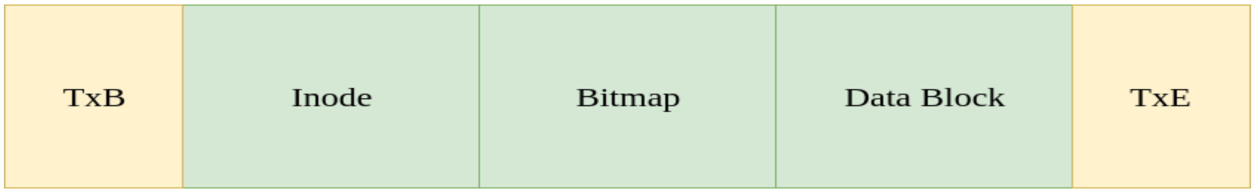
What is a File System?

A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device.

Implement a journaling file system for improved data integrity.

Journaling, or **write-ahead logging** is a sophisticated solution to the problem of file system inconsistency in operating systems. Inspired by database management systems, this method first writes down a summary of the actions to be performed into a "log" before actually writing them to the disk. Hence the name, "write-ahead logging". In the case of a crash, the OS can simply check this log and pick up from where it left off. This saves multiple disk scans to fix inconsistency, as is the case with FSCK. Good examples of systems that implement data journaling include Linux ext3 and ext4 file systems, and Windows NTFS. **Data Journaling:** A log is stored in a simple data structure called the journal. The figure below shows its structure, which comprises of three components.



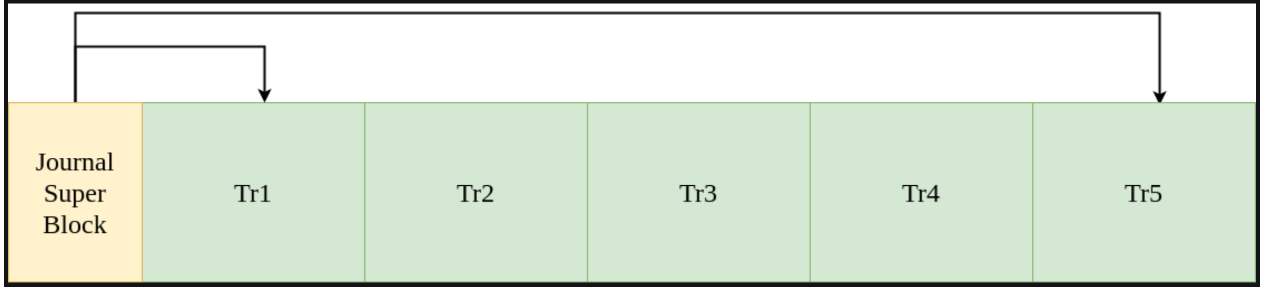
1. **TxB (Transaction Begin Block):** This contains the transaction ID, or the TID.
2. **Inode, Bitmap and Data Blocks (Metadata):** These three blocks contain a copy of the contents of the blocks to be updated in the disk.
3. **TxE (Transaction End Block)** This simply marks the end of the transaction identified by the TID.

As soon as an update is requested, it is written onto the log, and thereafter onto the file system. Once all these writes are successful, we can say that we have reached the **checkpoint** and the update is complete. **What if a crash**

occurs during journaling ? One could argue that journaling, itself, is not atomic. Therefore, how does the system handle an un-checkpointed write ? To overcome this scenario, journaling happens in two steps: simultaneous writes to TxB and the following three blocks, and then write of the TxE. The process can be summarized as follows.

1. **Journal Write:** Write TxB, inode, bitmap and data block contents to the journal (log).
2. **Journal Commit:** Write TxE to the journal (log).
3. **Checkpoint:** Write the contents of the inode, bitmap and data block onto the disk.

A crash may occur at different points during the process of journaling. If a crash occurs at step 1, i.e. before the TxE, we can simply skip this transaction altogether and the file system stays consistent. If a crash occurs at step 2, it means that although the transaction has been logged, it hasn't been written onto the disk completely. We cannot be sure which of the three blocks (inode, bitmap and data block) were actually updated and which ones suffered a crash. In this case, the system scans the log for recent transactions, and performs the last transaction again. This does lead to redundant disk writes, but ensures consistency. This process is called **redo logging**. **Using the Journal as a Circular Buffer:** Since many transactions are made, the journal log might get used up. To address this issue, we can use the journal log as a circular buffer wherein newer transactions keep replacing the old ones in a circular manner. The figure below shows an overall view of the journal, with tr1 as the oldest transaction and tr5 the newest.



The benefits of journaling, or write-ahead logging, in file systems, are as follows:

- **Improved Recovery Time:** It ensures quick recovery after a crash occurs, all actions are logged prior to being written on disk on file for examination. This eliminates the need to perform lengthy disk scans or consistency checks.
- **Enhanced Data Integrity:** It ensures data integrity by maintaining the consistency of the file system. By writing the actions to the journal before committing them to the disk, the system can ensure that updates are complete and recoverable. In case of a crash, the system can recover by referring to the journal and redoing any incomplete transactions.
- **Reduced Disk Scans:** It minimizes the need for full disk scans to fix file system inconsistencies. Instead of scanning the entire disk to identify and repair inconsistencies, the system can rely on the journal to determine the state of the file system and apply the necessary changes. This leads to faster recovery and reduced overhead.

2.Create a file version control system for tracking changes in files.

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

Why Version Control system is so Important?
As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes to some specific kind of functionality/features. So in order to contribute to the product, they made modifications to the source code(either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

Basically Version control system keeps track on changes made on a particular software and take a snapshot of every modification. Let's suppose if a team of developer add some new functionalities in an application and the updated version is not working properly so as the version control system keeps track of our work so with the help of version control system we can omit the new changes and continue with the previous version.

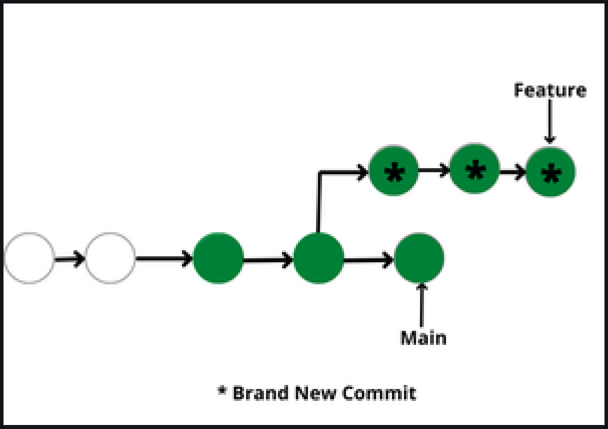
Benefits of the version control system:

- Enhances the project development speed by providing efficient collaboration,
- Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,
- Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
- Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this **VCS,**
- For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is **Git, Helix core, Microsoft TFS,**
- Helps in recovery in case of any disaster or contingent situation,
- Informs us about Who, What, When, Why changes have been made.

Use of Version Control System:

- **A repository:** It can be thought of as a database of changes. It contains all the edits and historical versions (snapshots) of the project.
- **Copy of Work (sometimes called as checkout):** It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.
- **Working in a group:** Consider yourself working in a company where you are asked to work on some live project. You can't change the main code as it is in production, and any change may cause inconvenience to the user, also you are working in a team so you need to collaborate with your team to and adapt their changes. Version

control helps you with the, merging different requests to main repository without making any undesirable changes. You may test the functionalities without putting it live, and you don't need to download and set up each time, just pull the changes and do the changes, test it and merge it back. It may be visualized as.



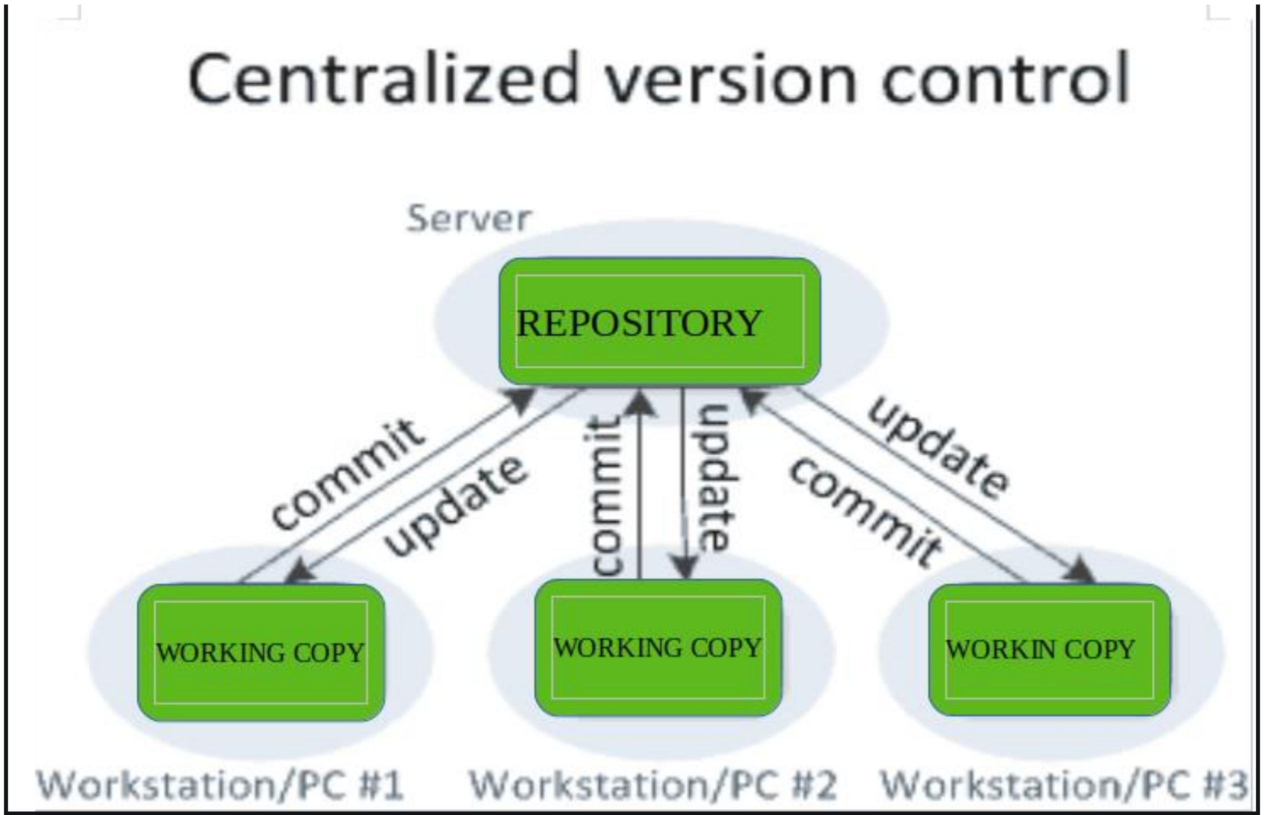
Types of Version Control Systems:

- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems

Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized Version Control Systems: Centralized version control systems contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.
Two things are required to make your changes visible to others which are:

- You commit
- They update



The **benefit** of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.

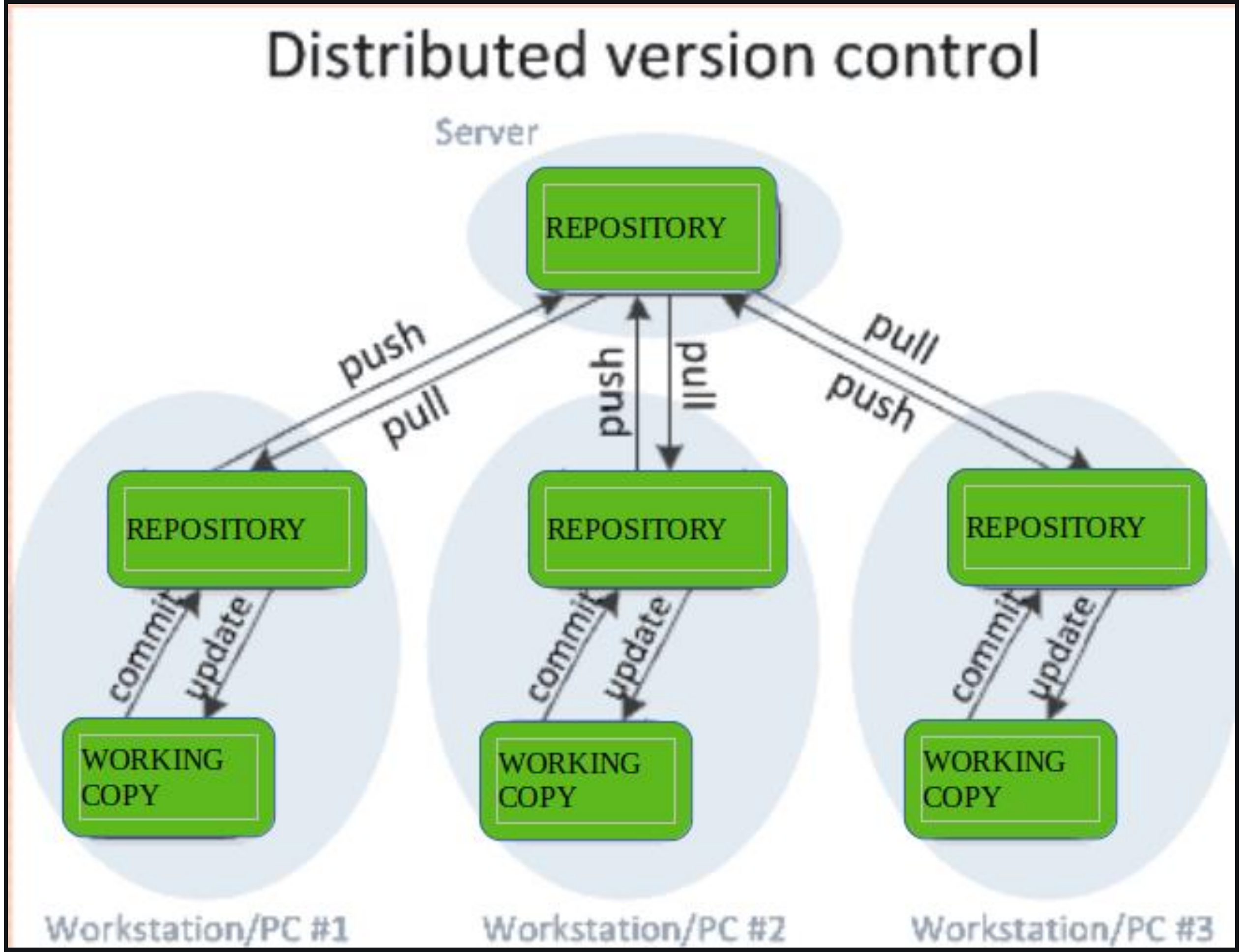
It has some **downsides** as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything.

Distributed Version Control Systems: Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the

central repository. Similarly, When you update, you do not get others' changes unless you have first pulled those changes into your repository.
To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull
- They update

The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.



Purpose of Version Control:

- Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.

- It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
- It integrates the work that is done simultaneously by different members of the team. In some rare cases, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.
- Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

Understanding File System Encryption

Before we delve into the specifics, let's start by understanding what file system encryption entails. At its core, file system encryption is the process of encoding data on a storage device in such a way that it cannot be accessed without the correct decryption key. This encryption protects files by converting them into an unreadable format, rendering them incomprehensible to anyone without the authorized access credentials.

WHAT IS FILE SYSTEM ENCRYPTION?

File system encryption involves the conversion of data into a coded form, making it inaccessible to unauthorized individuals. It provides an added layer of security by ensuring that even if someone gains physical or remote access to your storage devices, they would not be able to read or understand the contents of your files.

WHY IS FILE SYSTEM ENCRYPTION IMPORTANT FOR PROTECTING SENSITIVE DATA?

The importance of file system encryption in protecting sensitive data cannot be overstated. Consider the case of a stolen laptop or a compromised cloud storage account. Without encryption, the thief or attacker would have unrestricted access to your documents, emails, or other personal information. By encrypting your files, you significantly reduce the risk of unauthorized access, thereby safeguarding your data against potential breaches.