DevOps Crash Course

by Oleksii Yakivchik

softserve

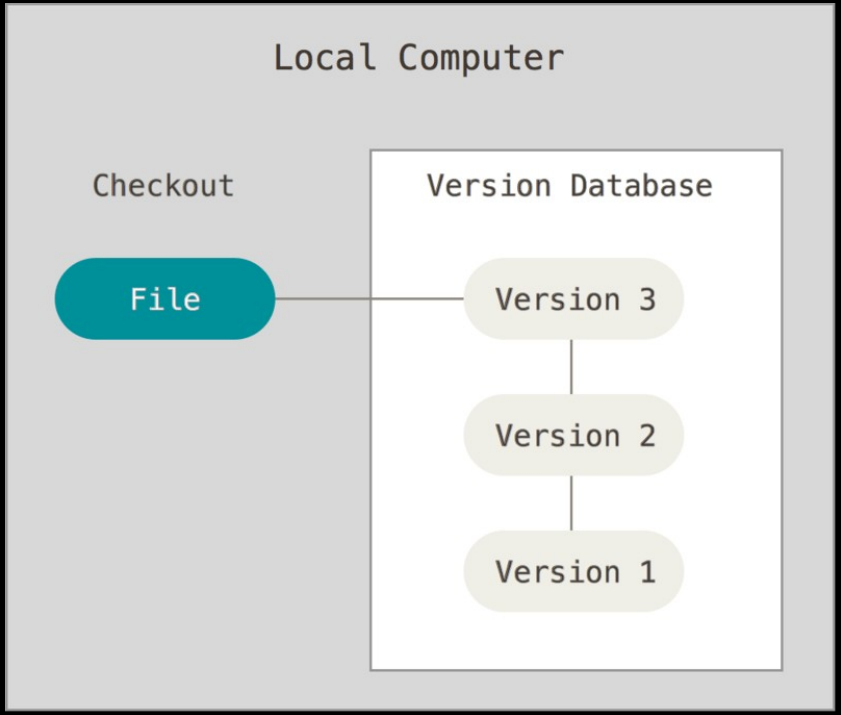
About Version Control

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

softserve

Local Version Control Systems

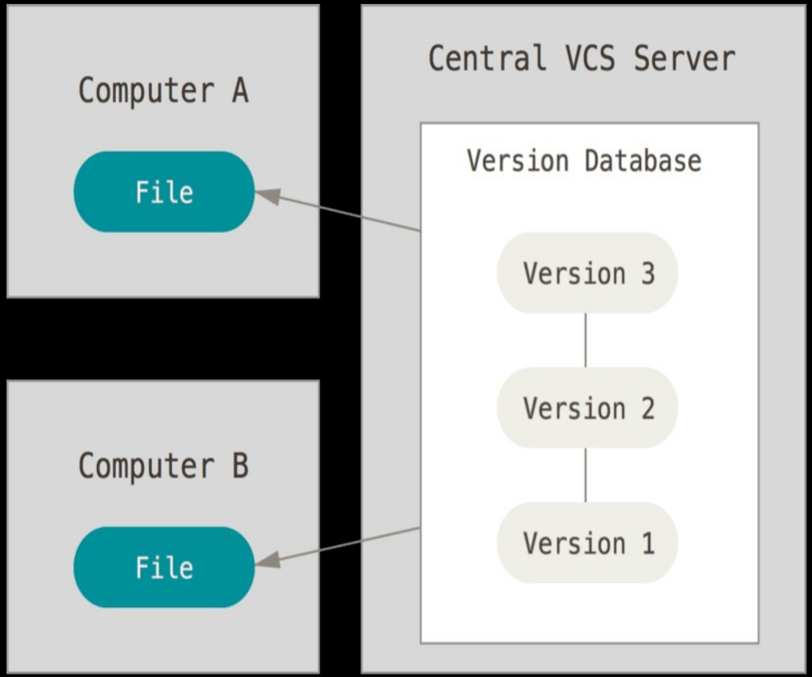
- Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone.
- To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.



softserve

Centralized Version Control Systems

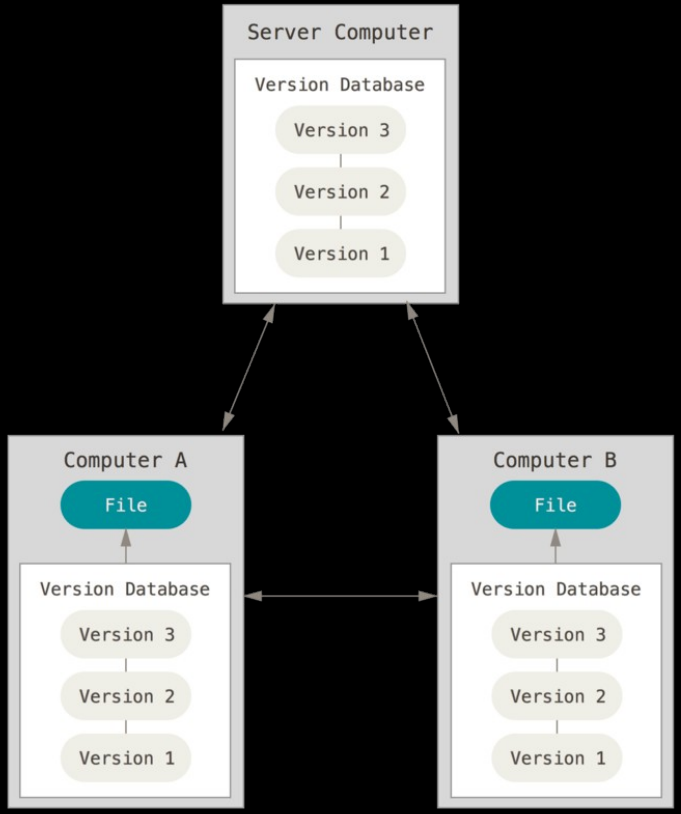
- The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on.



softserve

Distributed Version Control Systems

- In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.



softserve

Git

- Fast
- It has simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

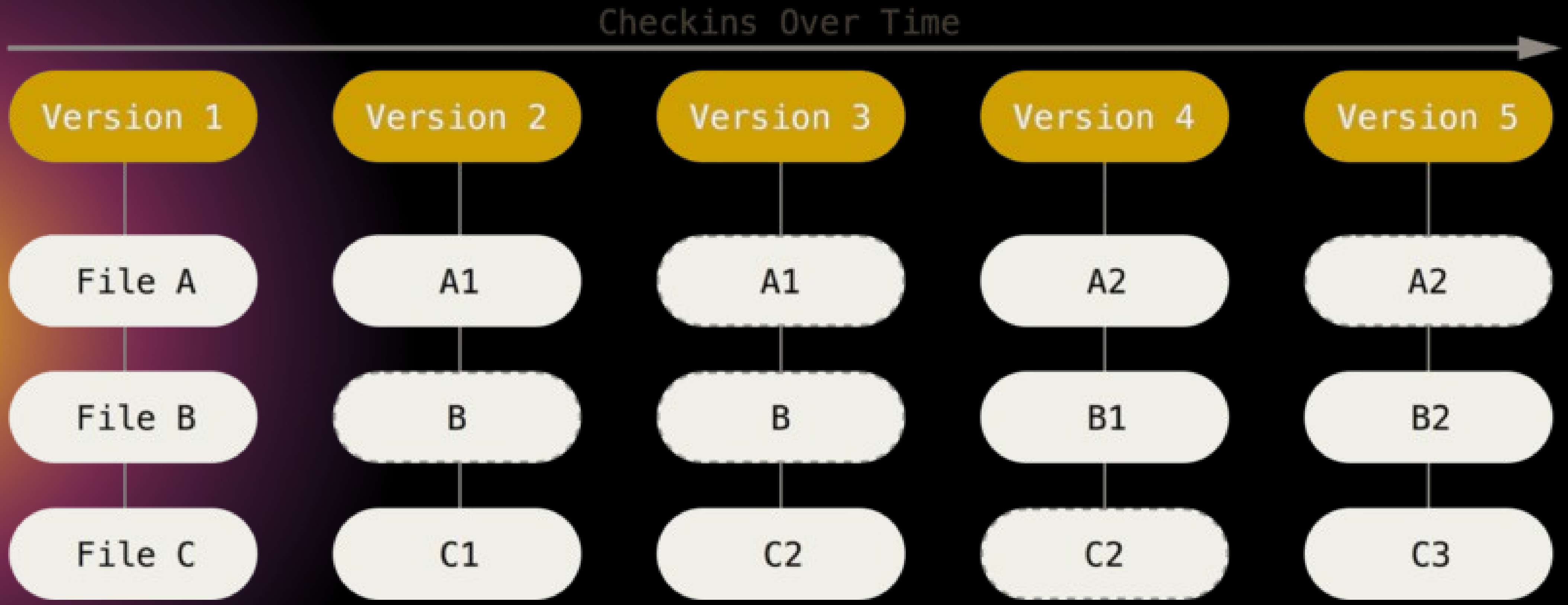
softserve

Git’s data handling

- Git thinks of its data more like a series of snapshots of a miniature filesystem. With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn’t store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a stream of snapshots.

softserve

Git's data handling



softserve

Git’s locality

- Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network.
- For example, to browse the history of the project, Git doesn’t need to go out to the server to get the history and display it for you — it simply reads it directly from your local database. This means you see the project history almost instantly.

Git’s Integrity

- Everything in Git is checksummed before it is stored and is then referred to by that checksum. This means it’s impossible to change the contents of any file or directory without Git knowing about it. This functionality is built into Git at the lowest levels and is integral to its philosophy. You can’t lose information in transit or get file corruption without Git being able to detect it.

softserve

Git Generally Only Adds Data

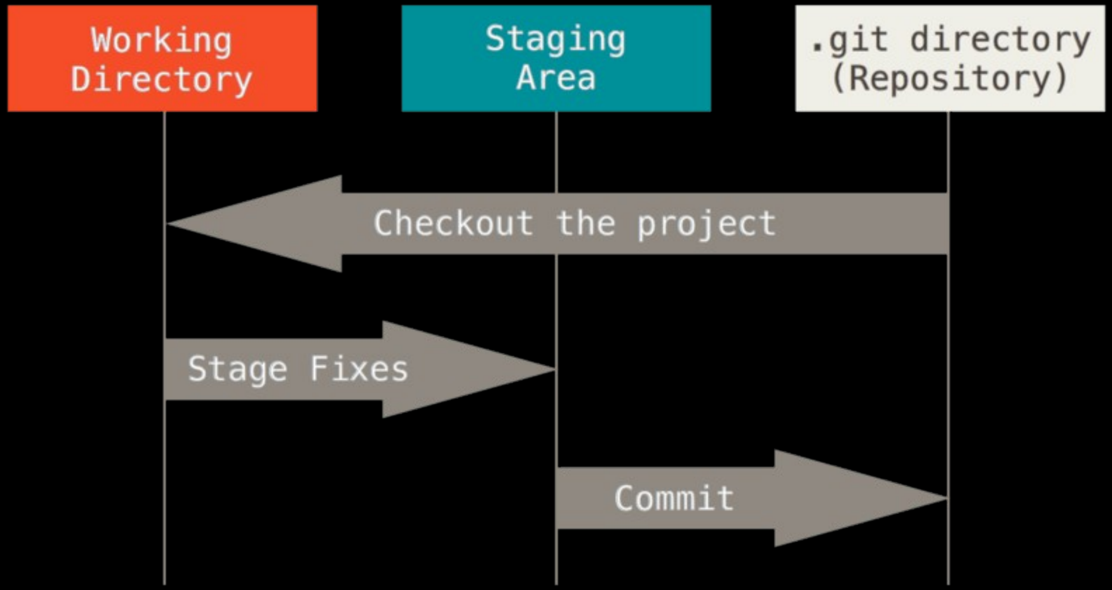
- When you do actions in Git, nearly all of them only add data to the Git database. It is hard to get the system to do anything that is not undoable or to make it erase data in any way. As with any VCS, you can lose or mess up changes you haven’t committed yet, but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

The Three States

- Git has three main states that your files can reside in: modified, staged, and committed:
- **Modified** means that you have changed the file but have not committed it to your database yet.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- **Committed** means that the data is safely stored in your local database.
- This leads us to the three main sections of a Git project: the working tree, the staging area, and the Git directory.

softserve

The Three States



softserve

Let’s pracice!

https://githowto.com/