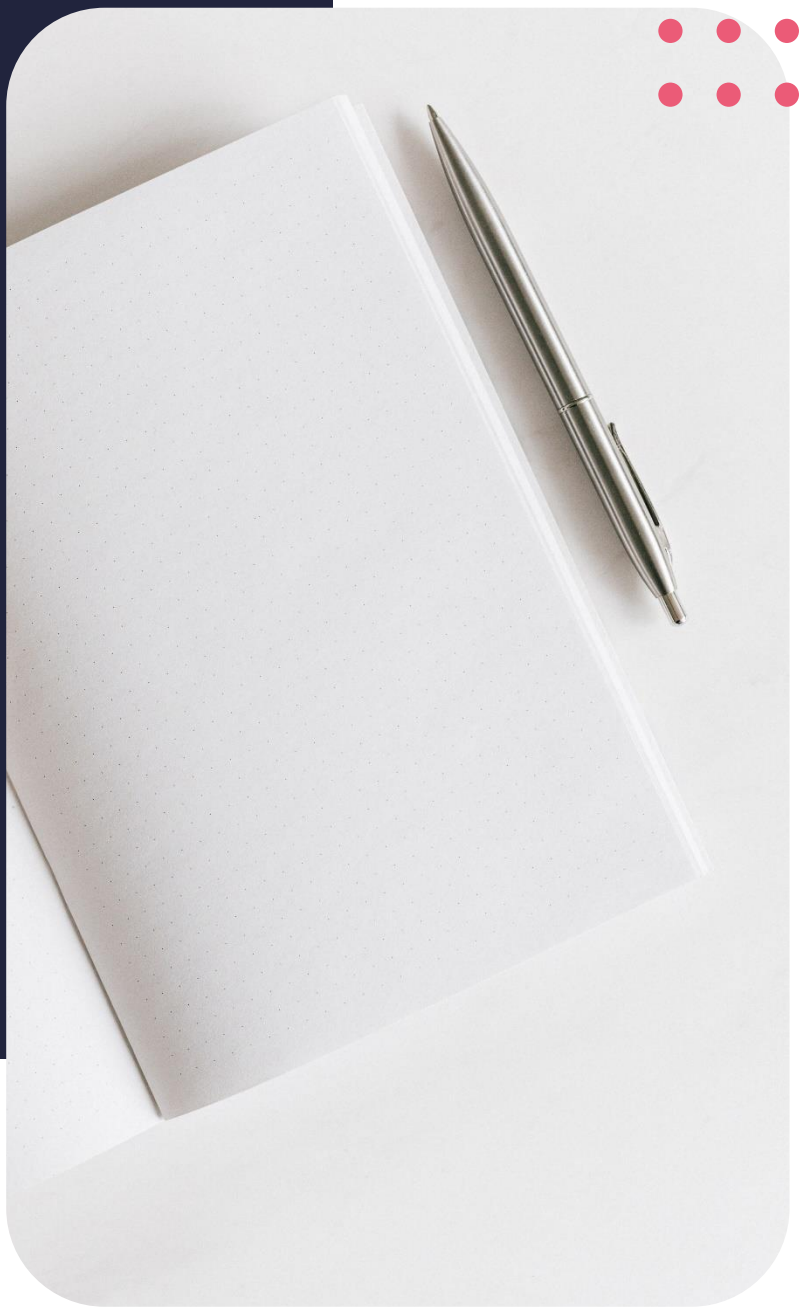
Diploma in Computer Science

Scaling Up

Contents



Lesson objectives

By the end of this lesson, you should be able to:

- Explain how code is handled on a large scale
- Explore version control
- Outline the benefits of version control
- Discuss version control systems used in industry

Introduction

Our lesson today is entitled scaling up and it is all about the strategies that are used when you are writing large scale code. When many developers and millions of lines of code are involved, things are done a little bit differently and this is what we are going to explore today.

Version control systems in software development

If you have used the internet, which is pretty obvious because you are watching this lesson on the Internet, there's a good chance that you have directly interacted with a version control system. These are systems that ensure that many people can work on one thing and not get tangled up in all the changes and versions. This also makes sure that we don't lose stuff. Let's look at why exactly this is such a big deal in software development.

Life before version control

Programming today is a very different picture from what it was 50 years ago. Back in the day, 1000 lines of code was an impressive feat! Yes, you heard me right, 1000 lines! You are probably laughing to yourself and thinking that you can sit on a computer right now and type 1000 lines of code without cracking your head. You can and it is entirely possible! In the early days, computers were notoriously difficult to program and couldn't really do much even if you could program them. This meant that programs couldn't get any bigger than a few 100 lines. In contrast, add modern operating system can contain upwards of 1,000,000 lines of code. Shoo! There just has to be a way of keeping track of all of this.

Life with version control

When developing software, especially using agile methodology, there are a lot of versions of the source code. The code itself can be a LOT of code. Remember the million lines of code that we just talked about? Ultimately, only one version of the software is released for use in the target environment while developers are working on the project; they use what is called a version control system.

This ensures that whenever they are writing their code and they decide to change something or remove it entirely, there is always a way of going back without doing a ton of work.

The need for a logical way to arrange and monitor revisions has existed for almost as long as writing has existed, but version control became even more essential, and complex, when the age of computers began. The numbering of **book editions** and of specification revisions are examples that date back to the print-only era. Fast forward to the modern day, the **most elaborate** version control systems are the ones used in software development, where a team of people may concurrently make changes to the same files.

The first popular version control system was a proprietary UNIX tool called **SCCS** (Source Code Control System) from way back in the 1970s. It was followed by the **Revision Control System**, and later Concurrent Versions System. Fast forward to today, the most popular version control systems are **Subversion and GIT.** GIT, especially, is so popular that you hardly ever talk about software development without mentioning it.

Any time you hear the words **version control**, source control, source code management systems, or revision control systems, know that they all refer to the same thing. They are all methods, or more appropriately, systems that keep multiple versions of your files, so that when you modify a file you can still access the previous revisions.

We have all come across tools that help in **recording chan**ges made to files. You may have already seen this in cloud services such as **Google** drive, which allow you to download previous versions of files. Pretty much any and every cloud service that stores documents incorporates a version control system in one form or the other.

It may sound like it isn't such a big deal but think about the last time you accidentally wrote over your photos after editing them. when using a version control system, you would have the original photo, along with the new edited file, and that would pretty much save your bacon if you were working on a super important project where you can't afford to lose anything.

Version control systems (VCSes) are usually stand-alone applications, but revision control is sometimes built into other software as part of the package, such as word processors and spreadsheets. Microsoft office provides such a feature, even Windows, Mac and other operating systems provide such functionality. Some large websites that allow community collaboration have version control, a shining example here being Wikipedia. Content management systems like WordPress also come with version control as standard. Revision control enables users to revert a document to a previous revision, and this is really great news for editors, as they can track each other's edits, correct mistakes, and undo unwanted changes such as spamming and vandalism, which are now quite common as you may have already encountered on the internet.

Version Control systems in software engineering

In software engineering, version control is a **class** of systems responsible for managing changes to computer programs, documents, and pretty much any fairly large collection of data or information. So basically, version control is a **component** of software configuration management.

Identifying changes

Changes typically carry some sort of identification that is used to tell them apart. This identification is usually in the form of a letter, number or a mix of both. If, for instance, a file created the edited twice, then the 2nd version of the file can be called 'Rev 2'. Each revision has a timestamp, which is literally what it sounds like, that is, the date and time at which the change was carried out as together with the name of the person that made the change. This provides functionality such as comparison, revision, reverts and merges.

Imagine three of you are working on a document, then one accidentally deletes the document at 11pm because they were working late and got tired and careless? Yikes! If you have a version control system, you can easily revert to the last copy of the document that was saved. The exact same thing applies to software development. Without using version control, you are literally skating on a knife! Any rookie mistake and all your clever algorithms are GONE! A version control system provides backup, especially if the code has a lot of reverts. It also allows software teams to work efficiently as the team scales to include more developers. Each member of the team has all the copies of files they need and can look at the file history if they need something that was removed in a later revision.

Here are a few things that an efficient control system allows us to do:

- Complete change history of files – this means that every file has all of its iterations preserved.
- Work concurrently – several developers can work on the same project at the same time.
- Branch and merge - this refers to creation of a separate branch in order to develop a new feature or work on a discovered bug. This is done so that whatever work is done, especially if it is experimental, does not cause disruptions and unwanted alterations in the master branch. If everything goes according to plan, then the new changes will be merged to the master branch and if it doesn't it will just simply be discarded. This is particularly important for large projects.

Track development -a version control system allows the development team to track the development of the software package. Since there are many versions of the software package that are readily available, the software development team can keep track of every little thing that is created of changed in the software package.

Types of version control systems

There are three main types of version control systems: the localised version control system, the centralised version control system, and the distributed version control system. Let's look at each one in a bit more detail.

Localised is the simplest of all version control systems and the most common. This simplicity is not without its drawbacks. We all like nice, simple things, although the simplicity carries quite a lot of risk. Data can be lost quite easily, as you may accidentally overwrite files. We've all probably been at a point where we lost all photos from a particularly cherished moment all because of an absent-minded click.

The high likelihood of losses drove the development of the centralised version control system approach.

The centralised version control system attempts to do away with the risks of having precious code stored on one machine with a human element in the loop. This allows developers to access the files they need from a single machine. Unlike the localised system, files are readily available to all developers as all on a central location.

Everyone on the system has access to what their peers are doing on the project.

A certain degree of control is available to administrators on the project.

It does, however, not deal with the fact that there is a single point of failure, which is the same thing we had to deal with in localised systems.

Centralised Version Control Systems use a central server to store all the database and any other data that is used in the team's collaboration. This is fertile ground for a single point failure, which won't be such a nice thing after you've written ten thousand lines of complex code! This is one of the strongest driving points of our next stop, the Distributed Version Control System.

In a Distributed Version Control System each of the developers, or more broadly, users have a local copy of the entire repository. So, clients essentially check out a snapshot of the latest file on the repository. The local repository contains all the files and metadata that is found in the main repository.

DVCS allow for automatic management of branching and merging. This means that except for pushing and pulling, the developer does not need to worry about those and other rudimentary housekeeping operations. This also means that the developer can work offline, since they have a copy of the entire repository on their machine, which also serves as a backup, which is also a nice little thing as the backup is no longer located on a single machine. Technically, every machine connected to and communicating with the main server is a backup location. Although the data is technically stored centrally, these systems do not necessarily depend on the central server to store all the versions of a project file. Every checkout is a full backup of all the data.

Common terms in version control systems

As with any system, there is a set of standard terminology that comes with version control systems. We will now look at a handful of the most common terms and dive into a bit of detail as to what they entail.

Traditional revision control systems use a centralised model where all the revision control functions take place on a shared server. Without a defined access control system, if two developers try to edit the same file, the developers may end up overwriting each other's work. Centralised revision control systems attempt to tackle this problem head on by using two different source management models, namely file locking and revision merging.

File locking is the simplest method, which works by denying subsequent access requests and allows access to one request at a time. This essentially means that only one developer can access the copies on the central server at a

time. It is only when the developer ‘checks out’ a file, that others can read that file, but no one can make changes to that file until the developer with the lock ‘checks in’ the updated version or cancels the checkout.

File locking has key advantages and disadvantages.

It can provide some protection against difficult merge conflicts when a developer is busy with significant changes to a fairly large file or a number of sections in a file, or even a number of files. You may already have had the question at the back of your head, that what if it takes too long to make the changes? It essentially means the files will have that exclusive lock for the entire duration and other developers may be tempted to bypass the revision control software and change the files on their machines, since they have a copy anyway. This is a perfect recipe for tangled spaghetti when the other changes are finally checked in.

Sometimes it might not be so easy to see who has a file checked out. In a large organisation, files can be left ‘checked out’ and locked and lost in the sands of time as other work is taken on, especially if the file in question is not used often. This is the problem that the next type we are going to look at attempts to solve.

Most version control systems allow multiple developers to edit the same file at the same time. The first developer to ‘check in’ changes to the central repository is the one who gets access. The system typically allows users to merge further changes into the central repository while preserving whatever changes that the first developer made when other developers check in.

Most version control systems allow multiple developers to edit the same file at the same time. The first developer to ‘check in’ changes to the central repository is the one who gets access. The system typically allows users to merge further changes into the central repository while preserving whatever changes that the first developer made when other developers check in.

Merging two files can be a slippery slope, which often requires the files in question to be simple in nature, such as text files. You may have already guessed from our previous lessons that text files are among the simplest format, and they would logically be the easiest to merge. It’s a wildly different story if the files in question are binary files. The result of a merge of two image files might not result in an image file at all! You might have encountered this if you have tried to merge files before. The second developer checking in the code will need to take care with the merge, to make sure that the changes are compatible and that the merge operation does not introduce its own logic errors within the files. This is where the shortfalls of automatic and semi-automatic merge operations start to show. Unless a specific merge plugin is available for the file types, it’s not a simple walk in the park.

The use of reserved edits can provide a different way of explicitly locking a file for exclusive write access, even when the system is capable of merging.

The terms baseline, labels and tags are common in most systems and refer to the process of identifying a snapshot ‘label the project’ or the record of the snapshot ‘try it with baseline X’. Typically, only one of the terms is used in documentation or discussion, although they are considered to mean pretty much the same thing. In most projects, some snapshots are more significant than others, such as those used to indicate published releases, branches, or milestones.

When both the term baseline and either of label or tag are used together in the same context, label and tag usually refer to the mechanism within the tool of identifying or making the record of the snapshot, and baseline indicates the increased significance of any given label or tag.

Glossary of terms used in version control systems

As with any software package, the terms used in each system may be different, but most terms that describe core functionality typically remain the same, we are going to run through this list of commonly used terms and their meaning.

This glossary of terms according to Wikipedia is a perfectly simplified version yet encompasses all the most popular terms across the board. The other great thing about this list in particular is it's as simple as it gets, let's run through the list.

So a baseline is basically an approved revision of a document or source file to which subsequent changes can be made.

Atomic operations: An operation is atomic if the system is left in a consistent state even if the operation is interrupted. The commit operation is usually the most critical in this sense. Commits tell the revision control system to make a group of changes final, and available to all users. Not all revision control systems have atomic commits; notably, CVS lacks this feature.

Branch: A set of files under version control may be branched or forked at a point in time so that, from that time forward, two copies of those files may develop at different speeds or in different ways independently of each other.

Change: A change, or diff, or delta represents a specific modification to a document under version control. The granularity of the modification considered a change varies between version control systems.

Change list: On many version control systems with atomic multi-change commits, a change list (or CL), change set, update, or patch identifies the set of changes made in a single commit. This can also represent a sequential view of the source code, allowing the examination of source as of any particular change list ID.

Checkout: To check out (or co) is to create a local working copy from the repository. A user may specify a specific revision or obtain the latest. The term 'checkout' can also be used as a noun to describe the working copy. When a file has been checked out from a shared file server, it cannot be edited by other users. Think of it like a hotel, when you check out, you no longer have access to its amenities.

Clone: Cloning means creating a repository containing the revisions from another repository. This is equivalent to pushing or pulling into an empty, newly initialised repository. As a noun, two repositories can be said to be clones if they are kept synchronised and contain the same revisions.

Commit as a noun. A 'commit' or 'revision' (SVN) is a modification that is applied to the repository.

Commit as a verb. To commit, check in, ci or, more rarely, install, submit or record is to write or merge the changes made in the working copy back to the repository. A commit contains metadata, typically the author information and a commit message that describes the change.

Conflict: A conflict occurs when different parties make changes to the same document, and the system is unable to reconcile the changes. A user must resolve the conflict by combining the changes, or by selecting one change in favour of the other.

Merge: A merge or integration is an operation in which two sets of changes are applied to a file or set of files. Some sample scenarios are as follows:

A user, working on a set of files, updates, or syncs their working copy with changes made, and checked into the repository, by other users.

A user tries to check in files that have been updated by others since the files were checked out, and the revision control software automatically merges the files, typically, after prompting the user if it should proceed with the automatic merge, and in some cases only doing so if the merge can be clearly and reasonably resolved.

A branch is created, the code in the files is independently edited, and the updated branch is later incorporated into a single, unified trunk.

A set of files is branched, a problem that existed before the branching is fixed in one branch, and the fix is then merged into the other branch. This type of selective merge is sometimes known as a cherry pick to distinguish it from the complete merge in the previous case.

Promote: Copying file content from a less controlled location into a more controlled location. For example, from a user's workspace into a repository, or from a stream to its parent.

Pull, push: Copying revisions from one repository into another. Pull is initiated by the receiving repository, while push is initiated by the source. Fetch is sometimes used as a synonym for pull, or to mean a pull followed by an update.

Pull request: A developer asking others to merge their ‘pushed’ changes.

Repository: The repository or ‘repo’ is where files' current and historical data are stored, often on a server. Sometimes also called a depot.

Resolve: The act of user intervention to address a conflict between different changes to the same document.

This list covers only the most commonly used terms. You will find others in the summary notes.

As is now the norm in this section of our course, the last stretch of our lesson is dedicated to our project. Today we are going to flesh out the pseudocode and outline what exactly the functions that we chose for our program are going to do. This is going to serve as the low-level document for our code. Our demo today is primarily centred on our project. You might have started yours already since we gave the problem statement in our previous lesson. Remember there is no universal solution, so what I am going to show you is by no means the only way that you can solve the problem. As we always say, there are a million ways to kill a cat. You might see that your pseudo code is very different from what we are going to show you, but it's pretty much will give you the same result. Remember the focus in computer science is just to make sure that the code is doing the right thing and is doing the right thing using the best way possible in terms of efficiency and effectiveness. Enough chit chat let's go!

Project

As is now the norm in this section of our course, the last stretch of our lesson is dedicated to our project. Today we are going to flesh out the pseudocode and outline what exactly the functions that we chose for our program are going to do. This is going to serve as the low-level document for our code. Our demo today is primarily centred on our project. You might have started yours already since we gave the problem statement in our previous lesson. Remember there is no universal solution, so what I am going to show you is by no means the only way that you can solve the problem. As we always say, there are a million ways to kill a cat. You might see that your pseudo code is very different from what we are going to show you, but it's pretty much will give you the same result. Remember the focus in computer science is just to make sure that the code is doing the right thing and is doing the right thing using the best way possible in terms of efficiency and effectiveness. Enough chit chat let's go!

Conclusion

With that we have come to the end of our 7th lesson, where we explored the various types of version control systems that are used in software development. Most of the time was dedicated to our projects where we looked at the intricate details of how the system is going to operate when it is completed. As we mentioned in module 1, pseudocode is the best place to start writing code as it allows you to get a good picture of what the code will do outside the bounds of the strict syntax of whatever language you'll be using. In this case, we just wrote thoughts our program will do and next time we'll be looking at changing this into actual code. You will see that it will be a far easier process to just change pseudocode into actual code rather than trying to think of how to address a particular problem while trying to adhere to the strict rules that C requires in all your source code

References

By Sheekha Jariwala (2018). What is Version Control and Version Control System? [online] TOOLSQA. Available at: https://www.toolsqa.com/git/version-control-system/

By Sheekha Jariwala (2018). What is Version Control and Version Control System? [online] TOOLSQA. Available at: https://www.toolsqa.com/git/version-control-system/

Sabyasachi Samadder (2019). Version Control Systems - GeeksforGeeks. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/version-control-systems/

Tutorialspoint.com. (2021). Continuous Integration - Version Control - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/continuous_integration/continuous_integration_version_control.htm

Wikipedia Contributors (2021). GitHub. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/GitHub

Wikipedia Contributors (2021). Version control. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Version_control

www.javatpoint.com. (2011). Git Version Control System - javatpoint. [online] Available at: https://www.javatpoint.com/git-version-control-system