Git and GitHub Training

simplilearn

Getting Started with Git

Learning Objectives

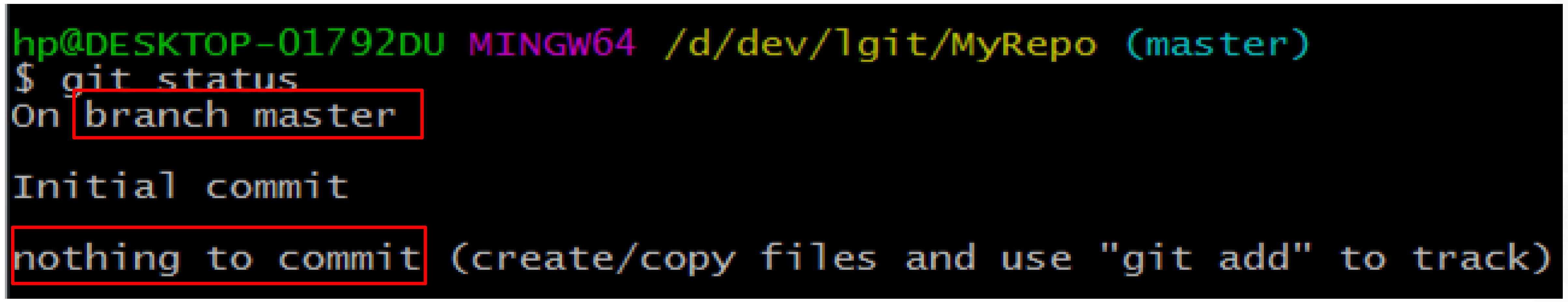
By the end of this lesson, you will be able to:

- Create a Git Repository
- Explain the different types of Git workflows
- Demonstrate various operations like tracking, reverting, deleting, ignoring, and renaming files in Git

Creating a Git Repository

Steps to Create a Git Repository

Create an empty directory	Command: **$ mkdir <directory-name>**
Convert the directory into a repository	Command: **$ git init**
Run git status	Command: **$ git status**



```
hp@DESKTOP-01792DU MINGW64 /d/dev/lgit/MyRepo (master)
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

NOTE

$ git init command creates a hidden directory that stores all the metadata required for it to function.

Create a Git Repository

Problem Statement: To create a Git repository to share the project files with your coworkers.

Steps to Perform:

1. Create a repository
2. Add files (README, project files) to the repository
3. Commit changes

ASSISTED PRACTICE

Clone a GitHub Repository

Problem Statement: Clone the GitHub repository shared by your coworker to access the project files.

Steps to Perform:

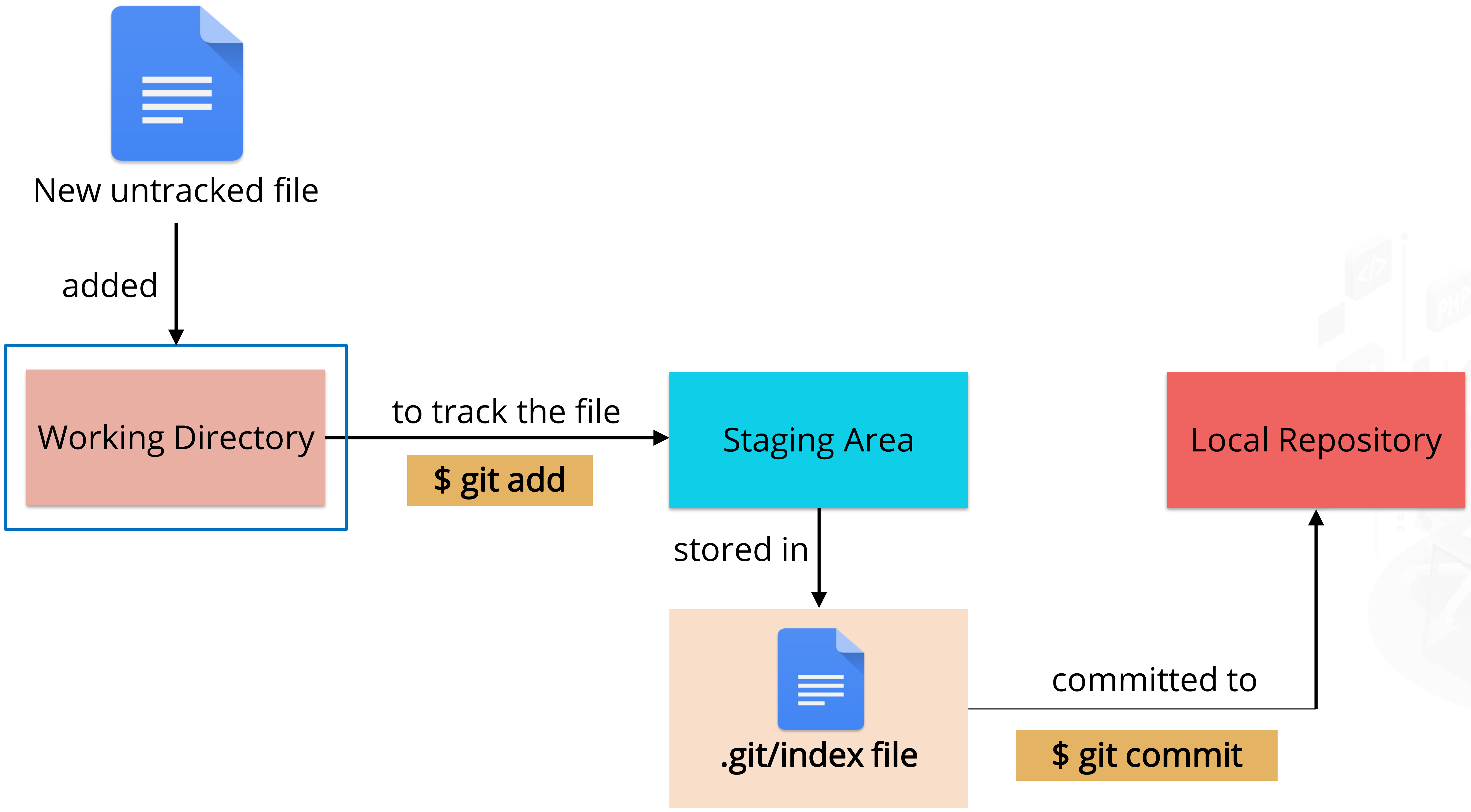
1. Clone the repository
2. Add files in the repository
3. Commit the changes
4. Push the commits to the primary repository

ASSISTED PRACTICE

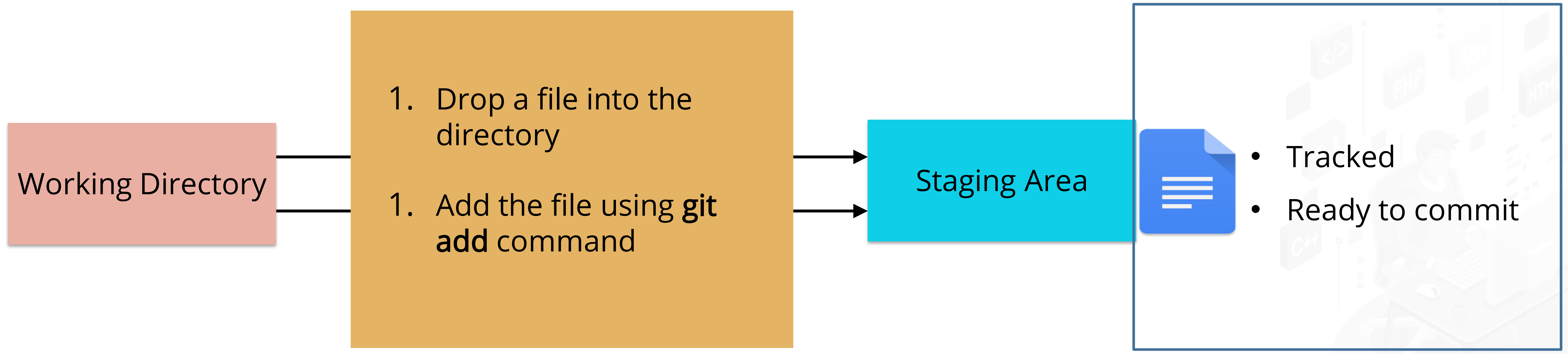
FULL STACK

Git Workflow

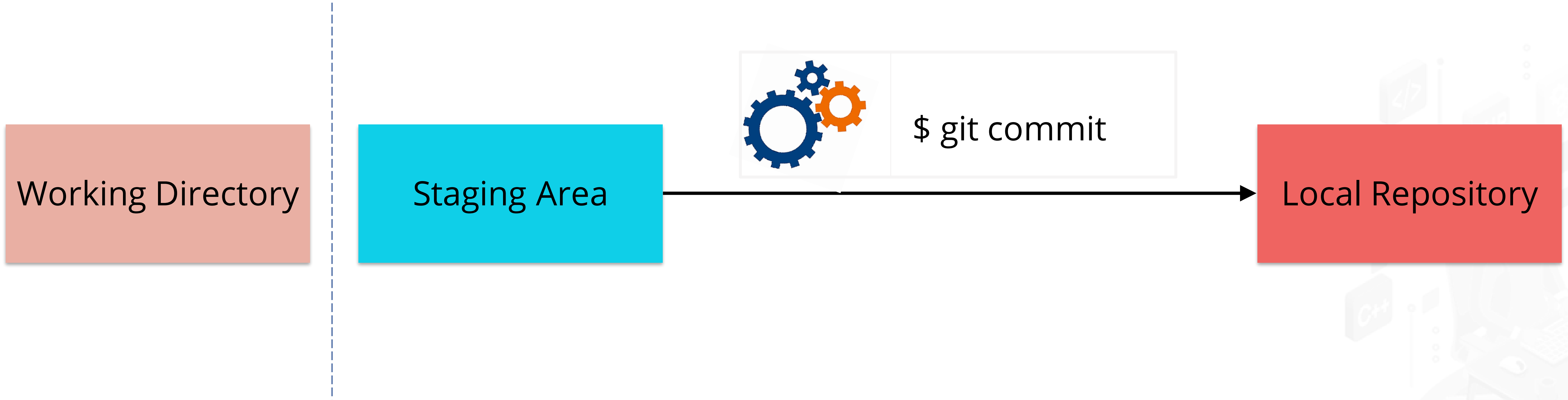
Git’s Three-Stage Workflow



Working Directory to Staging Area


Working Directory
1. Drop a file into the directory
1. Add the file using **git add** command
Staging Area
• Tracked
• Ready to commit

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

To modify a file in Git → Update → Add → Commit

$ git log → View the changes in a file

- Commit ID
- Author
- Date
- Commit Message

NOTE

To restrict the output to one-line, execute: **$ git log --oneliner**, this will display the information in a single line.

Types of Git Workflow

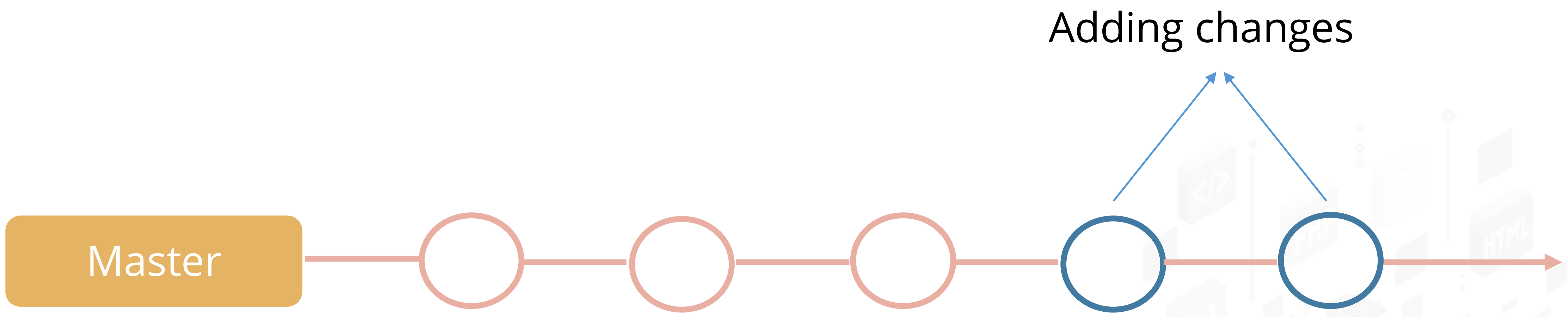
01 Centralized Workflow

02 Feature Branch Workflow

03 Gitflow Workflow

04 Forking Workflow

Centralized Workflow



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

Centralized Workflow

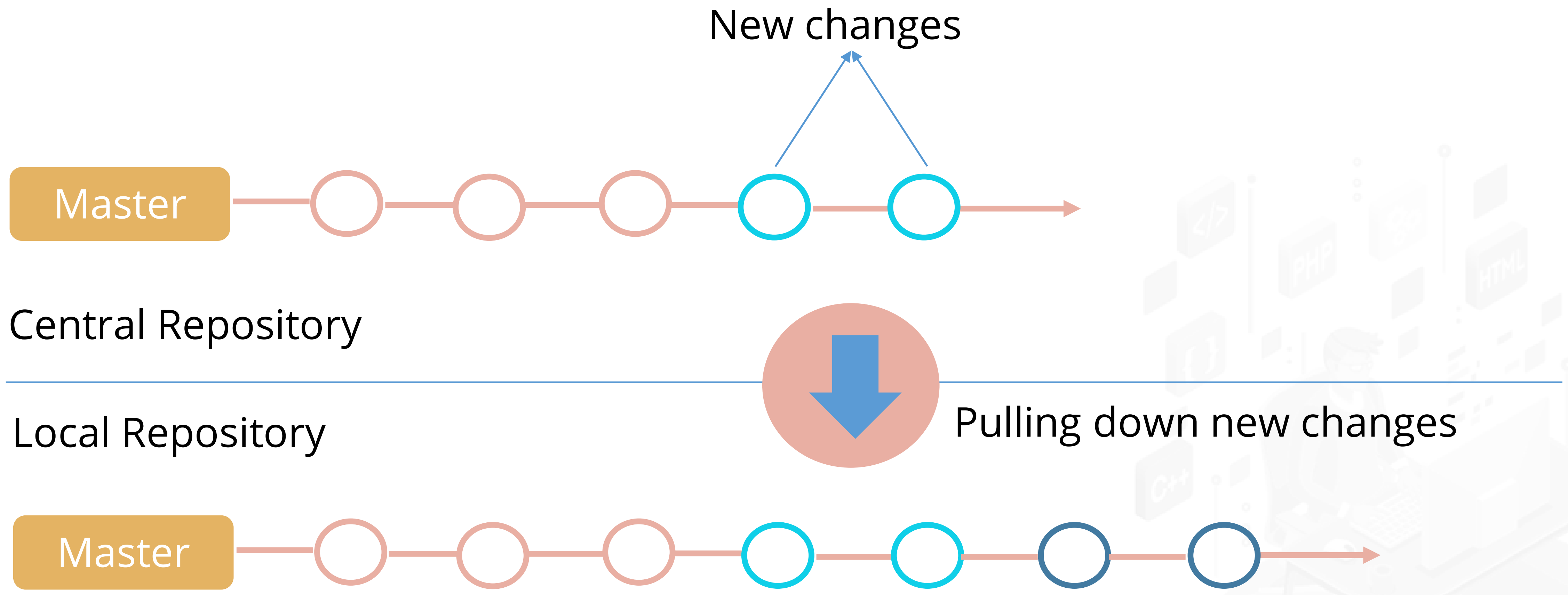
Feature Branch Workflow

Gitflow Workflow

Forking Workflow

Centralized Workflow

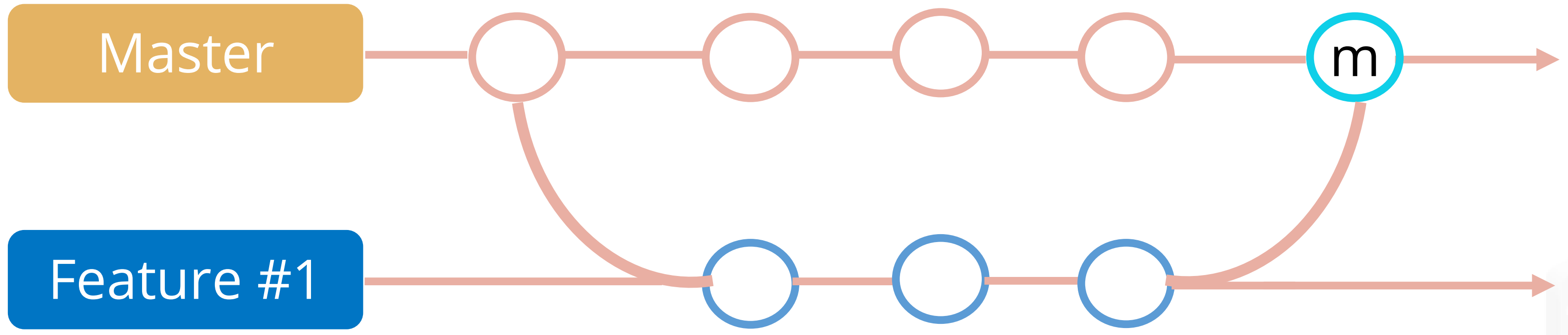
Sync with central repository:



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.

Feature Branch 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.

Centralized Workflow

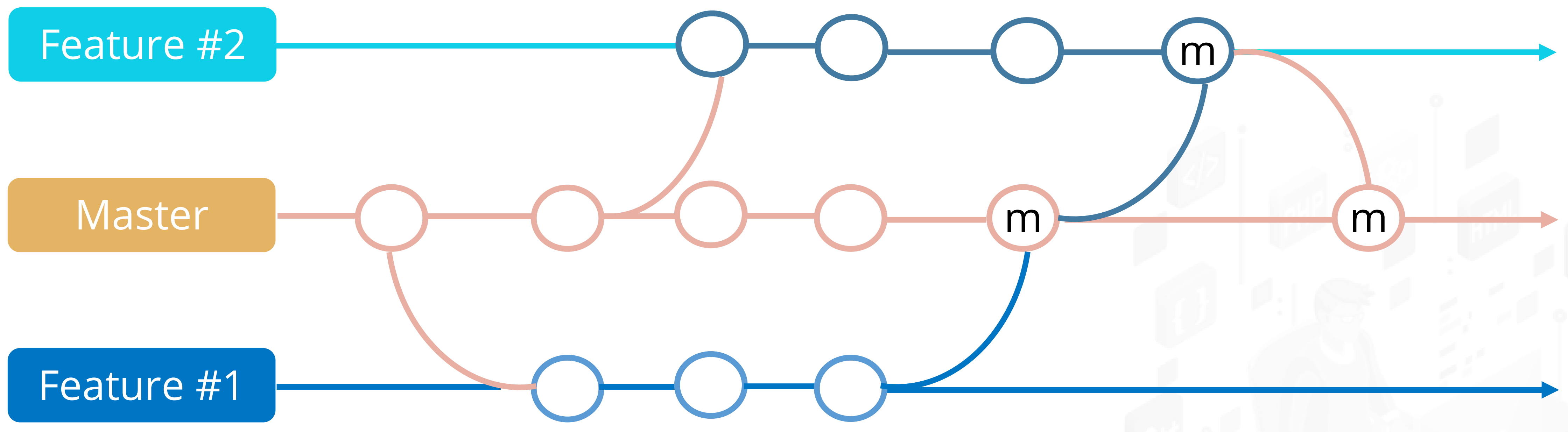
Feature Branch Workflow

Gitflow Workflow

Forking Workflow

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.

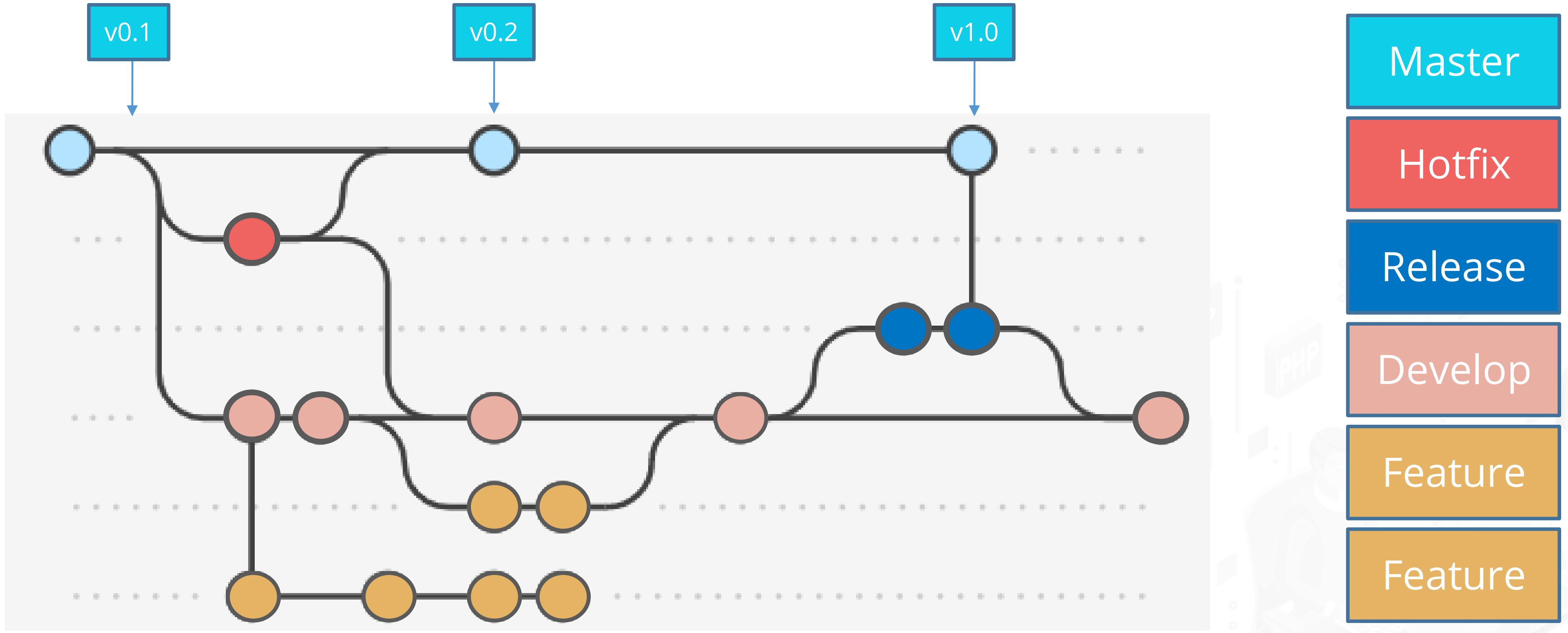
Centralized Workflow

Feature Branch Workflow

Gitflow Workflow

Forking Workflow

Gitflow Workflow



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

Centralized Workflow

Feature Branch Workflow

Gitflow Workflow

Forking Workflow

Forking Workflow

Flow of Forking:

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.

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. Create a file in the local repository
2. Check the status
3. Use **git add** to add the file
4. Commit the file
5. Check the status again
6. Update the file
7. Add the file
8. Check the status and commit the file again

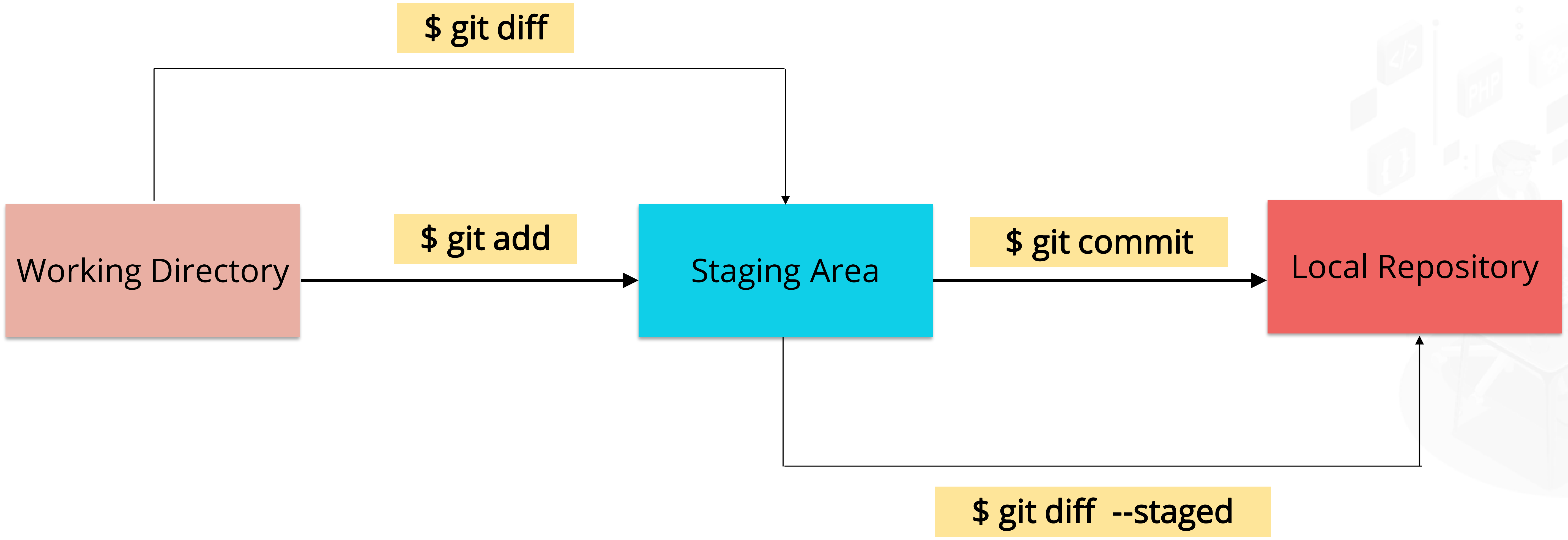
FULL STACK

Tracking File Changes

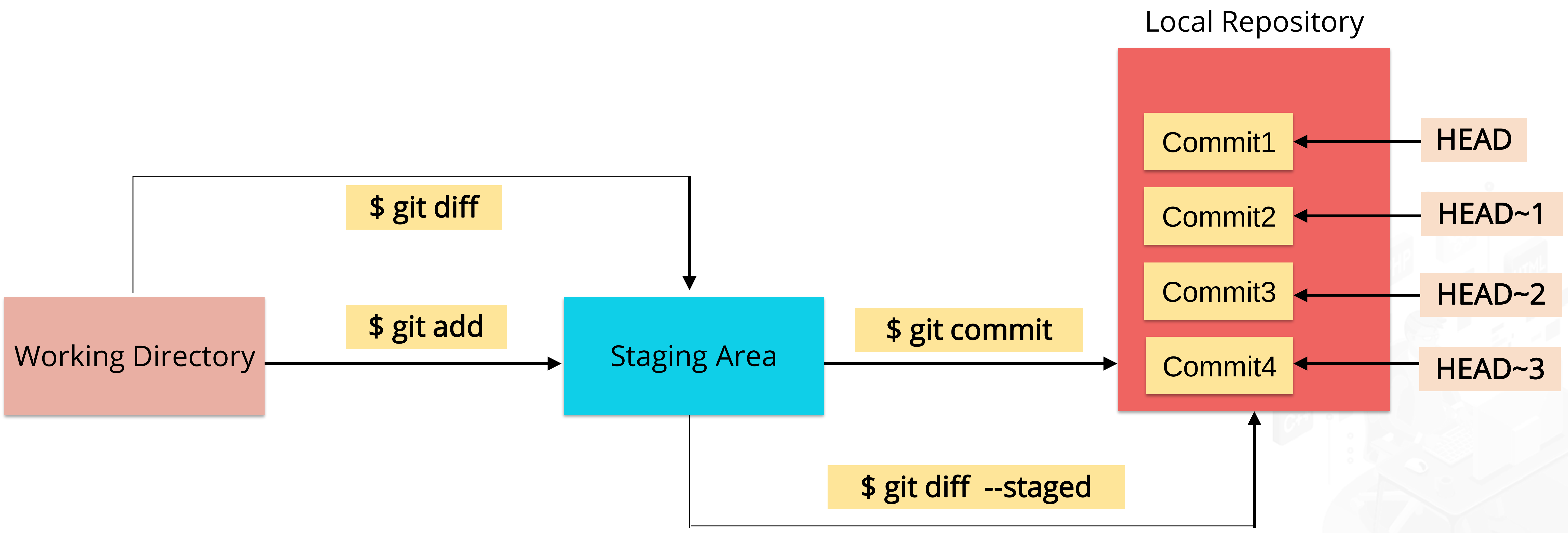
simplilearn

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

1. Command "**git diff –staged**" is similar to "**git diff HEAD**"
2. If the number of commits increase beyond a certain count, use the hashcode command. Example: **$ git diff <commit hashcode>**

Tracking File Changes

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

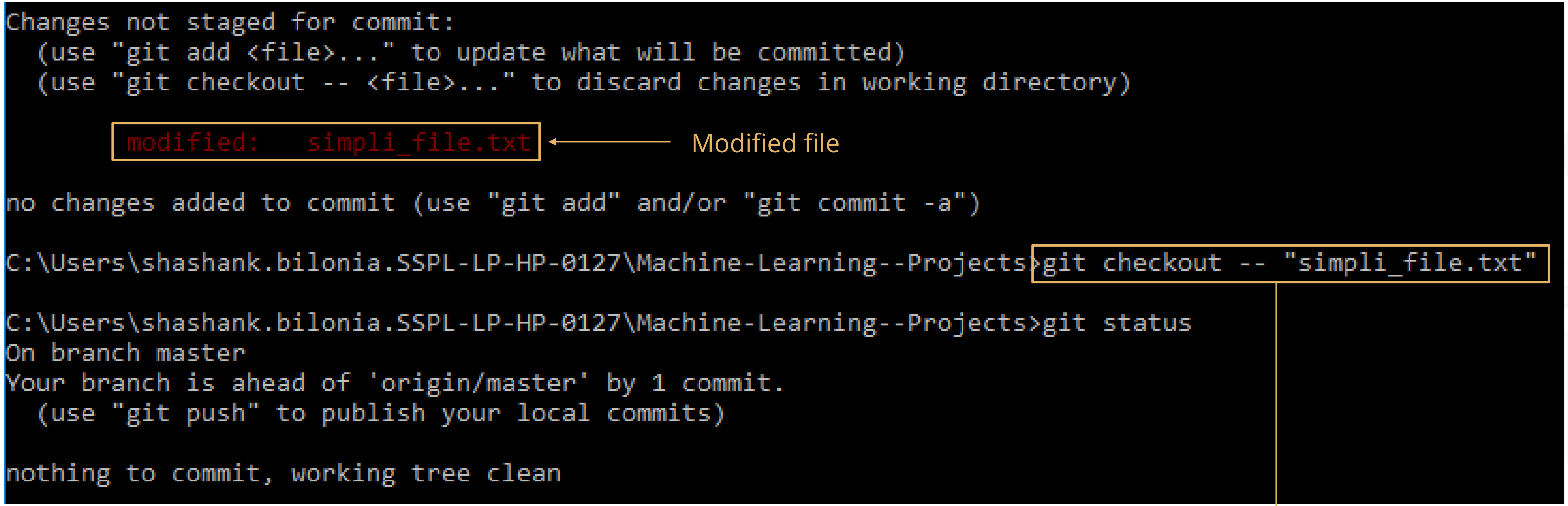
Steps to Perform:

1. Check the git status
2. Open a file from your local repository and make some changes
3. Execute **git diff index.html** command to compare the file
4. Add the file to the staging area
5. Execute **git log --oneline** to check the recent log of commits
6. Execute the **git diff --staged HEAD**
7. Execute **git diff --staged HEAD**~1 to refer to the commit prior to that.
8. Check the git status and commit the changes.

Reverting to Earlier Commits

Revering to Earlier Commits

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

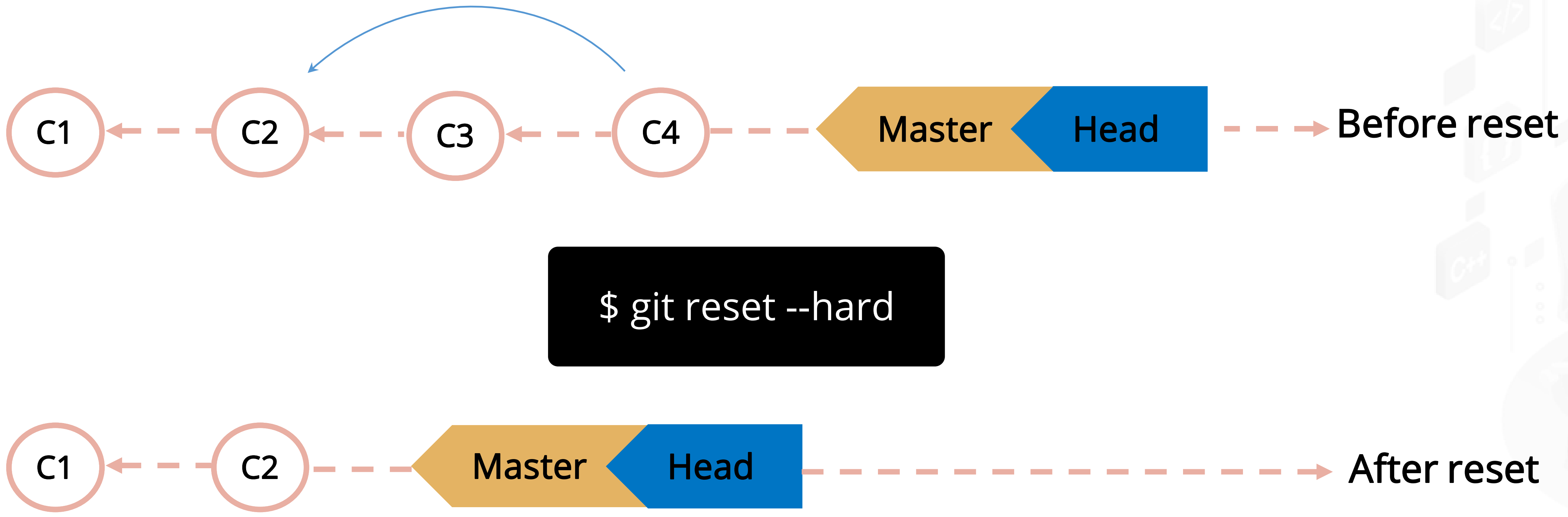


```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   simpli_file.txt

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.



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

Steps to Revert to Earlier Commits in Git

1	$ git log	List details of the commit associated with a file
2	$ git checkout	Stage the file
3	$ git status	Share the command to un-stage the file
4	$ git reset	Remove the file from the staging area

Reverting to Earlier 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.

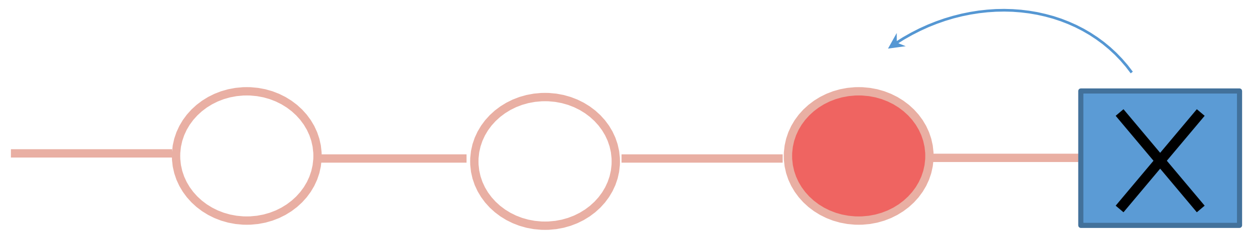
Steps to Perform:

1. Check the log of the recent commits
2. Execute **git checkout HEAD~1 <file name>** to revert to the previous commit
3. Check the content of the file
4. Check the git status
5. Remove the file from the staging area and bring it back to the working directory

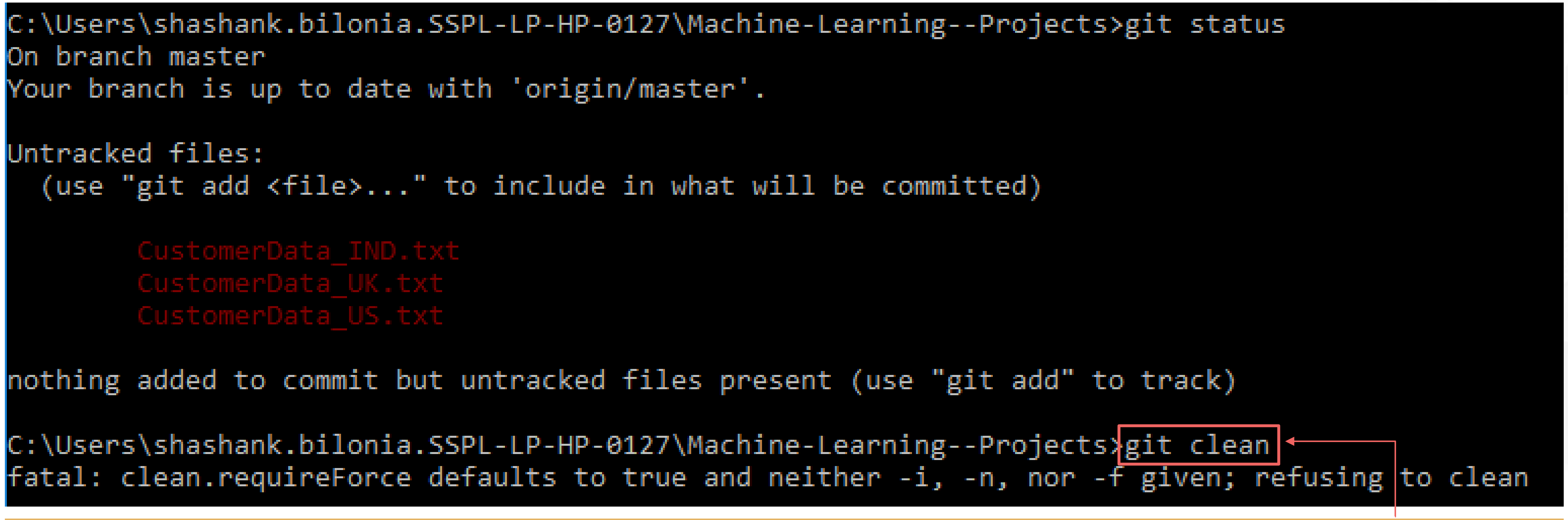
ASSISTED PRACTICE

Cleaning the Working Directory

Git Clean



"git clean" removes the working tree's untracked files.

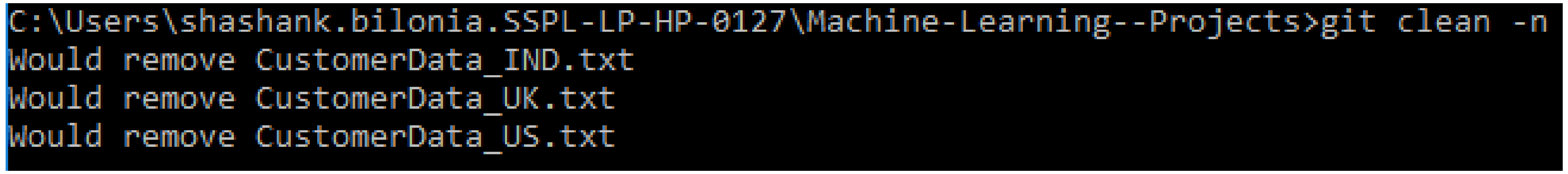


```
C:\Users\shashank.bilonia.SSPL-LP-HP-0127\Machine-Learning--Projects>git status
On branch master
Your branch is up to date with 'origin/master'.

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

        CustomerData_IND.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.

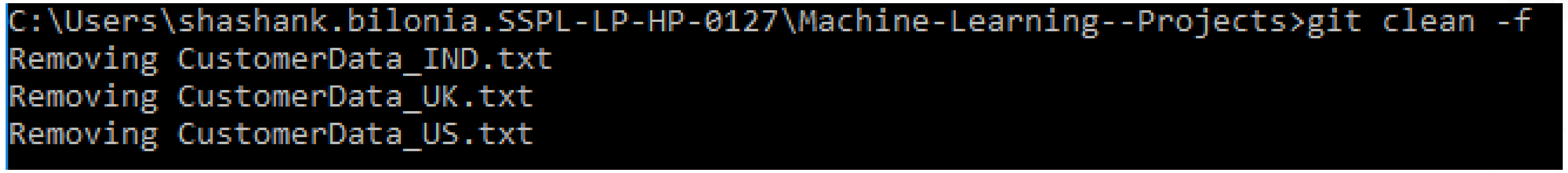
Git Clean: Options

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 –xf: 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.

Cleaning the Working Directory

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

Steps to Perform:

1. Create a new repository
2. Create a new file
3. Additionally, create another file and a directory
4. Check the git status
5. Execute various **git clean** options

ASSISTED PRACTICE

Undoing Things in Git

Fixing the Last 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”

$ git add simpli_file.txt

$ git commit –amend –m “fix bug 299”
```

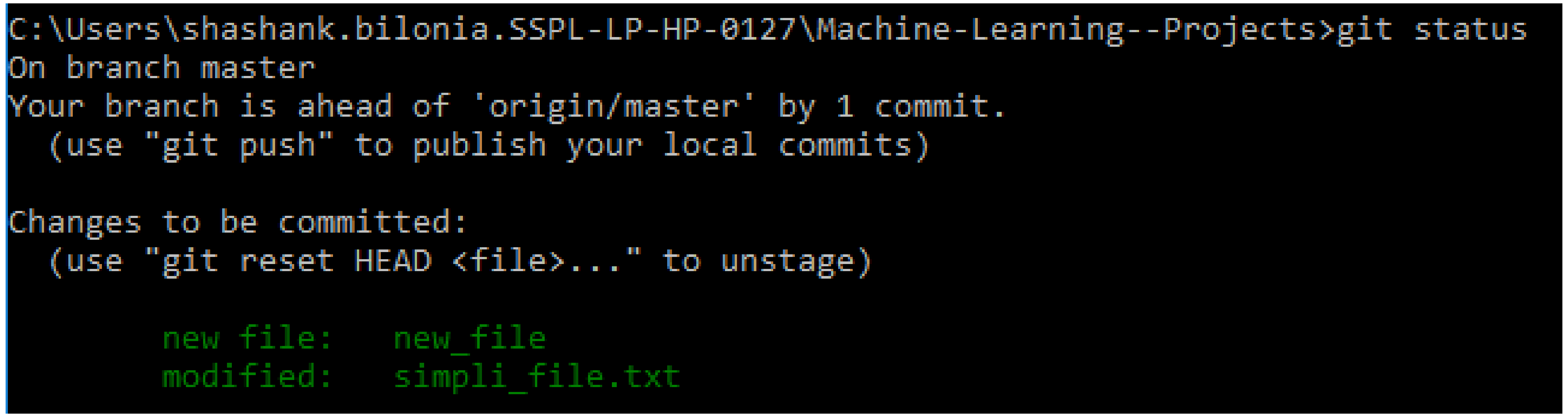
Committed a mistake in your message

Add the changes

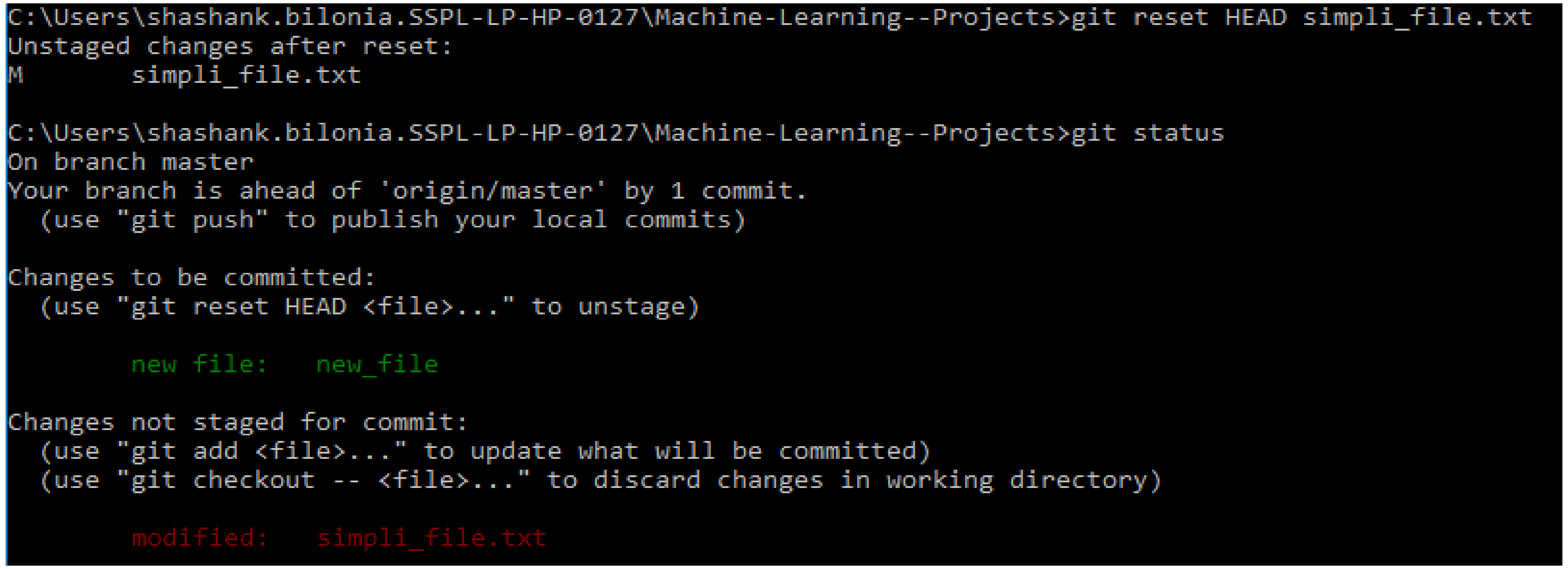
Commit using the **--amend** option with the correct message

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

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. Check the log of the recent commits
2. Open the file in the text editor to implement the necessary changes
3. Add the file to the staging area
4. Execute **git commit --amend** to modify the commit message
5. Execute **git log –oneline** to check the changes in the commit

ASSISTED PRACTICE

FULL STACK

Deleting Files in Git

Deleting Files from Staging Area and Working Directory

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

Deletes the file from staging area and working directory

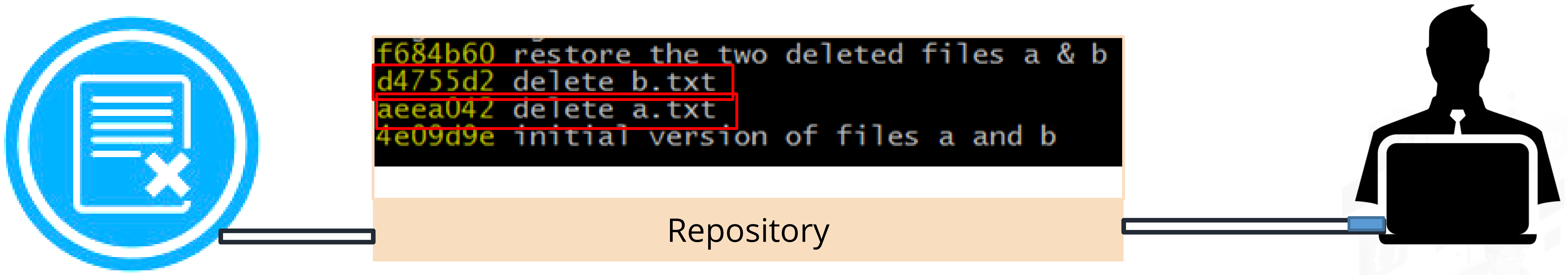
Delete a file only from the staging area: `$ git rm --cached <filename>`

```
$ 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
aeea042 delete a.txt
4e09d9e initial version of files a and b
```

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

Deleting Files in Git

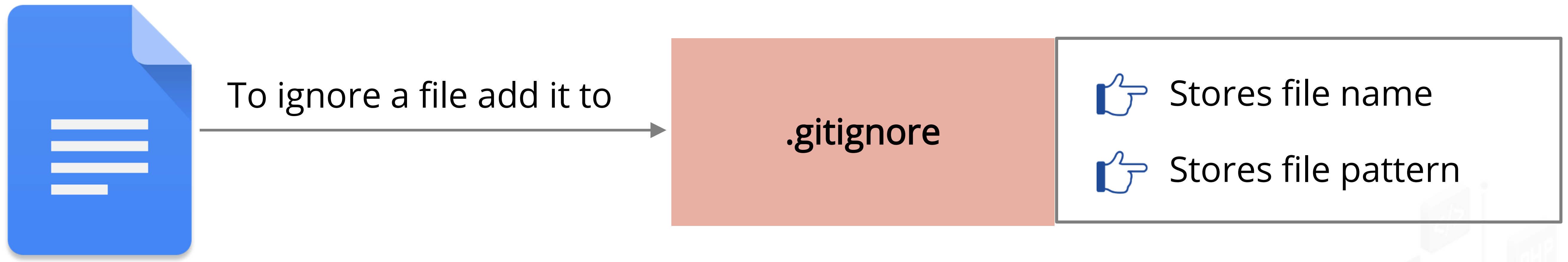
Problem Statement: You have a file in the working directory which is not required in your project anymore. Find a solution to remove the file from the staging index and the working directory.

Steps to Perform:

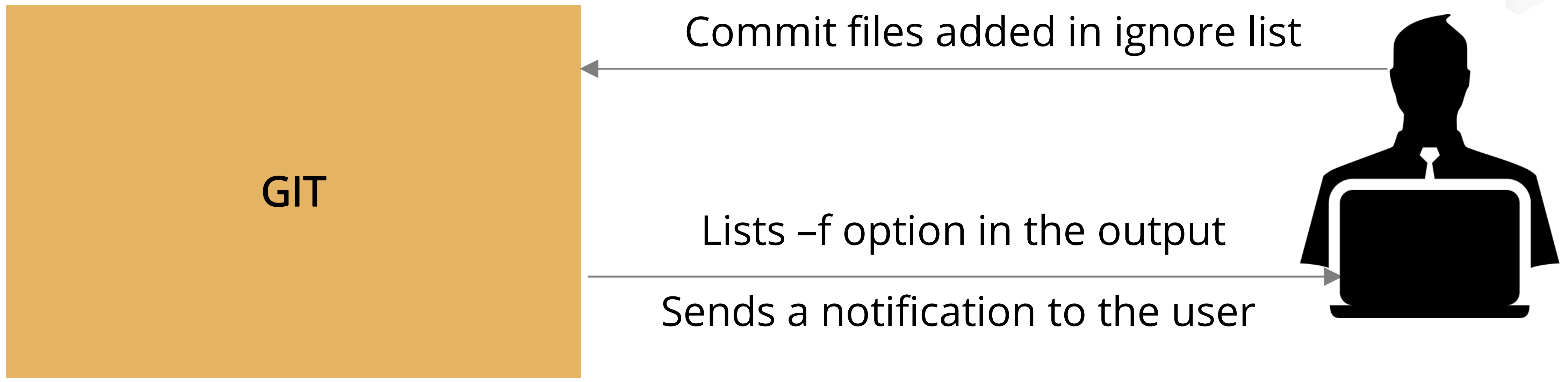
1. Check the list of files in the repository
2. Execute **git rm <file name>** to remove the file from your repository
3. Check the git status
4. Commit the changes

Ignoring Files in Git

How to Ignore Files in Git



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


GIT
Commit files added in ignore list
Lists –f option in the output
Sends a notification to the user

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. Create a file with **.gitignore** extension.
2. Edit this file as needed. Each new line should list an additional file or folder that you want Git to ignore.
3. Try to add those files or folders to the Git repository again, and it will be ignored.

ASSISTED PRACTICE

FULL STACK

Renaming Files in Git

Steps to Rename Files in Git

1

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

2

```
$ git add
```

```
$ git commit
```

3

```
$ git status
```

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

ASSISTED PRACTICE

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 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.

Knowledge Check

Knowledge Check

1

Which of the following would result in a file being “untracked” in Git?

a. A file being added into the staging area

b. A file being committed into the local repository

c. A file being modified since last commit

d. A file being added to the working directory

Knowledge Check

1

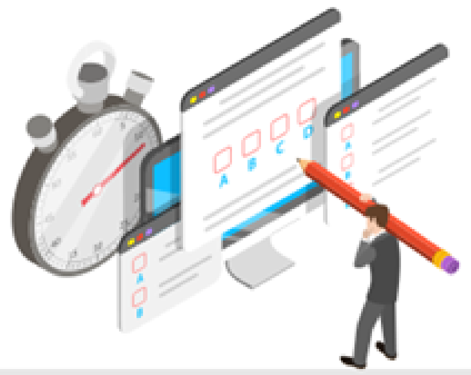
Which of the following would result in a file being “untracked” in Git?

a. A file being added into the staging area

b. A file being committed into the local repository

c. A file being modified since last commit

d. A file being added to the working directory



The correct answer is d

When a new file is created or copied in the working directory, it is “untracked.”

Knowledge Check

2

Which of the following is correct about the “staging area”?

a. A location that consists of all committed files

b. A location that collects all the files that would be included in the next commit

c. A location consisting of the metadata that Git uses to manage files

d. A location where untracked files reside

Knowledge Check

2

Which of the following is correct about the “staging area”?

a. A location that consists of all committed files

b. A location that collects all the files that would be included in the next commit

c. A location consisting of the metadata that Git uses to manage files

d. A location where untracked files reside

The correct answer is b

Staging area is a virtual place which consists of all files to be considered for the next commit. All files have to be staged before commit.

Knowledge Check

3

Which of the following statements is correct about the .git directory content?

a. It contains all the staged files

b. It contains all the staged and tracked files

c. It contains the metadata that Git needs to function

d. It contains the configurations specific to that repository

Knowledge Check

3

Which of the following statements is correct about the .git directory content?

a. It contains all the staged files

b. It contains all the staged and tracked files

c. It contains the metadata that Git needs to function

d. It contains the configurations specific to that repository

The correct answer is **c**

The hidden .git directory contains all the information that is required for Git to function.

Knowledge Check

4

Which of the following command will display the status of the repository in Git?

a. “git status”

b. “git log”

c. “git init”

d. “git branch”

Knowledge Check

4

Which of the following command will display the status of the repository in Git?

a. “git status”

b. “git log”

c. “git init”

d. “git branch”

The correct answer is a

The command “git status” displays the current status of the repository.

Knowledge Check

5

What does the Git command “git rm –cached” do?

a. It would delete the file from the working directory.

b. It would delete the file from the working directory and staged area.

c. It would only delete the file from the staged area, but not from the working directory. However, the file would be tracked by Git.

d. It would only delete the file from the staged area, but not from the working directory. However, the file would be marked untracked.

Knowledge Check

5

What does the Git command “git rm –cached” do?

a. It would delete the file from the working directory.

b. It would delete the file from the working directory and staged area.

c. It would only delete the file from the staged area, but not from the working directory. However, the file would be tracked by Git.

d. It would only delete the file from the staged area, but not from the working directory. However, the file would be marked untracked.

The correct answer is d

The file would be deleted from the staged area but retained in the working directory. Git would not track the file.

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

LESSON-END PROJECT