Version Control Systems

**What is Version Control?**

By keeping track of the changes that we make to our files, Version Control Systems (VCS) let us know when a file changed, who changed it, and also let us easily roll back those changes. It makes collaboration easier by allowing us to merge changes from lots of different sources.

There are many different version control systems, each with their own implementation and with their own advantages and disadvantages. But, no matter how the VCS is implemented internally, they always access the history of our files. Let us retrieve past versions of the file or directory and see who changed which files, how each file was changed and when the file was changed. On top of this, we can make edits to multiple files and treat that collection of edits as a single change which is commonly known as a, commit. A VCS even provides a mechanism to allow the author of a commit to record why the change was made, including what bugs, tickets or issues were fixed by the change. This information can be a lifesaver when trying to understand a complex series of changes, or to debug some obscure issue.

**Version Control and Automation**

A VCS can be invaluable, even in a one-person IT department. A VCS stores your code and configuration. It also stores the history of that code and configuration. A version control system can function a lot like a time machine, giving you insights into the decisions of the past. Whenever you write a commit message, after making a change, it's as if the current version of yourself is explaining your decisions to a future you or others who might work on the same scripts and configurations in the future. This can help you avoid finding yourself staring at a piece of code that you or someone else wrote three months ago and puzzling over how it works or even why it exists. With a VCS, you can view, track and select snapshots from the history of your project. So nothing you do is lost, and since we can use a VCS to store both code and configuration files, we can make the overall IT systems more scalable and reliable.

For example, let's say you've stored the DNS zone file for your company in a VCS and in case you don't remember, a DNS zone file is a configuration file that specifies the mappings between IP addresses and host names in your network. When you update the zone information, always use good explanatory commit messages. That way, you'll have access to meta information about your new IP addresses and host names present in the zone file. Like when they were added and for what purpose. If anything breaks after you add a new entry, you can rely on the VCS to tell you what the file looked like before the change.

You can then revert to the old version quickly, so you can fix the problem fast and figure out what went wrong later. This functionality enhances the reliability of systems you operate. Because of the audit trail provided by the VCS, you know exactly what version of the zone file to rollback to, which reduces the time it takes to fix the problem. It's generally better to quickly roll back first and stop errors before spending time figuring out what went wrong. You can curb the fix after the bleeding has stopped. Figuring out the bug might take up valuable time or worse, your first attempt at a solution can have its own bugs.

Let's look at a different example. The configuration for a DHCP damon can be replicated in two or more machines, where one acts as a primary server and the other one acts as standby machine. The standby machines won't do much while the primary is up. But if the primary goes down for any reason, a standby machine can become primary and start responding to DHCP queries. For this to work, the configuration files on all machines need to be identical. This is because the DHCP protocol doesn't provide a way for standby machines to get an up-to-date version of the configuration files and the way DNS does. To deal with this, we can keep the up-to-date version of the DHCP configuration in a version control system and have the machines download the configuration from the VCS. This means all the machines will have the exact same files. That's already handy enough. But after using it for a while, you're bound to see other benefits.

Say you get an urgent alert over the weekend, telling you that your DHCP server isn't responding to any queries. You look at the history of the changes and you find that one of the changes added on Friday evening, included a duplicated entry causing the server to misbehave. By using a VCS, you can easily roll back the change and have the servers back to health in no time. You might come across a second unexpected benefit, when it's time to replace the server with a new one. By having all the server configuration and a version control system, it's much easier to automate the task of deploying a new server.

Summary:

Why is a version control system useful, even if it's used only by a single person?

* One of the main benefits of a VCS is that you can see the history of how files changed and understand what changed at each step and what motivated the change.
* By having each change tracked in the VCS, it's very easy to go back to previous versions when an issue with a change is discovered.

**What is Git?**

Git is now one of the most popular version control systems out there and is used in millions of projects. Unlike some version control systems that are centralized around a single server, Git has a distributed architecture. This means that every person contributing to a repository has full copy of the repository on their own development machines.

Collaborators can share and pull in changes that others have made as they need. And because the repositories are all local to the computer being used to create the files, most operations can be done really fast. If you want to collaborate with others, it usually makes sense to set up a repository on a server to act as a kind of hub for everyone to interact with. But Git doesn't rely on any kind of centralized server to provide control organizations to its workflow. Git can work as a standalone program as a server and as a client. This means that you can use Git on a single machine without even having a network connection. Or you can use it as a server on a machine where you want to host your repository. And then we can use Git as a client to access the repository from another machine or even the same one.

Git clients can communicate with Git servers over the network using HTTP, SSH or Git's own special protocol. So you can use Git with or without a network connection. You can use it for small projects with like one developer or huge projects with thousands of contributors. You can use it to track private work that you can keep to yourself or you can share your work with others by hosting a code on public servers like GitHub, Gitlab or others.

When looking for information online you might notice that the official Git website is called git-scm.com. And wonder what's the SCM at the end for? It's actually another acronym similar to VCS. It stands for Source Control Management. While both terms mean the same, we generally prefer VCS, because as we call that already, these systems can actually be used to store much more than just source code.

Summary:

What characteristics make Git particularly powerful?

* It's a distributed VCS, which means that each developer has a full copy of the repository - Because each contributor to a Git repo has a full copy of the repository, they can interact with the tracked files without needing a coordinating server. In turn, this improves collaboration.
* Repositories can be used by as many developers as needed - Because of the way Git was designed, repositories can be useful for any number of developers, from one to thousands.