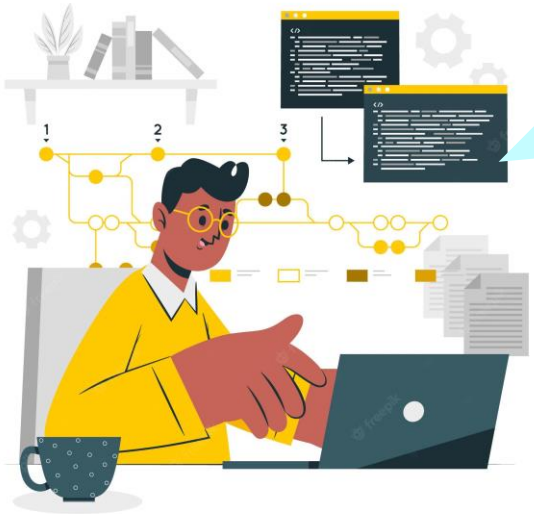
Git Training

Part - 1

Index

Basic Concept about Git

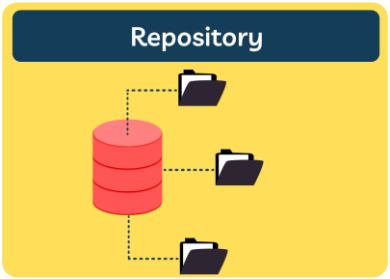


Git is a popular **version control system** designed to make it easier to have **multiple versions** of a code base, sometimes **across multiple developers** or teams.

Basic Concept about Git

Git is **installed** and maintained on your **local system** rather than in the cloud.

Git Repository is a place where Git can store versions of our files.

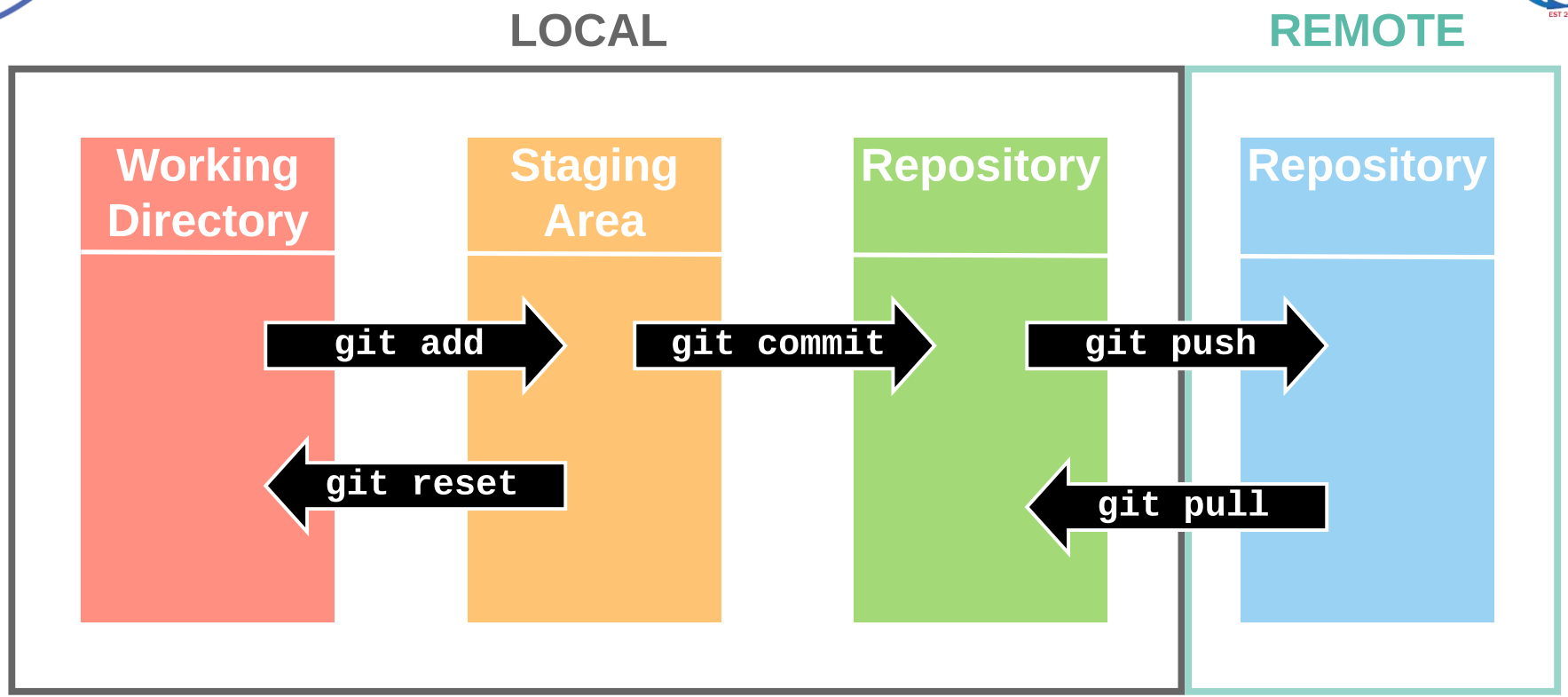


Basic Concept about Git

It is used for:
Tracking code changes
Tracking who made changes
Coding collaboration with other developers

Why Git?
Over 70% of developers use Git!
Developers can work together from anywhere in the world.
Developers can see the full history of the project.
Developers can revert to earlier versions of a project.

Git Diagram



Download and Install Git

In order to **use Git**, we have to **install** it on **our computer**. We can **download** the **latest version** of Git for free from the **following** official **website**.

You can download for your operating system from the options given.

https://www.git-scm.com/downloads

Download and Install Git

For **Windows**, you can use **Git bash**, which comes included in Git for Windows.

For **Mac** and **Linux**, you can use the **built-in terminal**.

[Ubuntu] Install at your workstation, follow the default options.

```
$ sudo apt -get update &
$ sudo apt -get install git-core
```

Download and Install Git

To **verify** the **Git installation**, we can **run** this **command**. This will show you the current version installed on your PC.

```
$ git --version
```

Configure Git Initially

Now let Git know who you are.

You need to register your username and email address in your Git repository. Git will use this information to identify who made specific changes to files.

```
$ git config --global user.name "Nahid Hasan"
$ git config --global user.email
"nahid.hasan@bjitgroup.com"
```

// To verify these are set

```
$ git config –list
```

Create and Initialize a Project in Git

It is now time to **create** our **project**. This will **tell Git** to get ready **to start watching our files** for every change that occurs.

We can navigate to our new project's location and create new project folder:

```
$ mkdir test-project
```

// Navigate to project working directory.

```
$ pwd
$ cd test-project
```

Create and Initialize a Project in Git

Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

```
$ git init
```

This will create a **.git** directory in your current directory. Then you can commit files in that directory into the repository.

// Check our current working directory

```
$ ls -a
```

Add new Files, Update Files and Remove Current Changes

Let's **add/create** some files in the current working directory. Then we can check the Git status and see if it is a part of our repository:

```
$ touch test.txt
$ git status
```

Now Git is **aware** of the **file** but has not added it to our repository!

Add new Files, Update Files and Remove Current Changes

Files in your Git repository folder can be in one of 2 states:

- **Tracked -** files that Git knows about and are added to the repository
- **Untracked -** files that are in your working directory, but not added to the repository.

When you first **add files** to an **empty repository**, later you will **change/add files**, they are **all untracked**.

To get Git to **track them**, you need to **stage them**, or add them to the staging environment.

Add new Files, Update Files and Remove Current Changes

Every time **before** we commit **changes**, we must **add** it to the **staging area**. Only staged files will be committed. Now, we can add files to the Staging Environment.

// Add a single file to the Staging Environment

```
$ git add test.txt
```

// Add multiple files to the Staging Environment

```
$ git add index.html bluestyle.css
```

// Add all files (new, modified, and deleted) in the current directory to the Staging Environment

```
$ git add --all
```

Add new Files, Update Files and Remove Current Changes

// Add all untracked files. Please note that this will add your untracked or new files.

```
$ git add --a
```

// Add all tracked and untracked files

```
$ git add -A
```

// To check the file status

```
$ git status
```

Add new Files, Update Files and Remove Current Changes

```
// To see what is modified but unstaged
$ git diff
```

```
// To see a list of staged changes
$ git diff --cached
```

```
// To remove the staged file, but keep the file in working directory
$ git rm --cached <filename>
```

Git Commit

When all of our changes are added in the staging area, we can commit using the following command.

```
$ git commit -m "Your commit message here. Refs #22333"
```

The first part of the command git commit tells Git that all the files staged are ready to be committed.

The second part **-m** "first commit" is the commit message.

Git Commit

//To add all of your tracked files into stage and commit in a single command

```
$ git commit -a -m "Your commit message here. Refs #22333"
```

// To see a log of all changes in your local repository

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

Git Commit

// To show only the 5 most recent updates

```
$ git log -5
```

// To update your last commit without making a new one, use the following command

```
$ git commit --amend -m "Your amend message here"
```

Local Branch - Create, Checkout, Delete

Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project.

// To create a new local branch

```
$ git branch <branch_name>
```

// To check the available branches. Asterisk (*) beside master branch specifies that we are currently on that branch.

```
$ git branch      // List all the local branches in the project.
$ git branch –r   // List all the remote branches.
$ git branch –a   // List all the local and remote branches.
```

Local Branch - Create, Checkout, Delete

// To switch to a branch

```
$ git checkout <branchname>
```

// To create and immediately switch

```
$ git checkout -b <branchname>
```

// To delete the created local branch, as it is no longer needed. You cannot delete the branch that you are currently on.

```
$ git branch –d <branchname>
```

// It may not be possible to delete a branch if there are any uncommitted changes. To delete that branch, run the following command.

```
$ git branch -D <branchname>
```

Merge Local Branch and Resolve Conflict

// To merge the updated branch with your working branch. Git Merge is used to combine two branches.

```
$ git merge <branchname>
```

How Conflicts

When Git encounters a conflict during a merge, It will edit the content of the affected files with visual indicators that mark both sides of the conflicted content. These visual markers are: <<<<<<<, =======, and >>>>>>>. Generally the content before the ======= marker is the receiving branch and the part after is the merging branch.

Merge Local Branch and Resolve Conflict

Conflict Resolve and Commit

Once you've identified conflicting sections, you can check and fix the merge conflict. When you're ready to finish the merge, all you have to do is run git add on the conflicted file(s) to tell Git they're resolved. Then, you run a normal git commit to generate the merge commit.

```
$ git commit -m "Your commit message here. Refs #22333"
```

Git Ignore

Specifies files that you don't want Git to track under version control. Commonly used for compiled files, binaries, large asset files (e.g. images).

You can use wildcards (e.g. *.dat, *.png, images/*, temp/*, etc.)

Create .gitignore file and specify files and folders

```
$ touch .gitignore
```

Git Help

If you are having trouble remembering commands or options for commands, you can use Git help.

// To see all the available options for the specific command

```
$ git command -help
```

// To see all possible commands

```
$ git help --all
```

Thank You