Application Delivery Fundamentals 2.0 B: Java

Introduction to GIT and SVN

Configuration Management

Parameswari Ettiappan

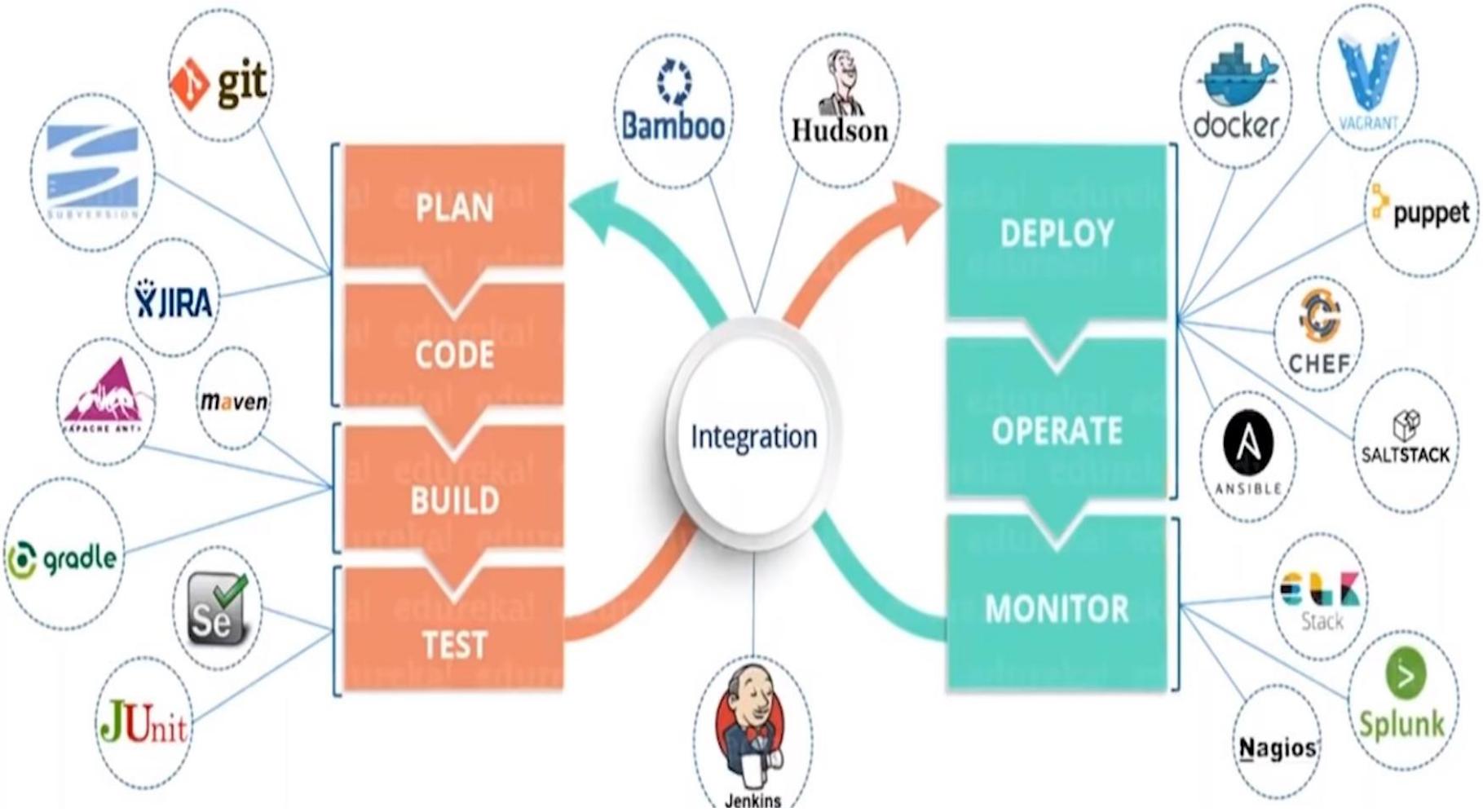
Goals

- Devops
- CI/CD Pipeline
- GIT
- SVN

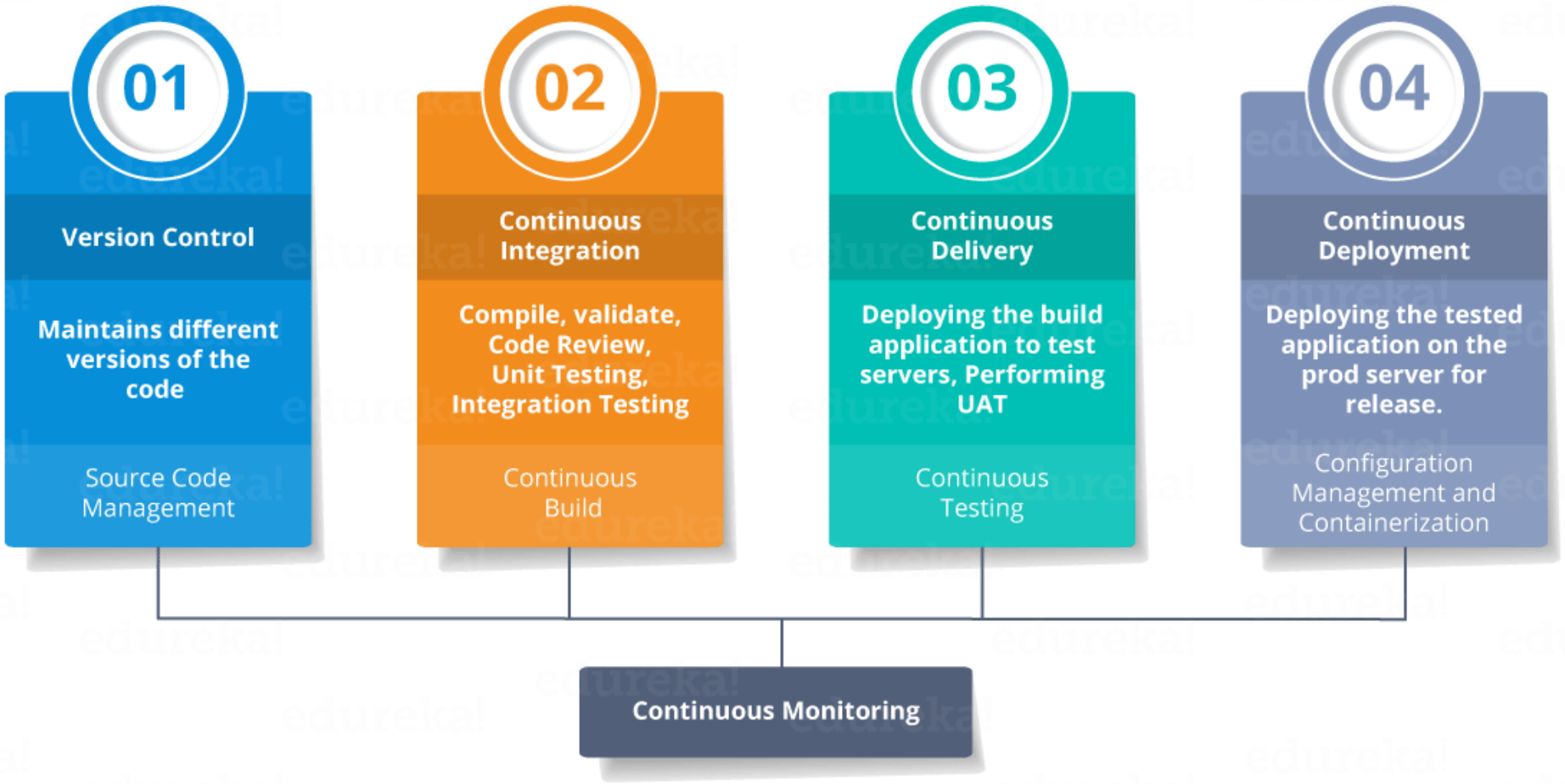
Software Requirements

- Minimum java 8
- github/gitlab
- SVN

What is Devops



CI/CD Pipeline

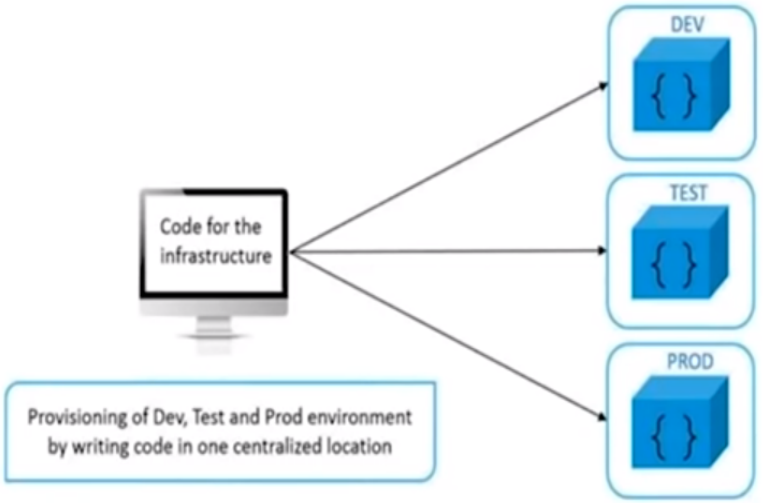


Configuration Management

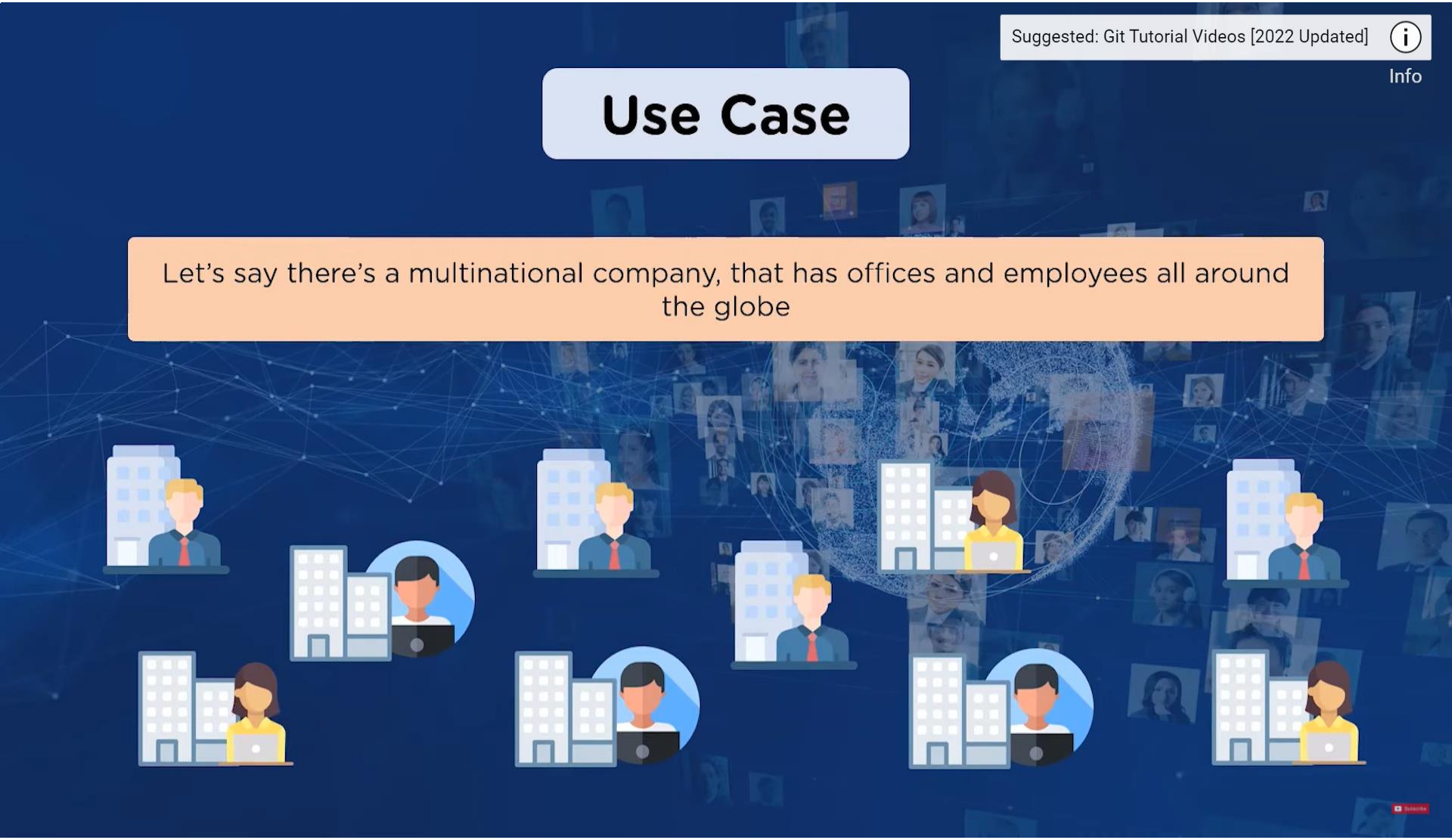
Configuration Management is the practice of handling changes systematically so that a system maintains its integrity over time

Configuration Management (CM) ensures that the current design and build state of the system is known, good & trusted

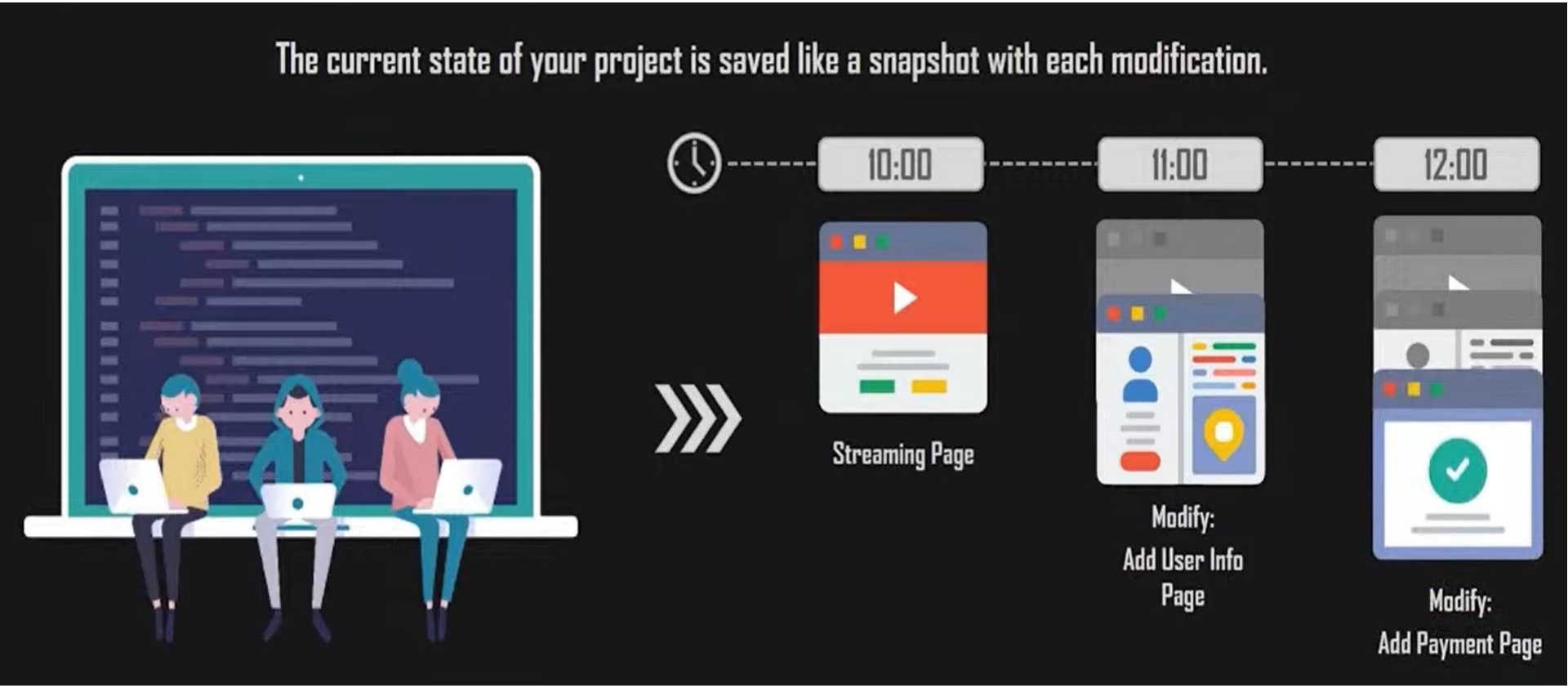
It doesn't rely on the tacit knowledge of the development team



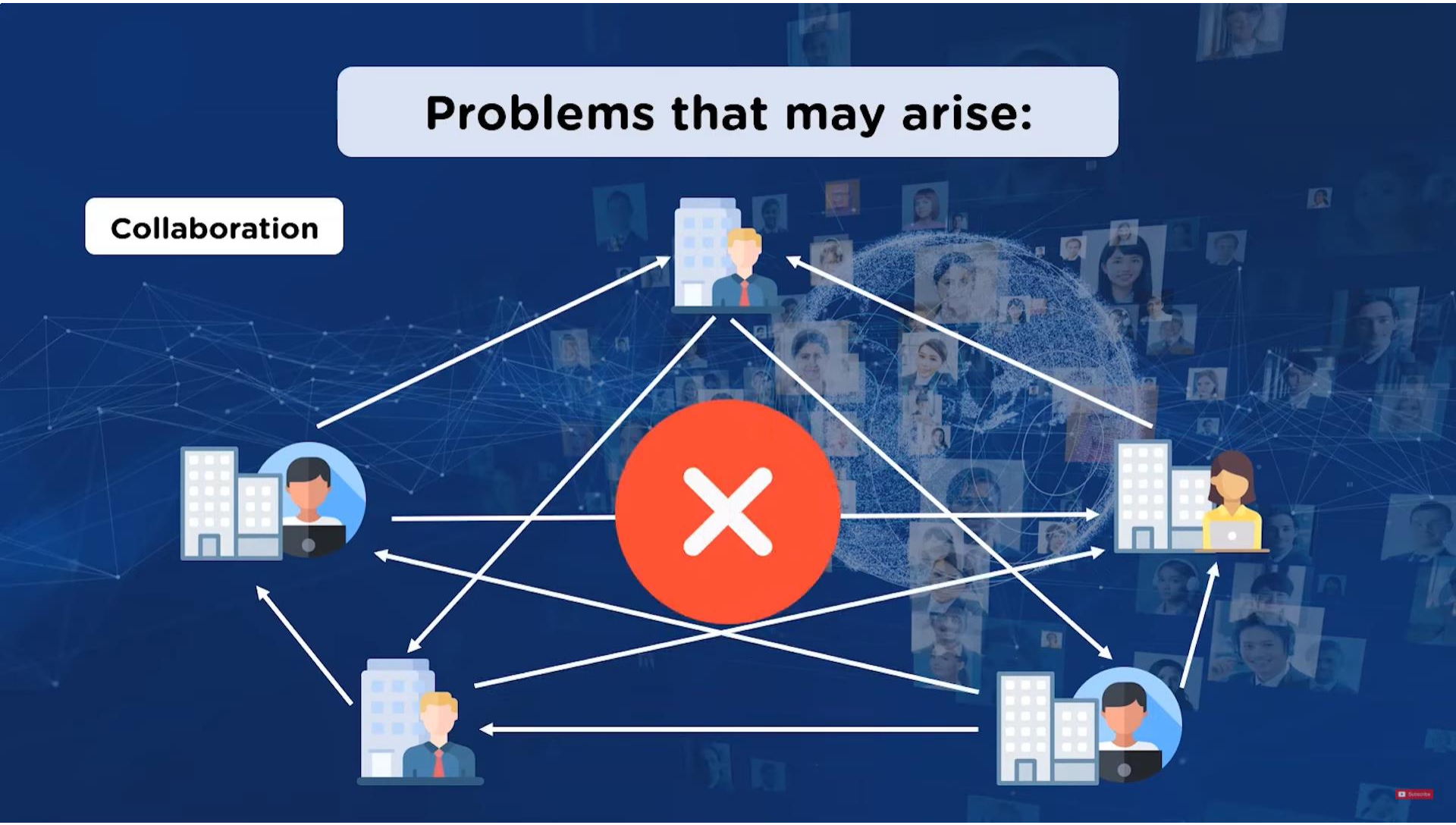
Why Version Control System?



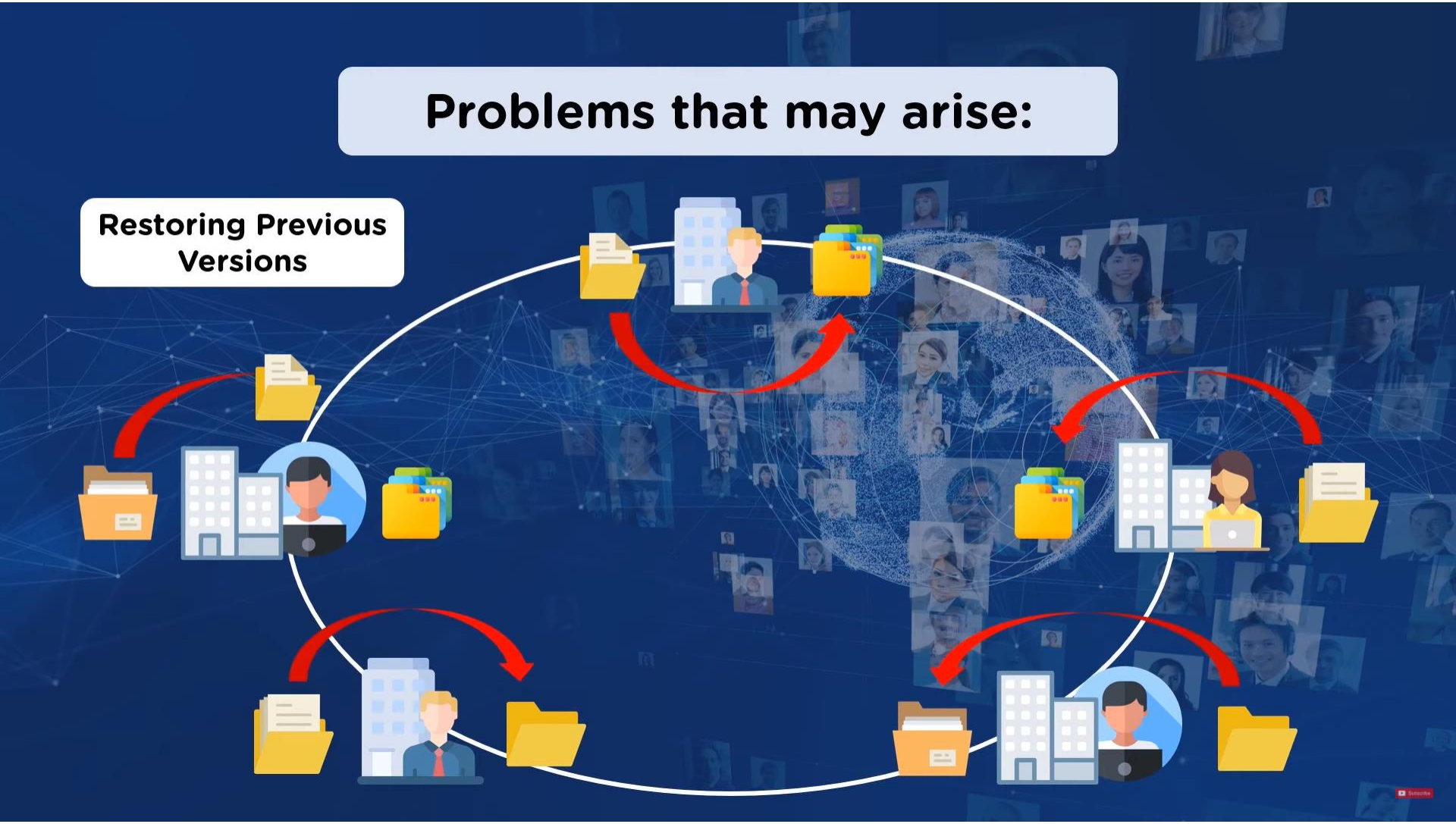
Why Version Control System?



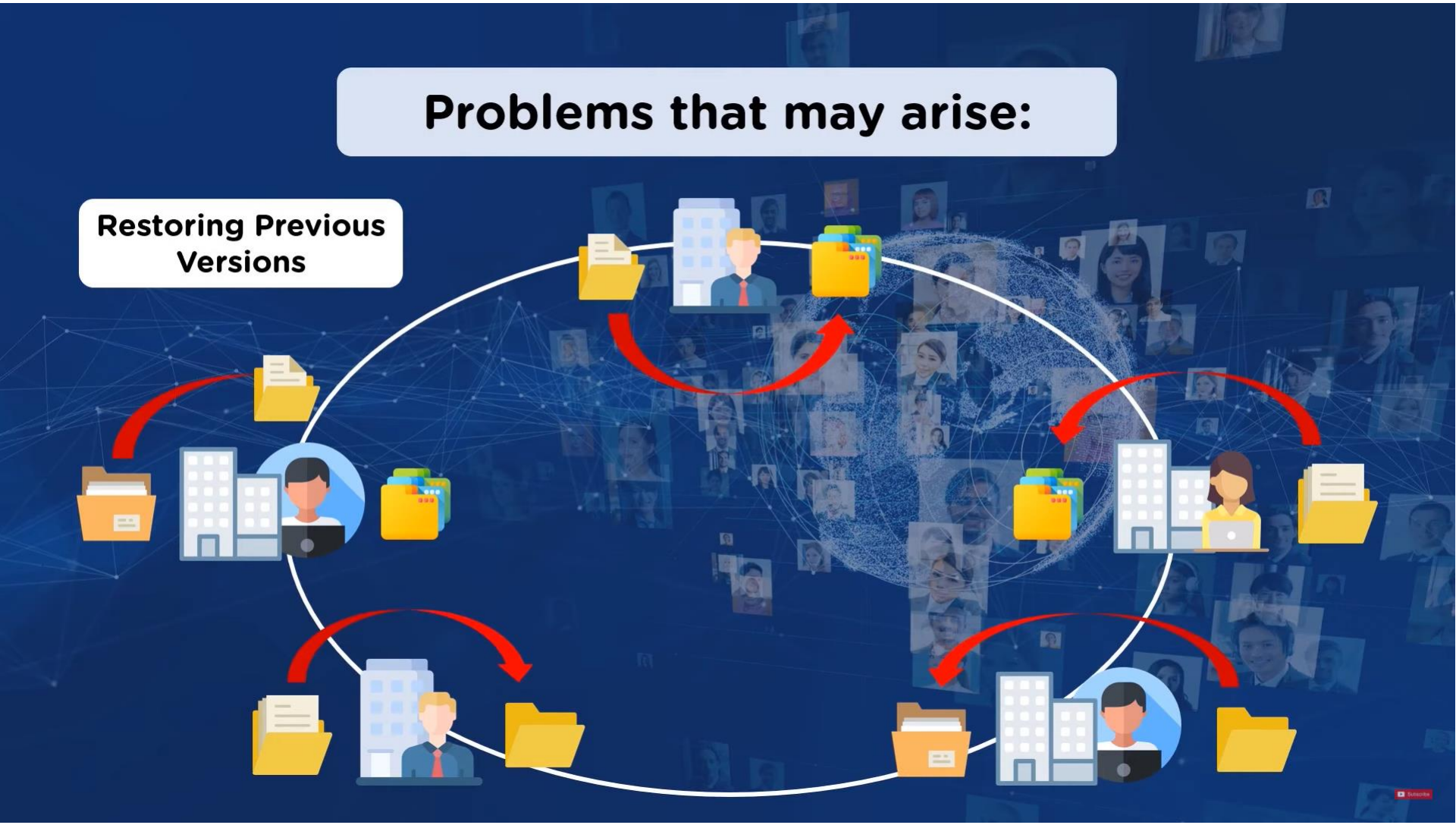
Why Version Control System?



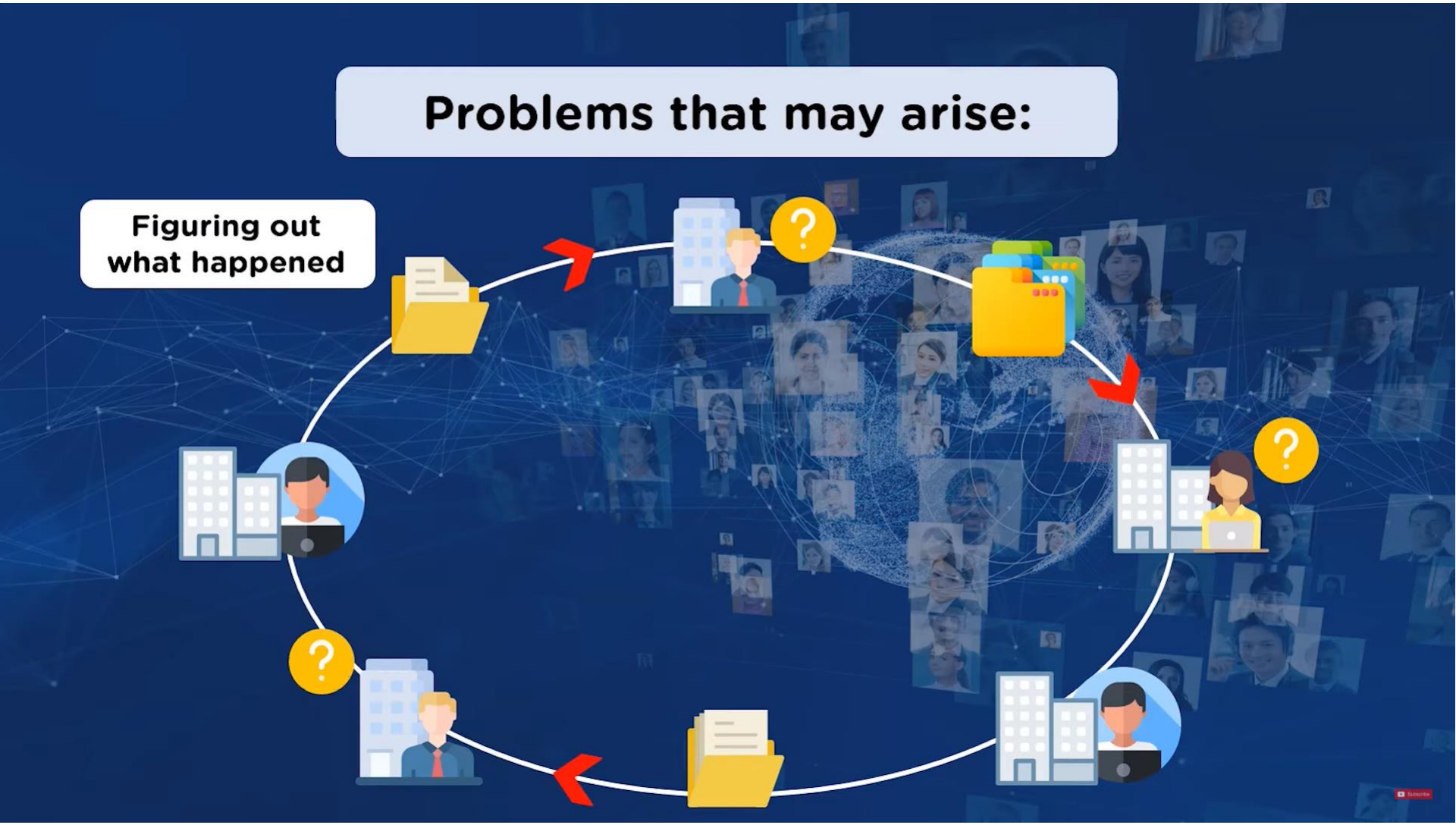
Why Version Control System?



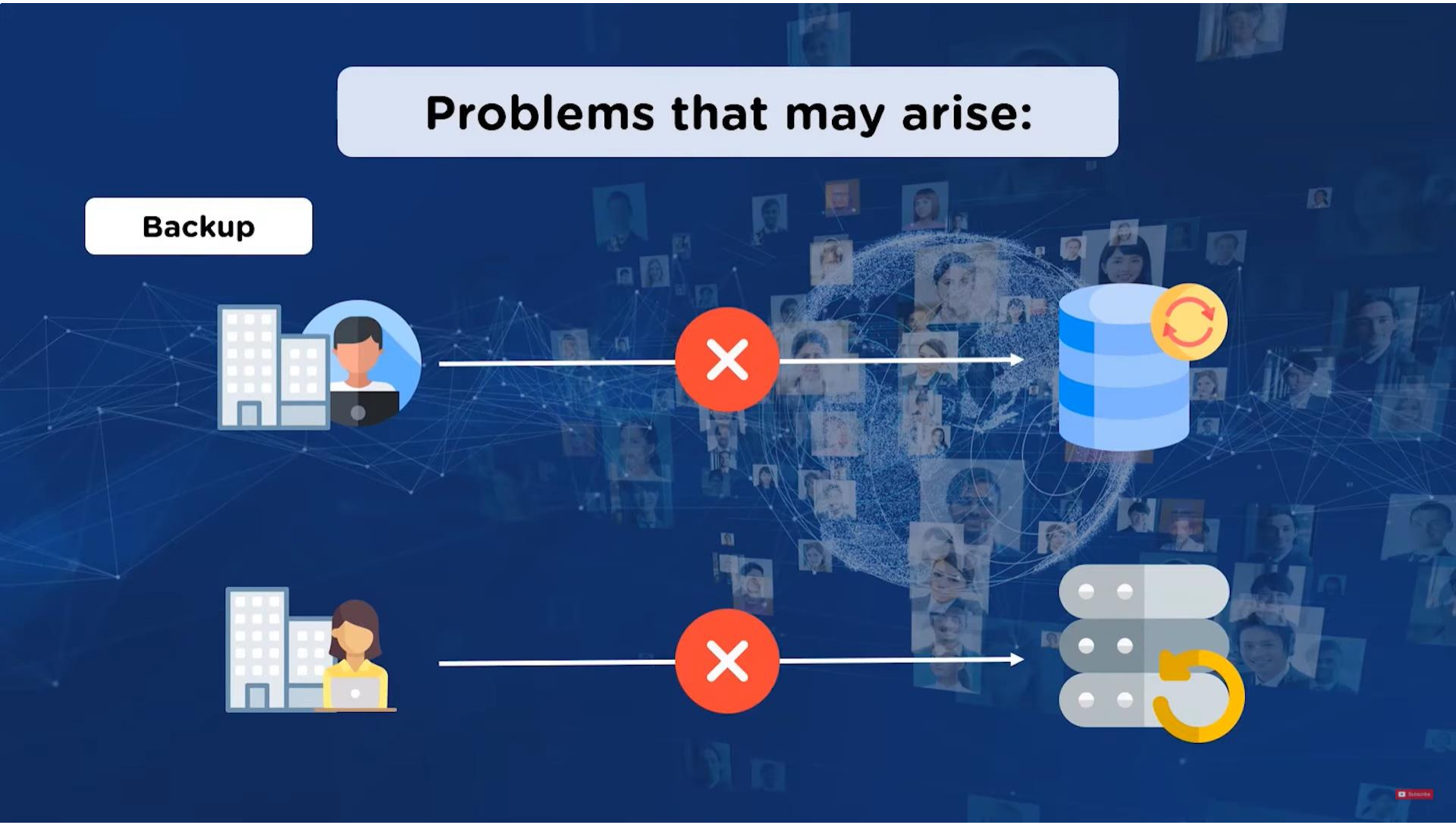
Why Version Control System?



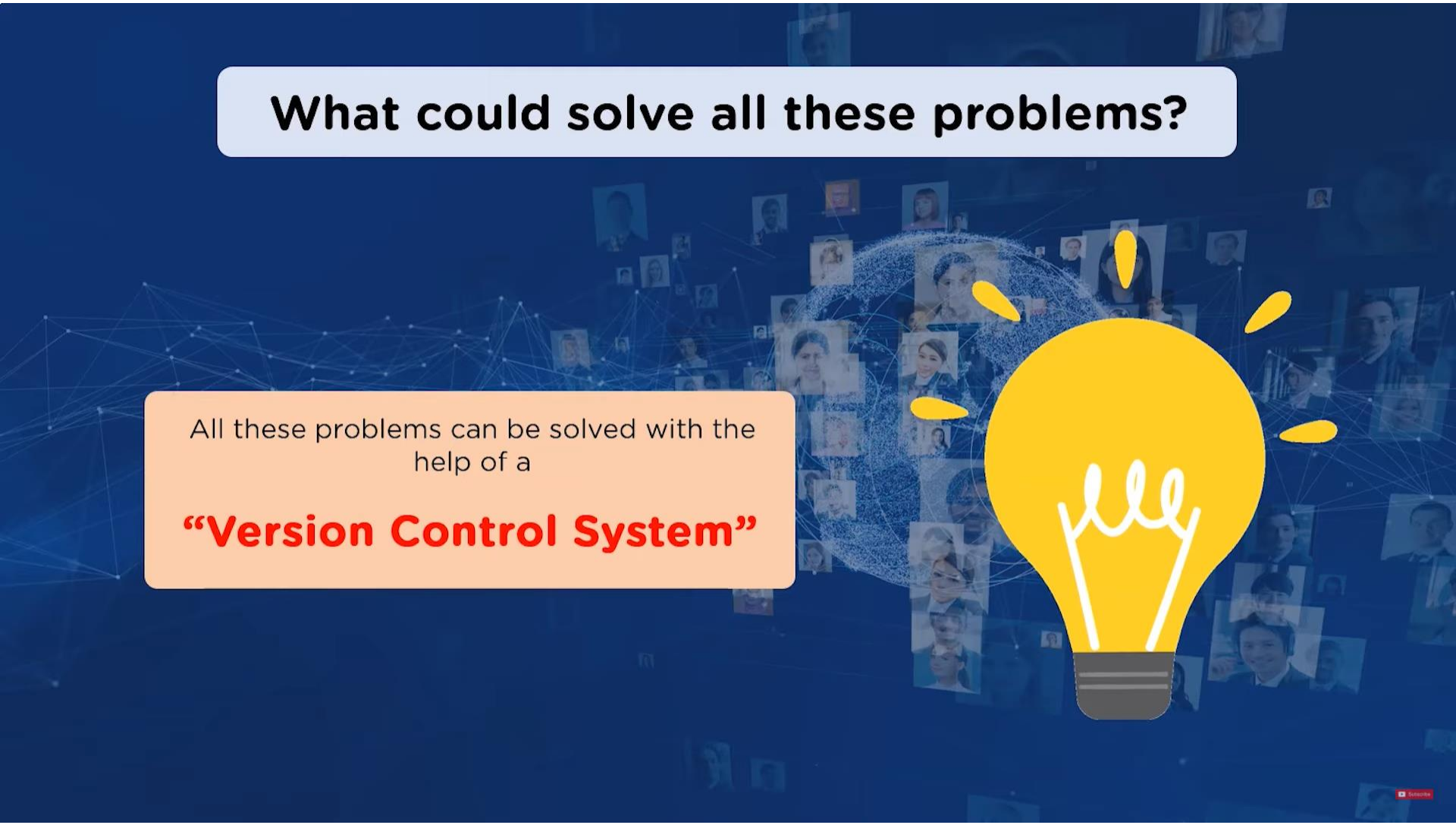
Why Version Control System?



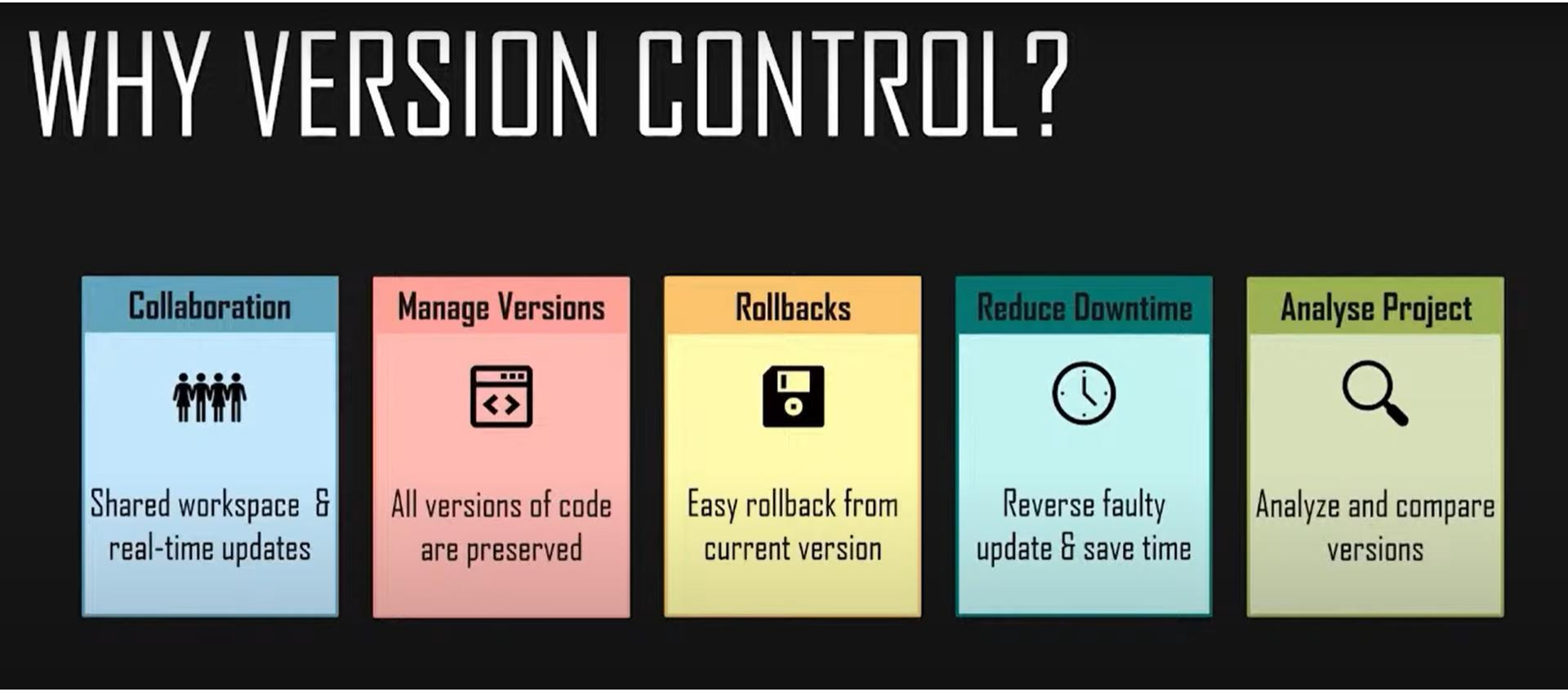
Why Version Control System?



Why Version Control System?



Why Version Control System?



What is Version Control System?

- A version control system allows users to keep track of the changes in software development projects and enable them to collaborate on those projects.
- Using it, the developers can work together on code and separate their tasks through branches.
- There can be several branches in a version control system, according to the number of collaborators.
- The branches maintain individuality as the code changes remain in a specified branch(s).

What is Version Control System?

- Developers can combine the code changes when required.
- Further, they can view the history of changes, go back to the previous version(s) and use/manage code in the desired fashion.

What is Repository

What is a Repository?

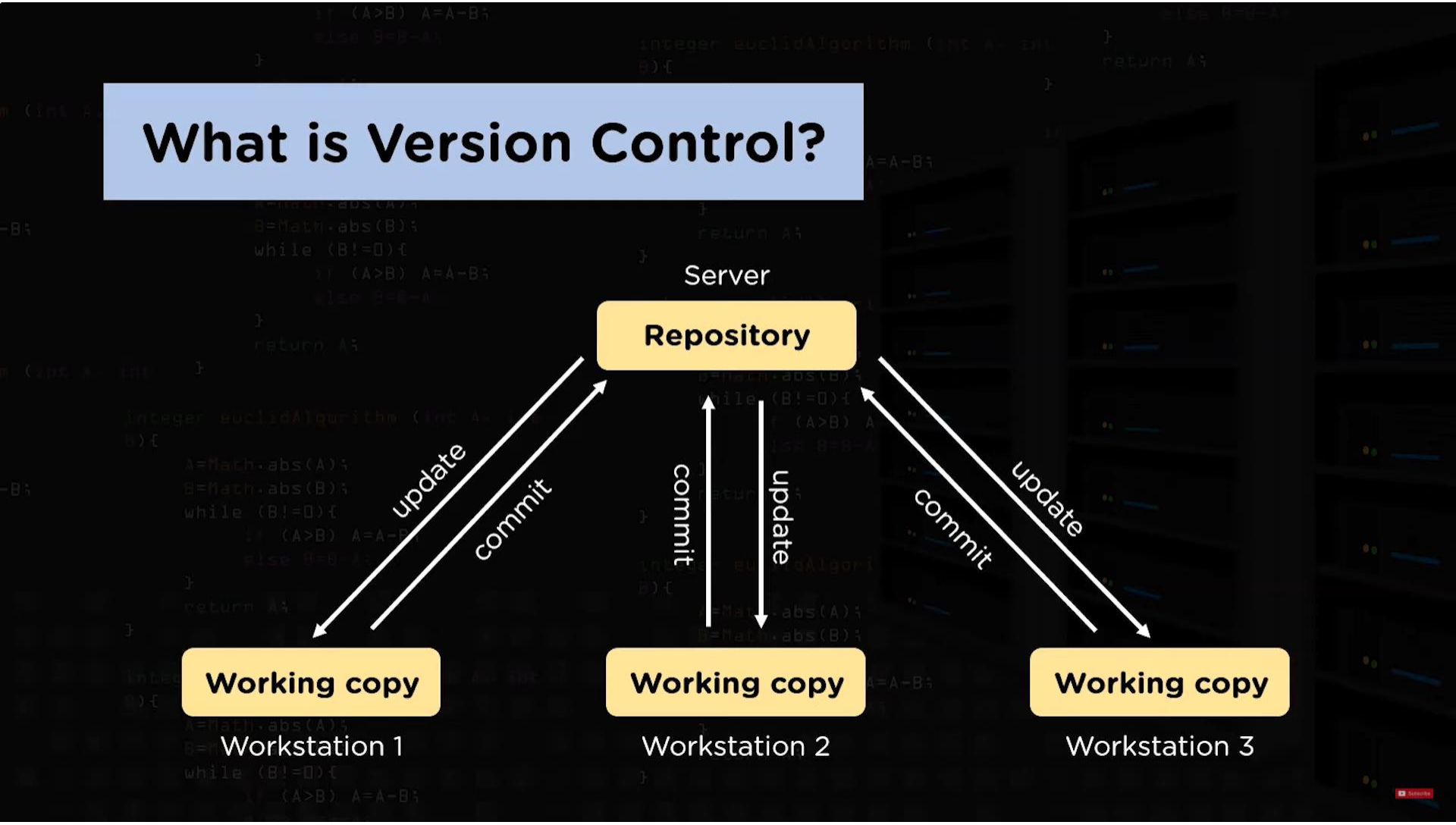
A **repository**(or a repo) is a **directory** or storage space where your projects can live. It can be local to a folder on your computer, or it can be a storage space on another online host (such as Github). You can keep code files, text files, image files, you name it, inside a repository.

What is Version Control System?

What is Version Control?

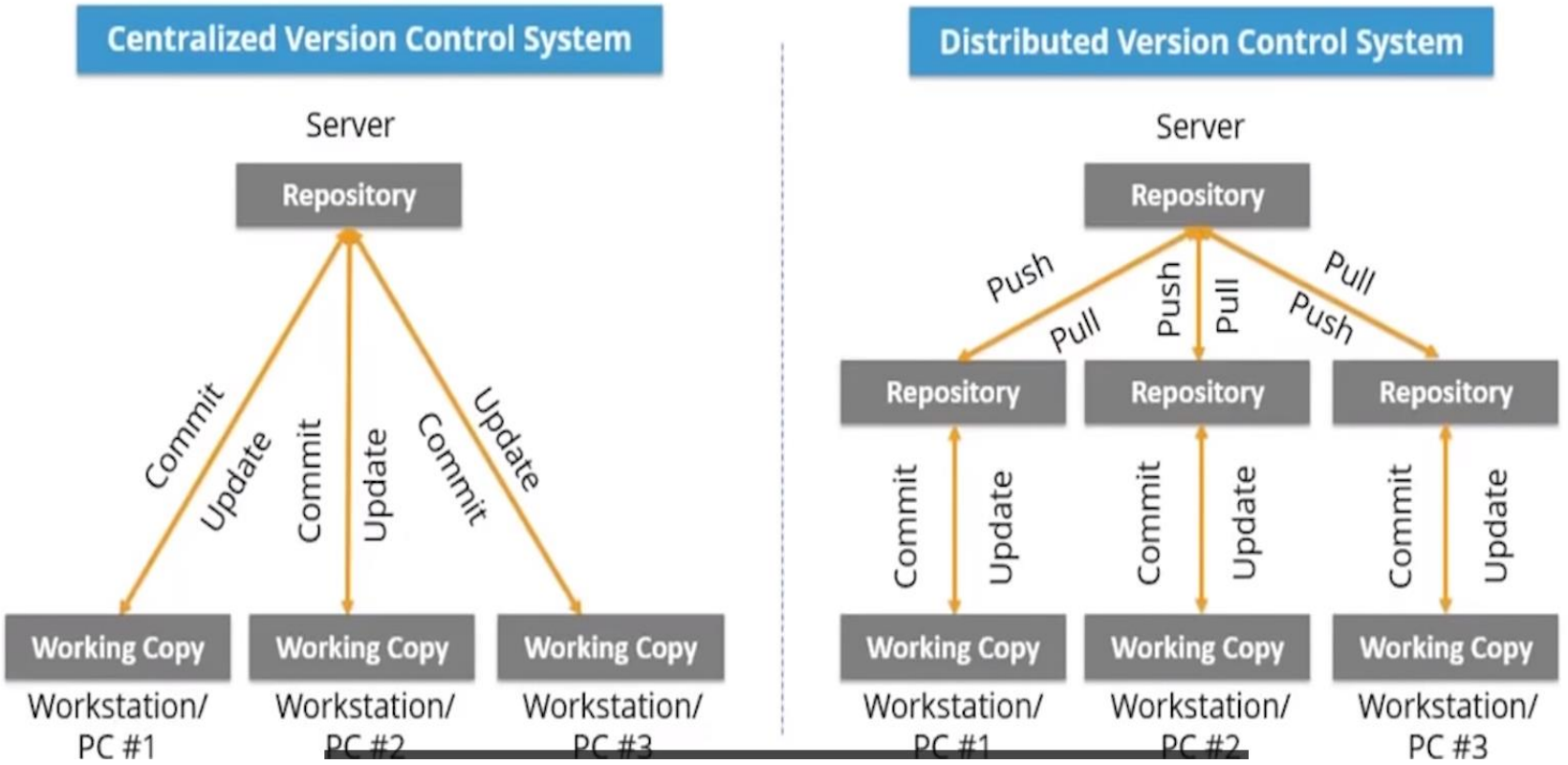
- A Version Control System records all the changes made to a file or set of files, so a specific version may be called later if needed
- The system makes sure that all the team members are working on the latest version of the file

What is Version Control System?

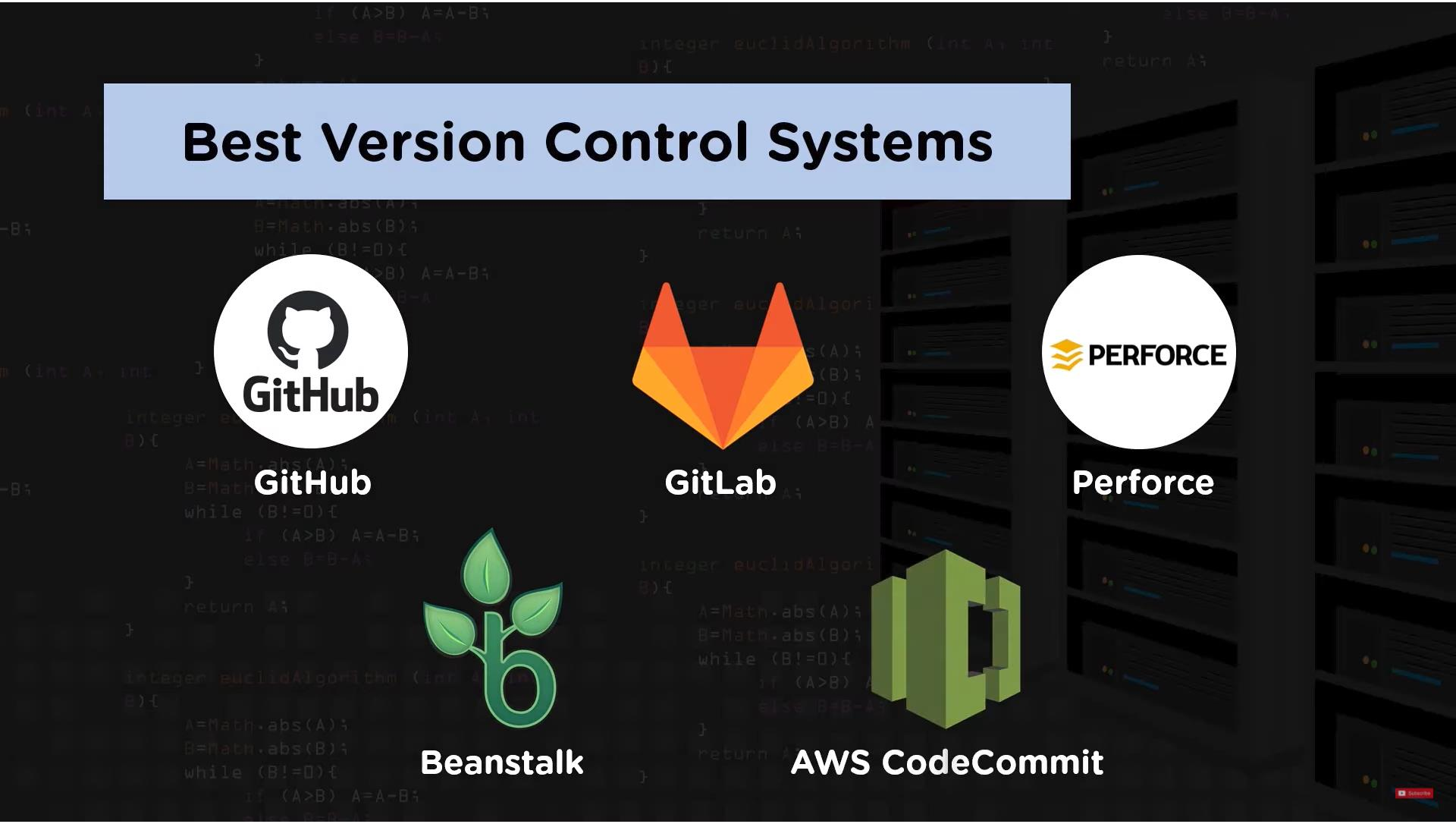


Source Code Management

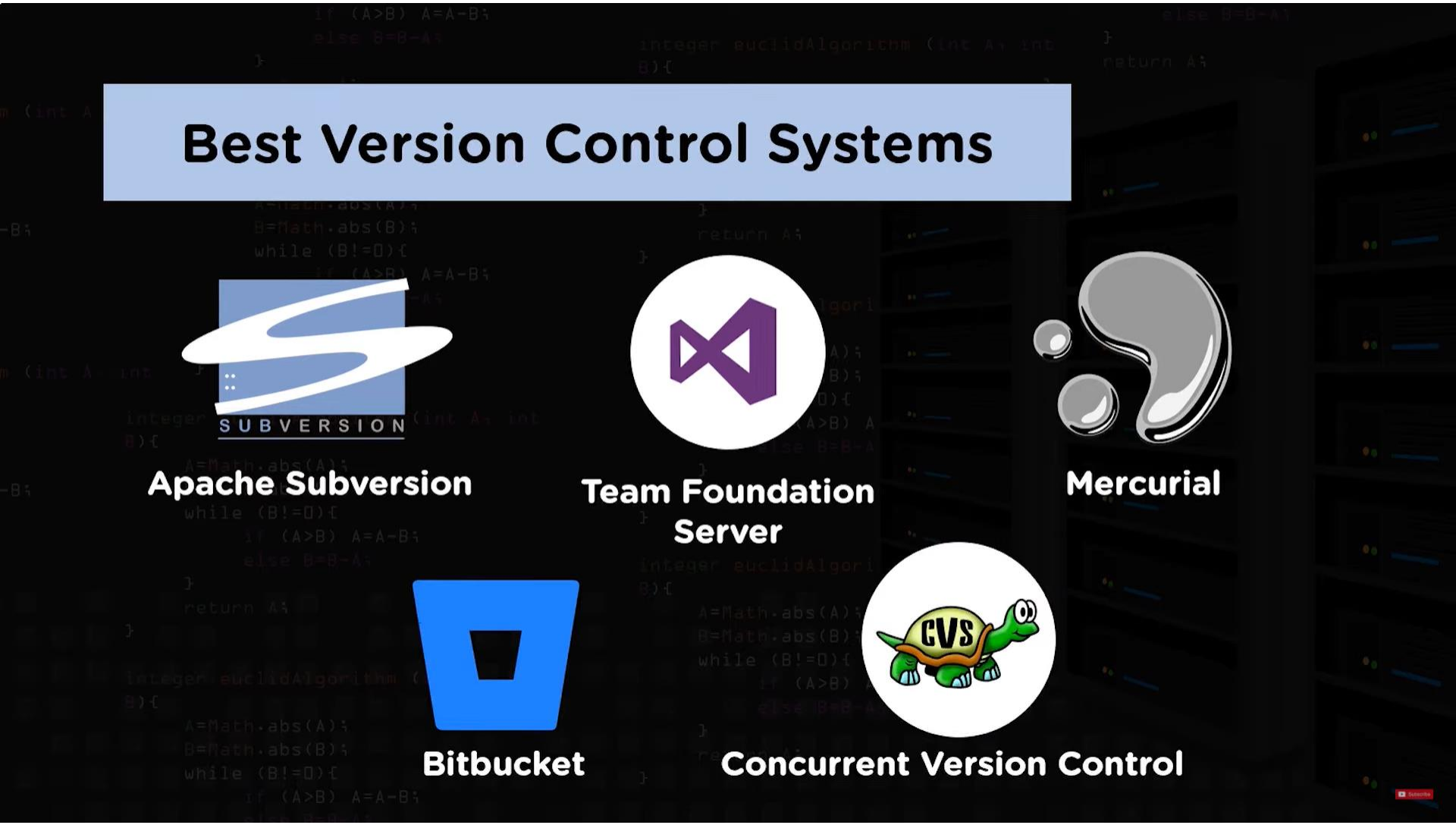
The management of changes to documents, computer programs, large websites and other collection of information



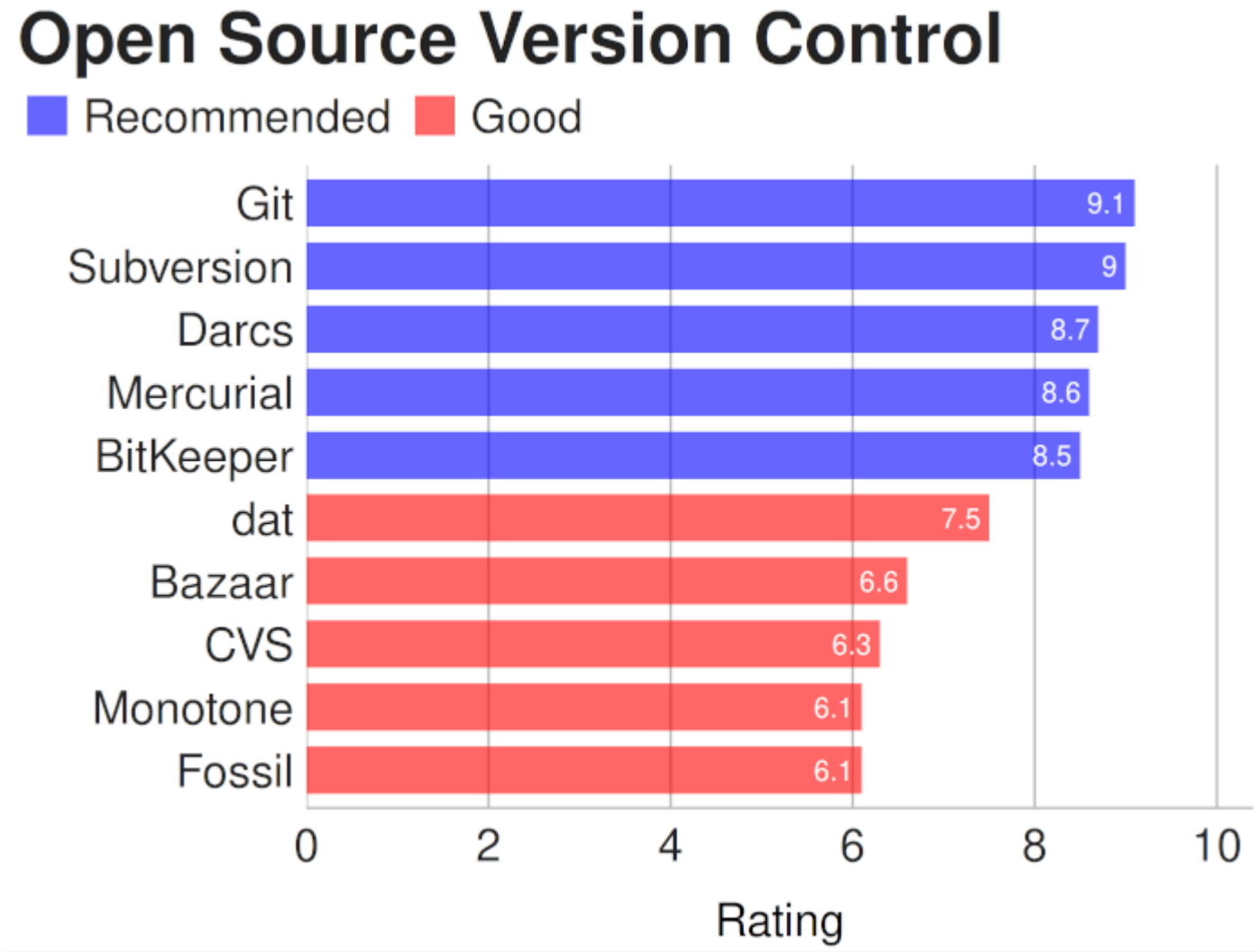
Different Version Control Systems in the market



Different Version Control Systems in the market



Different Version Control Systems in the market

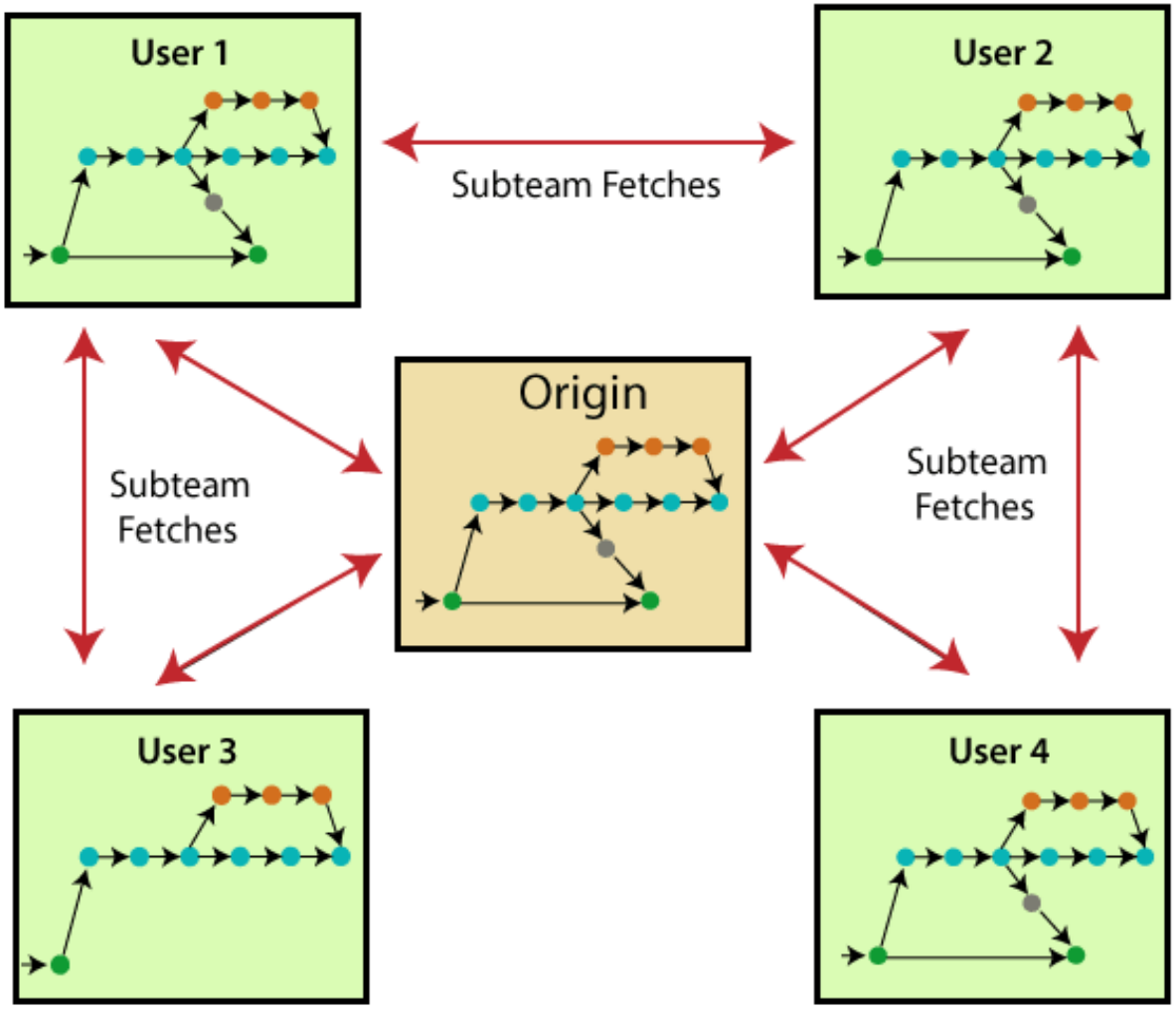


What is Git?

- Git is an open-source distributed version control system.
- It is designed to handle minor to major projects with high speed and efficiency.
- It is developed to co-ordinate the work among the developers.
- The version control allows us to track and work together with our team members at the same workspace.
- Git is foundation of many services like GitHub and GitLab, but we can use Git without using any other Git services. Git can be used privately and publicly.
- Git was created by Linus Torvalds in 2005 to develop Linux Kernel. It is also used as an important distributed version-control tool for the DevOps.
- Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase.

Features of Git

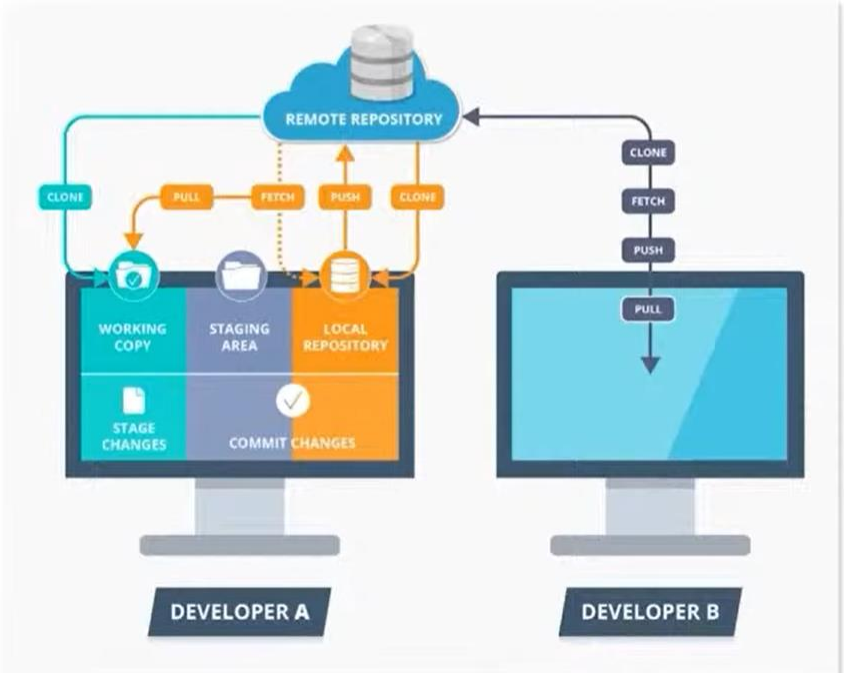
Features of Git



GIT Workflow

BUT HOW DOES IT WORK ?

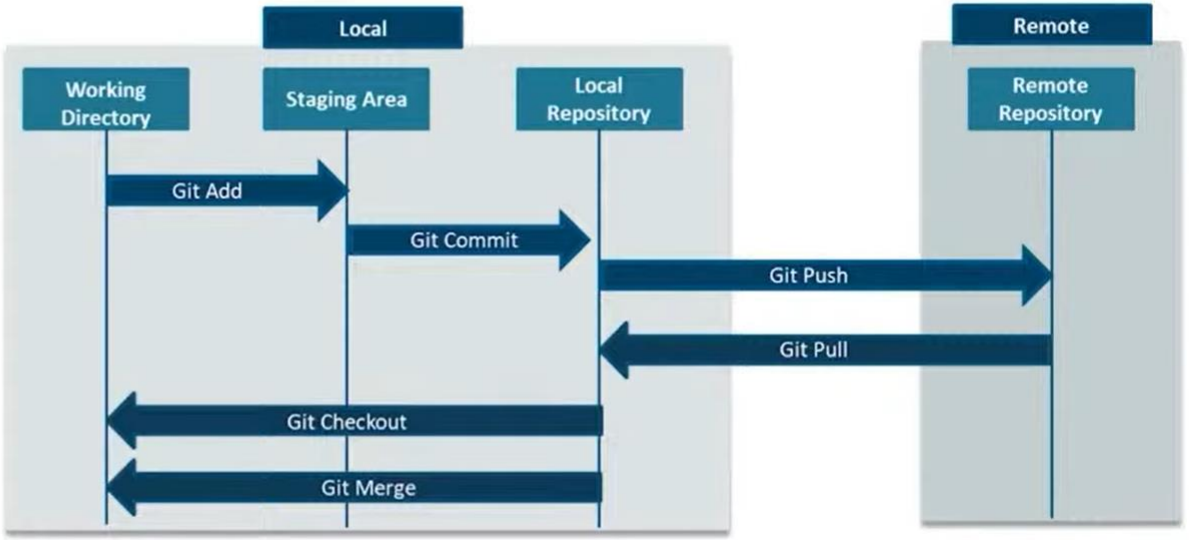
Following is the basic workflow of Git:



GIT Workflow

BUT HOW DOES IT WORK ?

Following is the basic workflow of Git:



Features of Git

- **Open Source**
  - Git is an open-source tool. It is released under the GPL (General Public License) license.
- **Scalable**
  - Git is scalable, which means when the number of users increases, the Git can easily handle such situations.
- **Distributed**
  - One of Git's great features is that it is distributed.
  - Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository.
  - Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project.
  - We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.

Features of Git

- **Security**
  - Git is secure. It uses the SHA1 (Secure Hash Function) to name and identify objects within its repository.
  - Files and commits are checked and retrieved by its checksum at the time of checkout.
  - It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit.
  - Once it is published, one cannot make changes to its old version.
- **Speed**
  - Git is very fast, so it can complete all the tasks in a while.
  - Most of the git operations are done on the local repository, so it provides a huge speed.
  - Also, a centralized version control system continually communicates with a server somewhere.
  - Performance tests conducted by Mozilla showed that it was extremely fast compared to other VCSs.
  - Fetching version history from a locally stored repository is much faster than fetching it from the remote server.
  - The core part of Git is written in C, which ignores runtime overheads associated with other high-level languages.
  - Git was developed to work on the Linux kernel; therefore, it is capable enough to handle large repositories effectively. From the beginning, speed and performance have been Git's primary goals.

Features of Git

- **Branching and Merging**
  - Branching and merging are the great features of Git, which makes it different from the other SCM tools.
  - Git allows the creation of multiple branches without affecting each other.
  - We can perform tasks like creation, deletion, and merging on branches, and these tasks take a few seconds only.

  Below are some features that can be achieved by branching:

  - We can create a separate branch for a new module of the project, commit and delete it whenever we want.
  - We can have a production branch, which always has what goes into production and can be merged for testing in the test branch.
  - We can create a demo branch for the experiment and check if it is working. We can also remove it if needed.
  - The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.

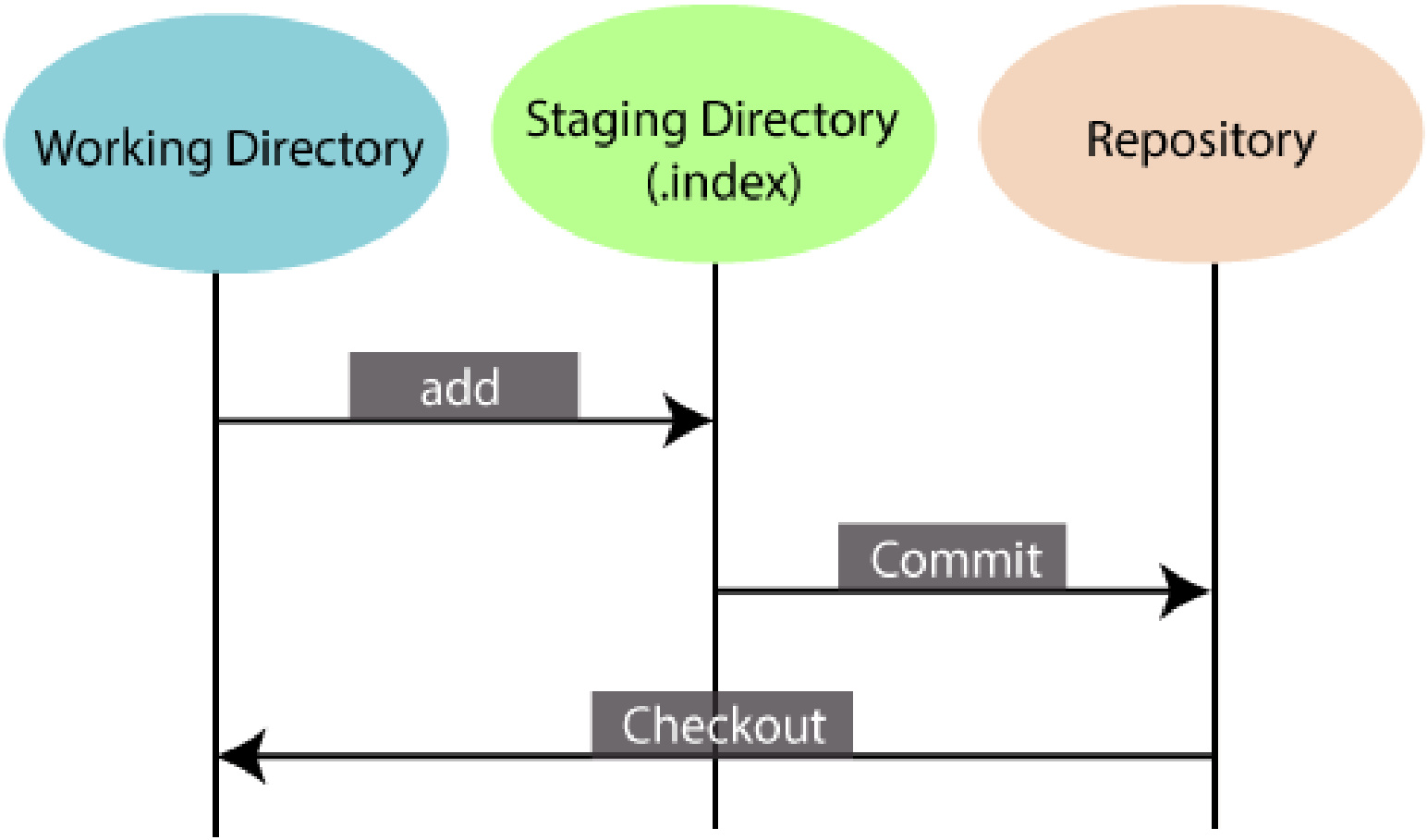
Features of Git

- **Data Assurance**
  - The Git data model ensures the cryptographic integrity of every unit of our project.
  - It provides a unique commit ID to every commit through a SHA algorithm.
  - We can retrieve and update the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.
- **Maintain the clean history**
  - Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

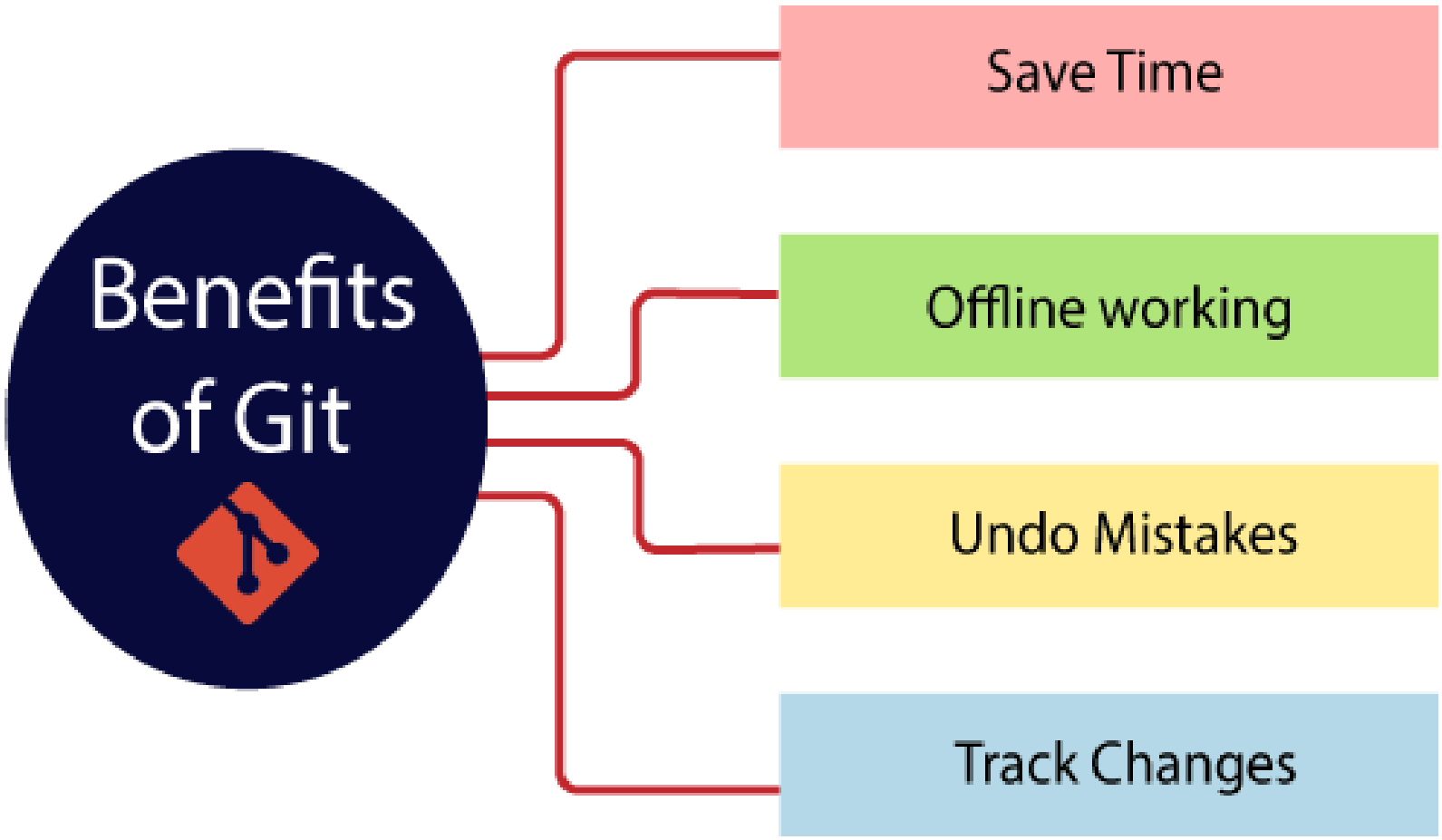
Features of Git

- **Staging Area**
  - The Staging area is also a unique functionality of Git. It can be considered as a preview of our next commit, moreover, an intermediate area where commits can be formatted and reviewed before completion.
  - When you make a commit, Git takes changes that are in the staging area and make them as a new commit.
  - We are allowed to add and remove changes from the staging area.
  - The staging area can be considered as a place where Git stores the changes.
  - Although, Git doesn't have a dedicated staging directory where it can store some objects representing file changes (blobs). Instead of this, it uses a file called index.

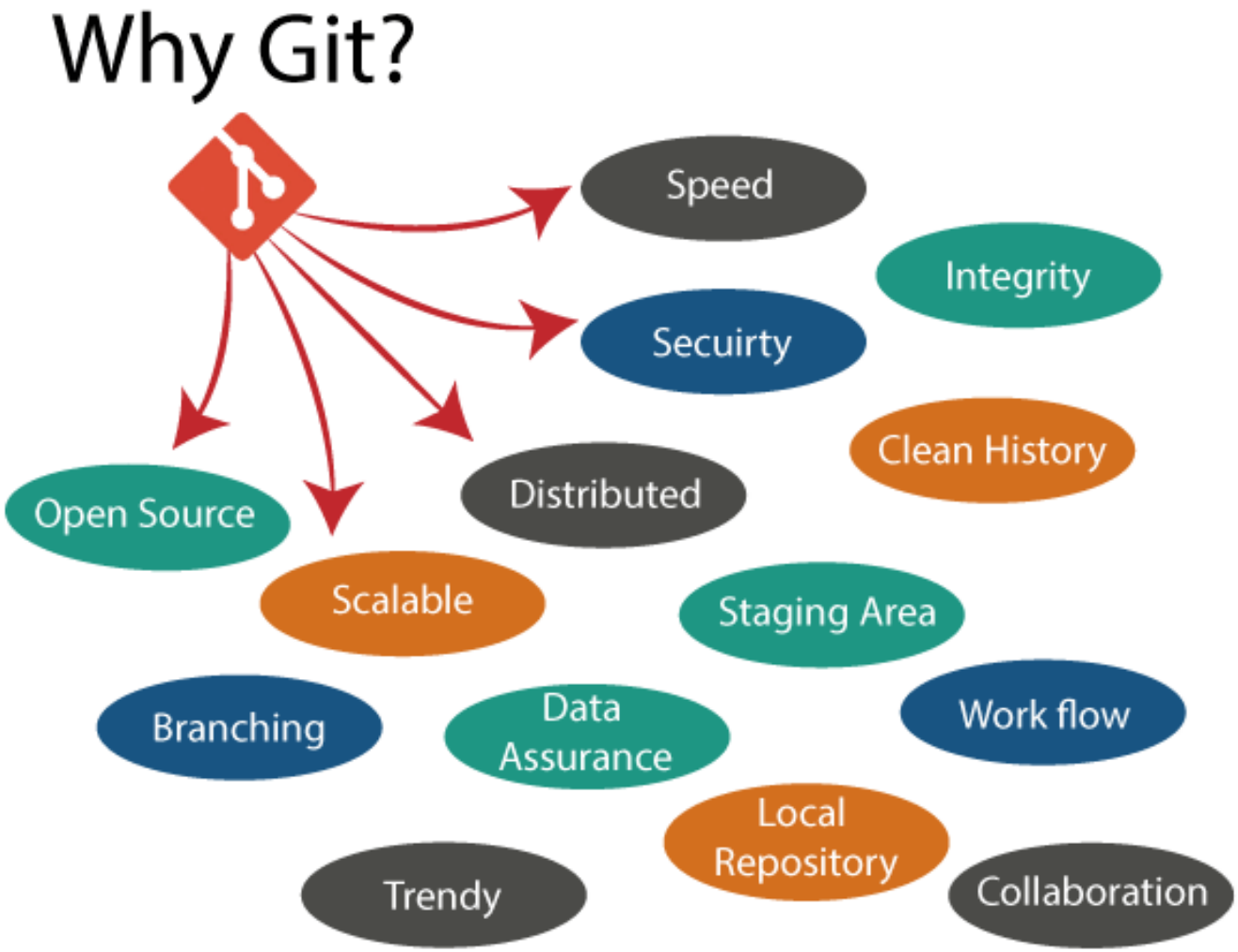
Features of Git



Benefits of Git

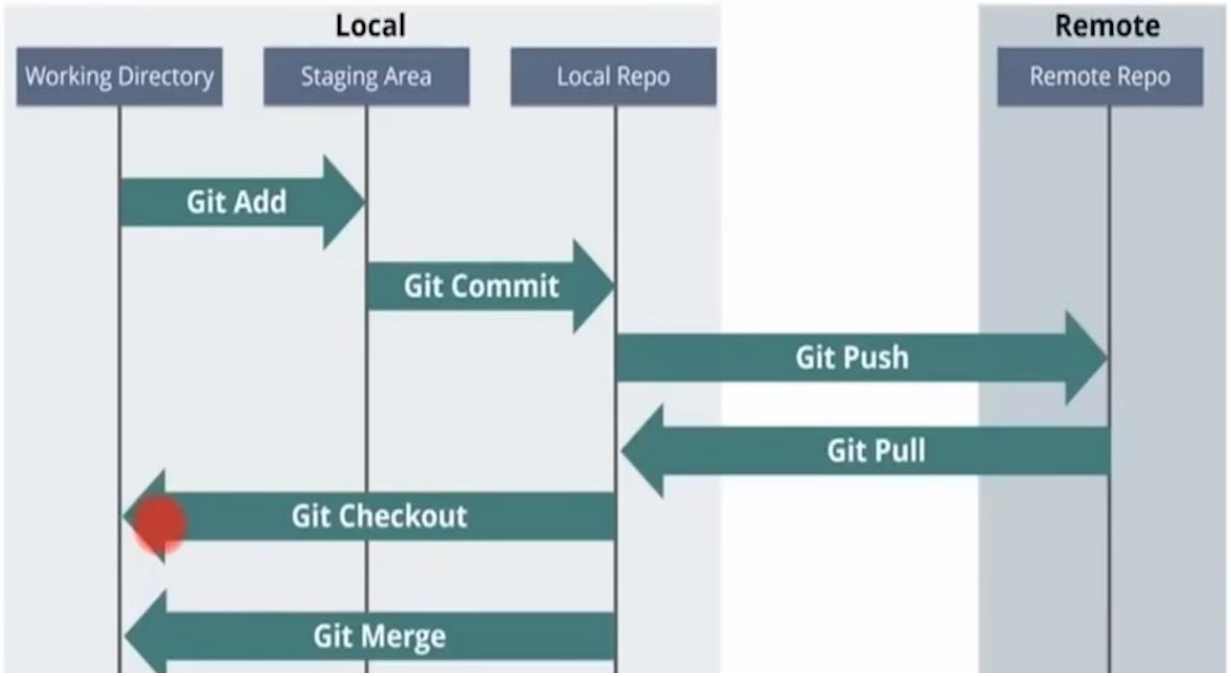


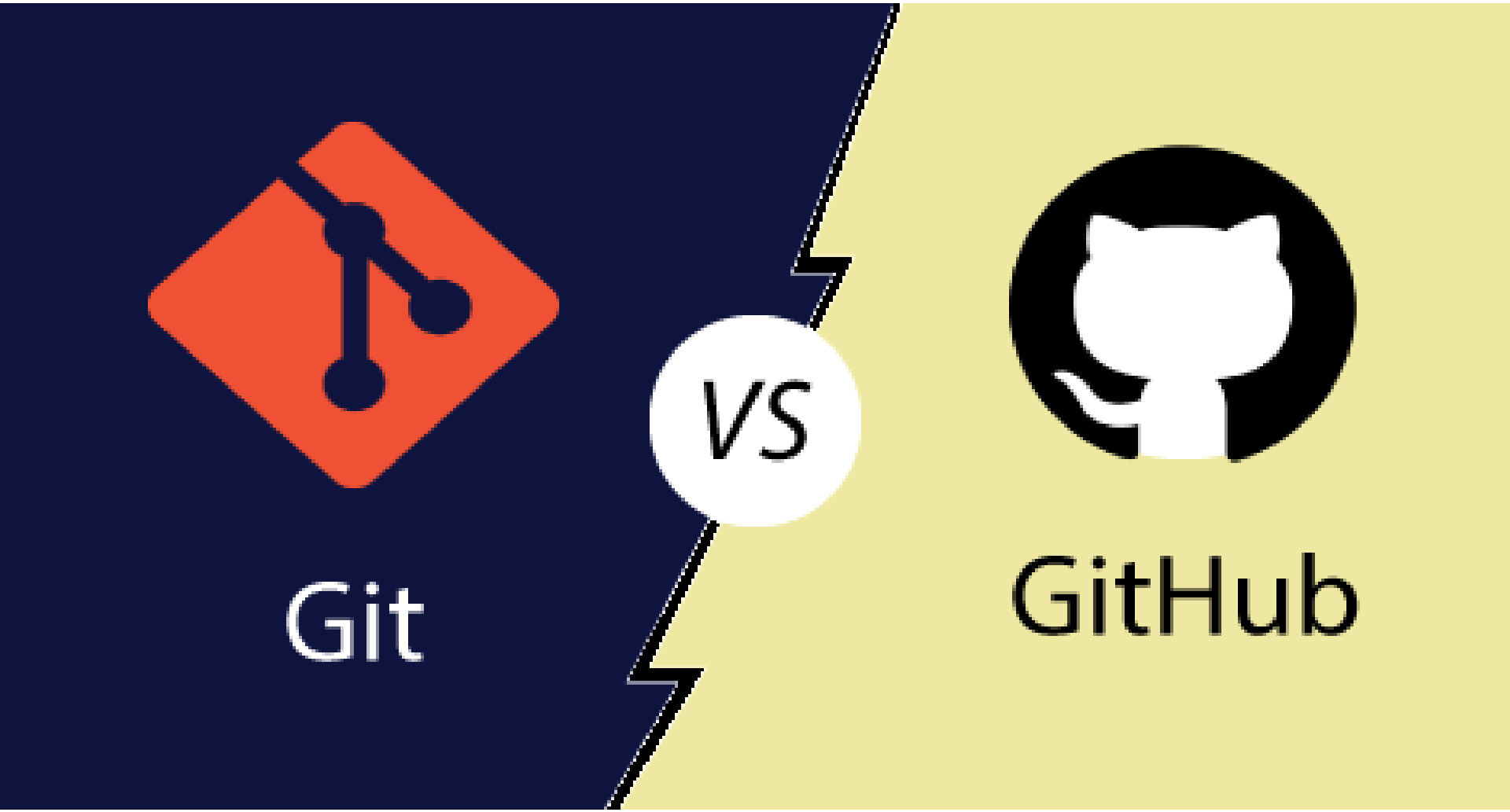
Why Git


Why Git?
Speed
Integrity
Securirty
Clean History
Distributed
Open Source
Scalable
Staging Area
Branching
Data Assurance
Work flow
Local Repository
Trendy
Collaboration

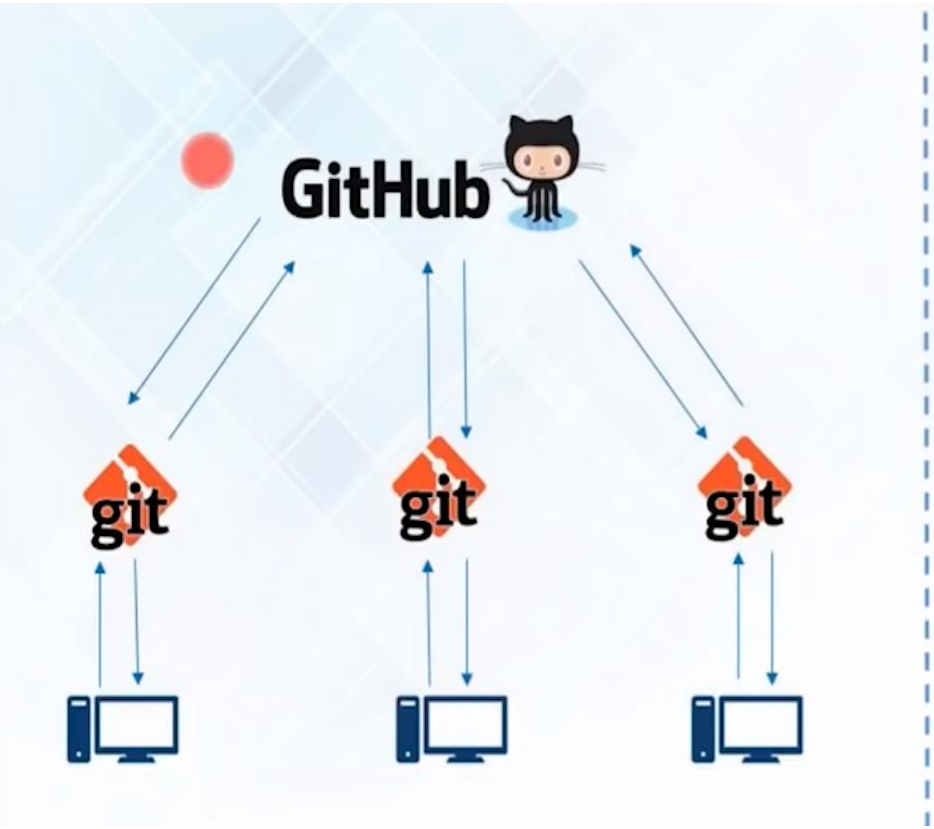
Source Code Management

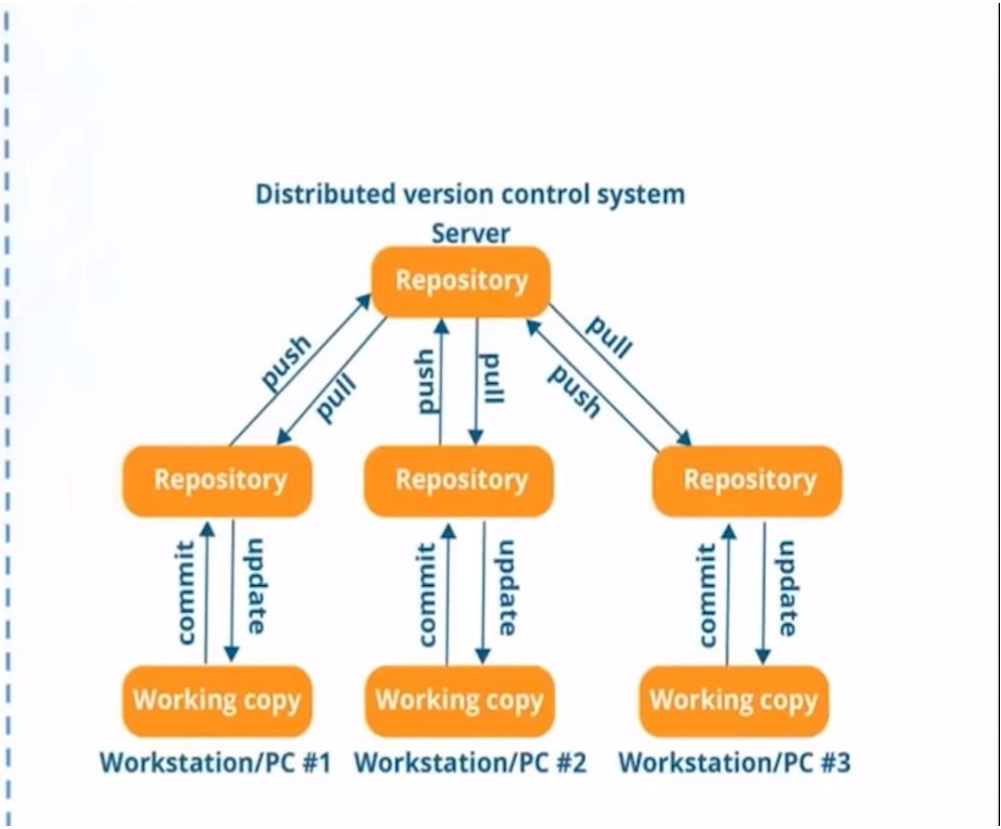
Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software




Git
VS
GitHub

Git vs Github

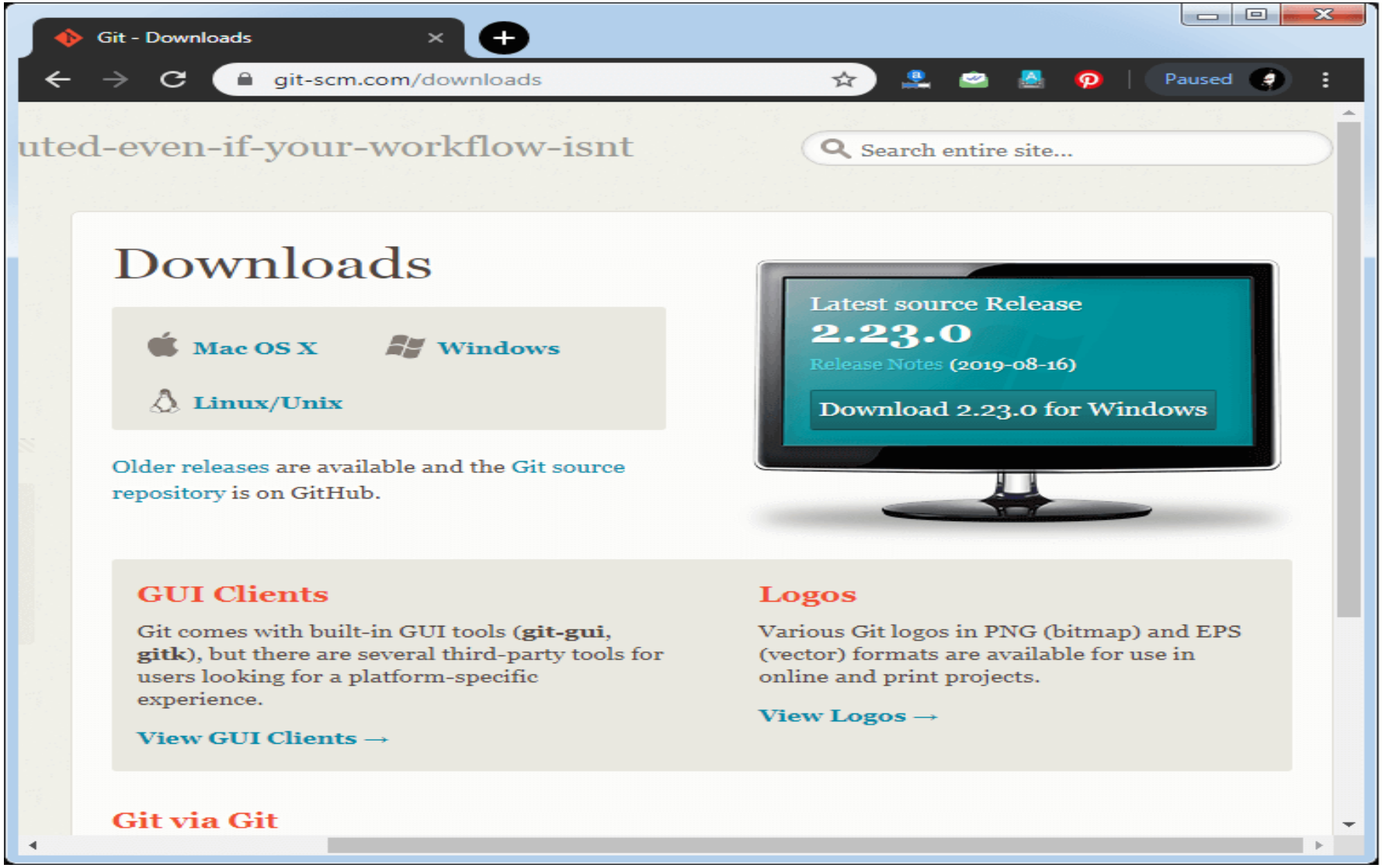




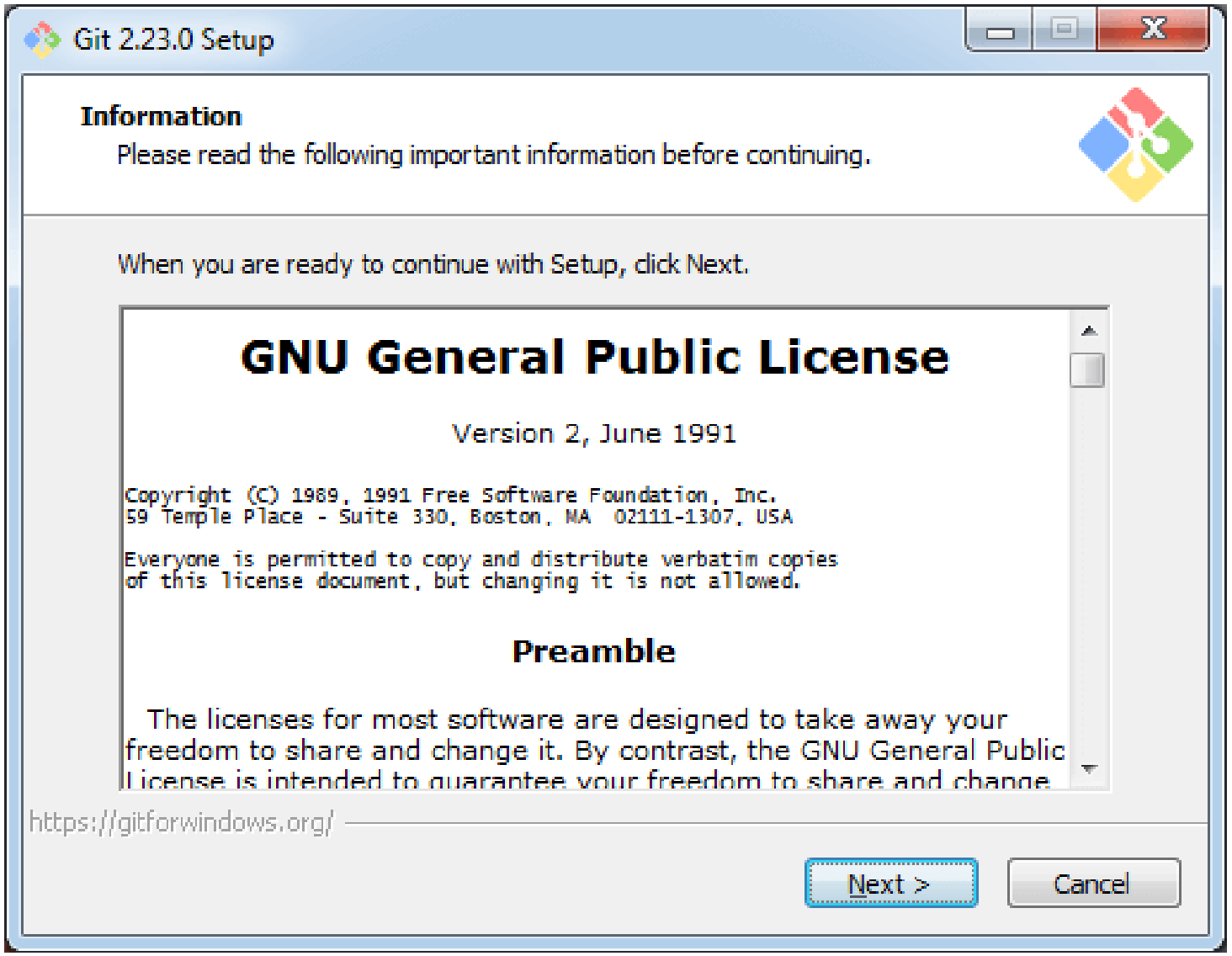
Git vs Github

Git	GitHub
Git is a distributed version control tool that can manage a programmer's source code history.	GitHub is a cloud-based tool developed around the Git tool.
A developer installs Git tool locally.	GitHub is an online service to store code and push from the computer running the Git tool.
Git focused on version control and code sharing.	GitHub focused on centralized source code hosting.
It is a command-line tool.	It is administered through the web.
It facilitates with a desktop interface called Git Gui.	It also facilitates with a desktop interface called GitHub Gui.
Git does not provide any user management feature.	GitHub has a built-in user management feature.
It has minimal tool configuration feature.	It has a market place for tool configuration.

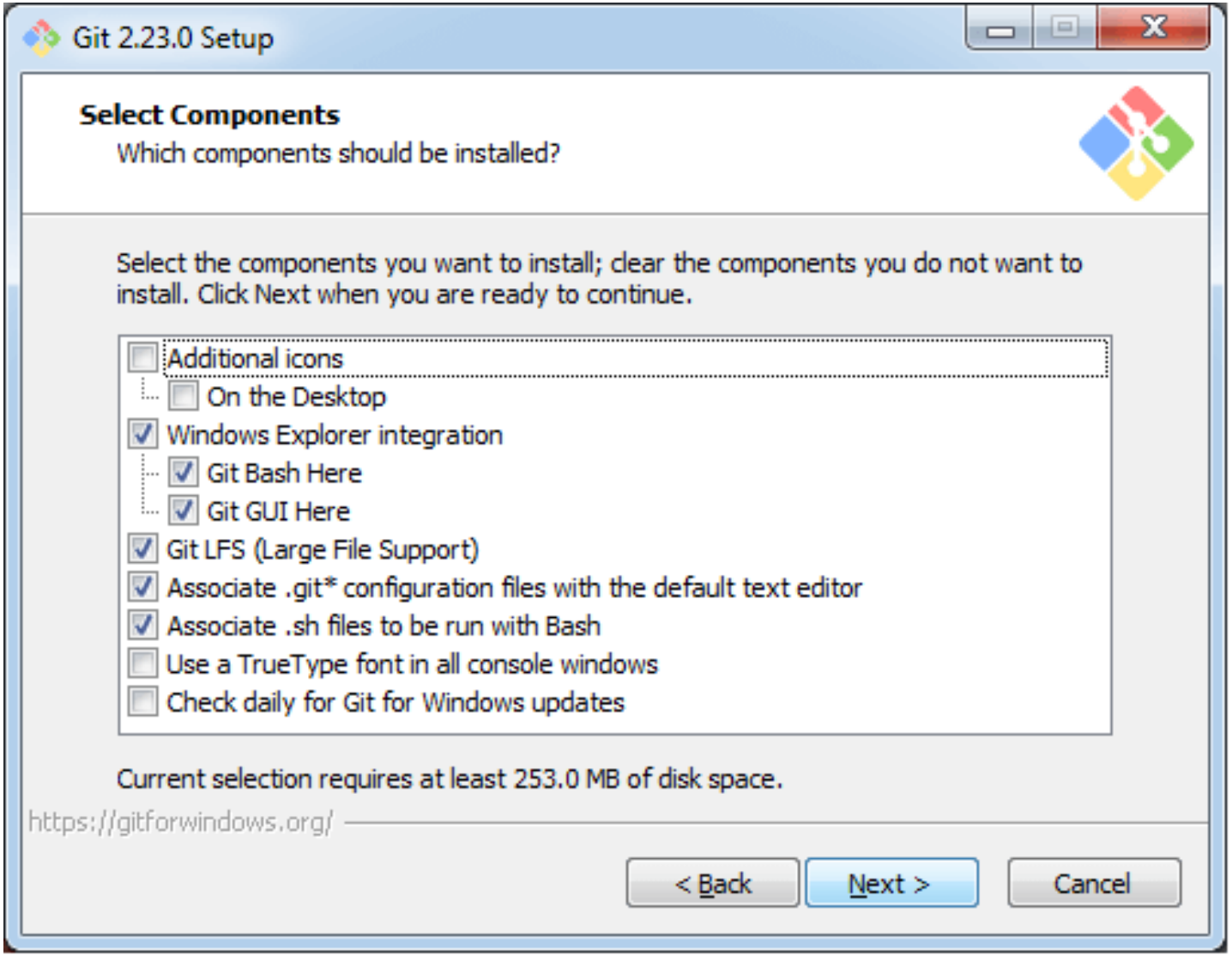
How to Install Git on Windows



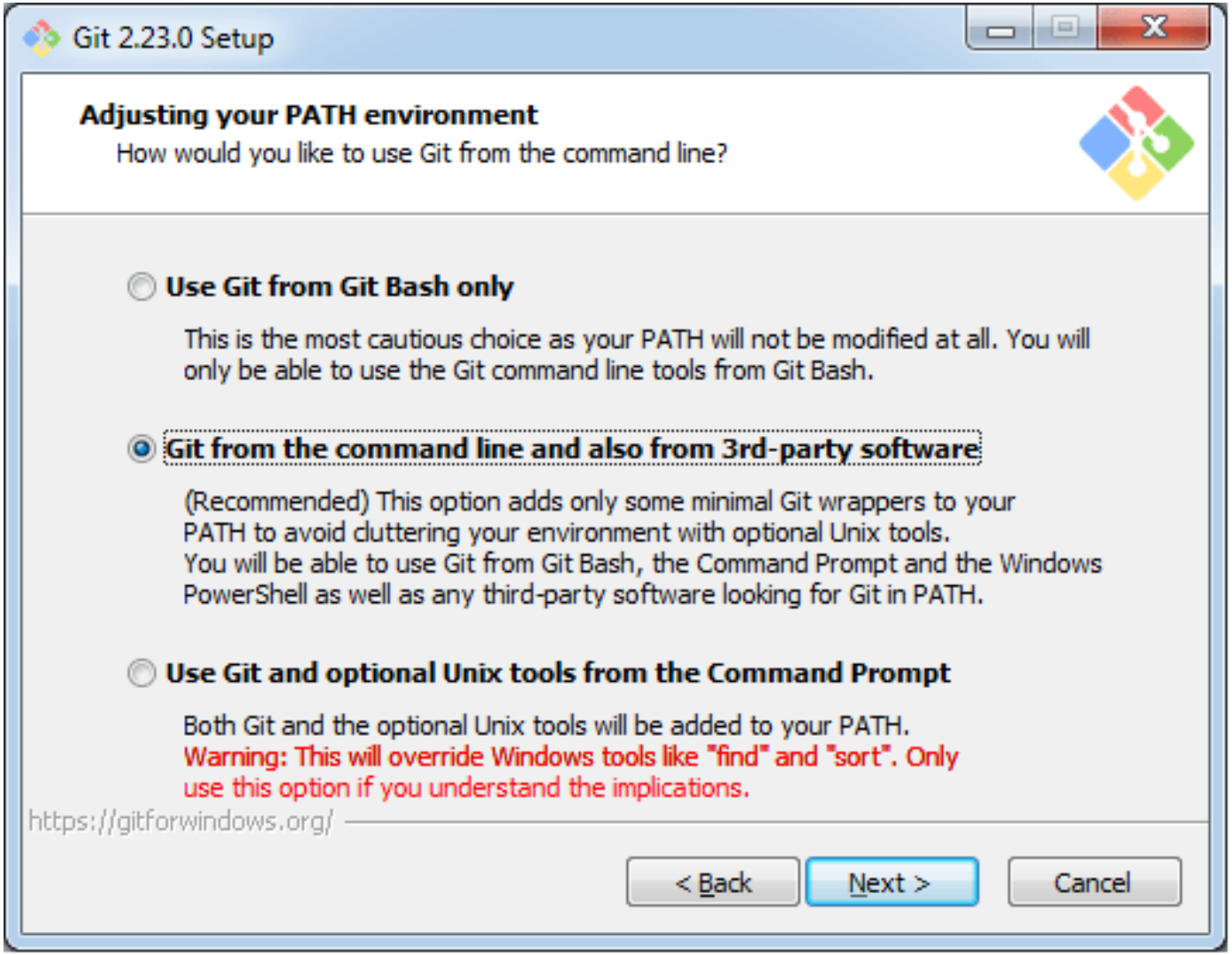
How to Install Git on Windows



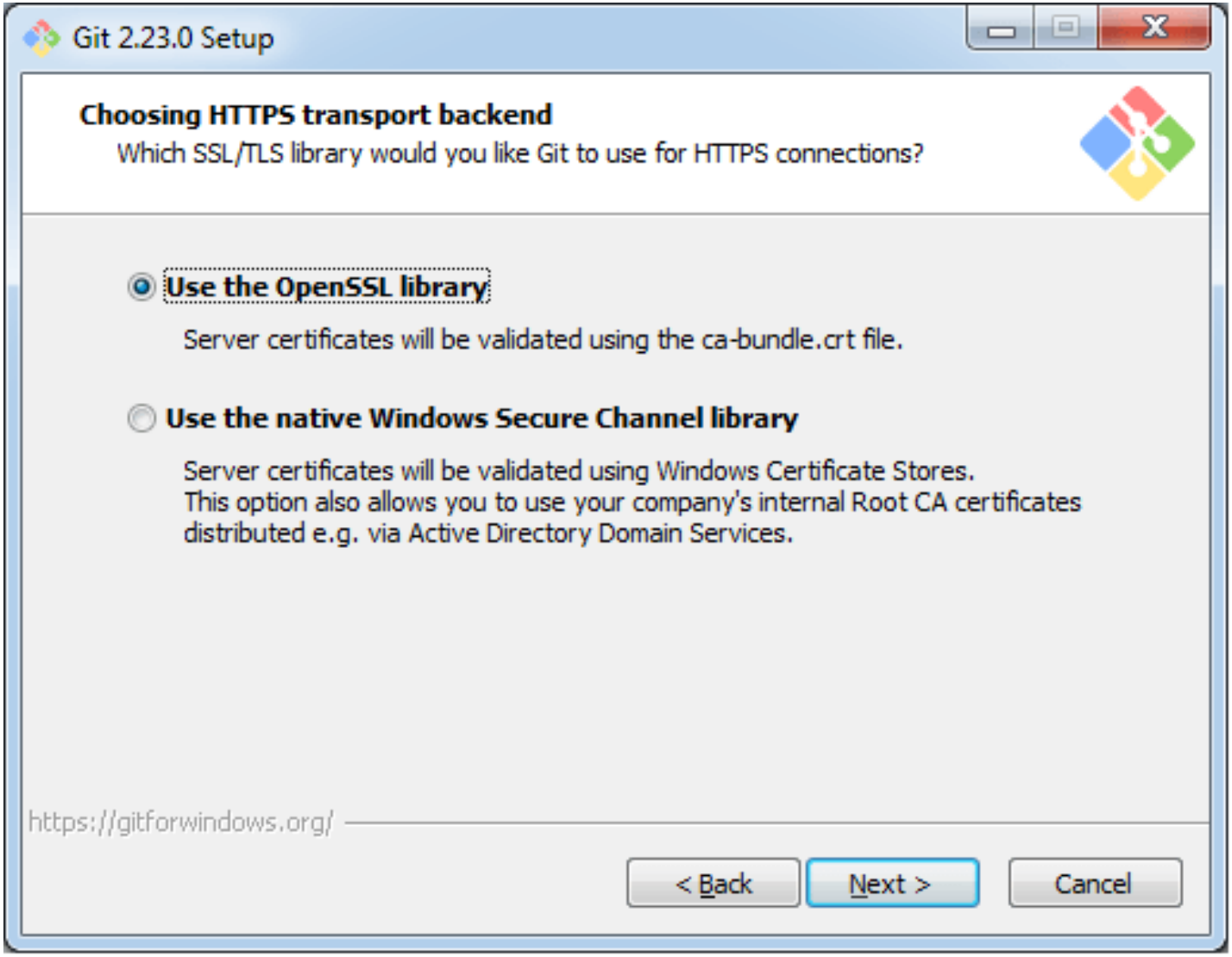
How to Install Git on Windows



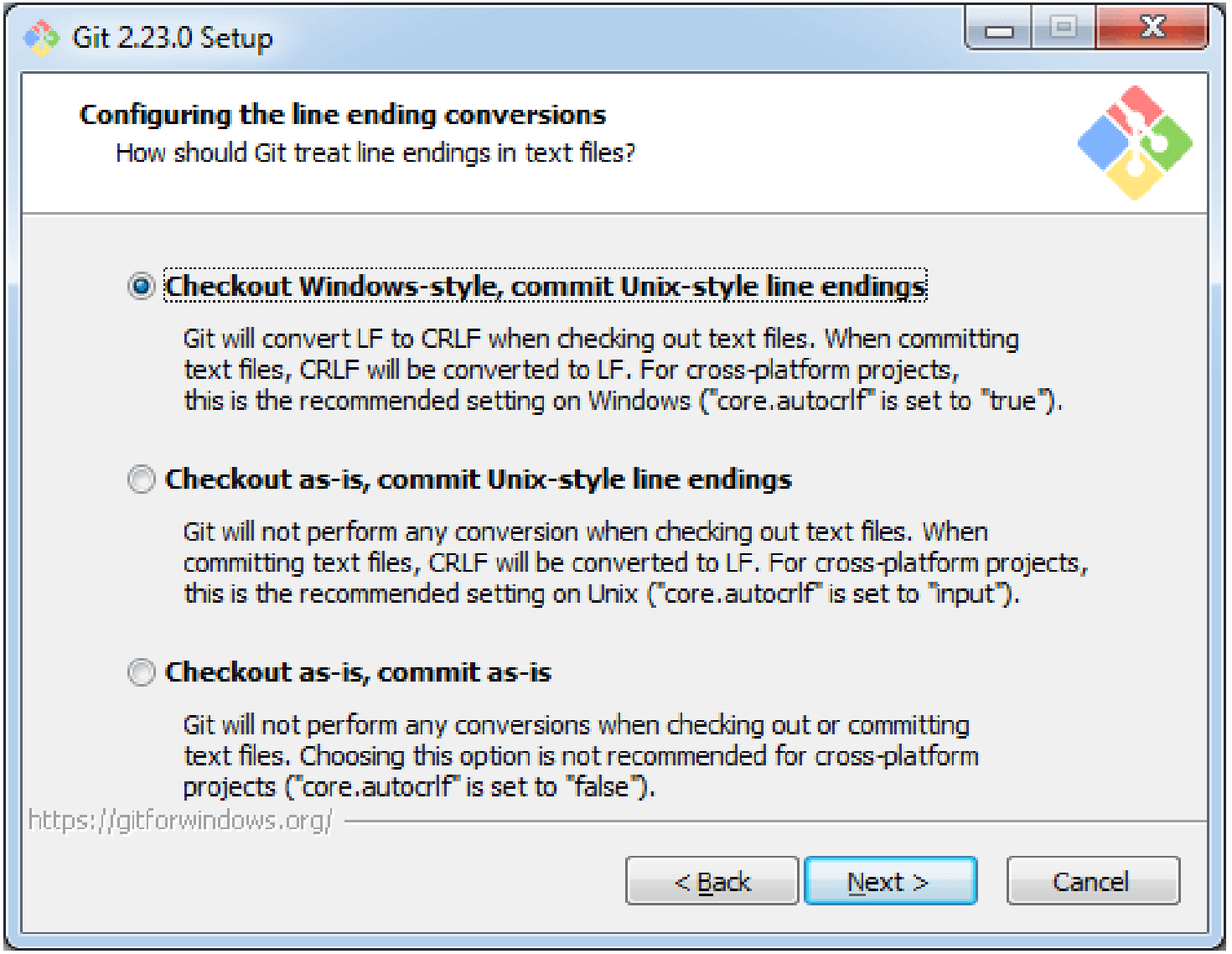
How to Install Git on Windows



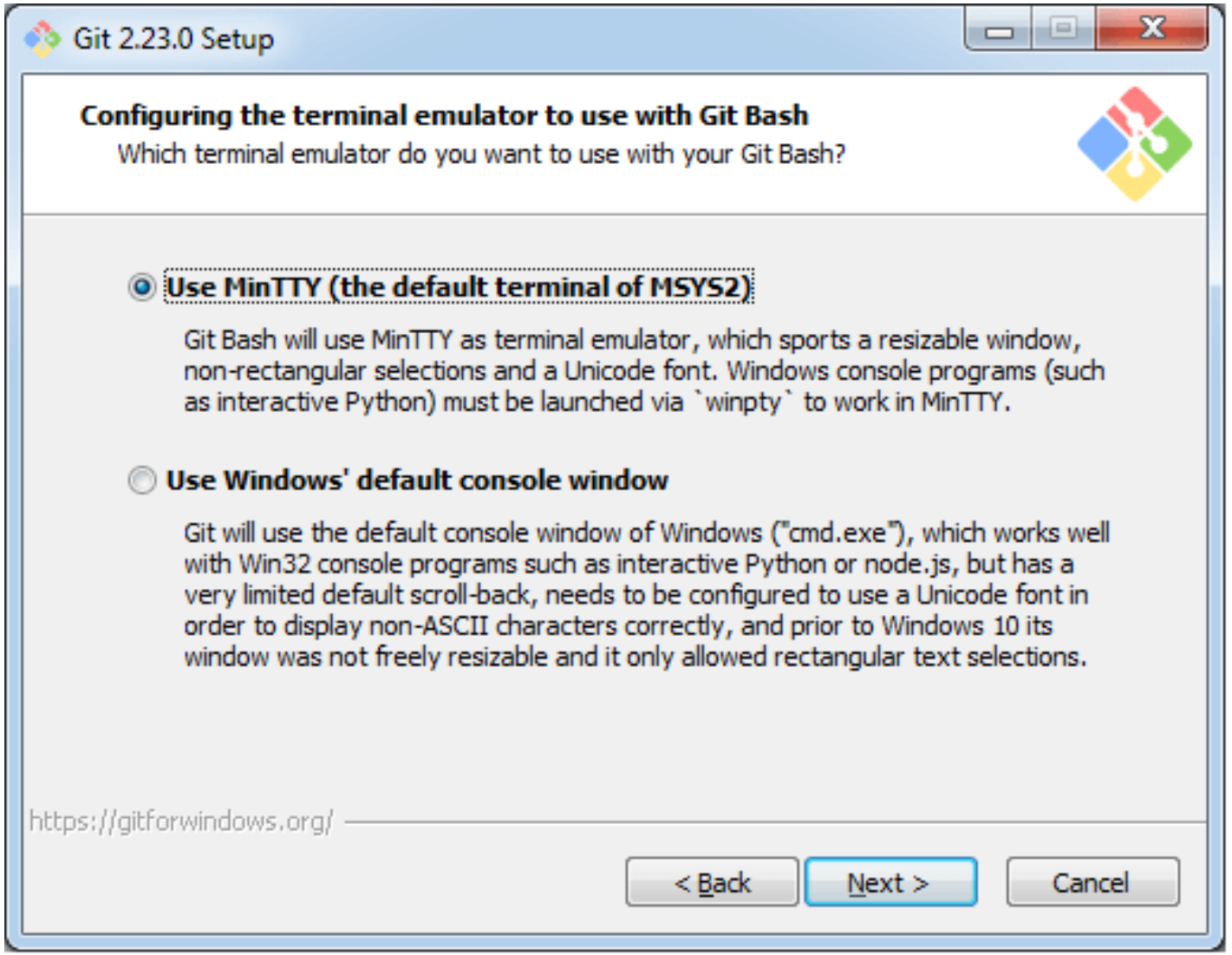
How to Install Git on Windows



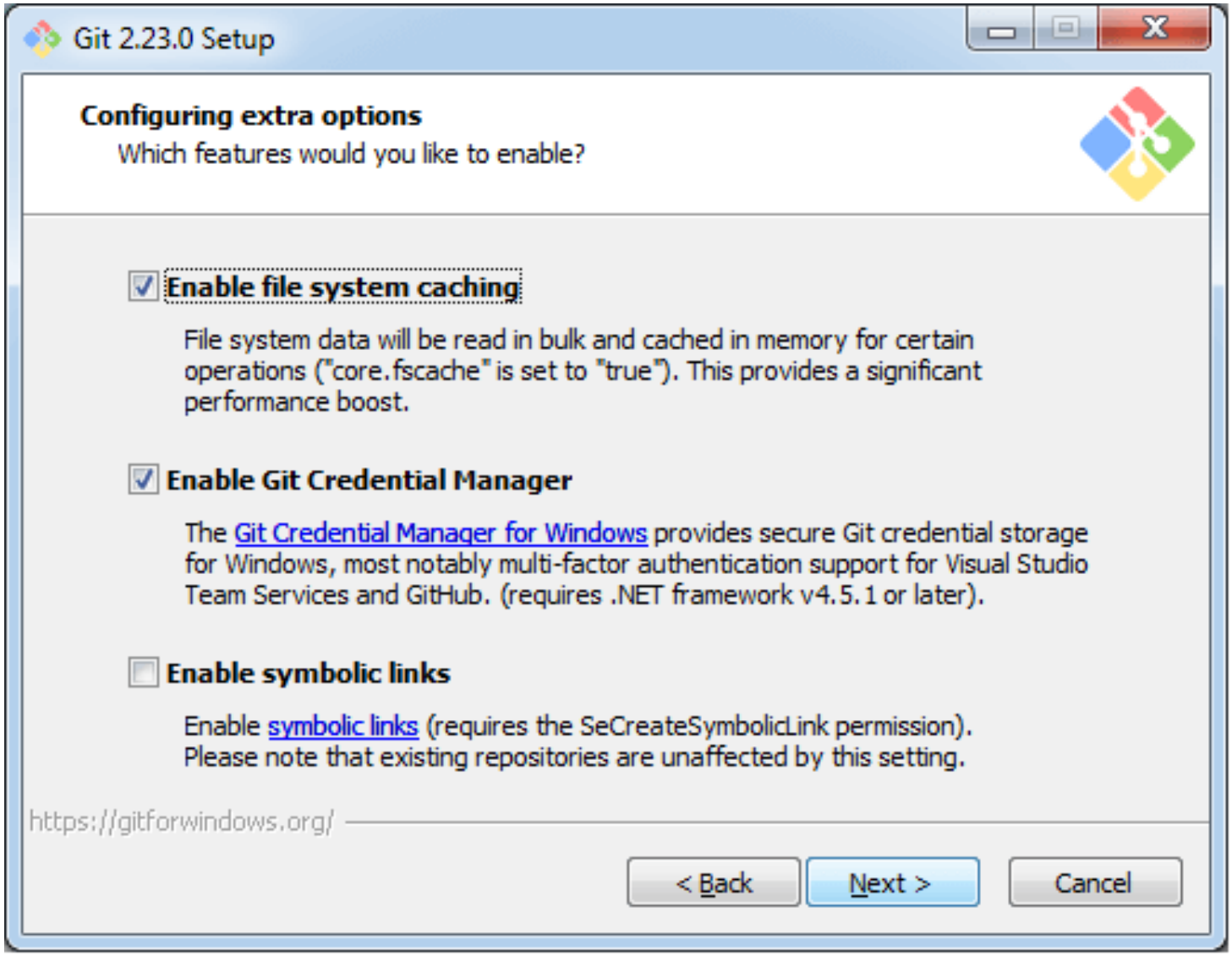
How to Install Git on Windows



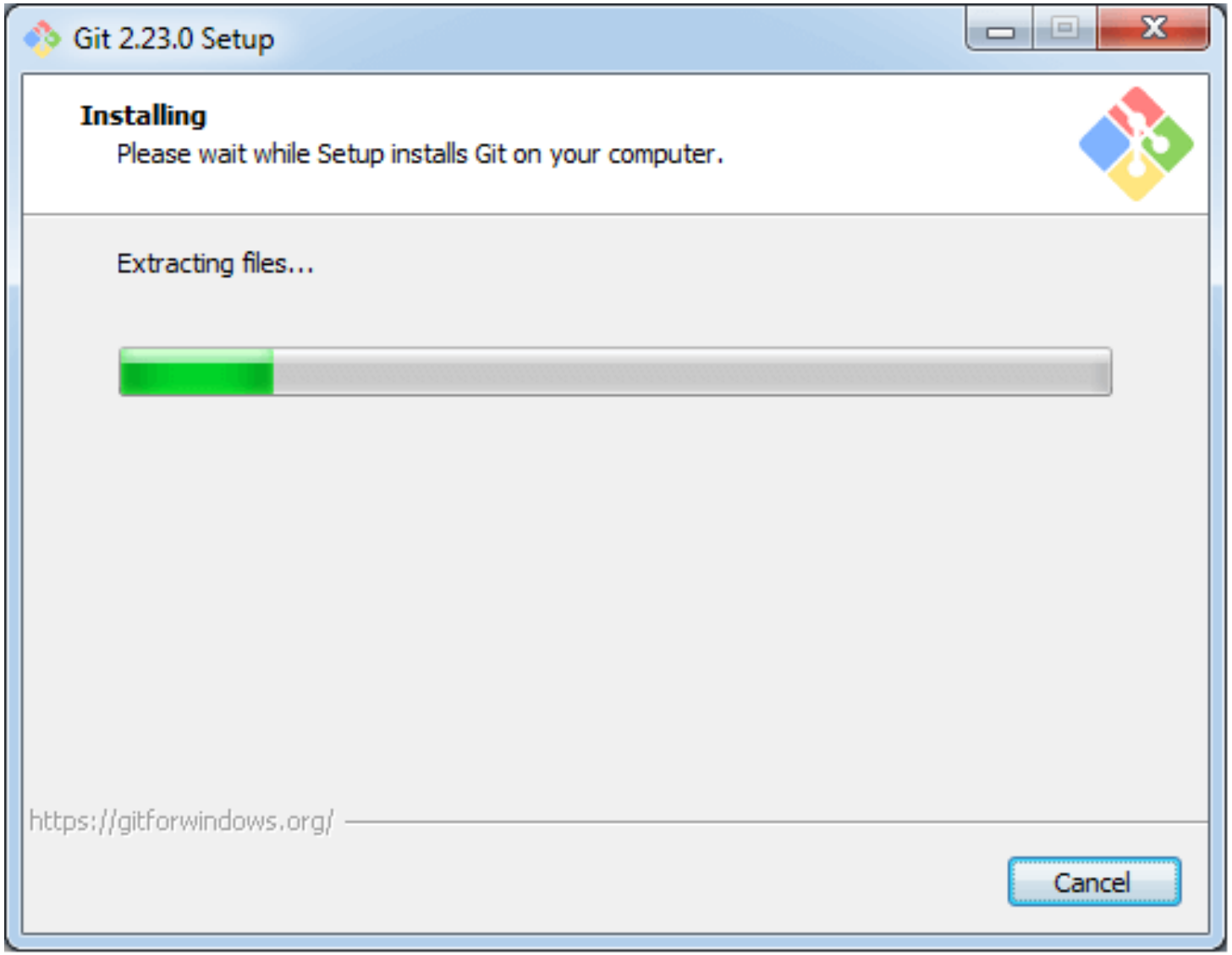
How to Install Git on Windows



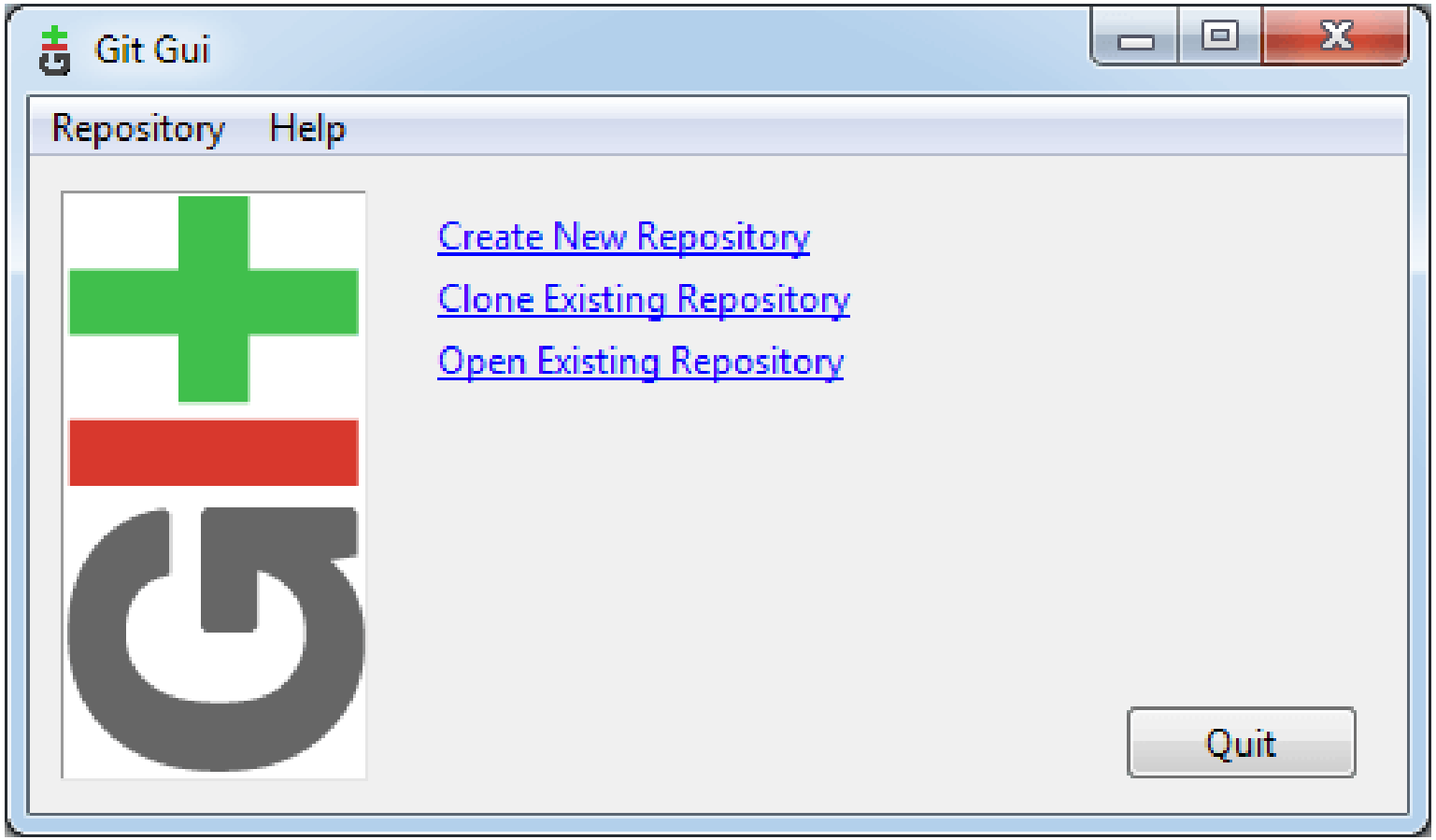
How to Install Git on Windows



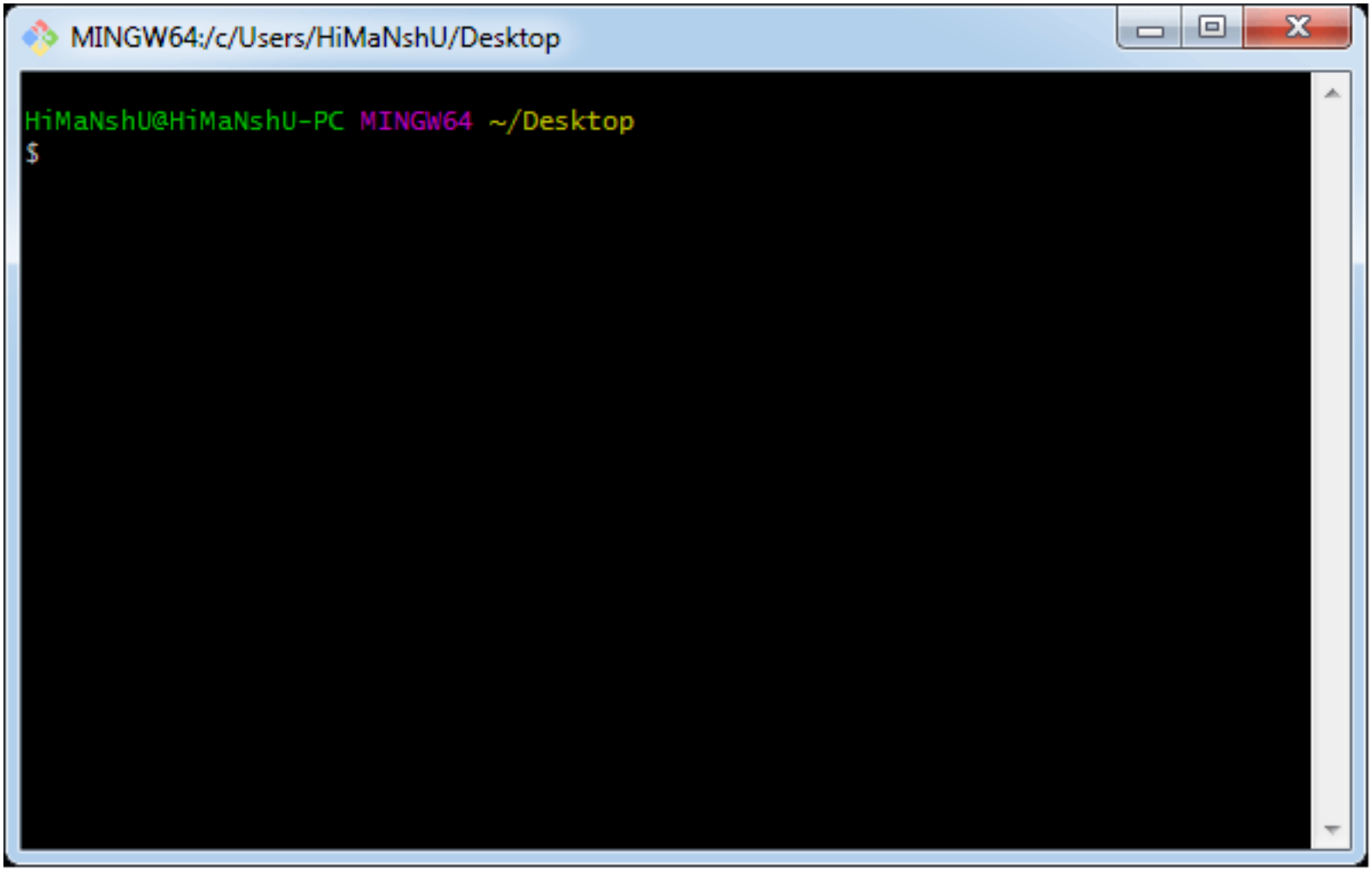
How to Install Git on Windows



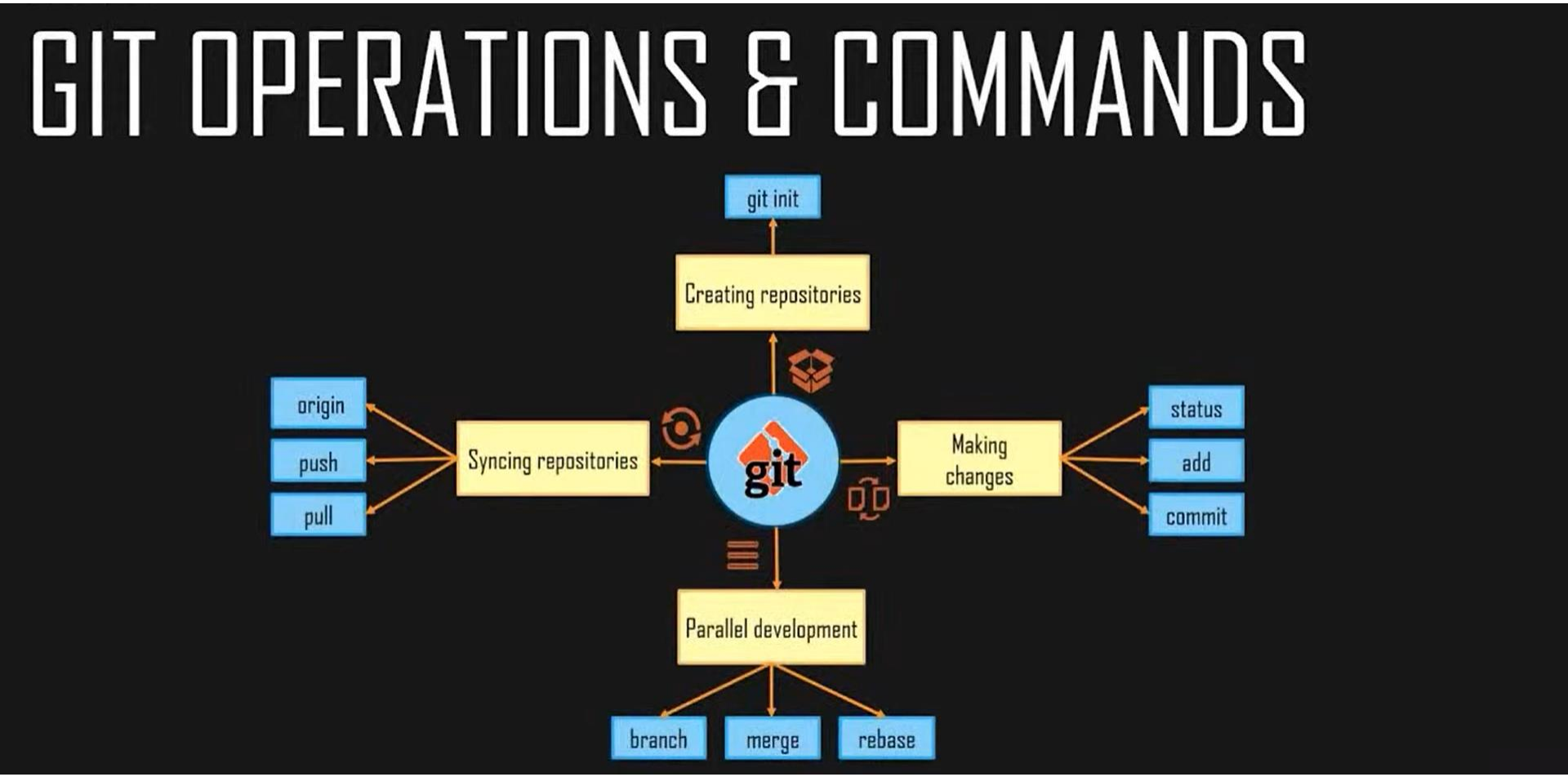
How to Install Git on Windows



How to Install Git on Windows



GIT Commands



GIT Commands

GIT OPERATIONS

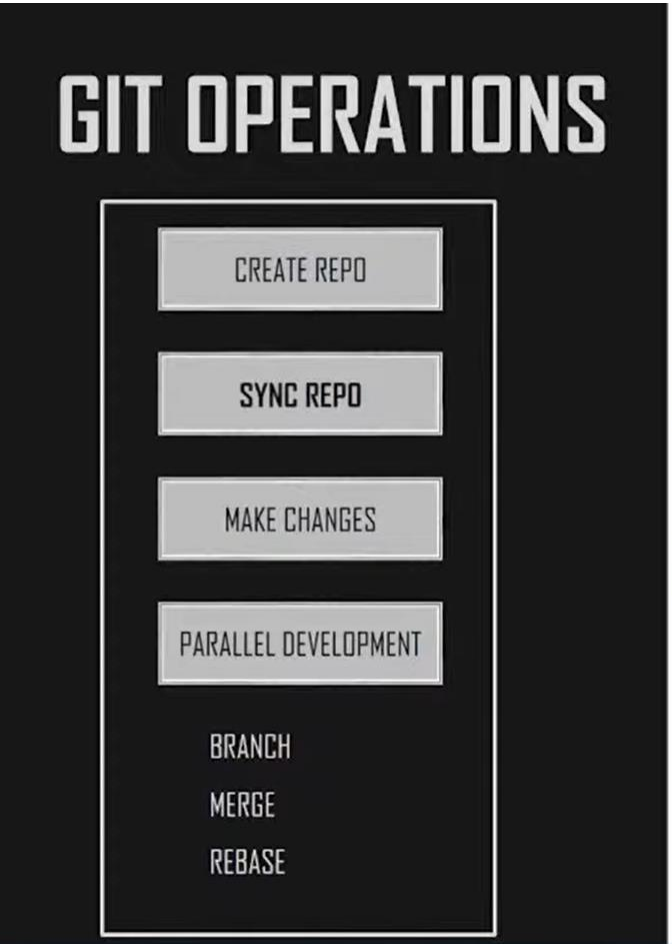
- CREATE REPO
- SYNC REPO
- MAKE CHANGES
- PARALLEL DEVELOPMENT
- BRANCH
- MERGE
- REBASE

git init — The **git init** command creates a new **Git** repository.

git clone —  When you **clone** a repository, you create a copy of the original repository on your **Local machine**.

git fork — When you **fork** a repository, you create a copy of the original repository on your **GitHub account**.

GIT Commands



git origin	`git remote add origin <repo_link>`	Lets you **add** a remote repository.
git pull	`git pull origin master`	Lets you **copy** all the files from the master branch of **remote repository to your local repository**.
git push	`git push origin master`	Lets you push your **local changes into central repository**

Git Environment Setup

- Git supports a command called git config that lets you get and set configuration variables that control all facets of how Git looks and operates.
- It is used to set Git configuration values on a global or local project level.
- Setting user.name and user.email are the necessary configuration options as your name and email will show up in your commit messages.

Git Environment Setup

- $ git config --global user.name “Hemanth"
- $ git config --global user.email hemanth@gmail.com
- git config --global core.editor Vim
- git config - -list
- Git config -global color.ui true

Git configuration levels

- The git config command can accept arguments to specify the configuration level. The following configuration levels are available in the Git config.
  - local
  - global
  - system

Git configuration levels

- --local
- It is the default level in Git. Git config will write to a local level if no configuration option is given. Local configuration values are stored in .git/config directory as a file.
- --global
- The global level configuration is user-specific configuration. User-specific means, it is applied to an individual operating system user. Global configuration values are stored in a user's home directory. ~ /.gitconfig on UNIX systems and C:\Users\\.gitconfig on windows as a file format.

Git Terminology

- **Branch**
  - A branch is a version of the repository that diverges from the main working project. It is an essential feature available in most modern version control systems. A Git project can have more than one branch. We can perform many operations on Git branch-like rename, list, delete, etc.
- **Checkout**
  - In Git, the term checkout is used for the act of switching between different versions of a target entity. The git checkout command is used to switch between branches in a repository.
- **Cherry-Picking**
  - Cherry-picking in Git is meant to apply some commit from one branch into another branch. In case you made a mistake and committed a change into the wrong branch, but do not want to merge the whole branch. You can revert the commit and cherry-pick it on another branch.

Git Terminology

- **Clone**
  - The git clone is a Git command-line utility. It is used to make a copy of the target repository or clone it. If I want a local copy of my repository from GitHub, this tool allows creating a local copy of that repository on your local directory from the repository URL.
- **Fetch**
  - It is used to fetch branches and tags from one or more other repositories, along with the objects necessary to complete their histories. It updates the remote-tracking branches.
- **HEAD**
  - HEAD is the representation of the last commit in the current checkout branch. We can think of the head like a current branch. When you switch branches with git checkout, the HEAD revision changes, and points the new branch.

Git Terminology

- **Index**
  - The Git index is a staging area between the working directory and repository. It is used as the index to build up a set of changes that you want to commit together.
- **Master**
  - Master is a naming convention for Git branch. It's a default branch of Git. After cloning a project from a remote server, the resulting local repository contains only a single local branch. This branch is called a "master" branch. It means that "master" is a repository's "default" branch.
- **Merge**
  - Merging is a process to put a forked history back together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch.

Git Terminology

- **Origin**
  - In Git, "origin" is a reference to the remote repository from a project was initially cloned. More precisely, it is used instead of that original repository URL to make referencing much easier.

- **Pull/Pull Request**
  - The term Pull is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory. The git pull command is used to make a Git pull.

  - Pull requests are a process for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their remote server account. Pull request announces all the team members that they need to review the code and merge it into the master branch.

Git Terminology

- **Push**
  - The push term refers to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repository. Pushing is capable of overwriting changes; caution should be taken when pushing.
- **Rebase**
  - In Git, the term rebase is referred to as the process of moving or combining a sequence of commits to a new base commit. Rebasing is very beneficial and visualized the process in the environment of a feature branching workflow.
  - From a content perception, rebasing is a technique of changing the base of your branch from one commit to another.
- **Remote**
  - In Git, the term remote is concerned with the remote repository. It is a shared repository that all team members use to exchange their changes. A remote repository is stored on a code hosting service like an internal server, GitHub, Subversion and more.
  - In case of a local repository, a remote typically does not provide a file tree of the project's current state, as an alternative it only consists of the .git versioning data.

Git Terminology

- **Repository**
  - In Git, Repository is like a data structure used by VCS to store metadata for a set of files and directories. It contains the collection of the file as well as the history of changes made to those files. Repositories in Git is considered as your project folder. A repository has all the project-related data. Distinct projects have distinct repositories.
- **Stashing**
  - Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The git stash command enables you to switch branch without committing the current branch.
- **Tag**
  - Tags make a point as a specific point in Git history. It is used to mark a commit stage as important. We can tag a commit for future reference. Primarily, it is used to mark a projects initial point like v1.1. There are two types of tags.
- **Light-weighted tag**
- **Annotated tag**

Git Terminology

- **Upstream And Downstream**
  - The term upstream and downstream is a reference of the repository. Generally, upstream is where you cloned the repository from (the origin) and downstream is any project that integrates your work with other works. However, these terms are not restricted to Git repositories.
- **Git Revert**
  - In Git, the term revert is used to revert some commit. To revert a commit, git revert command is used. It is an undo type command. However, it is not a traditional undo alternative.
- **Git Reset**
  - In Git, the term reset stands for undoing changes. The git reset command is used to reset the changes. The git reset command has three core forms of invocation. These forms are as follows.
    - **Soft**
    - **Mixed**
    - **Hard**

Git Terminology

- **Git Ignore**
  - In Git, the term ignore used to specify intentionally untracked files that Git should ignore. It doesn't affect the Files that already tracked by Git.
- **Git Diff**
  - Git diff is a command-line utility. It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc.
- **Git Cheat Sheet**
  - A Git cheat sheet is a summary of Git quick references. It contains basic Git commands with quick installation. A cheat sheet or crib sheet is a brief set of notes used for quick reference. Cheat sheets are so named because the people may use it without no prior knowledge.

Git Terminology

- **Git Fork**
- A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project.
- Great use of using forks to propose changes for bug fixes. To resolve an issue for a bug that you found, you can:
  - Fork the repository.
  - Make the fix.
  - Forward a pull request to the project owner.

Git command line

- Git Config command
- Git init command
- Git clone command
- Git add command
- Git commit command
- Git status command
- Git push Command
- Git pull command
- Git Branch Command
- Git Merge Command
- Git log command
- Git remote command

Git command line

- **Git config command**
- This command configures the user. The Git config command is the first and necessary command used on the Git command line. This command sets the author name and email address to be used with your commits. Git config is also used in other scenarios.
- Syntax
  - **$ git config --global user.name "ImHemanth"**
  - **$ git config --global user.email "Hemanth@gmail.com"**
- **Git Init command**
- This command is used to create a local repository.
- Syntax
  - **$ git init Demo**

Git command line

- **Git clone command**
- This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.
- Syntax
- **$ git clone URL**

Git command line

- **Git add command**
- This command is used to add one or more files to staging (Index) area.
- Syntax
- To add one file
- **$ git add Filename**
- To add more than one file
- **$ git add***

Git command line

- **Git commit command**
- Commit command is used in two scenarios. They are as follows.
- **Git commit -m**
- This command changes the head. It records or snapshots the file permanently in the version history with a message.
- Syntax
- $ **git commit -m " Commit Message"**
- **Git commit -a**
- This command commits any files added in the repository with git add and also commits any files you've changed since then.
- Syntax
- $ **git commit -a**

Git command line

- **Git status command**
- The status command is used to display the state of the working directory and the staging area.
- It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git.
- It does not show you any information about the committed project history.
- For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.
- Syntax
- **$ git status**

Git command line

- **Git status command**
- The status command is used to display the state of the working directory and the staging area.
- It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git.
- It does not show you any information about the committed project history.
- For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.
- Syntax
- **$ git status**

Git command line

- **Git push Command**
  - It is used to upload local repository content to a remote repository.
  - Pushing is an act of transfer commits from your local repository to a remote repo.
  - It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches.
  - Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.
  - Git push command can be used as follows.
  - Git push origin master
  - This command sends the changes made on the master branch, to your remote repository.
- Syntax
- $ **git push [variable name] master**

Git command line

- **Git pull command**
- Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.
- **Syntax**
- **$ git pull URL**

Git command line

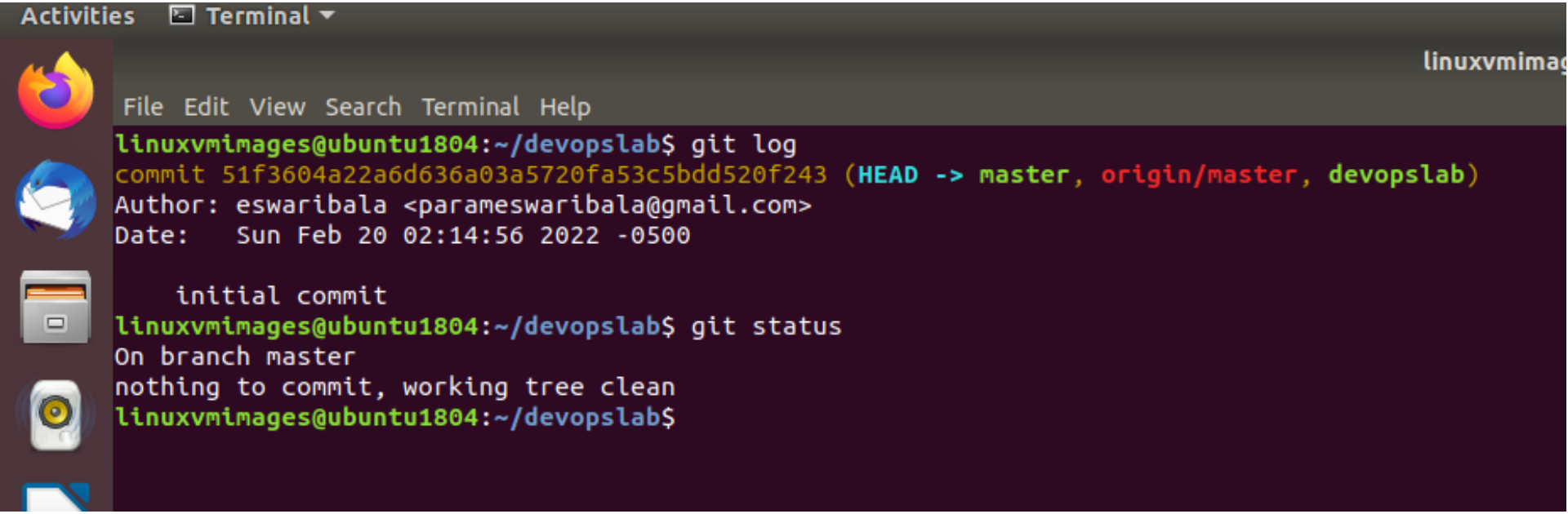
- **Git Branch Command**
  - This command lists all the branches available in the repository.
- Syntax
- **$ git branch**
- **Git Merge Command**
  - This command is used to merge the specified branch?s history into the current branch.
- Syntax
- **$ git merge BranchName**

Git command line

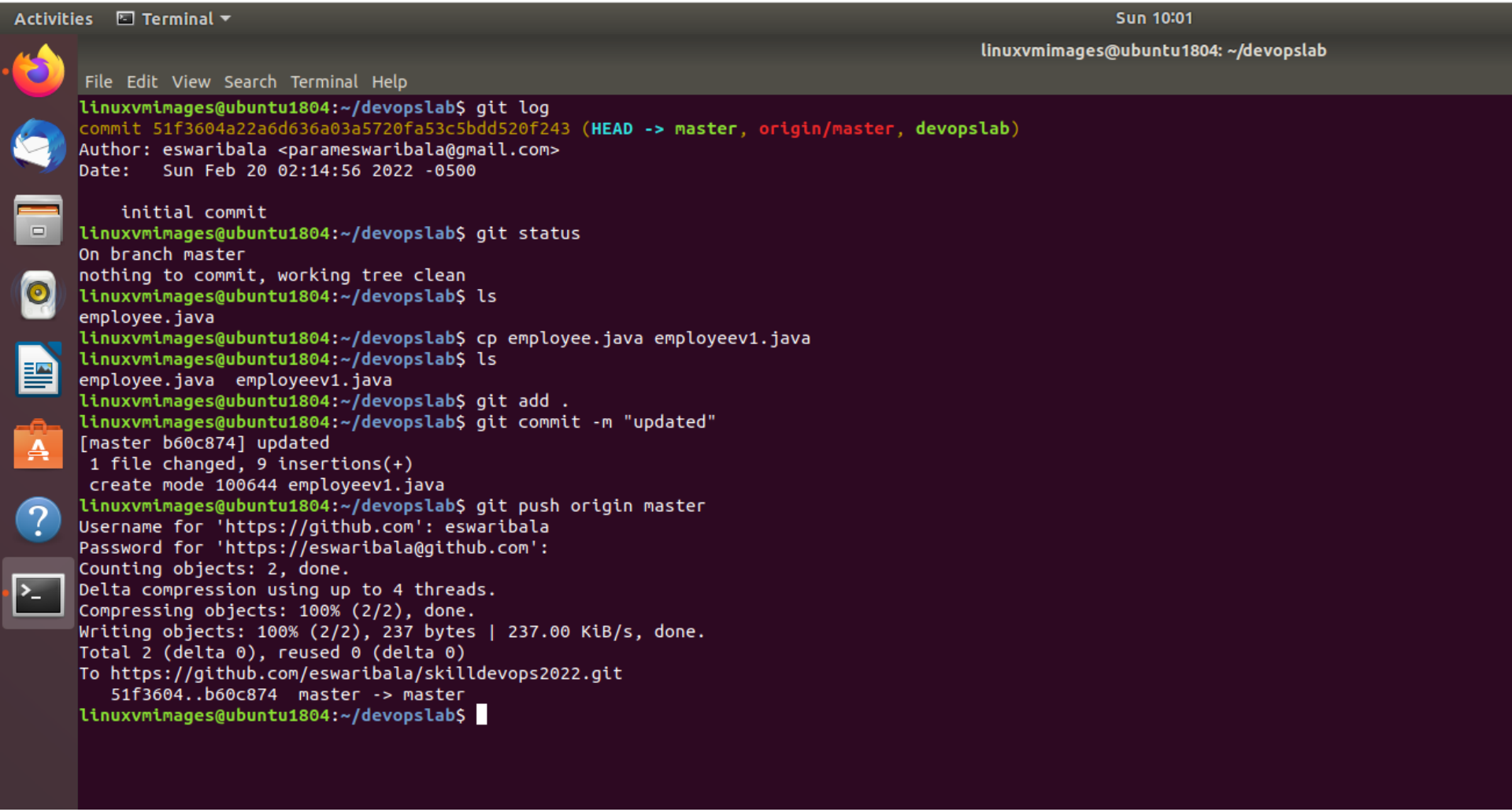
- **Git log Command**
  - This command is used to check the commit history.
- Syntax
- **$ git log**

- **Cheatsheet**

GIT Commands



GIT Commands



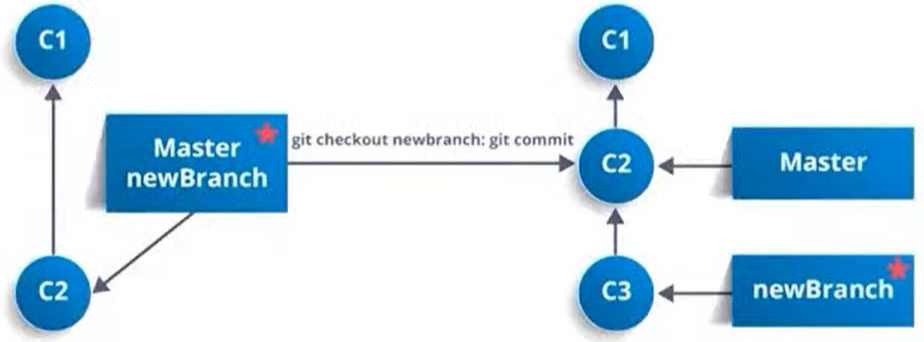
Parallel Development

GIT

GIT OPERATIONS

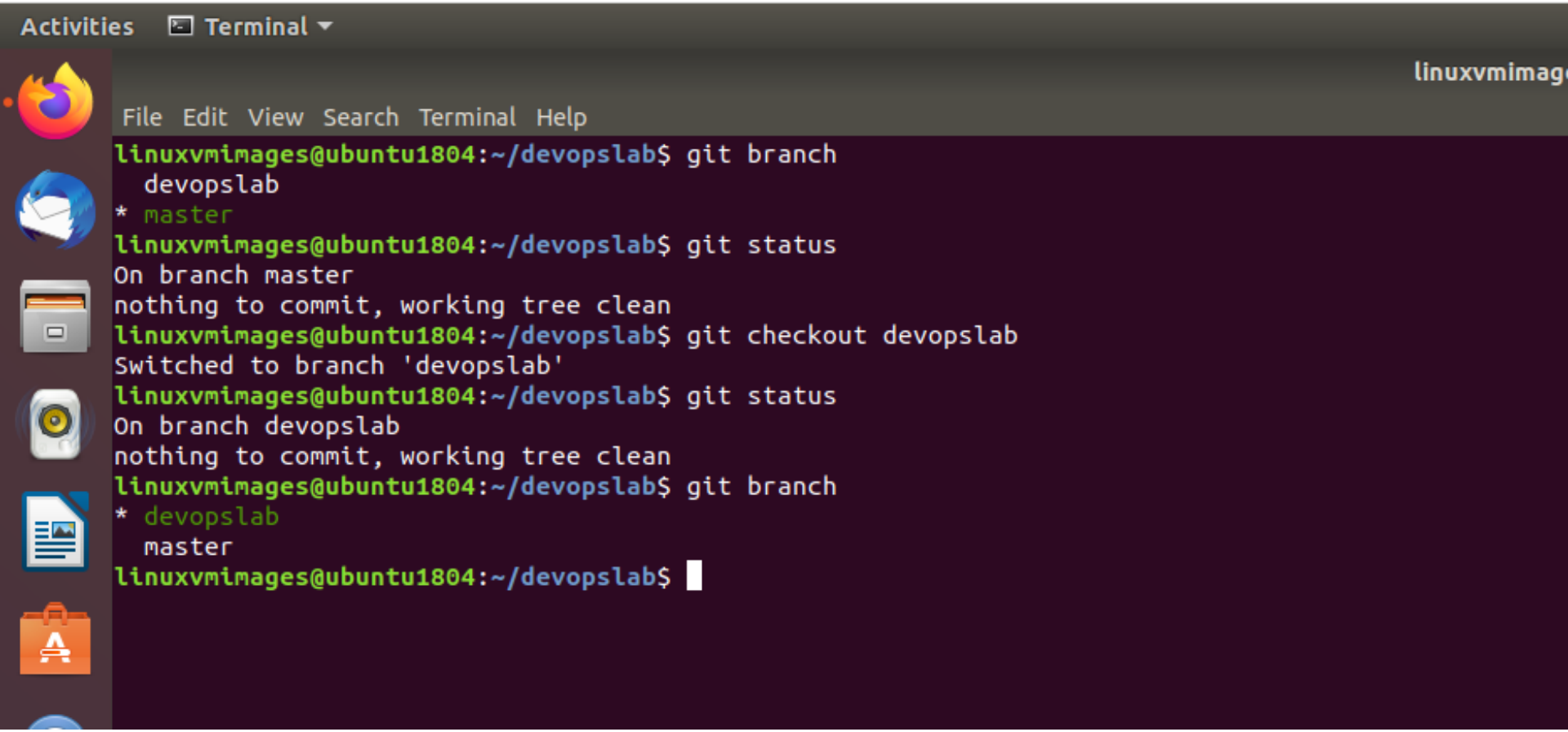
- CREATE REPO
- SYNC REPO
- MAKE CHANGES
- PARALLEL DEVELOPMENT
  - BRANCH
  - MERGE
  - REBASE

Branching is an integral part of any Version Control System. Unlike other VCS, Git **does not create a copy** of existing files for new branch. It points to snapshot of the changes you have made in the system

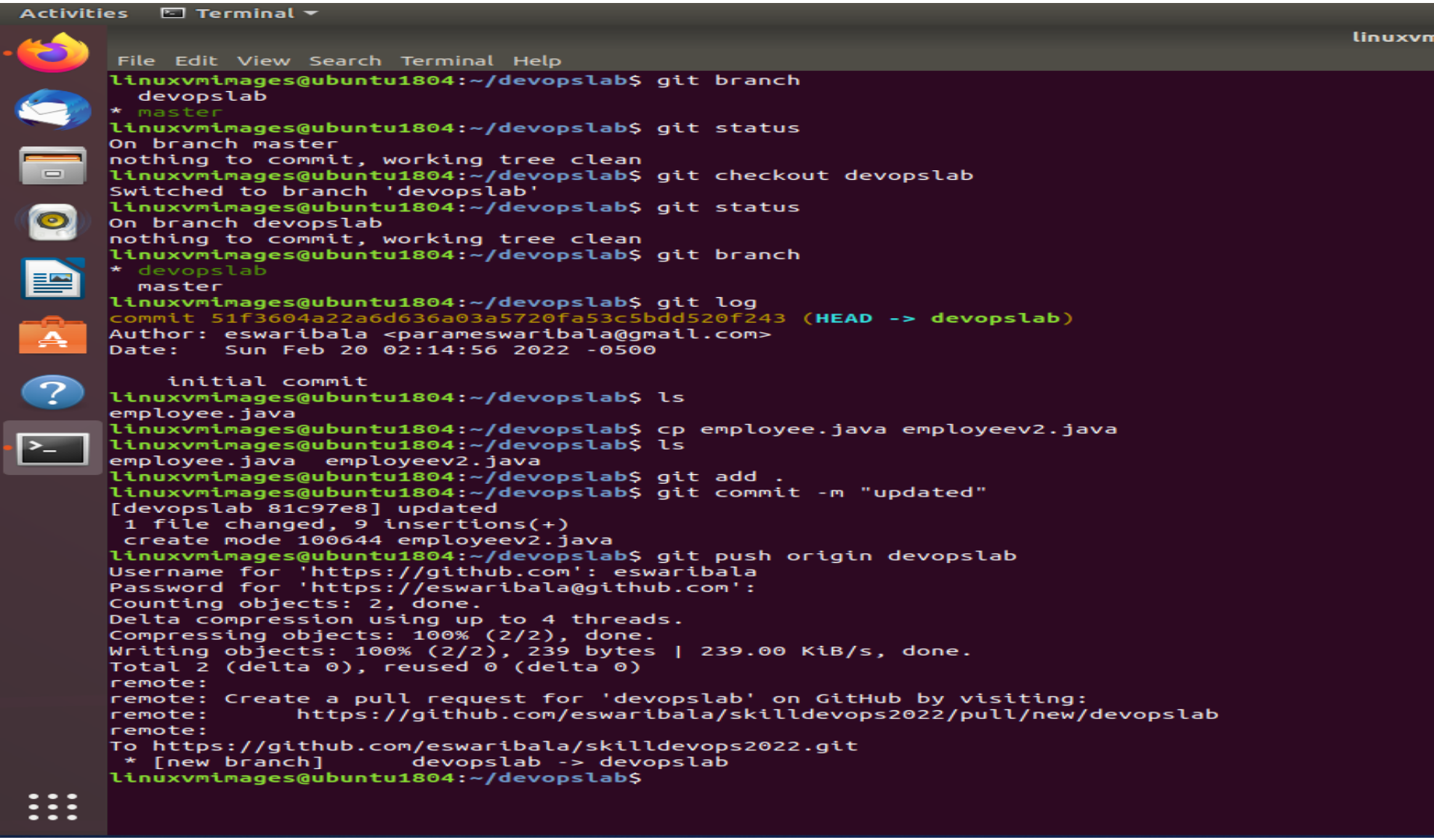



```
git branch <branch_name>
```

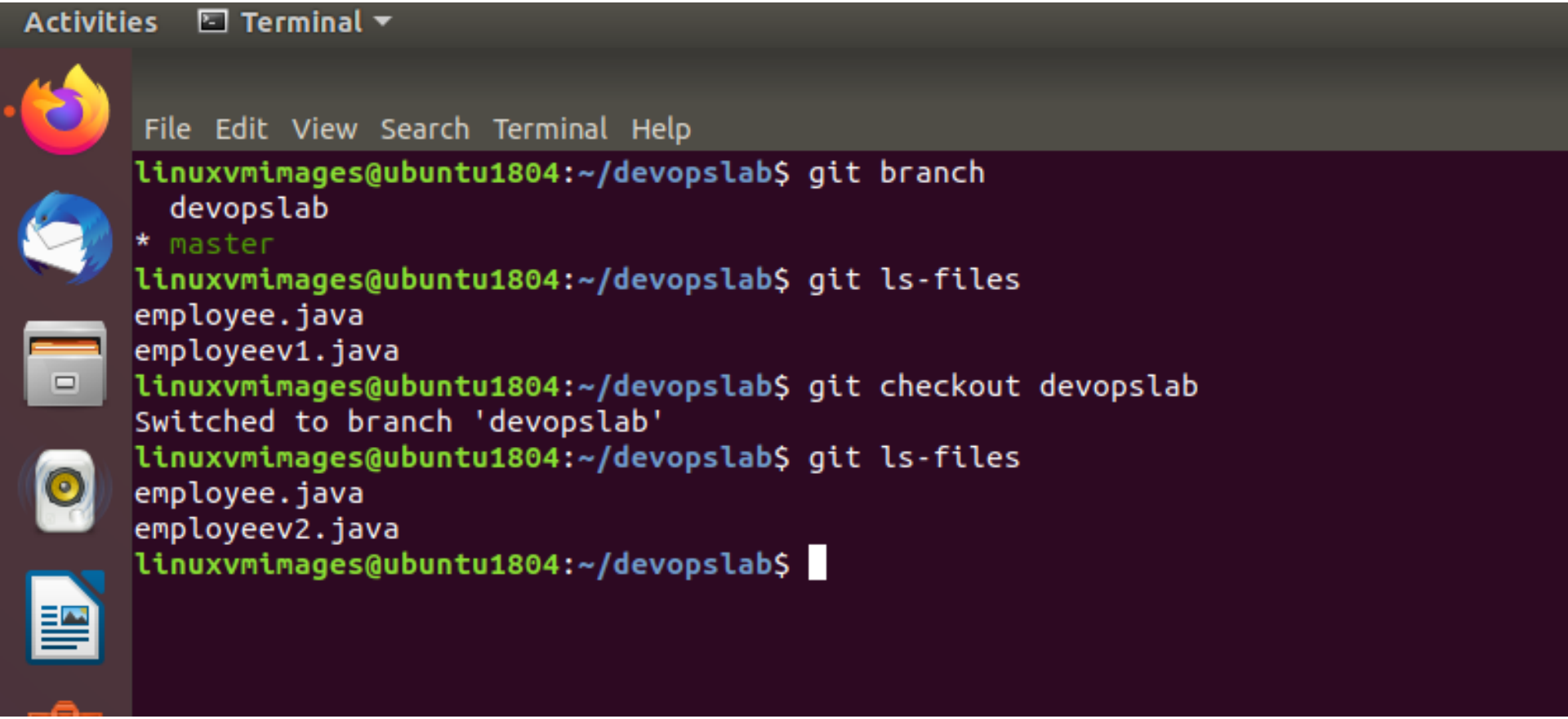
Parallel Development



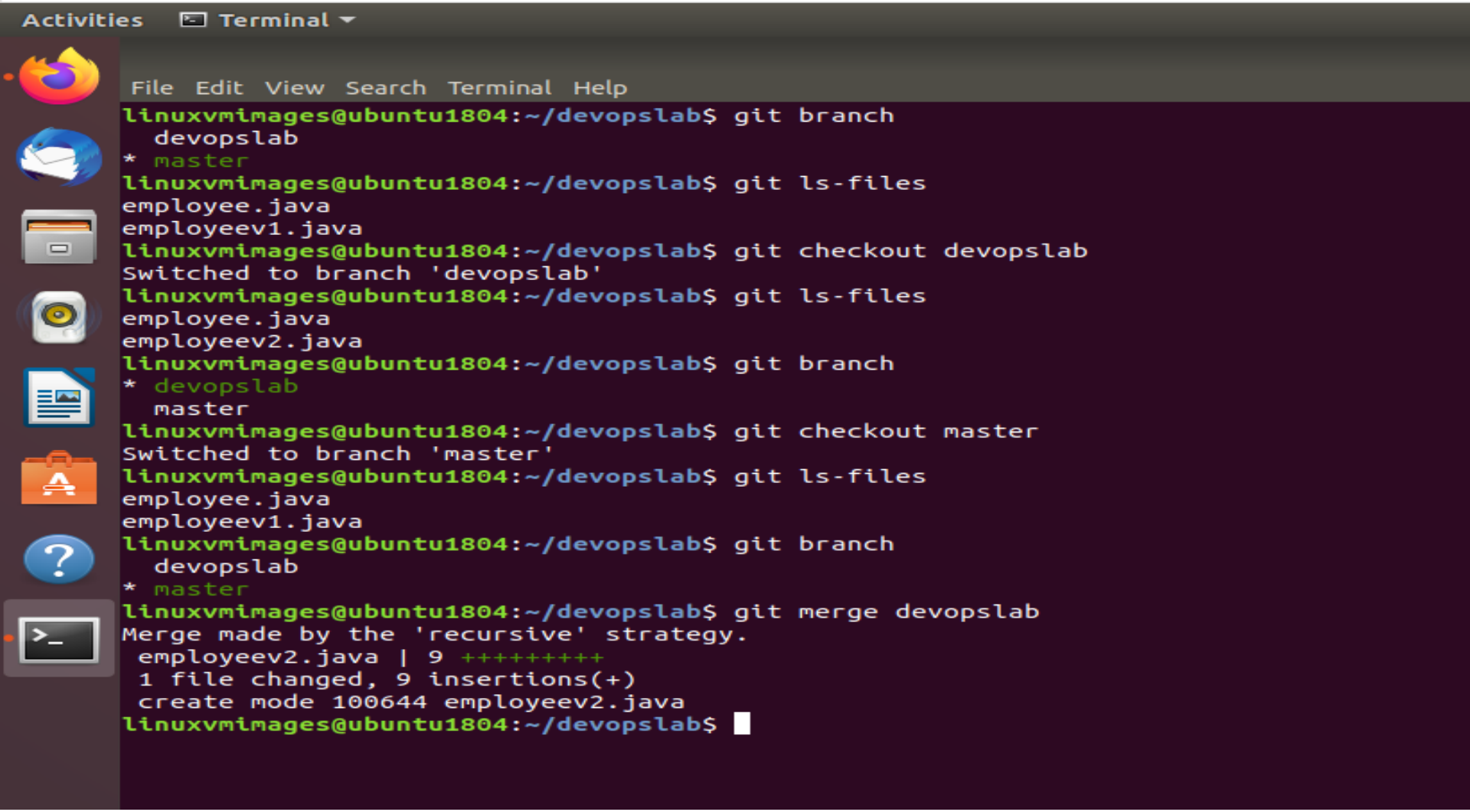
Parallel Development



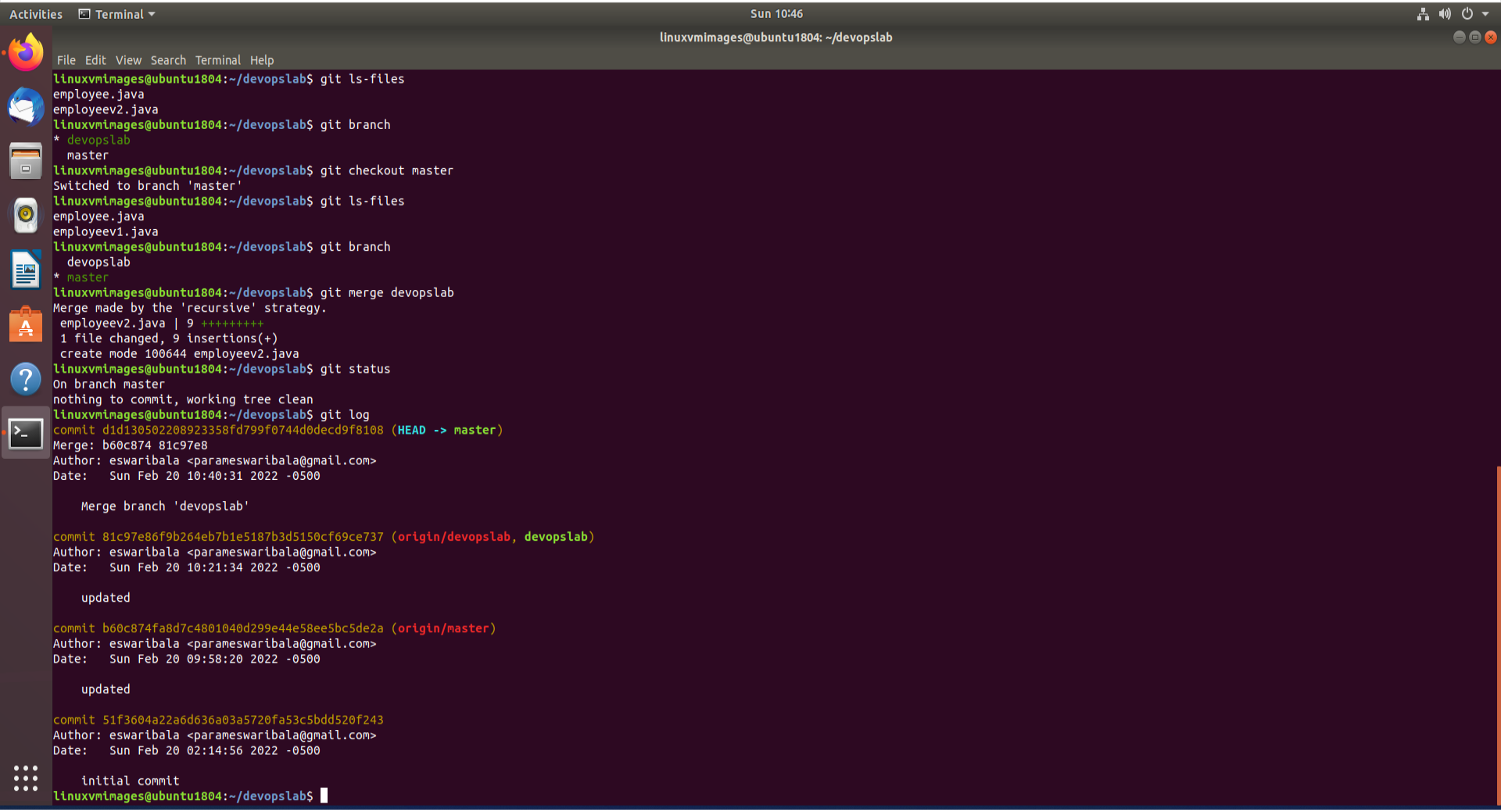
Parallel Development



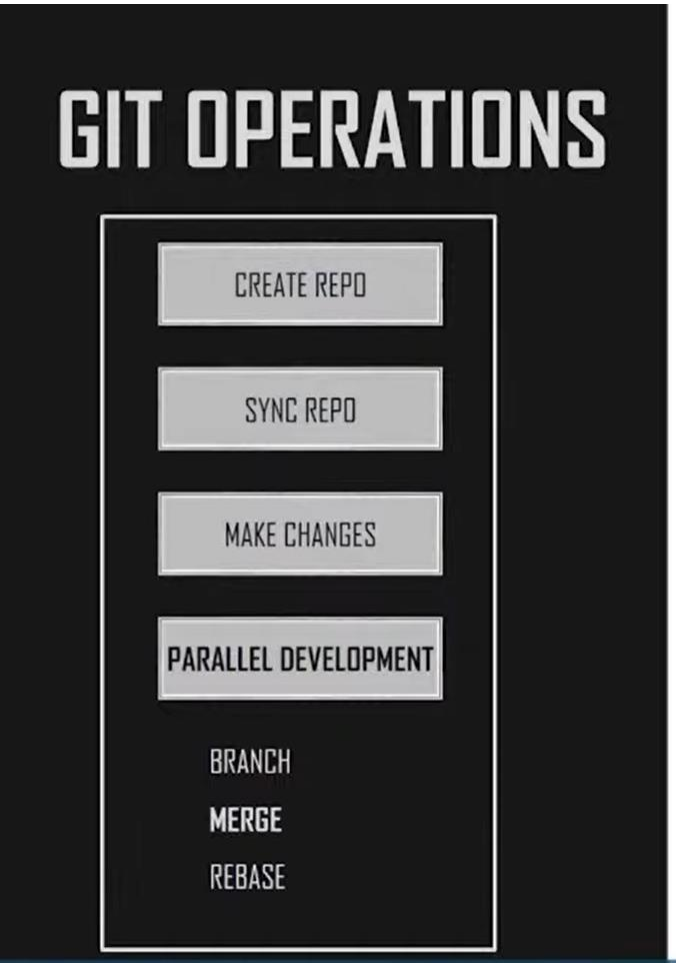
Parallel Development

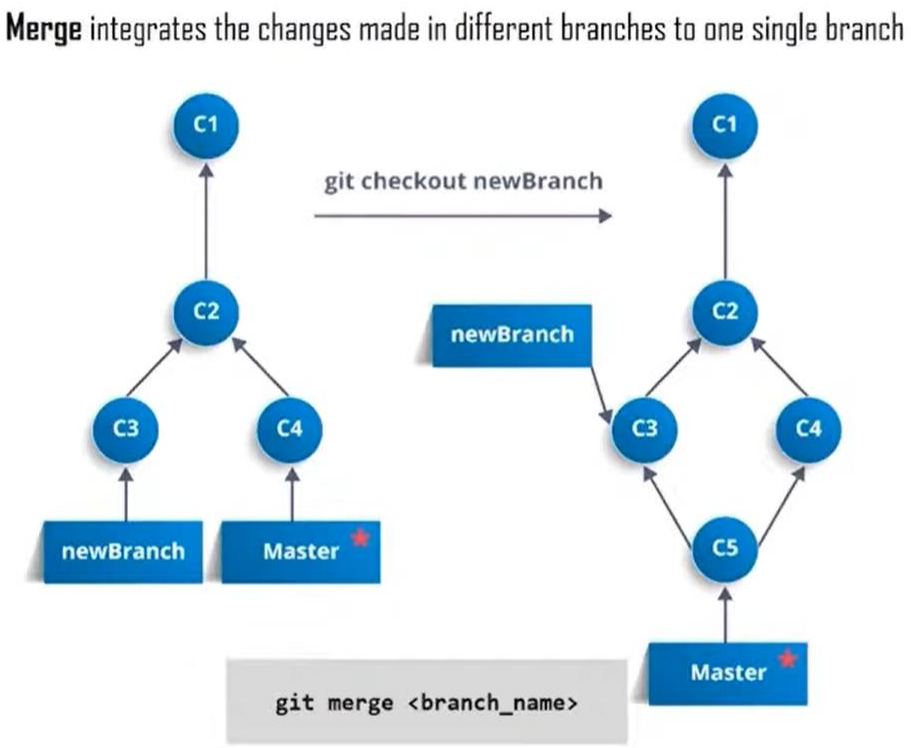


Parallel Development

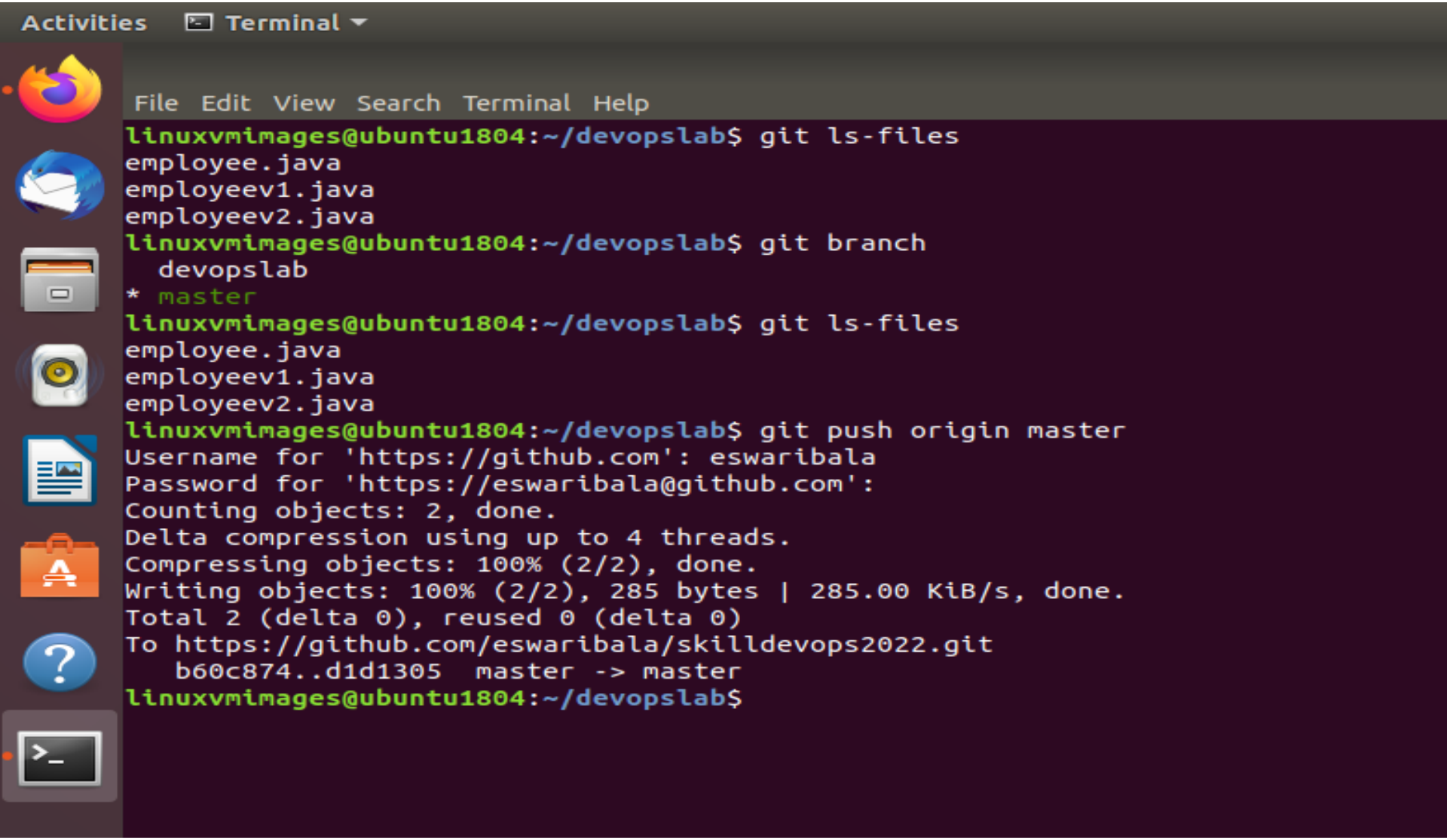


Parallel Development





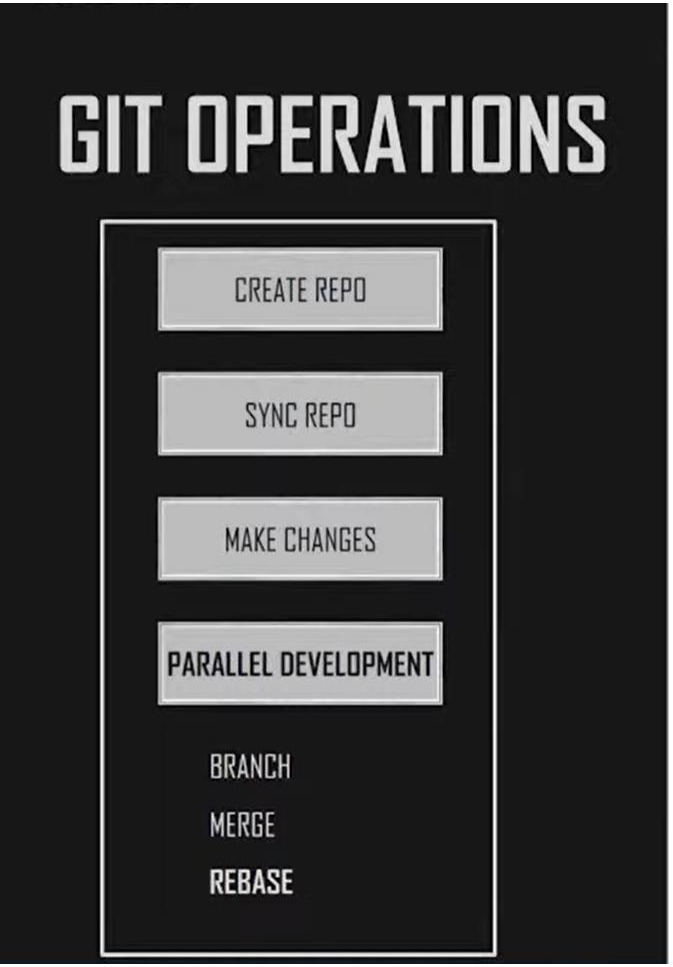
Parallel Development



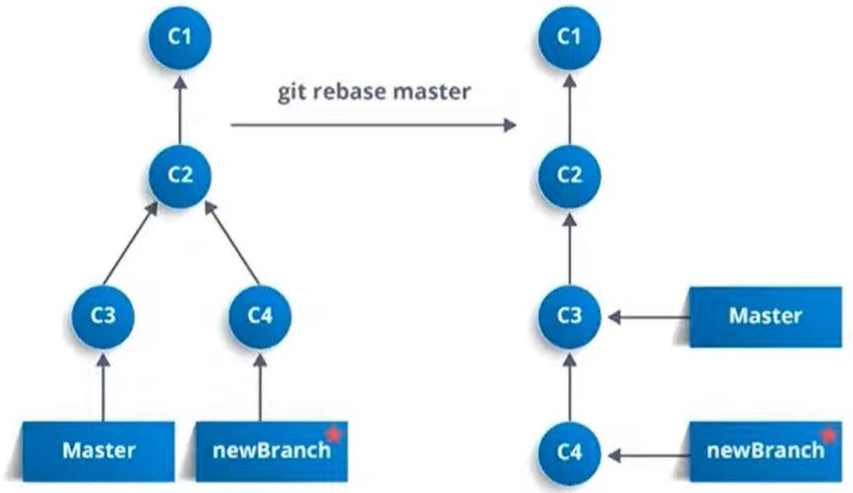
Rebase

- A rebase is what you do when you combine a commit or series of commits to a new commit.
- It is similar to merging in that it moves changes from one branch to another.
- Rebasing allows you to rewrite the history of a Git repository.
- When you run a rebase operation, it merges the entire history of two branches into one.
- This will create brand new commits for each commit in another branch inside the branch you are rebasing.
- Rebasing is just like changing the base of a branch from one commit to another.
- This will change your repository’s history to make it look like you created a branch from another commit.

Rebase

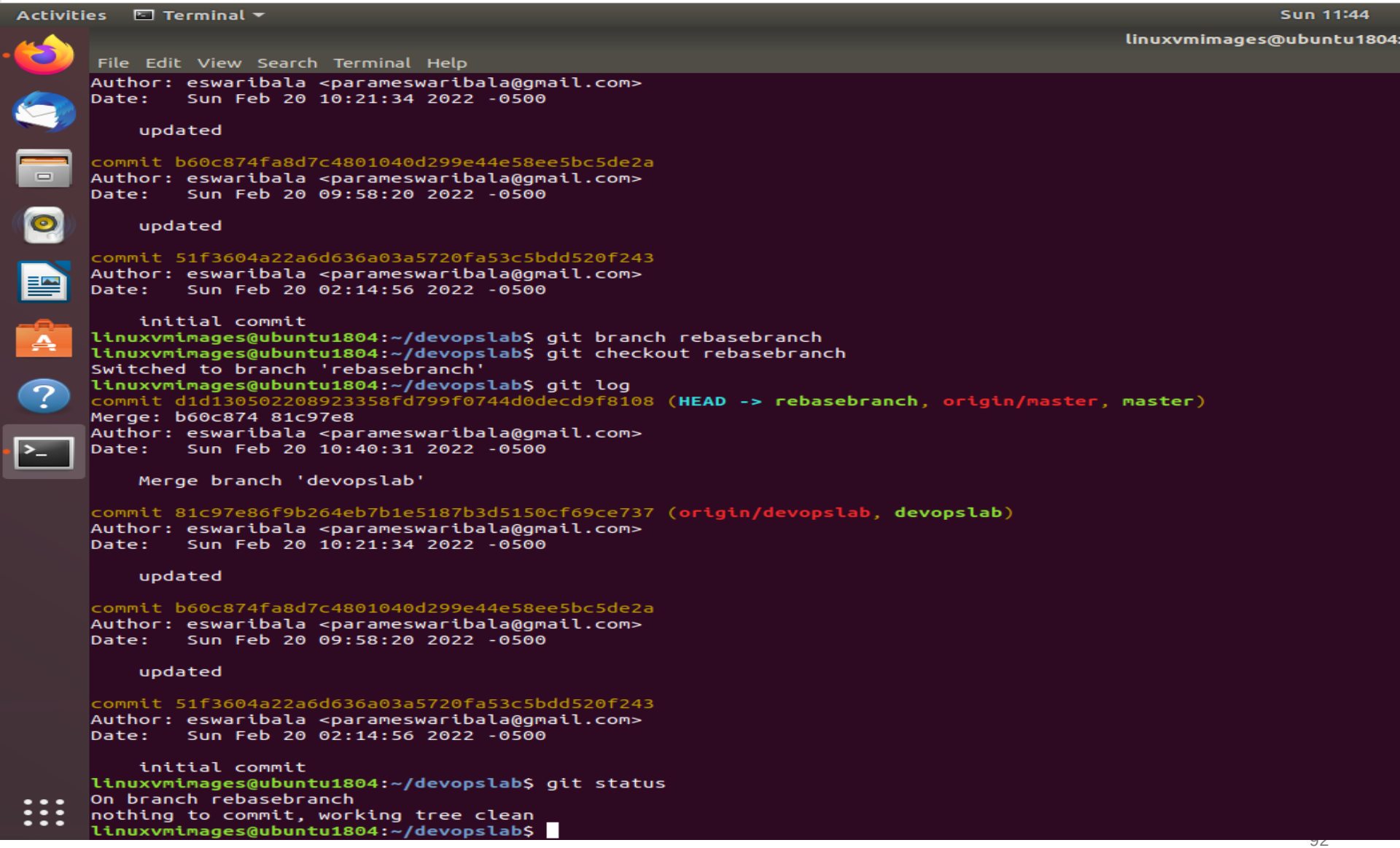


Rebase is used when changes made in one branch needs to be reflected in another branch

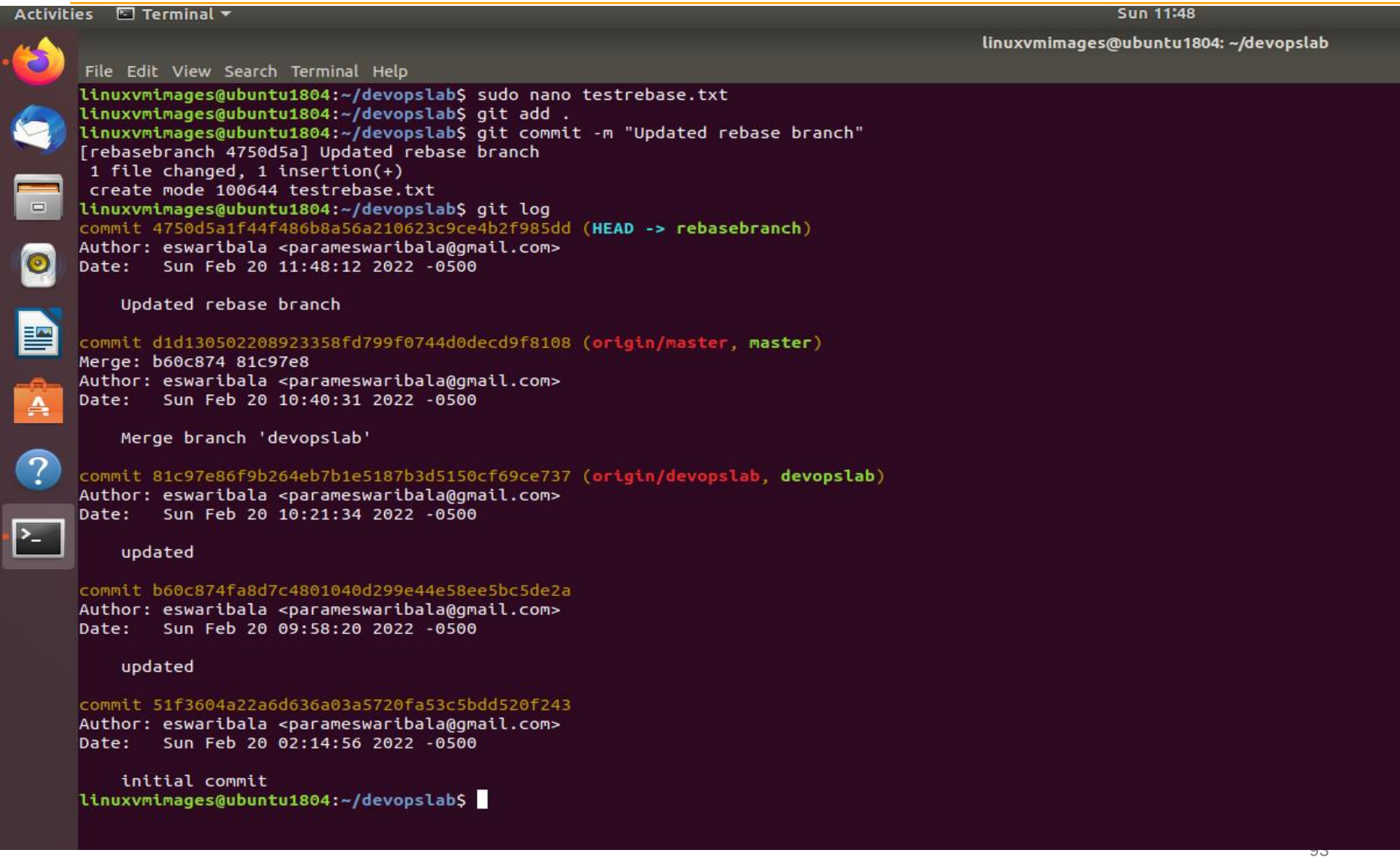



```
git rebase master
```

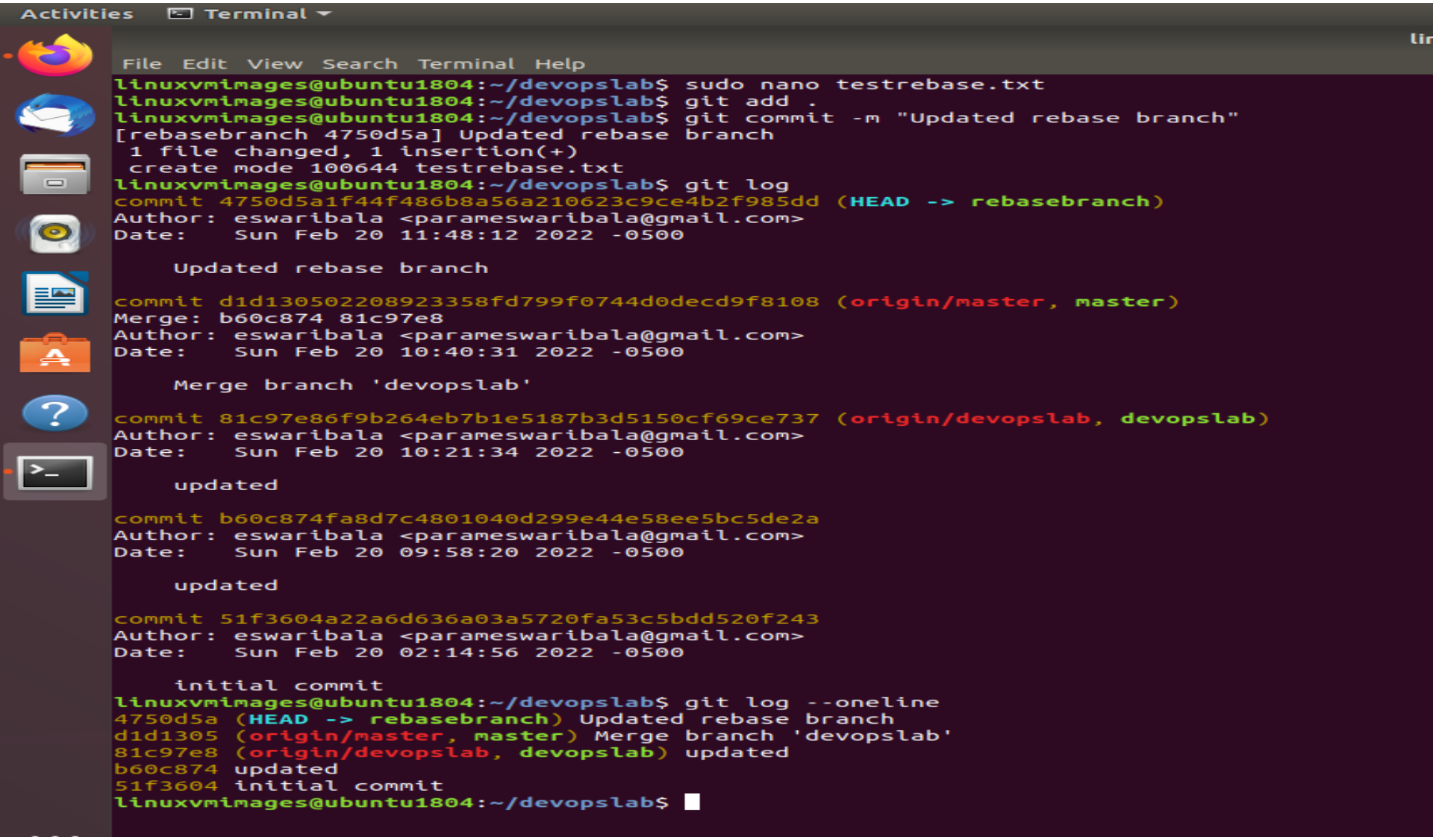
Rebase


Activities
Terminal
Sun 11:44
linuxvmimages@ubuntu1804:
File Edit View Search Terminal Help
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:21:34 2022 -0500
updated
commit b60c874fa8d7c4801040d299e44e58ee5bc5de2a
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 09:58:20 2022 -0500
updated
commit 51f3604a22a6d636a03a5720fa53c5bdd520f243
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 02:14:56 2022 -0500
initial commit
linuxvmimages@ubuntu1804:~/devopslab$ git branch rebasebranch
linuxvmimages@ubuntu1804:~/devopslab$ git checkout rebasebranch
Switched to branch 'rebasebranch'
linuxvmimages@ubuntu1804:~/devopslab$ git log
commit d1d130502208923358fd799f0744d0decd9f8108 (HEAD -> rebasebranch, origin/master, master)
Merge: b60c874 81c97e8
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:40:31 2022 -0500
Merge branch 'devopslab'
commit 81c97e86f9b264eb7b1e5187b3d5150cf69ce737 (origin/devopslab, devopslab)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:21:34 2022 -0500
updated
commit b60c874fa8d7c4801040d299e44e58ee5bc5de2a
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 09:58:20 2022 -0500
updated
commit 51f3604a22a6d636a03a5720fa53c5bdd520f243
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 02:14:56 2022 -0500
initial commit
linuxvmimages@ubuntu1804:~/devopslab$ git status
On branch rebasebranch
nothing to commit, working tree clean
linuxvmimages@ubuntu1804:~/devopslab$

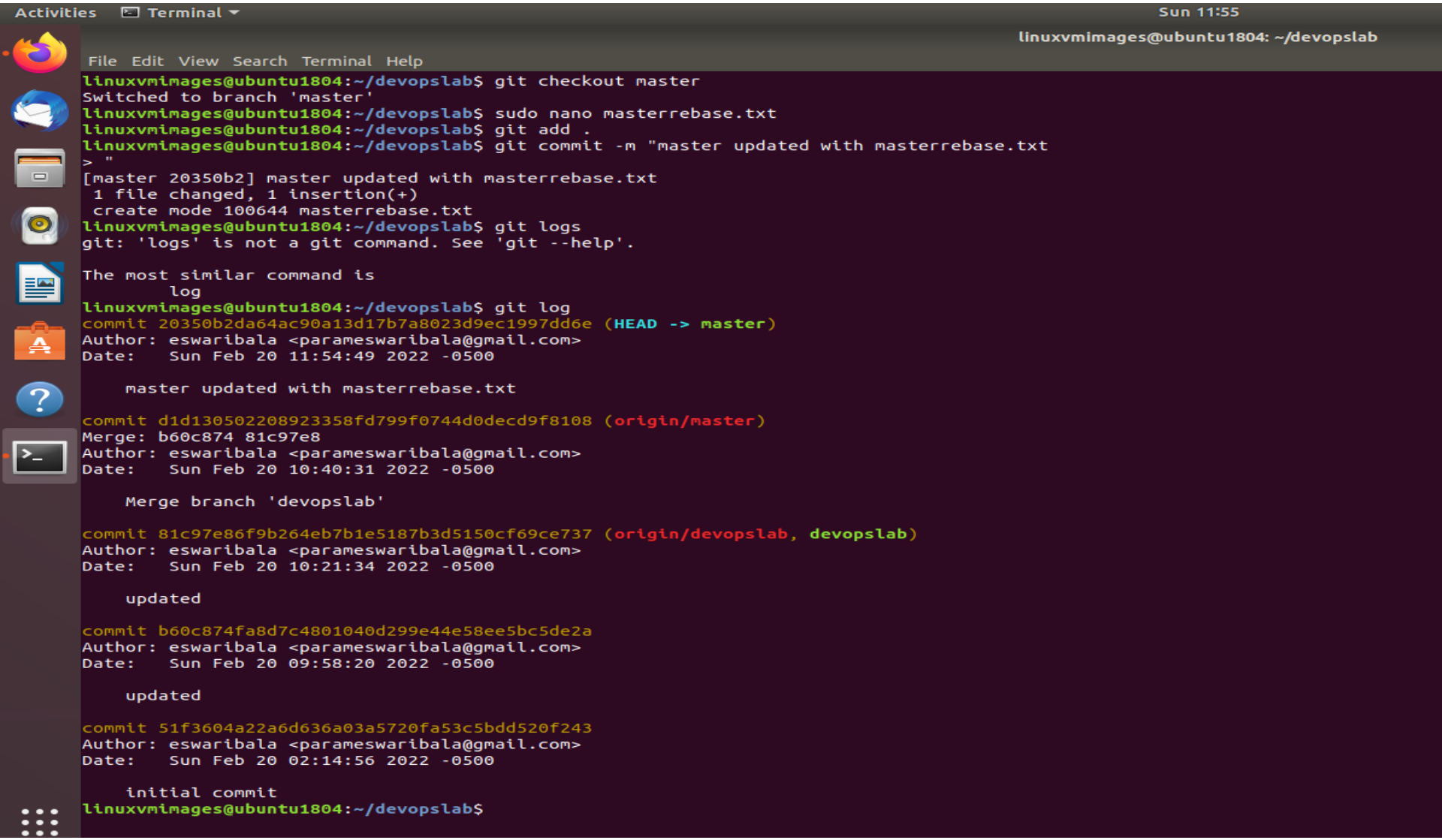
Rebase


Activities
Terminal
Sun 11:48
linuxvmimages@ubuntu1804: ~/devopslab
File Edit View Search Terminal Help
linuxvmimages@ubuntu1804:~/devopslab$ sudo nano testrebase.txt
linuxvmimages@ubuntu1804:~/devopslab$ git add .
linuxvmimages@ubuntu1804:~/devopslab$ git commit -m "Updated rebase branch"
[rebasebranch 4750d5a] Updated rebase branch
1 file changed, 1 insertion(+)
create mode 100644 testrebase.txt
linuxvmimages@ubuntu1804:~/devopslab$ git log
commit 4750d5a1f44f486b8a56a210623c9ce4b2f985dd (HEAD -> rebasebranch)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 11:48:12 2022 -0500
Updated rebase branch
commit d1d130502208923358fd799f0744d0decd9f8108 (origin/master, master)
Merge: b60c874 81c97e8
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:40:31 2022 -0500
Merge branch 'devopslab'
commit 81c97e86f9b264eb7b1e5187b3d5150cf69ce737 (origin/devopslab, devopslab)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:21:34 2022 -0500
updated
commit b60c874fa8d7c4801040d299e44e58ee5bc5de2a
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 09:58:20 2022 -0500
updated
commit 51f3604a22a6d636a03a5720fa53c5bdd520f243
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 02:14:56 2022 -0500
initial commit
linuxvmimages@ubuntu1804:~/devopslab$

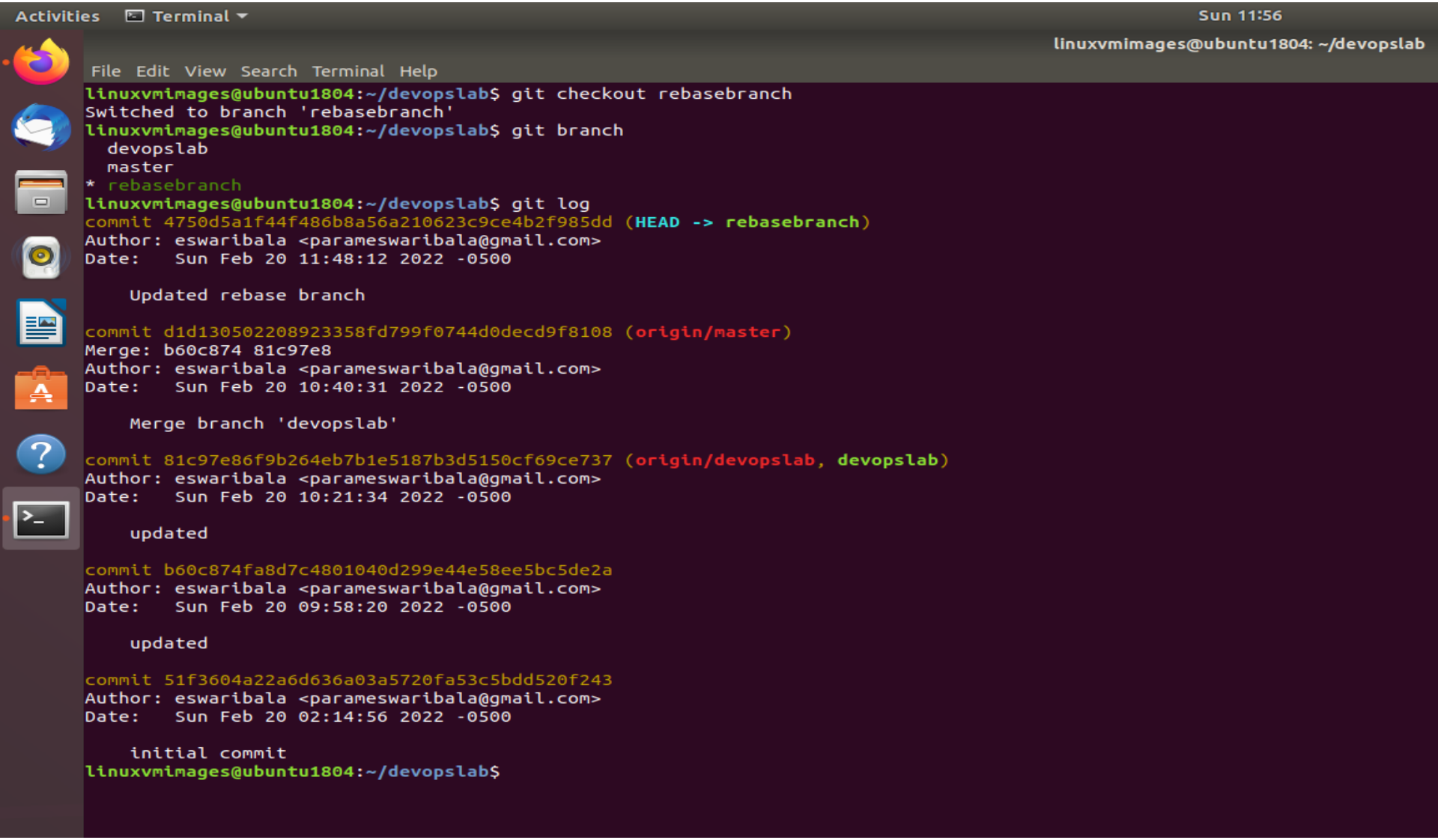
Rebase


Activities
Terminal
File Edit View Search Terminal Help
linuxvmimages@ubuntu1804:~/devopslab$ sudo nano testrebase.txt
linuxvmimages@ubuntu1804:~/devopslab$ git add .
linuxvmimages@ubuntu1804:~/devopslab$ git commit -m "Updated rebase branch"
[rebasebranch 4750d5a] Updated rebase branch
1 file changed, 1 insertion(+)
create mode 100644 testrebase.txt
linuxvmimages@ubuntu1804:~/devopslab$ git log
commit 4750d5a1f44f486b8a56a210623c9ce4b2f985dd (HEAD -> rebasebranch)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 11:48:12 2022 -0500
Updated rebase branch
commit d1d130502208923358fd799f0744d0decd9f8108 (origin/master, master)
Merge: b60c874 81c97e8
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:40:31 2022 -0500
Merge branch 'devopslab'
commit 81c97e86f9b264eb7b1e5187b3d5150cf69ce737 (origin/devopslab, devopslab)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:21:34 2022 -0500
updated
commit b60c874fa8d7c4801040d299e44e58ee5bc5de2a
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 09:58:20 2022 -0500
updated
commit 51f3604a22a6d636a03a5720fa53c5bdd520f243
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 02:14:56 2022 -0500
initial commit
linuxvmimages@ubuntu1804:~/devopslab$ git log --oneline
4750d5a (HEAD -> rebasebranch) Updated rebase branch
d1d1305 (origin/master, master) Merge branch 'devopslab'
81c97e8 (origin/devopslab, devopslab) updated
b60c874 updated
51f3604 initial commit
linuxvmimages@ubuntu1804:~/devopslab$

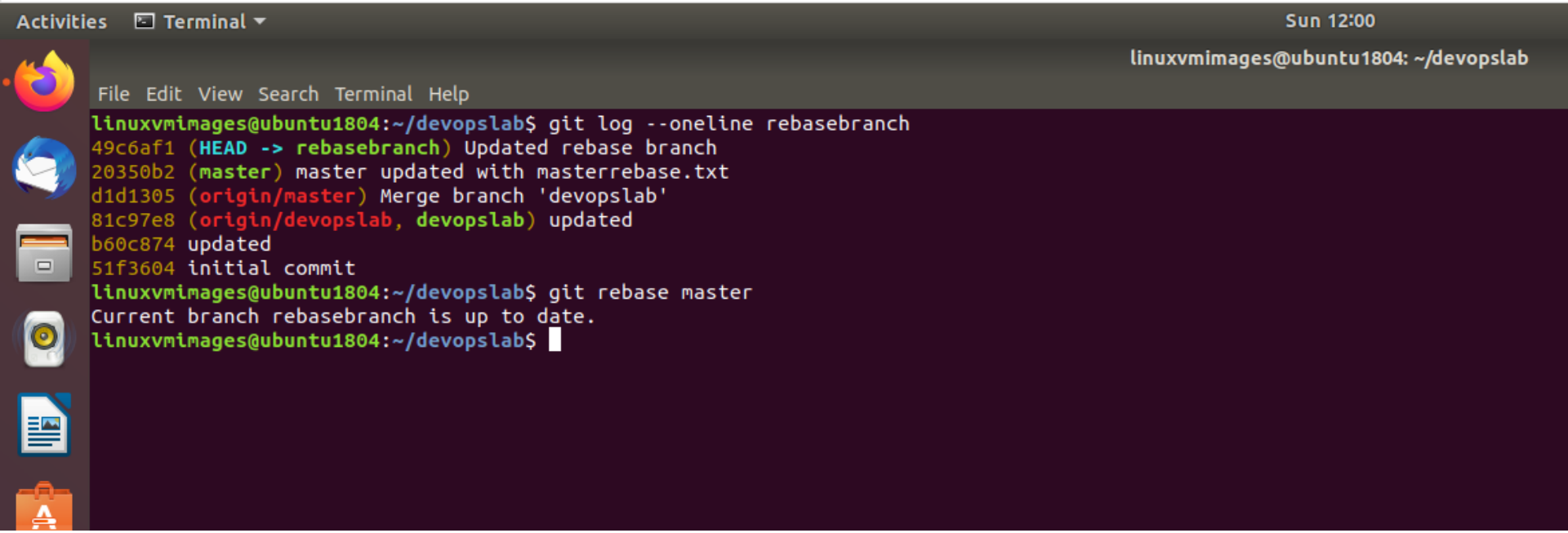
Rebase


Activities
Terminal
Sun 11:55
linuxvmimages@ubuntu1804: ~/devopslab
File Edit View Search Terminal Help
linuxvmimages@ubuntu1804:~/devopslab$ git checkout master
Switched to branch 'master'
linuxvmimages@ubuntu1804:~/devopslab$ sudo nano masterrebase.txt
linuxvmimages@ubuntu1804:~/devopslab$ git add .
linuxvmimages@ubuntu1804:~/devopslab$ git commit -m "master updated with masterrebase.txt
> "
[master 20350b2] master updated with masterrebase.txt
1 file changed, 1 insertion(+)
create mode 100644 masterrebase.txt
linuxvmimages@ubuntu1804:~/devopslab$ git logs
git: 'logs' is not a git command. See 'git --help'.
The most similar command is
log
linuxvmimages@ubuntu1804:~/devopslab$ git log
commit 20350b2da64ac90a13d17b7a8023d9ec1997dd6e (HEAD -> master)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 11:54:49 2022 -0500
master updated with masterrebase.txt
commit d1d130502208923358fd799f0744d0decd9f8108 (origin/master)
Merge: b60c874 81c97e8
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:40:31 2022 -0500
Merge branch 'devopslab'
commit 81c97e86f9b264eb7b1e5187b3d5150cf69ce737 (origin/devopslab, devopslab)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:21:34 2022 -0500
updated
commit b60c874fa8d7c4801040d299e44e58ee5bc5de2a
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 09:58:20 2022 -0500
updated
commit 51f3604a22a6d636a03a5720fa53c5bdd520f243
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 02:14:56 2022 -0500
initial commit
linuxvmimages@ubuntu1804:~/devopslab$

Rebase


Activities
Terminal
Sun 11:56
linuxvmimages@ubuntu1804: ~/devopslab
File Edit View Search Terminal Help
linuxvmimages@ubuntu1804:~/devopslab$ git checkout rebasebranch
Switched to branch 'rebasebranch'
linuxvmimages@ubuntu1804:~/devopslab$ git branch
devopslab
master
* rebasebranch
linuxvmimages@ubuntu1804:~/devopslab$ git log
commit 4750d5a1f44f486b8a56a210623c9ce4b2f985dd (HEAD -> rebasebranch)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 11:48:12 2022 -0500
Updated rebase branch
commit d1d130502208923358fd799f0744d0decd9f8108 (origin/master)
Merge: b60c874 81c97e8
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:40:31 2022 -0500
Merge branch 'devopslab'
commit 81c97e86f9b264eb7b1e5187b3d5150cf69ce737 (origin/devopslab, devopslab)
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 10:21:34 2022 -0500
updated
commit b60c874fa8d7c4801040d299e44e58ee5bc5de2a
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 09:58:20 2022 -0500
updated
commit 51f3604a22a6d636a03a5720fa53c5bdd520f243
Author: eswaribala <parameswaribala@gmail.com>
Date: Sun Feb 20 02:14:56 2022 -0500
initial commit
linuxvmimages@ubuntu1804:~/devopslab$

Rebase


Activities
Terminal
Sun 12:00
linuxvmimages@ubuntu1804: ~/devopslab
File Edit View Search Terminal Help
linuxvmimages@ubuntu1804:~/devopslab$ git log --oneline rebasebranch
49c6af1 (HEAD -> rebasebranch) Updated rebase branch
20350b2 (master) master updated with masterrebase.txt
d1d1305 (origin/master) Merge branch 'devopslab'
81c97e8 (origin/devopslab, devopslab) updated
b60c874 updated
51f3604 initial commit
linuxvmimages@ubuntu1804:~/devopslab$ git rebase master
Current branch rebasebranch is up to date.
linuxvmimages@ubuntu1804:~/devopslab$

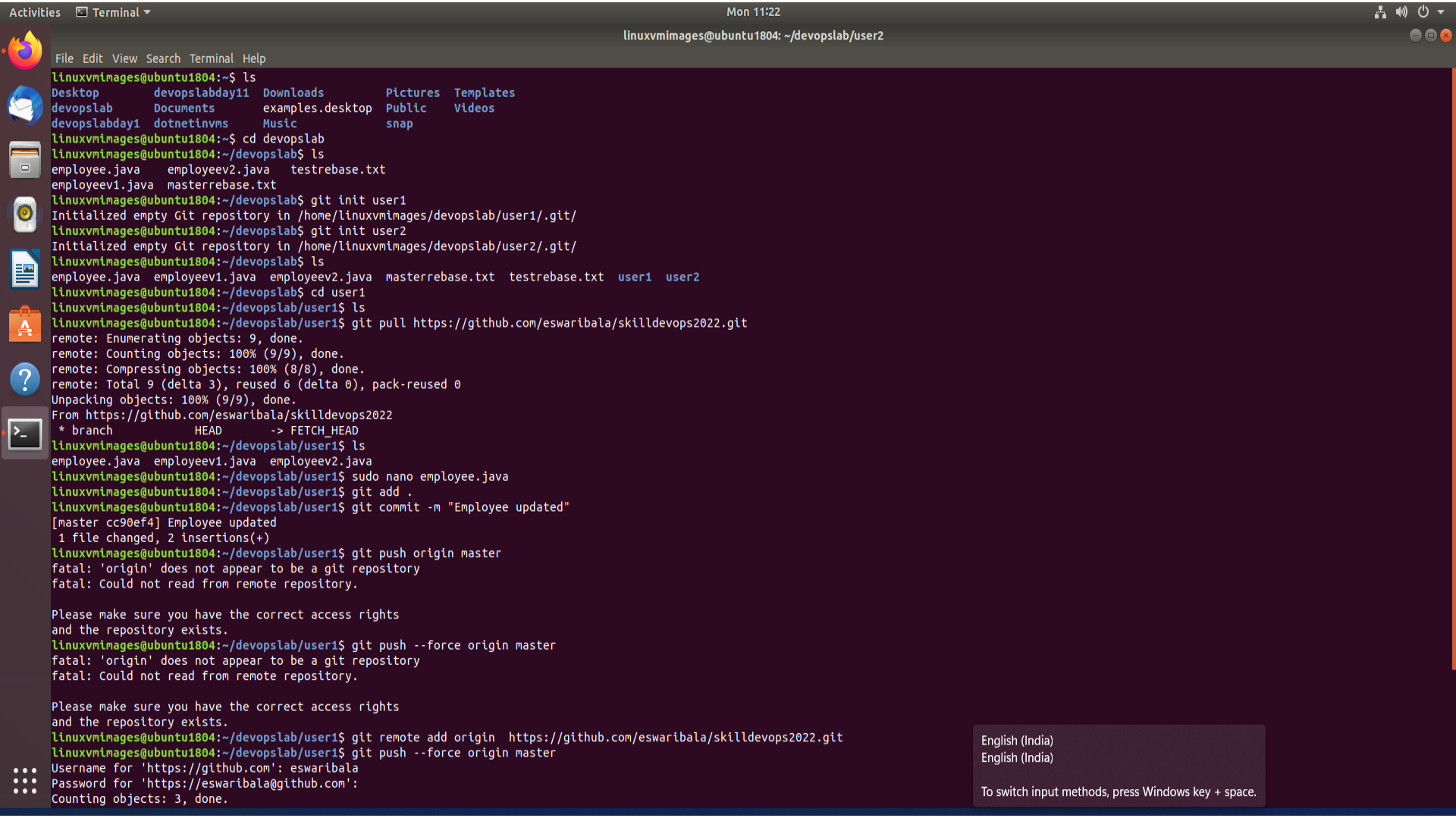
Git Rebase vs. Git Merge

- Both git rebase and git merge have their ideal use cases.
- The git merge command is best used if you want to merge a branch into another branch.
- The merge command creates a merge commit which ties together the histories of the projects.

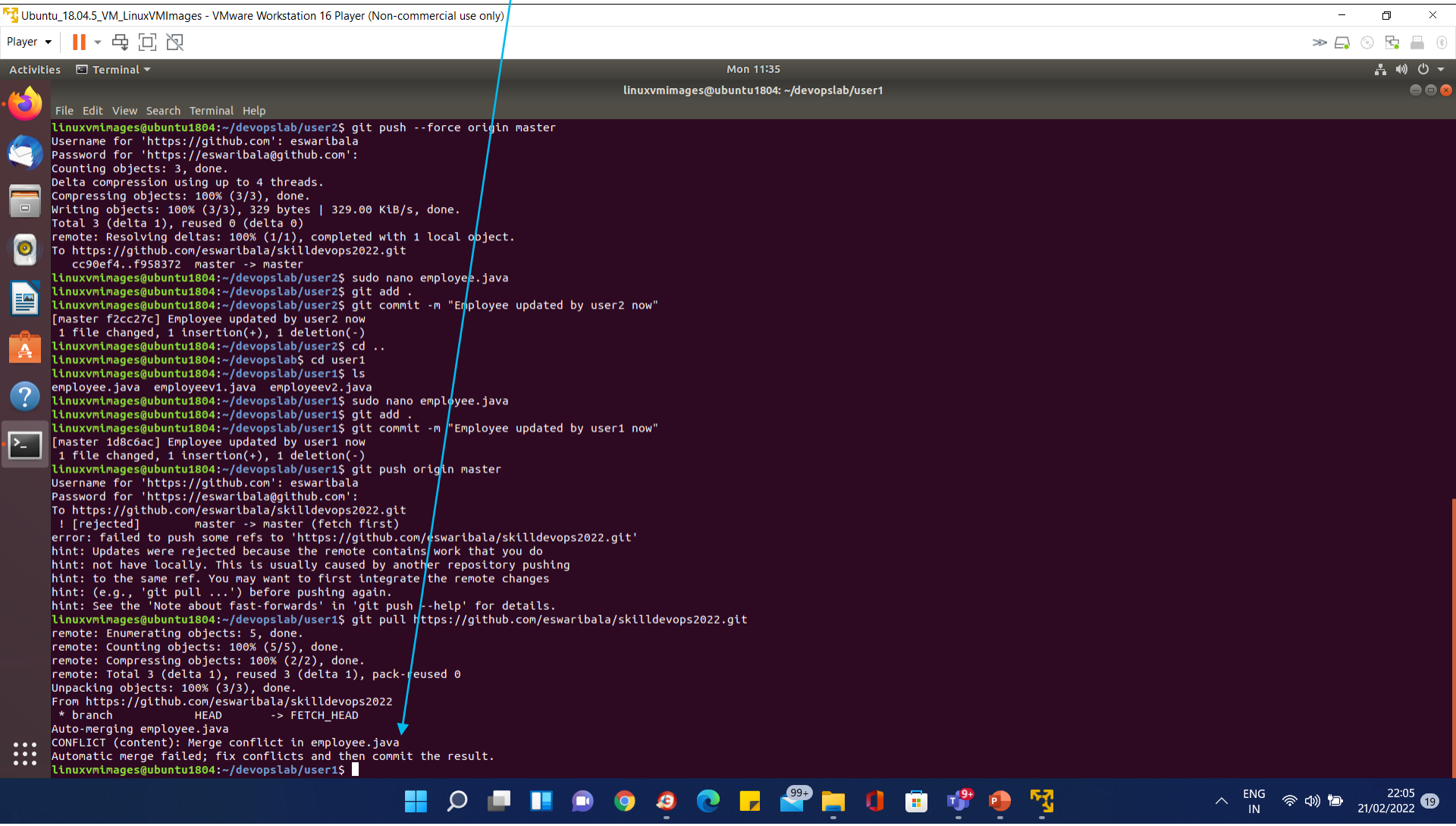
Git Rebase vs. Git Merge

- The rebase command is best used if you need to rewrite the history of a project.
- Rebase will create new commits for each commit in the master branch.
- This will ensure that the final version of your repository contains all the history from both branches.
- You should use the git merge command if you are making changes to a public repository.
- This is because the git merge command does not rewrite history.
- It maintains a record that is forward-looking.

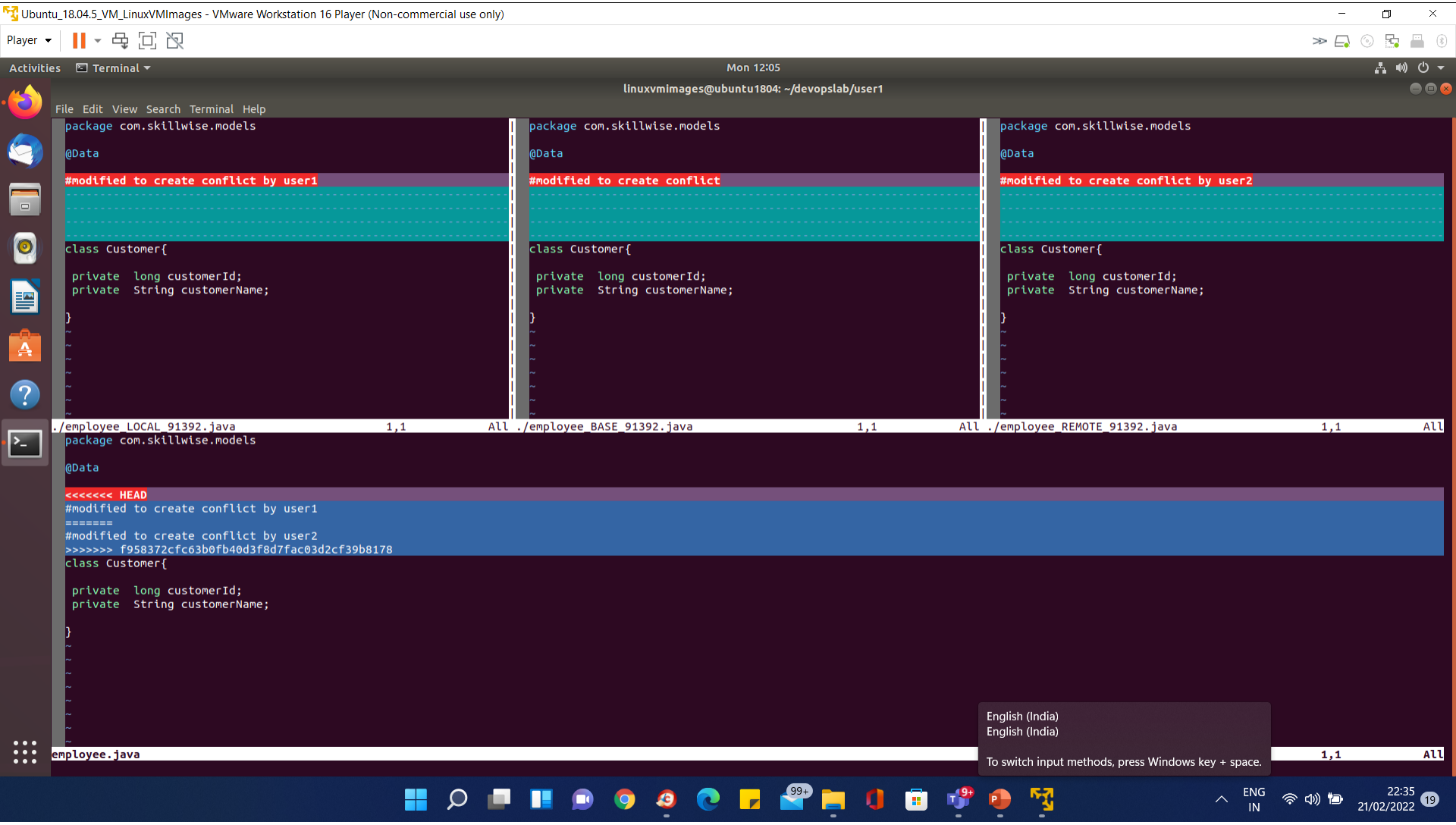
GIT Merge Conflicts



GIT Merge Conflicts



GIT Merge Conflicts



GIT Additional Commands

A few bonus commands you should keep in mind :

Archive your Repository

```
git archive master --format=zip  -
output=../name_of_file.zip
```

GIT Additional Commands

A few bonus commands you should keep in mind :

Archive your Repository

Bundle your Repository

```
git bundle create ../repo.bundler master
```

GIT Additional Commands

A few bonus commands you should keep in mind :

Archive your Repository

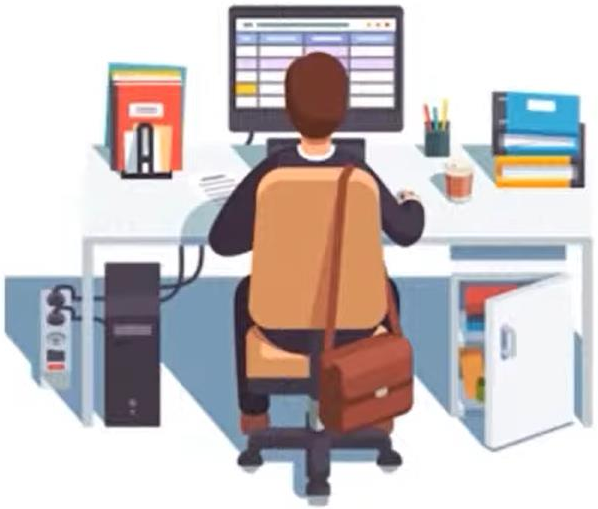
Bundle your Repository

Stash Uncommitted Changes

```
git stash
git stash apply
```

GITHUB

THE NEED FOR GITHUB



- Developers need a web/cloud based code hosting platform
- Useful for version control
- Enables effective collaboration
- Download projects and files in one go
- Easy evaluation of each other's work

GITHUB

THE NEED FOR GITHUB

GitHub

But what makes GitHub so popular?

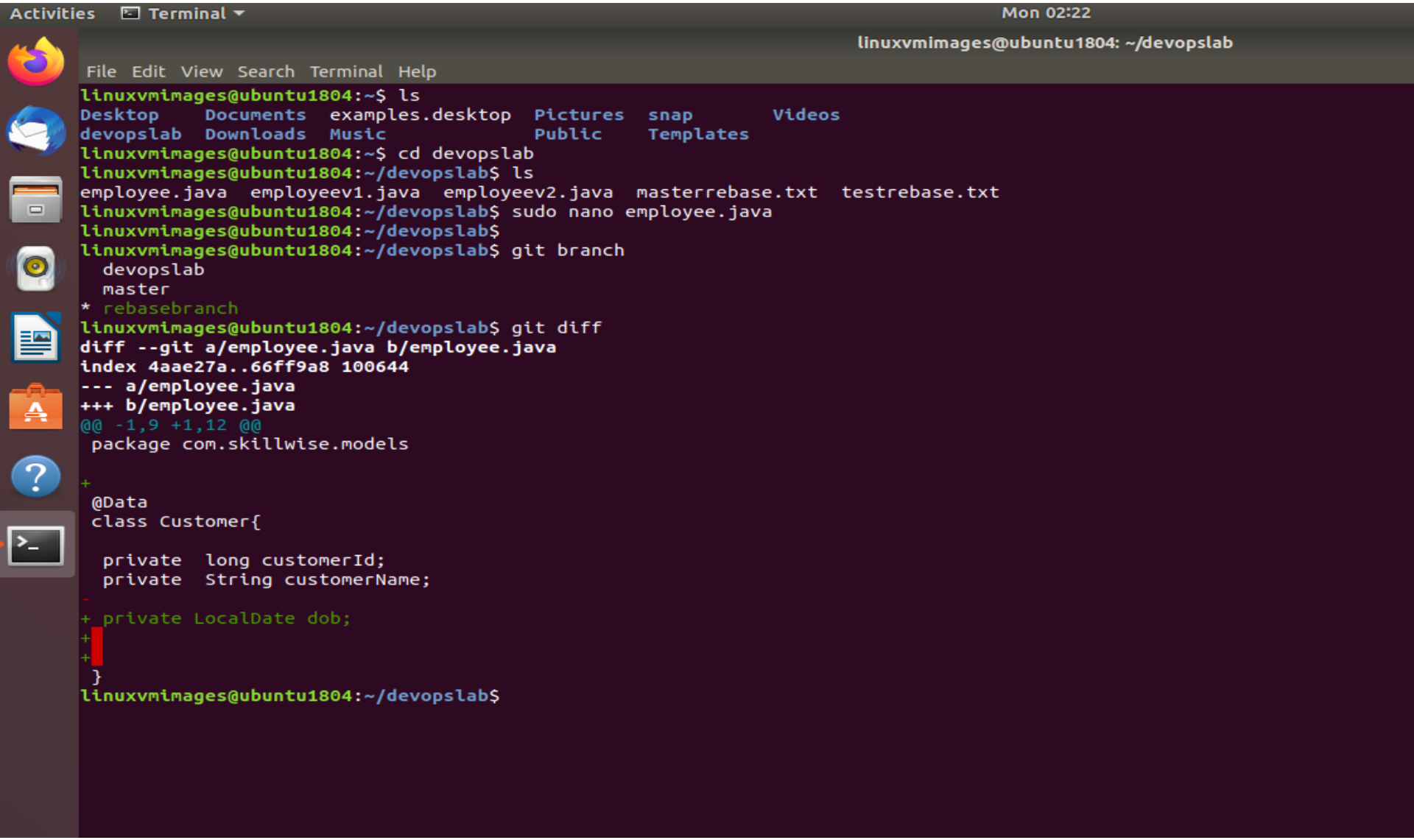
- Immensely powerful community
- The largest shared repository
- Easy version control
- Secure cloud storage

Git Diff

- Git diff is a command-line utility. It's a multiuse Git command.
- When it is executed, it runs a diff function on Git data sources.
- These data sources can be files, branches, commits, and more.
- It is used to show changes between commits, commit, and working tree, etc.
- It compares the different versions of data sources.
- The version control system stands for working with a modified version of files.
- So, the diff command is a useful tool for working with Git.

Git Diff


GIT


Activities
Terminal
Mon 02:22
linuxvmimages@ubuntu1804: ~/devopslab
File Edit View Search Terminal Help
linuxvmimages@ubuntu1804:~$ ls
Desktop Documents examples.desktop Pictures snap Videos
devopslab Downloads Music Public Templates
linuxvmimages@ubuntu1804:~$ cd devopslab
linuxvmimages@ubuntu1804:~/devopslab$ ls
employee.java employeev1.java employeev2.java masterrebase.txt testrebase.txt
linuxvmimages@ubuntu1804:~/devopslab$ sudo nano employee.java
linuxvmimages@ubuntu1804:~/devopslab$
linuxvmimages@ubuntu1804:~/devopslab$ git branch
devopslab
master
* rebasebranch
linuxvmimages@ubuntu1804:~/devopslab$ git diff
diff --git a/employee.java b/employee.java
index 4aae27a..66ff9a8 100644
--- a/employee.java
+++ b/employee.java
@@ -1,9 +1,12 @@
package com.skillwise.models
+
@Data
class Customer{
private long customerId;
private String customerName;
-
+ private LocalDate dob;
+
+
}
linuxvmimages@ubuntu1804:~/devopslab$

Git Diff


```
  master
* rebasebranch
linuxvmimages@ubuntu1804:~/devopslab$ git diff
diff --git a/employee.java b/employee.java
index 4aae27a..66ff9a8 100644
--- a/employee.java
+++ b/employee.java
@@ -1,9 +1,12 @@
 package com.skillwise.models

+
 @Data
 class Customer{

  private  long customerId;
  private  String customerName;
-
+ private LocalDate dob;
+
+
 }
linuxvmimages@ubuntu1804:~/devopslab$ git diff --staged
linuxvmimages@ubuntu1804:~/devopslab$ git diff master devopslab
diff --git a/employeev1.java b/employeev1.java
deleted file mode 100644
index 4aae27a..0000000
--- a/employeev1.java
+++ /dev/null
@@ -1,9 +0,0 @@
-package com.skillwise.models
-
-@Data
-class Customer{
-
- private  long customerId;
- private  String customerName;
-
-}
diff --git a/masterrebase.txt b/masterrebase.txt
deleted file mode 100644
index 2ada78e..0000000
--- a/masterrebase.txt
+++ /dev/null
@@ -1 +0,0 @@
-log "Master rebase test done..."
linuxvmimages@ubuntu1804:~/devopslab$ ^C
linuxvmimages@ubuntu1804:~/devopslab$
```

GIT Rename Branch

- Start by switching to the local branch which you want to rename:
  - git checkout <old_name>
  - Rename the local branch by typing:
  - git branch -m <new_name>
- At this point, you have renamed the local branch.
- If you've already pushed the <old_name> branch to the remote repository , perform the next steps to rename the remote branch.
  - Push the <new_name> local branch and reset the upstream branch:
  - git push origin -u <new_name>
  - Delete the <old_name> remote branch:
  - git push origin --delete <old_name>

Branch and Tag

- In git, you can use tags to keep track of reference points during the development phase.
- So instead of people or accounts, you tag your code.
- This way you can reference it easily. See, not that different.
- To create a new tag execute we use:
- git tag <tagname>

Branch and Tag

- Tag old commit
- git tag -a v1.2 15027957951b64cf874c3557a0f3547bd83b3ff6
- Retag
- git tag -a -f v1.4 15027957951b64cf874c3557a0f3547bd83b3ff6

Branch and Tag

- To checkout a specific git tag meaning to go to that specific code in that point of development we use:
- $ git checkout tags/<tag> -b <branch>

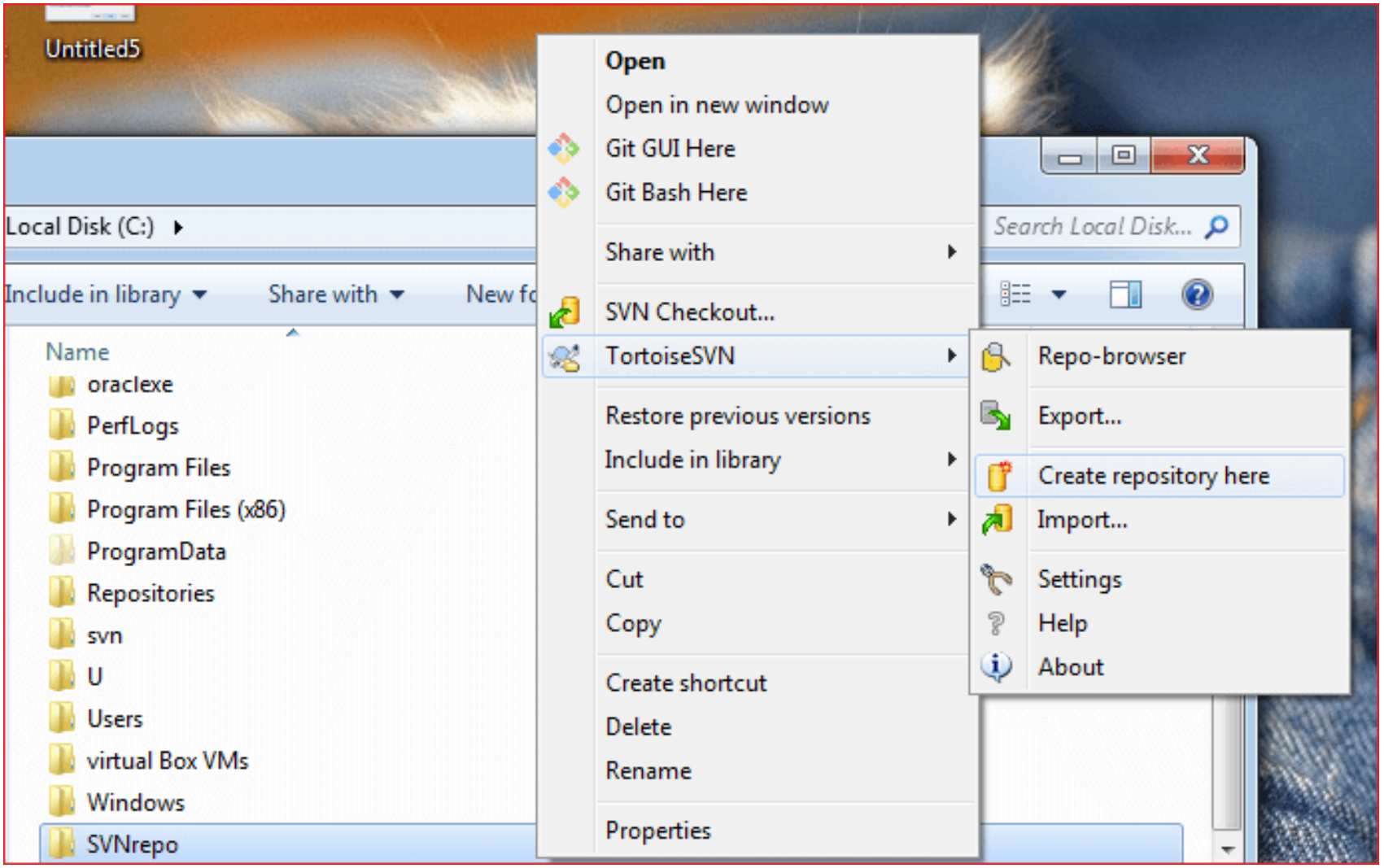
SVN Repository

- SVN repository is a collection of files and directories.
- These files and directories are bundled together in a particular database.
- SVN also records the complete history of all the modifications that have ever been made to these files.
- Generally, the SVN repository can be considered as a folder or directory on our computer.
- These repositories may contain a collection of different or similar types of files.
- An SVN repository typically stores all the files and directories of a single project or maybe a collection of the interrelated projects.

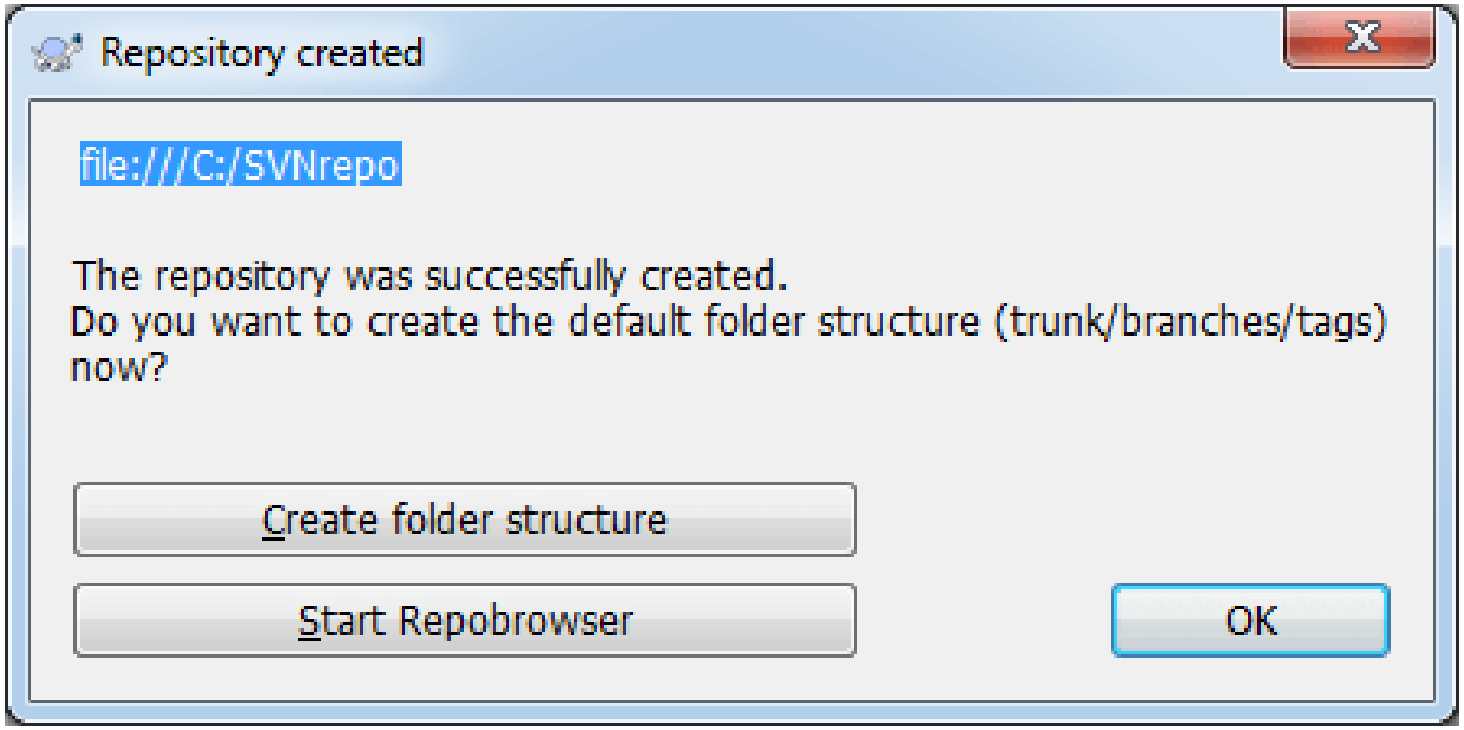
Create a repository using the command line

- To create an SVN repository using the command-line client, follow the below steps:
- Step1: Create an empty folder with the name svn (e.g., C:\svn\). However, we can create it with any name. It is used as the root of all our repositories.
- Step2: Create another folder newrepo inside C:\svn\.
- Step3: Open the command prompt, change directory to D:\svn\newrepo and type the below command:
- svnadmin create --fs-type fsfs newrepo

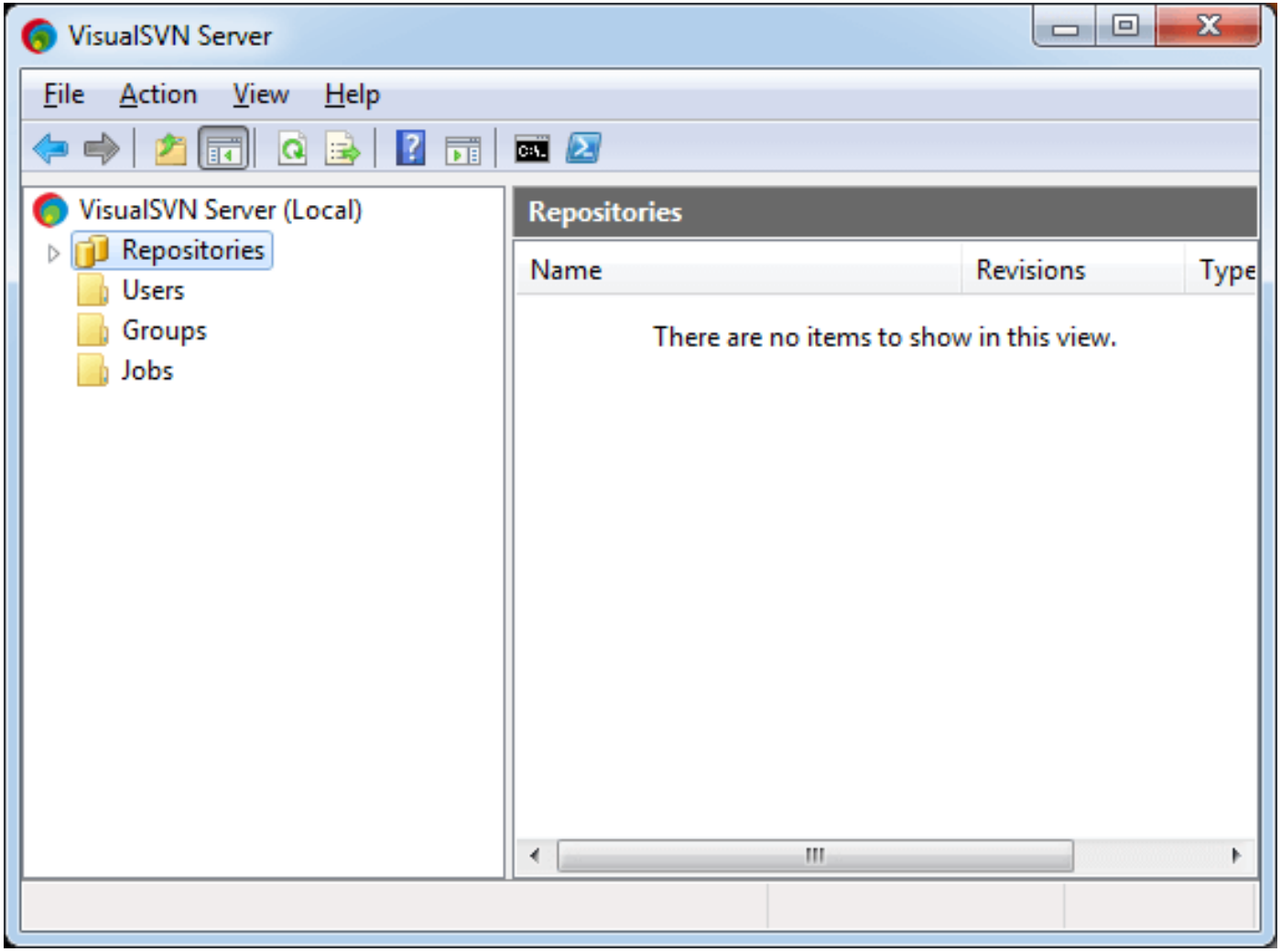
Create a Repository using TortoiseSVN



Create a Repository using TortoiseSVN



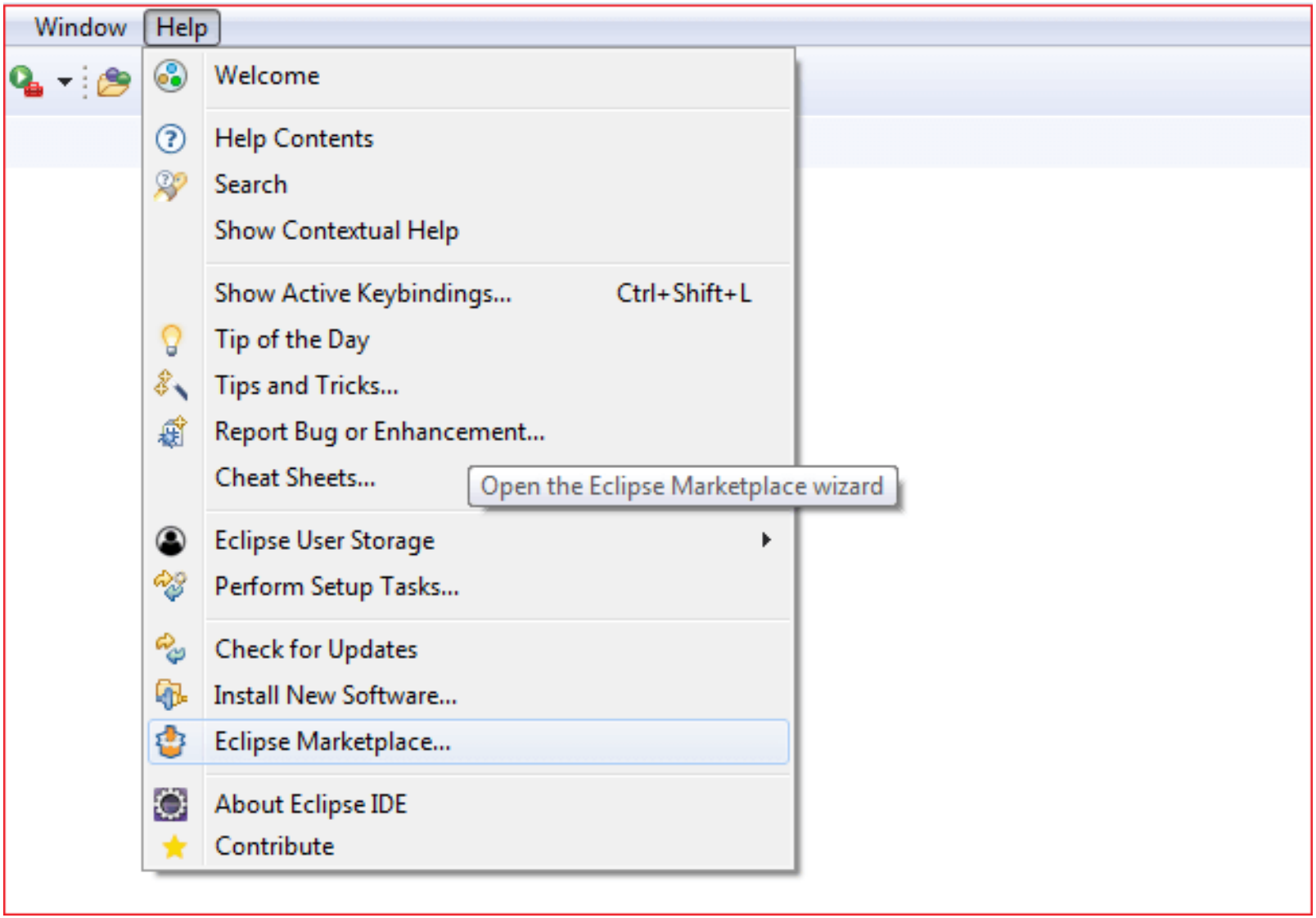
Create a Repository using TortoiseSVN



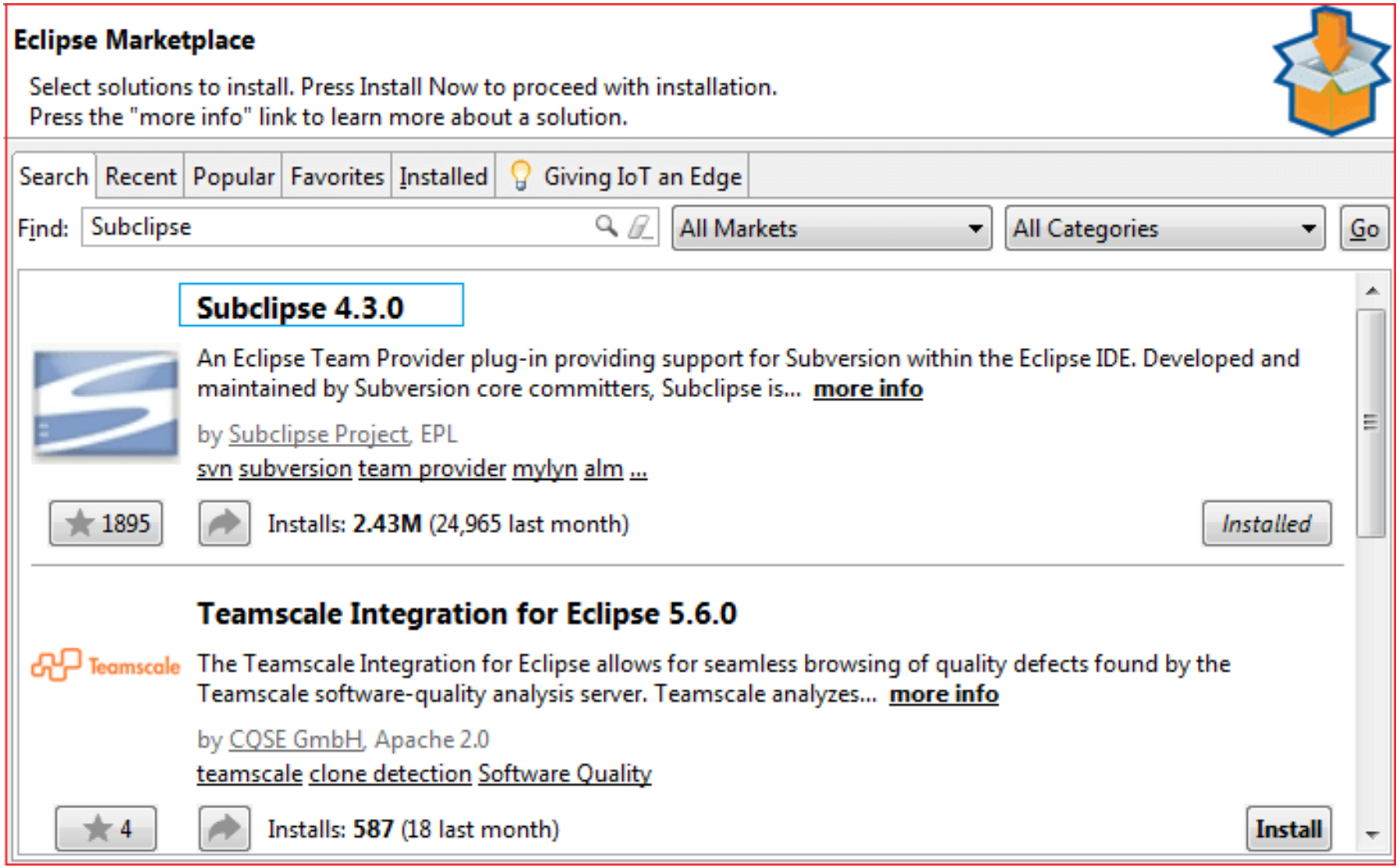
SVN Commands

- SVN Checkout Command
- SVN Add Command
- SVN Delete Command
- SVN Commit Command
- SVN Diff Command
- SVN Status Command
- SVN Log Command
- SVN Move Command
- SVN Rename Command
- SVN List Command
- SVN Update Command
- SVN Info Command
- SVN Merge Command

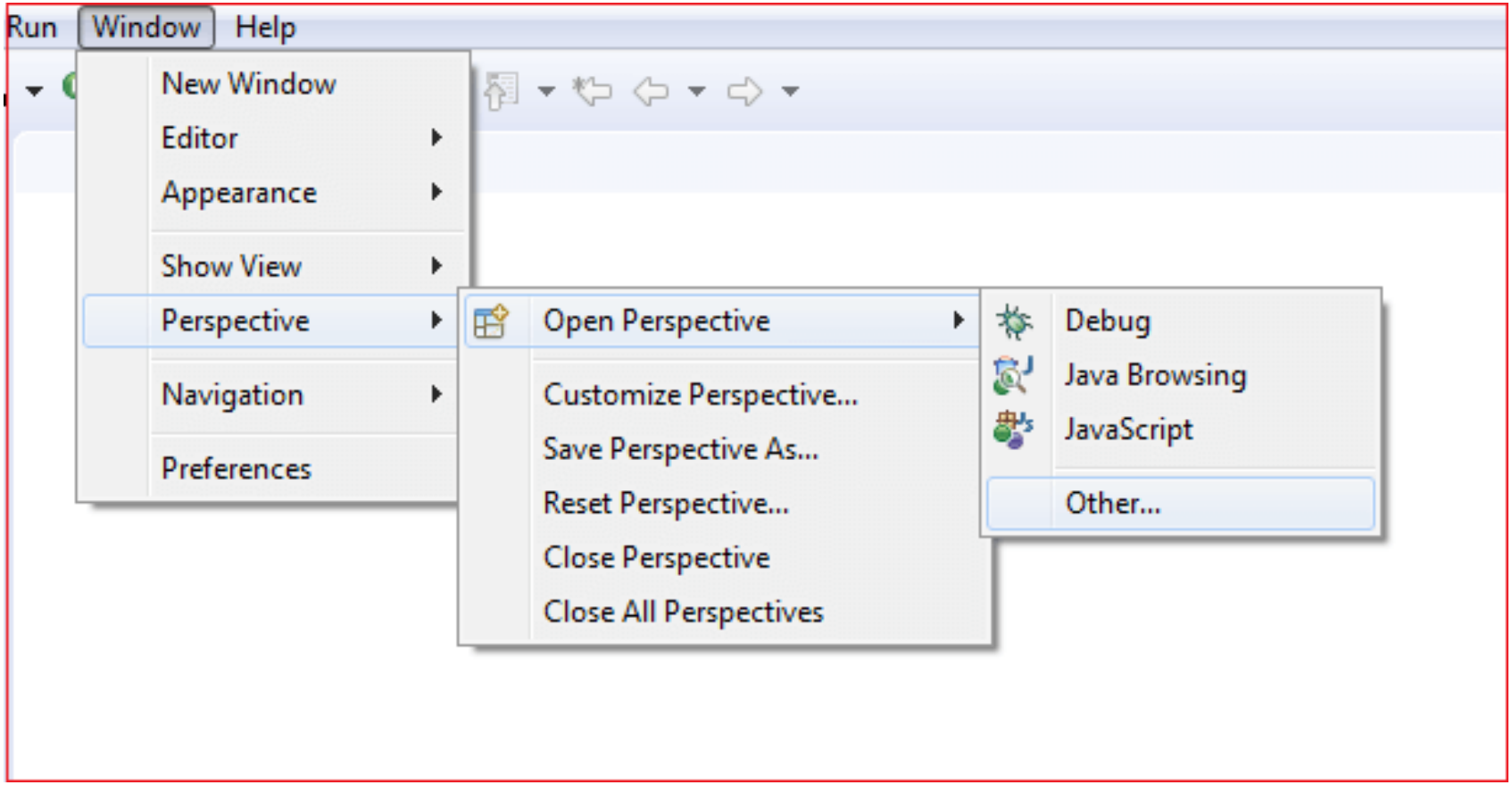
SVN for Eclipse



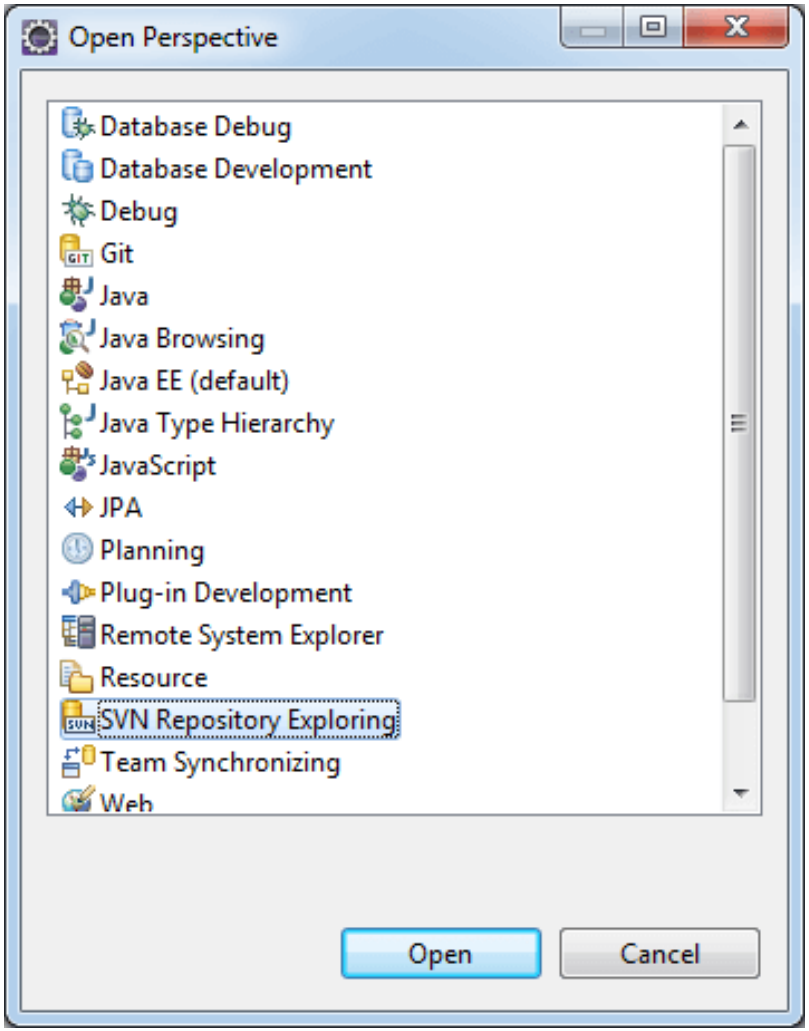
SVN for Eclipse

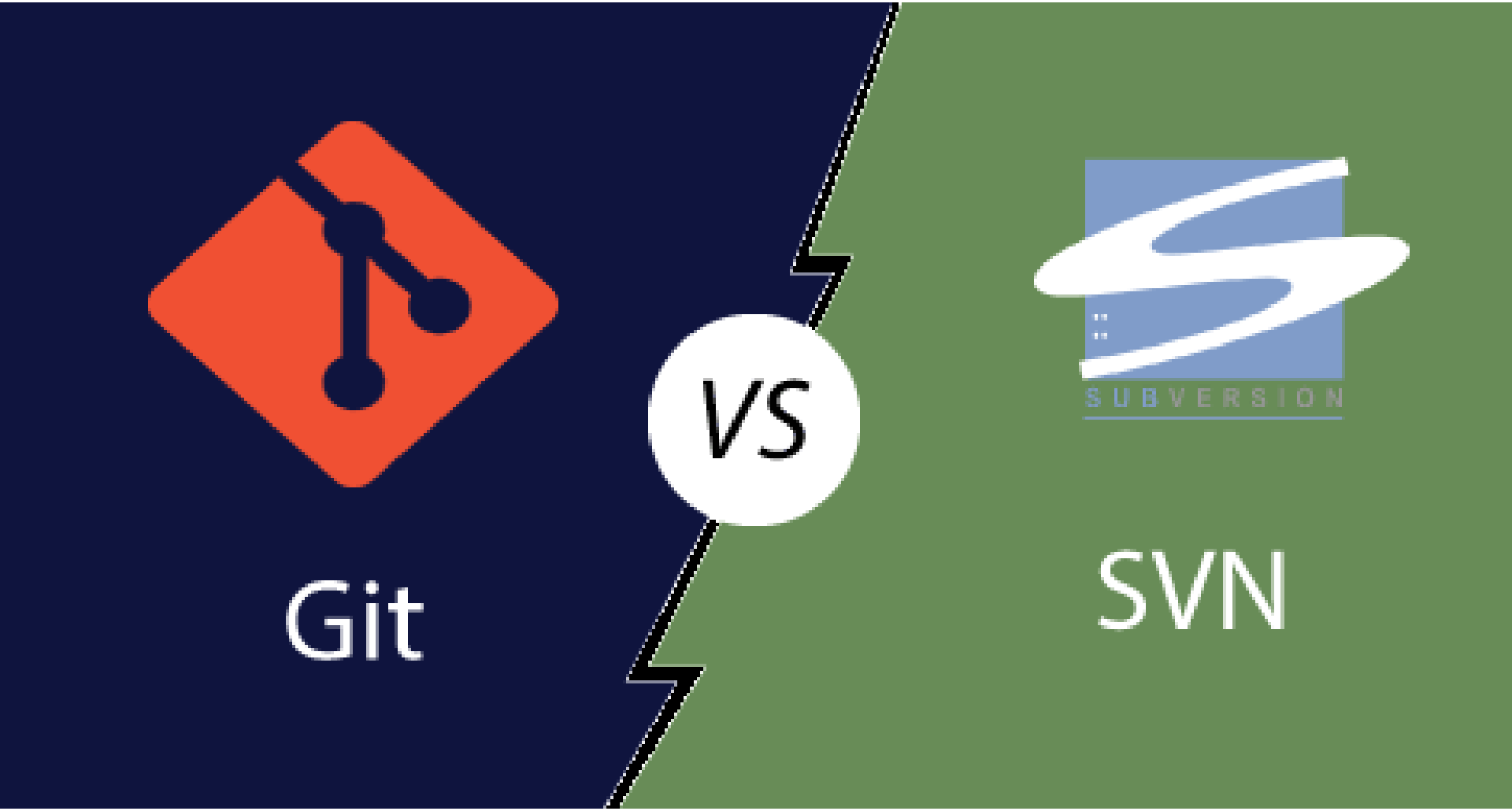


SVN for Eclipse



SVN for Eclipse




Git
VS
SUBVERSION
SVN

Git vs SVN

Git	SVN
It's a distributed version control system.	It's a Centralized version control system
Git is an SCM (source code management).	SVN is revision control.
Git has a cloned repository.	SVN does not have a cloned repository.
The Git branches are familiar to work. The Git system helps in merging the files quickly and also assist in finding the unmerged ones.	The SVN branches are a folder which exists in the repository. Some special commands are required For merging the branches.
Git does not have a Global revision number.	SVN has a Global revision number.
Git has cryptographically hashed contents that protect the contents from repository corruption taking place due to network issues or disk failures.	SVN does not have any cryptographically hashed contents.
Git stored content as metadata.	SVN stores content as files.
Git has more content protection than SVN.	SVN's content is less secure than Git.
Linus Torvalds developed git for Linux kernel.	CollabNet, Inc developed SVN.
Git is distributed under GNU (General public license).	SVN is distributed under the open-source license.

Questions

Module Summary

- In this module we discussed
  - Overview of Maven
  - Maven archetypes
  - Maven life cycle phases
  - The pom.xml file
  - Creation of Java projects using Maven
  - Creation of war files