

DevOps



Caltech

Center for Technology &
Management Education

Post Graduate Program in DevOps



Getting Started with Git

Learning Objectives

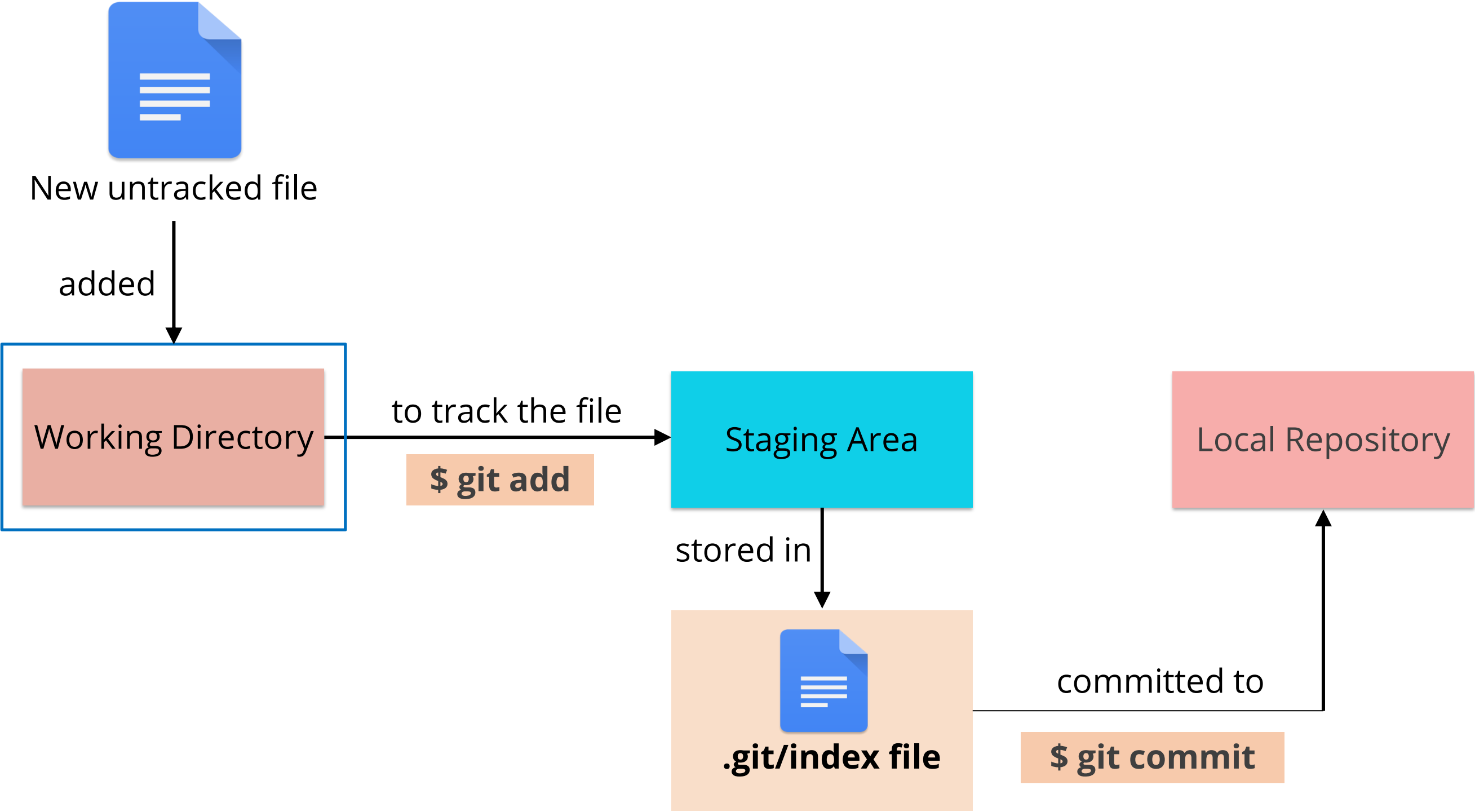
By the end of this lesson, you will be able to:

- Demonstrate the centralized git workflow
- Explain the different types of Git workflows
- Demonstrate various operations like tracking, reverting, deleting, ignoring, and renaming files in Git

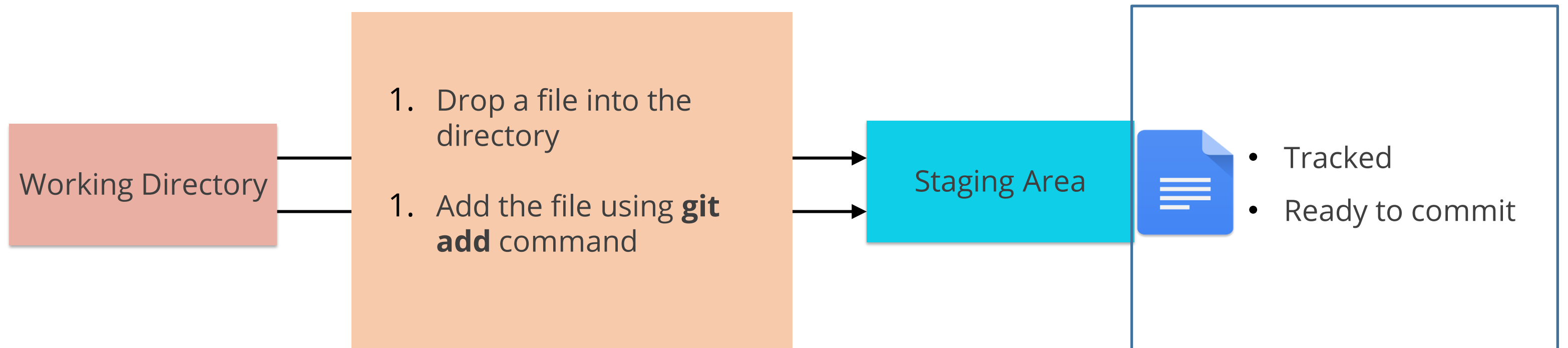


Git Workflow

Git's Three-Stage Workflow



Working Directory to Staging Area



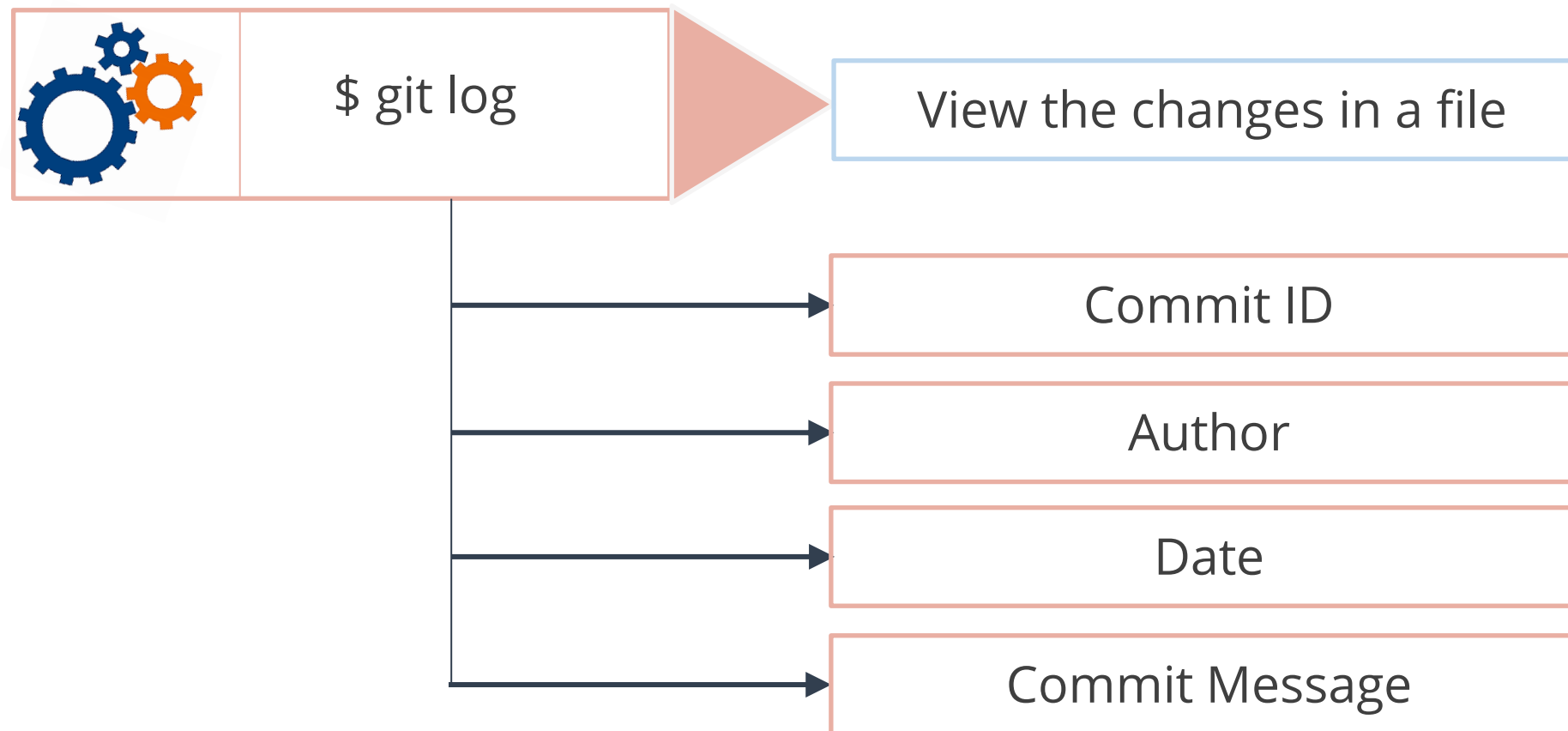
Staging Area to Local Repository



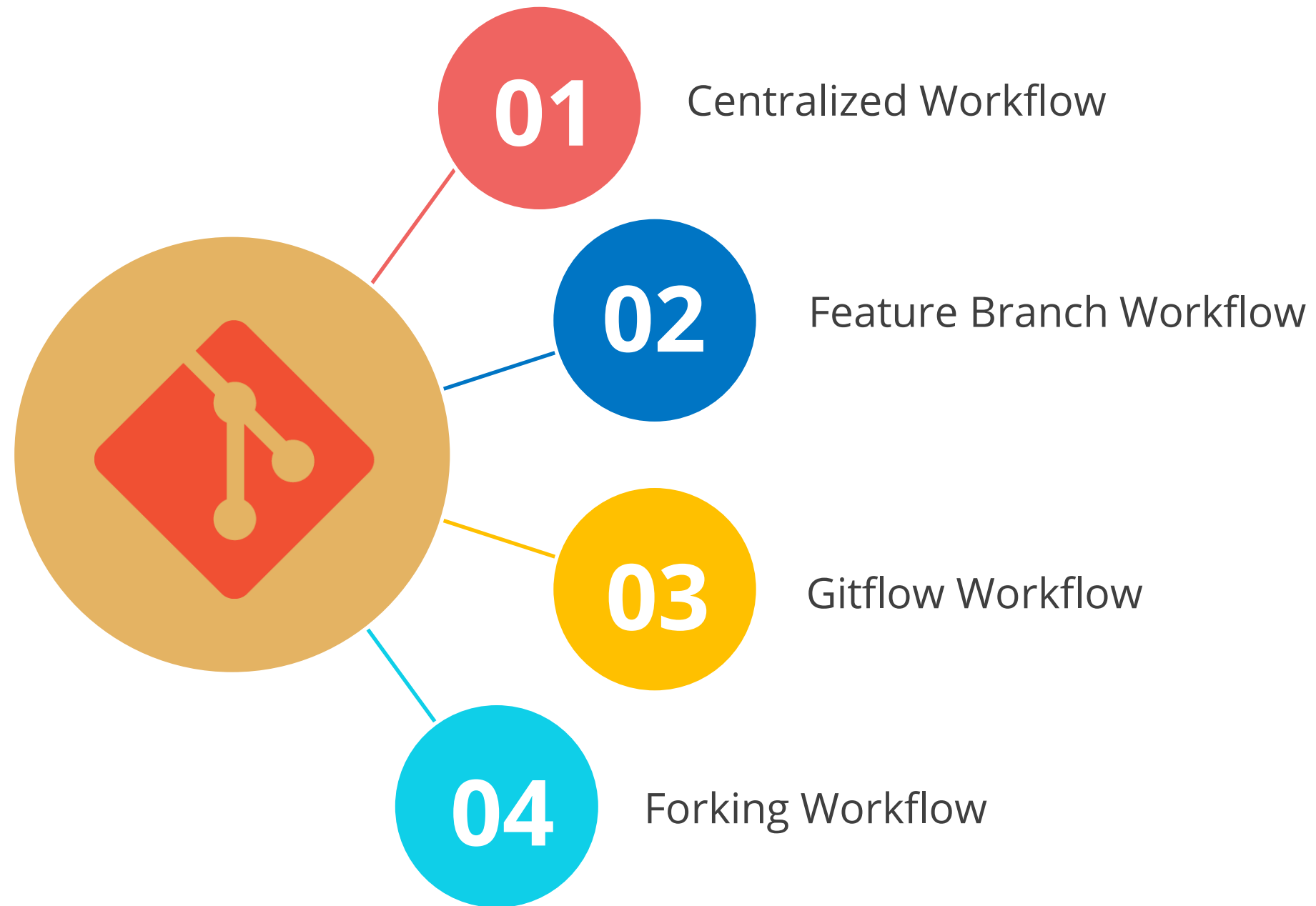
NOTE

While committing, it is important to add a message string using the `-m` flag. If missed, a default editor opens asking for comments.

Managing and Viewing Changes



Types of Git Workflow



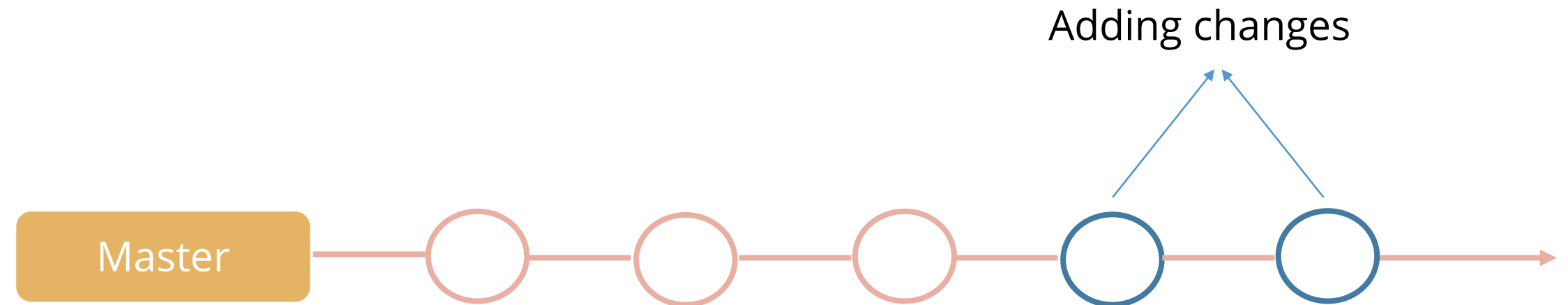
Centralized Workflow

Centralized Workflow

Feature Branch
Workflow

Gitflow Workflow

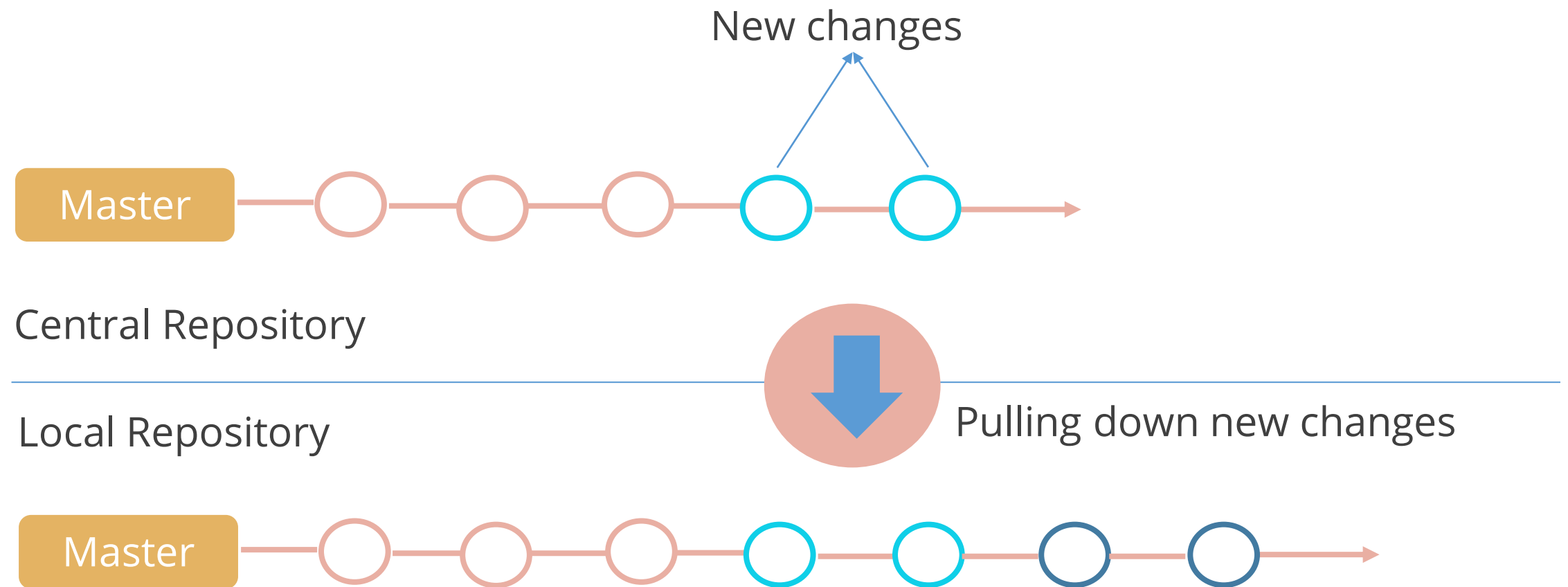
Forking Workflow



1. It has one main line called master
2. All the changes are being made on the main line

Centralized Workflow

Sync with central repository:



Changes made to the central repository can be pulled down and Git automatically applies those changes to the local repository.

In case of conflict, Git allows you to merge the changes manually. You can commit and merge those changes to your local branch which you can then push to the centralized.

Centralized Workflow

Feature Branch
Workflow

Gitflow Workflow

Forking Workflow

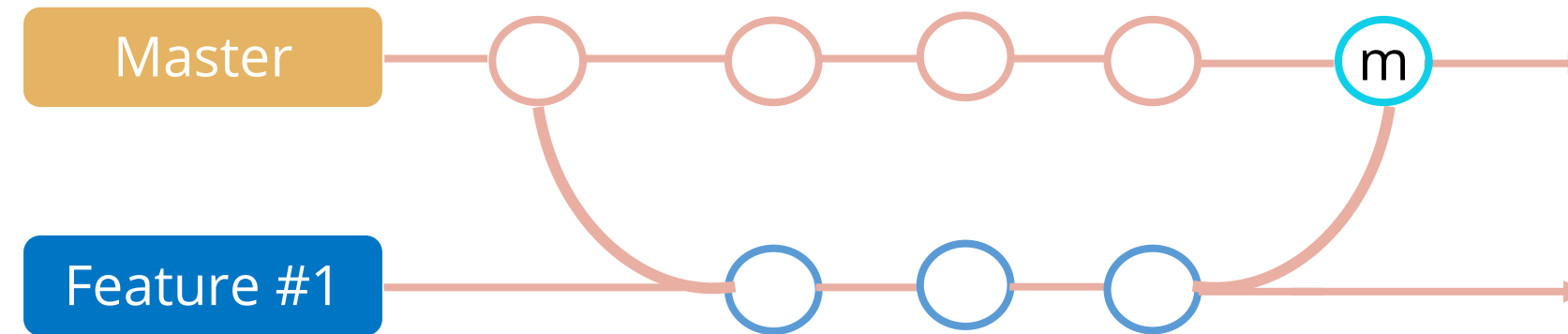
Feature Branch Workflow

Centralized Workflow

Feature Branch
Workflow

Gitflow Workflow

Forking Workflow



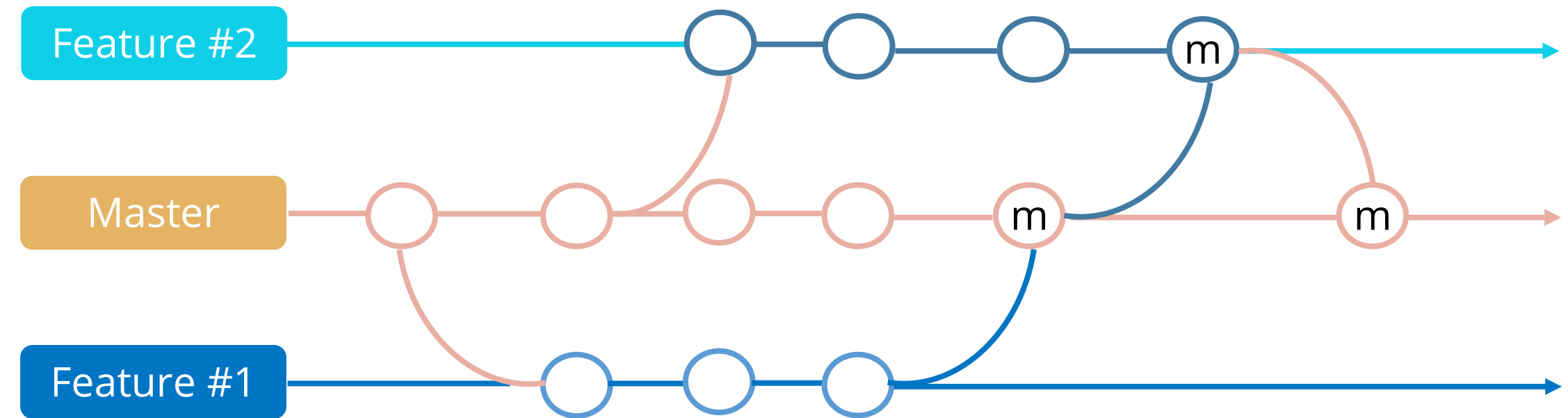
Any direct change to the master is avoided and all the bug fixes or feature work should happen on the feature branch.

Adding changes:

1. Pick a commit from a master that you want to branch off. Create a new branch every time you start to work on a new feature. You can update, add, commit, and push changes to this branch.
1. When ready, push your commits to your feature branch and then merge the changes back to the master branch.

Feature Branch Workflow

Working on feature in parallel:



You can have multiple features in parallel. Also, you can branch off Master at different points, commit changes, and merge it back to the master branch.

To ensure that your changes integrate with the master branch, you should merge the changes from master back into the feature branch. Then take the merge results which were tested locally, and then merge it back to the master.

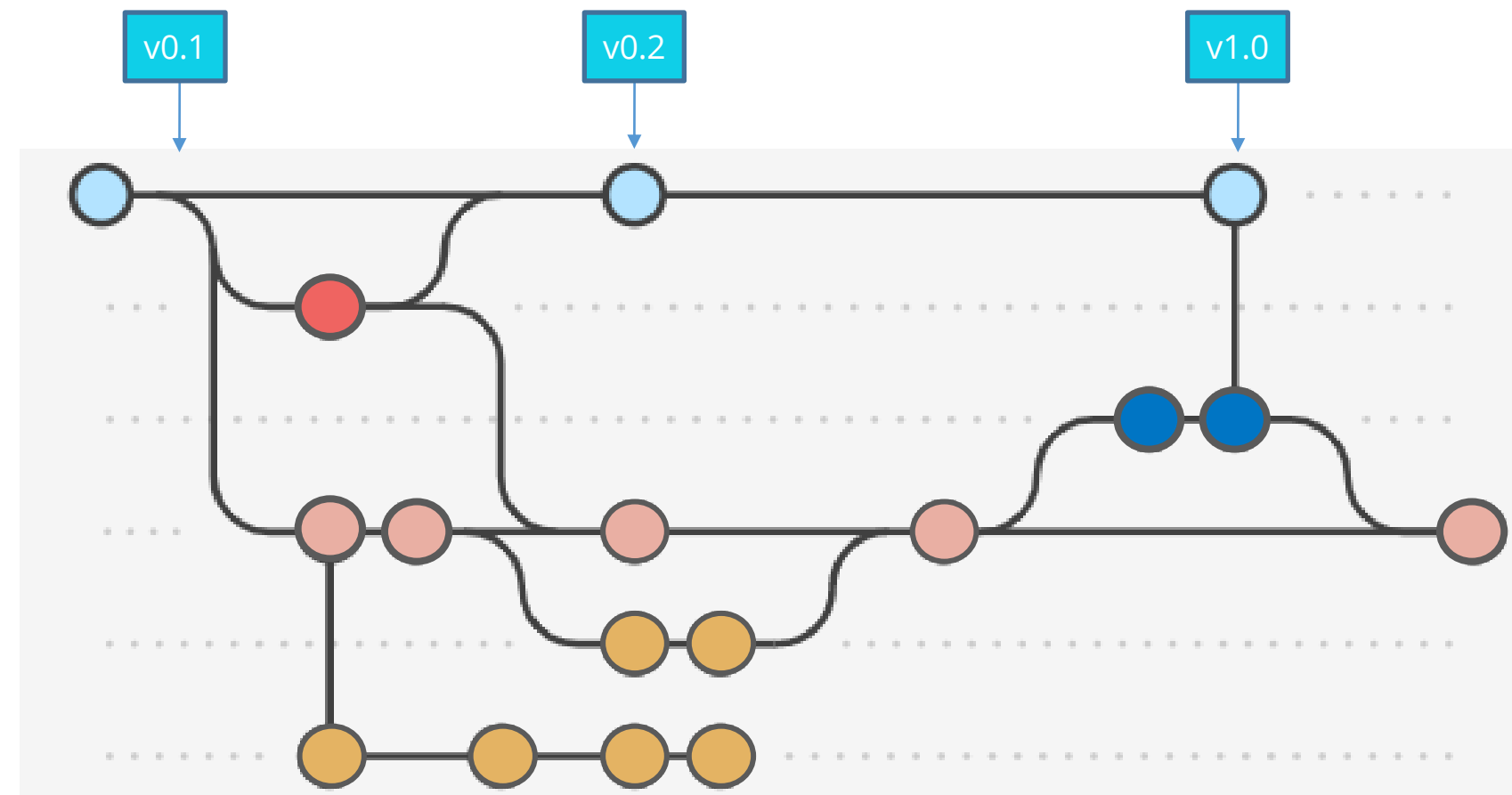
Centralized Workflow

Feature Branch Workflow

Gitflow Workflow

Forking Workflow

Gitflow Workflow



Centralized Workflow

Feature Branch Workflow

Gitflow Workflow

Forking Workflow

Flow of Gitflow:

1. A develop branch is created from master
2. A release branch is created from develop
3. Feature branches are created from develop
4. When a feature is complete, it is merged into the develop branch
5. When the release branch is done, it is merged into develop and master
6. If an issue in master is detected, a hotfix branch is created from master
7. Once the hotfix is complete, it is merged to both develop and master

Forking Workflow

Flow of Forking:

Centralized Workflow

Feature Branch
Workflow

Gitflow Workflow

Forking Workflow

01

A developer forks a copy of the official repository on the server. This new copy serves as their personal public repository.

02

The developer performs a git clone to get a copy of it onto their local machine.

03

The developer creates a new local feature branch. To publish a local commit, they push the commit to their own public repository.

04

The developer files a pull request from the new branch to the main repository.

05

The pull request gets approved for merge and is merged into the original server-side repository.

Assisted Practice

Demonstrate the Centralized Git Workflow

Problem Statement: You are a team lead in an organization that uses centralized git workflow.

Demonstrate the workflow to your coworkers to help them share the project files among themselves.

Steps to Perform:

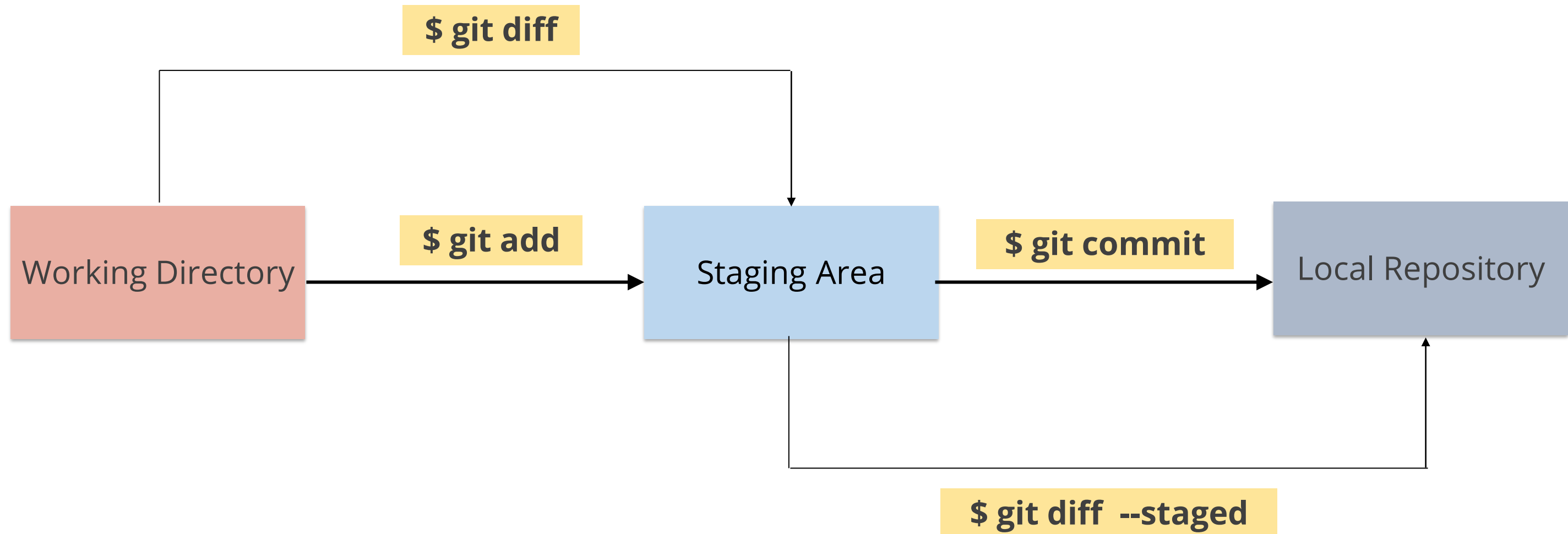
1. Cloning a GitHub repository
2. Adding a file to the cloned repository
3. Checking the status of the repository
4. Adding the file to the staging area
5. Committing the changes and pushing the files to the GitHub repository

Tracking File Changes

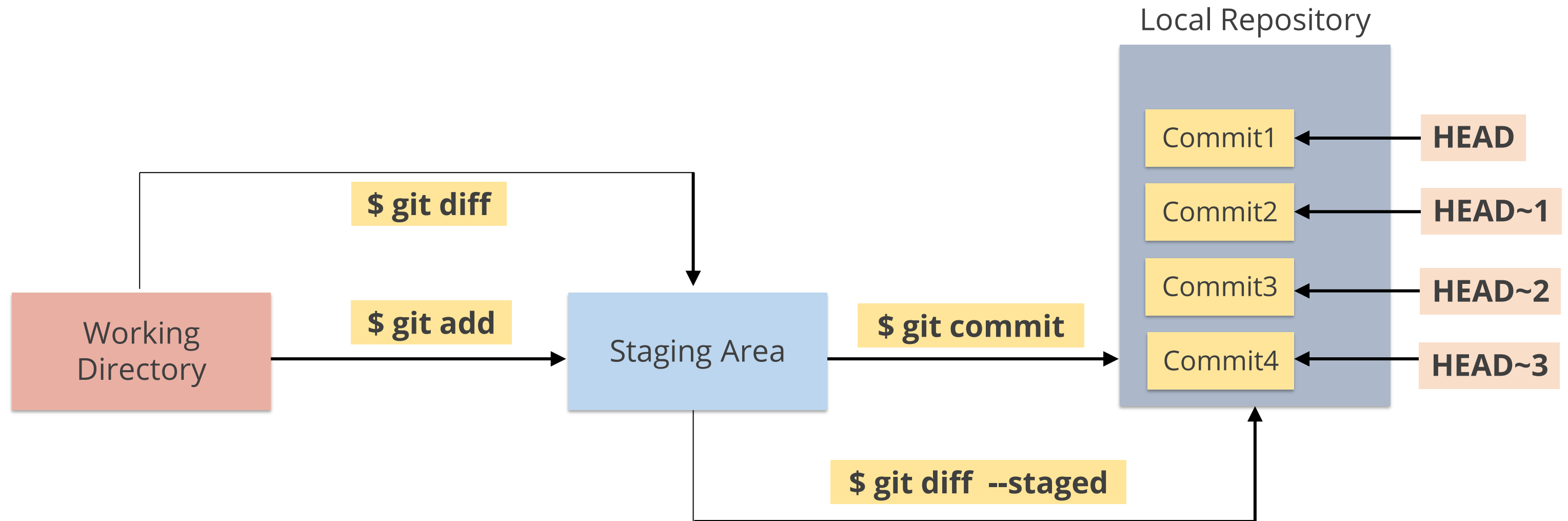
Track Changes between Stages



Tracking is the ability to identify changes between the different versions of the same file, spread across various repositories.



Track Changes between Stages



NOTE

Command **"git diff -staged"** is similar to **"git diff HEAD"**

If the number of commits increase beyond a certain count, use the hashcode command. Example:
\$ git diff <commit hashcode>

Assisted Practice

Tracking File Changes

Problem Statement: Being a Project Lead, demonstrate git's ability to compare different commits and file changes.

Steps to Perform:

1. Updating a file from the repository
2. Tracking the changes in the file
3. Adding the file to the staging area
4. Comparing git commits to track file changes

Reverting to Earlier Commits

Reverting to Earlier Commits

Git checkout deletes the changes from the working directory and puts the directory in a state before the commit happened.

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git checkout -- <file>..." to discard changes in working directory)
```

```
modified:   simpli_file.txt
```

← Modified file

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

```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git checkout -- "simpli_file.txt"
```

```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git status
```

```
On branch master
```

```
Your branch is ahead of 'origin/master' by 1 commit.
```

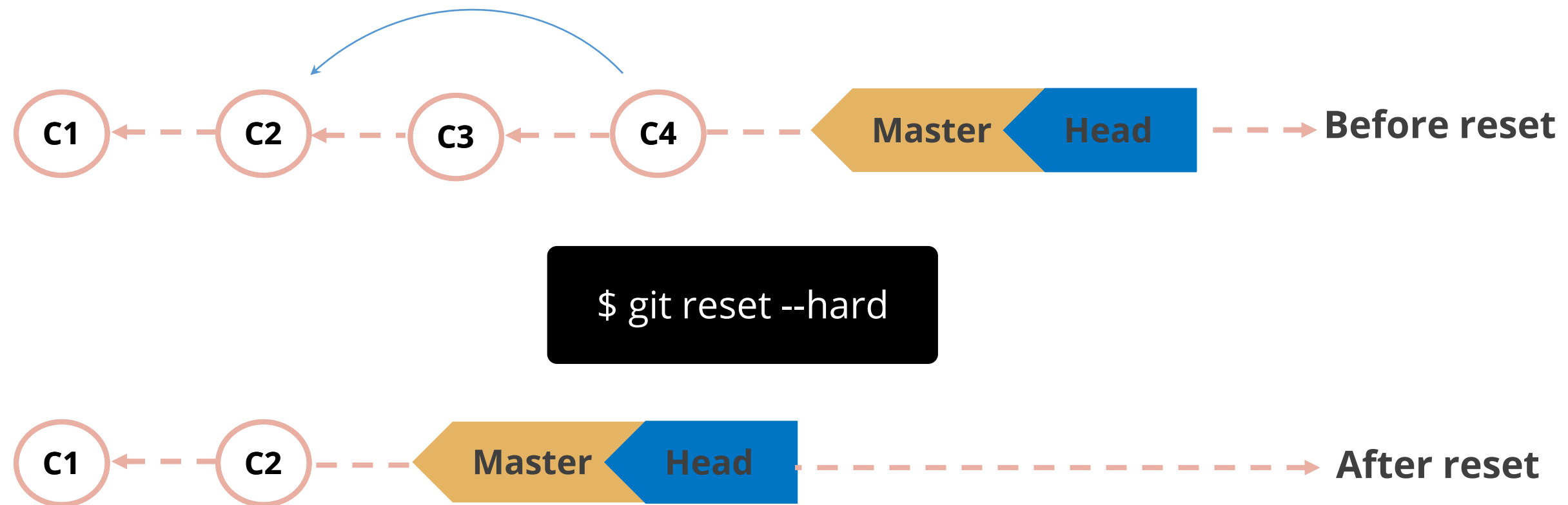
```
(use "git push" to publish your local commits)
```

```
nothing to commit, working tree clean
```





Deletes the changes made to the file

Resetting to Earlier Commits

“git reset --hard” sets an older version of the HEAD pointer. The Staging Index and Working Directory are then reset to match the commit specified.



Steps to Revert to Earlier Commits in Git

-  **\$ git log** List details of the commit associated with a file
-  **\$ git checkout** Stage the file
-  **\$ git status** Share the command to un-stage the file
-  **\$ git reset** Remove the file from the staging area

Assisted Practice

Rolling Back to Previous Commits

Problem Statement: You have updated the project file with some code and have committed the changes in the repository. However, your co-worker needs the previous version of the file to fix some code. Help your co-worker in providing the previous version of the file.

Steps to Perform:

1. Rolling back to a previous commit using reset command
2. Rolling back to a previous commit using revert command

Cleaning the Working Directory

Git Clean



“git clean” removes the working tree’s untracked files.

```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git status
On branch master
Your branch is up to date with 'origin/master'.

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

        CustomerData_ID.txt
        CustomerData_UK.txt
        CustomerData_US.txt

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

C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git clean
fatal: clean.requireForce defaults to true and neither -i, -n, nor -f given; refusing to clean
```

The default git clean command at this point will result in a fatal error.
By default, Git is designed globally to allow the passing of a "power" option to trigger git clean.

Git Clean: Options

git clean -n: Shows the files that will be removed without removing them.

```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git clean -n
Would remove CustomerData_IND.txt
Would remove CustomerData_UK.txt
Would remove CustomerData_US.txt
```

git clean -f or --force: Removes the untracked files from the directory.

```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git clean -f
Removing CustomerData_IND.txt
Removing CustomerData_UK.txt
Removing CustomerData_US.txt
```

git clean -fx: Removes untracked files and any files that Git usually ignores from the current directory.

git clean -d: Git clean will not resort to untracked directories when there is no < path > specified to prevent too much deletion. -d is used to remove such directories.

Assisted Practice

Cleaning the Working Directory

Problem Statement: Demonstrate how to manage and keep the working directory clean.

Steps to Perform:

1. Cleaning the working directory

Updating the Previous Commit

Updating the Previous Commit

“git commit --amend” command let's you fix the very last commit if you want to redo that commit, make the additional changes you forgot, stage them, and commit again.

```
$ git commit -m "fix bug #209"
```

Committed a mistake in your message

```
$ git add simpli_file.txt
```

Add the changes

```
$ git commit --amend -m "fix bug 299"
```

Commit using the **--amend** option with the correct message

Unstaging a File

“git reset --head <file>” command is used to unstage a file.

```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   new_file
        modified:   simpli_file.txt
```

```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git reset HEAD simpli_file.txt
Unstaged changes after reset:
M       simpli_file.txt

C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   new_file

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   simpli_file.txt
```

Before Unstaging

\$ git reset -HEAD
simpli_file.txt

After Unstaging

Assisted Practice

Adding Changes to Your Last Commit

Problem Statement: You forgot to add a line of code in the file that you last committed.
Add the changes to the file and modify the last commit message.

Steps to Perform:

1. Checking the list of files in the repository
2. Opening the file in the vi editor to implement the necessary changes
3. Executing **git status** to check the status of the repository. It shows that the file is being modified
4. Using **git add** command to add the files to the staging area
5. Executing **git commit --amend** to modify the most recent commit
6. Executing **git log --oneline** to check the recent commit with the modified message

Deleting Files in Git

Deleting Files from Staging Area and Working Directory



```
$ git rm <filename>
```

Deletes the file from staging area and working directory



Delete a file only from the staging area: `$ git rm --cached <filename>`



```
$ git status
```

NOTE

This file will be untracked as it is removed from the staging area.

Restoring Deleted Files



```
f684b60 restore the two deleted files a & b  
d4755d2 delete b.txt  
aeaa042 delete a.txt  
4e09d9e initial version of files a and b
```

Repository



```
$ git checkout <version> <file name>
```

Assisted Practice

Deleting Files in Git

Problem Statement: You have a file in the working directory which is not required in your project anymore. Find a solution to remove the file from the staging index and the working directory.

Steps to Perform:

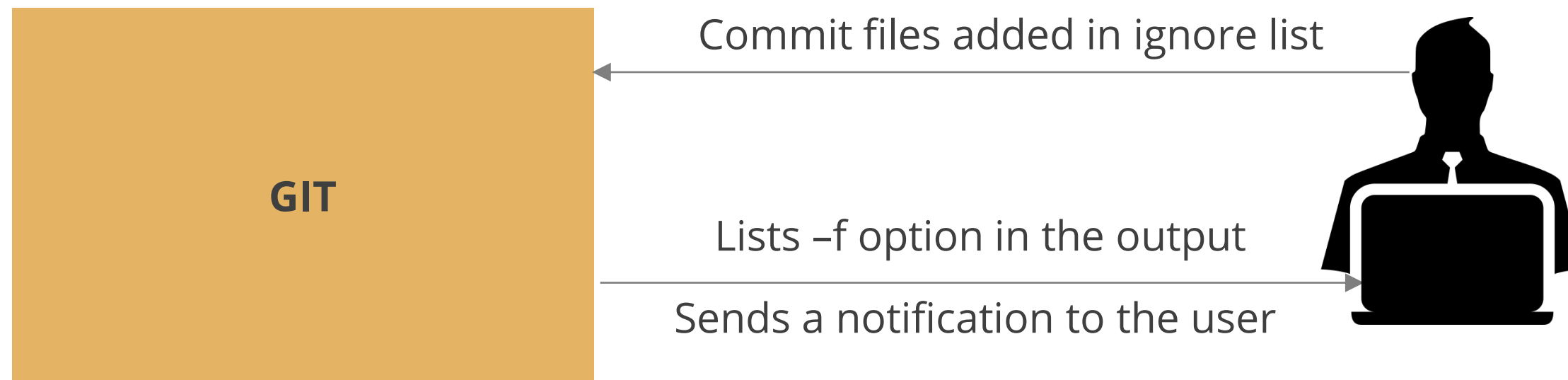
1. Checking the list of files in the repository
2. Removing a file from the repository
3. Checking the list of files again
4. Executing git status to show that the removal of the file is still in staging and needs to be
5. committed to the repository
6. Commit the changes

Ignoring Files in Git

How to Ignore Files in Git



The file name and file pattern can be overridden using the **-f flag**.



Assisted Practice

Ignoring Files in Git

Problem Statement: Your project consists of files that are generated by your IDE or other tools that aren't needed. You need to find a solution to ignore those unnecessary files or folders.

Steps to Perform:

1. Creating the .gitignore file
2. Checking the git status

Renaming Files in Git

Steps to Rename Files in Git

1



```
$ git mv <filename> <new-filename>
```

2



```
$ git add
```

```
$ git commit
```

3



```
$ git status
```

Assisted Practice

Renaming Files in Git



Problem Statement: Demonstrate how to rename files in Git.

Steps to Perform:

1. Execute **git mv <old file name> <new file name>** to rename the file
2. Check the status of the repository
3. Commit the changes

Git vs. Other Version Control Systems

Git vs. Other Version Control Systems

 Git	Other Version Controls Systems 
The users or the team maintains its own repository, instead of working from a central repository.	The users or the team uses a central code repository model.
The changes are stored as patches or can be characterized as sets.	The storage of changes in the central repository depends on the user's interest.
They focus on patches or change sets as a discrete unit that can be exchanged between repositories.	They track changes from version-to-version of different files or states of the directory.
They do not require central server.	They need a central server.
There is a no single point of failure.	There is a single point of failure.

Migration from SVN to Git

SVN: Definition



- It is a centralized version control system
- It is an open-source characterized by its reliability
- It supports the needs of a wide variety of users and projects
- It has all the source files and versions of the files

Drawback

It has a tedious branching model which adds complexity implementing a branch strategy

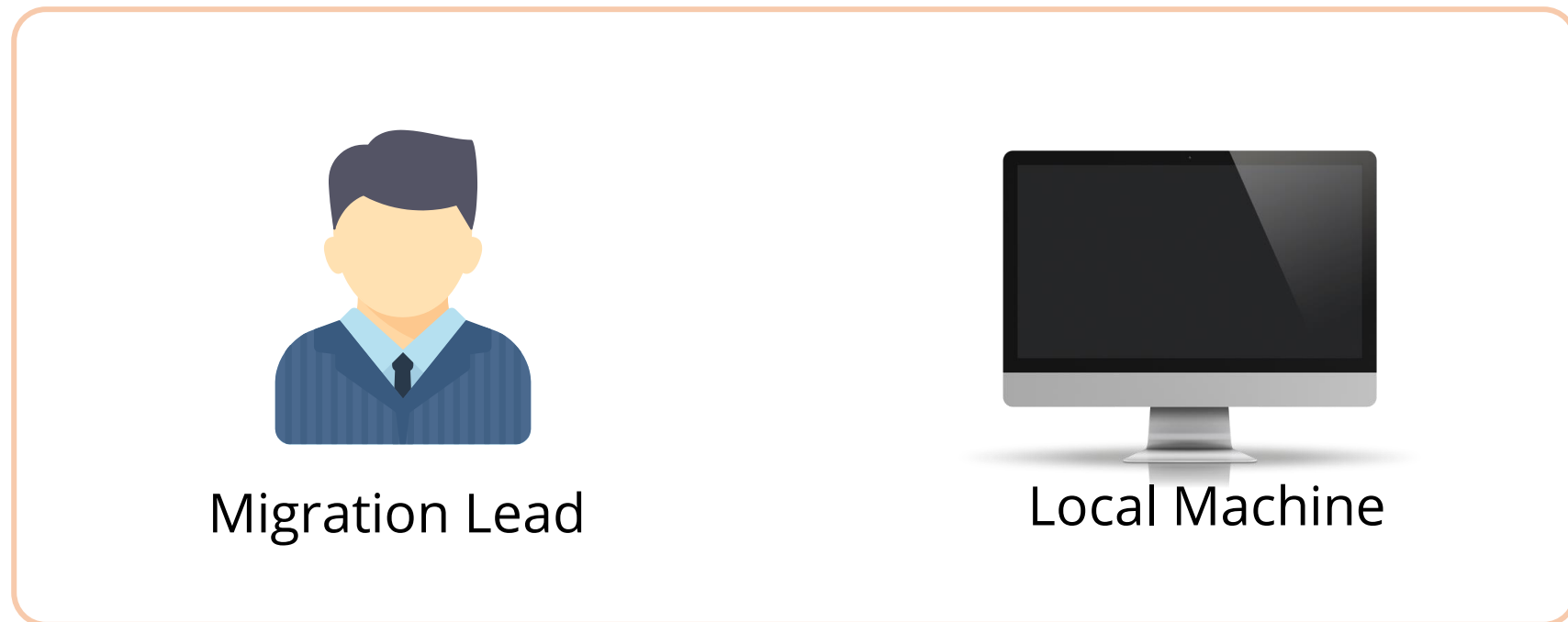
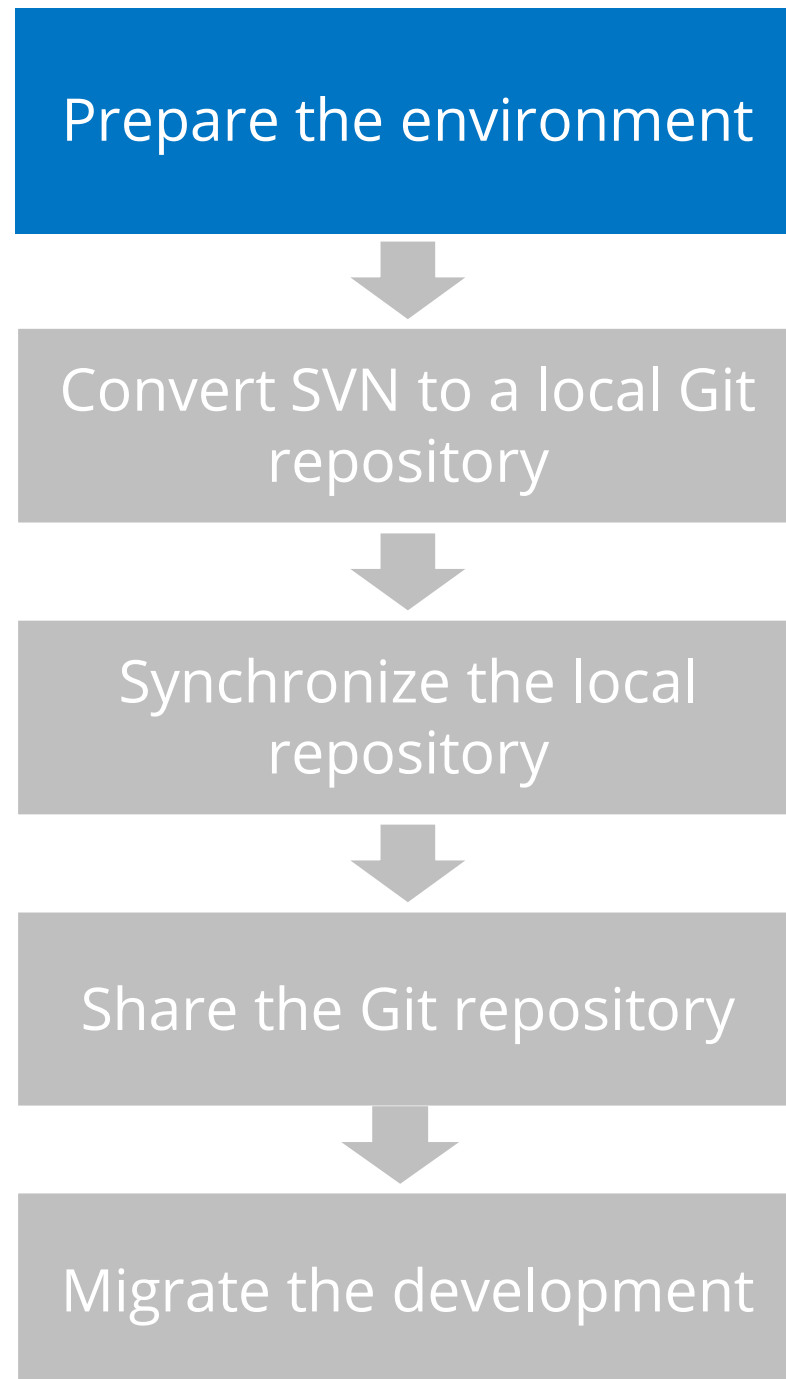
Git vs. SVN

Git	SVN
Used by 70% of the developers	Used by 25% of the developers
Supports distributed version control	Supports centralized version control
Can work locally (offline)	Must be connected to commit
Each user has a copy of the full repository	Each user has a copy of only the trunk
Easy to fork, branch, and merge	Easy to branch and merge



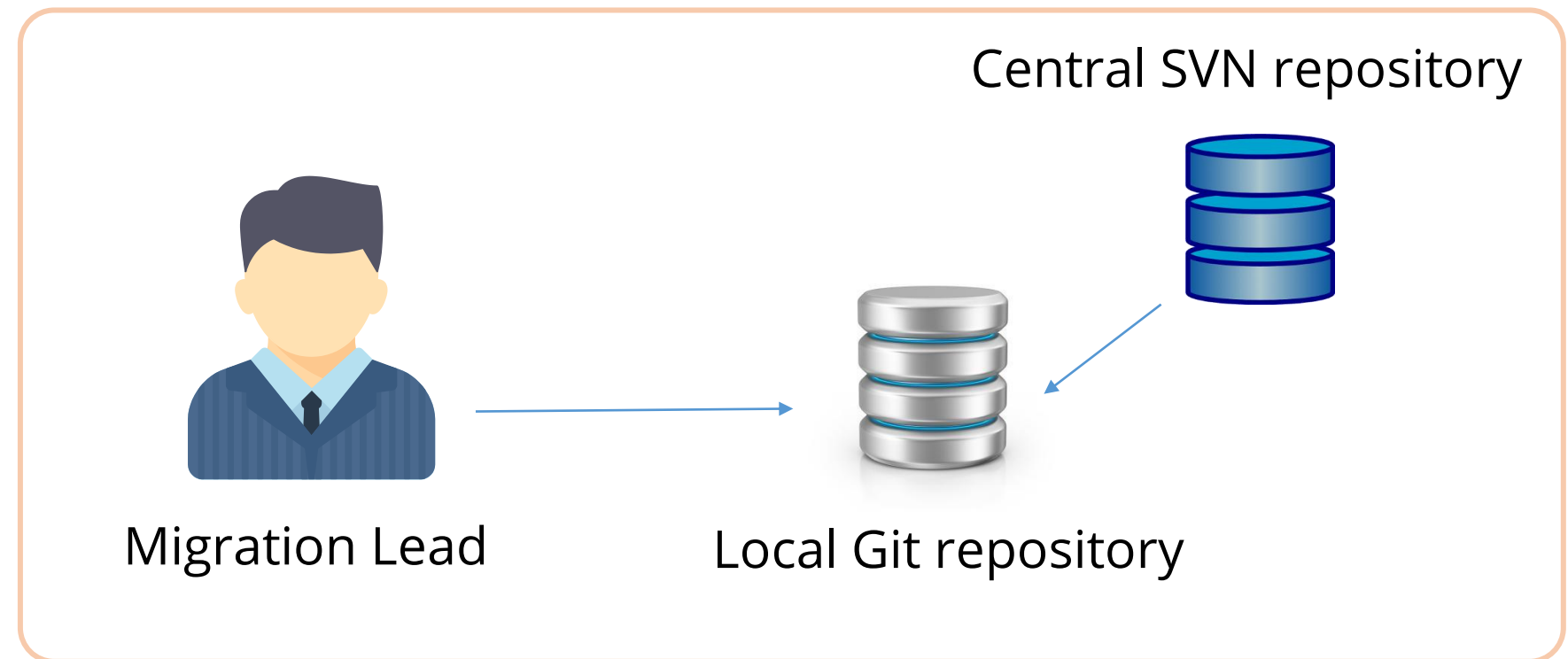
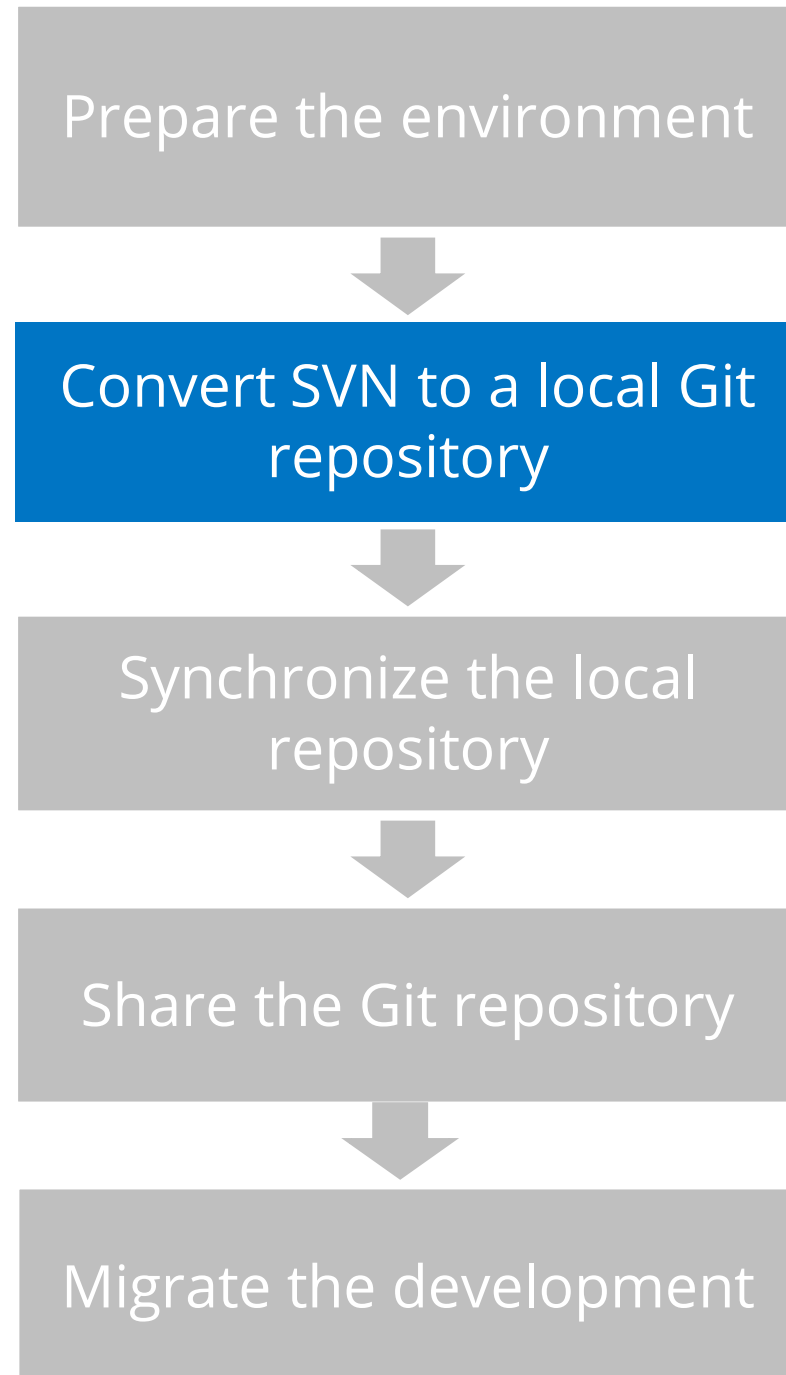
Migrate to Git from SVN

The process involves five steps:



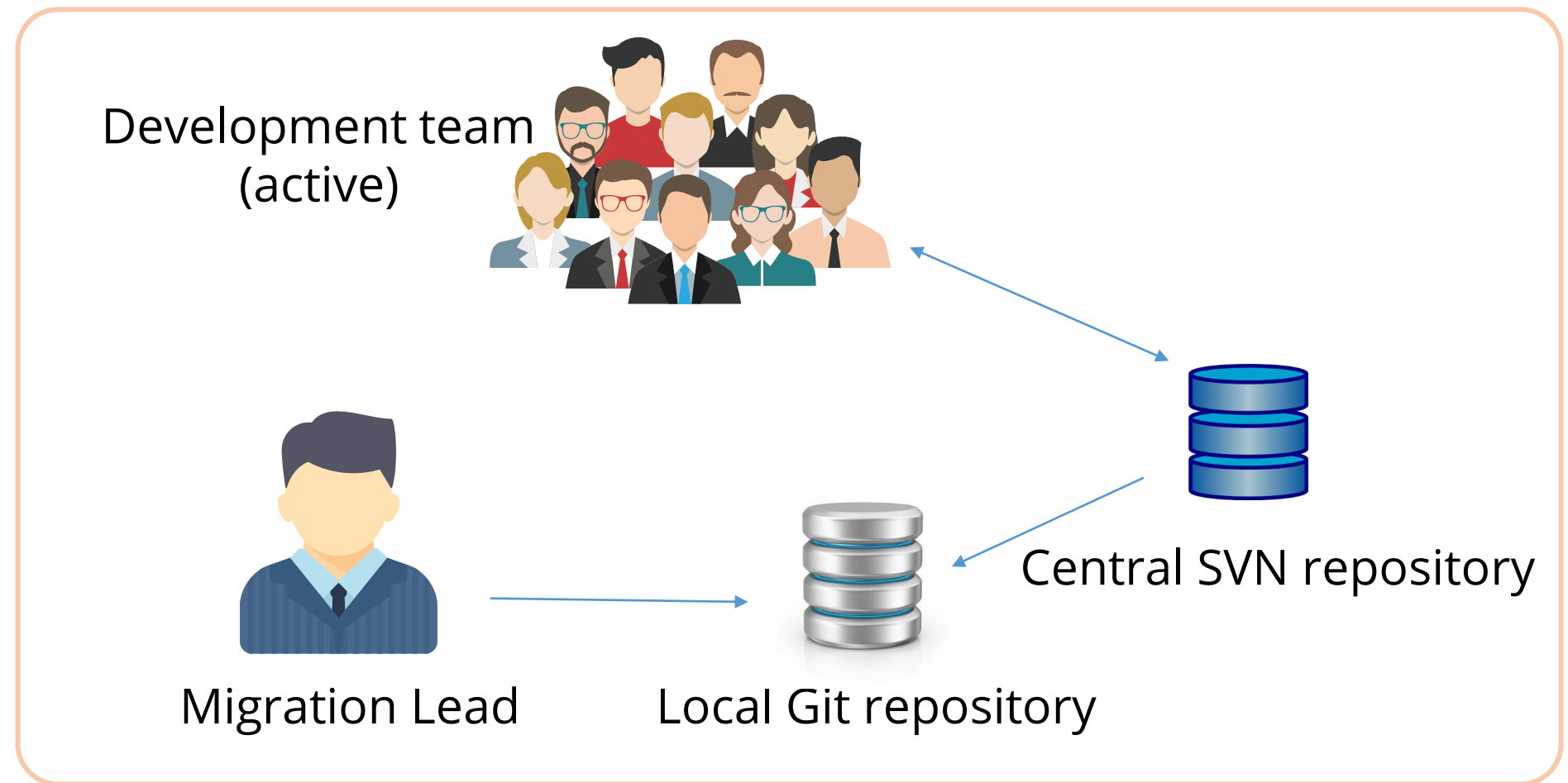
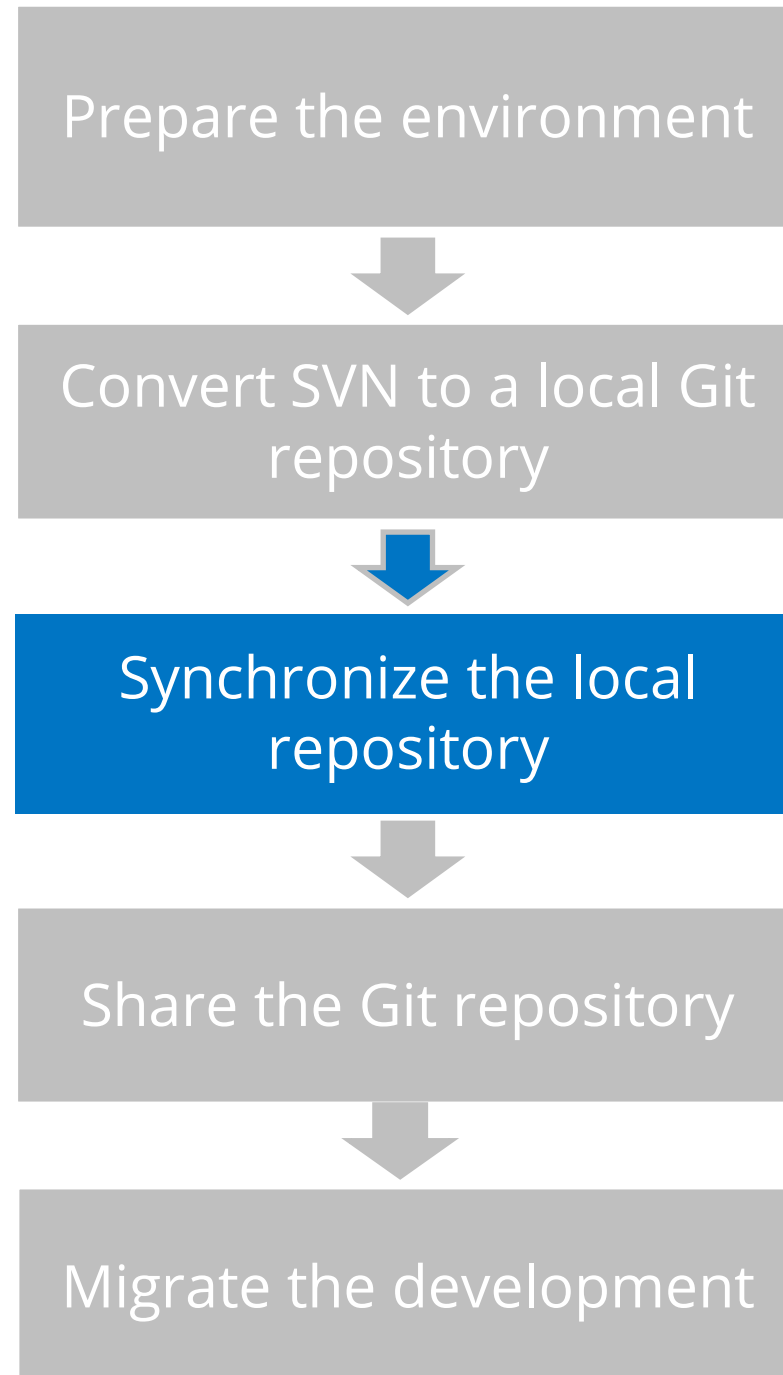
Migrate to Git from SVN

The process involves five steps:



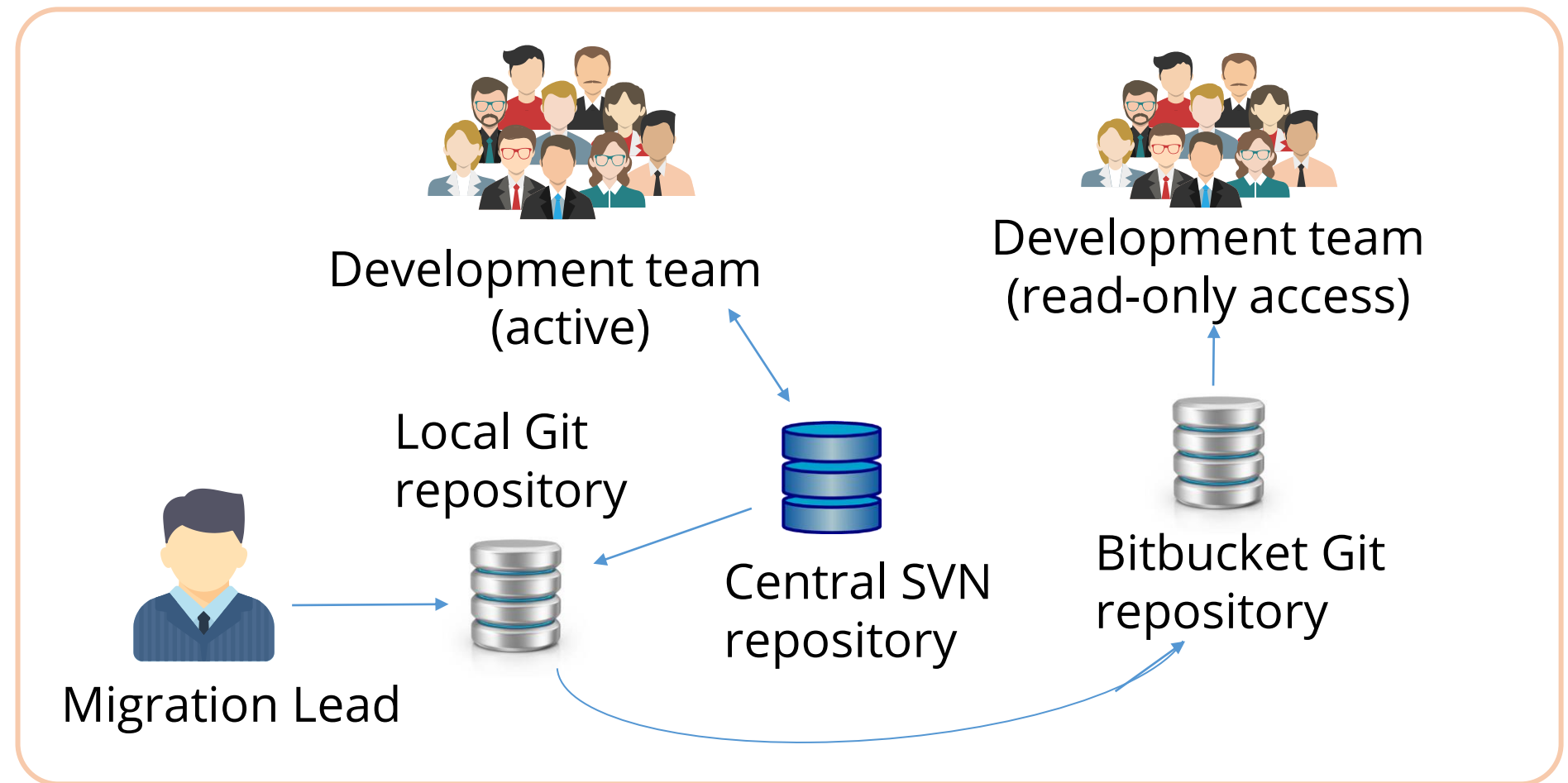
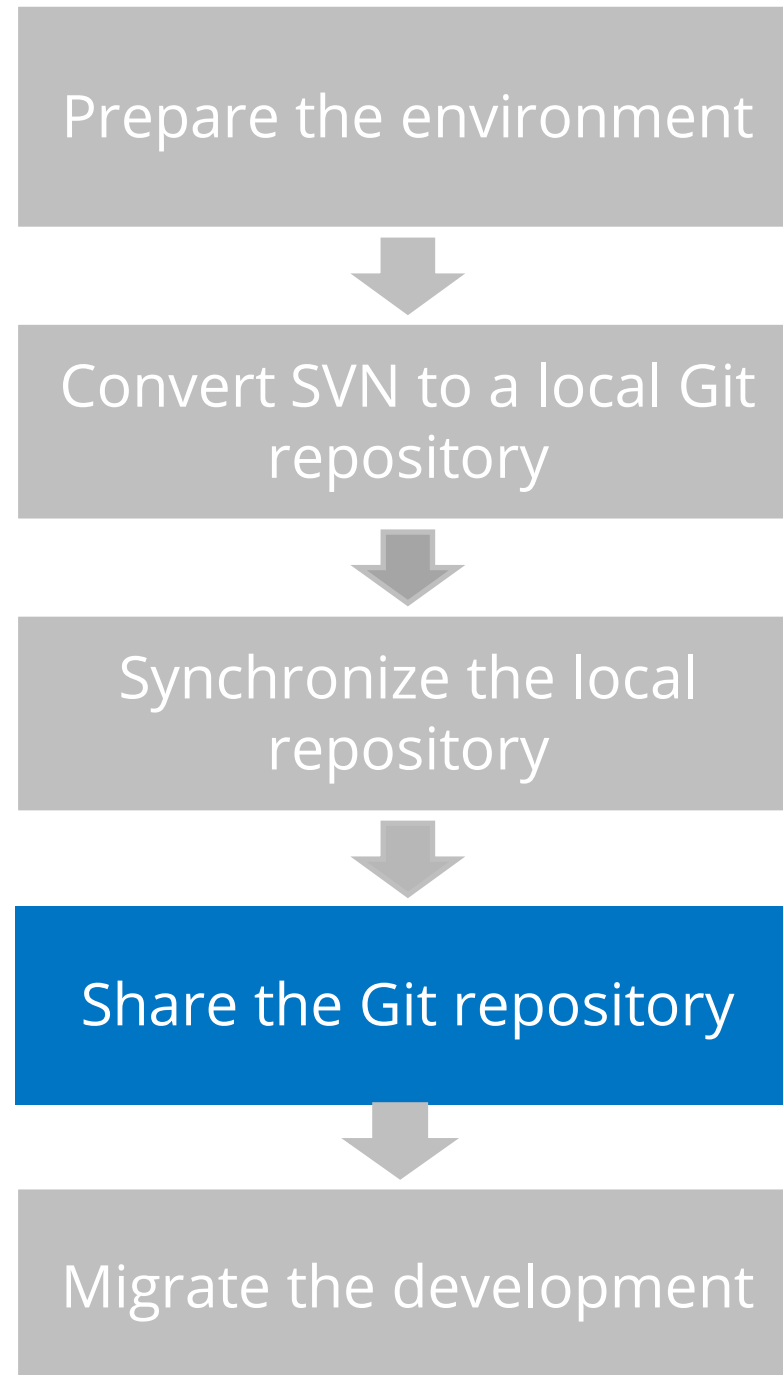
Migrate to Git from SVN

The process involves five steps:



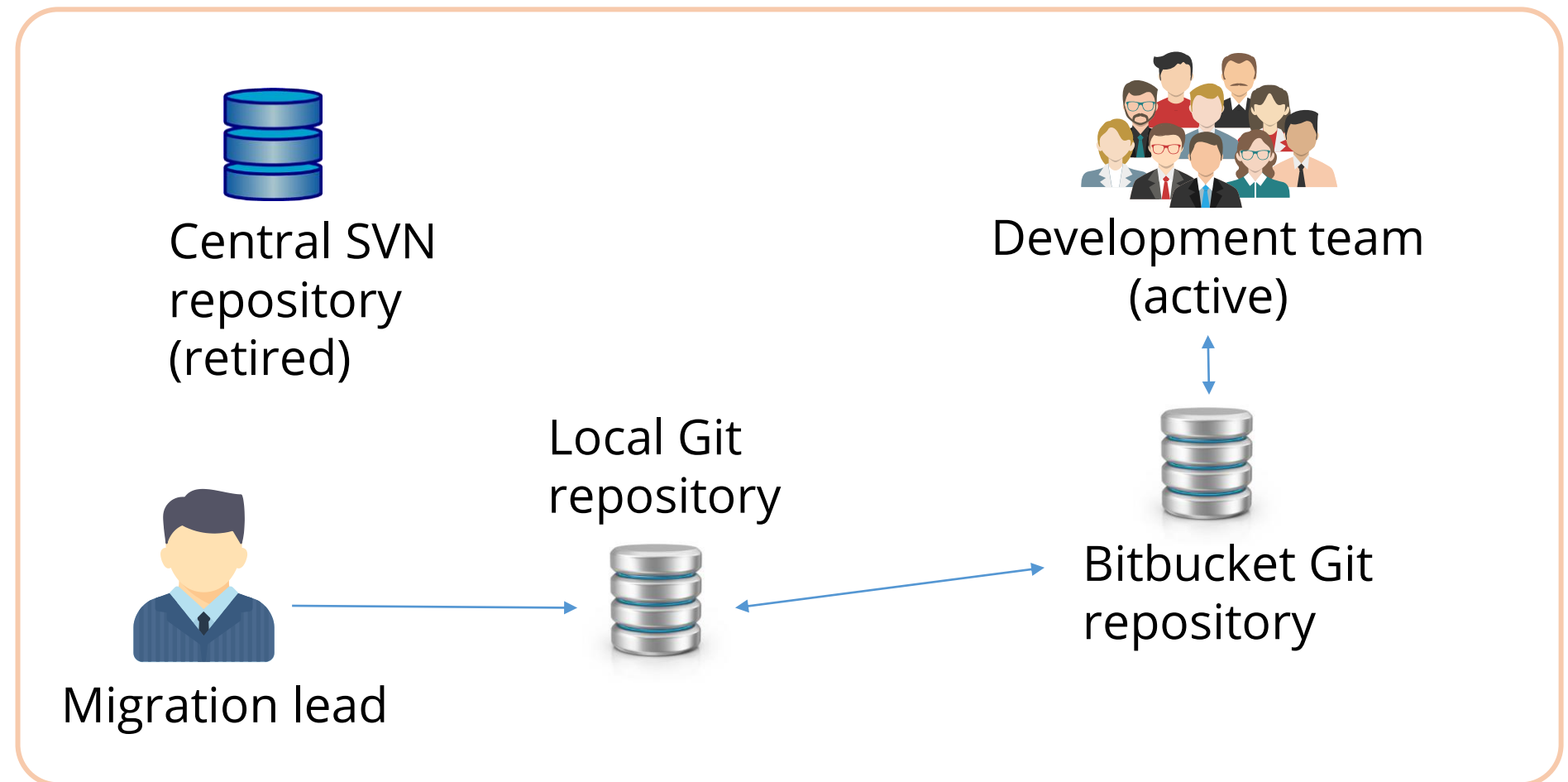
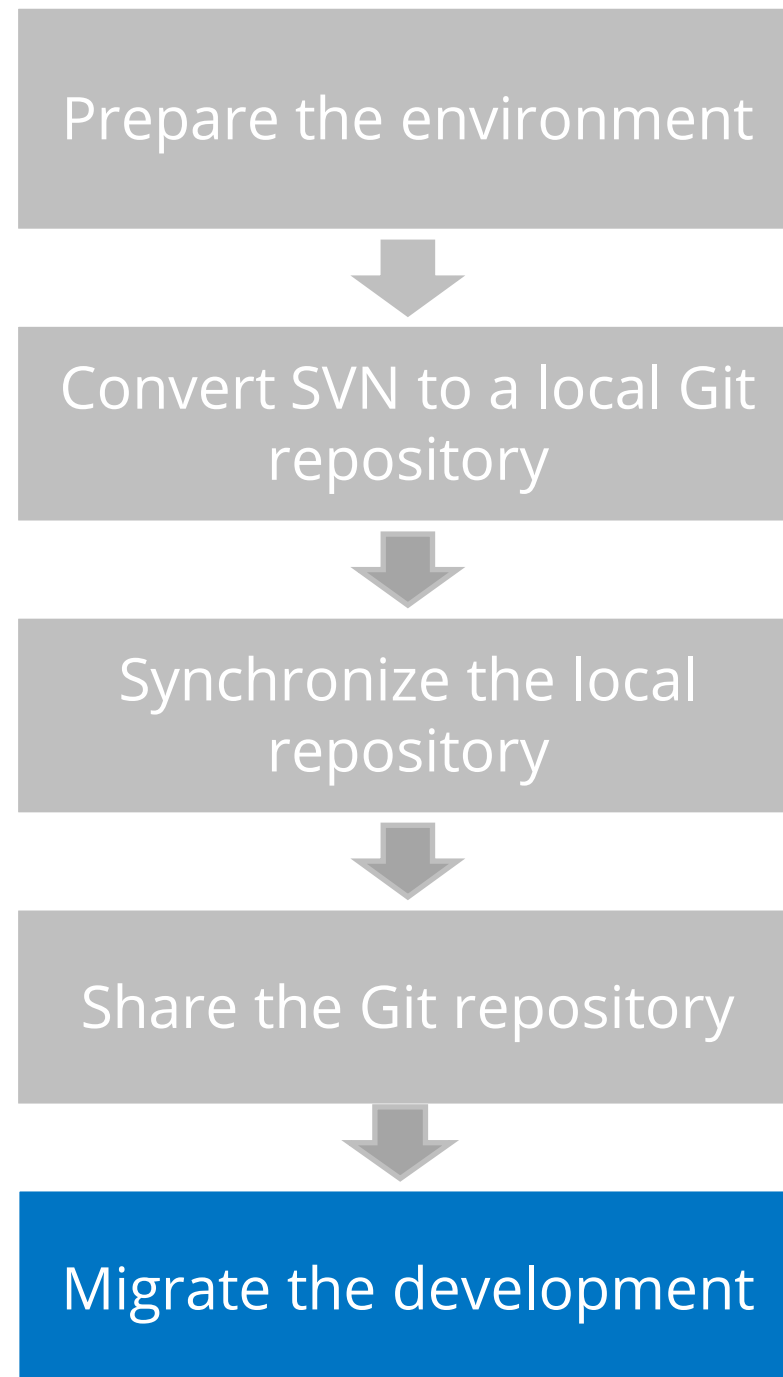
Migrate to Git from SVN

The process involves five steps:



Migrate to Git from SVN

The process involves five steps:



Key Takeaways

- Git init creates a new Git repo, can be used to convert an existing, unversioned project to a Git repo, or initialize a new, empty repo.
- Tracking is the ability to identify changes between the different versions of the same file, spread across various repositories
- “git commit --amend” command lets you fix the very last commit if you want to redo that commit, make the additional changes you forgot, stage them, and commit again.



Lesson-End Project

Create and Work with a Local Git Repository

Problem Statement: Your organization has been planning to adopt Git as a code management standard. Being a project manager and a DevOps practitioner, you have been advised to train your team on Git to accomplish the work in a consistent and productive manner.

You must use the following:

Git: To create and work with a local Git repository

