DevOps

Week 02

Murtaza Munawar Fazal

Centralized Source Control

- Centralized source control systems are based on the idea that there's a single "central" copy of your project somewhere. Programmers will check in (or commit) their changes to this central copy.
- "Committing" a change means to record the difference in the central system. Other programmers can then see this change.
- Also, it's possible to pull down the change. The version control tool will automatically update the contents of any files that were changed.

Centralized Source Control

- Most modern version control systems deal with "changesets," which are a group of changes (possibly too many files) that should be treated as a cohesive whole.

- Programmers no longer must keep many copies of files on their hard drives manually. The version control tool can talk to the central copy and retrieve any version they need on the fly.

- Some of the most common-centralized version control systems you may have heard of or used are Team Foundation Version Control (TFVC), CVS, Subversion (or SVN), and Perforce.

Typical Workflow for Centralized Source Control

- Get the latest changes other people have made from the central server.
- Make your changes, and make sure they work correctly.
- Check in your changes to the main server so that other programmers can see them.

Distributed Source Control

- These systems don't necessarily rely on a central server to store all the versions of a project's files. Instead, every developer "clones" a repository copy and has the project's complete history on their local storage. This copy (or "clone") has all the original metadata.
- The disk space is so cheap that storing many copies of a file doesn't create a noticeable dent in a local storage free space. Modern systems also compress the files to use even less space; for example, objects (and deltas) are stored compressed, and text files used in programming compress well (around 60% of original size, or 40% reduction in size from compression).

Distributed Source Control

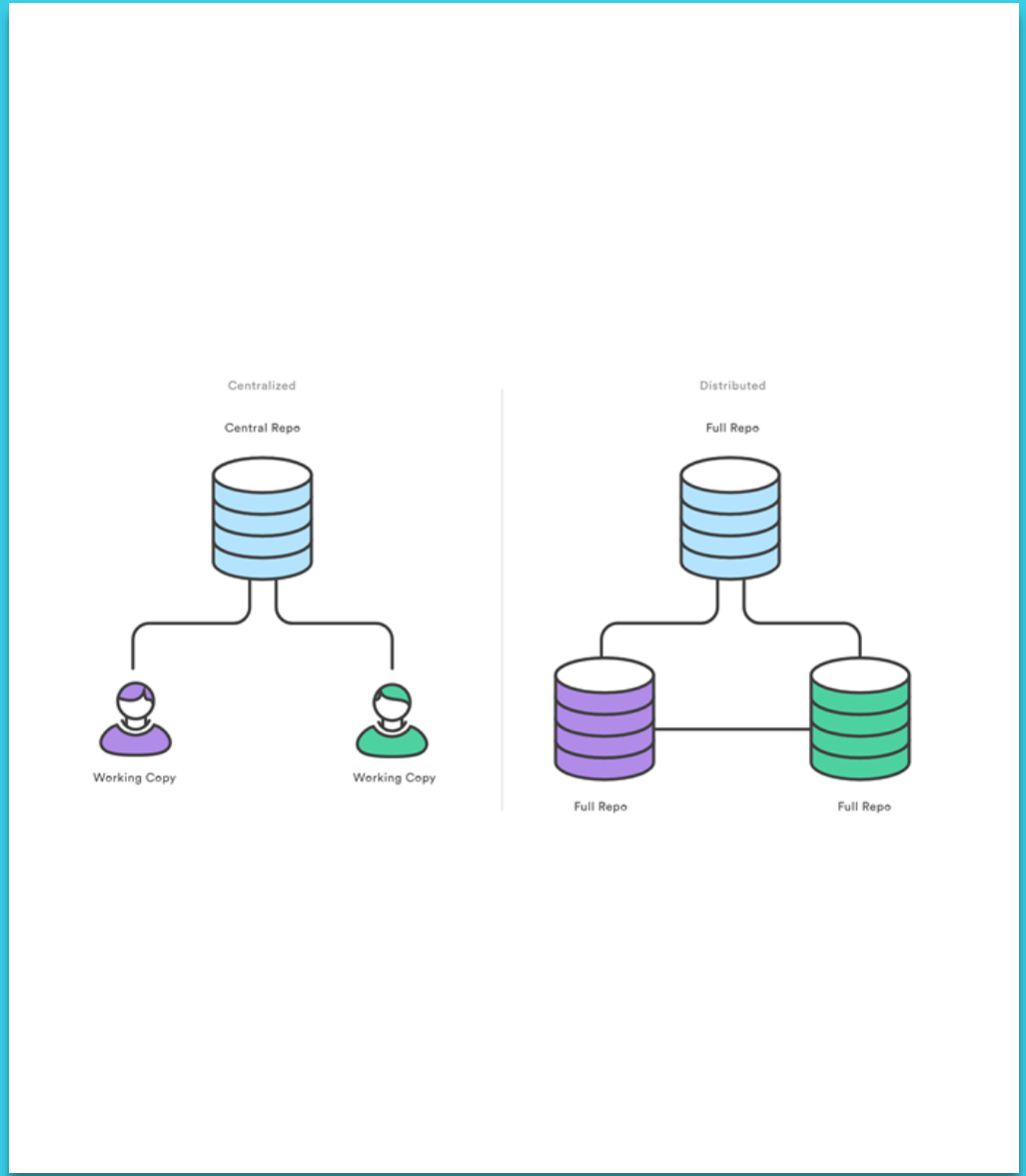
- Getting new changes from a repository is called "pulling." Moving your changes to a repository is called "pushing." You move changesets (changes to file groups as coherent wholes), not single-file diffs.

Distributed Source Control

- Doing actions other than pushing and pulling changesets is fast because the tool only needs to access the local storage, not a remote server.
- Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
- Everything but pushing and pulling can be done without an internet connection. So, you can work on a plane, and you won't be forced to commit several bug fixes as one large changeset.
- Since each programmer has a full copy of the project repository, they can share changes with one, or two other people to get feedback before showing the changes to everyone.

Distributed Development

- In TFVC, each developer gets a working copy that points back to a single central repository. Git, however, is a distributed version control system. Instead of a working copy, each developer gets their local repository, complete with an entire history of commits.
- Having a complete local history makes Git fast since it means you do not need a network connection to create commits, inspect previous versions of a file, or do diffs between commits.

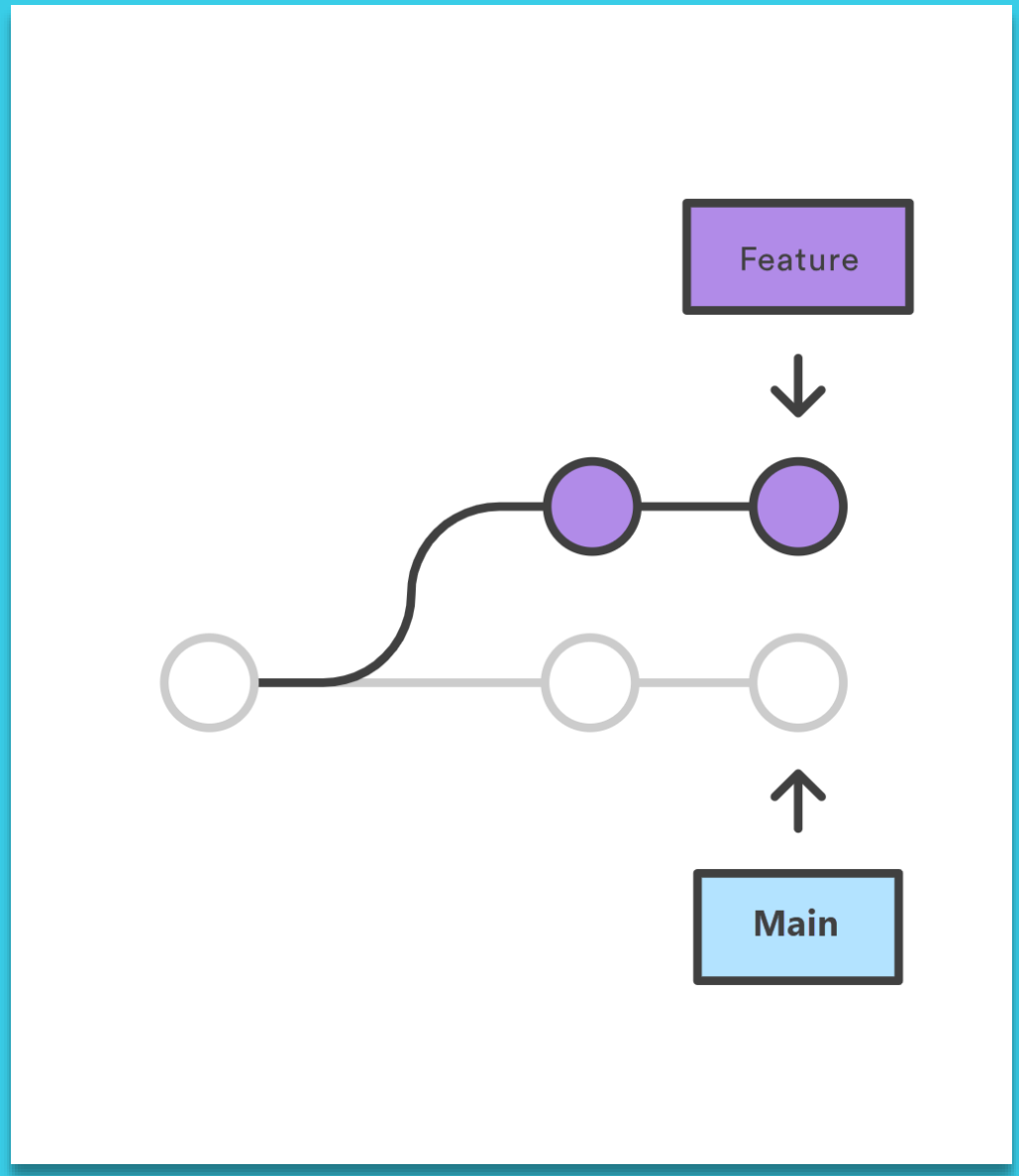


Distributed Development

- Distributed development also makes it easier to scale your engineering team. If someone breaks the production branch in SVN, other developers cannot check in their changes until it is fixed. With Git, this kind of blocking does not exist. Everybody can continue going about their business in their local repositories.
- And, like feature branches, distributed development creates a more reliable environment. Even if developers obliterate their repository, they can clone from someone else and start afresh.

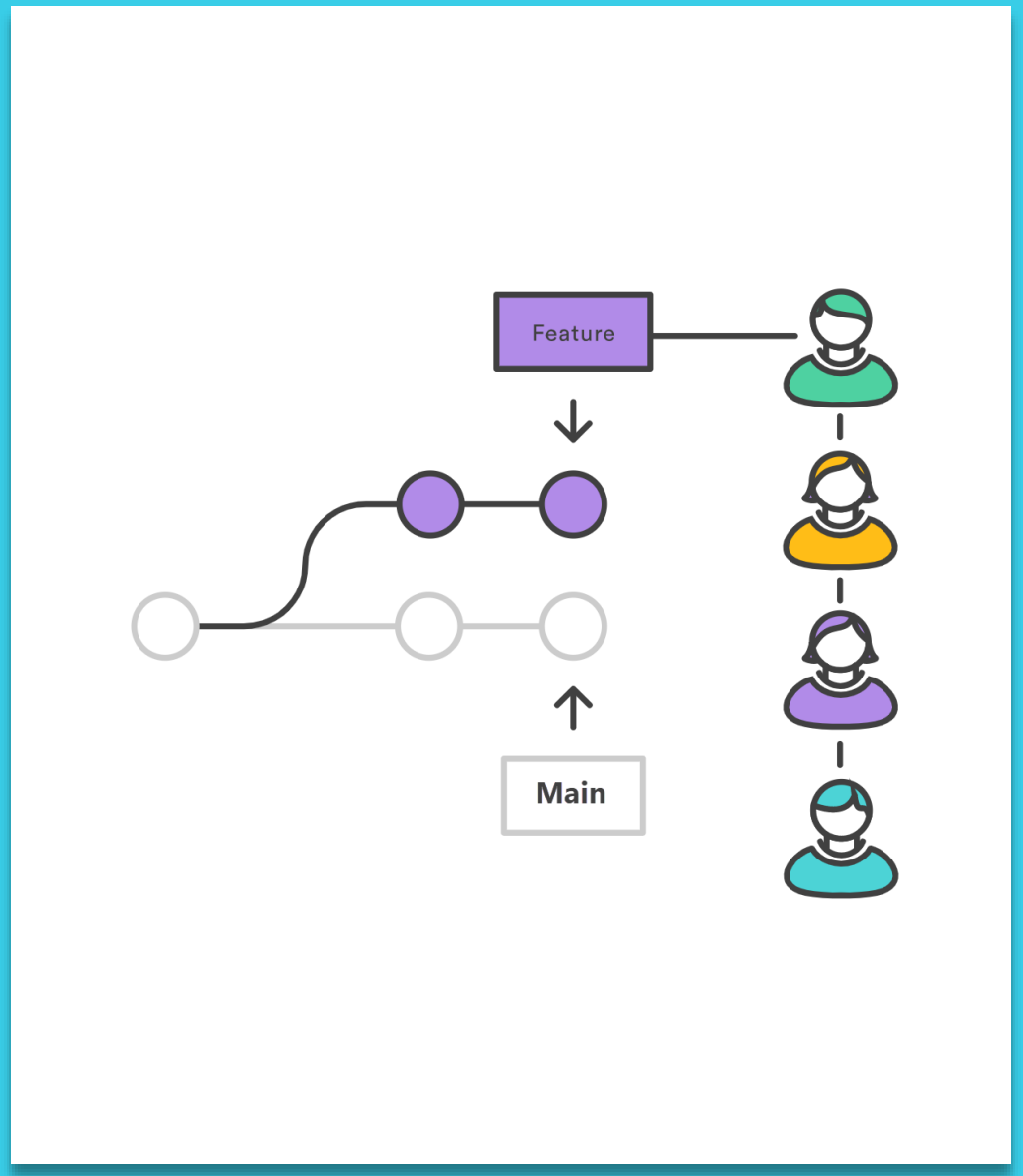
Trunk based Development

- One of the most significant advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge.
- Trunk-based development provides an isolated environment for every change to your codebase. When developers want to start working on something—no matter how large or small—they create a new branch. It ensures that the main branch always contains production-quality code.



Pull Request

- A pull request is a way to ask another developer to merge one of your branches into their repository.
- It makes it easier for project leads to keep track of changes and lets developers start discussions around their work before integrating it with the rest of the codebase.
- Since they are essentially a comment thread attached to a feature branch, pull requests are incredibly versatile.
- When a developer gets stuck with a complex problem, they can open a pull request to ask for help from the rest of the team.
- Instead, junior developers can be confident that they are not destroying the entire project by treating pull requests as a formal code review.



Git for Large Files

- Git works best with repos that are small and do not contain large files (or binaries).
- Every time you (or your build machines) clone the repo, they get the entire repo with its history from the first commit.
- It is great for most situations but can be frustrating if you have large files.
- Binary files are even worse because Git cannot optimize how they are stored.
- That is why Git LFS was created.
- It lets you separate large files of your repos and still has all the benefits of versioning and comparing.
- Also, if you are used to storing compiled binaries in your source repos, stop!

Demo

1. Open the Command Prompt and create a new-working folder:
   mkdir myWebApp
   cd myWebApp

2. In myWebApp, initialize a new Git repository:
   git init

3. Configure global settings for the name and email address to be used when committing in this Git repository:

```
git config --global user.name “Your Name"
git config --global user.email “youremail@domain.com"
```

4. Create a new ASP.NET core application. The new command offers a collection of switches that can be used for language, authentication, and framework selection. More details can be found on Microsoft docs.

```
dotnet new mvc
```

Launch Visual Studio Code in the context of the current-working folder:

```
code .
```

5. When the project opens in Visual Studio Code, select Yes for the Required assets to build and debug are missing from 'myWebApp.' Add them? Warning message. Select Restore for the There are unresolved dependencies info message. Hit F5 to debug the application, then myWebApp will load in the browser, as shown in the following screenshot:

Demo

If you prefer to use the command line, you can run the following commands in the context of the git repository to run the web application.\

```
dotnet build
dotnet run
```

You will notice the ".vscode" folder is added to your working folder. To avoid committing this folder to your Git repository, you can include it in the .gitignore file. Select a file from the ".vscode" folder, hit F1 to launch the command window in Visual Studio Code, type gitIgnore, and accept the option to include the selected file in the new .gitIgnore file.

To ignore an entire directory, you need to include the name of the directory with the slash / at the end.

Open your .gitignore, remove the file name from the path, and leave the folder with a slash, for example, .vscode/*.

Demo

6. To stage and commit the newly created myWebApp project to your Git repository from Visual Studio Code, navigate the Git icon from the left panel. Add a commit comment and commit the changes by clicking the checkmark icon. It will stage and commit the changes in one operation:

7. Now launch cmd in the context of the git repository and run git branch --list. It will show you that only the main branch currently exists in this repository. Now run the following command to create a new branch called feature-devops-home-page

```
git branch feature-devops-home-page
git checkout feature-devops-home-page
git branch --list
```

Demo

You have created a new branch with these commands and checked it out. The --list keyword shows you a list of all branches in your repository. The green color represents the branch that is currently checked out.

8. Now navigate to the file ~\Views\Home\Index.cshtml and replace the contents with the text below.

C#

```
@{
        ViewData["Title"] = "Home Page";
    }
```

```
<div class="text-center">
        <h1 class="display-4">Welcome</h1>
        <p>Learn about <a href="https://azure.microsoft.com/services/devops/">Azure DevOps</a>.</p>
</div>
```

9. Refresh the web app in the browser to see the changes.

Demo

- In the context of the git repository, execute the following commands. These commands will stage the changes in the branch and then commit them.

```
git status
git add .
git commit -m "updated welcome page.“
git status
```

- In Git, committing changes to a repository is a two-step process. Running: add . The changes are staged but not committed. Finally, running the commit promotes the staged changes in the repository.

Demo

11. To merge the changes from the feature-devops-home-page into the main, run the following commands in the context of the git repository.

    git checkout main

    git merge feature-devops-home-page

12. Run the below command to delete the feature branch.

    git branch --delete feature-devops-home-page

13. To see the history of changes in the main branch, run the command git log -v
14. To investigate the actual changes in the commit, you can run the command git log -p