Saltstack. | 4 | ITL803.1<br>ITL803.5 |
| VI | Provisioning | To Perform Software Configuration Management and provisioning using Chef/Puppet/Ansible or Saltstack. | 4 | ITL803.1<br>ITL803.6 |

# Program Outcomes

Engineering Graduates will be able to:

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

**2. 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.

**3. 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.

**4. 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.

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

**6. 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.

**7. 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.

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

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

**10. 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.

**11. 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.

**12. 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.

## Department of Information Technology

**Course : DevOps LAB**

**Semester: VIII**

**Class :BEIT**

# Course Outcomes / Lab Outcomes

| Course Code | Course Outcomes |
|---|---|
| | **On successful completion, of course, learner/student will be able to:** |
| ITL803.1 | Remember the importance of DevOps tools used in software development life cycle. |
| ITL803.2 | Understand the importance of Jenkins to Build, Deploy and Test Software Applications. |
| ITL803.3 | Examine the different Version Control strategies. |
| ITL803.4 | Analyze & Illustrate the Containerization of OS images and deployment of applications over Docker. |
| ITL803.5 | Summarize the importance of Software Configuration Management in DevOps. |
| ITL803.6 | Synthesize the provisioning using Chef/Puppet/Ansible or Saltstack. |

# Rubrics for Practical

| Rubrics Description | Maximum Marks Weight | 15-12 | 12-9 | 9-6 | 6-0 |
|---|---|---|---|---|---|
| **Implementation (R1)** | 5 | Successful completion with accurate output (5-4) | Output correct but not precise (4-3) | Few errors in the output (3-2) | Incorrect Output (2-0) |
| **Understanding (R2)** | 5 | Understanding Experiment and drawn correct conclusion (5-4) | Understand Experiment but conclusion less accurate (4-3) | Improper Conclusion (3-2) | No Conclusion (2-0) |
| **Punctuality and Discipline (R3)** | 5 | Submission within a week (5-4) | Submission after week (4-3) | Submission after two weeks (3-2) | Submission after three weeks and more (2-0) |

# TABLE OF CONTENTS

| Sr. No | Name of Experiment | Date of Conduction | Date of Submission | Page No. | Grade / Marks | Sign |
|---|---|---|---|---|---|---|
| | | | | | | |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |
| 6 | | | | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |

## Total Grade / Marks :-

| Avg. marks of Mini Project (A) | | Avg. marks of Assignments (B) | | Total Marks (A+B) |
|---|---|---|---|---|
| Obtained | Out of | Obtained | Out of | |
| | | | | |

_____

Practical Incharge

_____

Date

## EXPERIMENT NO. - 01

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

Practical Incharge

# PRACTICAL NO. 1

***Problem Definition:*** To perform installation of Git and work on local and remote git repositories.

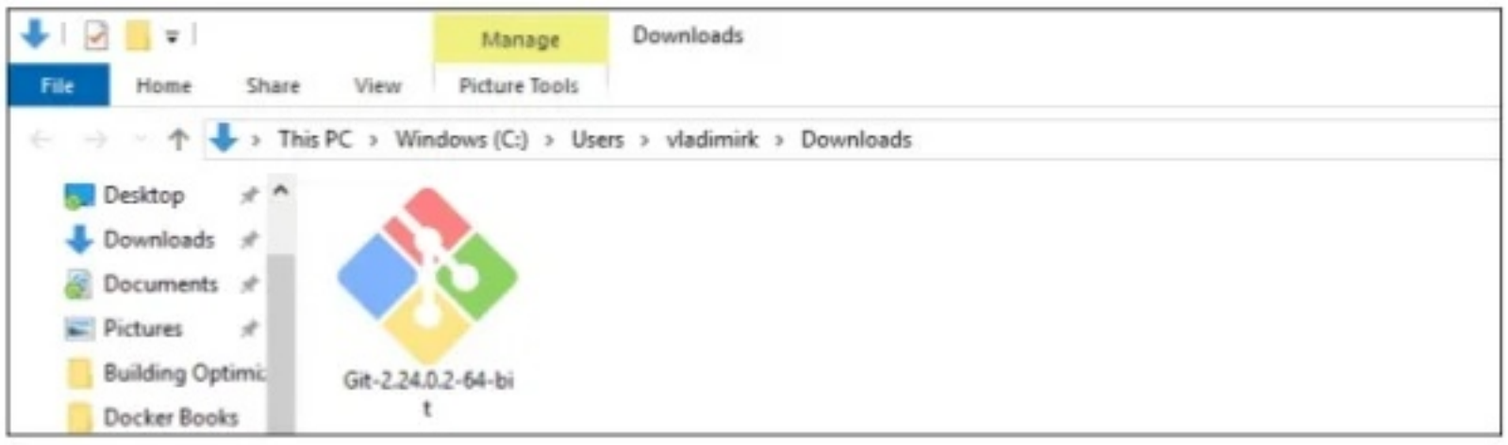
***Compiler / Tool:*** Git-2.35.1.2-64-bit

***Installation:***

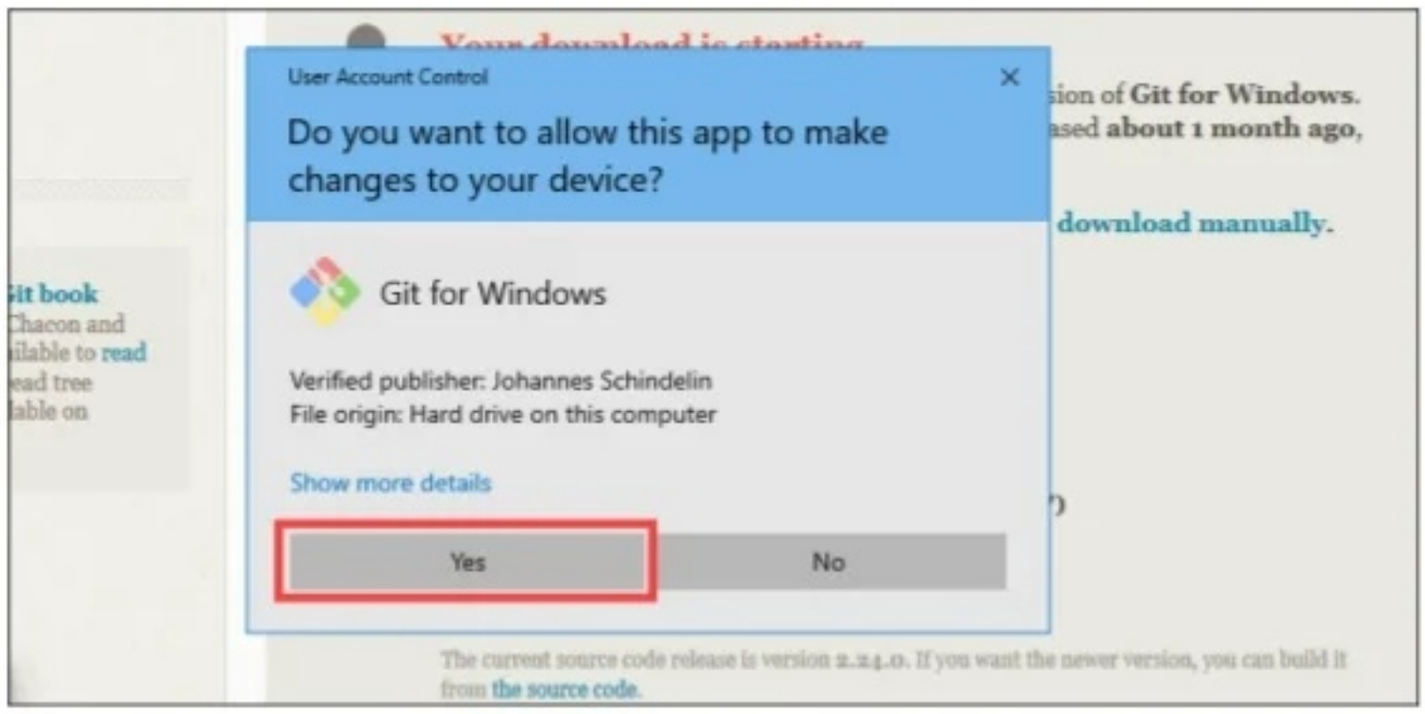
1. Browse to the official Git website: https://git-scm.com/downloads
2. Click the download link for Windows and allow the download to complete.


**Extract and Launch Git Installer**

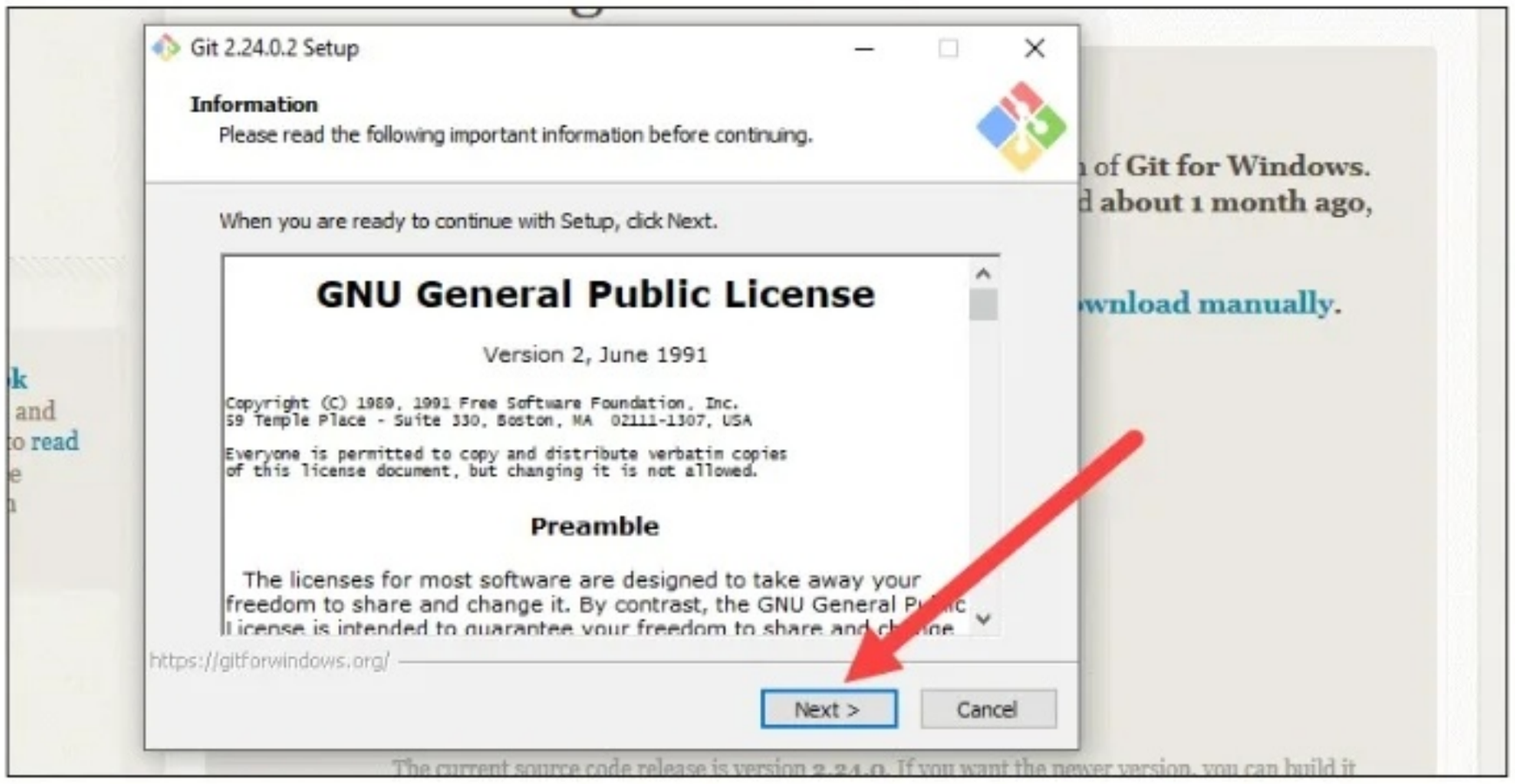
3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.



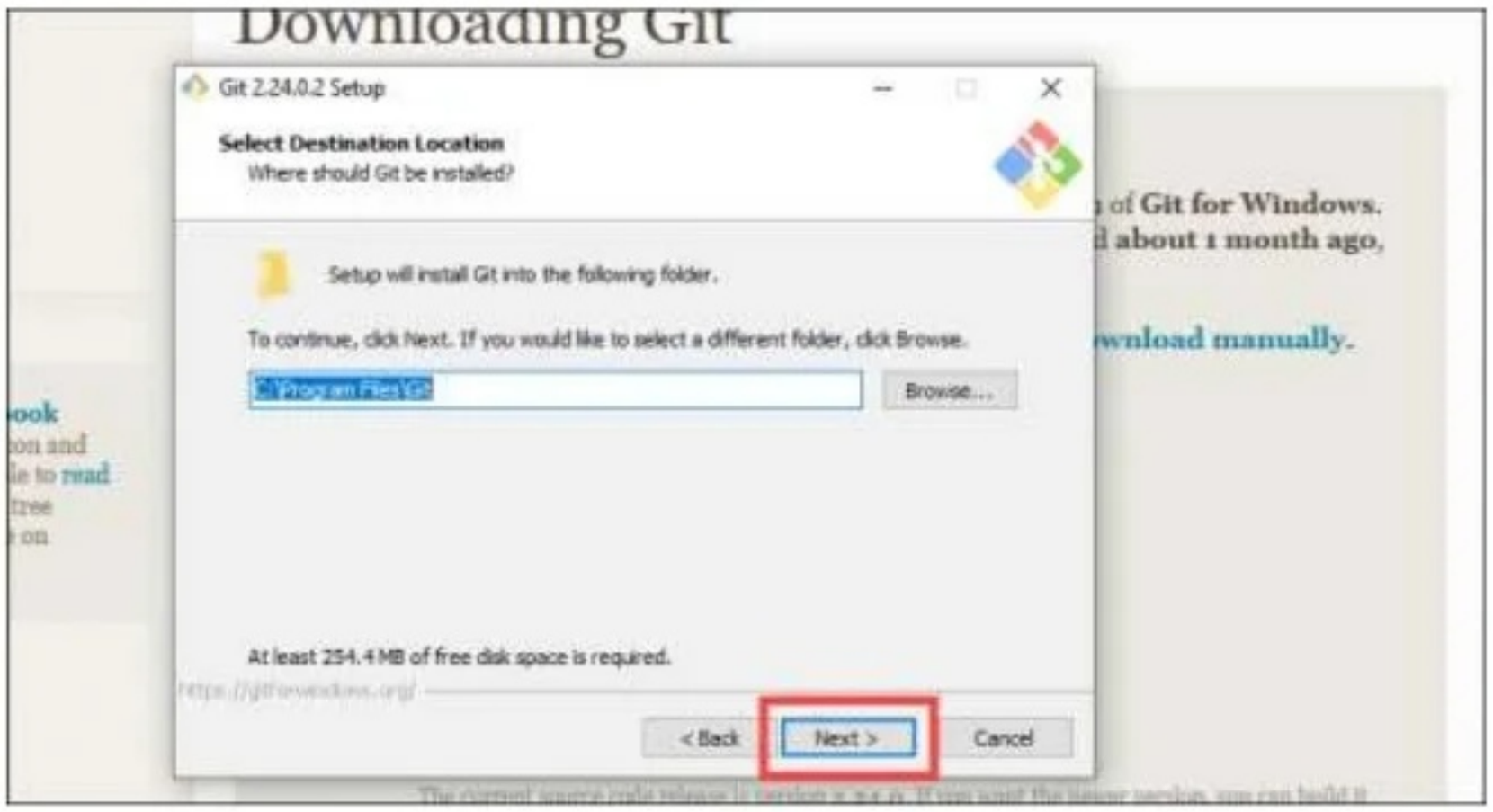

4. Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens.



5. Review the GNU General Public License, and when you're ready to install, click **Next**.

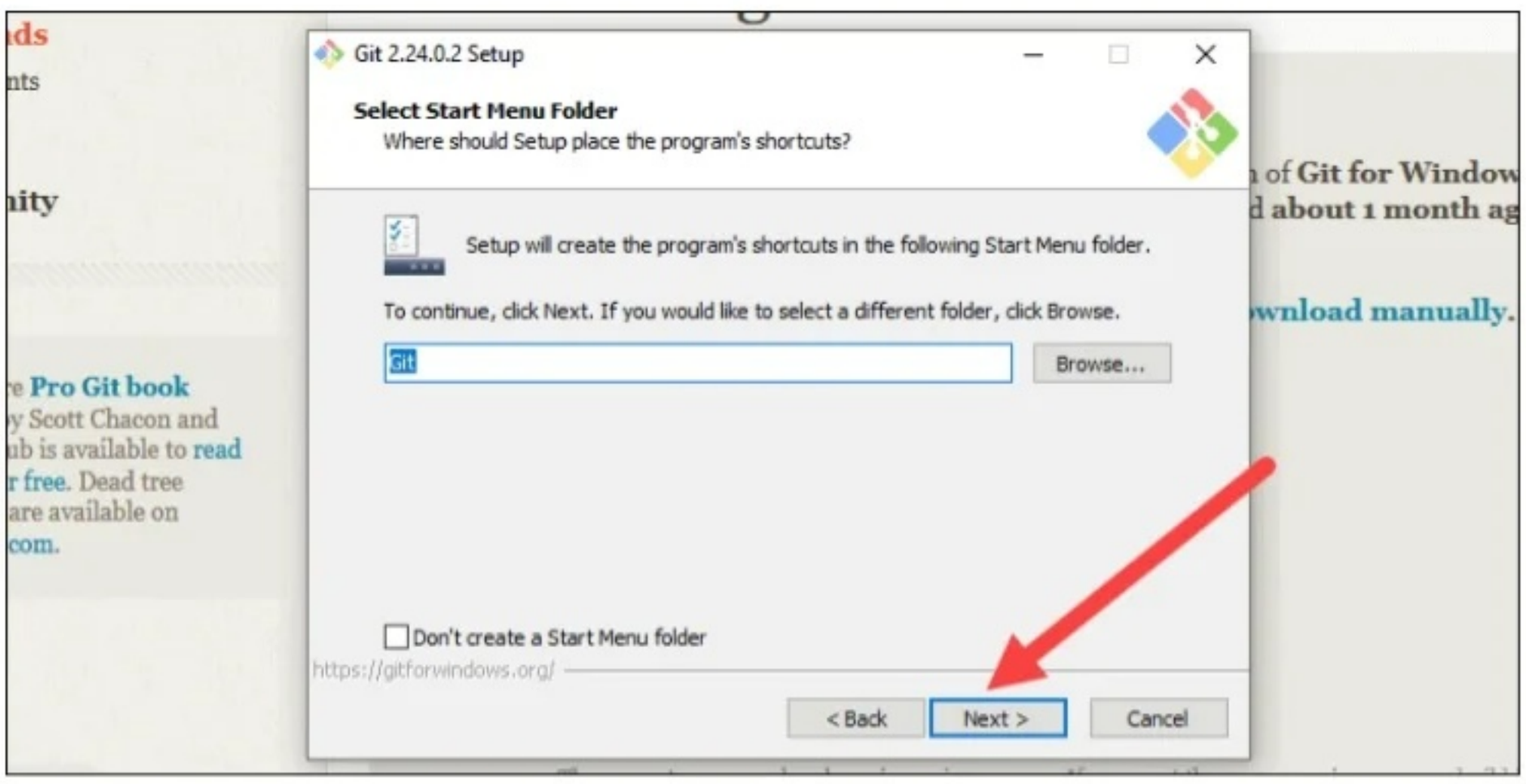



6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.

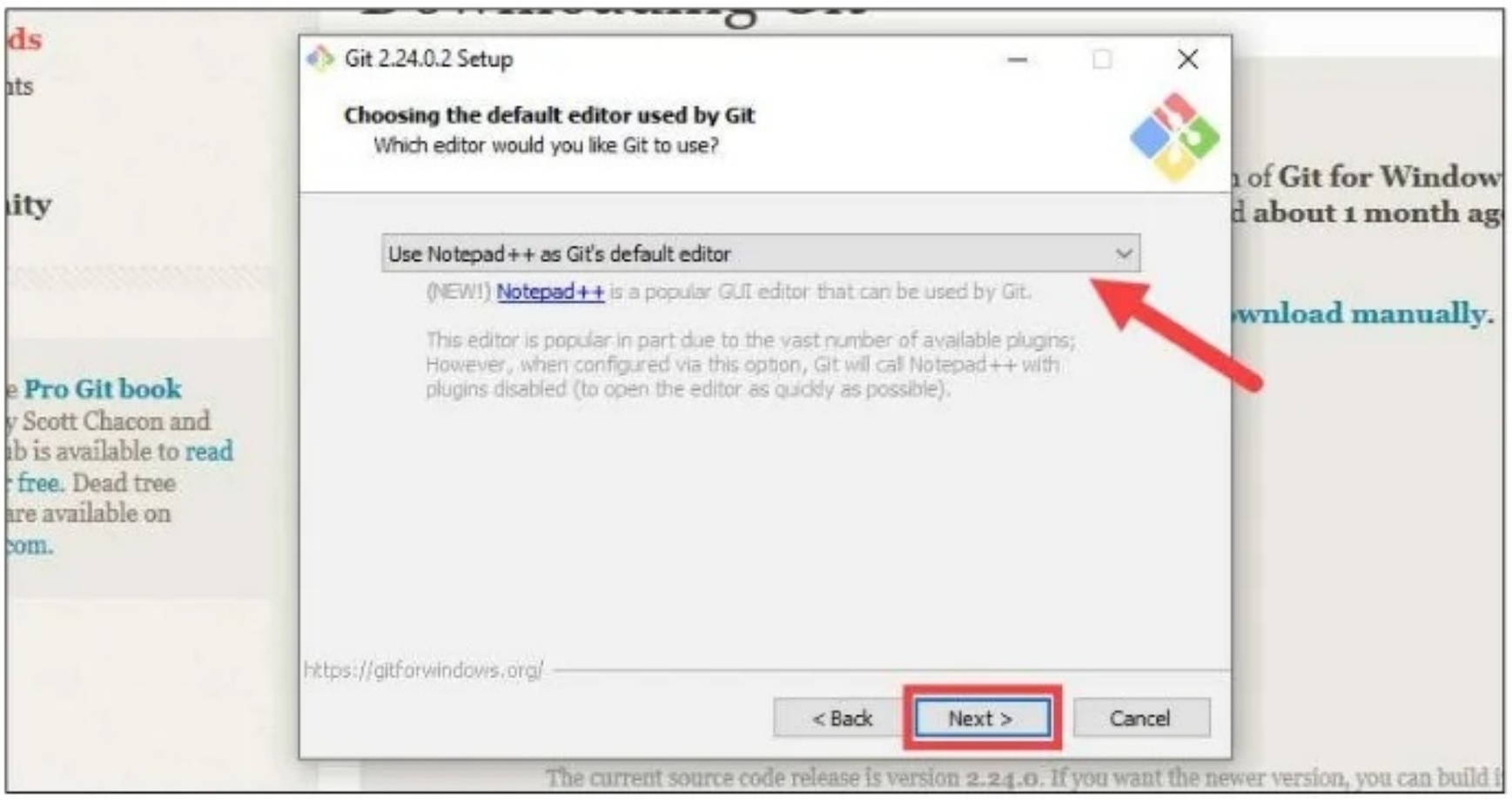



7. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.


8. The installer will offer to create a start menu folder. Simply click **Next**.

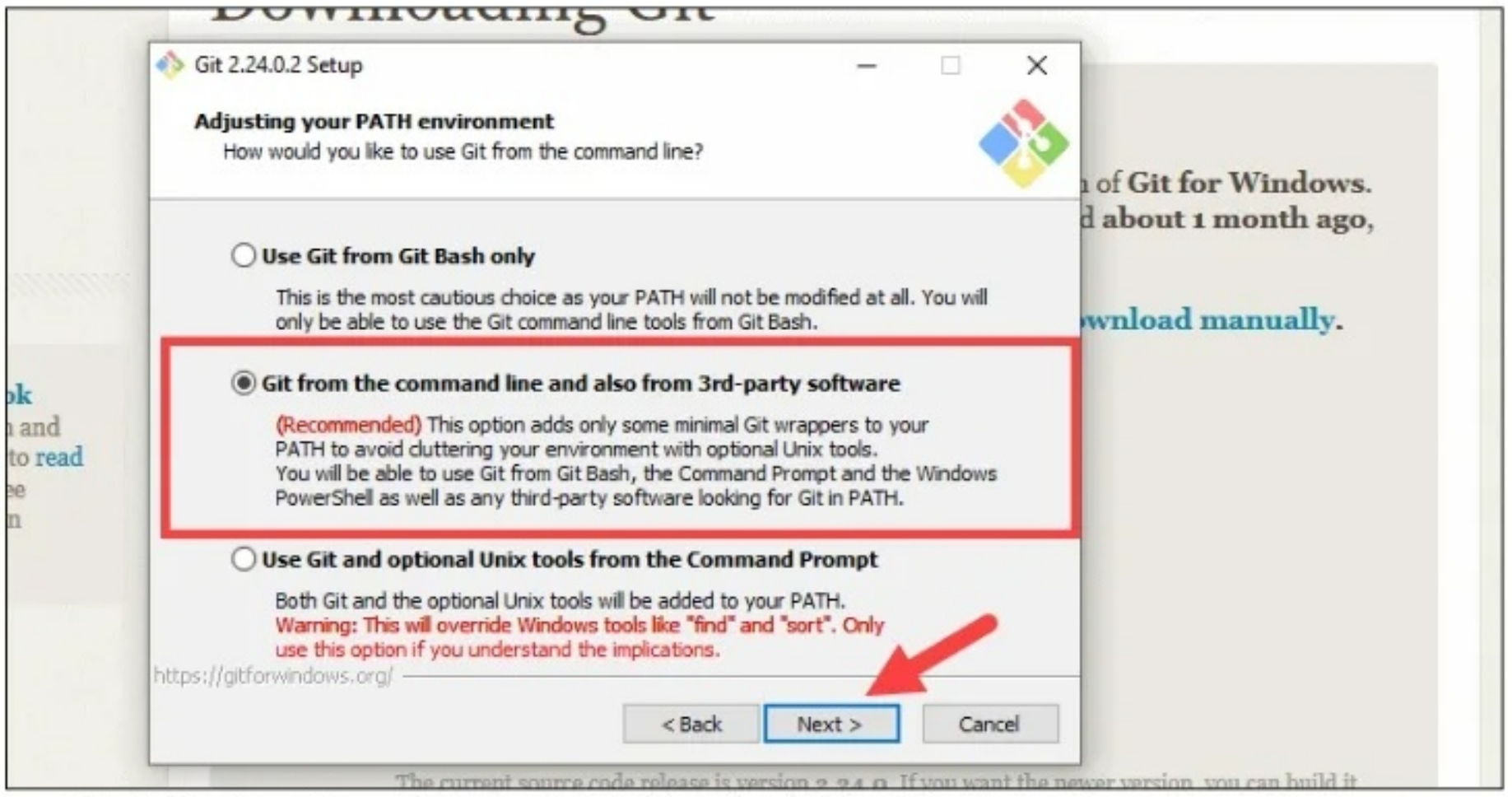



9. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



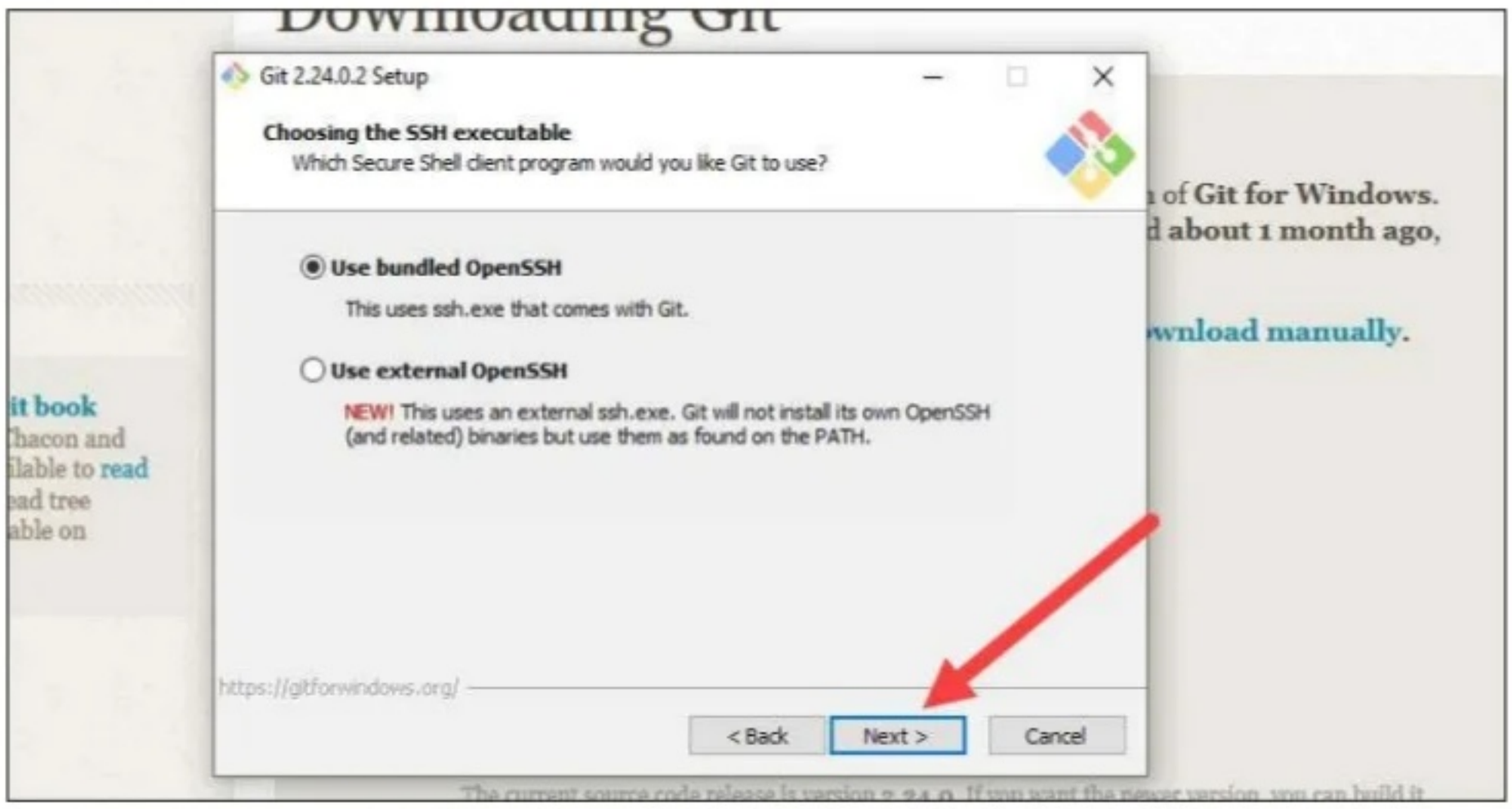

10. The next step allows you to choose a different name for your initial branch. The default is 'master.' Unless you're working in a team that requires a different name, leave the default option and click **Next.**


11. This installation step allows you to change the **PATH environment**. The **PATH** is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.




**Server Certificates, Line Endings and Terminal Emulators**

12. The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next.**



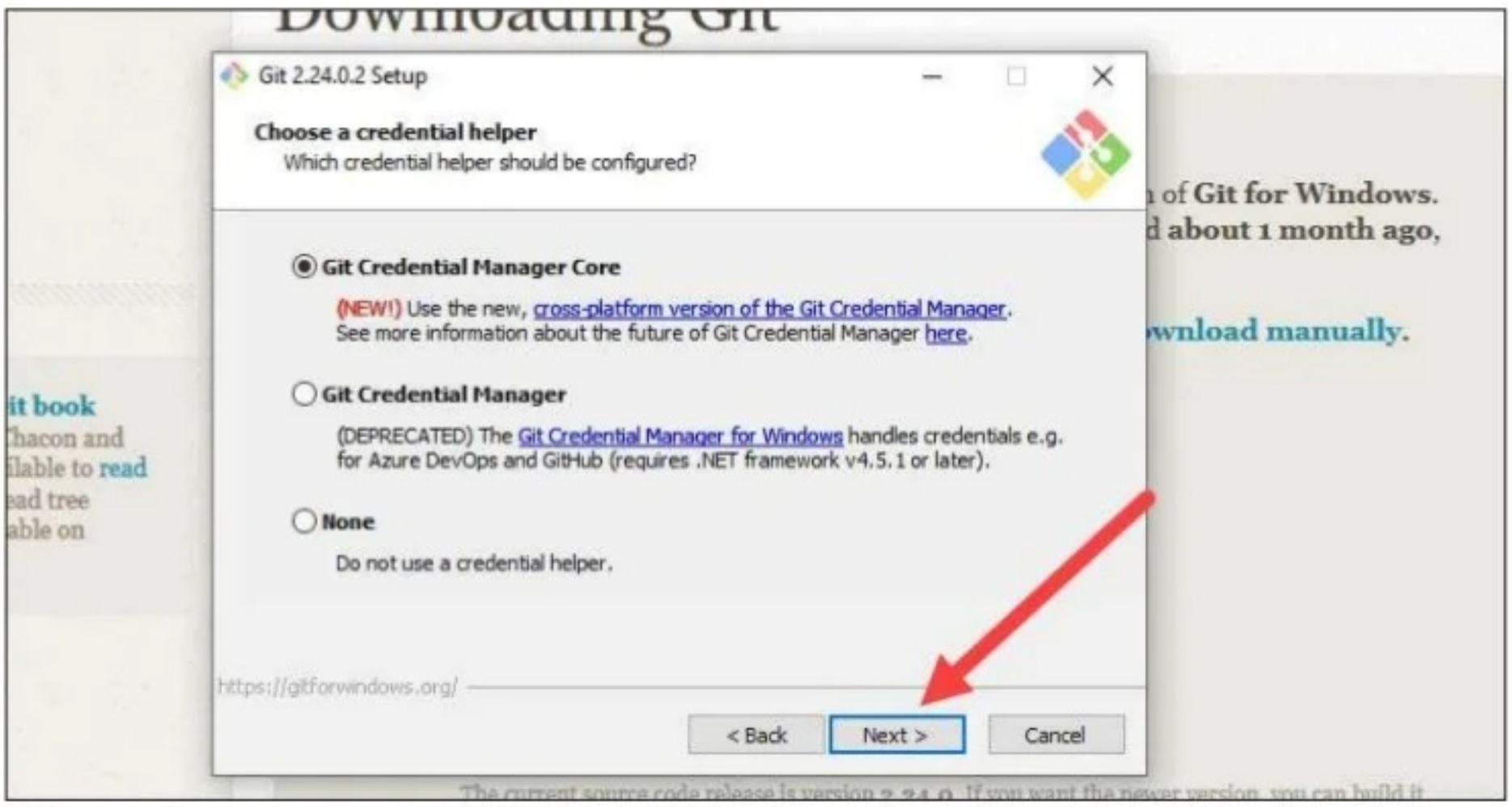

13. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.


14. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.


15. Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click **Next**.

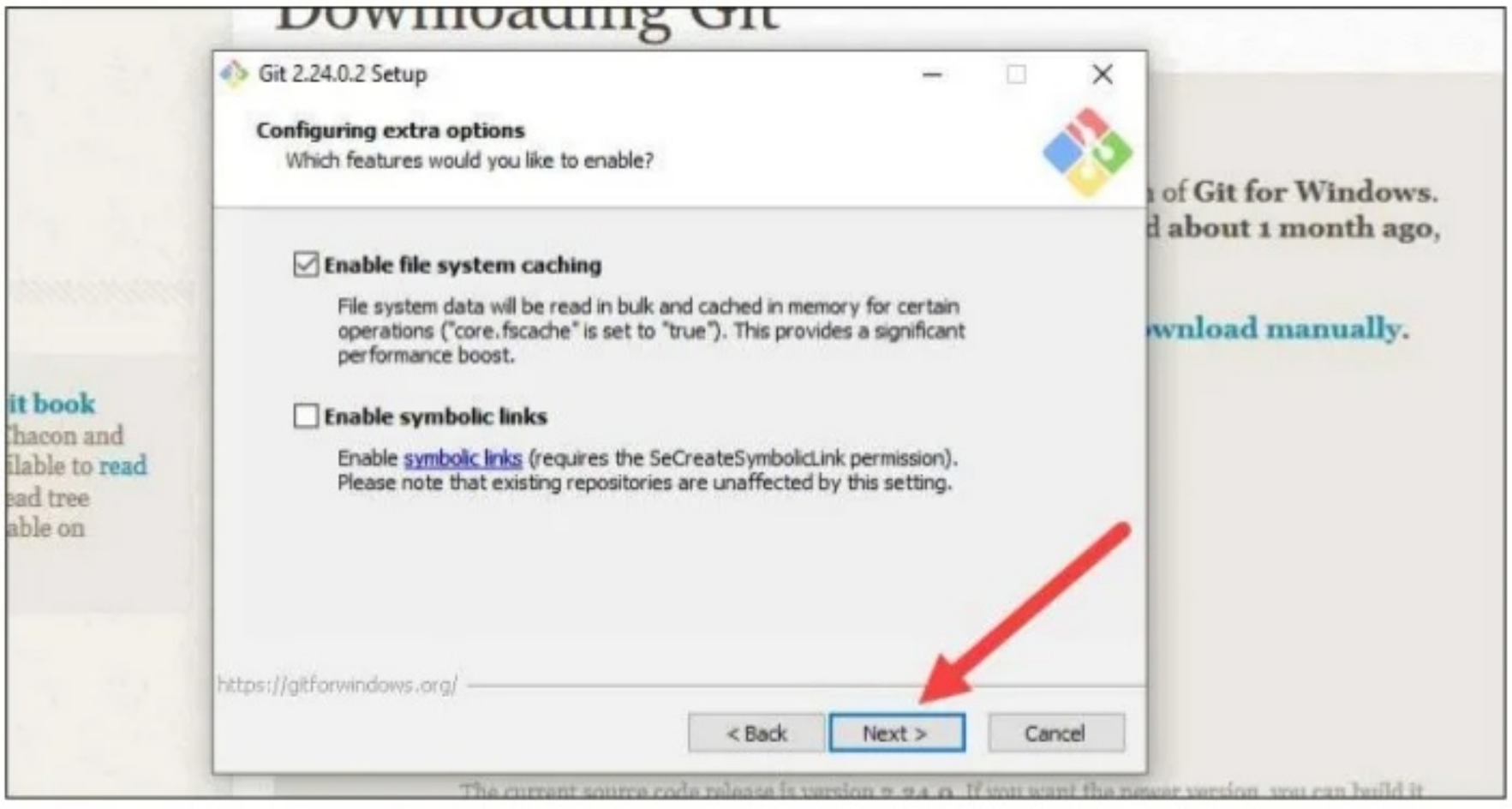

16. The installer now asks what the **`git pull`** command should do. The default option is recommended unless you specifically need to change its behavior. Click **Next** to continue with the installation.


17. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.




**Additional Customization Options**

18. The default options are recommended, however this step allows you to decide which extra option you would like to enable. If you use symbolic links, which are like shortcuts for the command line, tick the box. Click **Next**.




19. Depending on the version of Git you're installing, it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were offered. Unless you are feeling adventurous, leave them unchecked and click **Install**.


**Complete Git Installation Process**

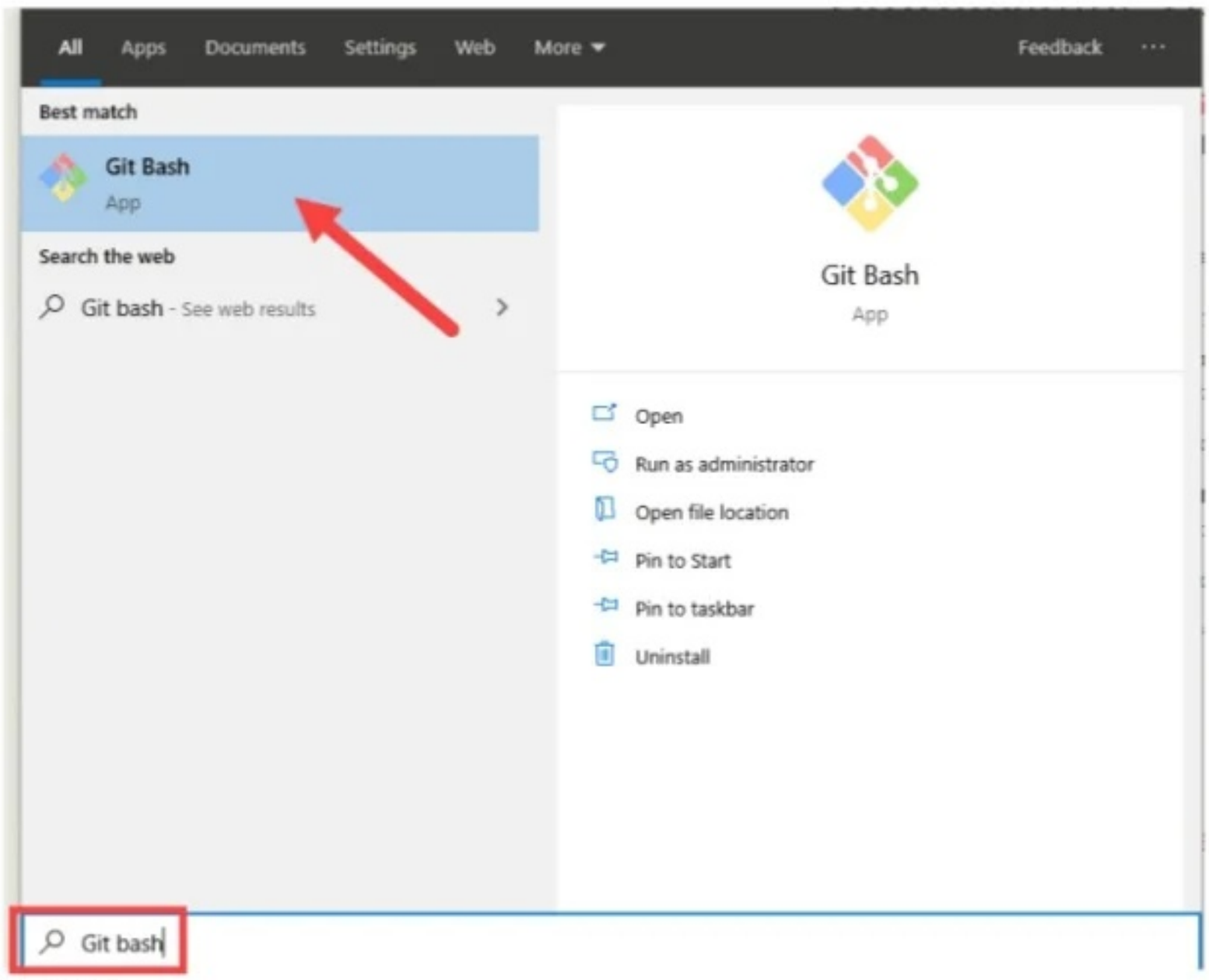
20. Once the installation is complete, tick the boxes to view the Release Notes or Launch Git Bash, then click **Finish**.


How to Launch Git in Windows

Git has two modes of use – a **bash scripting shell** (or command line) and a **graphical user interface** (GUI).

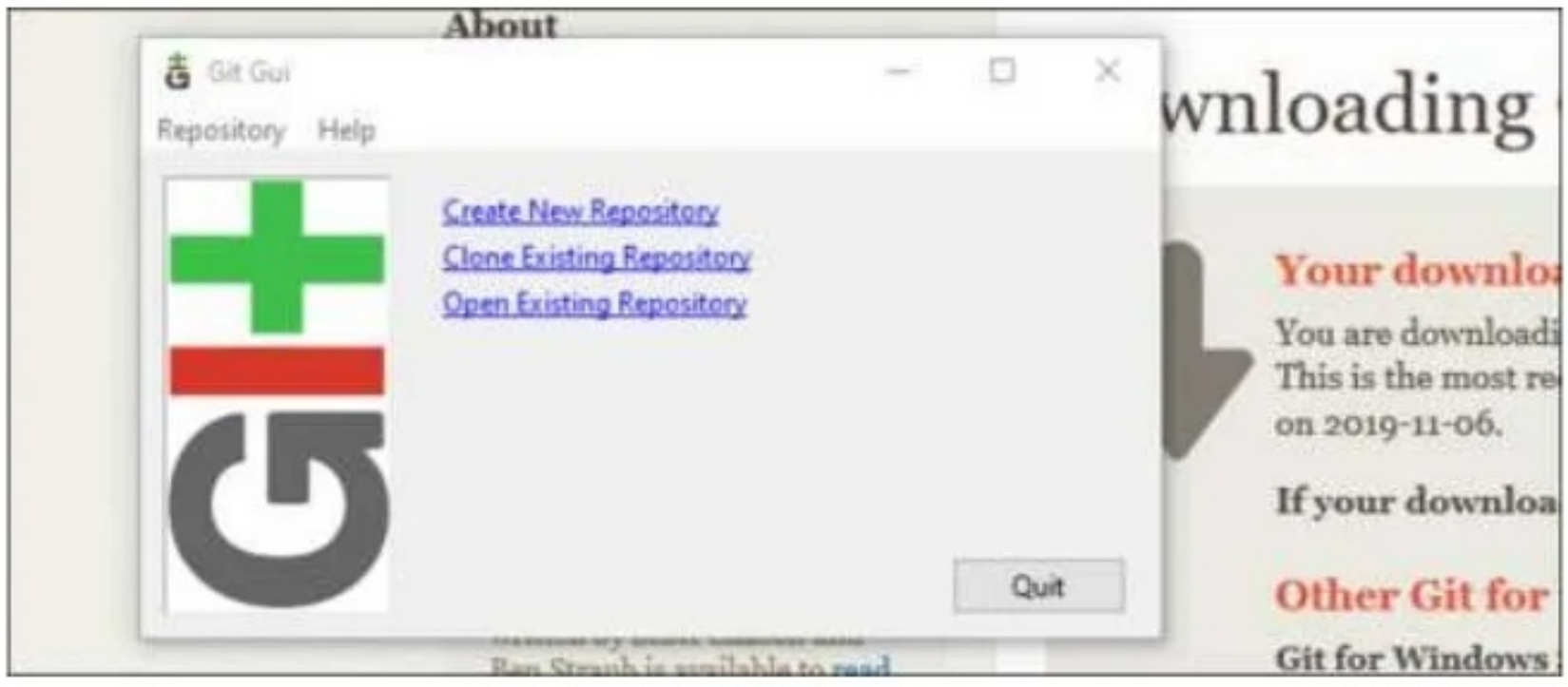
**Launch Git Bash Shell**

To launch **Git Bash** open the **Windows Start** menu, type ***git bash*** and press **Enter** (or click the application icon).




**Launch Git GUI**

To launch **Git GUI** open the **Windows Start** menu, type ***git gui*** and press **Enter** (or click the application icon).




***Commands:***

**Git Commands: Working With Local Repositories**

- **git init**
  - The command git init is used to create an empty Git repository.
  - After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

git init

- **git add**
  - Add command is used after checking the status of the files, to add those files to the staging area.
  - Before running the commit command, "git add" is used to add any new or modified files.

```
git add .
```

- **git commit**
  - The commit command makes sure that the changes are saved to the local repository.
  - The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

```
git commit -m “commit message”
```

- **git status**
  - The git status command tells the current state of the repository.
  - The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

- **git config**
  - The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
  - When git config is used with --global flag, it writes the settings to all repositories on the computer.

```
git config --global user.name “any user name”

git config --global user.email <email id>
```

- **git branch**
  - The git branch command is used to determine what branch the local repository is on.
  - The command enables adding and deleting a branch.

```
# Create a new branch
  git branch <branch_name>
# List all remote or local branches
  git branch -a
# Delete a branch
  git branch -d <branch_name>
```

- **git checkout**
  - The git checkout command is used to switch branches, whenever the work is to be started on a different branch.
  - The command works on three separate entities: files, commits, and branches.

```
# Checkout an existing branch
  git checkout <branch_name>
# Checkout and create a new branch with that name
  git checkout -b <new_branch>
```

- **git merge**

- The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch.
- It is used to merge the changes in the staging branch to the stable branch.

```
git merge <branch_name>
```

## Git Commands: Working With Remote Repositories

- **git remote**
  - The git remote command is used to create, view, and delete connections to other repositories.
  - The connections here are not like direct links into other repositories, but as bookmarks that serve as convenient names to be used as a reference.

```
git remote add origin <address>
```

- **git clone**
  - The git clone command is used to create a local working copy of an existing remote repository.
  - The command downloads the remote repository to the computer. It is equivalent to the Git init command when working with a remote repository.

```
git clone <remote_URL>
```

- **git pull**
  - The git pull command is used to fetch and merge changes from the remote repository to the local repository.
  - The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

```
git pull <branch_name> <remote URL>
```

- **git push**
  - The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository.
  - The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

```
git push -u origin master
```

- **git stash**
  - The git stash command takes your modified tracked files and saves it on a pile of incomplete changes that you can reapply at any time. To go back to work, you can use the stash pop.
  - The git stash command will help a developer switch branches to work on something else without committing to incomplete work.

```
# Store current work with untracked files
 git stash -u
# Bring stashed work back to the working directory
 git stash pop
```

- **git log**
  - The git log command shows the order of the commit history for a repository.
  - The command helps in understanding the state of the current branch by showing the commits that lead to this state.

git log

***Output Screenshots:***

***Conclusion:***

You have learned the basics of Git and the different commands that are used, and also saw different basic Git commands.

***References:(Put your own references)***

## EXPERIMENT NO. - 02

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 2

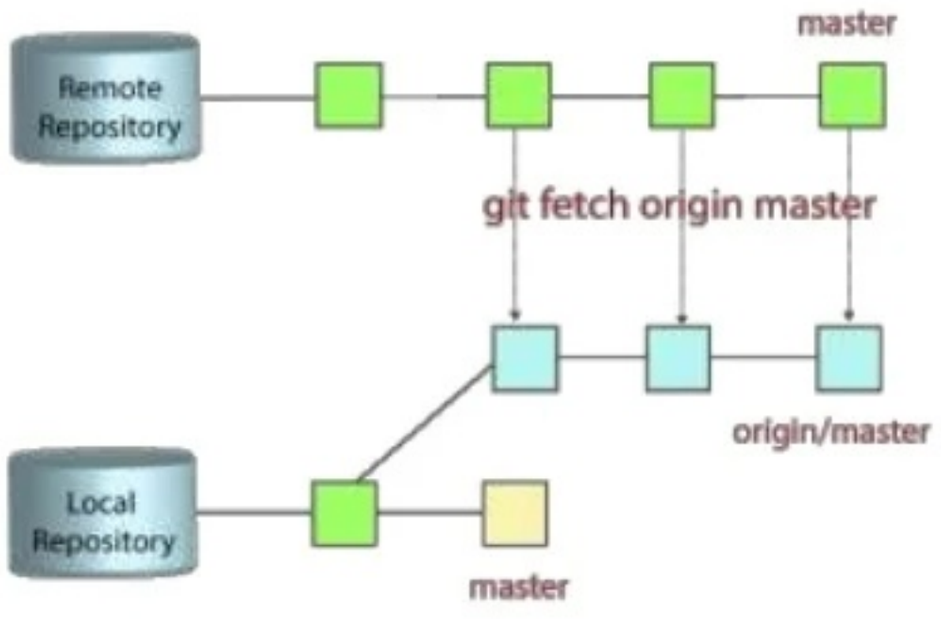
***Problem Definition:*** To fetch and synchronize Git repository.

***Compiler / Tool :*** Git-2.35.1.2-64-bit

***Implementation:***

**Git Fetch**

Git "fetch" Downloads commits, objects and refs from another repository. It fetches branches and tags from one or more repositories. It holds repositories along with the objects that are necessary to complete their histories to keep updated remote-tracking branches.




**The "git fetch"command**

The "**git fetch**" **command** is used to pull the updates from remote-tracking branches. Additionally, we can get the updates that have been pushed to our remote branches to our local machines. As we know, a branch is a variation of our repositories main code, so the remote-tracking branches are branches that have been set up to pull and push from remote repository.

**How to fetch Git Repository**

We can use fetch command with many arguments for a particular data fetch. See the below scenarios to understand the uses of fetch command.

**Scenario 1: To fetch the remote repository:**

We can fetch the complete repository with the help of fetch command from a repository URL like a pull command does.

**Syntax:**

```
$ git fetch< repository Url>
```

**Scenario 2: To fetch a specific branch:**

We can fetch a specific branch from a repository. It will only access the element from a specific branch.

**Syntax:**

$ git fetch <branch URL><branch name>

**Scenario 3: To fetch all the branches simultaneously:**

The git fetch command allows to fetch all branches simultaneously from a remote repository

**Syntax:**

```
$ git fetch -all
```

**Scenario 4: To synchronize the local repository:**

Suppose, your team member has added some new features to your remote repository. So, to add these updates to your local repository, use the git fetch command. It is used as follows.

**Syntax:**

```
$ git fetch origin
```

The git fetch can fetch from either a single named repository or URL or from several repositories at once. It can be considered as the safe version of the git pull commands.

The git fetch downloads the remote content but not update your local repo's working state. When no remote server is specified, by default, it will fetch the origin remote.

**Differences between git fetch and git pull**

To understand the differences between fetch and pull, let's know the similarities between both of these commands. Both commands are used to download the data from a remote repository. But both of these commands work differently. Like when you do a git pull, it gets all the changes from the remote or central repository and makes it available to your corresponding branch in your local repository. When you do a git fetch, it fetches all the changes from the remote repository and stores it in a separate branch in your local repository. You can reflect those changes in your corresponding branches by merging.

So basically,

git pull = git fetch + git merge

**Git Fetch vs. Pull**

| git fetch | git pull |
|---|---|
| Fetch downloads only new data from a remote repository. | Pull is used to update your current HEAD branch with the latest changes from the remote server. |
| Fetch is used to get a new view of all the things that happened in a remote repository. | Pull downloads new data and directly integrates it into your current working copy files. |
| Fetch never manipulates or spoils data. | Pull downloads the data and integrates it with the current working file. |
| It protects your code from merge conflict. | In git pull, there are more chances to create the **merge conflict**. |
| It is better to use git fetch command with git merge command on a pulled repository. | It is not an excellent choice to use git pull if you already pulled any repository. |

***Output Screenshot:***

***Conclusion:***

The git fetch can fetch from either a single named repository or URL or from several repositories at once. It can be considered as the safe version of the git pull commands.

The git fetch downloads the remote content but not update your local repo's working state. When no remote server is specified, by default, it will fetch the origin remote.

***References:***

## EXPERIMENT NO. - 03

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 3

***Problem Definition:*** To perform basic branching and merging in Git.

***Compiler / Tool:*** Git-2.35.1.2-64-bit

***Implementation:***

**Basic Branching and Merging**

Let's go through a simple example of branching and merging with a workflow that you might use in the real world. You'll follow these steps:

1. Do some work on a website.
2. Create a branch for a new user story you're working on.
3. Do some work in that branch.

At this stage, you'll receive a call that another issue is critical and you need a hotfix. You'll do the following:

1. Switch to your production branch.
2. Create a branch to add the hotfix.
3. After it's tested, merge the hotfix branch, and push to production.
4. Switch back to your original user story and continue working.

**Basic Branching**

First, let's say you're working on your project and have a couple of commits already on the `master` branch.

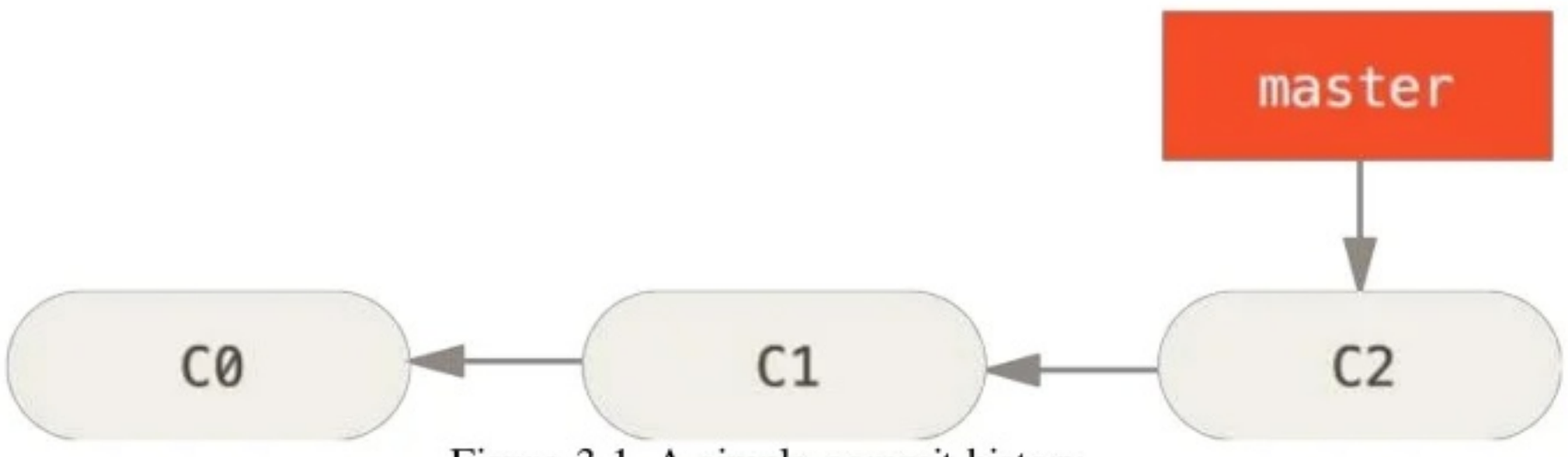



Figure 3.1. A simple commit history

You've decided that you're going to work on issue #53 in whatever issue-tracking system your company uses. To create a new branch and switch to it at the same time, you can run the `git checkout` command with the `-b` switch:

```
$ git checkout -b iss53
Switched to a new branch "iss53"
```

This is shorthand for:

```
$ git branch iss53
$ git checkout iss53
```

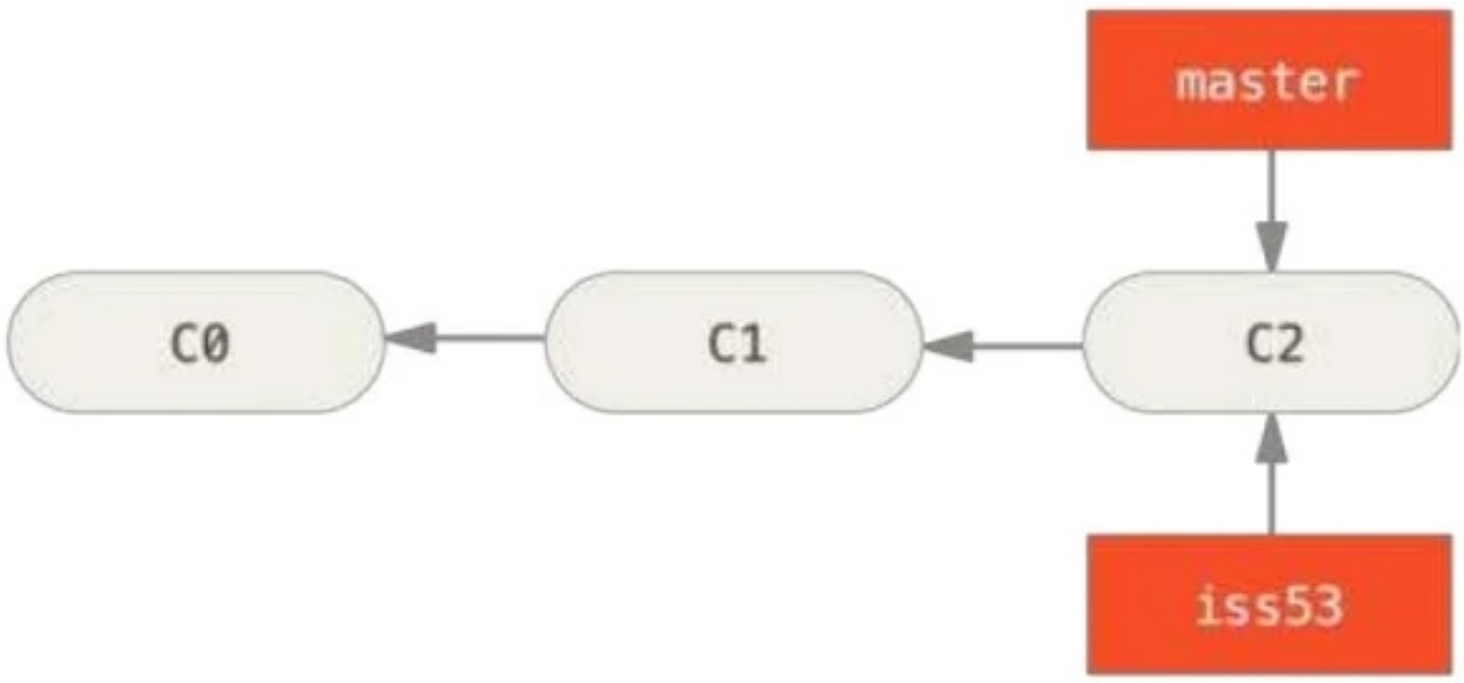



Figure 3.2. Creating a new branch pointer

You work on your website and do some commits. Doing so moves the `iss53` branch forward, because you have it checked out (that is, your `HEAD` is pointing to it):

```
$ vim index.html
$ git commit -a -m 'Create new footer [issue 53]'
```

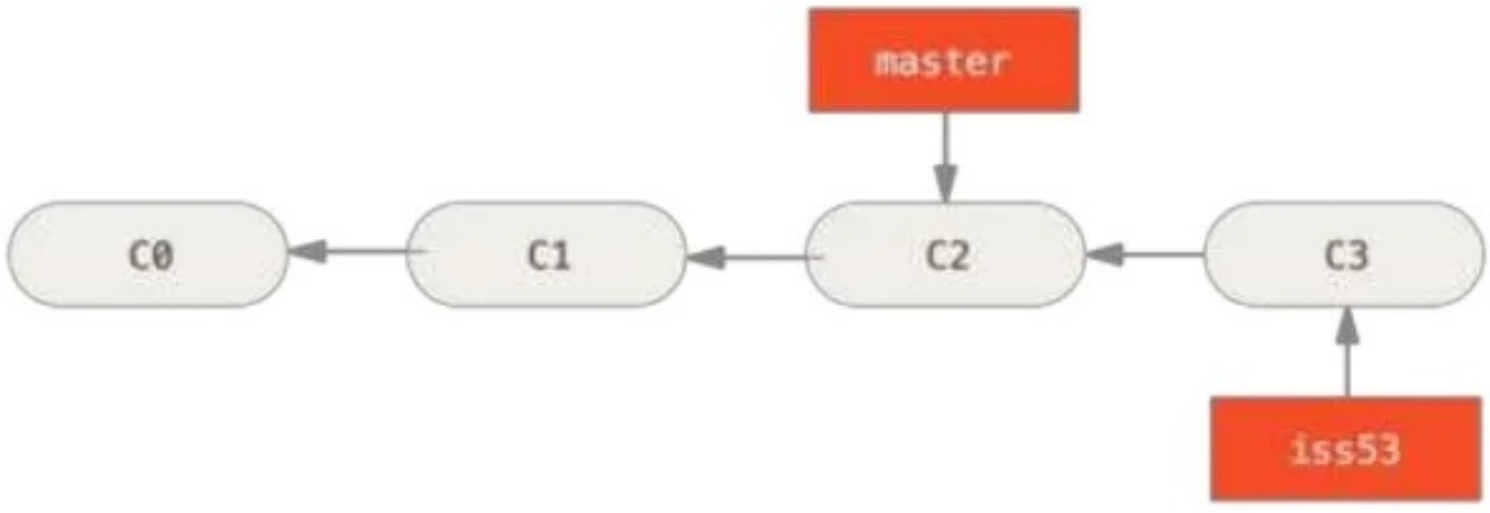



Figure 3.3. The `iss53` branch has moved forward with your work

Now you get the call that there is an issue with the website, and you need to fix it immediately. With Git, you don't have to deploy your fix along with the `iss53` changes you've made, and you don't have to put a lot of effort into reverting those changes before you can work on applying your fix to what is in production. All you have to do is switch back to your `master` branch.

However, before you do that, note that if your working directory or staging area has uncommitted changes that conflict with the branch you're checking out, Git won't let you switch branches. It's best to have a clean working state when you switch branches. There are ways to get around this (namely, stashing and commit amending) that we'll cover later on, in Stashing and Cleaning. For now, let's assume you've committed all your changes, so you can switch back to your `master` branch:

```
$ git checkout master
Switched to branch 'master'
```

At this point, your project working directory is exactly the way it was before you started working on issue #53, and you can concentrate on your hotfix. This is an important point to remember:

when you switch branches, Git resets your working directory to look like it did the last time you committed on that branch. It adds, removes, and modifies files automatically to make sure your working copy is what the branch looked like on your last commit to it.

Next, you have a hotfix to make. Let's create a `hotfix` branch on which to work until it's completed:

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'Fix broken email address'
[hotfix 1fb7853] Fix broken email address
 1 file changed, 2 insertions(+)
```

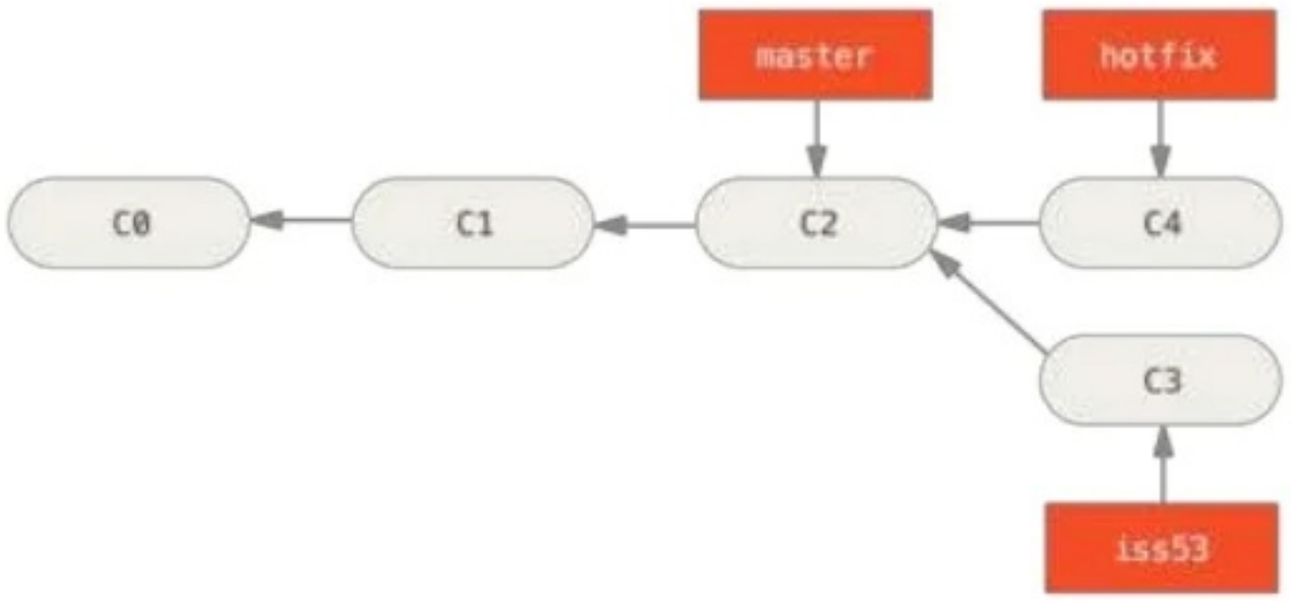



Figure 3.4. Hotfix branch based on `master`

You can run your tests, make sure the hotfix is what you want, and finally merge the `hotfix` branch back into your `master` branch to deploy to production. You do this with the `git merge` command:

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

You'll notice the phrase "fast-forward" in that merge. Because the commit `C4` pointed to by the branch `hotfix` you merged in was directly ahead of the commit `C2` you're on, Git simply moves the pointer forward. To phrase that another way, when you try to merge one commit with a commit that can be reached by following the first commit's history, Git simplifies things by moving the pointer forward because there is no divergent work to merge together — this is called a "fast-forward."

Your change is now in the snapshot of the commit pointed to by the `master` branch, and you can deploy the fix.

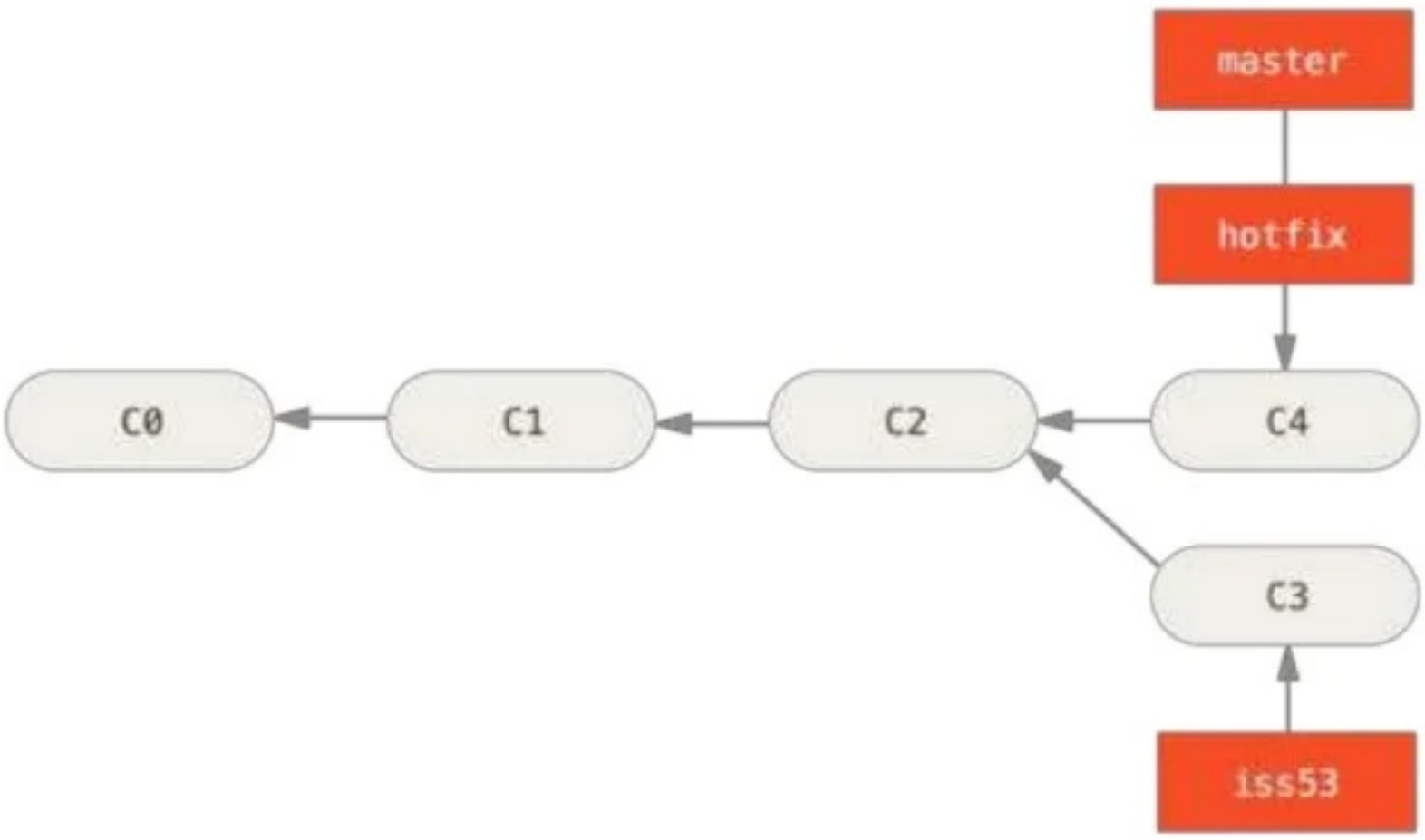



Figure 3.5. `master` is fast-forwarded to `hotfix`

After your super-important fix is deployed, you're ready to switch back to the work you were doing before you were interrupted. However, first you'll delete the `hotfix` branch, because you no longer need it — the `master` branch points at the same place. You can delete it with the `-d` option to `git branch`:

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

Now you can switch back to your work-in-progress branch on issue #53 and continue working on it.

```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'Finish the new footer [issue 53]'
[iss53 ad82d7a] Finish the new footer [issue 53]
1 file changed, 1 insertion(+)
```

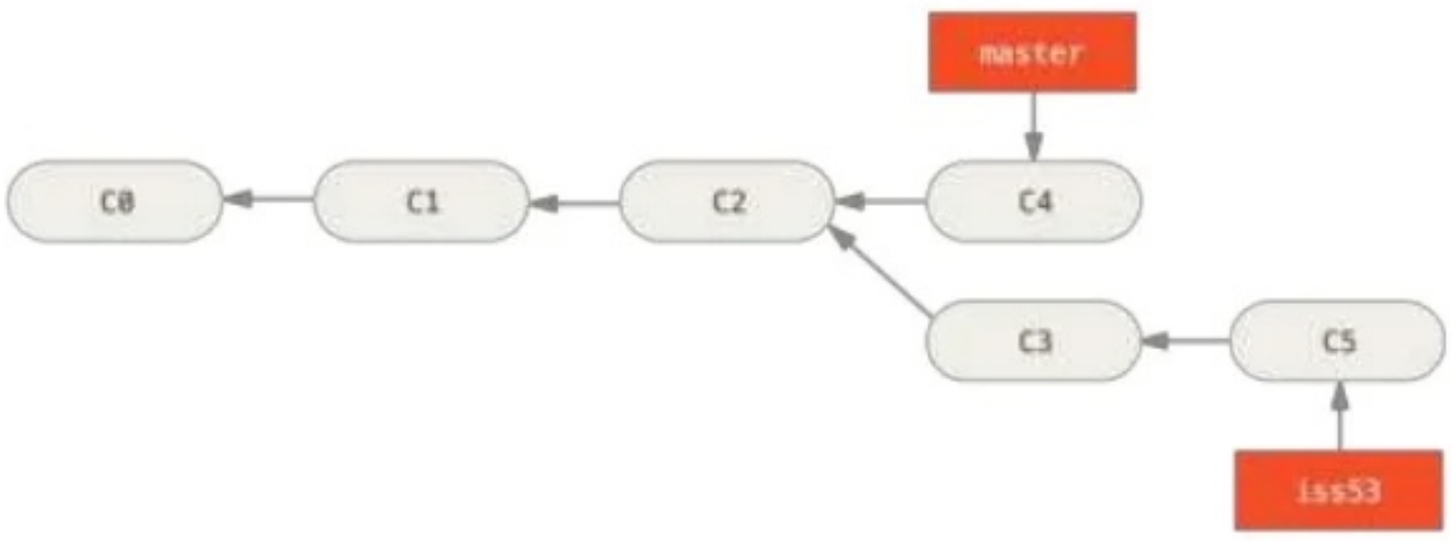



Figure 3.6. Work continues on `iss53`

It's worth noting here that the work you did in your `hotfix` branch is not contained in the files in your `iss53` branch. If you need to pull it in, you can merge your `master` branch into your `iss53` branch by running `git merge master`, or you can wait to integrate those changes until you decide to pull the `iss53` branch back into `master` later.

### Basic Merging

Suppose you've decided that your issue #53 work is complete and ready to be merged into your `master` branch. In order to do that, you'll merge your `iss53` branch into `master`, much like you merged your `hotfix` branch earlier. All you have to do is check out the branch you wish to merge into and then run the `git merge` command:

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

This looks a bit different than the `hotfix` merge you did earlier. In this case, your development history has diverged from some older point. Because the commit on the branch you're on isn't a direct ancestor of the branch you're merging in, Git has to do some work. In this case, Git does a simple three-way merge, using the two snapshots pointed to by the branch tips and the common ancestor of the two.

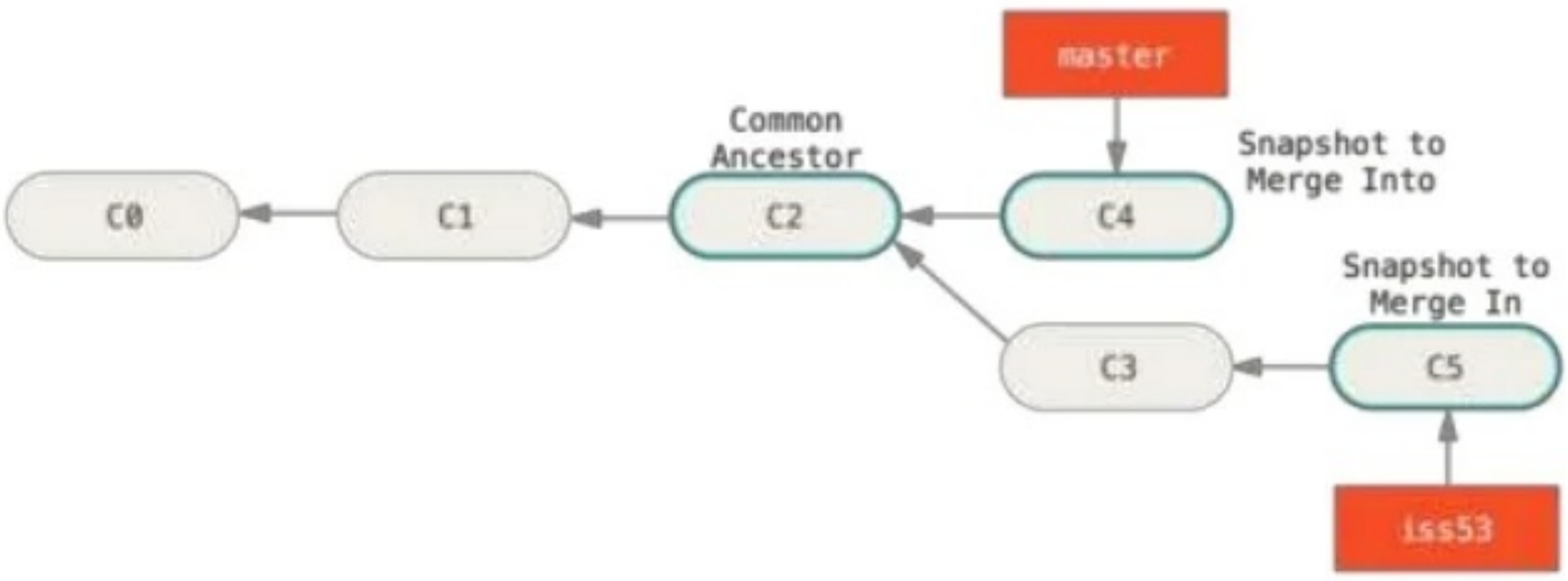



Figure 3.7. Three snapshots used in a typical merge

Instead of just moving the branch pointer forward, Git creates a new snapshot that results from this three-way merge and automatically creates a new commit that points to it. This is referred to as a merge commit, and is special in that it has more than one parent.

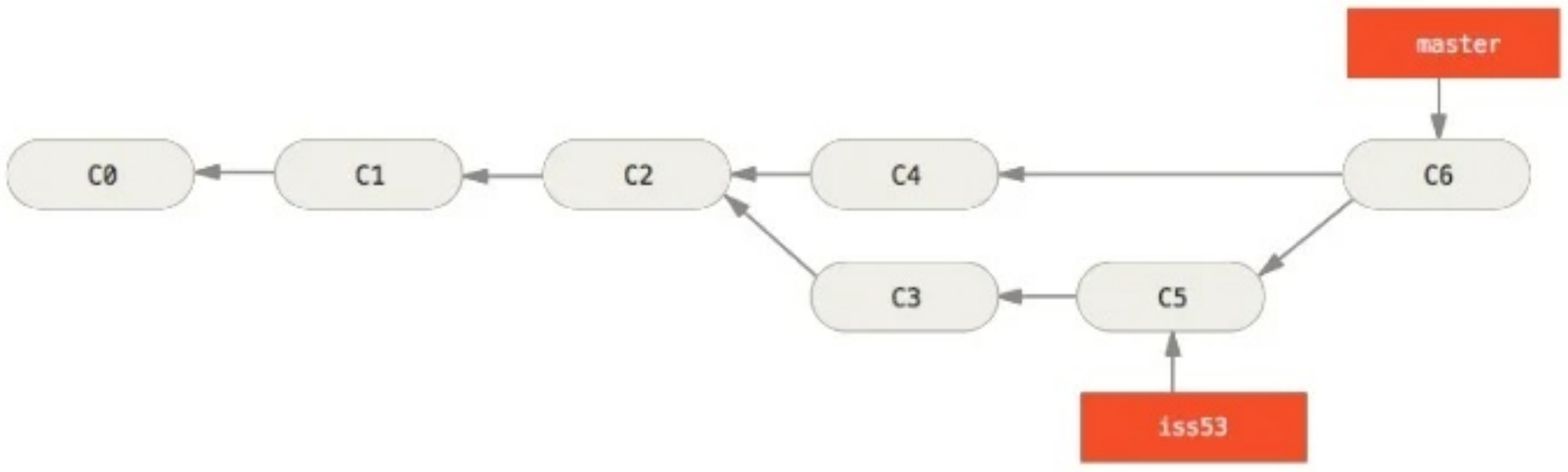



Figure 3.8. A merge commit

Now that your work is merged in, you have no further need for the `iss53` branch. You can close the issue in your issue-tracking system, and delete the branch:

```
$ git branch -d iss53
```

### Basic Merge Conflicts

Occasionally, this process doesn't go smoothly. If you changed the same part of the same file differently in the two branches you're merging, Git won't be able to merge them cleanly. If your fix for issue #53 modified the same part of a file as the `hotfix` branch, you'll get a merge conflict that looks something like this:

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Git hasn't automatically created a new merge commit. It has paused the process while you resolve the conflict. If you want to see which files are unmerged at any point after a merge conflict, you can run `git status`:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Anything that has merge conflicts and hasn't been resolved is listed as unmerged. Git adds standard conflict-resolution markers to the files that have conflicts, so you can open them manually and resolve those conflicts. Your file contains a section that looks something like this:

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
```

```
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

This means the version in `HEAD` (your `master` branch, because that was what you had checked out when you ran your merge command) is the top part of that block (everything above the `=======`), while the version in your `iss53` branch looks like everything in the bottom part. In order to resolve the conflict, you have to either choose one side or the other or merge the contents yourself. For instance, you might resolve this conflict by replacing the entire block with this:

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

This resolution has a little of each section, and the `<<<<<<<`, `=======`, and `>>>>>>>` lines have been completely removed. After you've resolved each of these sections in each conflicted file, run `git add` on each file to mark it as resolved. Staging the file marks it as resolved in Git.

If you want to use a graphical tool to resolve these issues, you can run `git mergetool`, which fires up an appropriate visual merge tool and walks you through the conflicts:

```
$ git mergetool
This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge
ecmerge p4merge araxis bc3 codecompare vimdiff emerge
Merging:
index.html

Normal merge conflict for 'index.html':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (opendiff):
```

If you want to use a merge tool other than the default (Git chose `opendiff` in this case because the command was run on a Mac), you can see all the supported tools listed at the top after "one of the following tools." Just type the name of the tool you'd rather use.

***Output Screenshots:***

***Conclusion:***

Git branching allows developers to diverge from the production version of code to fix a bug or add a feature. Developers create branches to work with a copy of the code without modifying the existing version. You create branches to isolate your code changes, which you test before merging to the main branch.

***References:***

## EXPERIMENT NO. - 04

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 4

***Problem Definition:*** To install and Jenkins and build a job in jenkins.

***Compiler / Tool :*** Jenkins ,Java 8 or Java 11, Modern web browsers - Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Apple Safari.

***Installation:***

**Steps to Install Jenkins on Windows**

**1. Install Java Development Kit (JDK)**

- Download JDK 8 and choose windows 32-bit or 64-bit according to your system configuration. Click on "accept the license agreement."

**2. 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\lab3>java -version
```

**3. Download and Install Jenkins**

- Download Jenkins. Under LTS, click on windows.
- After the file is downloaded, unzip it. Click on the folder and install it. Select "finish" once done.

**4. Run Jenkins on Localhost 8080**

- Once Jenkins is installed, explore it. Open the web browser and type "localhost:8080".
- Enter the credentials and log in. If you install Jenkins for the first time, the dashboard will ask you to install the recommended plugins. Install all the recommended plugins.

**5. Jenkins Server Interface**

- New Item allows you to create a new project.
- Build History shows the status of your builds.
- Manage System deals with the various configurations of the system.

**6. Build and Run a Job on Jenkins**

- Select a new item (Name - Jenkins_demo). Choose a freestyle project and click Ok.
- Under the General tab, give a description like "This is my first Jenkins job." Under the "Build Triggers" tab, select add built step and then click on the "Execute Windows" batch command.
- In the command box, type the following: echo "Hello... This is my first Jenkins Demo: %date%: %time% ". Click on apply and then save.
- Select build now. You can see a building history has been created. Click on that. In the console output, you can see the output of the first Jenkins job with time and date.

### Post Installation Setup Wizard: Accessing the Jenkins Home Page

You are now ready to access Jenkins home page at http:<YOUR IP ADDRESS>:<PORT>.

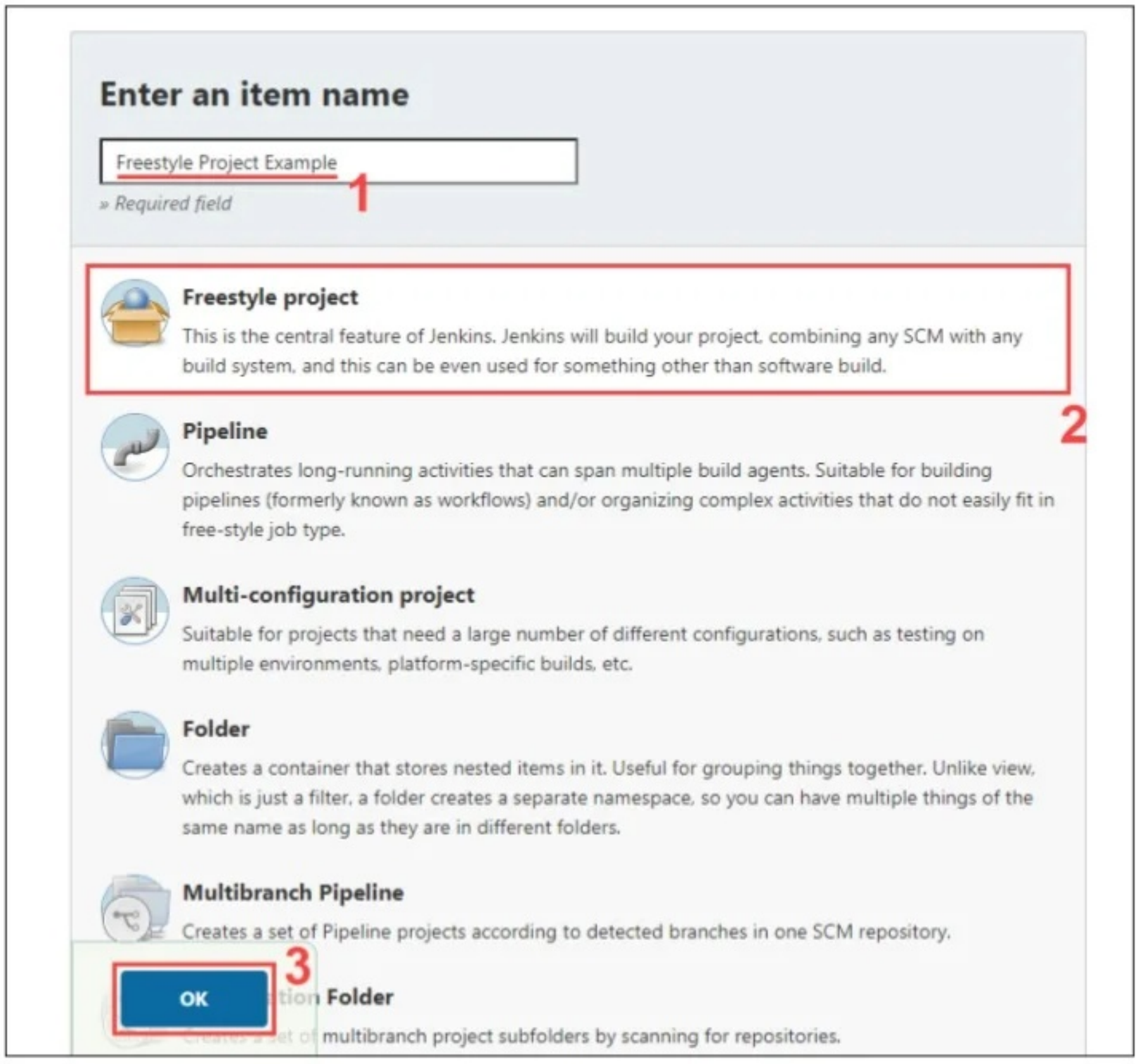

## How to Set up a Build Job in Jenkins

Follow the steps outlined below to set up and run a new Jenkins freestyle project.

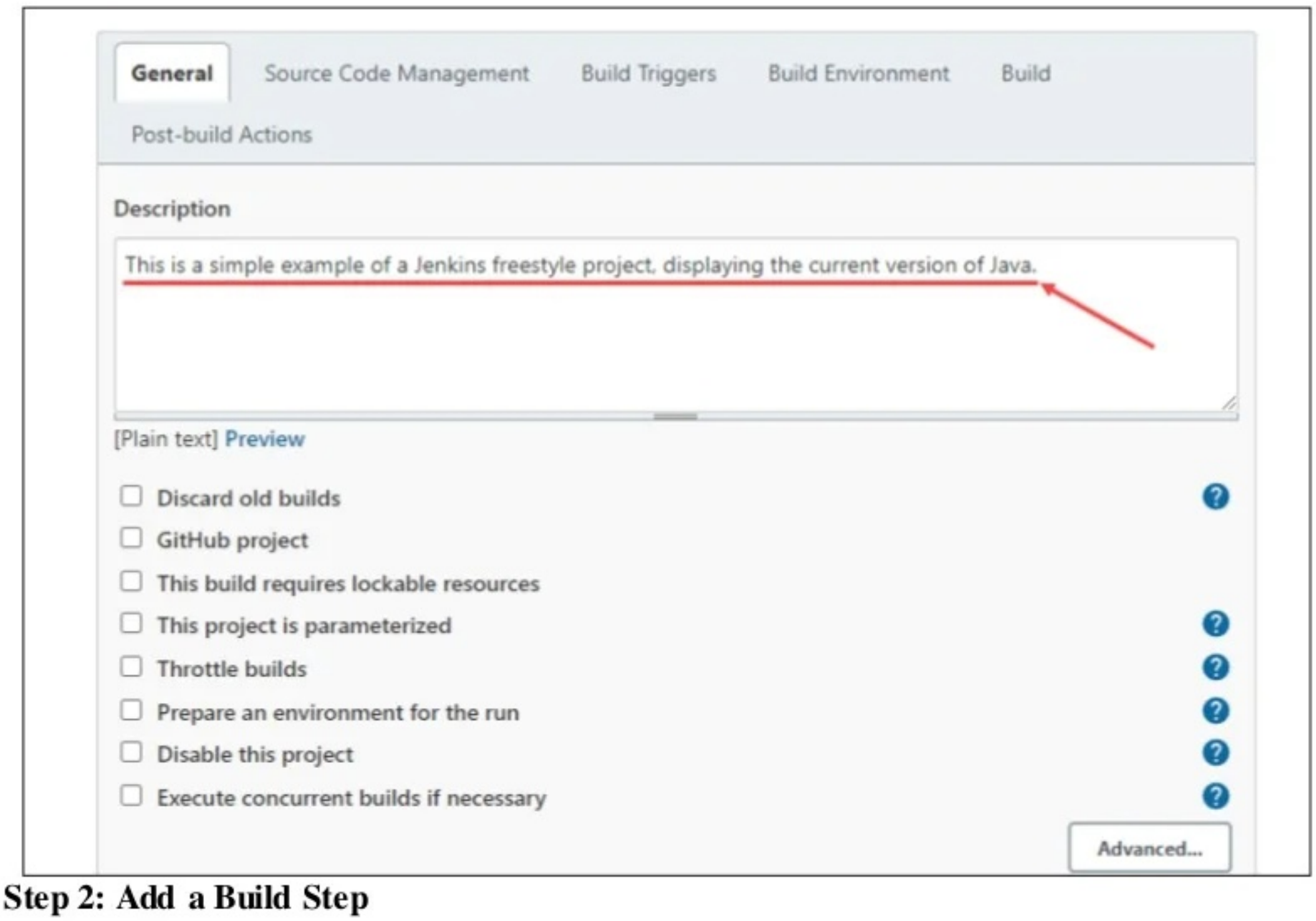
### Step 1: Create a New Freestyle Project

1. Click the **New Item** link on the left-hand side of the Jenkins dashboard.


2. Enter the new project's name in the **Enter an item name** field and select the **Freestyle project** type. Click **OK** to continue.



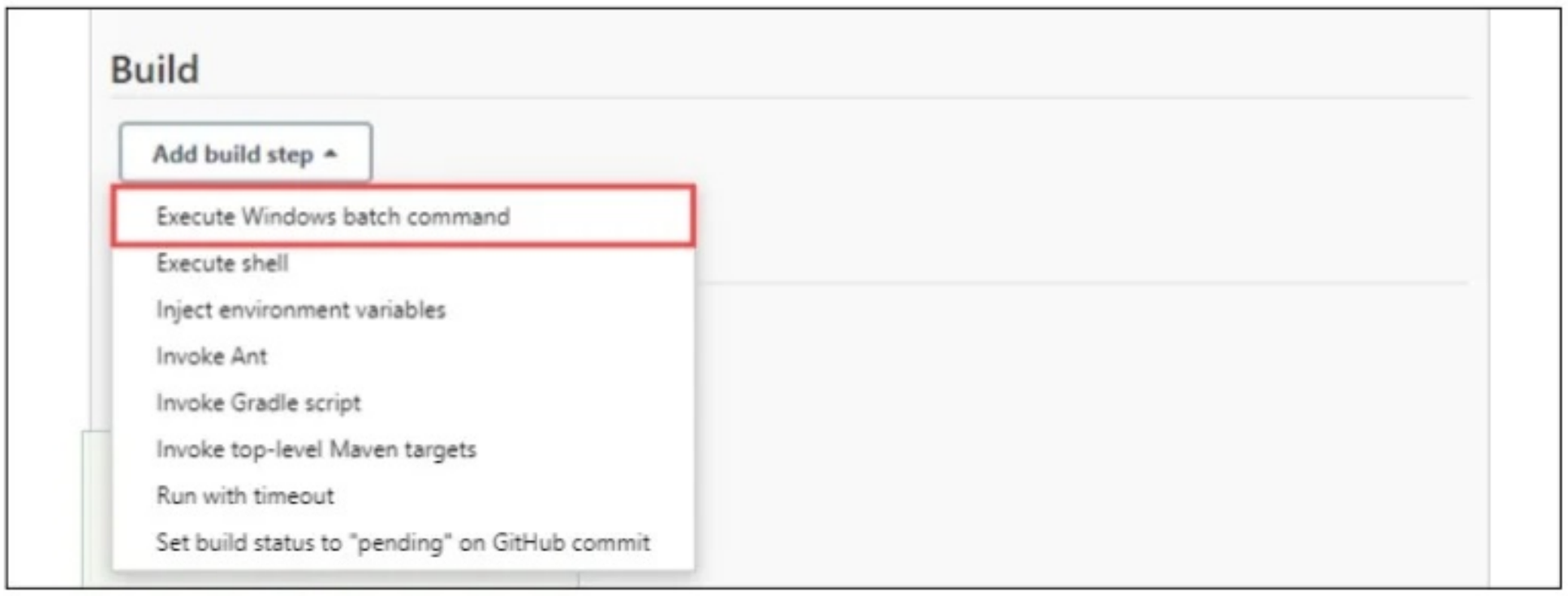

3. Under the *General* tab, add a project description in the **Description** field.




## Step 2: Add a Build Step

1. Scroll down to the *Build* section.

2. Open the **Add build step** drop-down menu and select **Execute Windows batch command**.



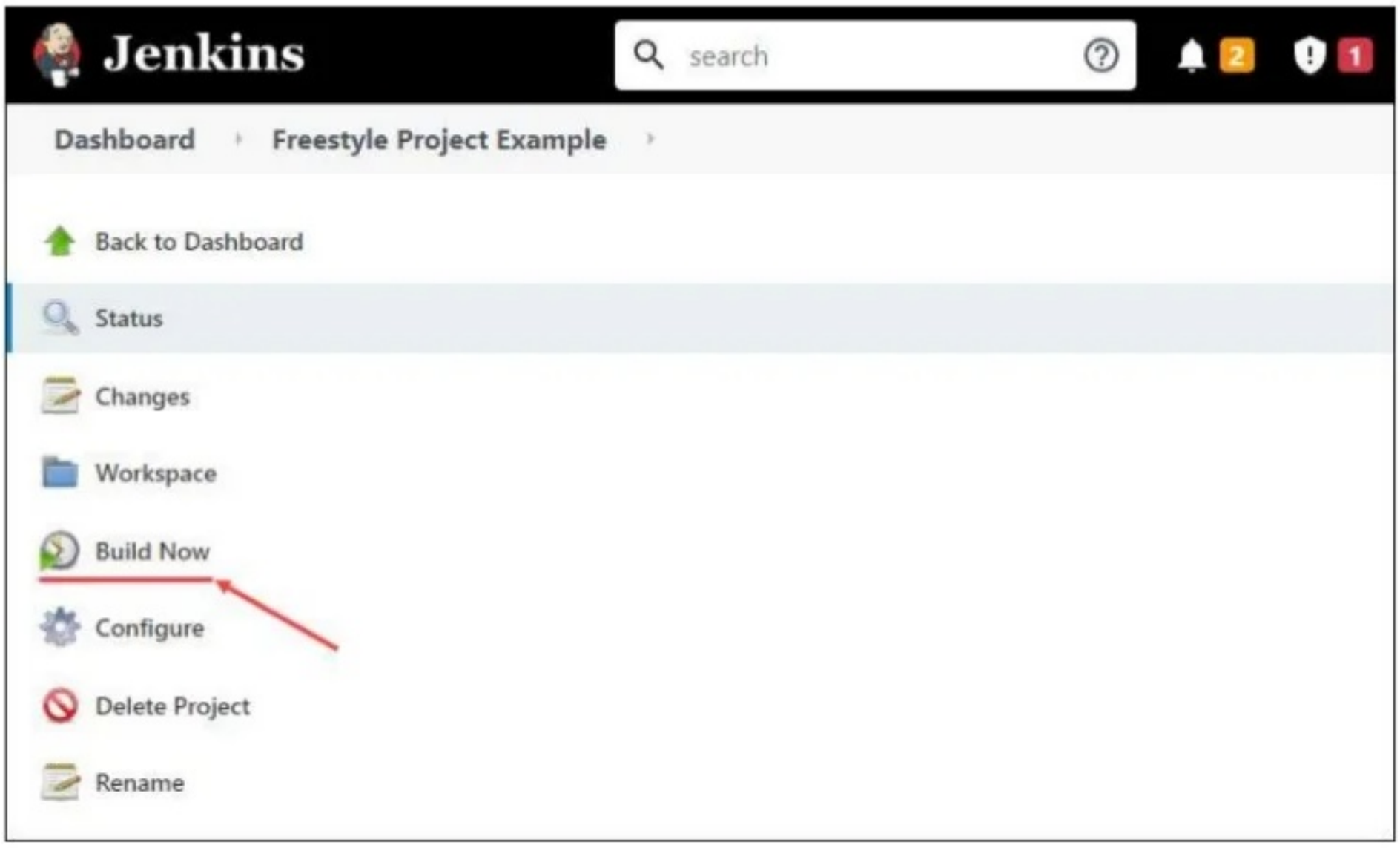

3. Enter the commands you want to execute in the **Command** field. For this tutorial, we are using a simple set of commands that display the current version of Java and Jenkins working directory:

```
java -version
dir
```

4. Click the **Save** button to save changes to the project.

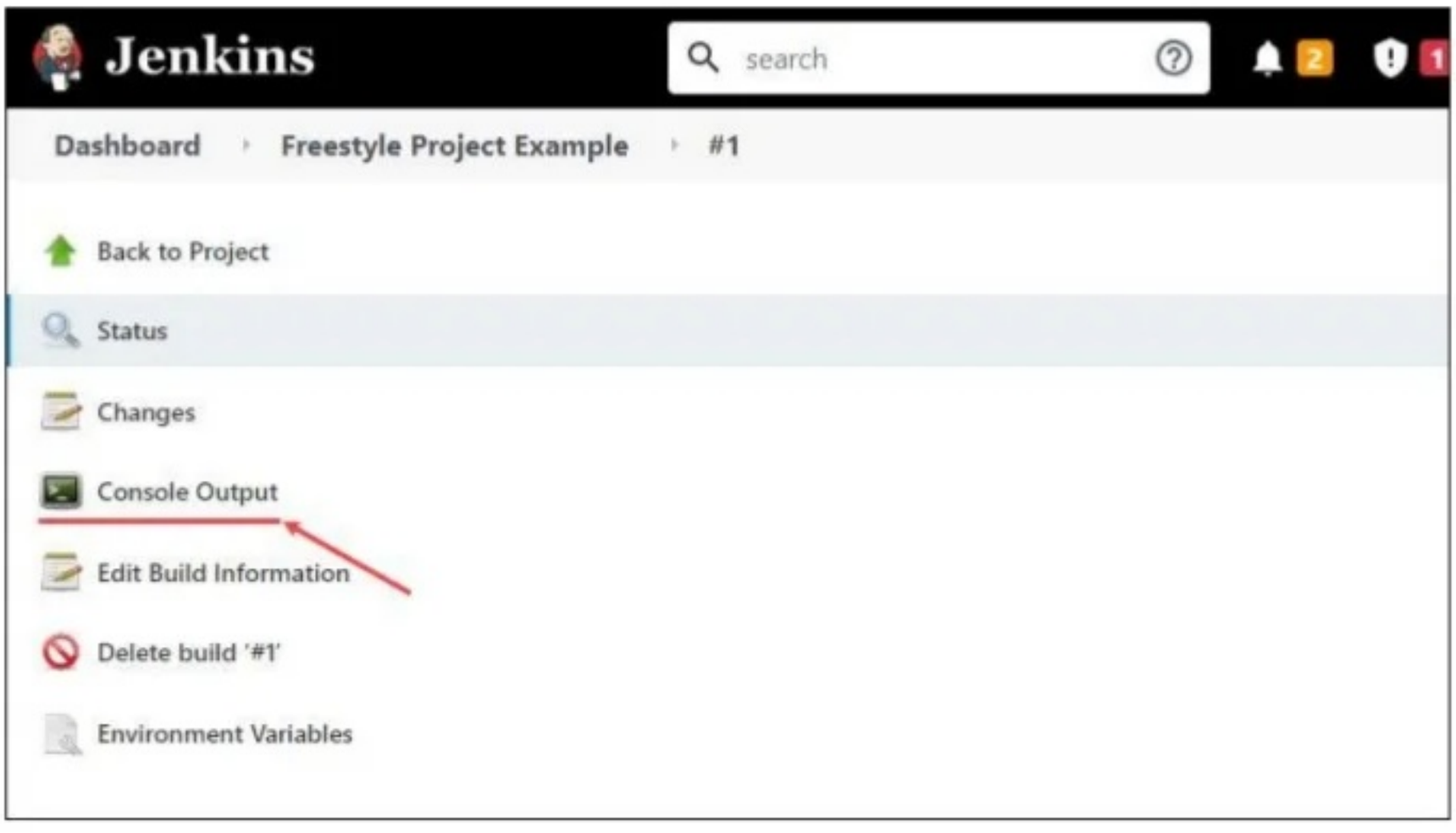

**Step 3: Build the Project**

1. Click the **Build Now** link on the left-hand side of the new project page.




2. Click the link to the latest project build in the *Build History* section.


3. Click the **Console Output** link on the left-hand side to display the output for the commands you entered.




4. The console output indicates that Jenkins is successfully executing the commands, displaying the current version of Java and Jenkins working directory.

***Output Screenshots:***

***Conclusion:***

You should be able to create and run your first Jenkins freestyle project.

***References:***

# EXPERIMENT NO. - 05

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission: _____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 5

***Problem Definition:*** To create a CI/CD pipeline in Jenkins

***Compiler / Tool :*** Jenkins ,Java 8 or Java 11, Modern web browsers - Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Apple Safari.

***Theory:***

**What is a CI/CD pipeline?**

A CI/CD pipeline automates the process of software delivery. It builds code, runs tests, and helps you to safely deploy a new version of the software. CI/CD pipeline reduces manual errors, provides feedback to developers, and allows fast product iterations.

CI/CD pipeline introduces automation and continuous monitoring throughout the lifecycle of a software product. It involves from the integration and testing phase to delivery and deployment. These connected practices are referred as CI/CD pipeline.

**What is Continuous Integration, Continuous Delivery, and Continuous Deployment?**

- **Continuous integration** is a software development method where members of the team can integrate their work at least once a day. In this method, every integration is checked by an automated build to search the error.
- **Continuous delivery** is a software engineering method in which a team develops software products in a short cycle. It ensures that software can be easily released at any time.
- **Continuous deployment** is a software engineering process in which product functionalities are delivered using automatic deployment. It helps testers to validate whether the codebase changes are correct, and it is stable or not.

***Implementation:***

**To Build a CI/CD Pipeline With Jenkins**

Go to your Jenkins Portal:

- Click on 'Create a job'.
- In the item name dialog box, you may enter the 'pipeline'.
- Select the pipeline job type in the list below.
- Click on OK.

A configuration related to the pipeline opens on the screen.

- Scroll down on that page.
- There in the dialog box, choose GitHub+Maven.

The next step is to integrate the Jenkins file into the Version Control system.

So, to do that, you must:

- Select 'Pipeline script from SCM'.

- Then in the SCM dialog box, select Git.
- 'Jenkins file' is the name of the Script.
- Add the Git repository URL.
- You can add the credentials if any.

The credentials can be added with the help of the 'Add' option.

- Then save the configuration

A page now appears on the screen that gives you various options like 'Build Now', 'Delete Pipeline', 'Configure', etc.

- Click on the Build Now option.

The pipeline will start downloading. The checkout will be visible on the screen and you can see the build being complete on the screen.

You can go to the console output option to check the log that is taking place.

***Output Screenshots:***

***Conclusion:***

Pipelines allow you to focus on the app and the business logic while the tooling handles the rest.

***References:***

## EXPERIMENT NO. - 06

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 6

***Problem Definition:*** To install Docker and execute basic command in Docker.

***Compiler / Tool :*** Docker

***Installation:***

**Step-By-Step Docker Installation on Windows**

1. Go to the website https://docs.docker.com/docker-for-windows/install/ and download the docker file.

Note: A 64-bit processor and 4GB system RAM are the hardware prerequisites required to successfully run Docker on Windows 10.

2. Then, double-click on the Docker Desktop Installer.exe to run the installer.

Note: Suppose the installer (Docker Desktop Installer.exe) is not downloaded; you can get it from Docker Hub and run it whenever required.

3. Once you start the installation process, always enable Hyper-V Windows Feature on the Configuration page.

4. Then, follow the installation process to allow the installer and wait till the process is done.

5. After completion of the installation process, click Close and restart.

***Implementation:***

**Run/Start Docker Application on Windows**

Once you have successfully installed the docker desktop application, you will have to start the application as it will not start automatically after installation.

Go to the **start menu**, and **search for Docker**, **click on the Docker Desktop**.

You will see a **moving whale icon** appear in your taskbar, which means the docker desktop application is getting started. Wait for this whale icon to stop moving and become steady, which means docker has started.

Once the docker desktop application starts, you can use the **windows command prompt to run docker commands**.

To verify the installation, run the following command to check the version of the docker installed:

```
docker --version
```

***Output Screenshots:***

***Conclusion:***

Bascis docker commands are executed.

***References:***

## EXPERIMENT NO. - 07

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 7

***Problem Definition:*** To build image from docker file.

***Compiler / Tool :*** Java Compile , Eclipse IDE

***Theory:***

The Docker engine includes tools that automate container image creation. While you can create container images manually by running the `docker commit` command, adopting an automated image creation process has many benefits, including:

- Storing container images as code.
- Rapid and precise recreation of container images for maintenance and upgrade purposes.
- Continuous integration between container images and the development cycle.

The Docker components that drive this automation are the Dockerfile, and the `docker build` command.

The Dockerfile is a text file that contains the instructions needed to create a new container image. These instructions include identification of an existing image to be used as a base, commands to be run during the image creation process, and a command that will run when new instances of the container image are deployed.

Docker build is the Docker engine command that consumes a Dockerfile and triggers the image creation process.

***Implementation:***

**Basic Syntax**

In its most basic form, a Dockerfile can be very simple. The following example creates a new image, which includes IIS, and a 'hello world' site.

A Dockerfile must be created with no extension. To do this in Windows, create the file with your editor of choice, then save it with the notation "Dockerfile" (including the quotes).

```
# Sample Dockerfile

# Indicates that the windowsservercore image will be used as the base image.

FROM mcr.microsoft.com/windows/servercore:ltsc2019

# Metadata indicating an image maintainer.
```

```
LABEL maintainer="jshelton@contoso.com"

# Uses dism.exe to install the IIS role.

RUN dism.exe /online /enable-feature /all /featurename:iis-webserver /NoRestart

# Creates an HTML file and adds content to this file.

RUN echo "Hello World - Dockerfile" > c:\inetpub\wwwroot\index.html

# Sets a command or process that will run each time a container is run from the new image.

CMD [ "cmd" ]
```

**Instructions**

Dockerfile instructions provide the Docker Engine the instructions it needs to create a container image. These instructions are performed one-by-one and in order. The following examples are the most commonly used instructions in Dockerfiles.

**FROM**

The `FROM` instruction sets the container image that will be used during the new image creation process. For instance, when using the instruction `FROM mcr.microsoft.com/windows/servercore`, the resulting image is derived from, and has a dependency on, the Windows Server Core base OS image. If the specified image is not present on the system where the Docker build process is being run, the Docker engine will attempt to download the image from a public or private image registry.

The FROM instruction's format goes like this:

```
FROM <image>
```

To download the ltsc2019 version windows server core from the Microsoft Container Registry (MCR):

```
FROM mcr.microsoft.com/windows/servercore:ltsc2019
```

**RUN**

The `RUN` instruction specifies commands to be run, and captured into the new container image. These commands can include items such as installing software, creating files and directories, and creating environment configuration.

The RUN instruction goes like this:

```
# exec form
```

```
RUN ["<executable>", "<param 1>", "<param 2>"]

# shell form

RUN <command>
```

The difference between the exec and shell form is in how the `RUN` instruction is executed. When using the exec form, the specified program is run explicitly.

Here's an example of the exec form:

```
FROM mcr.microsoft.com/windows/servercore:ltsc2019
RUN ["powershell", "New-Item", "c:/test"]
```

The resulting image runs the `powershell New-Item c:/test` command:

```
docker history doc-exe-method

IMAGE               CREATED             CREATED BY                      SIZE
COMMENT
b3452b13e472        2 minutes ago       powershell New-Item c:/test     30.76 MB
```

***Output Screenshots:***

***Conclusion:***
We have learn how to use Dockerfiles with Windows containers

***References:***

## EXPERIMENT NO. - 08

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 8

***Problem Definition:*** To deploy java application into docker.

***Compiler / Tool:*** Docker, Jdk

***Implementation:***

Step 1: install Java inside Docker

Step 2: install the app in your Docker container

***Output Screenshots:***

***Conclusion:***

It's an easy way to deploy, run, and manage applications using containers that are independent of elements like hardware and language, which makes these containers highly portable.

***References:***

## EXPERIMENT NO. - 09

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 9

***Problem Definition:*** To perform continuous testing of web applications using Selenium.

***Compiler / Tool:*** Eclipse IDE, Selenium

***Implementation:***

Selenium installation is a 3 step process:

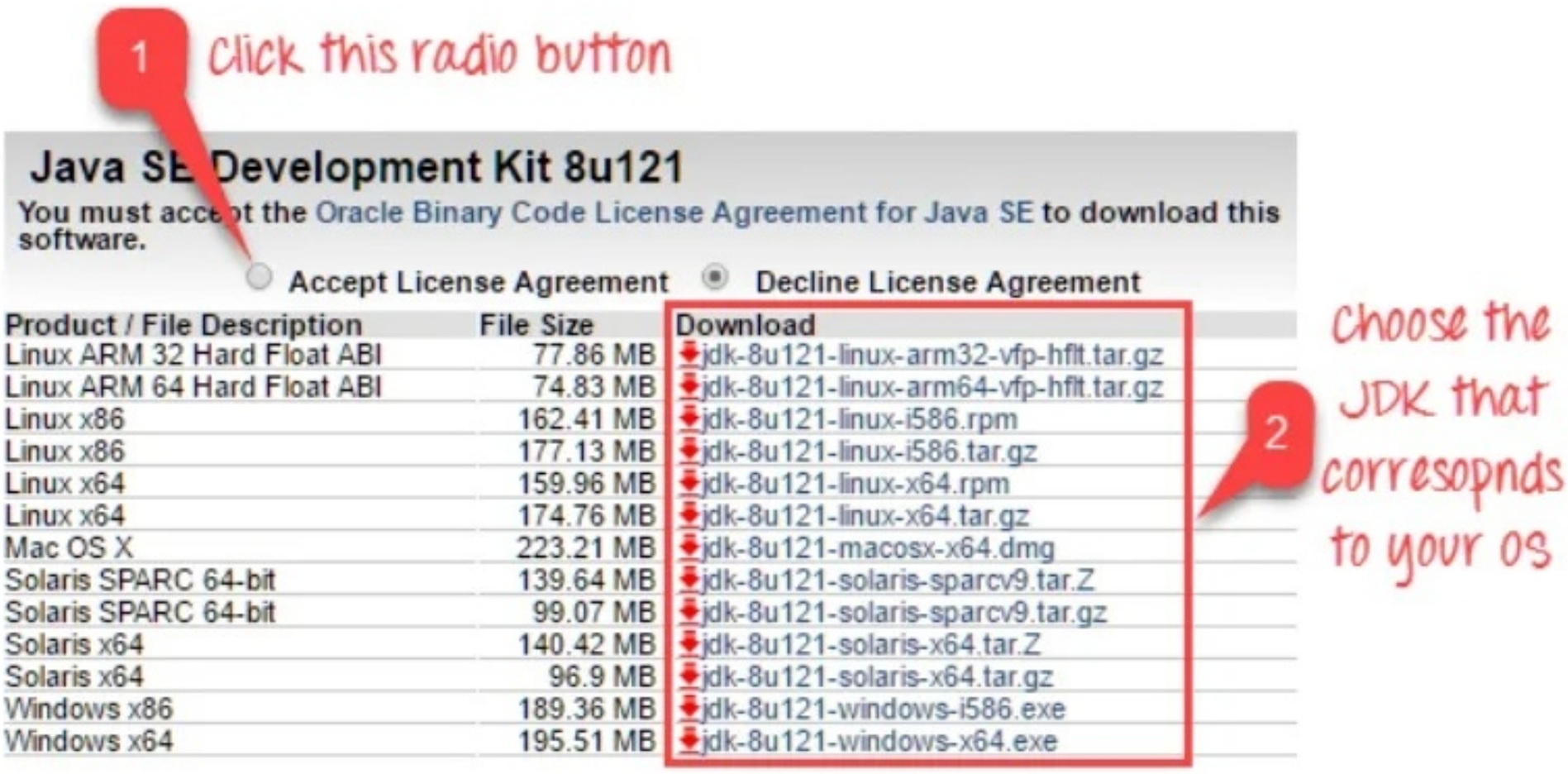
1. Install Java SDK
2. Install Eclipse
3. Install Selenium Webdriver Files

## Step 1 – Install Java on your computer

Download and install the **Java Software Development Kit (JDK)** here.

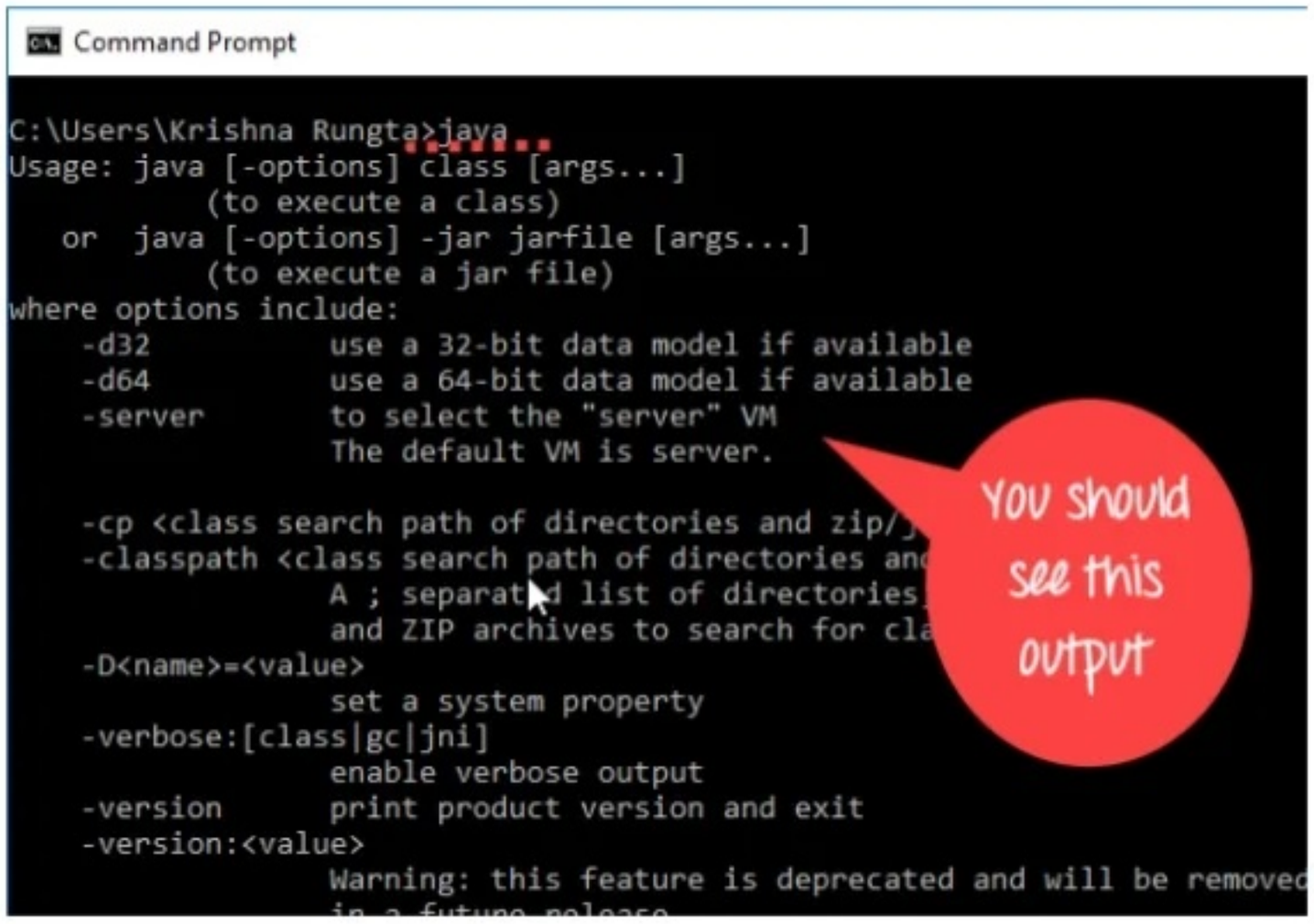


Next –




This JDK version comes bundled with Java Runtime Environment (JRE), so you do not need to download and install the JRE separately.

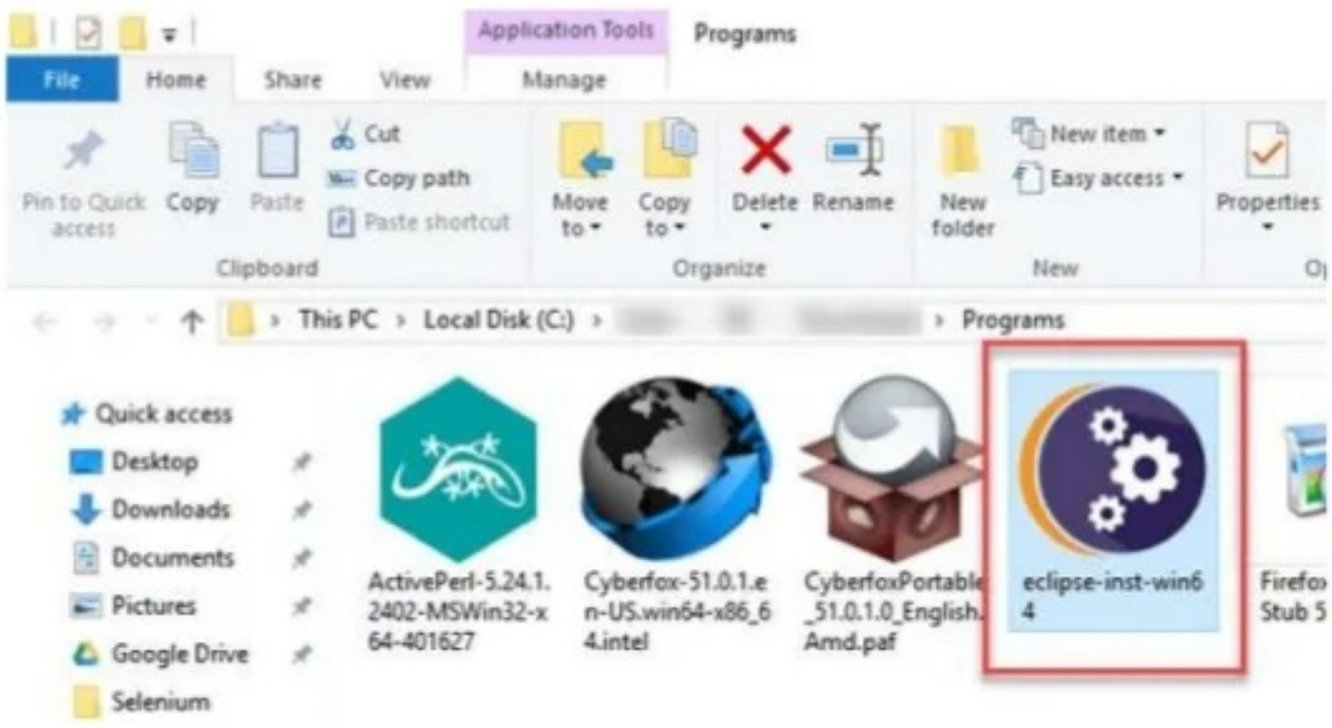
Once installation is complete, open command prompt and type "java". If you see the following screen you are good to move to the next step




## Step 2 – Install Eclipse IDE

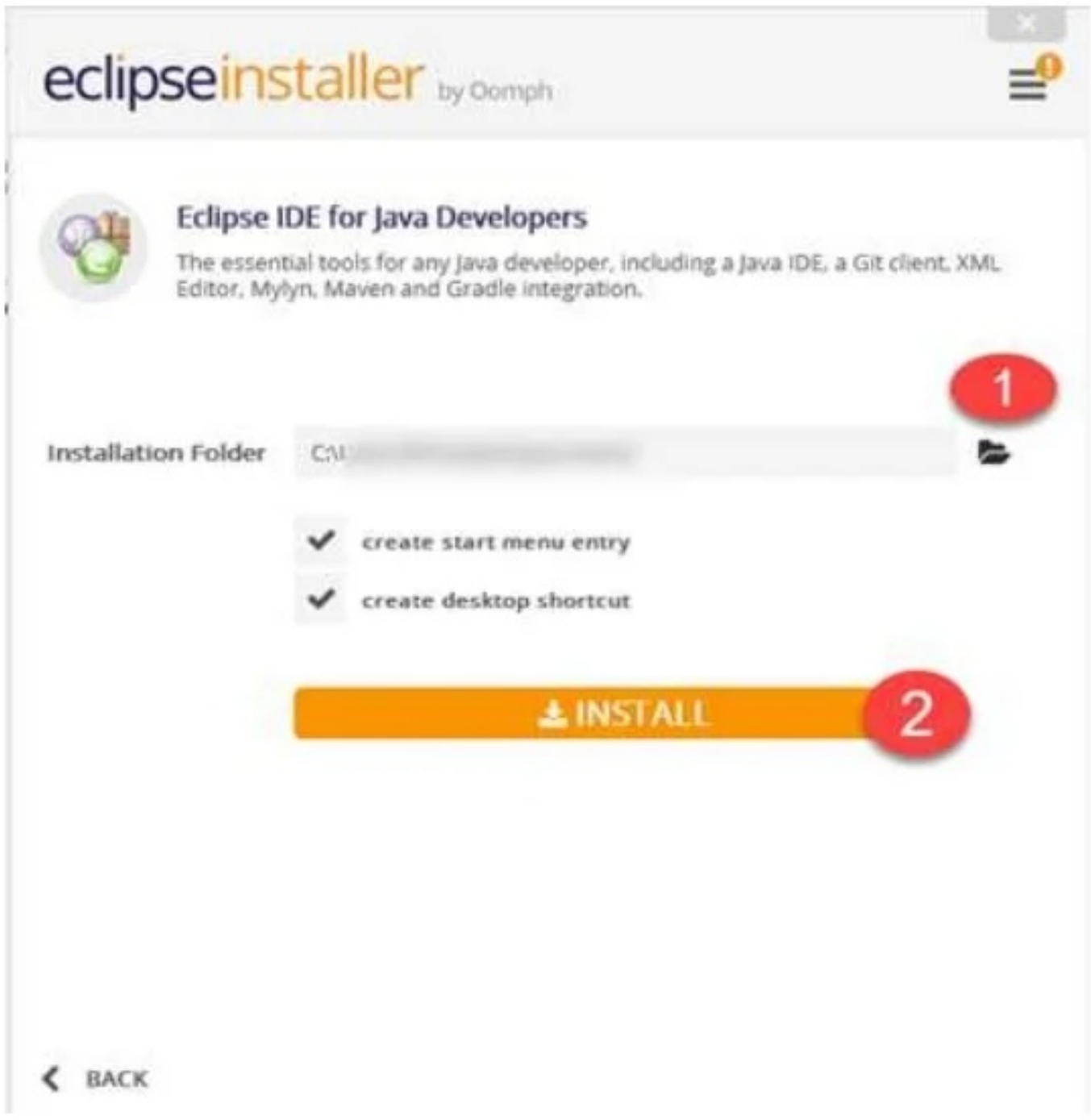
Download latest version of **"Eclipse IDE for Java Developers"** here. Be sure to choose correctly between Windows 32 Bit and 64 Bit versions.


You should be able to download an exe file named "eclipse-inst-win64" for Setup.

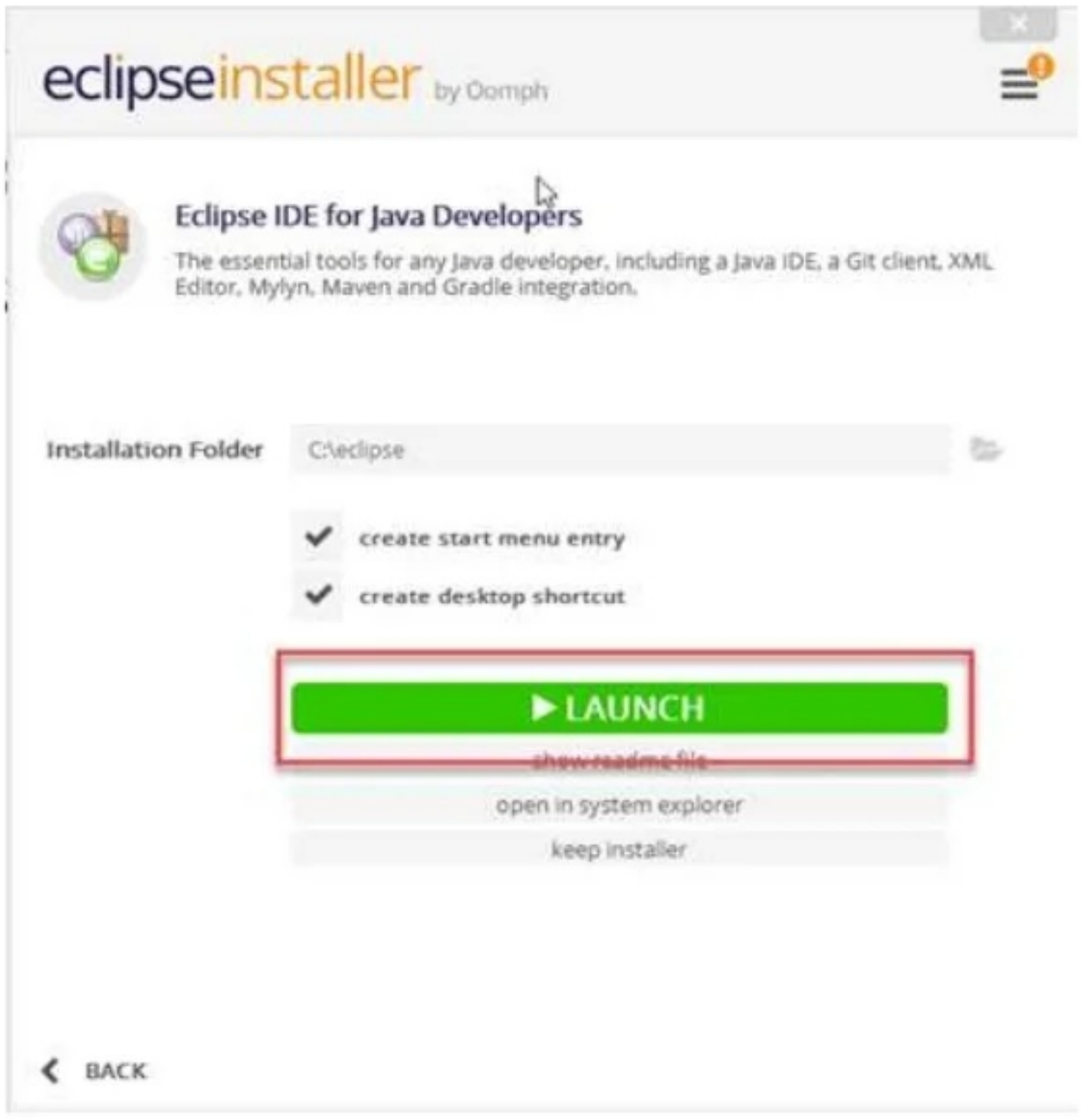



Double-click on file to Install the Eclipse. A new window will open. Click Eclipse IDE for Java Developers.


After that, a new window will open which click button marked 1 and change path to "C:\eclipse". Post that Click on Install button marked 2




After successful completion of the installation procedure, a window will appear. On that window click on Launch



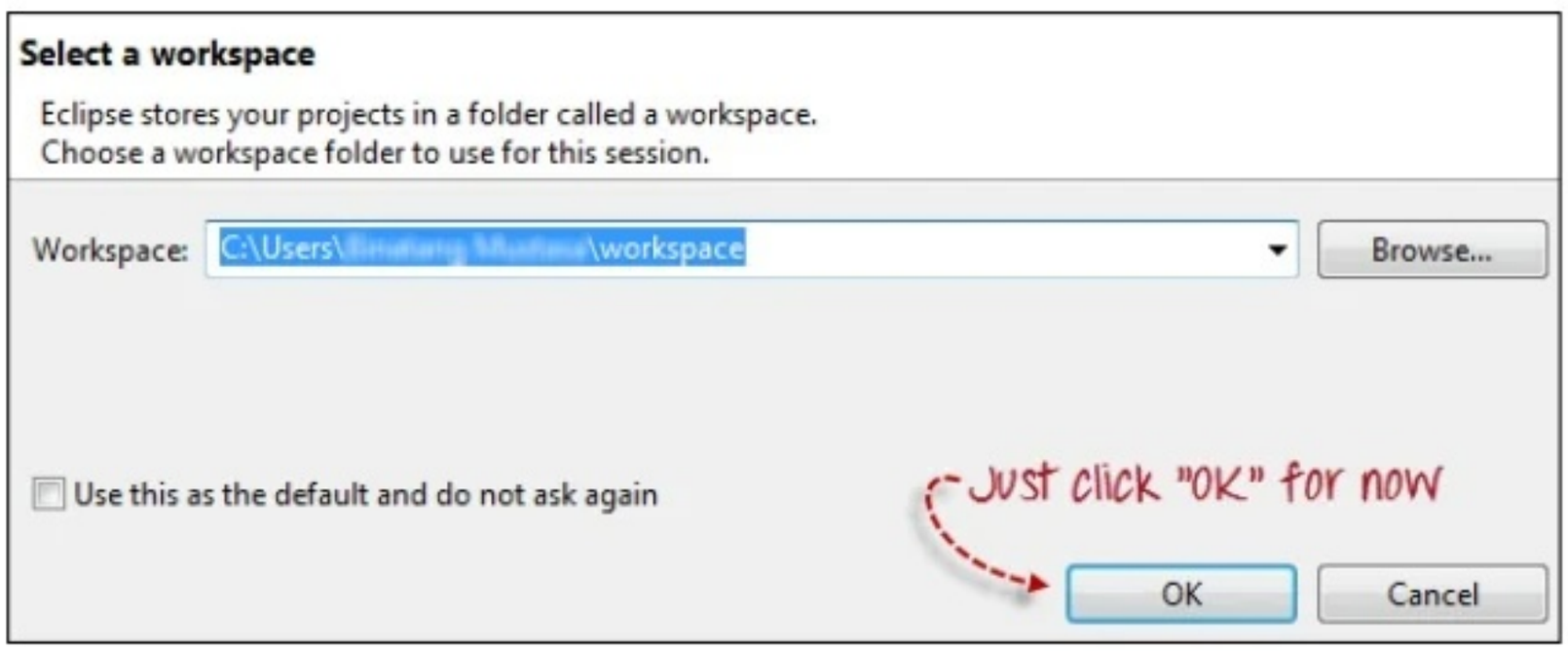

This will start eclipse neon IDE for you.

## Step 3 – Download the Selenium Java Client Driver

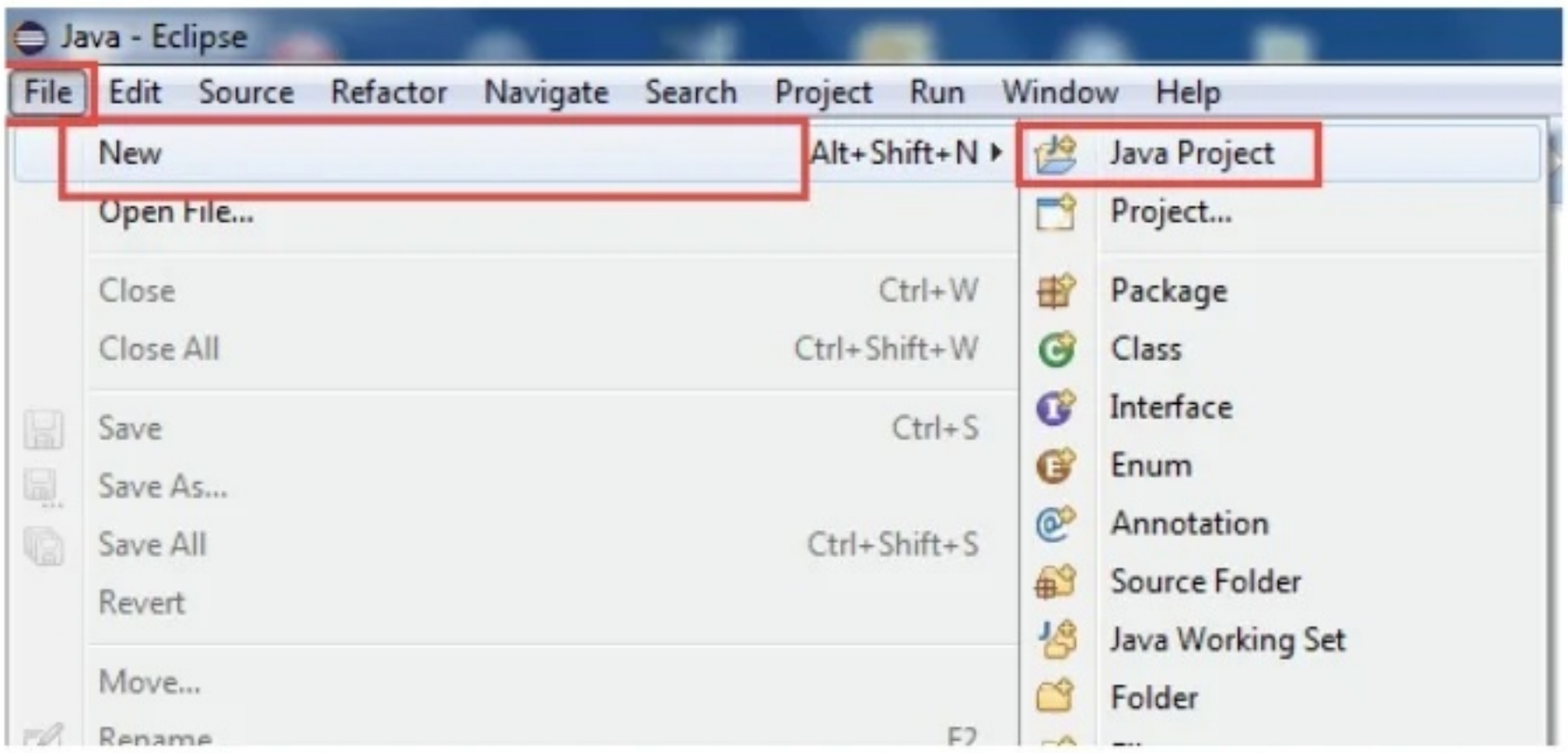
You can download **Selenium Webdriver for Java Client Driver** here. You will find client drivers for other languages there, but only choose the one for Java.


## Step 4 – Configure Eclipse IDE with WebDriver

1. Launch the "eclipse.exe" file inside the "eclipse" folder that we extracted in step 2. If you followed step 2 correctly, the executable should be located on C:\eclipse\eclipse.exe.
2. When asked to select for a workspace, just accept the default location.

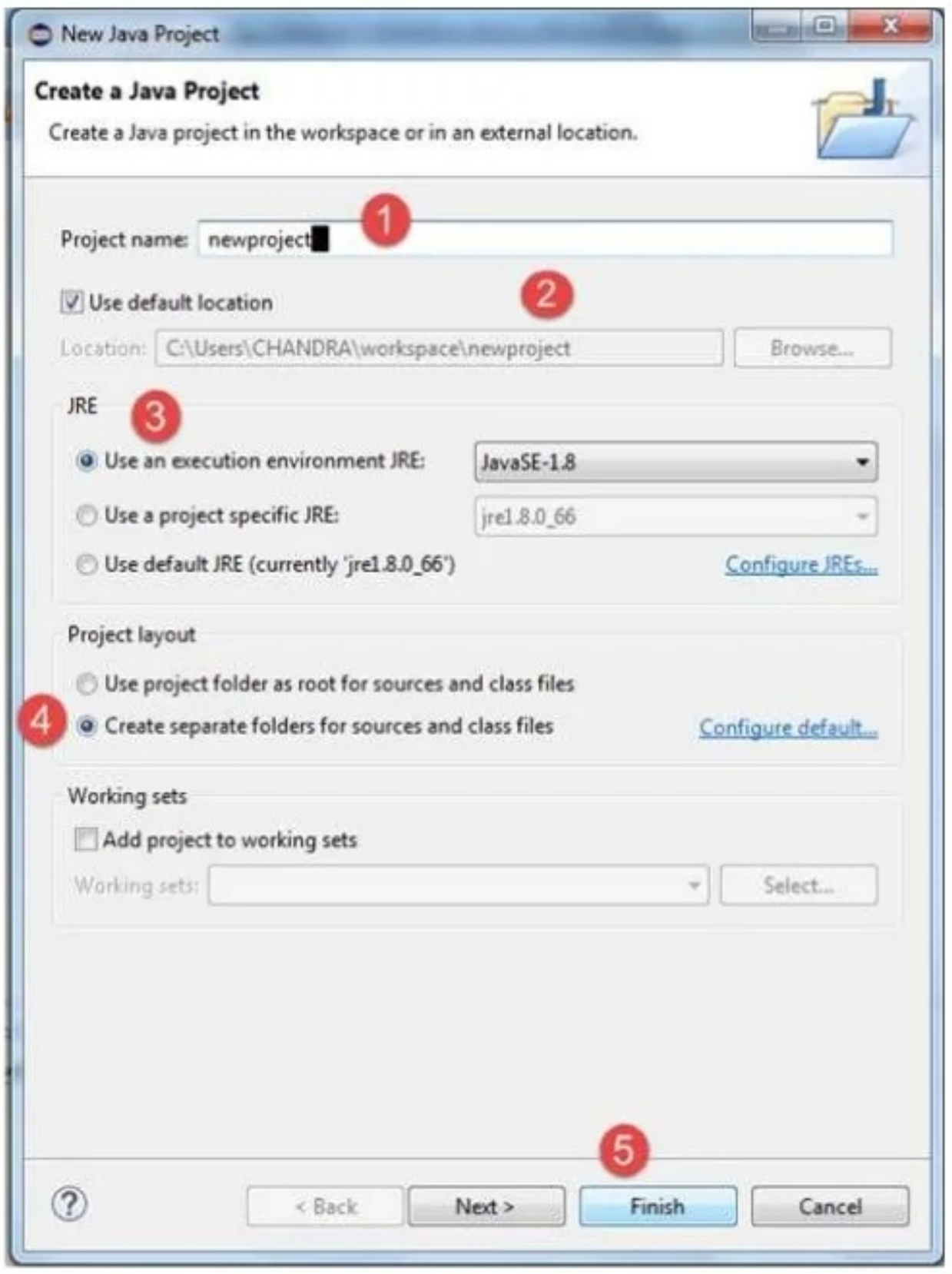



3. Create a new project through File > New > Java Project. Name the project as "newproject".



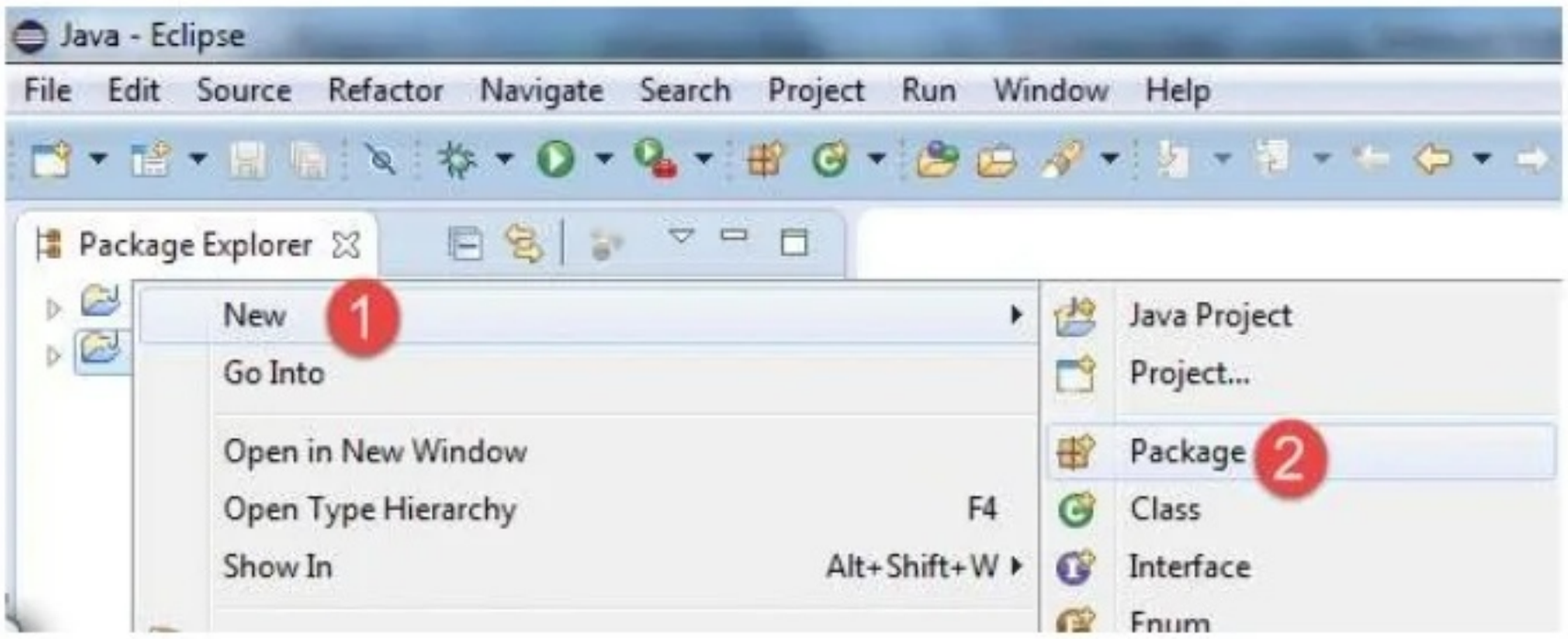

A new pop-up window will open enter details as follow

1. Project Name
2. Location to save project
3. Select an execution JRE
4. Select layout project option
5. Click on Finish button



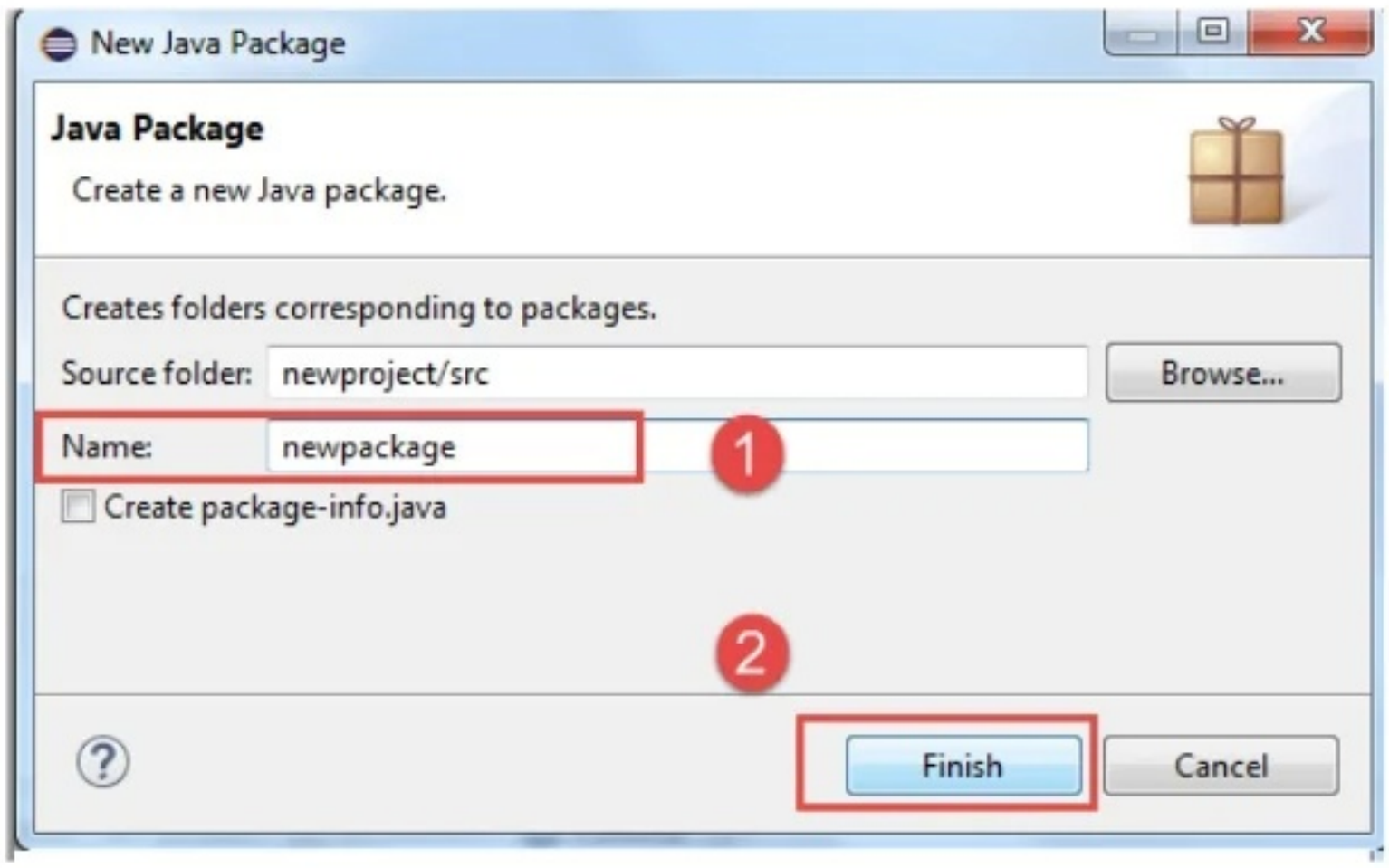

4. In this step,

1. Right-click on the newly created project and
2. Select New > Package, and name that package as "newpackage".

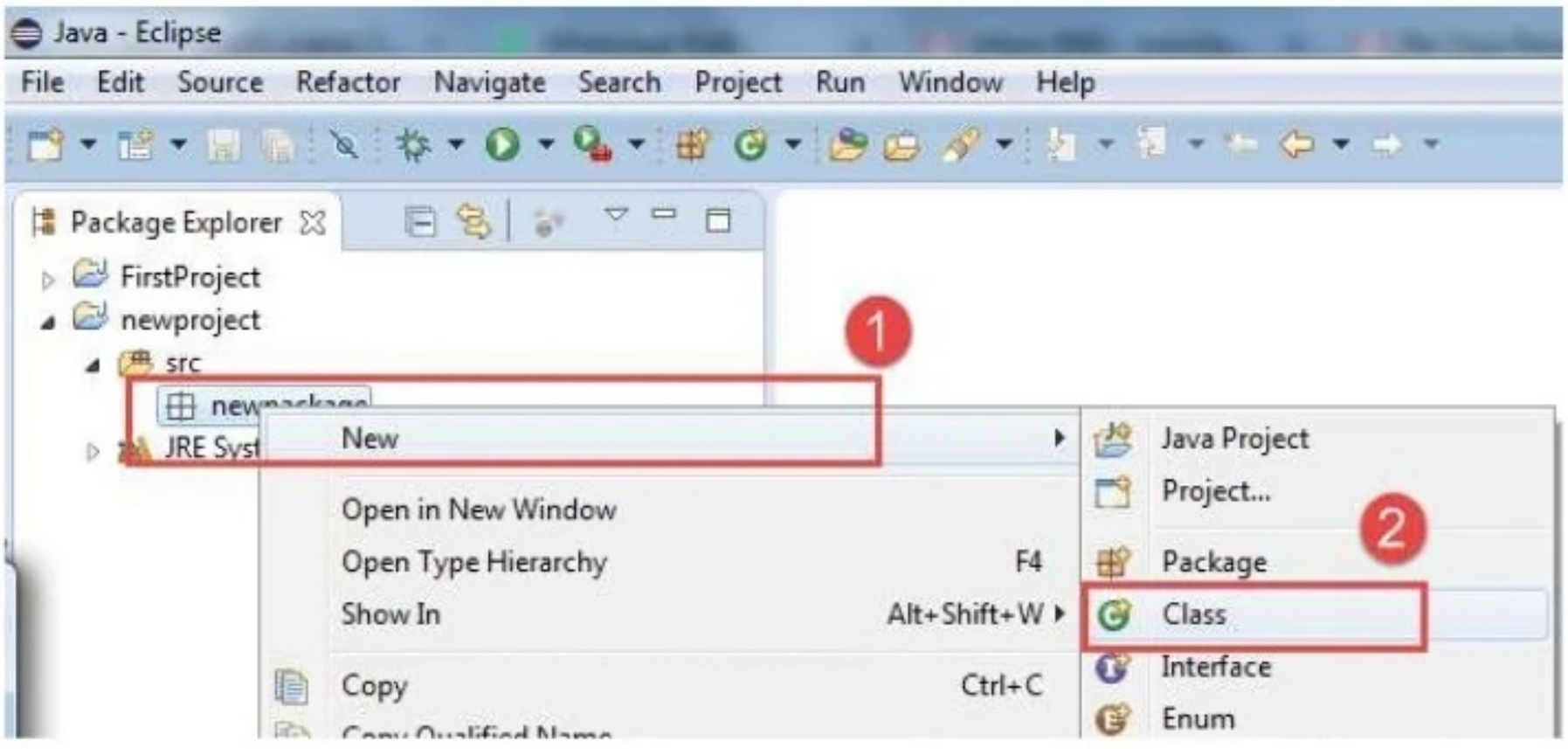



A pop-up window will open to name the package,

1. Enter the name of the package
2. Click on Finish button

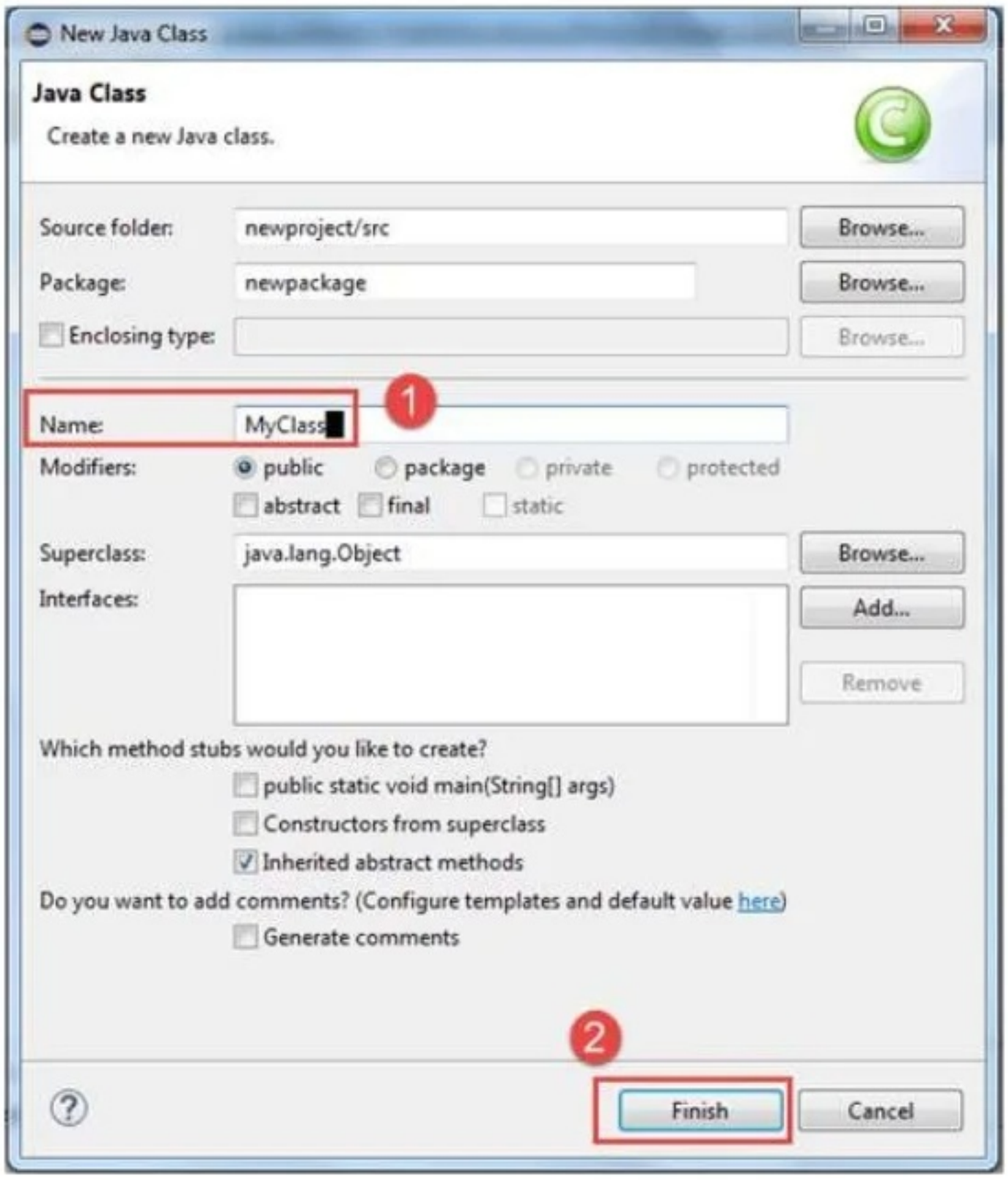



5. Create a new Java class under newpackage by right-clicking on it and then selecting- New > Class, and then name it as "MyClass". Your Eclipse IDE should look like the image below.

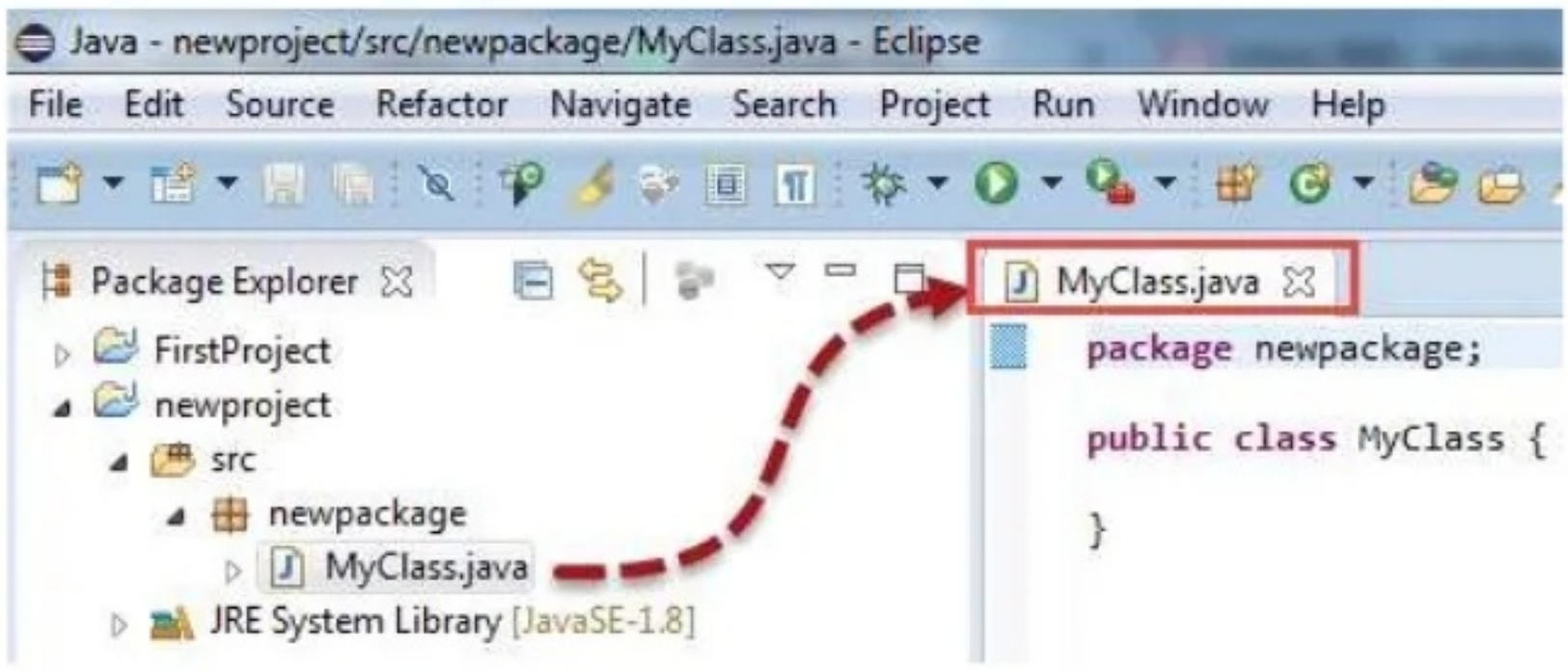



When you click on Class, a pop-up window will open, enter details as

1. Name of the class
2. Click on Finish button



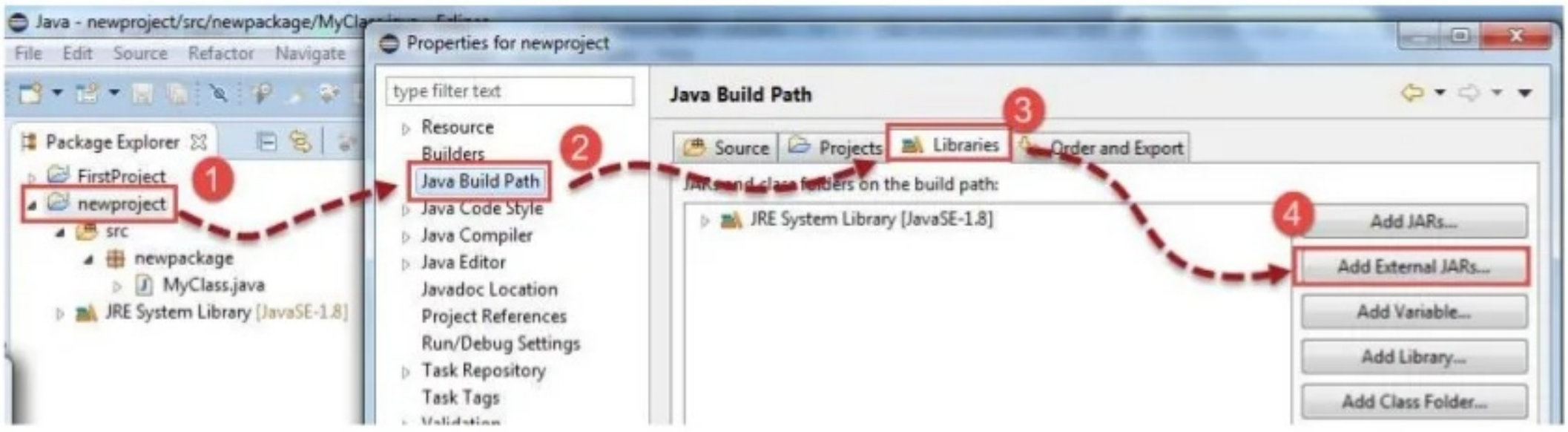

This is how it looks like after creating class.



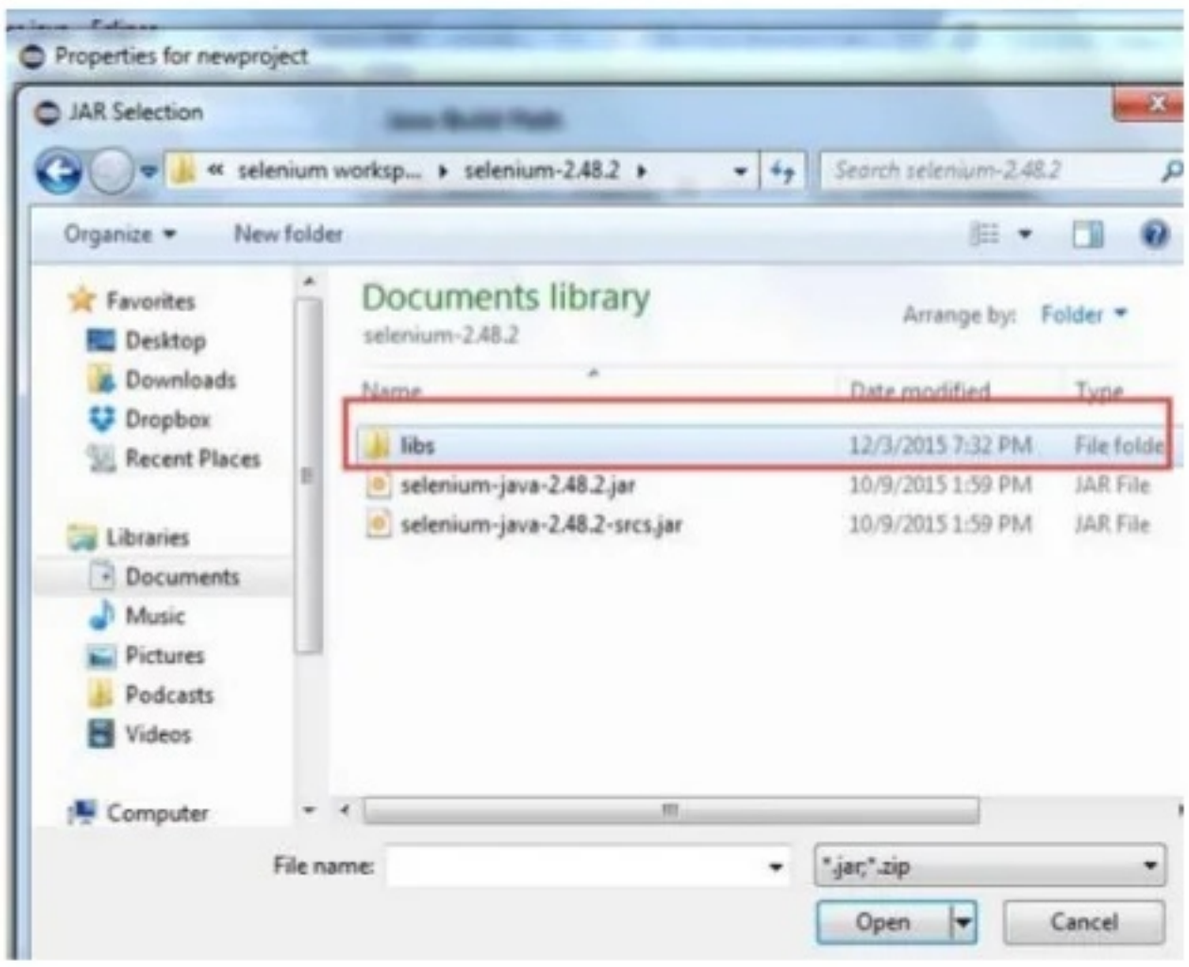

Now selenium WebDriver's into Java Build Path

In this step,

1. Right-click on "newproject" and select **Properties**.
2. On the Properties dialog, click on "Java Build Path".
3. Click on the **Libraries** tab, and then
4. Click on "Add External JARs.."

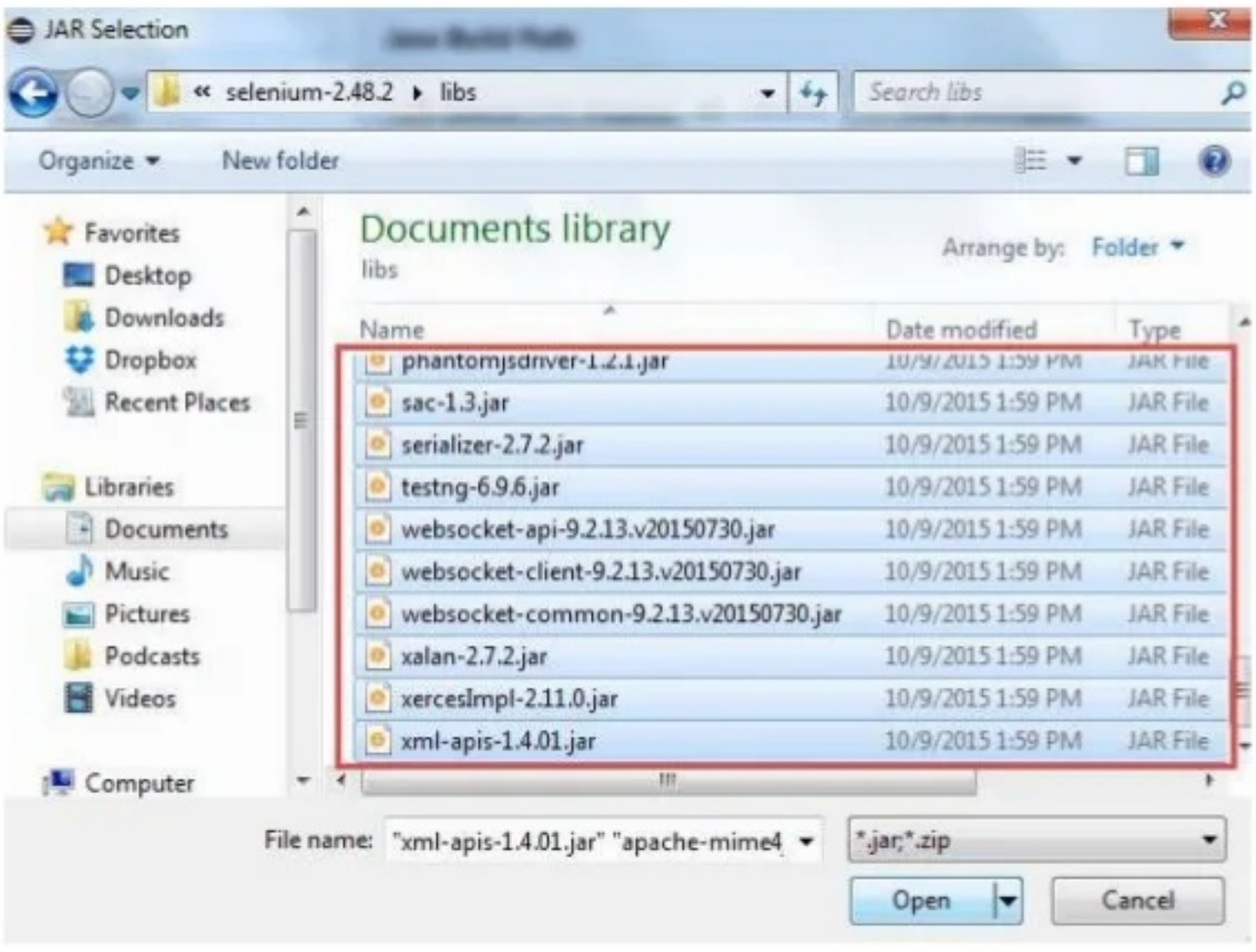



When you click on "Add External JARs.." It will open a pop-up window. Select the JAR files you want to add.

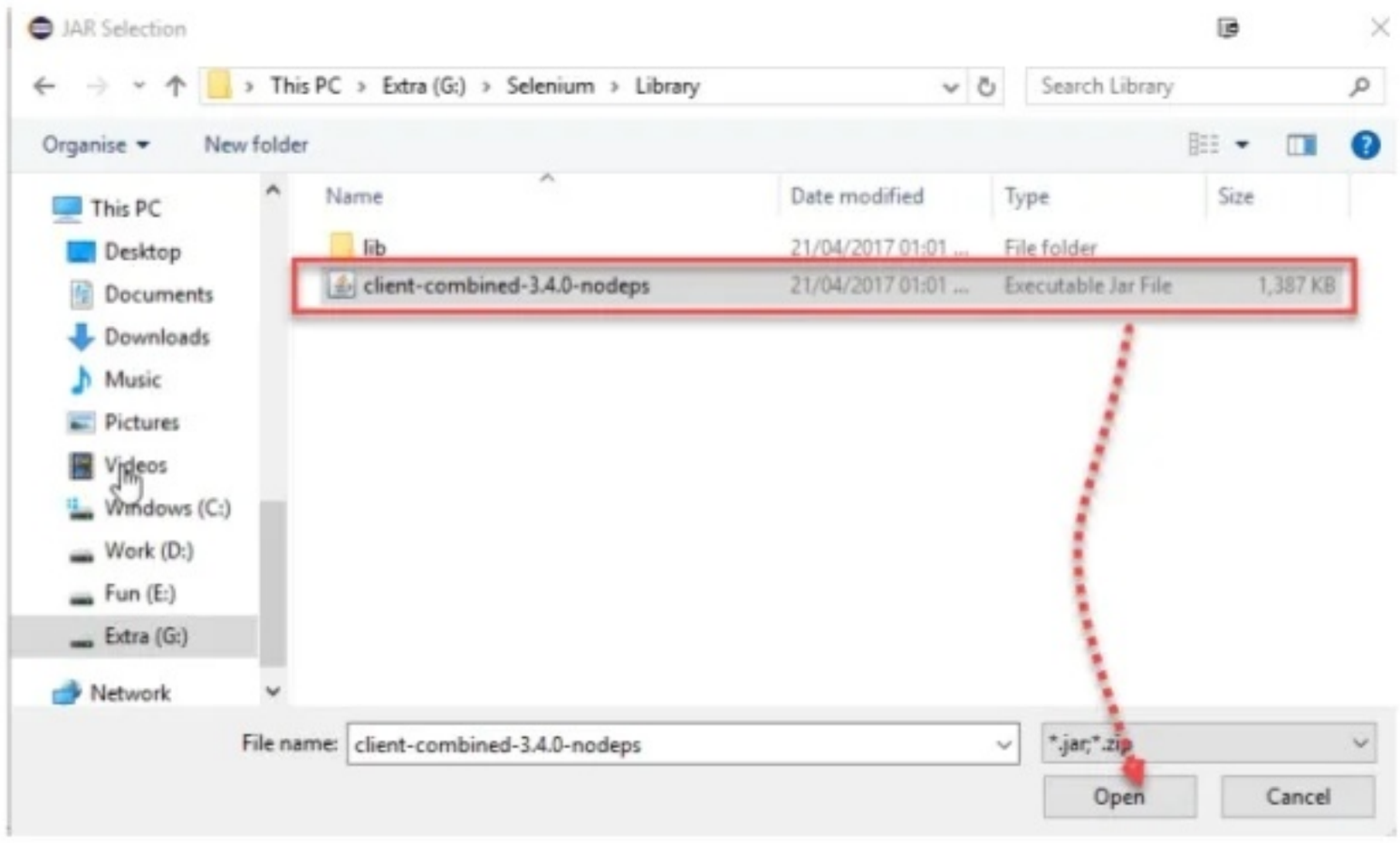



After selecting jar files, click on OK button.

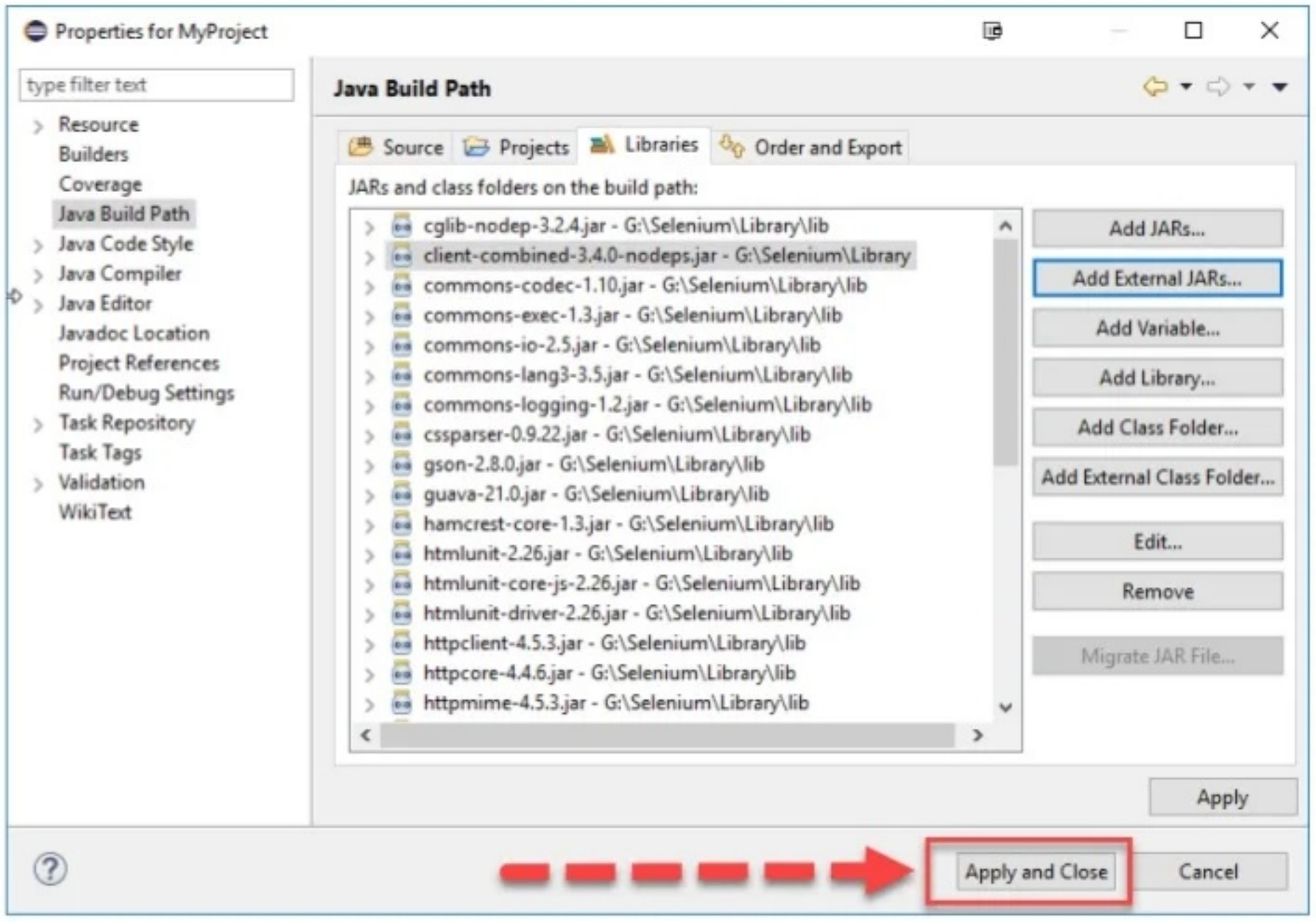
Select all files inside the lib folder.



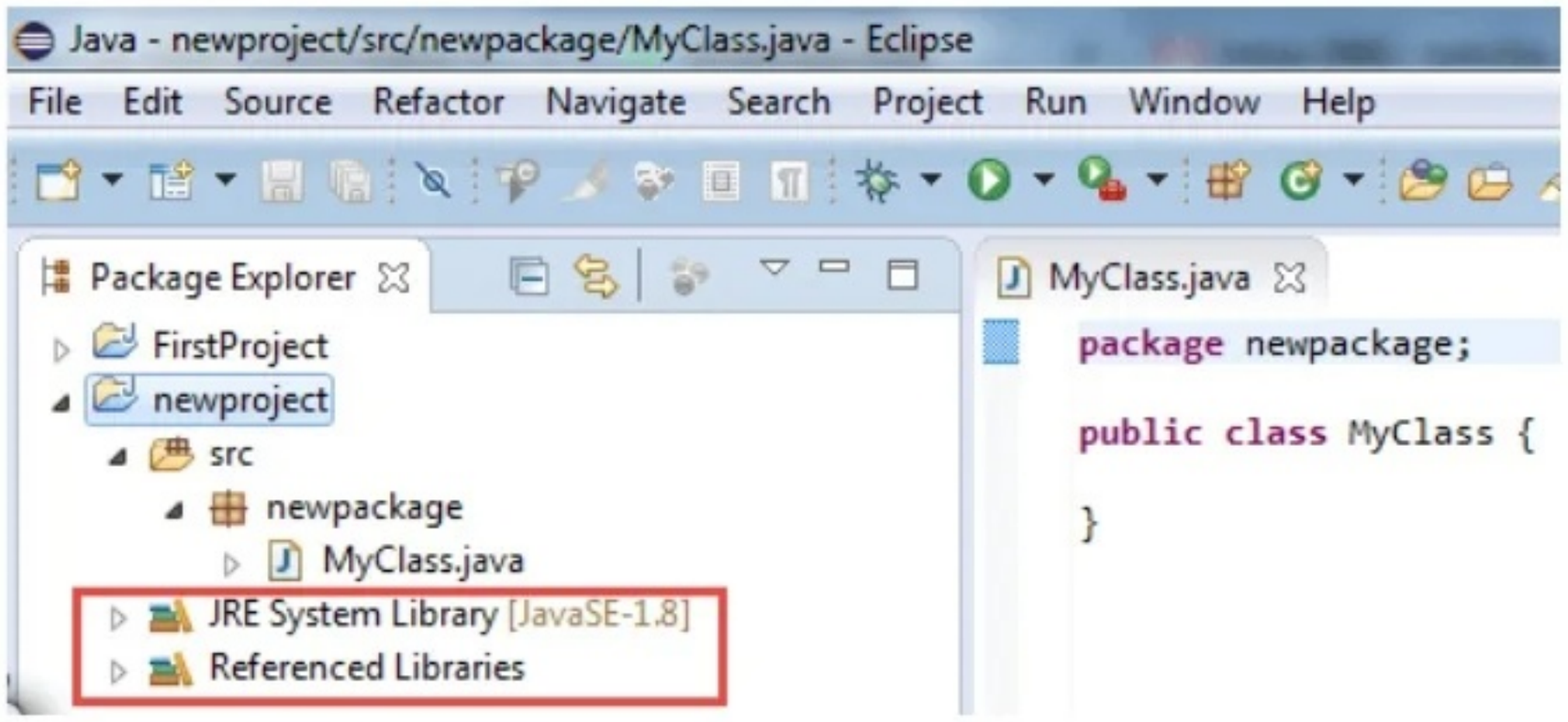

Select files outside lib folder




Once done, click "Apply and Close" button




6. Add all the JAR files inside and outside the "libs" folder. Your Properties dialog should now look similar to the image below.




7. Finally, click OK and we are done importing Selenium libraries into our project.

***Output Screenshots:***

***Conclusion:***

Testing of web app has been done using selenium

***References:***

## EXPERIMENT NO. - 10

Aim of the experiment :-

Course Outcome :-

Date of Conduction : _____ Date of Submission:_____

| Implementation (5) | Understanding (5) | Punctuality & Discipline (5) | Total (15) |
|---|---|---|---|
| | | | |

_____

Practical Incharge

# PRACTICAL NO. 10

***Problem Definition:*** Install puppet agent and puppet master on two separate virtual machines and establish connection between them.

***Compiler / Tool:*** Puppet, Oracle Virtual Box,Ubuntu

***Theory:***

Puppet is a configuration management tool that simplifies system administration. Puppet uses a client/server model in which your managed nodes, running a process called the Puppet *agent*, talk to and pull down configuration profiles from a Puppet *master*.

Puppet deployments can range from small groups of servers up to enterprise-level operations.

This guide will demonstrate how to install Puppet 6.1 on three servers:

- A Puppet master running Ubuntu 18.04
- A managed Puppet Agent node running Ubuntu 18.04

**What is puppet?**

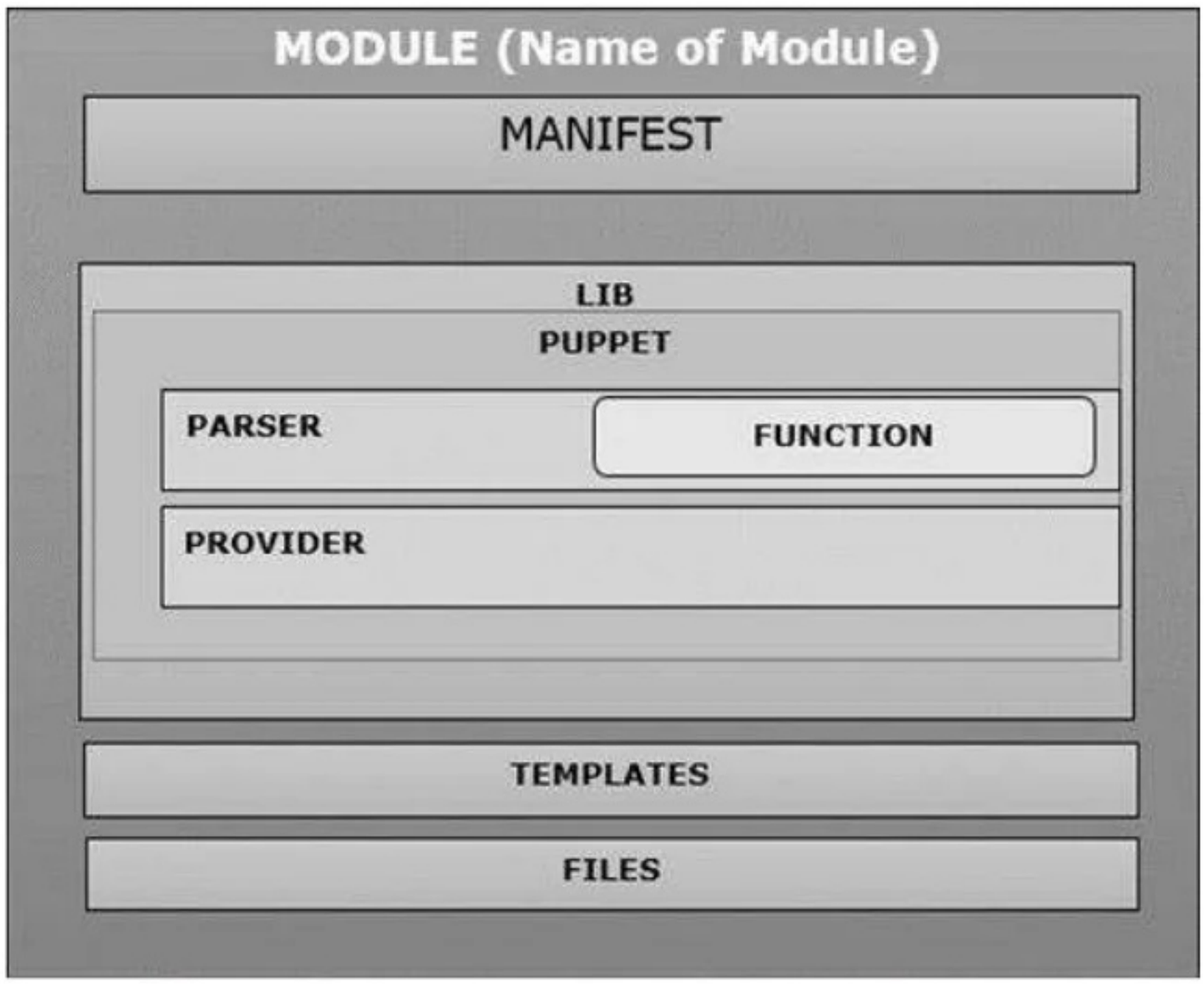
Puppet is a configuration management tool developed by Puppet Labs in order to automate infrastructure management and configuration. Puppet is a very powerful tool which helps in the concept of Infrastructure as code. Puppet follows the client-server model, where one machine in any cluster acts as the server, known as puppet master and the other acts as a client known as a slave on nodes. Puppet has the capability to manage any system from scratch, starting from initial configuration till the end-of-life of any particular machine.

**Features of puppet**

Following are the most important features of Puppet:

- **Idempotency:** Puppet supports Idempotency which makes it unique. Idempotency helps in managing any particular machine throughout its lifecycle starting from the creation of machine, configurational changes in the machine, till the end-of-life. Puppet Idempotency feature is very helpful in keeping the machine updated for years rather than rebuilding the same machine multiple times, when there is any configurationally change.
- **Cross-platform:** In Puppet, with the help of Resource Abstraction Layer (RAL) which uses Puppet resources, one can target the specified configuration of system without worrying about the implementation details and how the configuration command will work inside the system, which are defined in the underlying configuration file.

**Key Components of puppet**




**Puppet Resources**

Puppet resources are the key components for modelling any particular machine. These resources have their own implementation model. Puppet uses the same model to get any particular resource in the desired state.

**Providers**

Providers are basically fulfillers of any particular resource used in Puppet. For example, the package type 'apt-get' and 'yum' both are valid for package management. Sometimes, more than one provider would be available on a particular platform. Though each platform always have a default provider.

**Manifest**

Manifest is a collection of resources which are coupled inside the function or classes to configure any target system. They contain a set of Ruby code in order to configure a system.

**Modules**

Module is the key building block of Puppet, which can be defined as a collection of resources, files, templates, etc. They can be easily distributed among different kinds of OS being defined that they are of the same flavour. As they can be easily distributed, one module can be used multiple times with the same configuration.

**Templates**

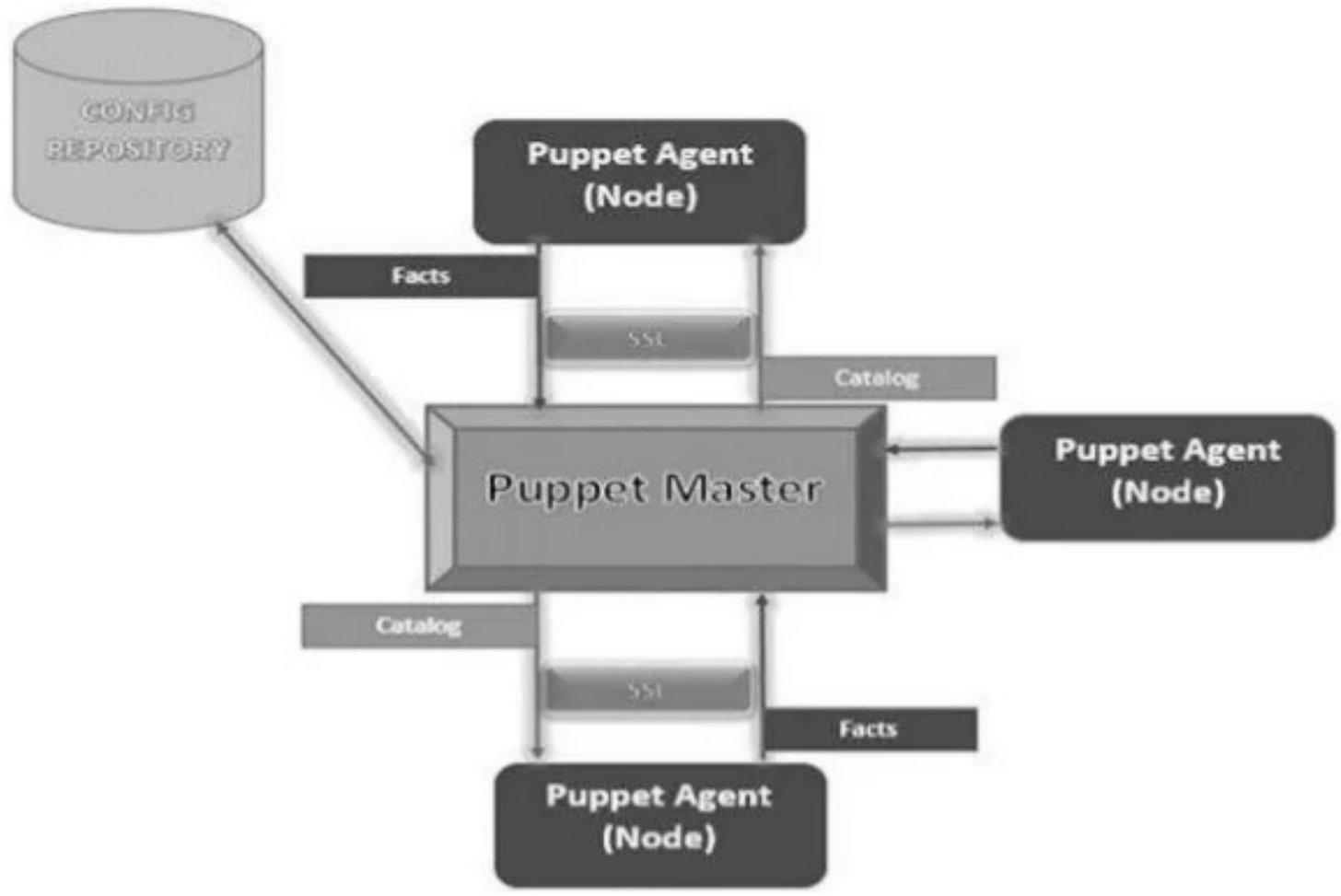
Templates use Ruby expressions to define the customized content and variable input.

**Static Files**

Static files can be defined as a general file which are sometimes required to perform specific tasks. They can be simply copied from one location to another using Puppet. All static files are located

inside the files directory of any module. Any manipulation of the file in a manifest is done using the file resource.

**Architecture of puppet**




**Puppet Master**

Puppet Master is the key mechanism which handles all the configuration related stuff. It applies the configuration to nodes using the Puppet agent.

**Puppet Agent**

Puppet Agents are the actual working machines which are managed by the Puppet master. They have the Puppet agent daemon service running inside them.

**Config Repository**

This is the repo where all nodes and server-related configurations are saved and pulled when required.

**Facts**

Facts are the details related to the node or the master machine, which are basically used for analysing the current status of any node. On the basis of facts, changes are done on any target machine. There are pre-defined and custom facts in Puppet.

**Catalog**

All the manifest files or configuration which are written in Puppet are first converted to a compiled format called catalog and later those catalogs are applied on the target machine

**How puppet master and puppet agent communicates?**

Master-agent communication follows this pattern:

1. An agent node sends facts to the master and requests a catalog.

2. The master compiles and returns the node's catalog using the sources of information the master has access to.

3. The agent applies the catalog to the node by checking each resource the catalog describes. If it finds resources that are not in their desired state, it makes the changes necessary to correct them. Or, in no-op mode, it assesses what changes would be needed to reconcile the catalog.

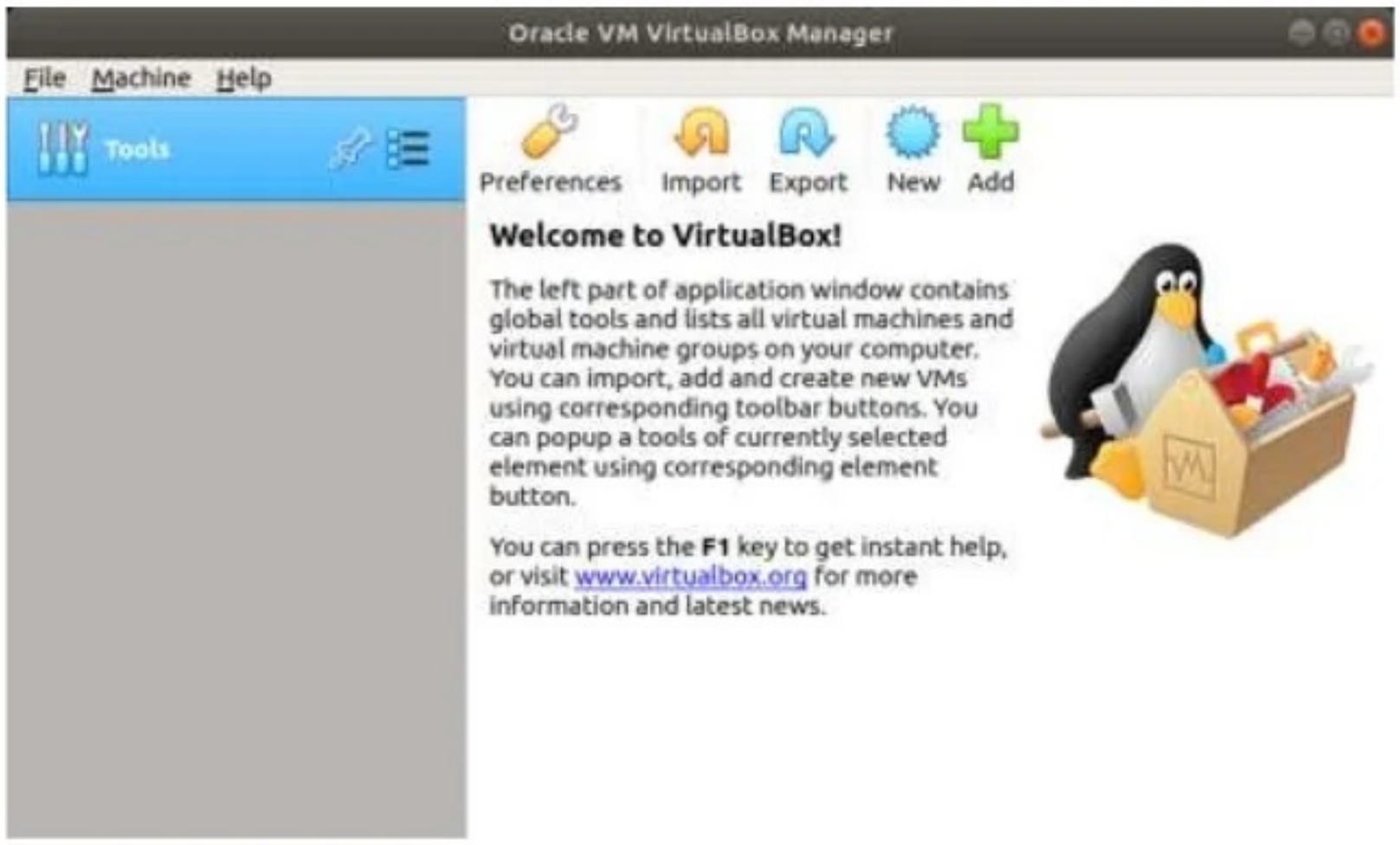
4. The agent sends a report back to the master.

***Implementation:***

Download Oracle virtualbox Debian Package from its official site and Install Oracle VirtualBox Manager using following command :

**ubuntu@ubuntu**$ dpkg -i virtualbox.deb

Following figure shows the initial appearance of the Oracle virtualbox.

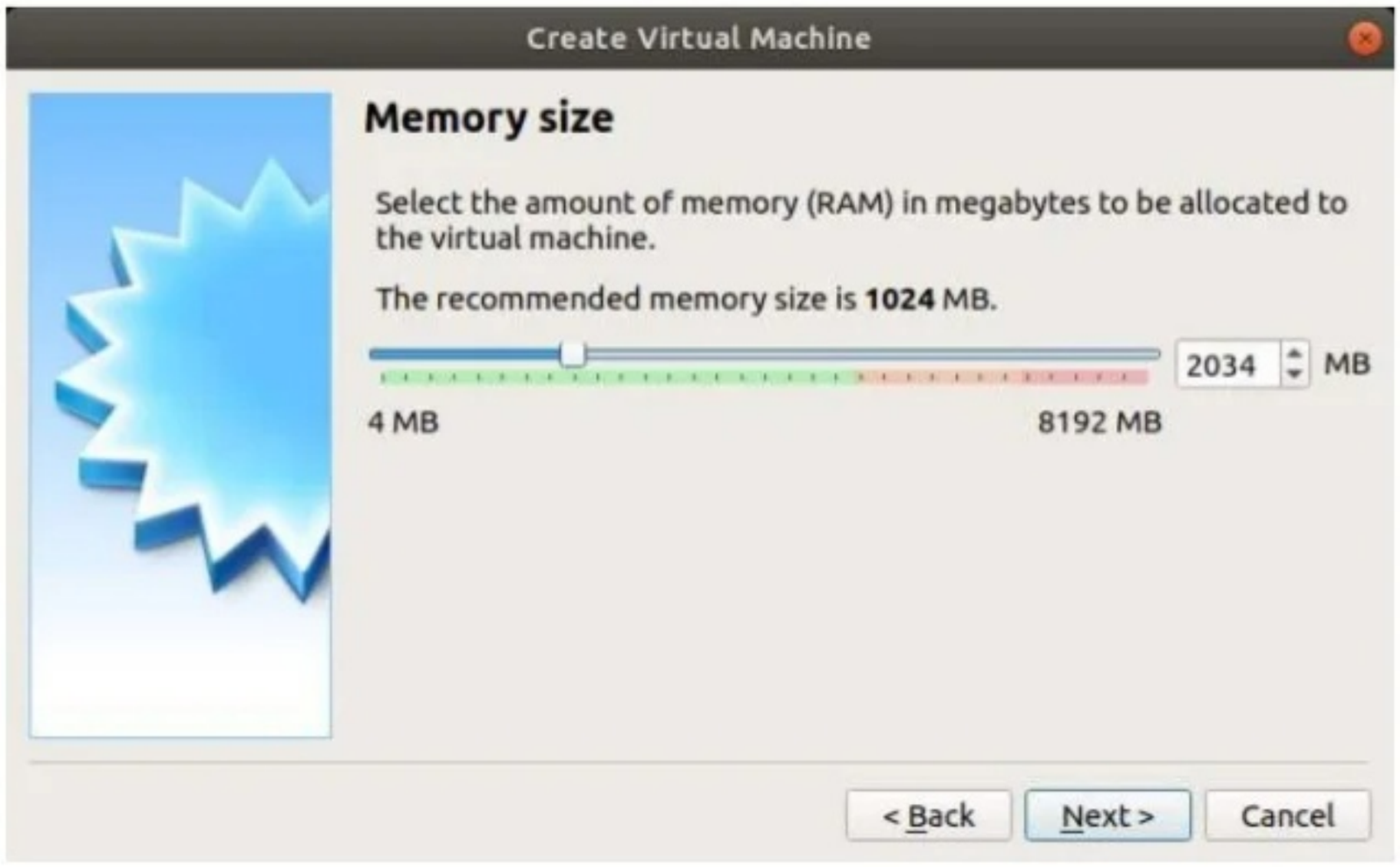



Now your task is to install and configure puppet-master and puppet-agent. Let's proceed with the installation of puppet mater virtual vm on oracle virtualbox.

Step 1: Click on the new option available on the screen.

Step 2: Selecting Ubuntu 64 bit as operating system

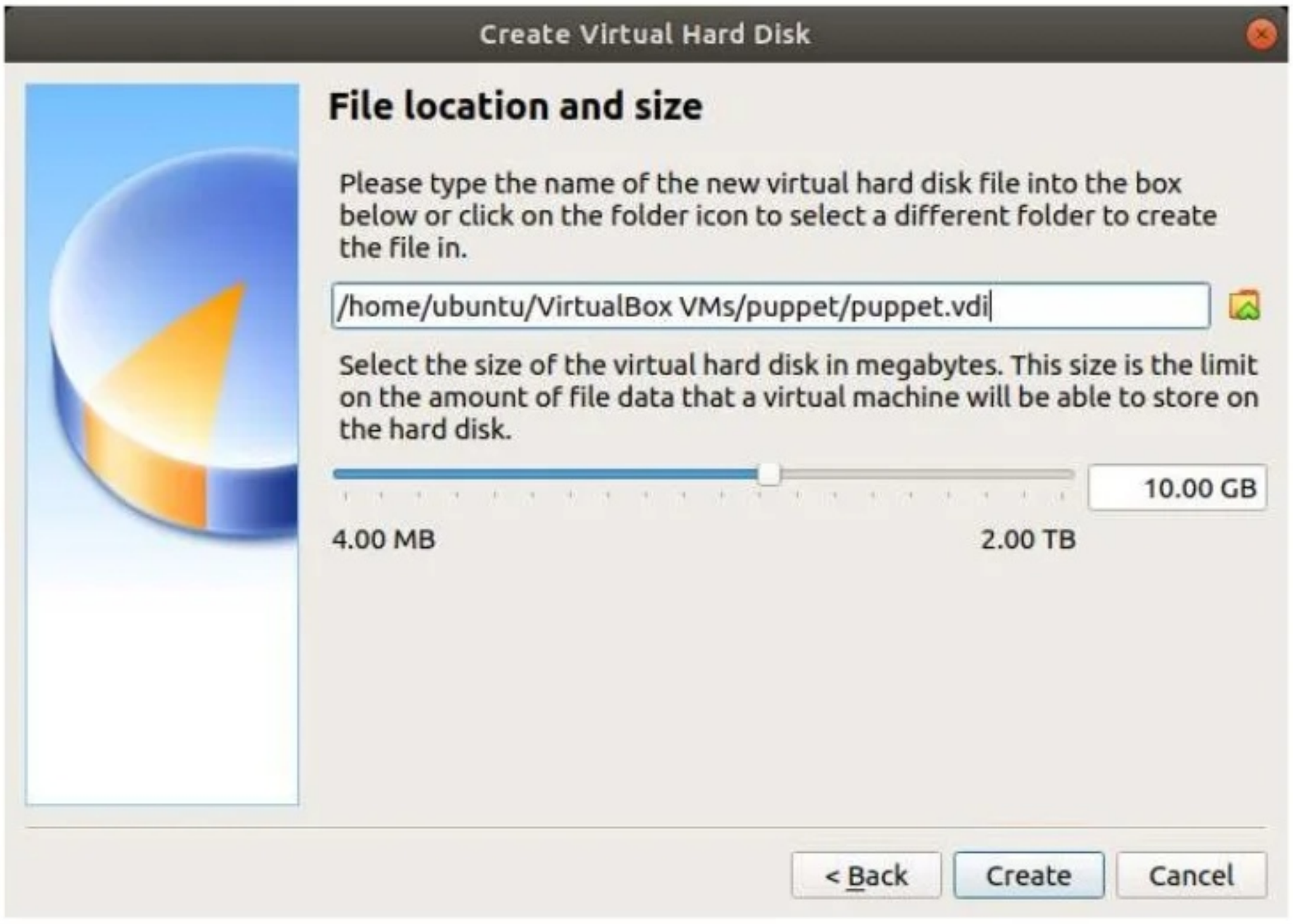
Step 3: Allocating memory of 2GB to the virtual os




Step 4: select option virtual hard disk

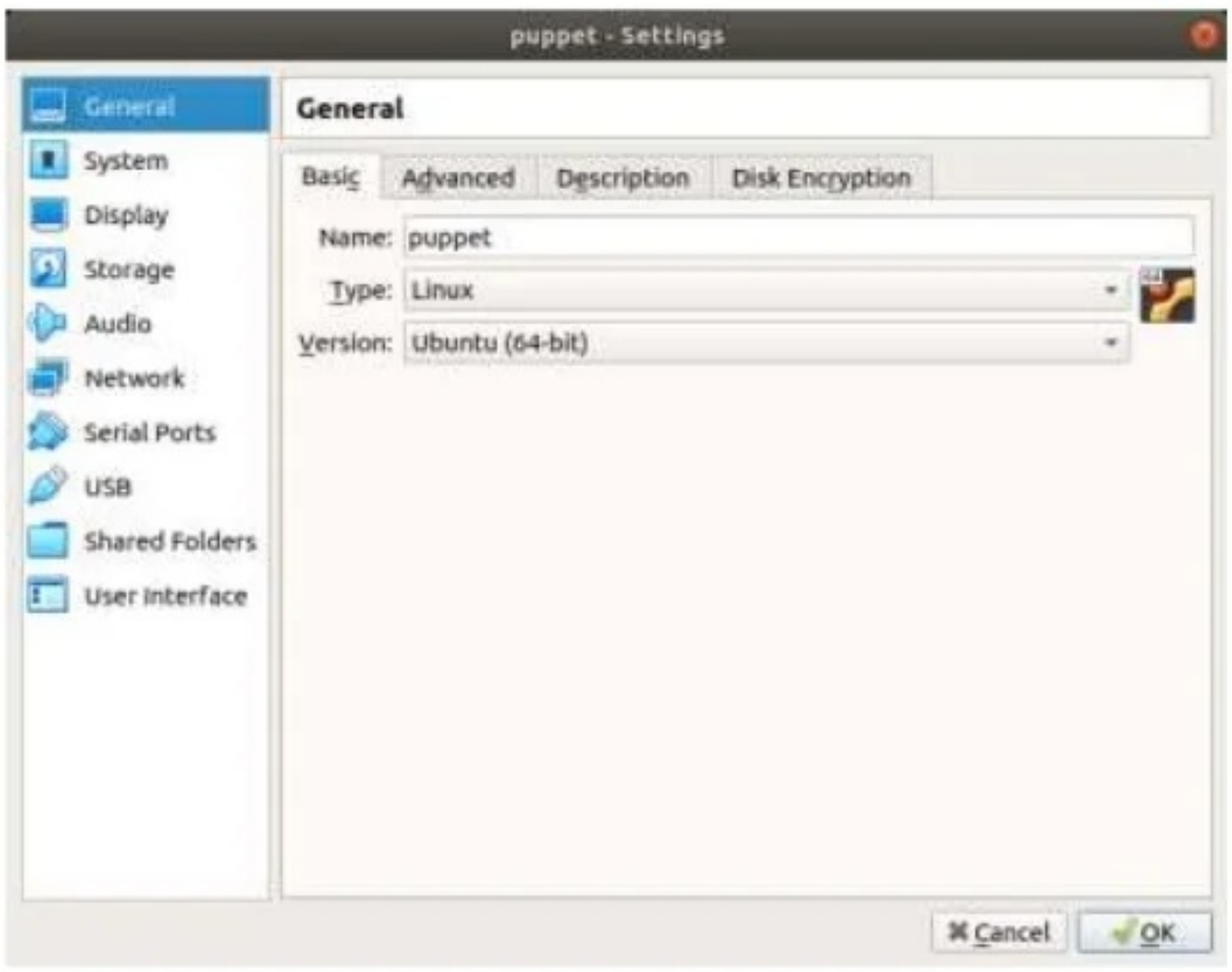

Step 5: Selecting virtual disk image


Step 6: Allocate file and specify the maximum size vm can take.

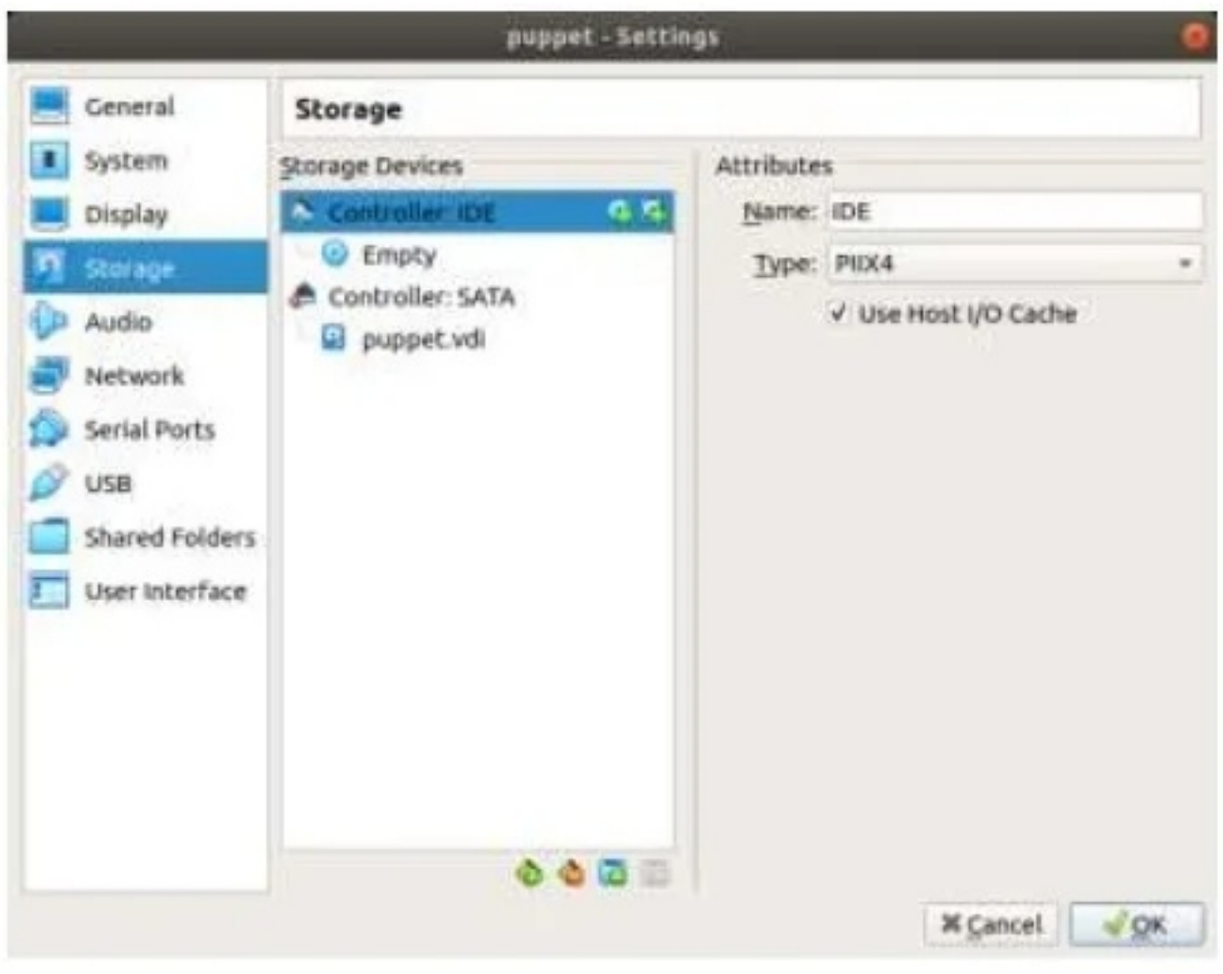



Step 7 : Allocate file and specify the maximum size vm can take.

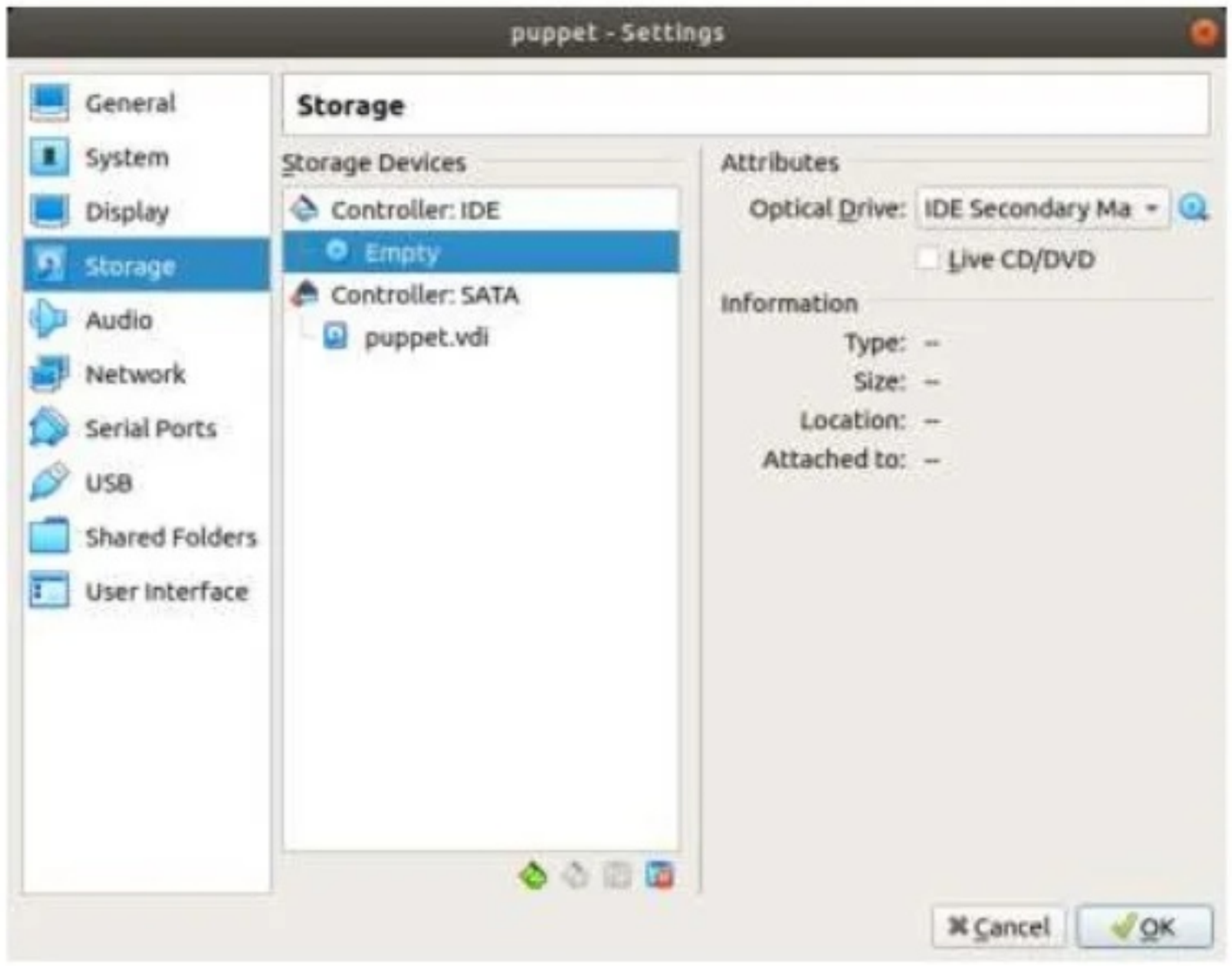

Step 8: Verify the general settings in the general setting tab.



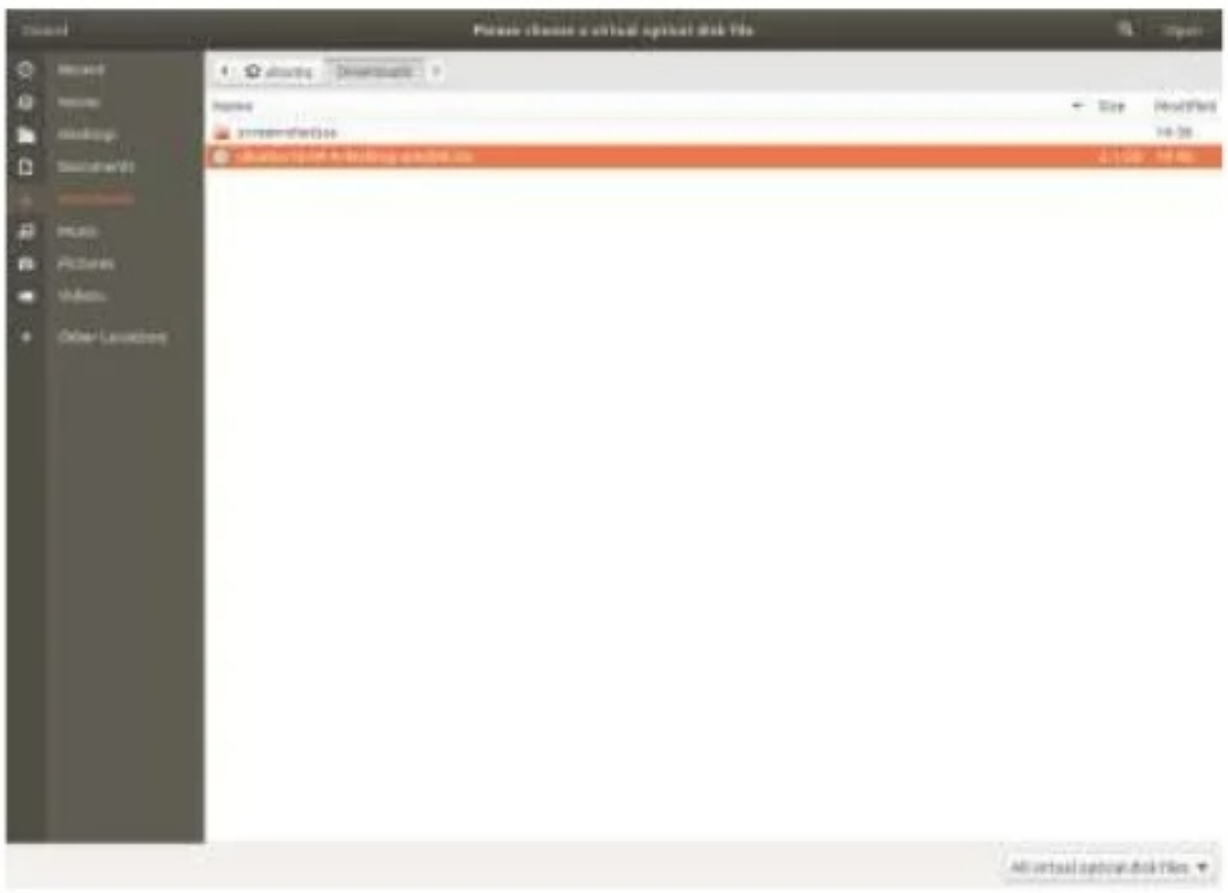

Step 9: Now go to storage option and click on controller

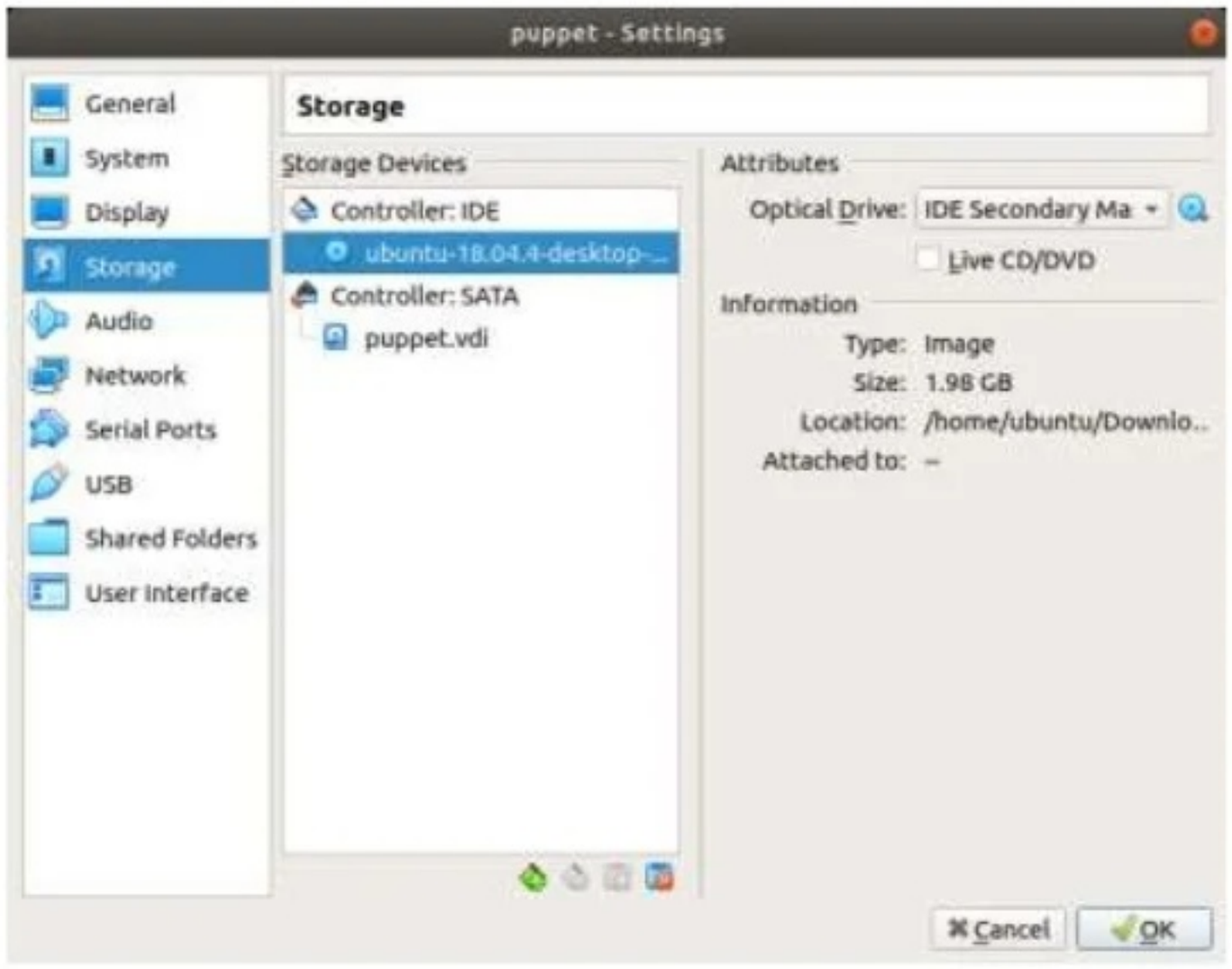


Step 10: Click on empty option and select disk option at the top right

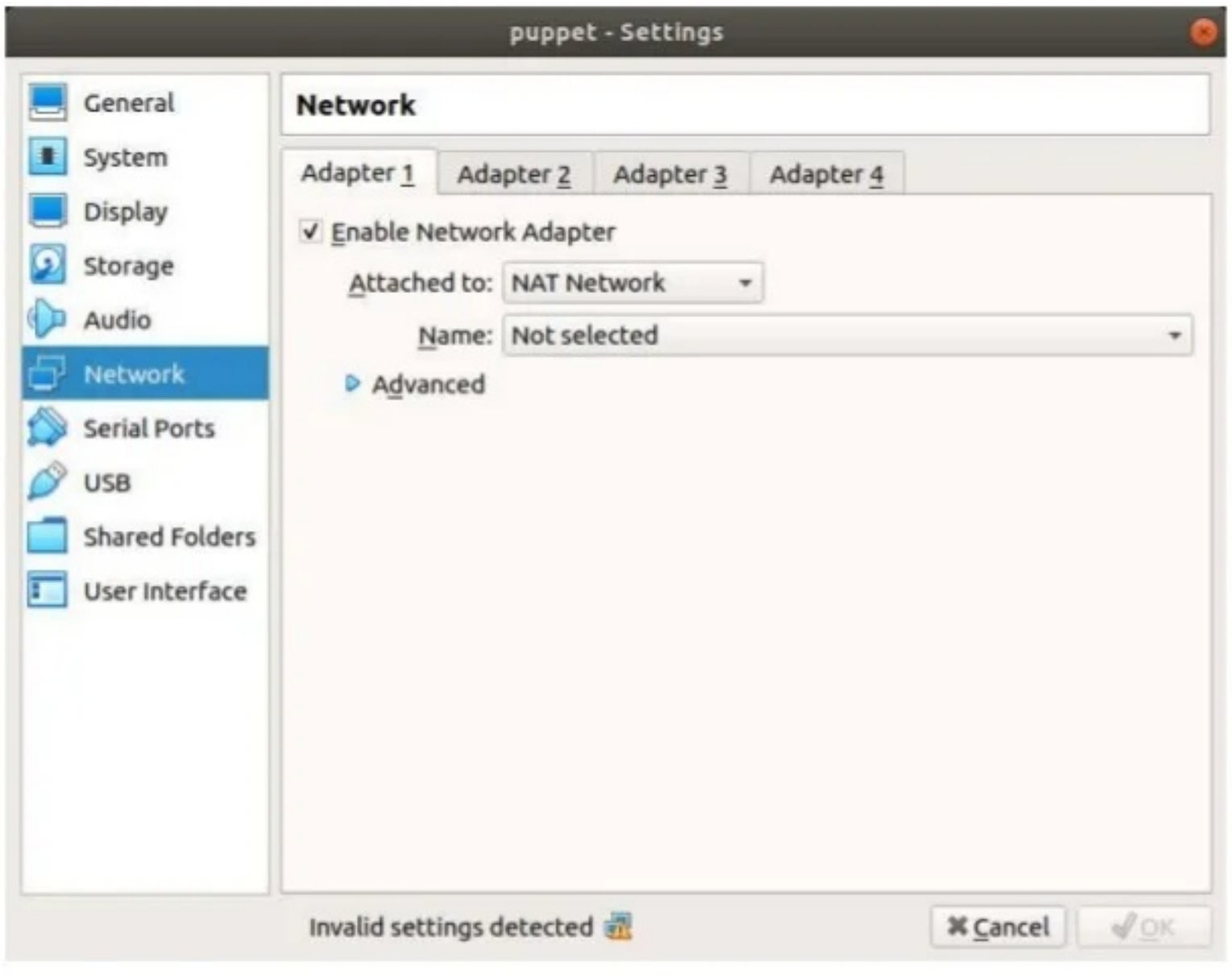



Step 11: Select ubuntu-18.04 iso file

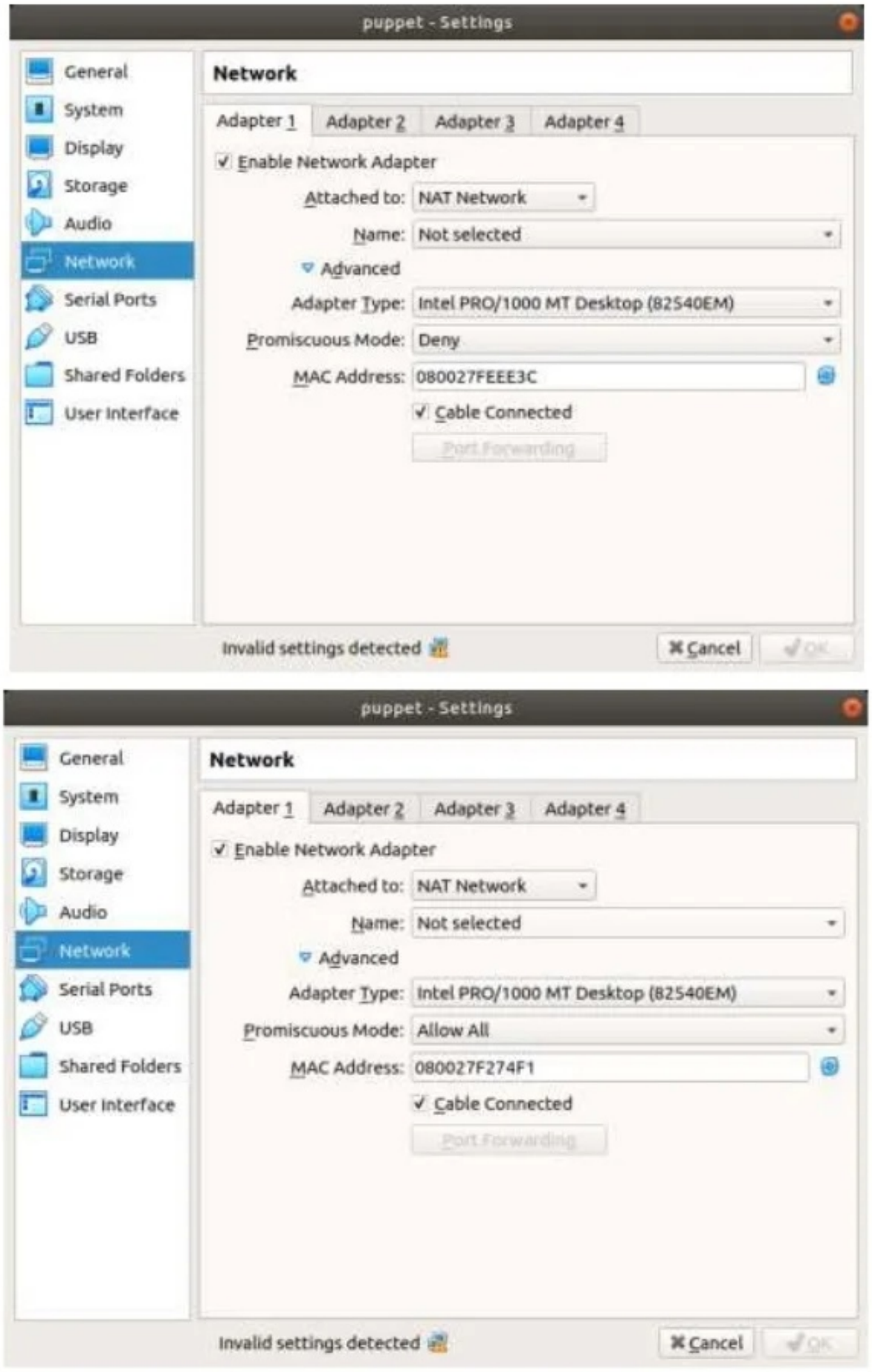




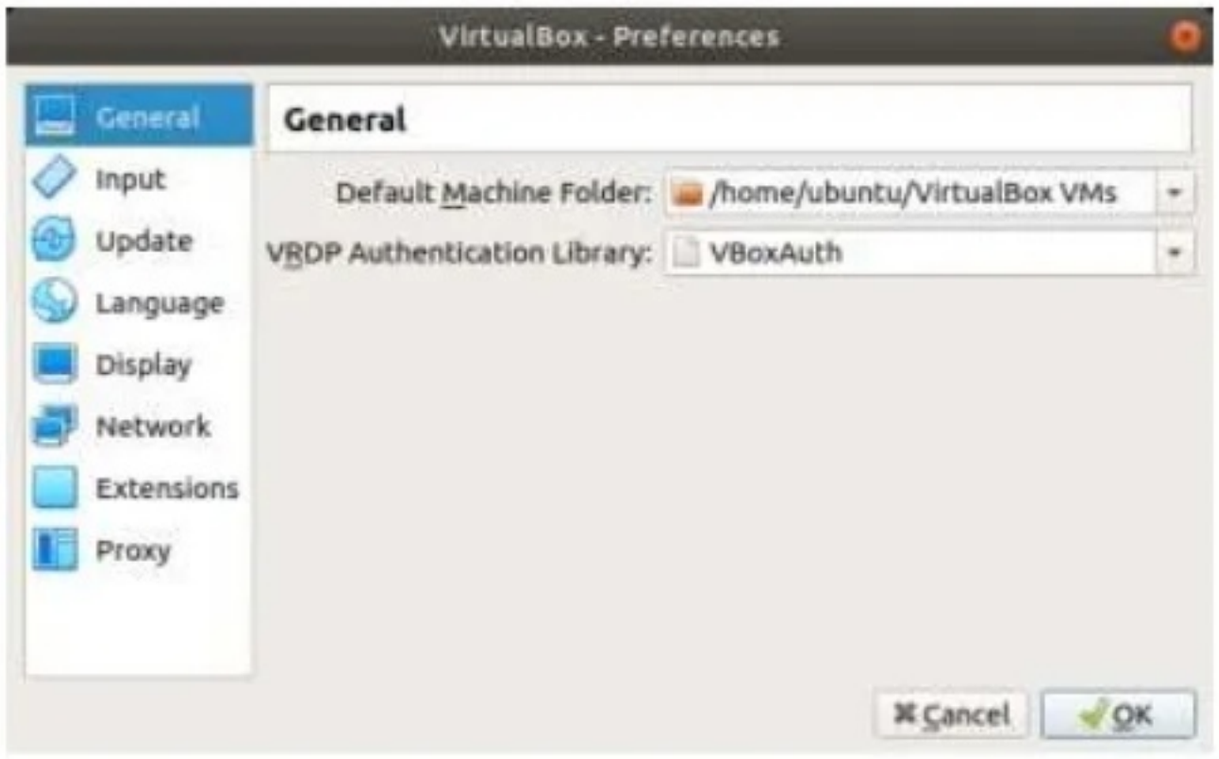

Step 12: Select the network tab and enable network adapter
Change attached to "Nat Network" from "Nat"




Step 13: Change network options as shown in below image




Step 14: In order to create new nat network click on file -> preferences

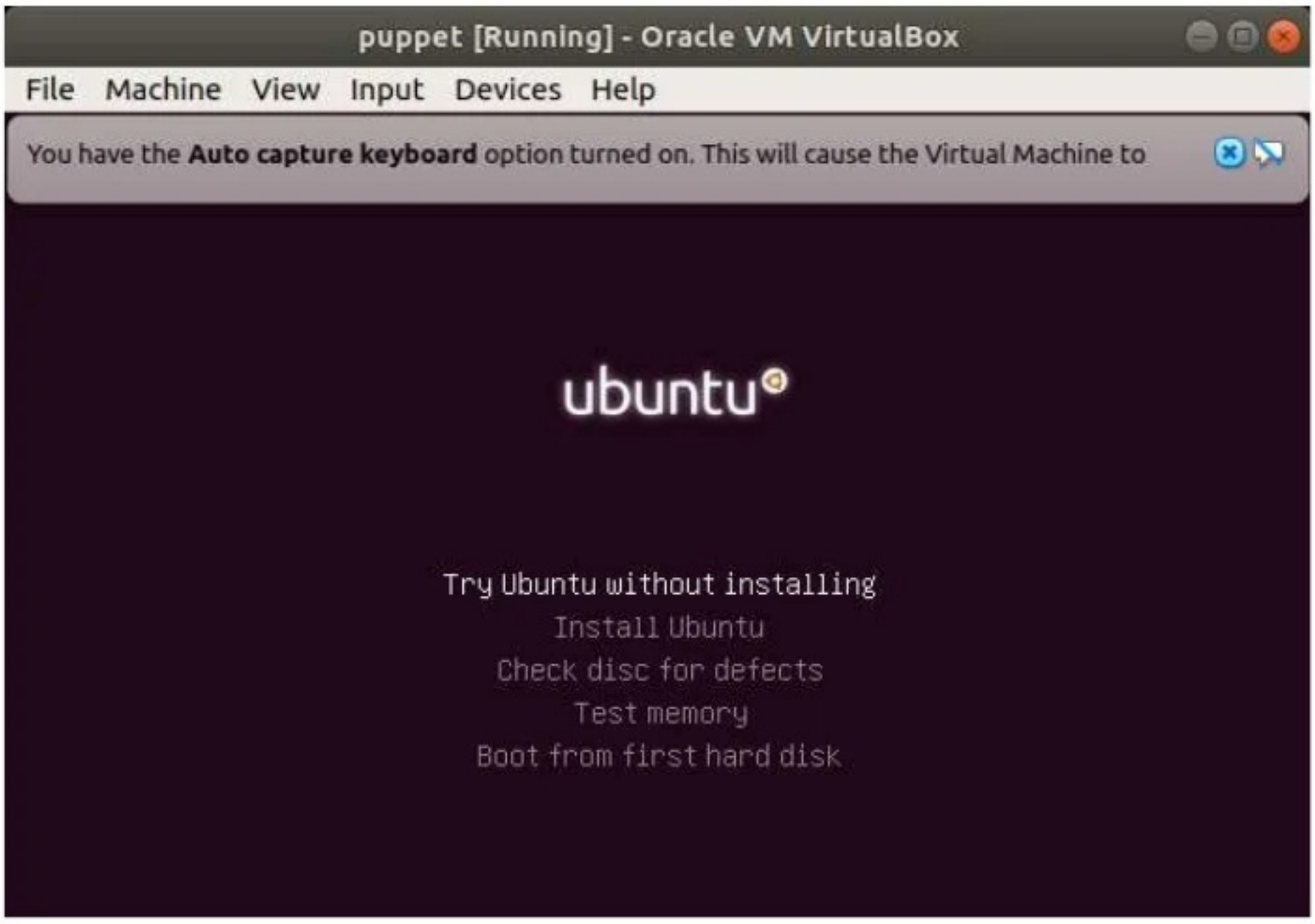


Step 15: Click on network tab and enter new nat network details

Step 16: Now again go to puppet server and set network configuration

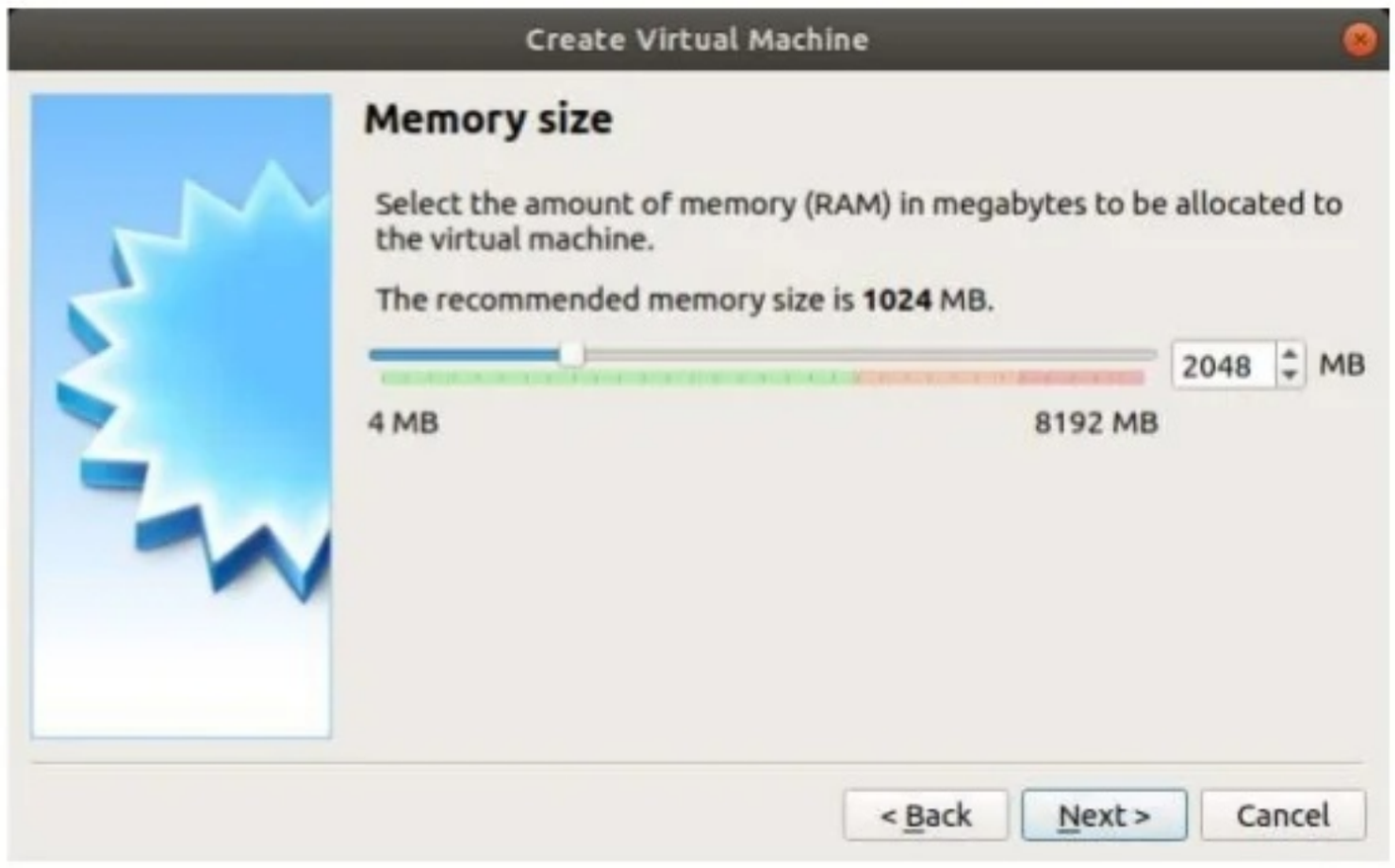

Step 17: Selecting NAT network and choosing created network.

Step 18: Start the created virtual machine i.e. puppet

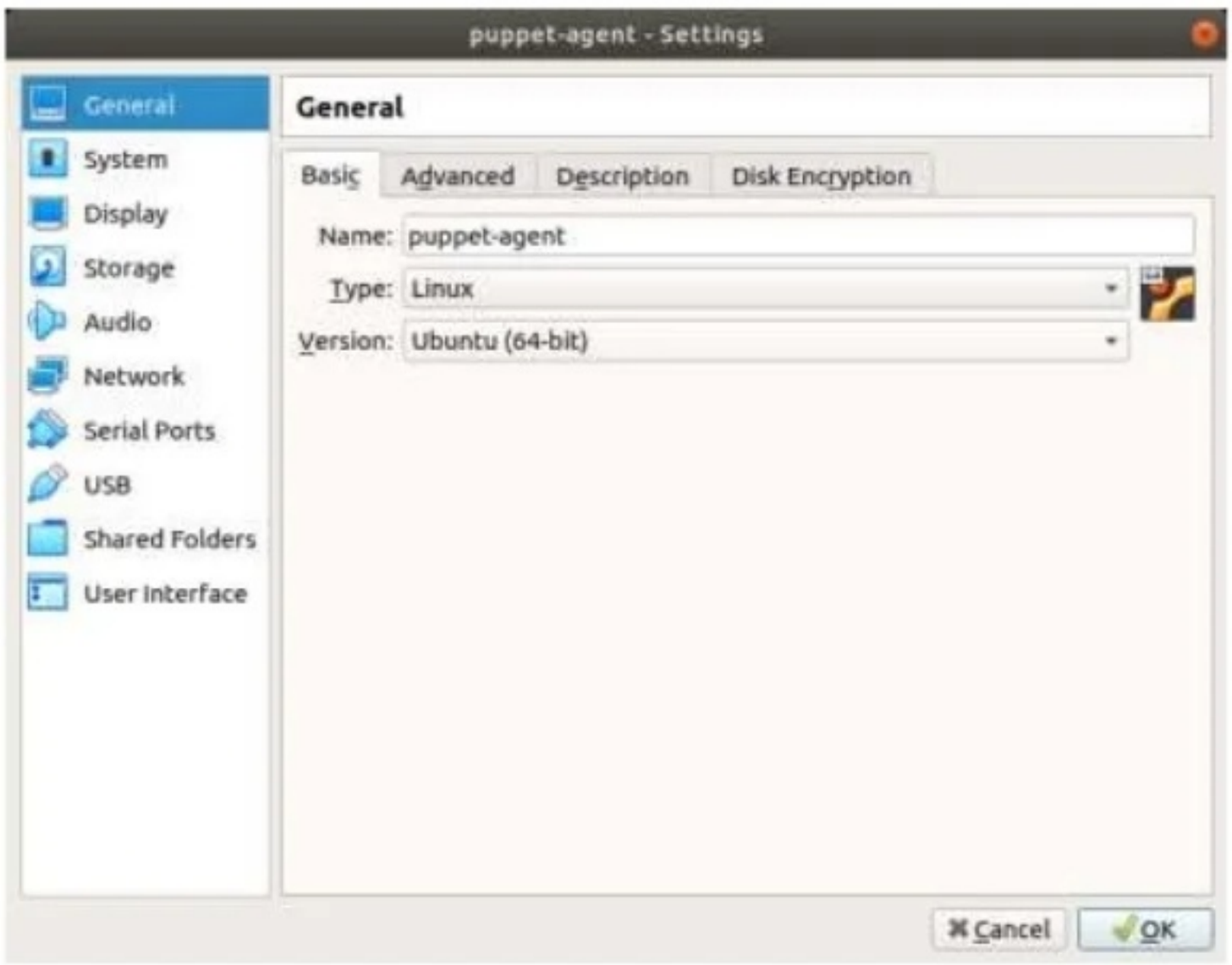



Step 19: Create another virtual machine as puppet-agent


Step 20: Allocate 2gb of memory




Step 21: Now click on settings to configure the puppet-agent



Step 22: Select newly created vm and hit enter key from your keyboard to start puppet-agent.


Step 23:Virtua1 Box will ask you to set up the start up boot select ubuntu iso file.


## *Conclusion*

In this experiment we have learn about puppet which is a configuration management tool. It follows the client-server model, where one machine in any cluster acts as the server, known as

puppet master and the other acts as a client known as a puppet agents. We have successfully installed puppet master and puppet agent and have established connection between them.

***References:***