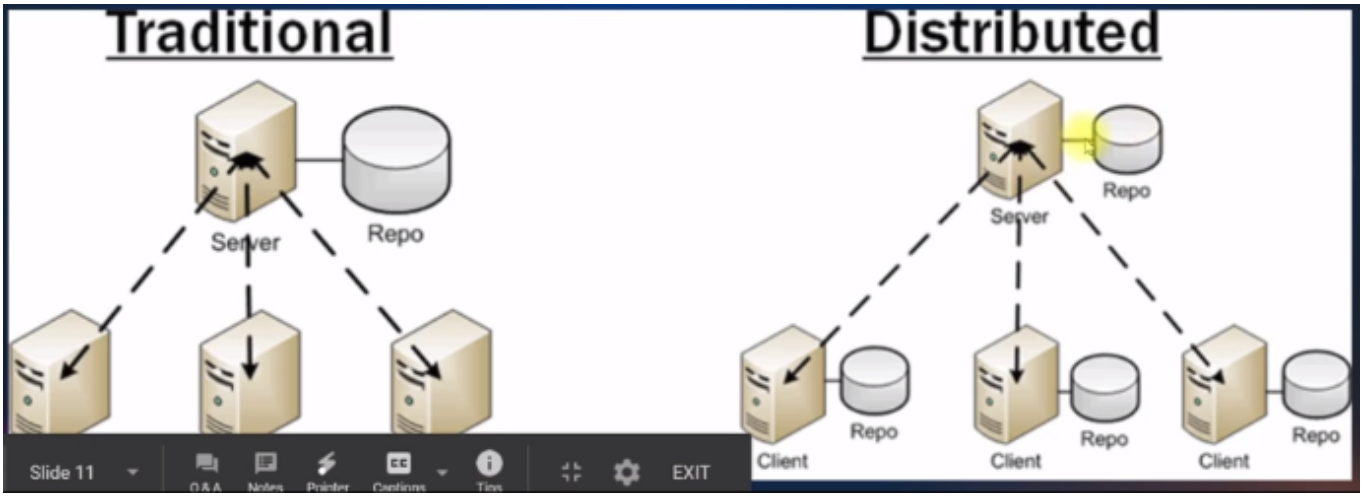
GIT for Devops

Localized and Centralized VCS

-A localized version control system keeps local copies of the files.
-In centralized source control, there is a server and a client. The server is the master repository which contains all of the versions of the code.

Distributed version control systems

• In a distributed version control system each user has a complete local copy of a repository on his individual computer.



`git init` -> initialize the empty git repository in the current directory
.git -> this folder maintains all the versions and all the changes
`git status` -> this will tell you the files that are untracked that should be included in the repo
`git add .` -> this will start tracking the files on the local changes
`git commit -m "First commit"` -> this command will commit the changed code into the local changes with some message (-m)

To Add the User to the GIT

`git config --global user.email "add ur mail"` -> This will add the user of the mail to change and commit the code in this file

`git config --global user.name "Your Name"` -> This will add the Name of the user to change things on the file code

--global -> this is used to set the user name for all the repositories

Local Repository Setup

1. **Set the name and email for Git to use when you commit:**
   - git config --global user.name "Imran Teli"
   - git config --global user.email imran@visualpath.com
2. **Create a directory**
3. **Initialize dir with**
   - git init
4. **Create Readme.md file**
   - git add (Staging)
   - git commit (Local commit)

Remote Repository

Remote Repository

- Create Remote repository on
  - Github, bitbucket, codecommit etc
- Clone Repo to local
  - git clone URL
- Local to Remote integration
  - cd to local repo
  - git remote add origin ssh://git@github.com/[username]/[repository-name].git
  - git push
  - git pull (to fetch latest changes)

Two Things to notice after the remote git repo is created

1) create a new repository on command line

To create new repository
command

Example

```
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/imranvisualpath/titanwork.git
git push -u origin main
```

2) Push an existing repo from the command line

url => https://github.com/SuganthAmar/reponame.git
username reponame with .git extension

In the `git push -u origin main` command, the `-u` option stands for `--set-upstream`.

When you push a local branch to a remote repository for the first time, you typically use the `-u` or `--set-upstream` option followed by the remote repository name (`origin` in this case) and the branch name (`main` in this case). This option tells Git to set up a tracking relationship between your local branch and the remote branch.

`cat .git/config` -> this will show the new entries

`git branch -M main` -> this will change the branch name from **master** to **main**

The `git branch -M main` command is used to rename the current branch to `main`. Here's what each part of the command does:

- `git branch`: This is the Git command used for working with branches.
- `-M`: This is a shortcut for the `--move` option. It tells Git to rename the branch.
- `main`: This is the new name you're assigning to the current branch.

This command is commonly used when you're working with Git repositories where the default branch name is `master`, but you want to rename it to `main`, which is a practice becoming increasingly common to remove potentially problematic terminology.

`git log` -> Shows all the pervious commits to the current branch
`git diff` -> Will show the difference between the file that you have changed i.e what exactly is changed
`git diff --cached` -> Will show the difference between the file that you have changed i.e what exactly is changed **after the file that is been staged**
`git log --oneline` -> Will give the commit id in smaller in character
`git show "commit_id"` -> Will show what things are newly added in different colour

`git pull` -> this command will pull the code that have been changed by other code to remote repo to local repo

Branches in GIT

`git branch -c testing`: //testing is the new branch name

- This command is used to create a new branch named `testing` based on the current branch.
- The `-c` option is short for `--copy`, which means it creates a new branch copying the current branch's content.

`git branch -a`:

- `git branch`: This is the Git command used for working with branches.
- `-a`: This option stands for "all" and tells Git to list both local and remote branches.

Switching Branches:

- To switch to an existing branch, you specify its name as an argument. For example:

  `git checkout main`

  we can also use `git switch main` -> to switch the branches

Creating and Switching to a New Branch:

- You can create a new branch and switch to it simultaneously by providing the `-b` option followed by the branch name. For example:

```
git checkout -b new-feature
```

Remove:

The `git rm` command is used to remove files from the working directory and the index (staging area) so that they are no longer tracked by Git. Here's how it works:

1. **Remove a File from Git Tracking**:
   - If you have a file in your working directory that you want to remove from version control, you can use `git rm` followed by the filename. For example: `git rm file.txt`
2. **Remove Files Matching a Pattern**:
   - You can also remove multiple files or files matching a pattern using wildcards. For instance: `git rm '*.txt'`
3. **Remove Files and Keep Them in the Working Directory**:
   - By default, `git rm` will remove the files from both the working directory and the index. If you want to keep the files in the working directory but remove them from the index, you can use the `--cached` option. For example: bashCopy code `git rm --cached file.txt`
4. **Force Removal of Files**:

   - If you want to remove files forcefully without checking their status, you can use the `-f` or `--force` option. Be cautious when using this option, as it can lead to permanent data loss. For example:

     ```
     `git rm -f file.txt`
     ```

     `

`git merge testing` -> this command will merge the testing branch to current branch

GitIgnore File

The `.gitignore` file is a special file in a Git repository that specifies intentionally untracked files to ignore. These files are typically ones that you don't want to include in your version control system for various reasons, such as build artifacts, temporary files, or sensitive information like API keys.

`git push --all origin` -> this will push all the code to that specific branch all at once

Clone

`git clone "git repo url"` -> this will clone the remote repo to local machine

Rollback on GIT

From Staging Area

`git restore --staged filename` ->

The `git restore --staged` command is used to unstage changes that have been added to the staging area (index) but haven't been committed yet. Here's what it does:

- `git restore`: This command is used to restore files in various ways, such as restoring changes in the working directory or restoring files to a specific commit.
- `--staged` (or `--cached`): This option specifies that the operation should be applied to the staged changes in the index.

`git diff previous_commit_id..current_commit_id` -> This will show the diff between the two commit difference in the remote repo

Revert

The `git revert` command is used to create a new commit that undoes the changes made by a previous commit. Unlike `git reset`, which rewrites history by removing commits, `git revert` creates new commits to reverse the effects of past commits while preserving the commit history.

Here's what `git revert` can do:

1. **Undo Specific Commits**:
    - You can use `git revert` to undo the changes introduced by one or more specific commits. For example: `git revert <commit-hash>`
2. **Undo Multiple Commits**:
    - You can revert multiple commits in a single operation by specifying a range of commits. For example: `git revert <start-commit>..<end-commit>`
3. **Undo Merge Commits**:

    - `git revert` can also be used to revert merge commits, which can be particularly useful for resolving conflicts or undoing changes introduced by a merge. For example: `git revert -m 1 <merge-commit-hash>`

HEAD -> will always point to the current latest commit

GIT SSH

To create a public and private key

`ssh-keygen.exe`

copy the content of the **public key** and paste it in the
github.com -> settings -> ssh and gpg keys -> give title and paste the key over there.

GIT CHEAT SHEET

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline.`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author= "<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.