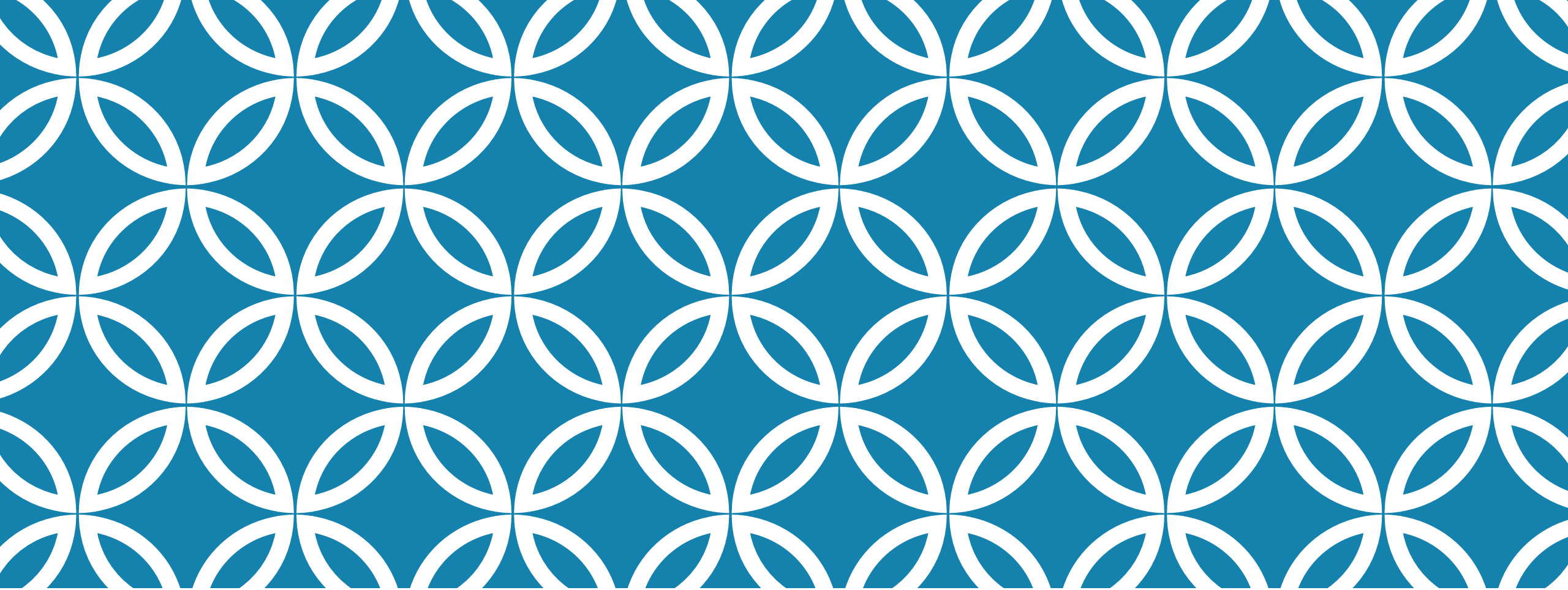


SCM AND GIT HUB INTRO

By Prof Ronak Sheth

SOURCE CODE MANAGEMENT

Source code management (SCM) is used to track modifications to a source code repository.

SCM tracks a running history of changes to a code base when merging updates from multiple contributors.

SCM is also synonymous with Version control.

IMPORTANCE OF SCM

When multiple developers are working within a shared codebase it is a common occurrence to make edits to a shared piece of code.

Separate developers may be working on a seemingly isolated feature, however this feature may use a shared code module.

Therefore developer 1 working on Feature 1 could make some edits and find out later that Developer 2 working on Feature 2 has conflicting edits.

Before the adoption of SCM this was a nightmare scenario.

SCM brought version control safeguards to prevent loss of work due to conflict overwriting. These safeguards work by tracking changes from each individual developer and identifying areas of conflict and preventing overwrites.

VERSION CONTROL(PART OF SCM)

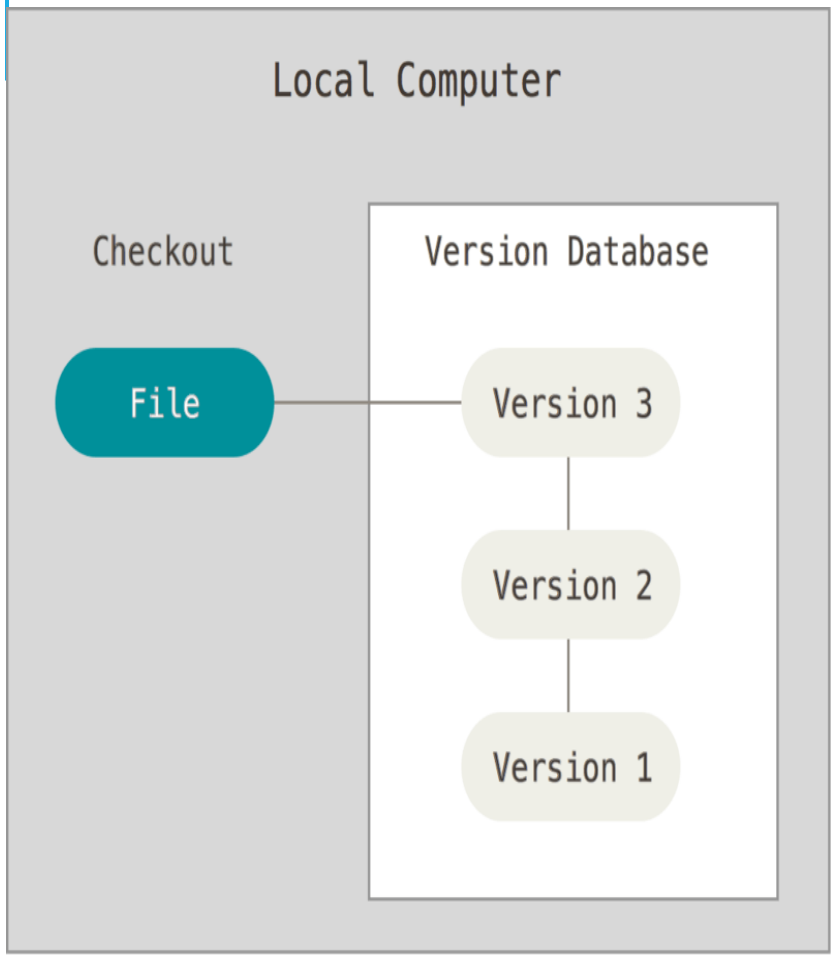
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later

1) Local Version Control

2) Centralized Version Control

3) Distributed Version Control

LOCAL VERSION CONTROL



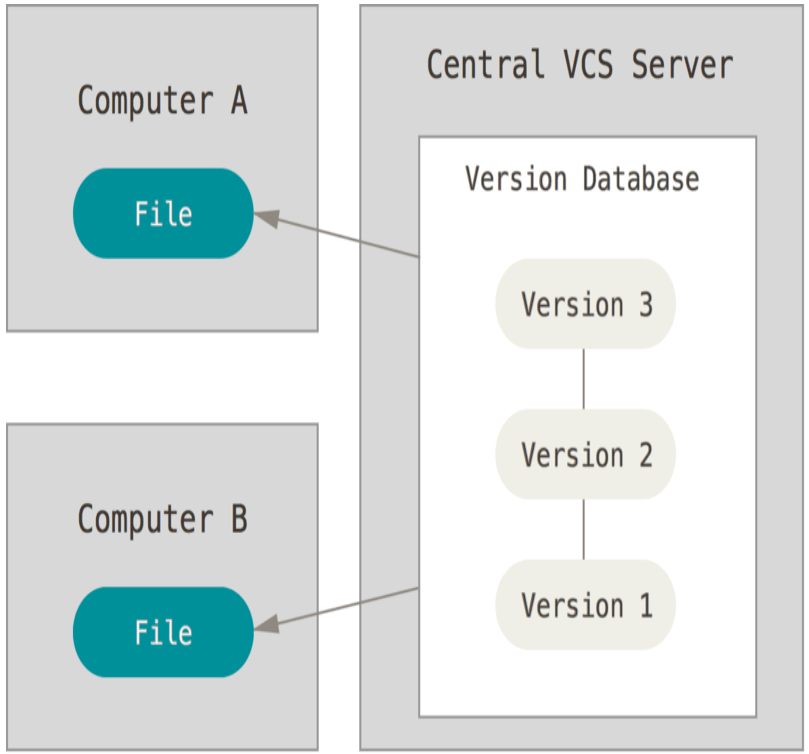
Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever).

This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

One of the most popular VCS tools was a system called RCS(Revision Control System), which is still distributed with many computers today.

RCS works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

CENTRALIZED VERSION CONTROL



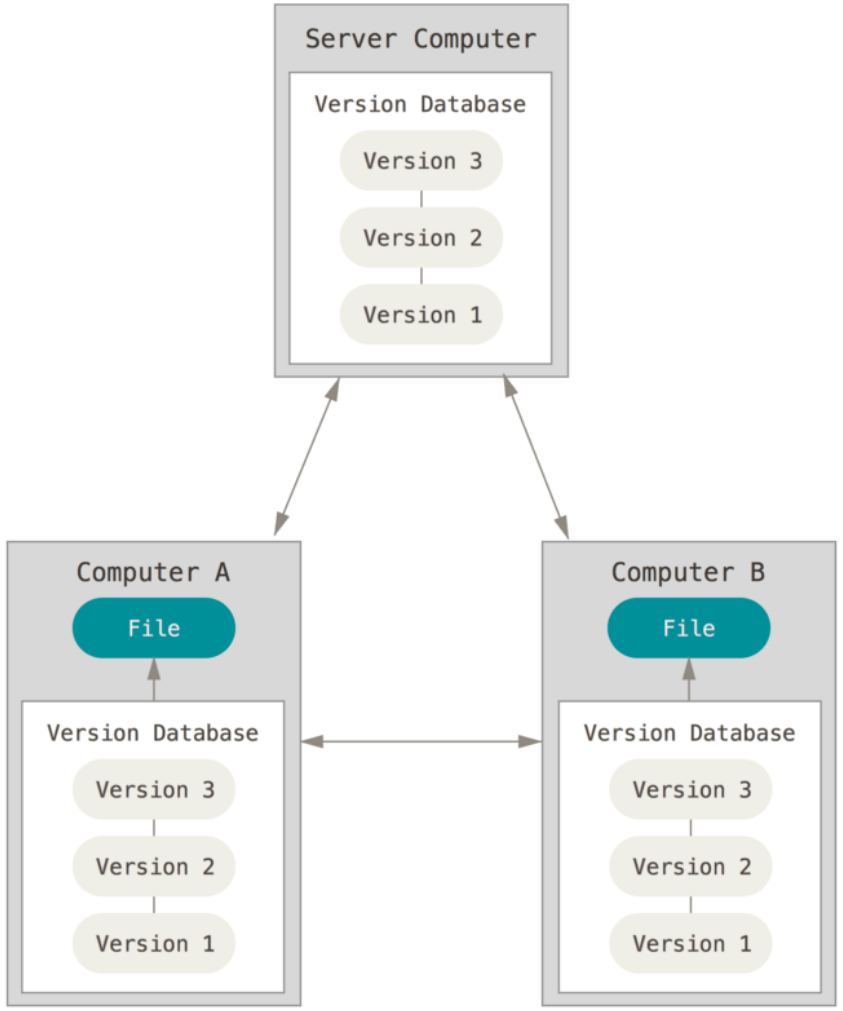
The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed.

These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place.

Advantage: everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what.

Disadvantage: single point of failure

DISTRIBUTED VERSION CONTROL



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.

Every clone is really a full backup of all the data.

GIT : EXAMPLE OF DISTRIBUTED VERSION CONTROL SYSTEM
HISTORY

During the early years of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files. In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper.

n 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked. This prompted the Linux development community (and in particular Linus Torvalds, the creator of Linux) to develop their own tool based on some of the lessons they learned while using BitKeeper. Some of the goals of the new system were as follows:

Speed ,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)

Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities. It's amazingly fast, it's very efficient with large projects, and it has an incredible branching system for non-linear development

GIT

1) **Snapshots, Not Differences**

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**

2) Nearly Every Operation Is Local

Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network

3) **Git Has 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.

GIT: BASICS

The Three States

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.

REFERENCES

- https://www.atlassian.com/git/tutorials/source-code-management
- https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control