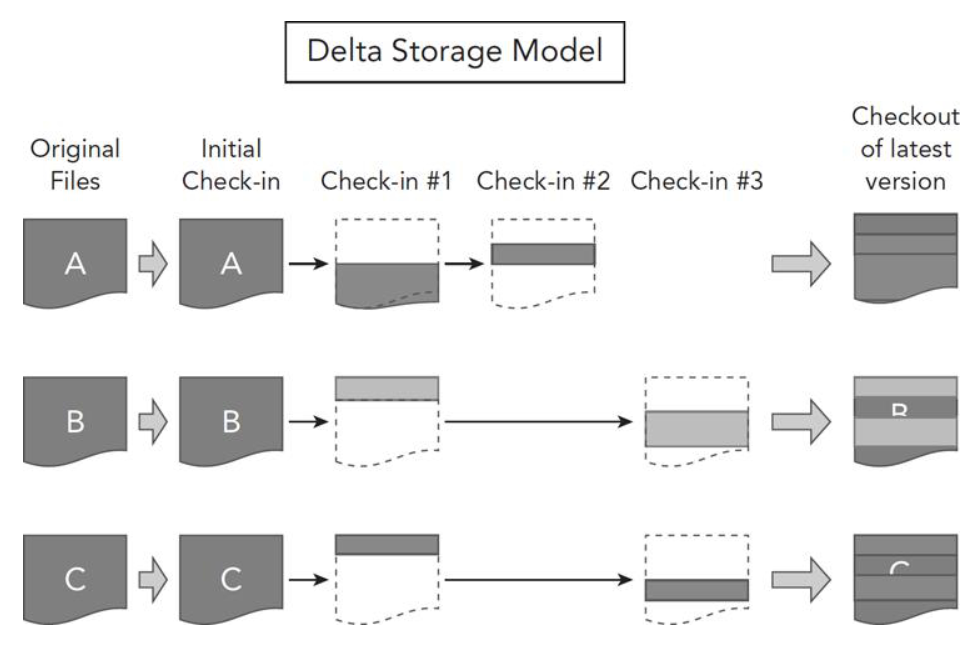
Git differs significantly from traditional source management systems is in the way it represents and stores **changes internally**.

**Delta Storage:**

In a traditional source management system, content is managed on a file-by-file basis. That is, each file is managed as an independent entity in the repository. When a set of files is added to a repository for the first time, each file is stored as a separate object in the repository, with its complete contents. The next time any changes to any of these files are checked in, the system computes the differences between the new version and the previous version for each file. It constructs a delta, or patch set, for each file from the differences. It then stores that delta as the file's next revision. This model is called delta storage.



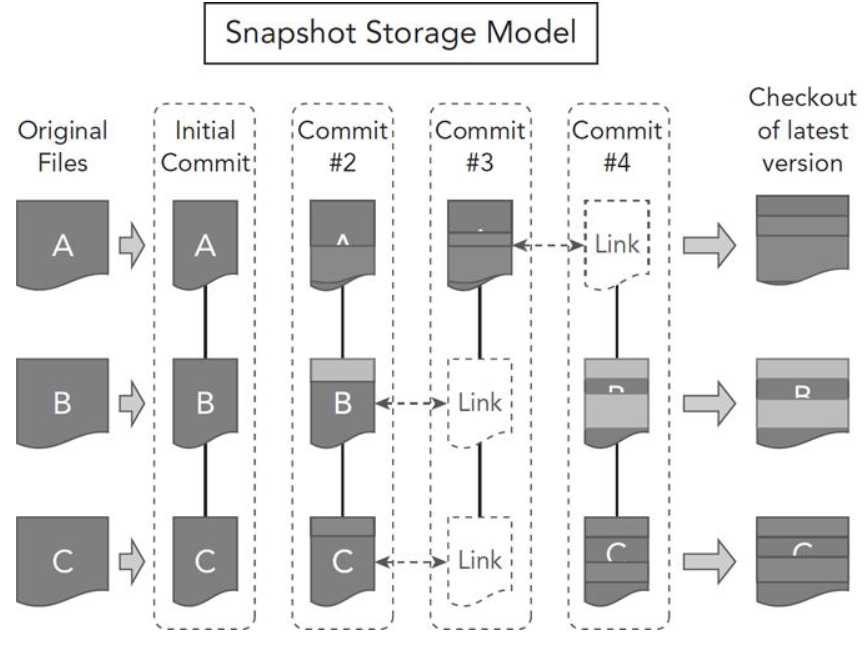
In the first iteration, files A, B, and C are checked in. Then, changes are made to the three files and those changes are checked in. When that occurs, the system computes the deltas between the current and previous versions. It then constructs the patch set that will allow it to re-create the current version from the previous version (the set of lines added, deleted, changed, and so on). That patch set is stored as the next revision in the sequence. The process repeats as more changes are made. Each delta is dependent on the previous one in order to construct that version of the file. In order to get the most current version of a file from the system when the client requests it, the system starts with the original version of the file and then applies each delta in turn to arrive at the desired version. As the files continue to be updated over time, more and more deltas are created. In turn, more deltas must be applied in sequence to deliver a requested version. Eventually, this can lead to performance degradation, among other issues.

**Snapshot Storage:**

Git uses a different storage model, called snapshot storage. Whereas in the delta model, revisions are tracked on a file-by-file basis, Git tracks revisions at the level of a directory tree. You can think of each revision within a Git repository as being a slice of a directory tree structure at a point in time—a snapshot. The structure that Git bases this on is the directory structure in your workspace .

When a commit is made into a Git repository, it represents a snapshot of part or all of the directory tree in the workspace, at that point in time. When the next commit is made, another snapshot is taken of the workspace, and so on. In each of these snapshots, Git is capturing the contents of all of the involved files and directories as they are in your workspace at that point in time. It's recording the full content, not computing deltas. There is no work to compute differences at that point.

The snapshot storage model is shown in [below](https://viewer.books24x7.com/assetviewer.aspx?bookid=117480&chunkid=792622719&resumebookmarkid=dd987e14-991b-ea11-bf06-0050569533bd) diagram. In this model, you have the same set of three files, A, B, and C. At the point they are initially put into the repository, a snapshot of their state in the workspace is taken and that snapshot (with each of the file's full contents) is stored in Git and referenced as a unit.



To summarize, there are two differences between delta and snapshot storage:

1. Delta storage manages content on a file-by-file basis, as opposed to snapshot storage where content is managed at a directory tree level.
2. Delta storage manages versions over time by figuring out the differences and storing that information from revision to revision (the delta). It reconstructs later revisions by starting with the base version and applying deltas on top of that. Because snapshot storage is storing a capture of the entire tree, it does not usually have to do any reconstruction, or only a very small amount if the content has been packed.