

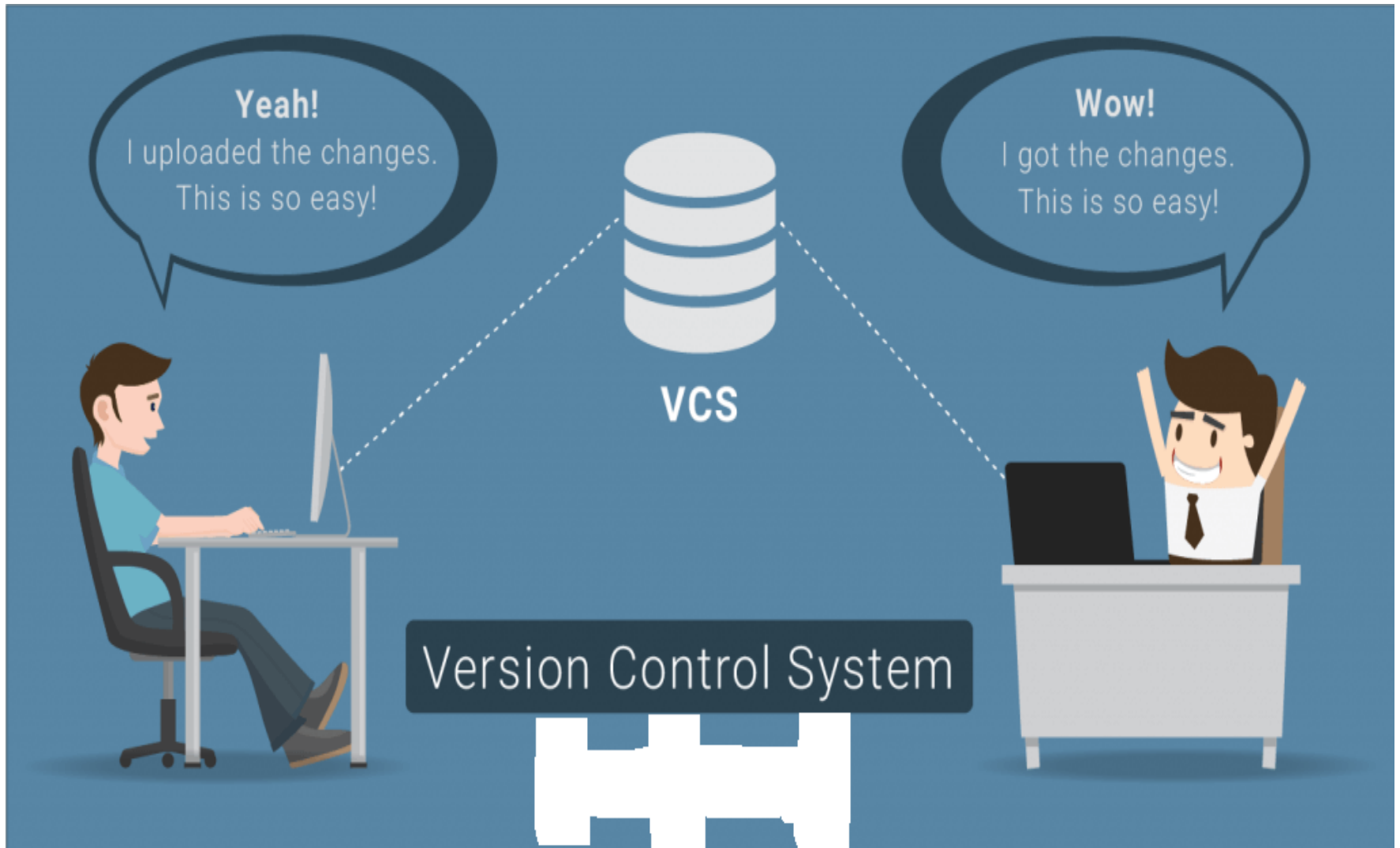


**git**



The diagram illustrates a source code management workflow using six stacks of code blocks. The blocks are connected by colored lines: a teal line connects the top-left and middle-left stacks; an orange line connects the top-left, top-middle, and bottom-right stacks; a yellow line connects the middle-left, bottom-middle, and bottom-right stacks; and a teal line connects the middle-left and middle-right stacks. The text 'Source Code Management' is overlaid in a large, blue, italicized serif font.

# *Source Code Management*



# What is a version control system?

A version control system (VCS) allows you to track the history of a collection of files. Each version captures a snapshot of the files at a certain point in time and the VCS allows you to switch between these versions. These versions are stored in a specific place, typically called a repository.

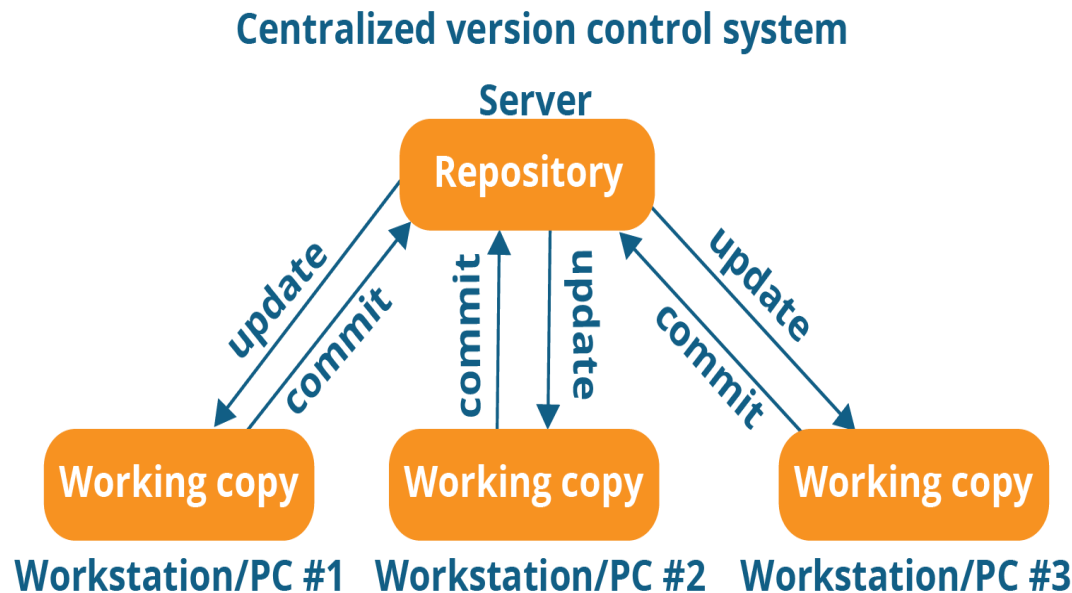
## VCS Types

- 1) Localized and centralized version control systems
- 2) Distributed version control systems

# Localized and centralized version control systems

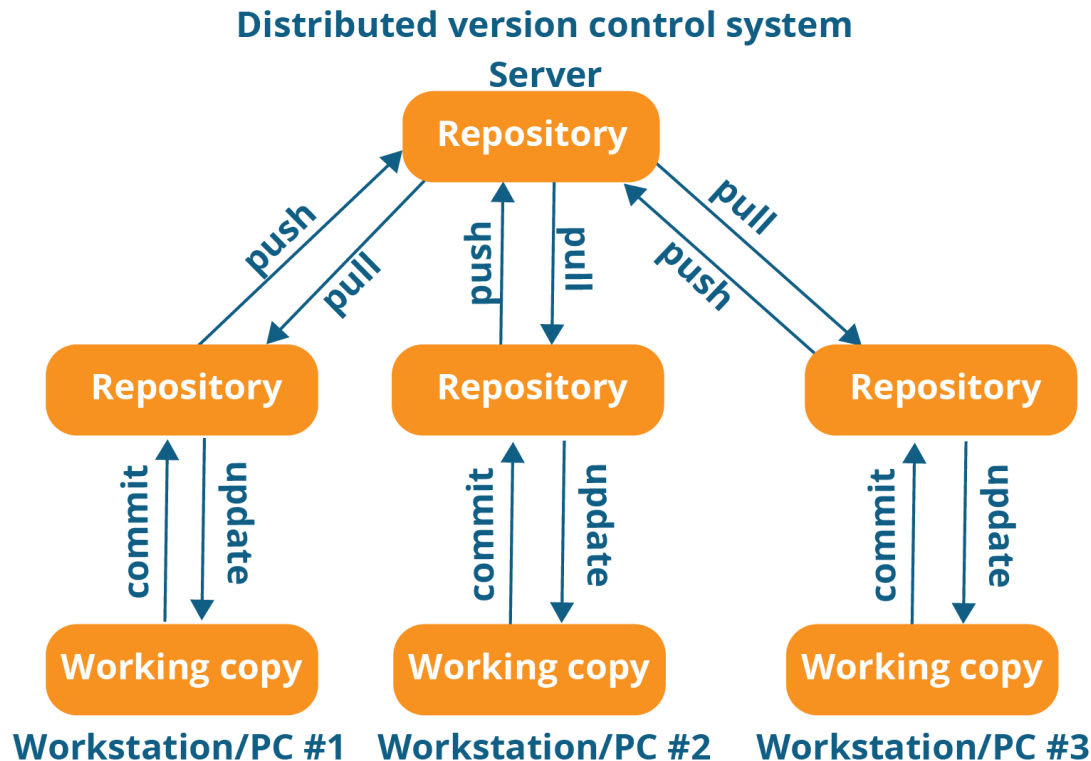
A centralized version control system provides a server software component which stores and manages the different versions of the files.

In a localized version control systems it is the individual computer and in a centralized version control systems it is the server machine. Both system makes it also harder to work in parallel on different features



# Distributed version control systems

In a distributed version control system each user has a complete local copy of a repository on his individual computer. A global repository 'github.com' used to keep data as centralised server.



# Centralized vs. Distributed Version Control Systems

Centralized Version Control Systems	Distributed Version Control Systems
Require Central Server to store the code master copy.	Central server is not required to store project code.
Each developer keeps a local copy. Developers use separate branches.	Each developer has a clone for the repository. Developers use either distributed repositories or branches.
Project history is not maintained locally, its only maintained on the server.	Each developer has a full project history.
Commits happen direct to the server	Commits are made locally.
Requires connectivity as commits happen directly to the server.	Connectivity is required only for push/pull operation.
Examples: SVN, CVS, Perforce	Mercurial, Git



# What is Git?

- *Git* is currently the most popular implementation of a distributed version control system.
- Git allows the user to synchronize the local repository with other (remote) repositories.

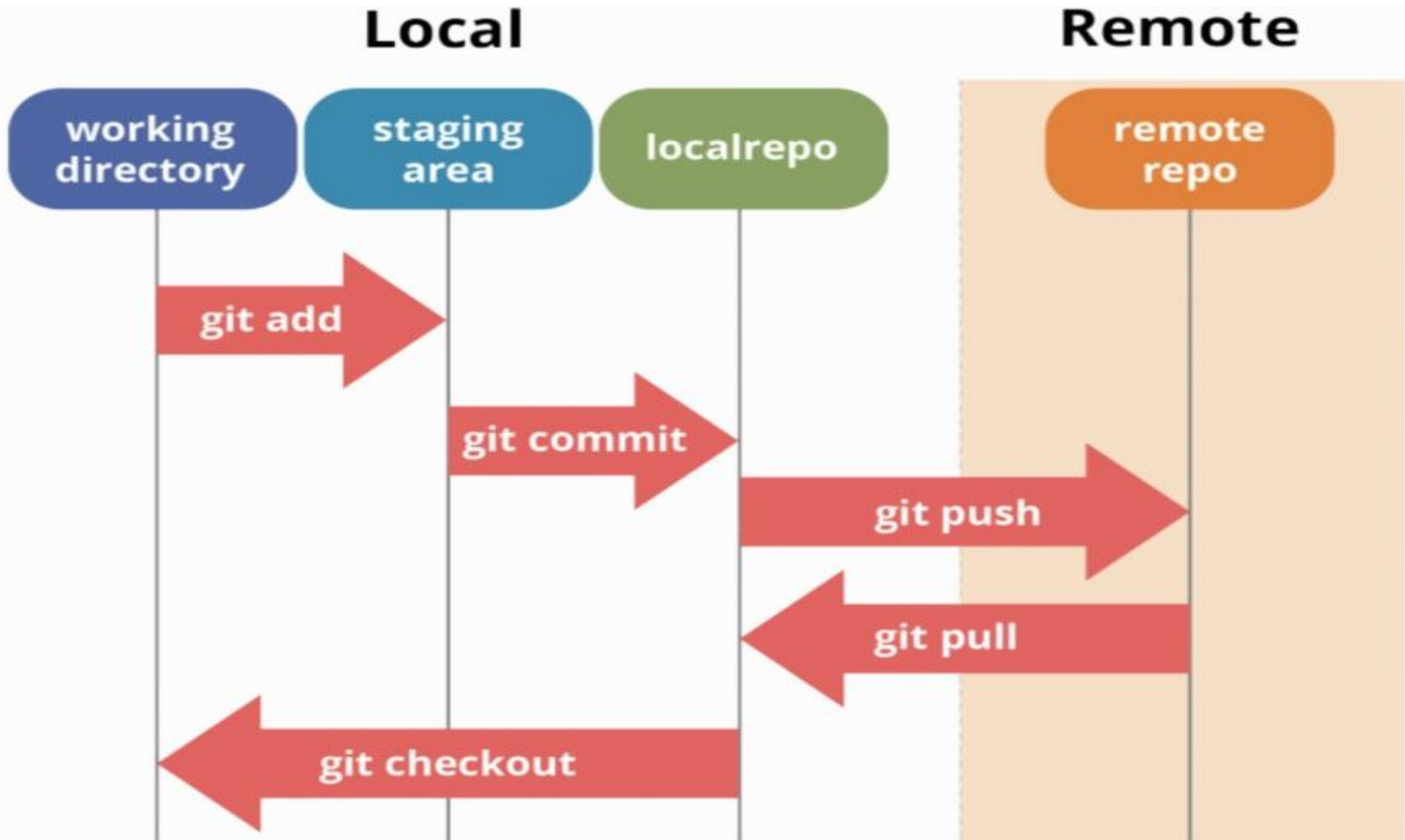




# Advantages of Git

- ✓ Free and open source
- ✓ Fast and small
- ✓ Implicit backup
- ✓ Security
- ✓ No need of powerful hardware
- ✓ Easier branching

# Git workflow



## Points to be covered in Git

- 1) Git Installation
- 2) Initialization, Add and commit
- 3) Creating account in Github
- 4) Linking Github with local server
- 5) Git push
- 6) Git clone
- 7) Git pull
- 8) Branching
- 9) Git log, Git diff
- 10) Versioning
- 11) Deleting content in local and global repository

# Git Installation

## In RHEL

- `#dnf install git-all` or `#yum install git-all`

## In Ubuntu

- `#apt-get update`
- `# apt install git-all`
- `# git --version`

## In Windows

- Download and Install gitbash

Additionally Open [github.com](https://github.com) and create your account then create a repository there

# Git operation

```
# mkdir mygitproject
# cd mygitproject
# git init      ( to initialising)
# cat > test1.txt
# cat > test2.txt
# git status
# git add test1.txt test2.txt      ( add these two files)
or
# git add .      ( add current directory)
# git status
# git commit -m " first commit adding test1.txt and test2.txt)
# git status
```

# Git operation

```
# git remote add origin
```

```
"https://github.com/depakkumarrrts/demo.git"
```

```
# git config --global user.name "deepak-kumar-rrts"
```

```
# git config --global user.email "deepak.amie.it@gmail.com"
```

```
# git config --list
```

```
# git push origin master
```

Now open [github.com](https://github.com) and check the files

# Git operation

```
# cat >test3.txt
# cat >test4.txt
# git status
test3.txt .....untracked file
test4.txt .....untracked file
# vi test2.txt
.....do some modification
# git status
modified test2.txt
# git add test3.txt
# git status
# git commit -m " my 2nd commit"
# git log --online
( this changes done in local repo only)
check in github.com - nothing changes happened in test2.txt
#git push origin master
```



# Cloning

1) Create a new Instance –Install git –create a directory –go there -----or  
In the same system switch to other directory

2) # git init

```
#git clone "https://github.com/depakkumarrrts/demo.git"
```

3) # ls

demo

we can see github account repository copied here

```
# cd demo
```

```
# ls
```

```
# nano test4.txt
```

```
# git add .
```

```
# git commit -m "adding clone file"
```

```
#git push origin master
```

Now open github.com and check the file test4.txt

# Pulling

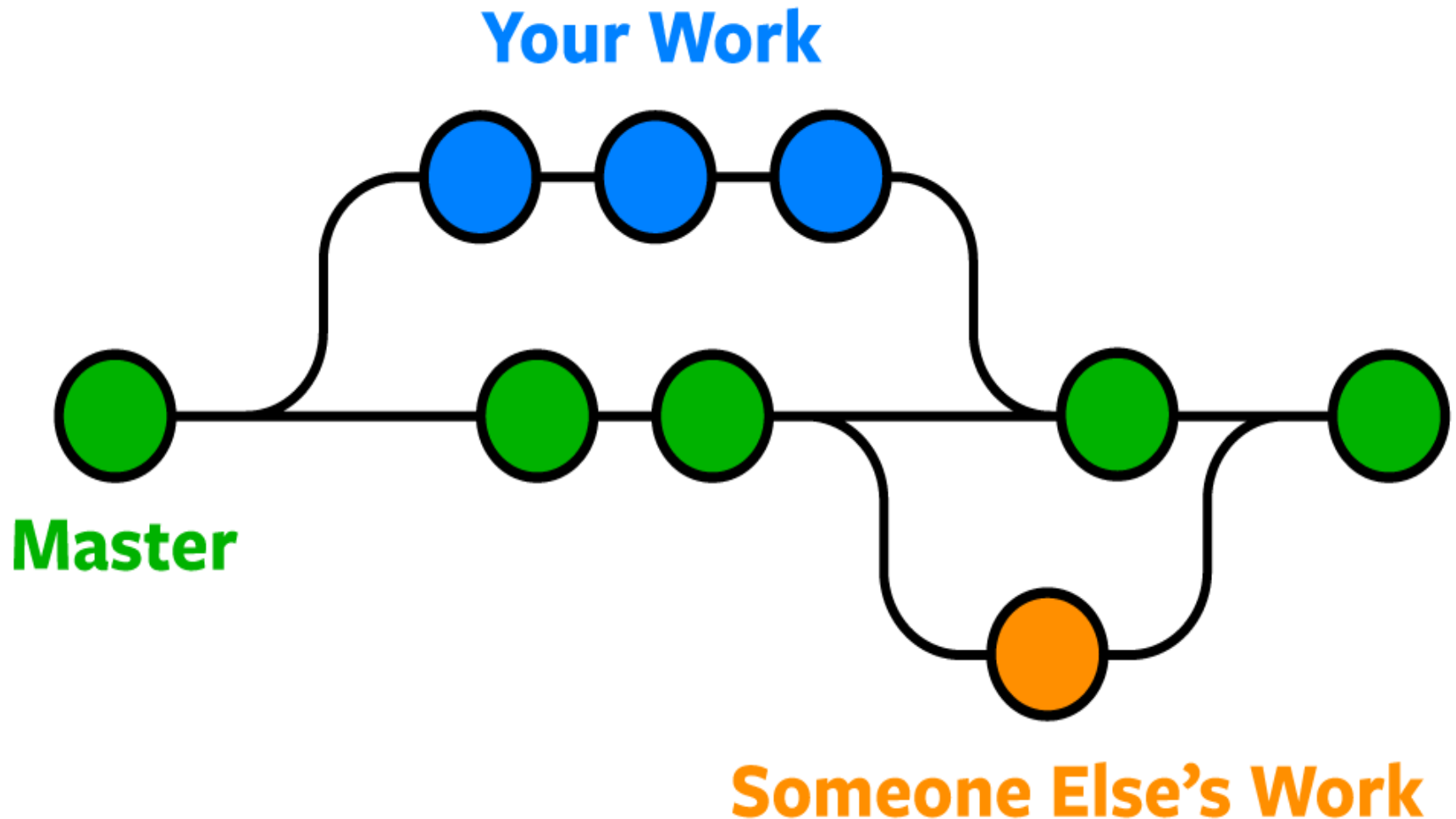
- 1) In developer instance or in the same instance –switch to developer directory
- 2) `#git pull origin master`

Now check test4.txt comes here

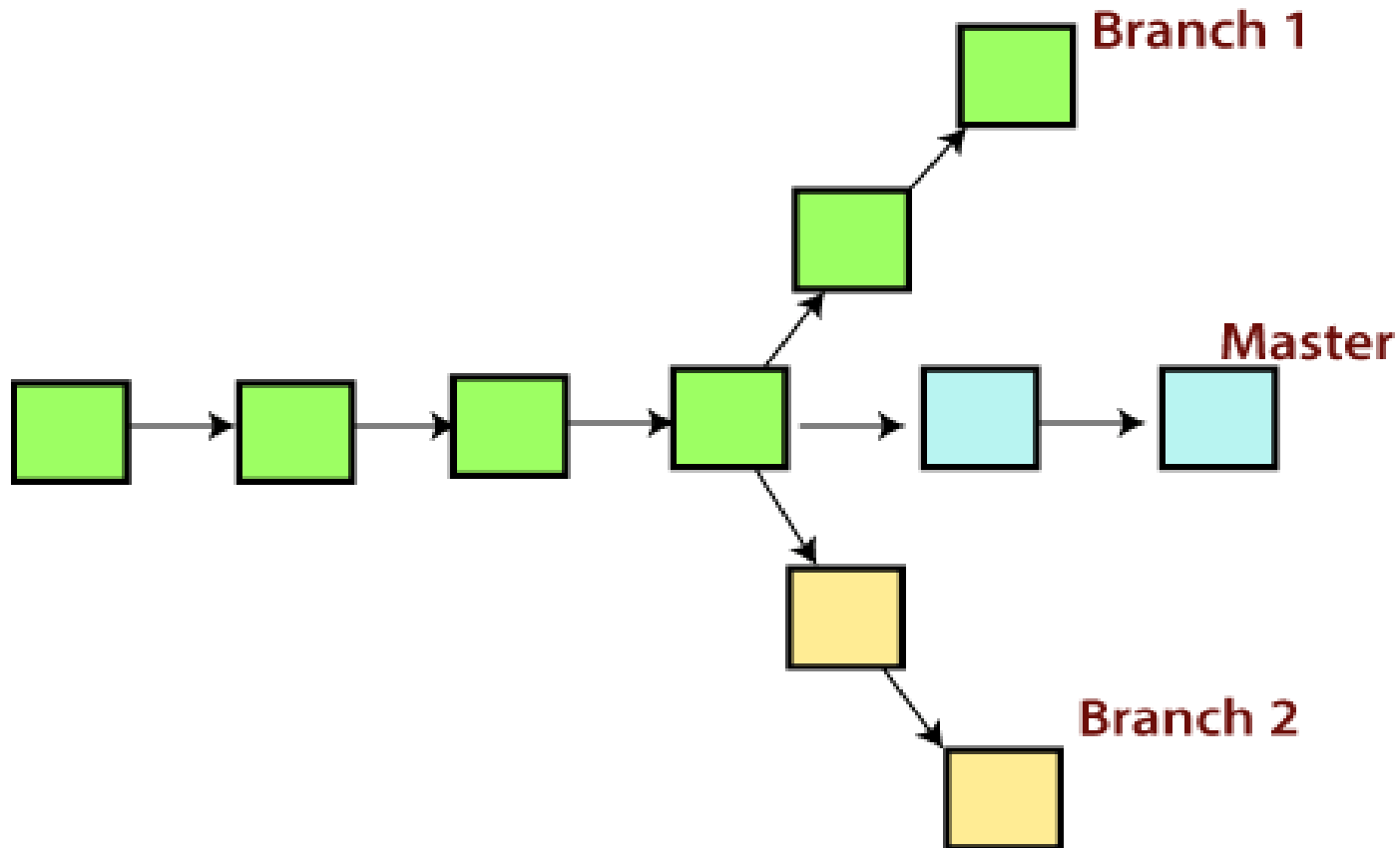
## Parallel Development(Branching)

- Our development model uses a variety of supporting branches to aid parallel development between team members, ease tracking of features, prepare for production releases and to assist in quickly fixing live production problems.
- Simply: Multiple developers working on the same project or repository. To handle the workspace of multiple developers, we use branches.

Branching: Last commit of master become the first commit of feature branch



Branching: Last commit of master become the first commit of feature branch



## Git operation in branching

```
#mkdir myproject
```

```
# cd myproject
```

```
# git init
```

```
# touch test11.txt test22.txt
```

```
# git branch
```

```
*master
```

```
# git branch feature1
```

```
# git branch
```

```
*master
```

```
feature1
```

```
#git checkout feature1
```

```
#ls
```

```
test11.txt test22.txt    ( last commit of master become the first  
commit of feature branch)
```

## Git operation in branching

```
# nano test33.txt
# git add .
# git commit -m "added to feature branch"
# git branch
*feature1
master
# git checkout master
# ls
test11.txt test22.txt
```

This feature is used for parallel development . No branch is interfering each other. When all codes are ok then only developer merge it with master branch.

```
# git status
```



## Git operation in branching

#git checkout feature1

on branch feature1

nothing to commit

# git push origin feature1

username:

password:

Now open github.com ---> Branch -master --feature1

check file in both branch

# Git tag

Tagging in git or any other VCS refers to create specific point in history for your repository/data.

This is usually done to mark release points ( version 1, version 2 ....)

## **Why should create tag:**

- > To mark release point for your code/data.
- > To create historic restore point

## **When to create tag:**

- > When you want to create a release point for a stable version of your code
- > create a historic point to restore the data

## How to create tag

checkout the branch where you want to create tag

```
# git checkout master
```

```
# git tag <tagname>    eg: :--> #git tag version1  #git tag  
version1.2  #git tag version2
```

```
# git tag : to show all tag
```

```
# git show version1 : to see only specific tag details
```

## How to push into github

logon into github.com

```
# git push origin version1
```

```
# git push --tags ( to push all tags)
```

Now go to github.com and check in release

# Delete Operation

## How to delete tag

--> `git tag -d version1`

--> `git tag` : to see deleted record

## To delete tag in remote github

--> `git push origin -d version1` ( delete tag in github)

## How to checkout tag

--> we cannot checkout tag in git

--> we can create a branch from tag and checkout the branch

# `git checkout -b branchname tagname`

## Delete Operation

Q: How to remove files from staged or workspace or remote repository

# git rm filename (removes the file from the repo but also deletes it from the local file system)

# git rm \*.txt

# git rm \*

# git rm --cached filename (remove the file from the repo and **not** delete it from the local file system )

# git rm -r foldername

# git commit -m " removing filename"

# git push origin master

Now check in [github.com](https://github.com)

## Edit Operation

Q: How edit github.com repository link from system

```
#git config --edit
```

Q: How to rename branch name

```
# git branch -m oldname newname
```

Q: How to remove github.com repository link from system

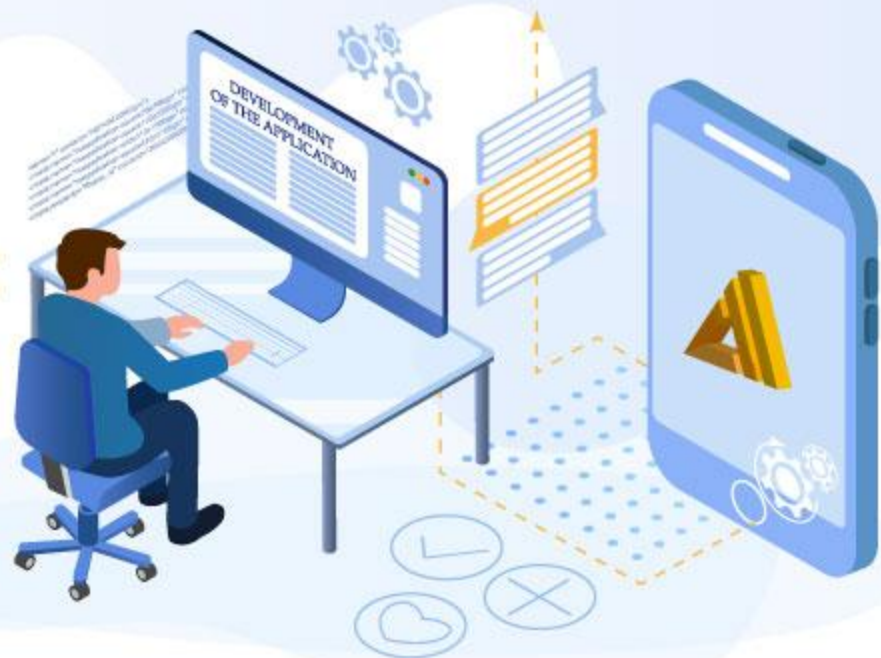
```
# git config --list
```

```
# git remote remove origin
```

```
# git config --list
```



For Next-Gen  
App Development





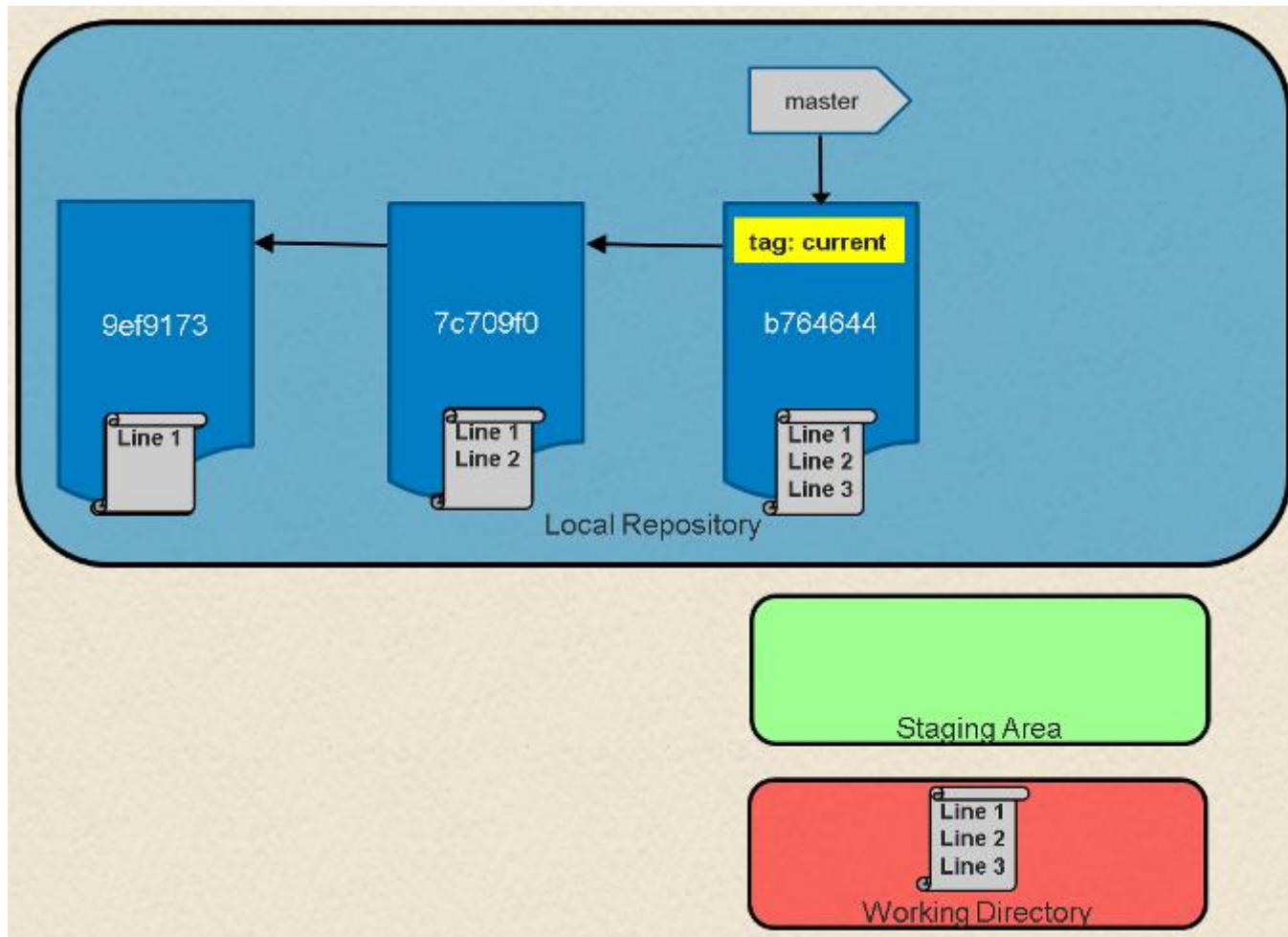
# AWS Amplify

AWS Amplify is a set of tools and services that enables mobile and front-end web developers to build secure, scalable full stack applications, powered by AWS.

With Amplify, it's easy to create custom onboarding flows, develop voice-enabled experiences, build AI-powered real-time feeds, launch targeted campaigns, and more.

## AWS Amplify Steps

- 1) Open aws console ---services —aws amplify-----verify github account—add master branch -----ok
- 2) Create some html file in system —push to github
- 3) Open aws amplify ---app setting general – reconnect repository – confirm
- 4) Rewrites and redirects —click here
- 5) Go to general —copy domain name and paste in new tab



# How to reset, revert, and return to previous states in Git

## Reset

```
#Git log --online
```

```
b764644 File with three lines
```

```
7c709f0 File with two lines
```

```
9ef9173 File with one line
```

we want to reset master to point to the commit two back from the current commit,  
we could use either of the following methods:

```
$ git reset 9ef9173
```

or

```
$ git reset current~2
```

```
$ git log --online
```

```
9ef9173 File with one line
```

# How to reset, revert, and return to previous states in Git

## Revert

Revert adds a new commit, Git will prompt for the commit message:

```
$ git revert HEAD
```

If we do a git log now, we'll see a new commit that reflects the contents before the previous commit.

```
$ git log --oneline
```

```
11b7712 Revert "File with three lines"
```

```
b764644 File with three lines
```

```
7c709f0 File with two lines
```

```
9ef9173 File with one line
```

```
$ git log --oneline master
```

```
6a92e7a C4
```

```
259bf36 C2
```

```
f33ae68 C1
```

```
5043e79 C0
```

```
$ git log --oneline feature
```

```
79768b8 C5
```

```
000f9ae C3
```

```
259bf36 C2
```

```
f33ae68 C1
```

```
5043e79 C0
```