MODULE 01 - 047 : Distributed Version Control Systems - Git

A **Distributed Version Control System (DVCS)**, like Git, is a type of **version control system** where **every developer has a full copy of the entire codebase and its history** locally.

This contrasts with older systems, like **Subversion (SVN)**, where **a single central server stores the codebase, and developers rely on it** for most operations.

How a DVCS Differs from a Centralized Systems

Feature	Centralized VCS	Distributed VCS
Dependency on Server	Requires server for most tasks.	Most operations are local.
Backup	Central repository only.	Every clone is a full backup.
Collaboration	Sequential access to files.	Concurrent collaboration.
Branching and Merging	Limited and complex.	Easy and efficient.
Offline Support	Minimal or none.	Fully supported.

How Does a DVCS Work?

In a DVCS: 1. **Local Cloning**: When you clone a repository, you download a complete copy of all the code, including its full history (commits, branches, etc.), onto your computer.

2. **Local and Remote Operations**:
   - Developers can perform most operations (like committing changes, creating branches, and viewing history) **locally**, without needing access to the central server.
   - Changes are shared by **pushing** to or **pulling** from a remote repository.
3.

Branching and Merging: Developers work on independent branches, which are later merged into shared branches in the remote repository.

Git

Git is a **free and open-source distributed version control system**.

Bruce Perens (born around 1958) is an American computer programmer and advocate in the free software movement. He created The Open Source Definition and published the first formal announcement and manifesto of open source.

- A **distributed system** allows multiple developers to work on the same codebase simultaneously.
- A **main repository** (often called the `master` or `main` branch) stores the entire project.
- Developers can **clone** the repository to their local machines, make changes, and then **push** updates back.

```
master/main ---- A ---- B ---- C (current state)
                  \
                   feature-branch ---- D ---- E (in progress)
```

- A, B, C: Main branch commits.
- D, E: Feature branch commits to be merged.

Git uses **branches** to manage simultaneous development efforts.
1. The `main` (or `master`) branch is the primary version of the code. 2. Developers create **feature branches** to work on specific tasks without affecting the main codebase. 3. After completing a feature, the branch is merged back into the main branch.

Branch Workflow

1. **Create a branch** for your feature:

   ```
   git checkout -b feature/my-new-feature
   ```

2. **Work on your branch**, then **commit changes**:

   ```
   git add .
   git commit -m "The explanation about what has be done"
   ```

3. **Push the branch** to the remote repository:

   ```
   git push origin feature/my-new-feature
   ```

4. **Create a pull request (PR)** for your branch to be reviewed and merged. *** ## VERSION CONTROL Git enables you to:

- **Track the changes**
- **Revert easily** by rolling-back when needed.

Version Milestones

Git stores changes efficiently:

- Instead of duplicating the entire codebase, it **tracks only the differences**.
- This is visualized as a series of commits (versions), often represented as nodes in a graph.

Git Workflow: Example

Cloning the Repository

```
git clone <repository_url>
```

Create a New Branch

```
git checkout -b feature/new-login-page
```

Make Changes and Commit

```
git add .
git commit -m "Implemented login page UI"
```

Push the Branch

```
git push origin feature/new-login-page
```

Merge Changes

1. Open a pull request on GitHub or another platform.
2. After review, merge into the main branch.

Video speech lesson

Git Overview

Welcome to this course on git fundamentals. In this guide we are going to walk through what git is and how we can use it in our applications.

Whenever I'm learning a new technology it helps me to walk through the definition. And that's how we're going to start off in this guide. So what is git? Git SCM which is the maintainer of the git code library says that git is a free and open

source distributed version control system.

“Git is a free and open source distributed version control system.” – git-scm

And if none of those words or very few of them made sense to you do not worry we are going to dissect this definition and we’re going to pick out the keywords and walk through those so we’re specifically going to discuss what it means for a project to be open source, what it means for it to be distributed, and lastly what version control is, and how you can use a version control system in your applications.

Now looking at the first phrase open source. What this means is that it’s free to use and it’s developed by a community of developers.
Many of the most popular programming languages and frameworks out there today are open source.
And the reason for that is because one it’s free to use in the very early days of computer science many of the programming languages and operating systems are created were proprietary which meant that you actually had to pay the company that created the language or the software in order to use it.

Open source means that you’re able to use it completely for free which means that there is a much lower barrier to entry. You don’t have to save up to pay for a software license the way that you used to have to do for say the Windows operating system or a Windows Server System with git its open source which means that you could go right now download onto your system for free and start working on projects.

Now the other side of that is that it is also developed by a large community of developers all over the world.
So that means that you have this team of talented computer scientists that are working on the project.
They’re adding new features they’re integrating security patches and you get the best breed of all kinds of different developers that are distributed over the entire world.

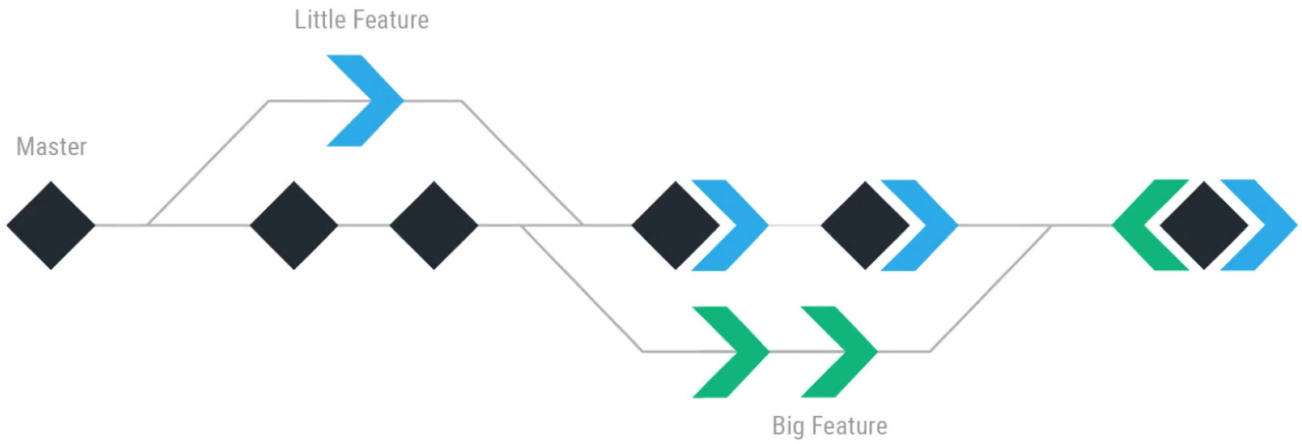
Now that we’ve reviewed the concept of open source let’s move on and let’s see what it means when we say that git is distributed with this essentially means is that gets keeps a centralized code repository and this gives you a tremendous amount of power not only for your own development but also for working with teams on projects.
There may be many projects where you work on the entire application completely by yourself.

However in most large applications and whenever you’re working for a company you’re going to be working as part of a team and so that means you may have one code base and you may have a few or even a large number of other developers all working on the same set of code files.
And so if you do not have a centralized place where those code vials could reside and where you could track changes then this could lead to a number of conflicts.

Imagine a scenario where you and another developer are working on the identical file and you make changes to it and you’re not aware of the changes and the other developer isn’t aware of the changes you’re making, that could break the entire application.
And so what git gives us is the ability to have a centralized workflow in how to process those kinds of changes.

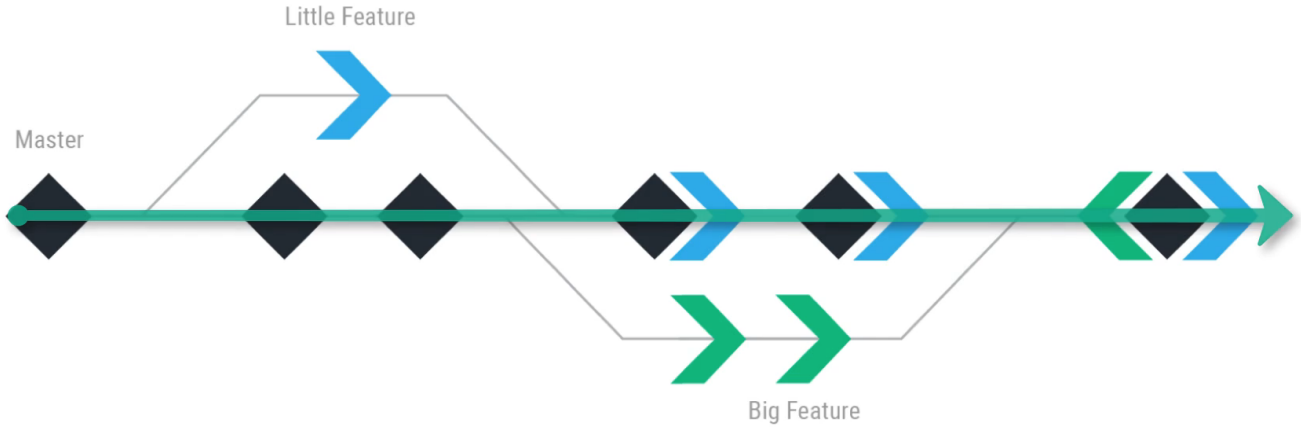
Let’s take a case study approach to this in this diagram right here.

Distributed:



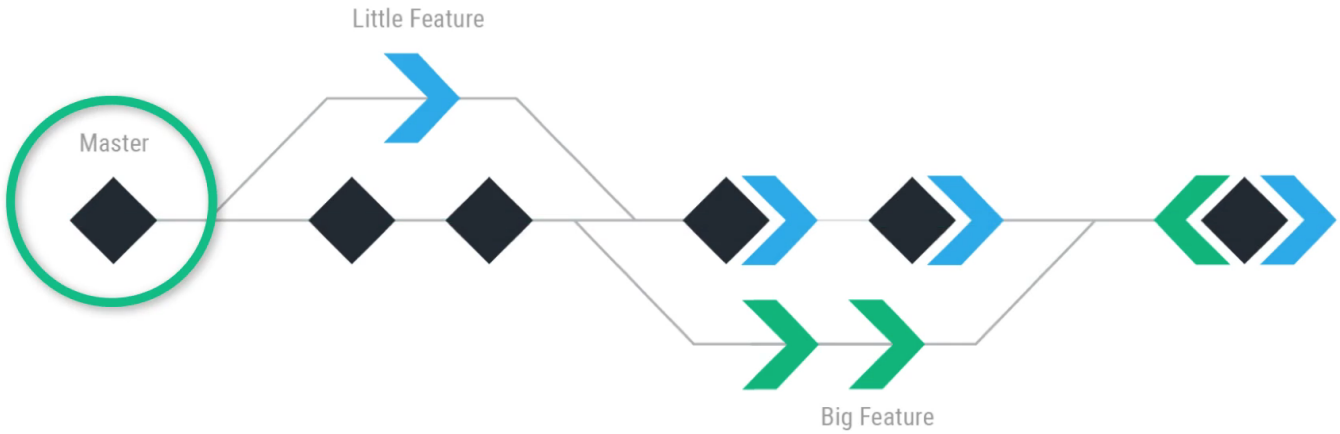
You can see that we have all of these little icons and they're all following a line going right through the center.

Distributed:



… right on the far left-hand side, we have what's called the master branch.

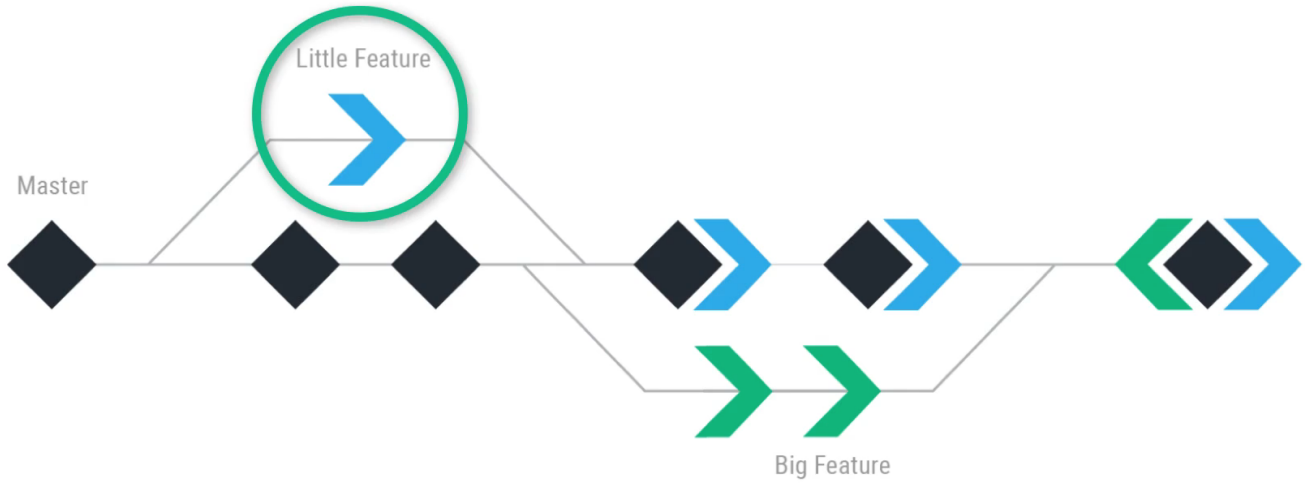
Distributed:



… and we're going to go all into what these names represent.
Right now I just want to talk about how git is a distributed system to give you a high-level idea on how it works.
So we have all of these little diamonds and what these represent each one of them represents a significant change in the code, in fact, it represents a version that will go into shortly.

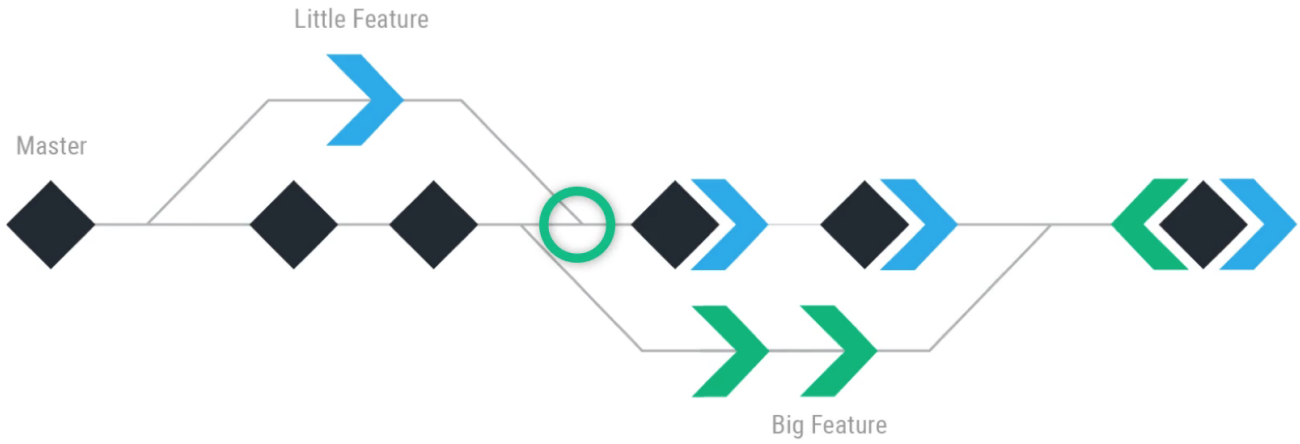
But imagine a scenario where you're working on some feature and that's represented by the little feature arrow there:

Distributed:



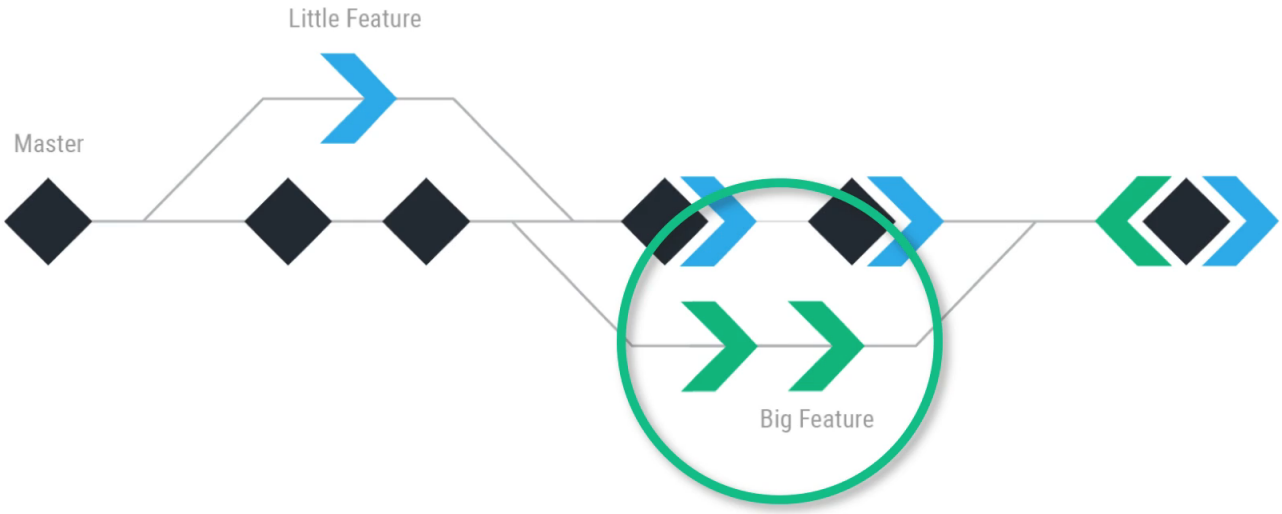
If you want to work on that feature you can actually perform what is called a check out of the code where you pull the entire code onto your system and you implement a feature.
And then from there, you can check it back into that centralized repository.

Distributed:



Imagine that one of your co-workers is working on an even bigger feature and that's represented by those green arrows there at the bottom.

Distributed:



What you can do with git is both of you are able to work on the identical project at the same time checking out different parts of the application and then submitting those back into that centralized repository.
So right there in the center that master branch straight line going through the center that represents the centralized code repository that the entire team has access to.

And then you as a developer a part of that project can work on different components of the application. While your other team members are as well and you're able to track the changes on the entire code base and they're able to track your changes.
So at a high level that's what it means when we say that a git system is distributed.
It means that there's a centralized repository where you and other team members can access the code.

Let's move on to what a version control system is.

Distributed:

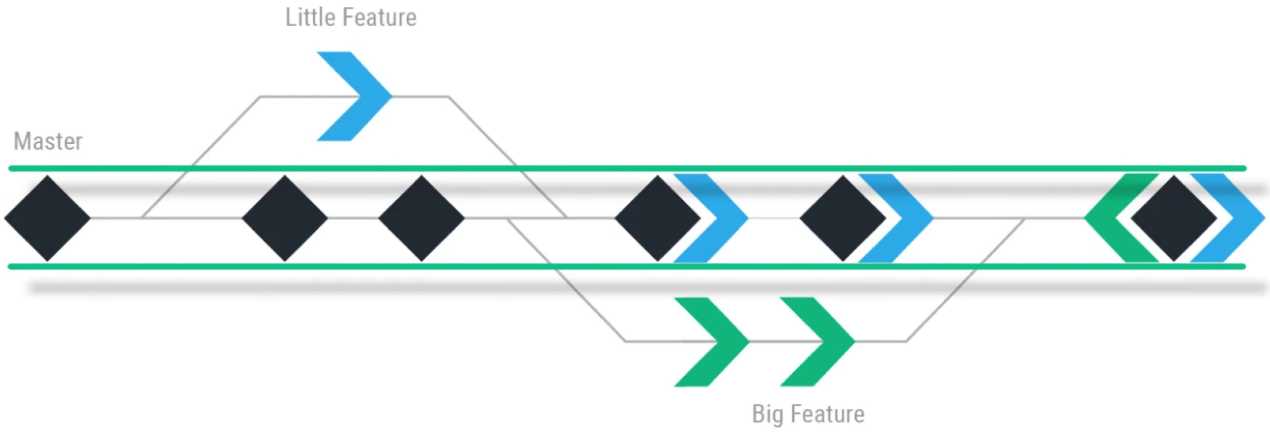
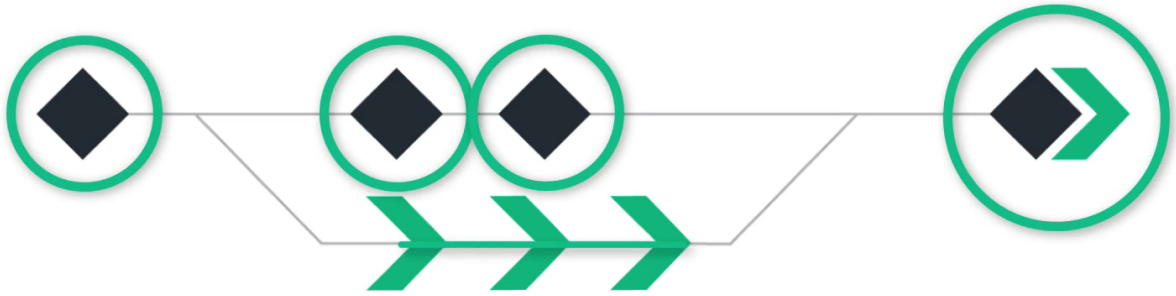


Figure 1: 01-047_IMG8

This also has an acronym that you will see and it's pretty common which is the VCS and that just stands for version control system and all it means is it gives you the ability to create milestones for a project.
There are many times especially in larger applications where you may want to build out a significant feature but there's a chance that that feature might break other parts of the application.

And so what a version control system allows you to do is to create milestones as you can see in this diagram right here. Each one of the diamonds and then the arrows represents a significant code change and it also represents a specific version of the code.

Version Control System:



So as great as your new feature may be, if you happen to make a mistake and something happens in the application where you break it you're going to be able to switch back to an older version of the code until you have it fixed and that way in case you do push up some change that breaks the entire system or shuts down the servers or something like that.
Instead of having to go through and manually make all of those code changes to try to revert it yourself what you can do with the version control system is you can tell that you simply want the entire application to revert back to one of those previous changes.

And this makes it much more efficient for you to take the entire code base and to switch it back and restore it to a working version.

Version Control System:

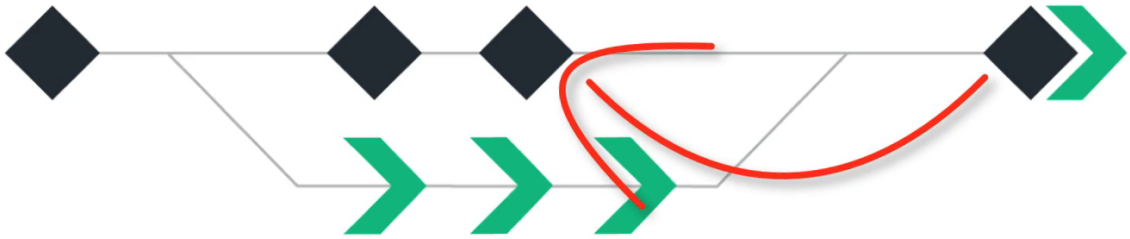
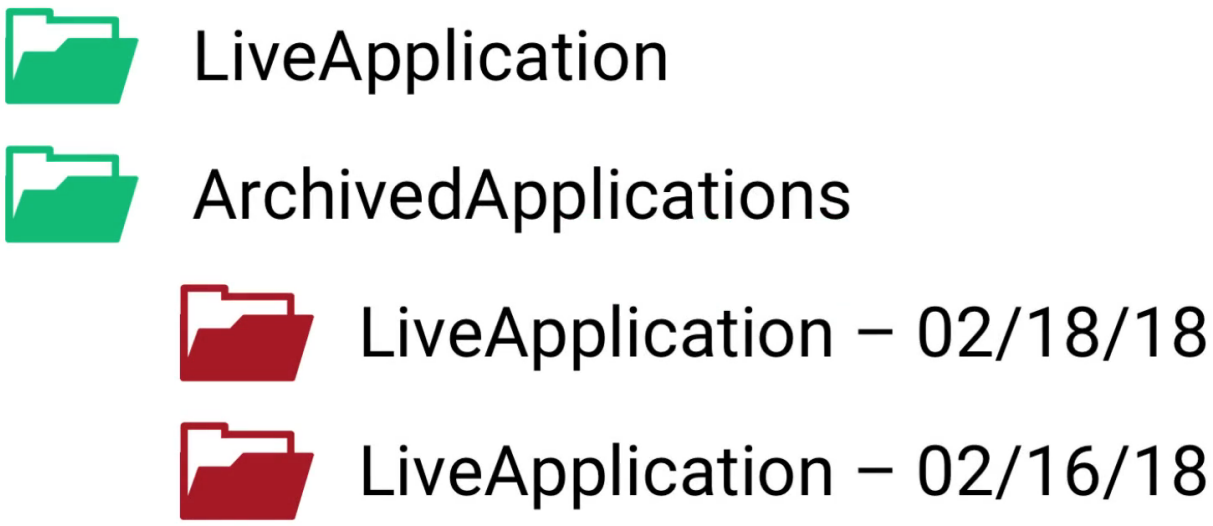


Figure 2: 01-047_IMG10

When I started working on large applications years ago and this was before git was even popularized in the development community.
I had to create my own kind of version management control and many developers who worked years ago had to do this as well.
And it was not a fun way to build applications and it looks something like this.

Version Control System:



I would have on the server the live application and then every time that I wanted to create a significant change in the application so if I was building out a new feature then to be safe I would copy the entire application into another folder on the server and I would call it something like archived applications and I would give it a day and sometimes even a time to represent at what stage the application was the last time before I made these significant changes.

The problem with this is I had to copy literally the entire application. So this was very resource intensive and it also is very prone to error.
Imagine a scenario where I changed 15 files if one of those 15 was the cause for the error it was very difficult to go in check

what the difference was the git system only focuses on tracking the changes that you make.

So each new version in git does not include the entire application and a new version of that application.
Instead, it only tracks those specific changes you made so you can track it much more efficiently.

So right here we have an example of a real-life application I have

Version Control System:

This is a significant version change and instead of copying the entire application all that git did was it tracked what the changes were and it gives a great visual for how you can manage that.
So as you can see on the left-hand side there where it is highlighted in red what good is saying is this the previous version. This is where all the changes occurred.

And if you look on the right-hand side where you have the plus sign and it's highlighting green.
This is where I'm able to see and all of my team members are able to see exactly what those changes were and if one of these changes had broken part of the application then it would be very straightforward to simply revert back to the version before the bug was introduced into the program.

And as you can see this is a much more targeted and efficient way of managing what the code changes were.
Instead of copying the entire application and then having to dig through what could be hundreds or even thousands of code files to see where a break was introduced,
I'm able to see literally line by line where each one of the changes occurred.
And that allows me to go back into the code base fix any changes and also to track all of the work that my team is performing.

So, in summary in answering what good is open source which means it is free to use and it's developed by the open source community distributed which means that you have access to a centralized repository of your codebase where all of your team members can analyze that code pull it and continually add new features to that centralized repository.
And then lastly it is a version control system which means that you can create milestones inside of your codebase, you can track changes and then you can efficiently revert back any time that you need to.