DevOps

Lesson 03- GIT

Lesson Objectives

- Introduction to GIT
- Version Control
- Repositories and Branches
- Working Locally with GIT
- Working Remotely with GIT

Introduction

- Initially developed by Linus Torvalds, Git is a distributed version control system

- Git is a distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity and support for distributed, non-linear workflows

- As with most other distributed revision control systems, and unlike most client–server systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server. Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2

Introduction

- Design Philosophy
  - Free & Open Source
  - Blazingly Fast
  - Distributed
  - Data Assurance
  - Strong support for non-linear development
  - Compatibility with existing systems/protocols
  - Toolkit-based design

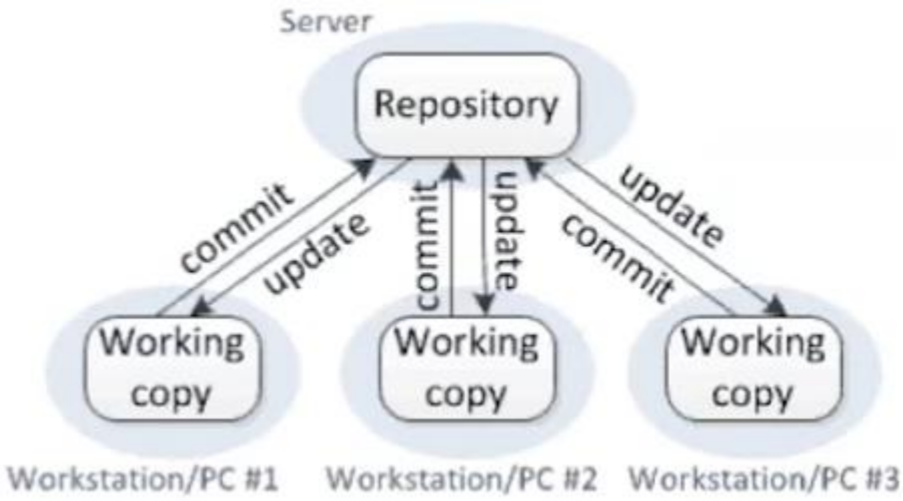
Version Control System

- Version control systems are software that manage changes to files e.g. documents, images, code etc.

- Benefits of Version Control
  - Saves from creating multiple backup files
  - Allow multiple people to work on same file
  - Track changes & see who had made changes
  - Easy to switch back to older version when needed
  - Increases productivity
- Two types of Version Control
  - Client sever Version Control- SVN, CVS, PerForce, IBM rational
  - Distributed Version Control -GIT

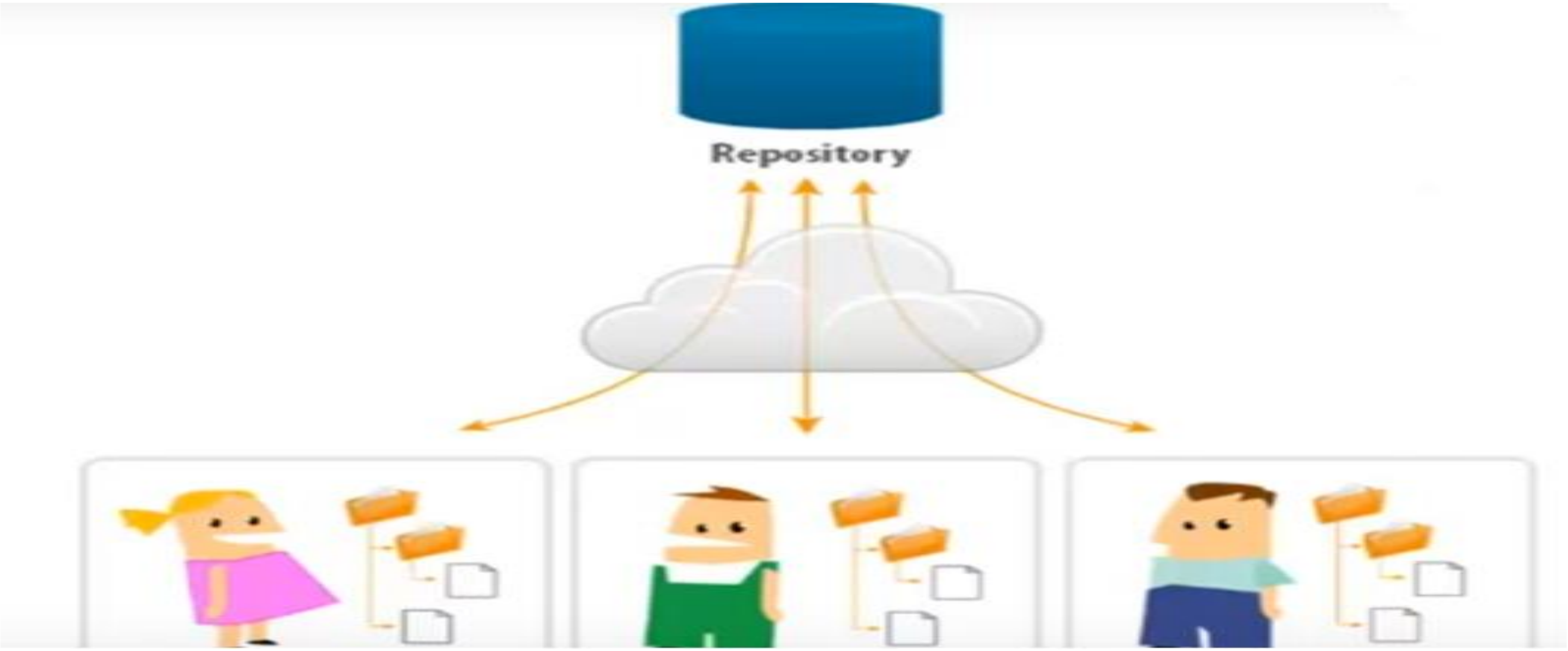
Problem with Client Server Version Control

- Client Server version control system works on a centralized model which has a single repository to which user check-in & check-out
- Some of the major benefits working with version control system are listed below:
  - Version control is not available on local system
  - If the central server get corrupted the entire history is lost

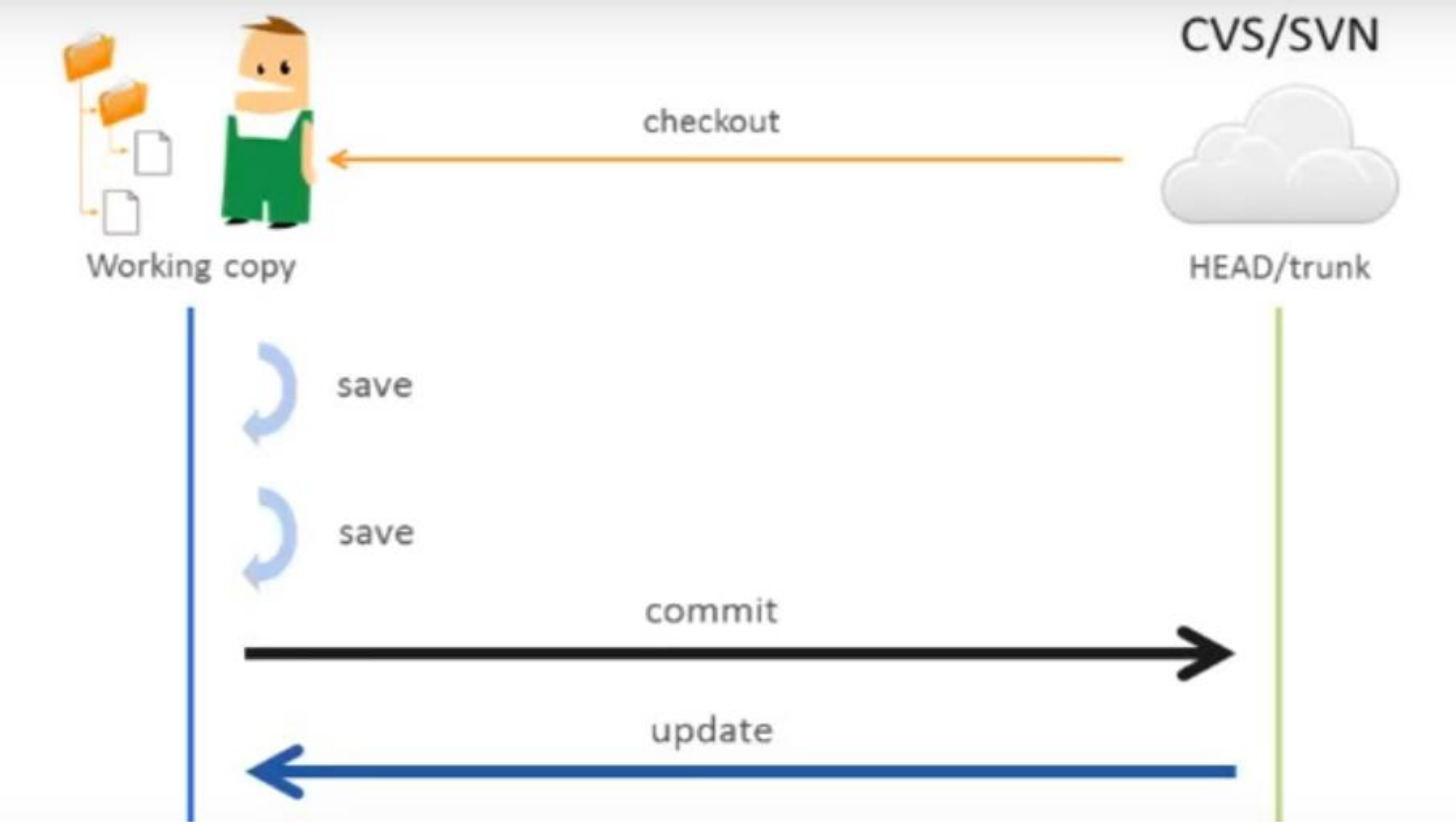


Problem with Client Server Version Control

- History in one Repository
- Client only gets a single revision per checkout
- All commits go into one repository

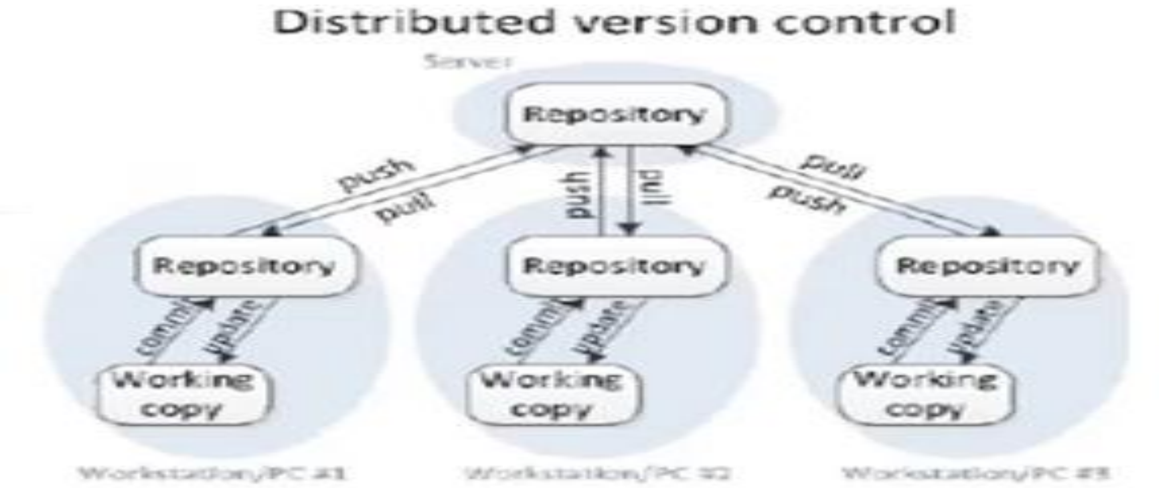


Problem with Client Server Version Control

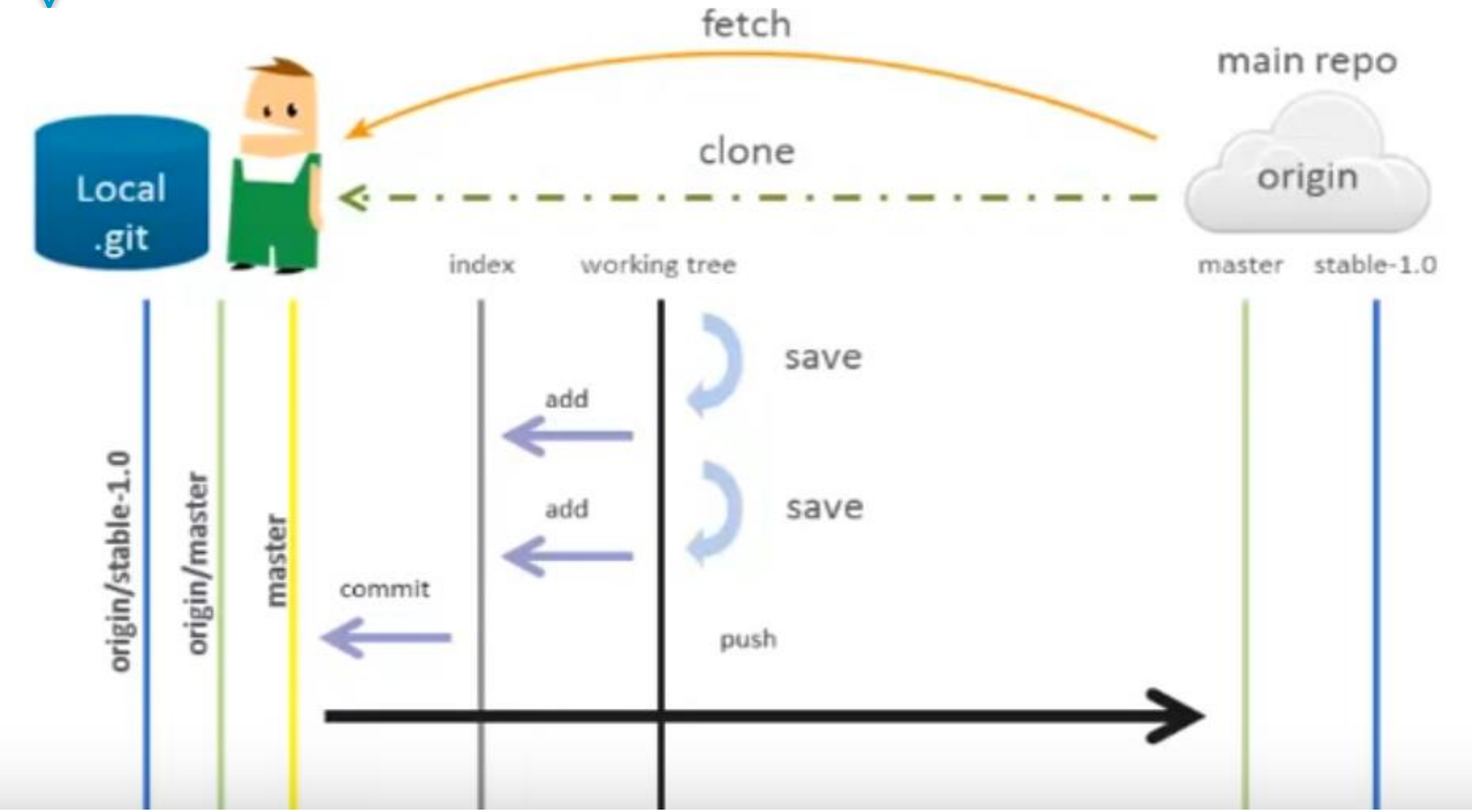


Advantage of Distributed Version Control-GIT

- Distributed version control system don’t rely on central server. It allows one of the repository on their own drive with entire history of the project
- Benefits of Distributed version control :
  - DVCS is also available on local machine.
  - No single point of failure as each user has the repository with entire history
  - Performs all action locally , even when not connected to internet



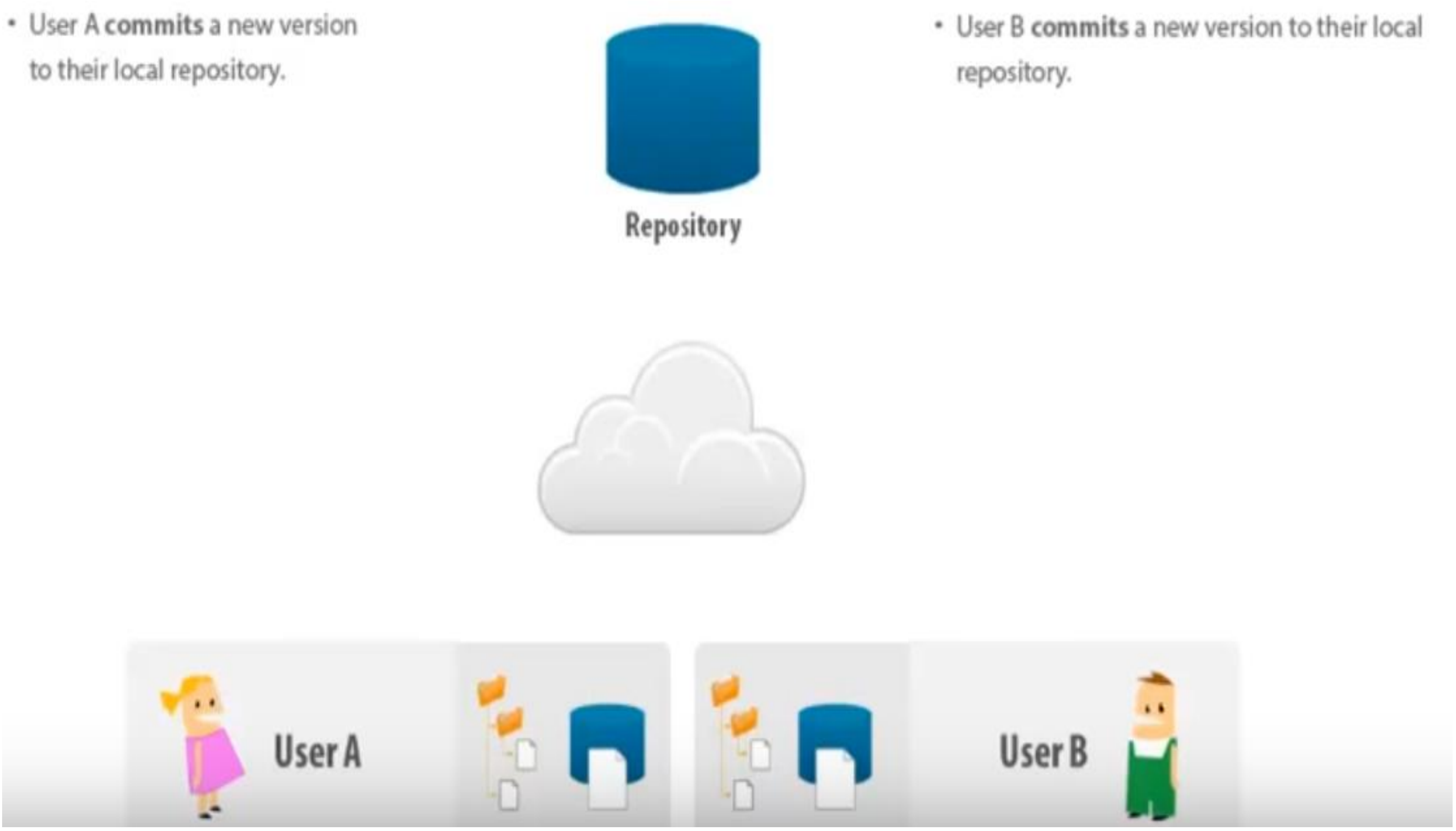
How Distributed Version Control-GIT Works



Version Control Parallel Development -GIT

- Version control system helps in parallel development and preventing one user from overwriting the work of another
- Two ways to solve parallel development problem:
  - Copy-modify-merge –GIT uses this
  - Lock-modify-unlock(practically not possible)

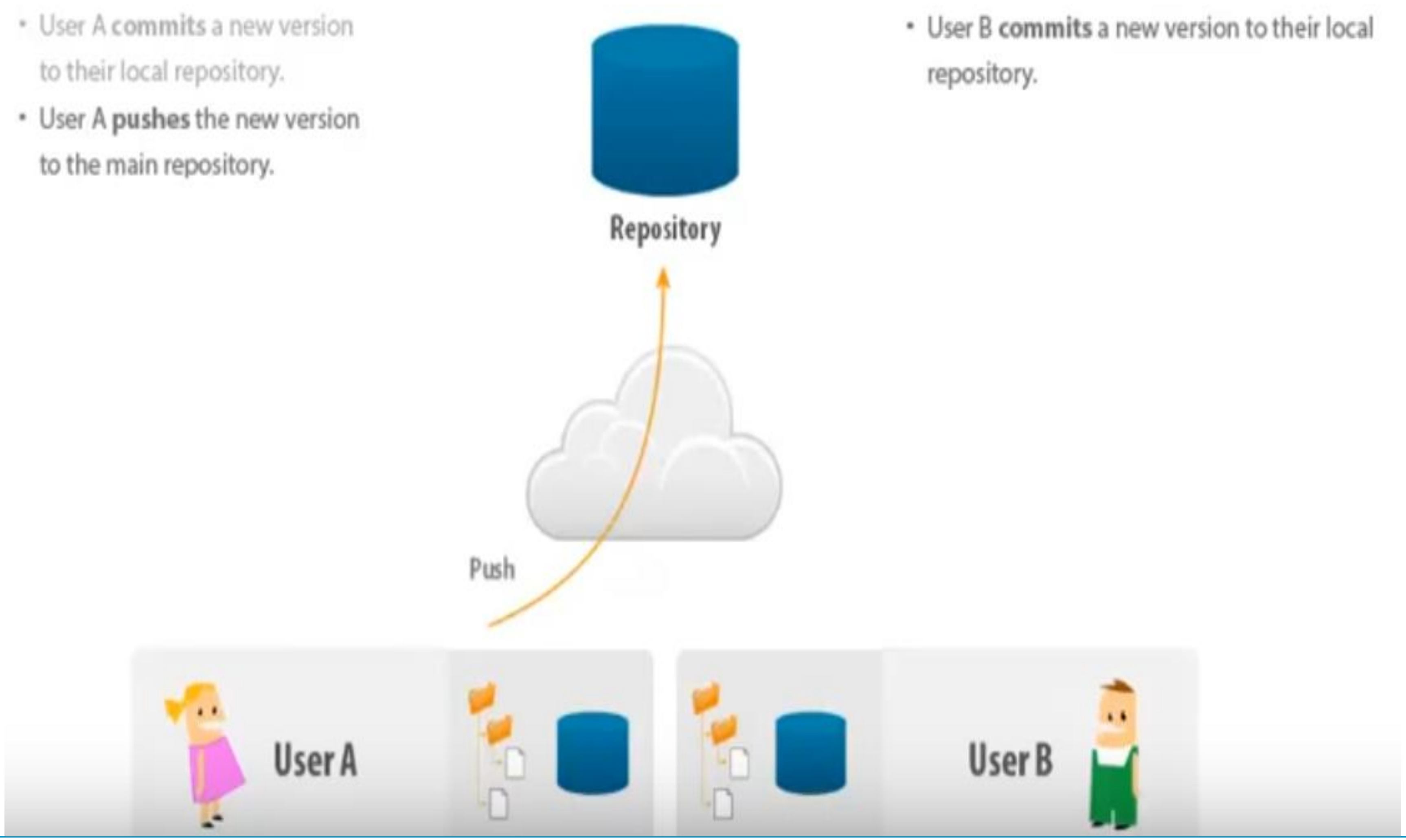
Version Control Parallel Development -GIT



- User A **commits** a new version to their local repository.

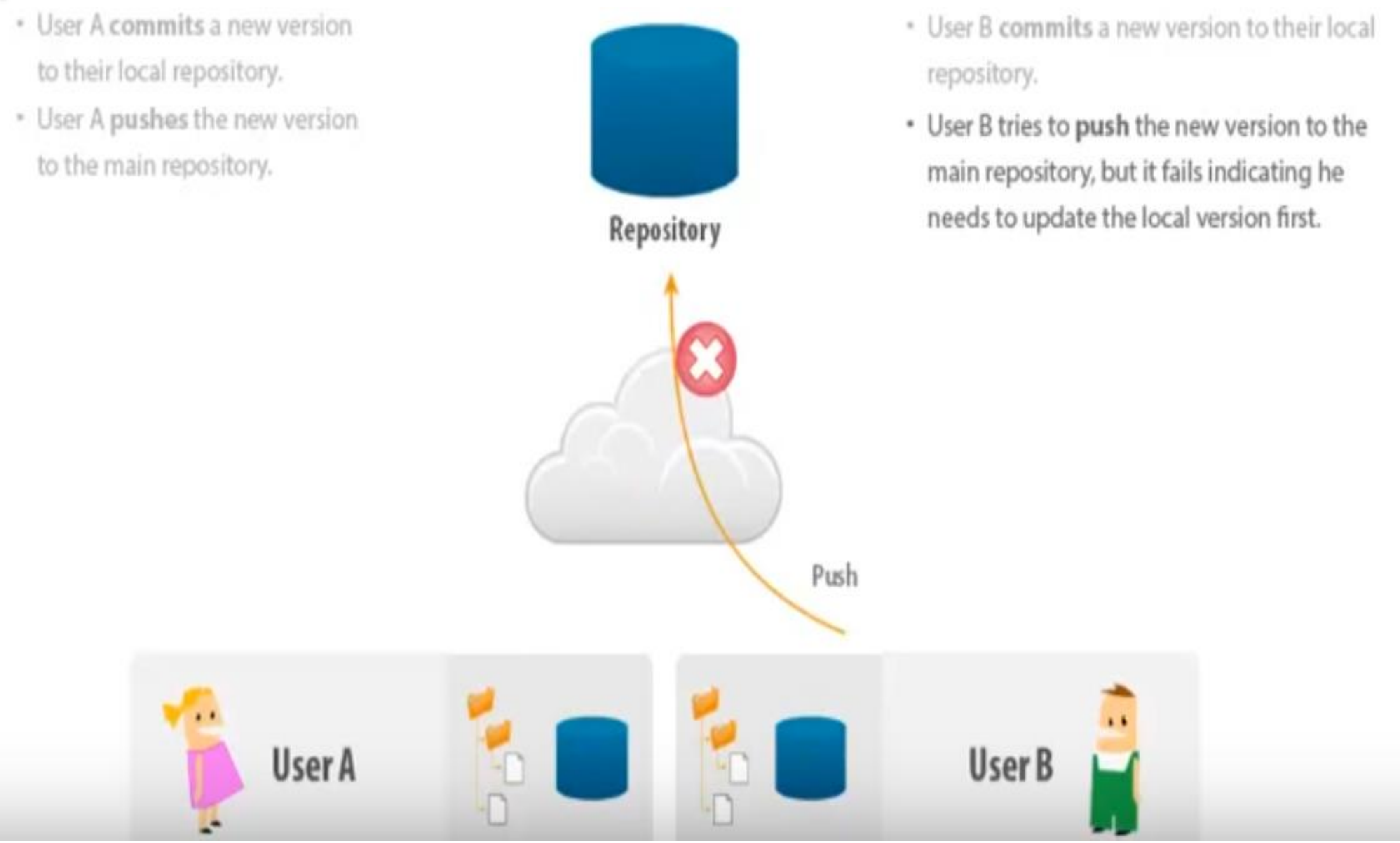
- User B **commits** a new version to their local repository.

Version Control Parallel Development -GIT



- User A **commits** a new version to their local repository.
- User A **pushes** the new version to the main repository.
- User B **commits** a new version to their local repository.

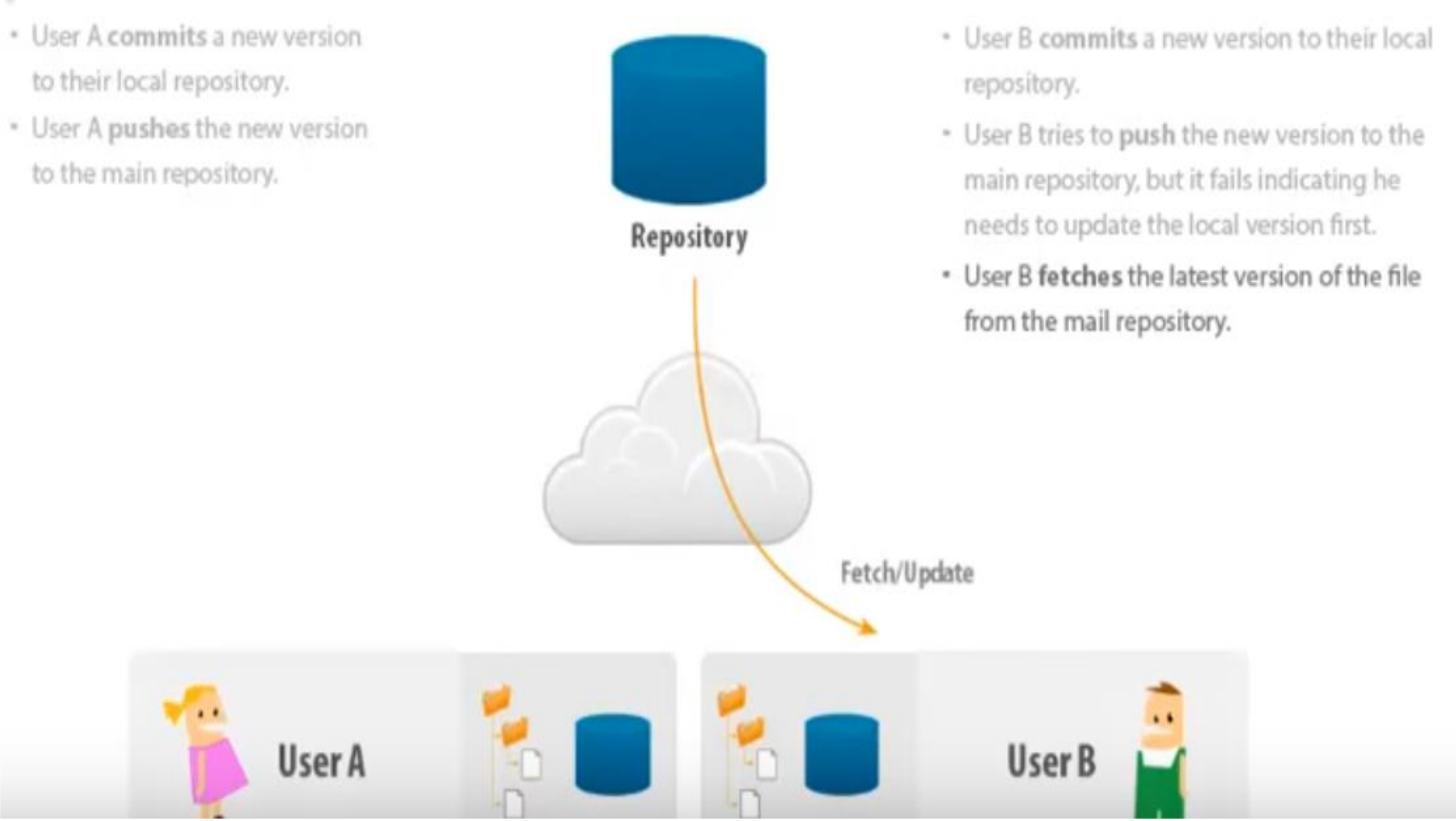
Version Control Parallel Development -GIT



- User A **commits** a new version to their local repository.
- User A **pushes** the new version to the main repository.

- User B **commits** a new version to their local repository.
- User B tries to **push** the new version to the main repository, but it fails indicating he needs to update the local version first.

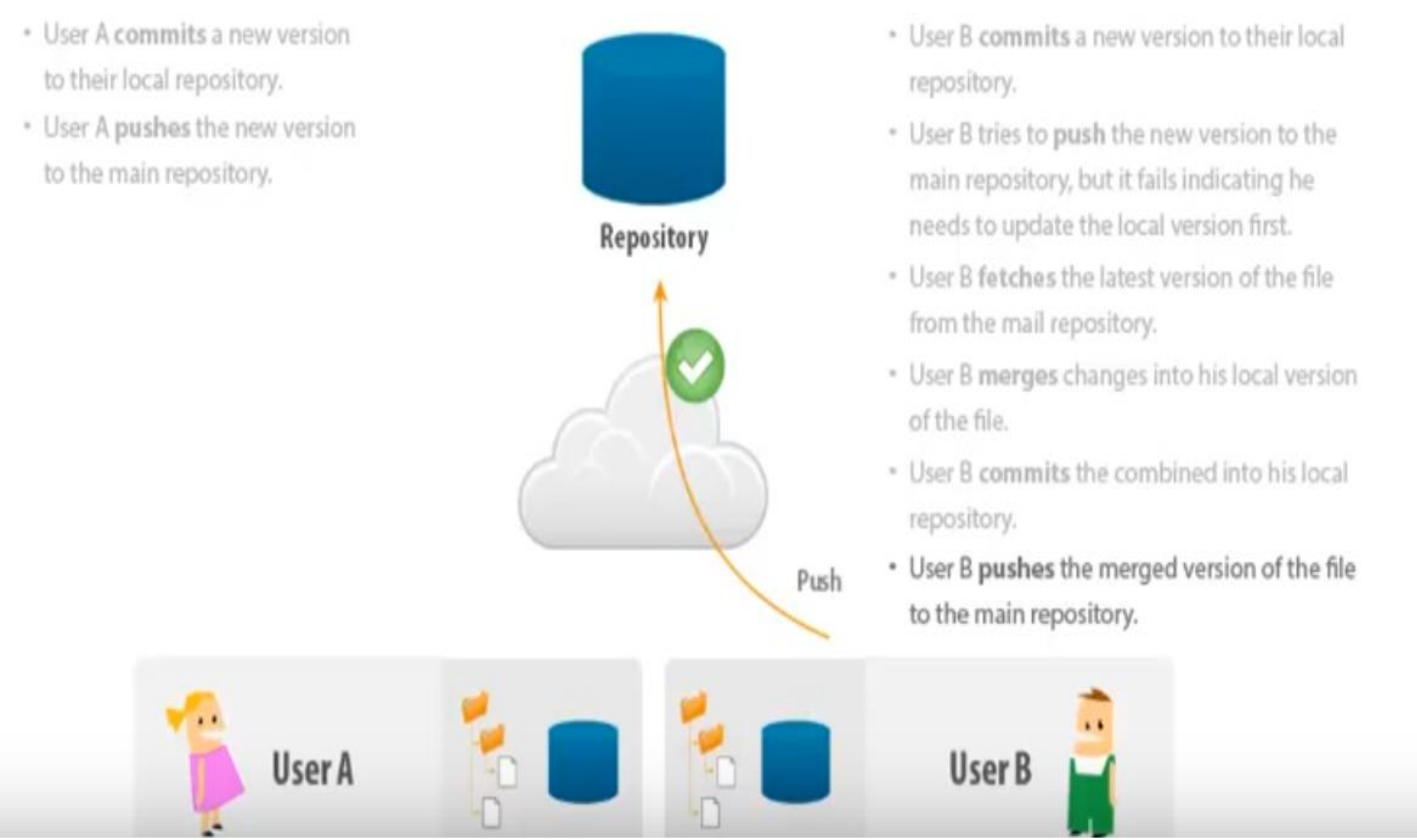
Version Control Parallel Development -GIT



- User A **commits** a new version to their local repository.
- User A **pushes** the new version to the main repository.

- User B **commits** a new version to their local repository.
- User B tries to **push** the new version to the main repository, but it fails indicating he needs to update the local version first.
- User B **fetches** the latest version of the file from the mail repository.

Parallel Development, Copy Modify Merge

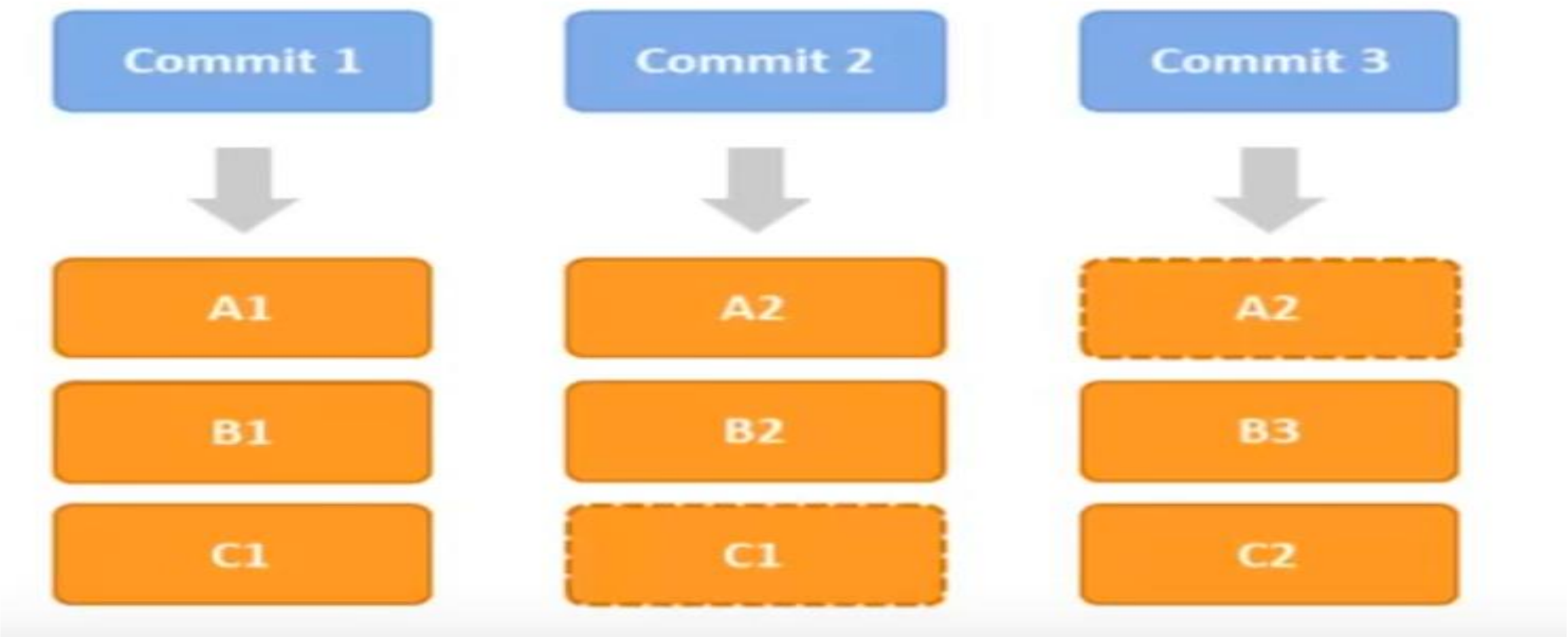


- User A **commits** a new version to their local repository.
- User A **pushes** the new version to the main repository.

- User B **commits** a new version to their local repository.
- User B tries to **push** the new version to the main repository, but it fails indicating he needs to update the local version first.
- User B **fetches** the latest version of the file from the mail repository.
- User B **merges** changes into his local version of the file.
- User B **commits** the combined into his local repository.
- User B **pushes** the merged version of the file to the main repository.

Git – Snapshot Storage

- Snapshot Storage –stores the complete files changed by a commit along with references to files that were not changed by that commit.



GIT-Repositories

- Repositories:
  - It is a collection of refs together with an object database containing all objects which are reachable from the refs, possibly accompanied by meta data from one or more porcelains. A repository can share an object database with other repositories via alternate mechanism.
- What to store in repositories?
  - Anything, however any sort of editable files are preferred.

Git – Snapshot Storage

- Creating repositories :
  - at default location
    - git init
  - at particular location
    - git init c:/testGIT
  - Bare repository
    - git init –bare

- How to get GIT repository:
  - $ git clone git://git.kernel.org/pub/scm/git/git.git

It does approx. 225 MB download

Repositories and Branches

- Branches:
  - A "branch" is an active line of development.
  - The most recent commit on a branch is referred to as the tip of that branch.
  - The tip of the branch is referenced by a branch head, which moves forward as additional development is done on the branch.
  - A single git repository can track an arbitrary number of branches, but working tree is associated with just one of them (the "current" or "checked out" branch), and HEAD points to that branch

Repositories and Branches

- Getting different versions of project:
  - Git is best thought of as a tool for storing the history of a collection of files. It stores the history as a compressed collection of interrelated snapshots of the project's contents. In git each such version is called a commit.
  - Those snapshots aren't necessarily all arranged in a single line from oldest to newest; instead, work may simultaneously proceed along parallel lines of development, called branches, which may merge and diverge.
  - A single git repository can track development on multiple branches. It does this by keeping a list of heads which reference the latest commit on each branch; the git-branch(1) command shows you the list of branch heads:
  - $ git branch * master
  - A freshly cloned repository contains a single branch head, by default named "master", with the working directory initialized to the state of the project referred to by that branch head.
  - Most projects also use tags. Tags, like heads, are references into the project's history, and can be listed using thegit-tag(1) command:
  - $ git tag -l

Understanding History-Repositories

- Commits:
  - Every change in the history of a project is represented by a commit. The git-show(1) command shows the most recent commit on the current branch:
  - $ git show
  - Every commit (except the very first commit in a project) also has a parent commit which shows what happened before this commit. Following the chain of parents will eventually take you back to the beginning of the project.
  - However, the commits do not form a simple list; git allows lines of development to diverge and then reconverge, and the point where two lines of development reconverge is called a "merge". The commit representing a merge can therefore have more than one parent, with each parent representing the most recent commit on one of the lines of development leading to that point.
  - The best way to see how this works is using the gitk(1)command; running gitk now on a git repository and looking for merge commits will help understand how the git organizes history.
  - In the following, commit X is "reachable" from commit Y if commit X is an ancestor of commit Y. Equivalently, Y is a descendant of X, or that there is a chain of parents leading from commit Y to commit X.

Understanding History-Trees

- The working tree is the current view into the repository. It has all the files from your project: the source code, build files, unit tests, and so on.
- Some VCSs refer to this as your working copy. People coming to Git for the first time from another VCS often have trouble separating the working tree from the repository. In a VCS such as Subversion, your repository exists “over there” on another server.
- In Git, “over there” means in the .git/ directory inside your project’s directory on your local computer. This means you can look at the history of the repository and see what has changed without having to communicate with a repository on another server.

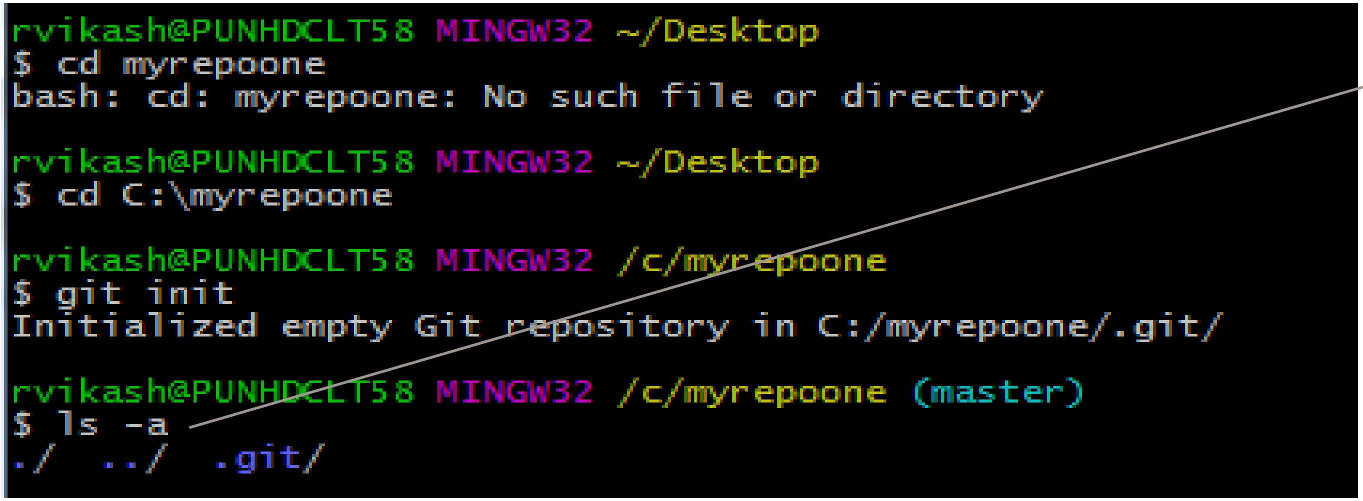
Locally Working with GIT

- Download GIT From

https://git-scm.com/downloads

- First Repository :
  - Create Empty directory on system myrepoone
  - Create a repository

  git init



```
rvikash@PUNHDCLT58 MINGW32 ~/Desktop
$ cd myrepoone
bash: cd: myrepoone: No such file or directory

rvikash@PUNHDCLT58 MINGW32 ~/Desktop
$ cd C:\myrepoone

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone
$ git init
Initialized empty Git repository in C:/myrepoone/.git/

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ ls -a
./  ../  .git/
```

Checking repository created or not

Locally Working with GIT

- Setting Name & email Id
  - Setting name

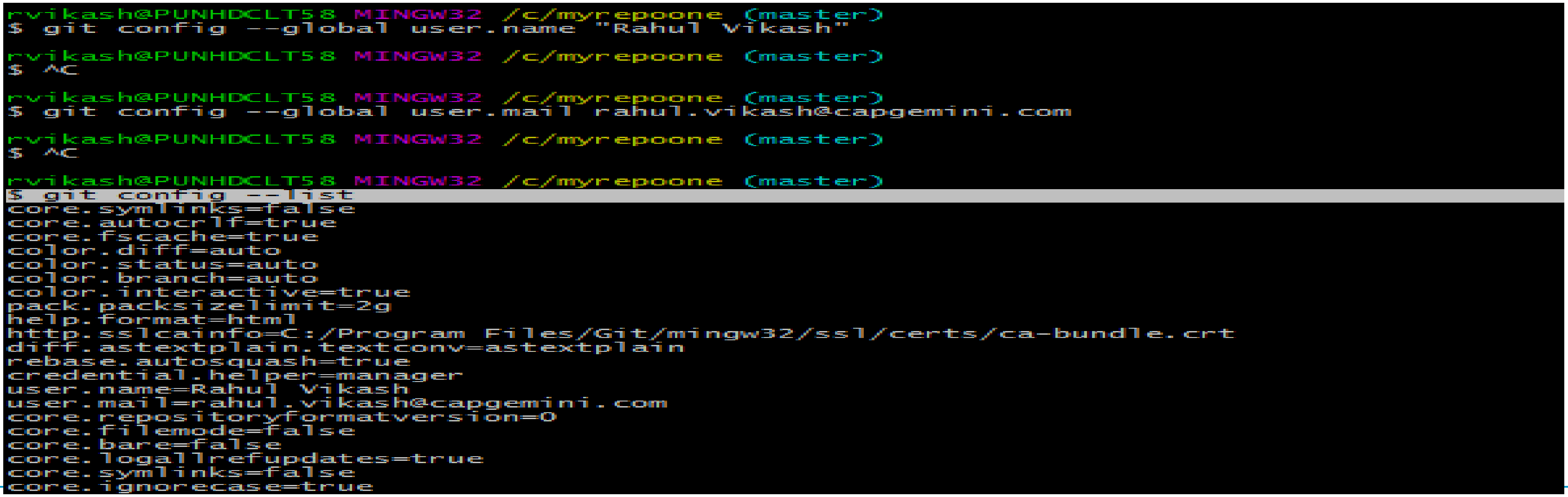
git config --global user.name "Rahul Vikash"

  - Setting email

$ git config --global user. Mail rahul.vikash@capgemini.com

  - Checking configuration

$ git config –list



```
rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git config --global user.name "Rahul Vikash"

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ ^C

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git config --global user.mail rahul.vikash@capgemini.com

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ ^C

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsizelimit=2g
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw32/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
credential.helper=manager
user.name=Rahul Vikash
user.mail=rahul.vikash@capgemini.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

Locally Working with GIT

- Adding File in repository

MyFirst.txt

- Check

```
$ ls
```

- Know the status

```
 $ git status
```

- Add the file in staging

```
$ git add MyFirst.txt
```

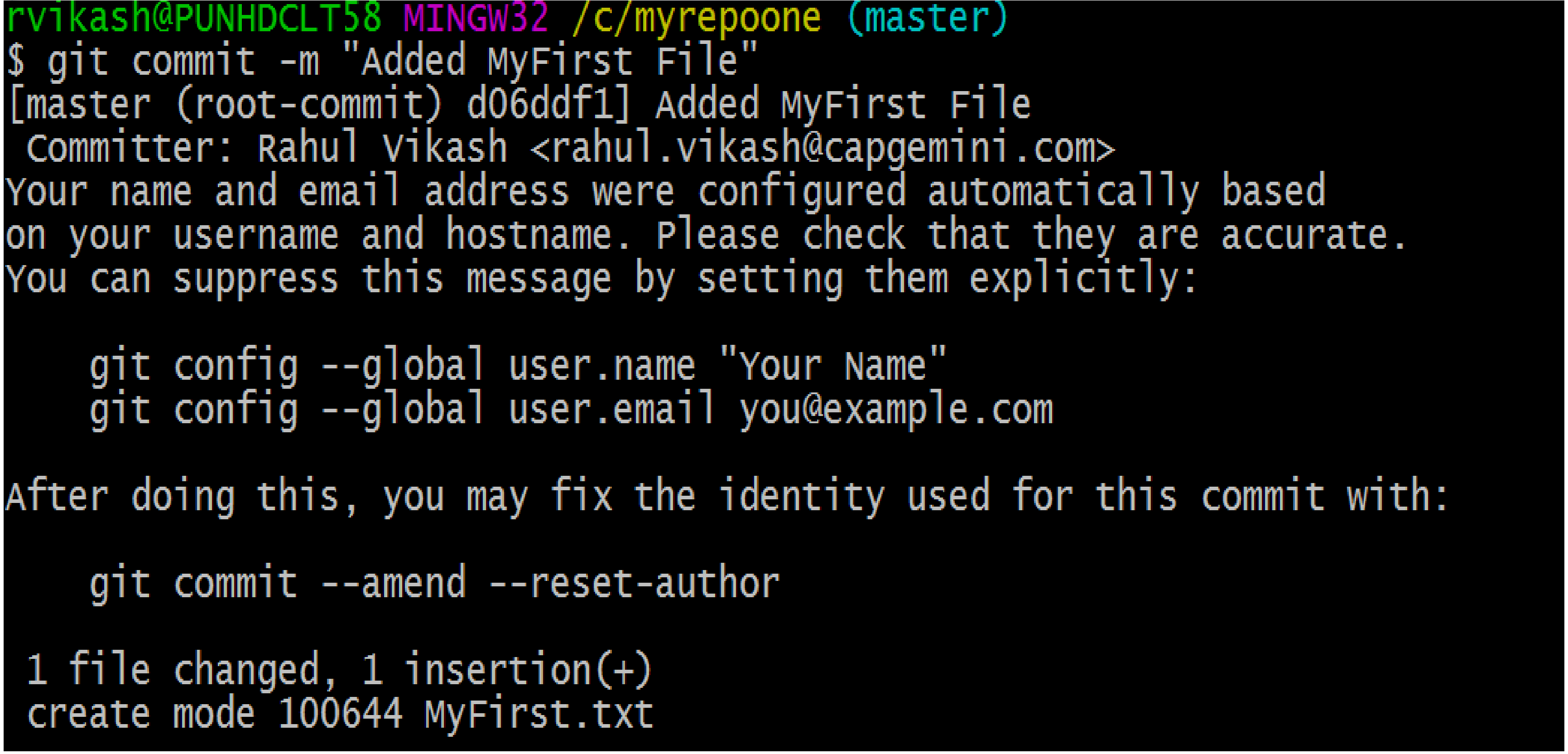
- Commit in git repository with lock message

```
$ git commit -m "Added MyFirst File“
```

- See log

```
$ git log
```

Locally Working with GIT

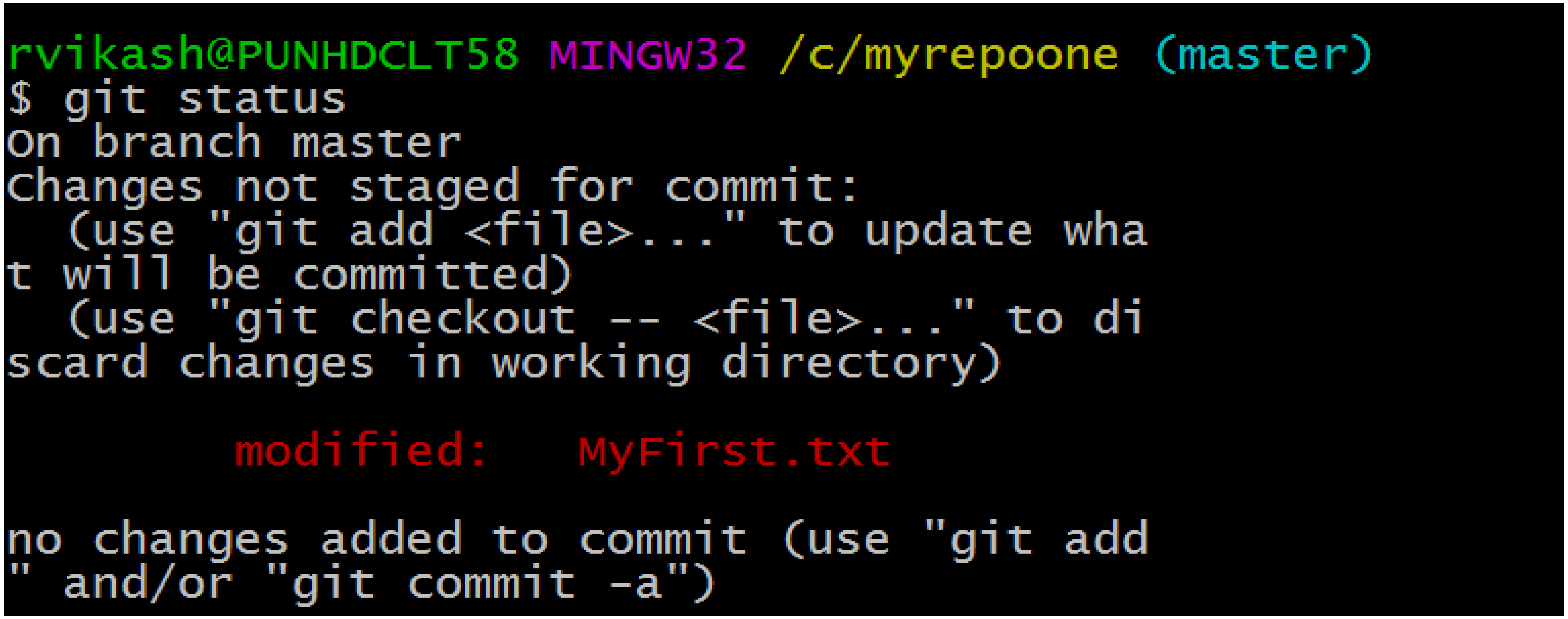


```
rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git commit -m "Added MyFirst File"
[master (root-commit) d06ddf1] Added MyFirst File
 Committer: Rahul Vikash <rahul.vikash@capgemini.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 MyFirst.txt
```

Locally Working with GIT-

Adding new data in MyFirst.txt

Now checking status $git status



```
rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update wha
t will be committed)
  (use "git checkout -- <file>..." to di
scard changes in working directory)

        modified:   MyFirst.txt

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

Commit $ git commit -am "Added New MyFirst File“

Again check log --$git log

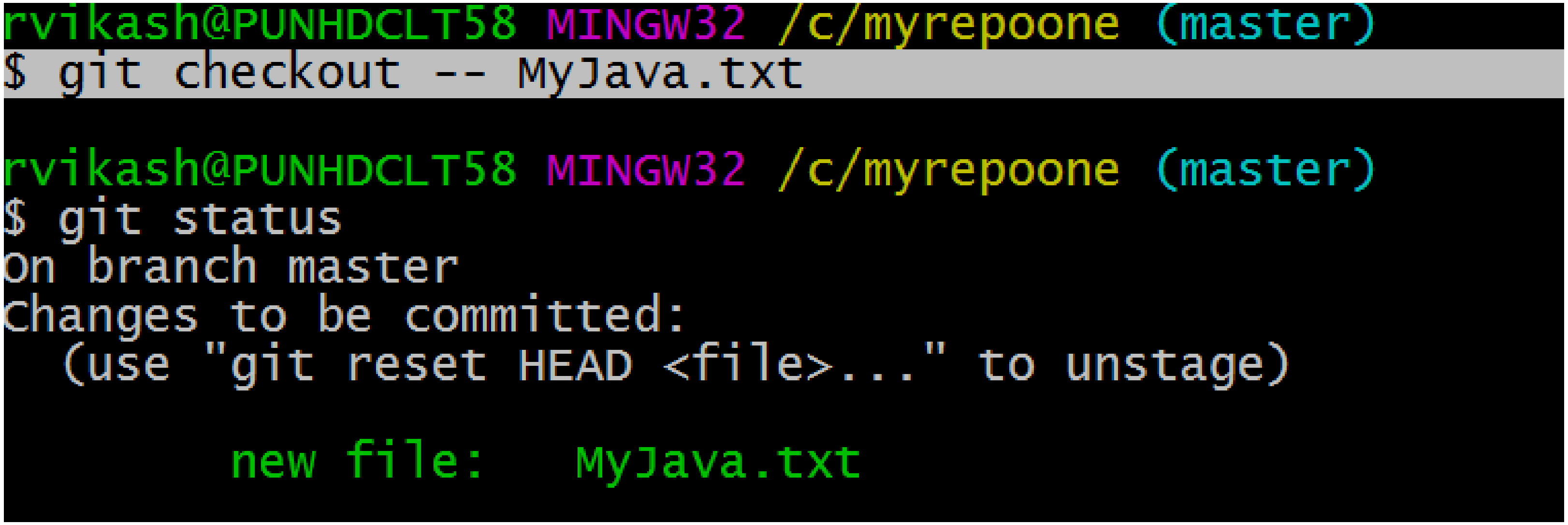
Locally Working with GIT-

- To check which line has changed

  $ git commit - -amend
- Add one more file –MyJava.txt
- Unstage the One File

$ git reset HEAD MyFirst.txt

- Unchanged the work done --$ git checkout -- MyJava.txt



```
rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git checkout -- MyJava.txt

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   MyJava.txt
```

Locally Working with GIT

- Multiple Reset for n number of file ,Goes to last commit

```
$ git reset -- hard
```

Remotely Working with GIT

Adding MyFirst.Java in Remote Repository

Create account in : https://github.com

```
$ git init
$ git add MyFirst.java
$ git commit -m "first commit"
$ git remote add origin https://github.com/rahulviki86/basicdemo.git
$ git push -u origin master
```

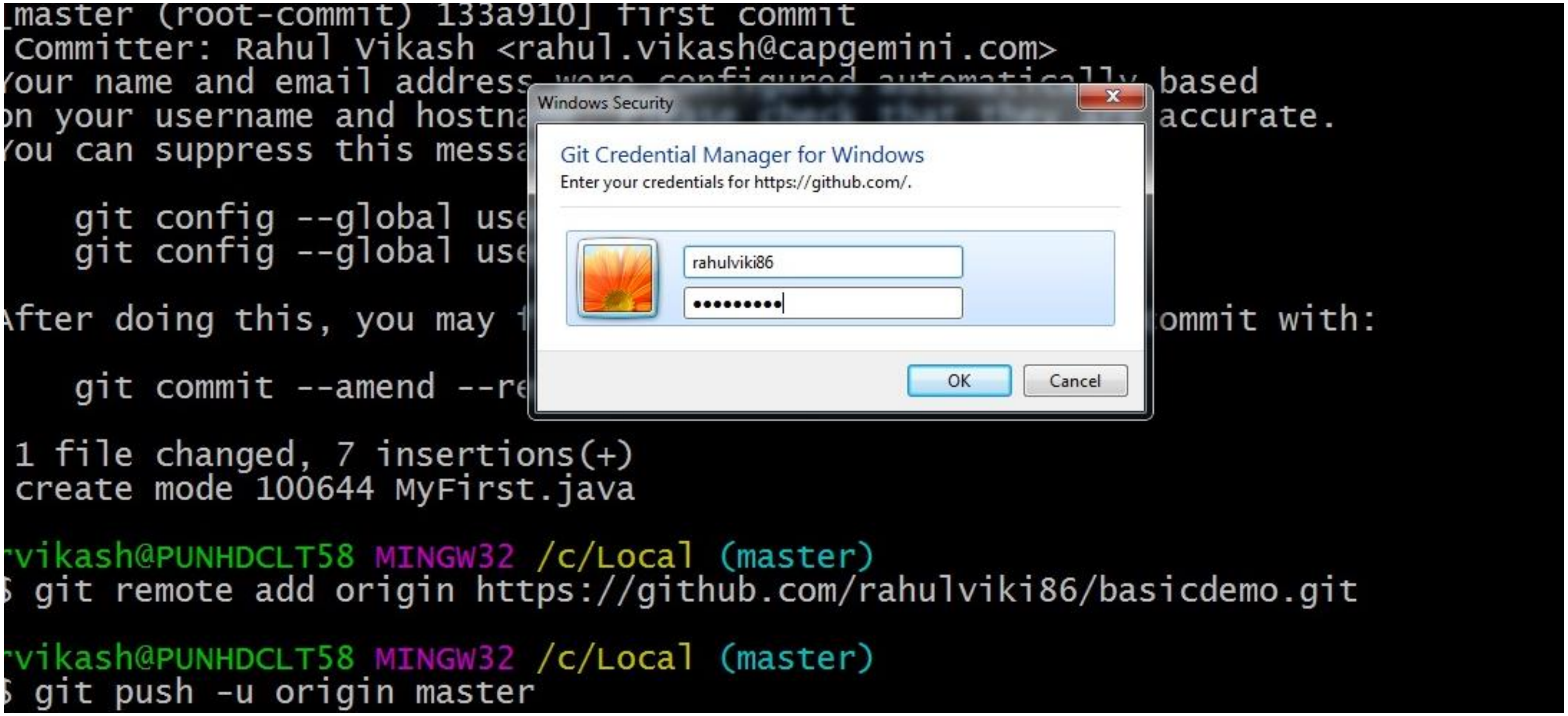
User ID

Repository Name

Remotely Working with GIT

After running

$ git push -u origin master – Ask for username & Password



```
[master (root-commit) 133a910] first commit
 Committer: Rahul Vikash <rahul.vikash@capgemini.com>
Your name and email address were configured automatically based
on your username and hostna                              accurate.
You can suppress this messa

    git config --global use
    git config --global use

After doing this, you may f                              commit with:

    git commit --amend --re

 1 file changed, 7 insertions(+)
 create mode 100644 MyFirst.java

rvikash@PUNHDCLT58 MINGW32 /c/Local (master)
$ git remote add origin https://github.com/rahulviki86/basicdemo.git

rvikash@PUNHDCLT58 MINGW32 /c/Local (master)
$ git push -u origin master
```

Windows Security

Git Credential Manager for Windows

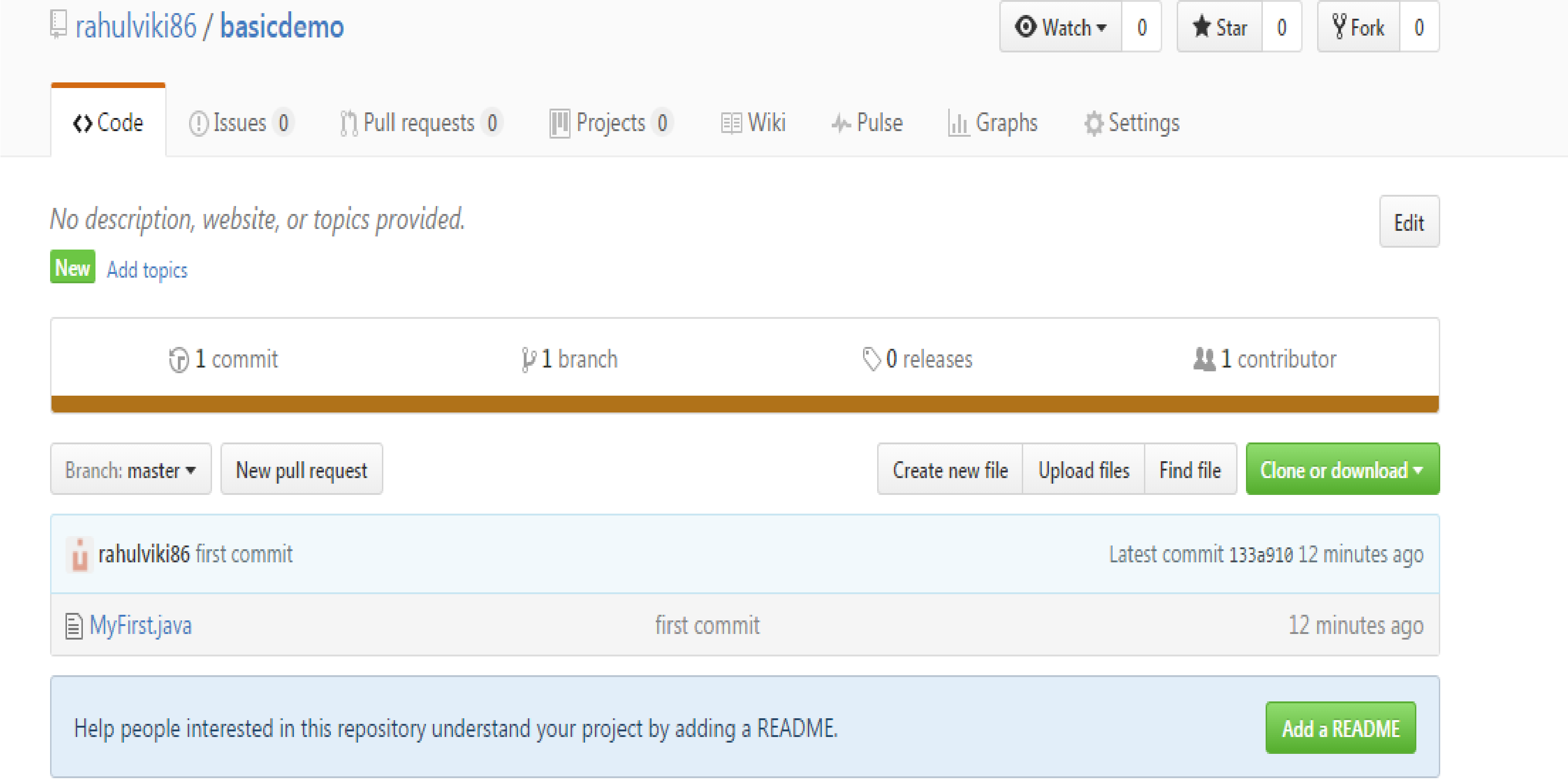
Enter your credentials for https://github.com/.

rahulviki86

••••••••••

Remotely Working with GIT

Added in the Remote Repository



rahulviki86 / basicdemo

No description, website, or topics provided.

rahulviki86 first commit — Latest commit 133a910 12 minutes ago

MyFirst.java	first commit	12 minutes ago

Help people interested in this repository understand your project by adding a README.

Remotely Working with GIT

- Cloneing the data ----git clone <repo> <directory>

  - When you run git clone, the following actions occur:
  - A new folder called repo is made
  - It is initialized as a Git repository
  - A remote named origin is created, pointing to the URL you cloned from
  - All of the repository's files and commits are downloaded there
  - The default branch (usually called master) is checked out
  - For every branch foo in the remote repository, a corresponding remote-tracking branch refs/remotes/origin/foo is created in your local repository. You can usually abbreviate such remote-tracking branch names to origin/foo.

Syntax:

$git clone https://github.com/rahulviki86/basicdemo.git

Remotely Working with GIT

- git-fetch - Download objects and refs from another repository, Use git fetch to retrieve new work done by other people. Fetching from a repository grabs all the new remote-tracking branches and tags *without* merging those changes into your own branches.

Syntax:

$git fetch *remotename*

- git-pull - Fetch from and integrate with another repository or a local branch, is a convenient shortcut for completing both git fetch and git merge in the same command.

Syntax:

$git pull *remotename branchname*

Remotely Working with GIT

- git-merge - Join two or more development histories together.Merging combines your local changes with changes made by others.

Syntax:

$git merge *remotename/branchname*

Demo

- Demo on GIT –Locally using git init, add, status, log
- Demo on GIT- Remotelly using git remote, pull, push, fetch, clone

Lab

- Lab 01

Summary

- Git is a distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity and support for distributed, non-linear workflows.
- Git working with local repository
- Git working with remote repository

Review Question

- To see the difference between which two branches the following command can be used?
  - git diff master branch_name
  - git --diff master branch_name
  - git merge master branch_name
  - git --stat master branch_name
- The git ______ command performs a git fetch and git merge.
  - Push
  - Pull
  - Clone
  - branch

Review Question

- Three files file1,file2 and file3 are modified. Identify the series of commands to view the modified files,

  then add their updated contents to the index and commit the changes.

  - '$ git status ->$ git commit
  - $ git diff –cached->$ git status->$ git commit –a
  - $ git add ->$ git diff –cached->$ git status->$ git commit
  - $ git add file1 file2 file3->$ git diff –cached->$ git status->$ git commit