**Day 6 Assignment 1: Architecture of GIT**

1. Working Directory: This is where you create, modify, and organize your files. It’s essential

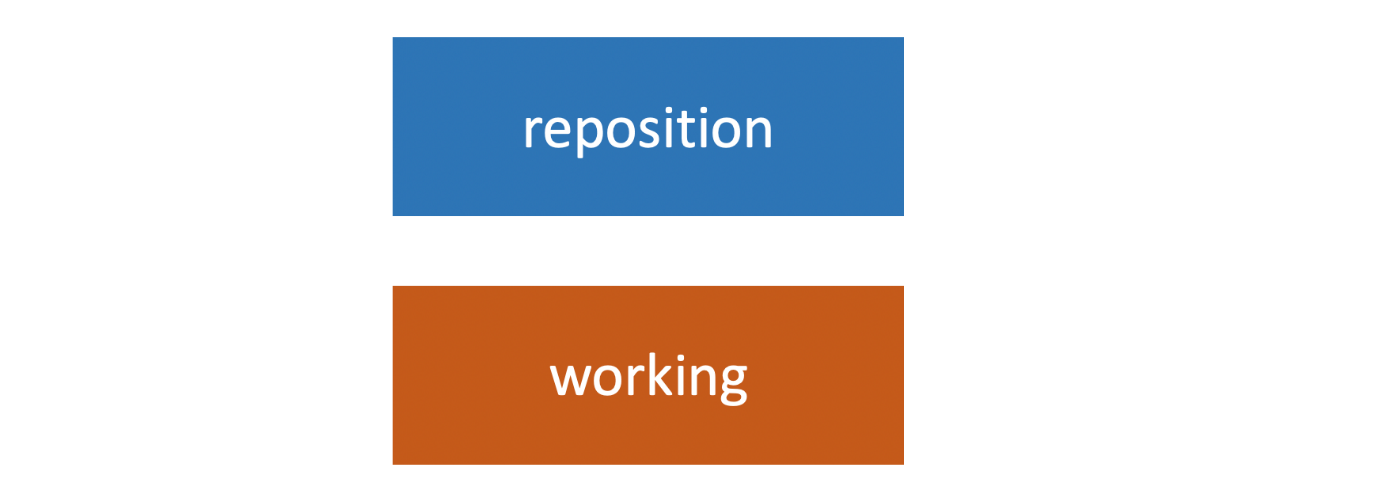
2. Staging Area (Index): The staging area acts as an intermediate step between your working directory and the local repository. You use it to select which changes you want to commit. When you add files or changes to the staging area, Git prepares them for the next commit.

3. Git Repository: The local repository stores all the committed changes. It includes the entire history of your project. When you commit changes from the staging area, they become part of the local repository

Three-Tree Architecture

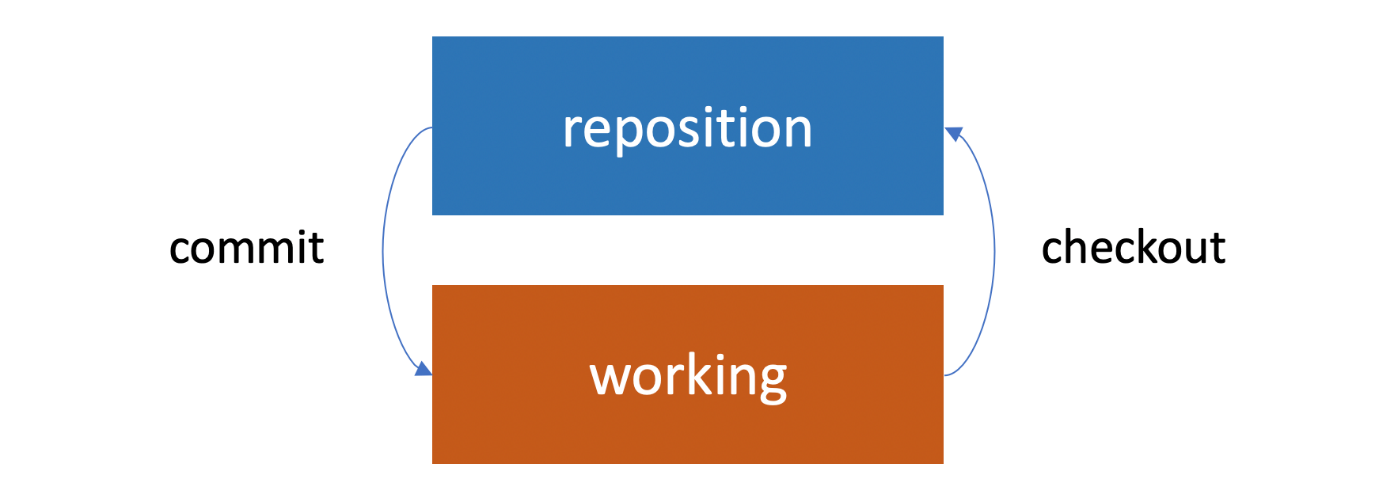
1. Two-tree architecture

Let’s start by taking a look at a typical two-tree architecture. This is what a lot of other source version control systems use. They have a *repository* and a *working copy*. These are the two trees.



We call them trees because they represent a file/folder structure. The main project directory is at the top, and below it might be 4 or 5 different folders with a few files inside. Maybe a few more folders, each of those folders has a few more folders inside. You can imagine that if you map that out, each of those folders would branch out like the branches of a tree.

The repository also has a set of files/folders in it, also arranged as a tree. When we want to move files between the repository and the working copy, we *checkout* copies, that’s the term we use. We check it out from our repo into the working directory, and when we finish making our changes to the files, we *commit* those changes back to the repo. We have two distinct trees because the files may not be the same.

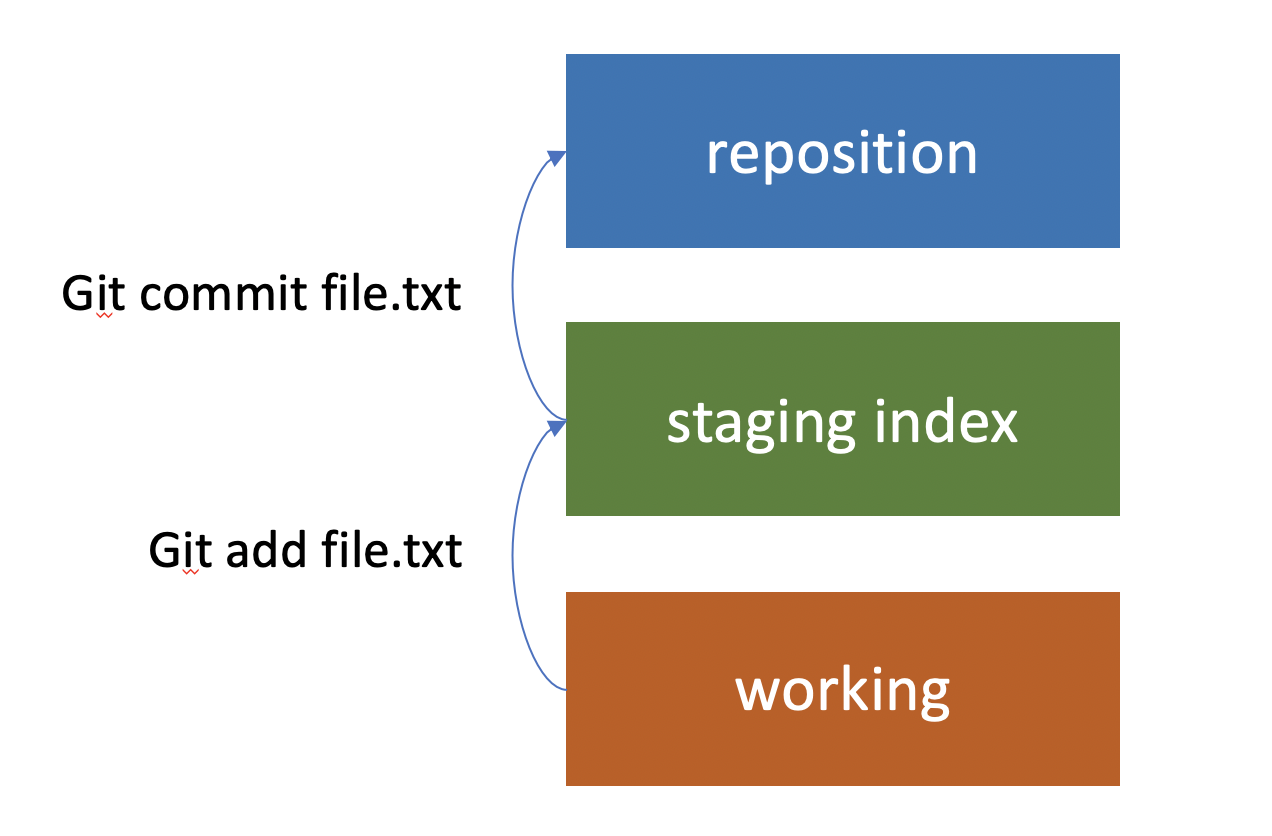


.

We can imagine another case. If the repository is a shared repository and many people are working from it, they may add their own changes to the repository. If we haven’t checked out a copy recently, then our working copy won’t have its changes. Now that’s a typical two-tree architecture.

2. Git uses a three-tree architecture

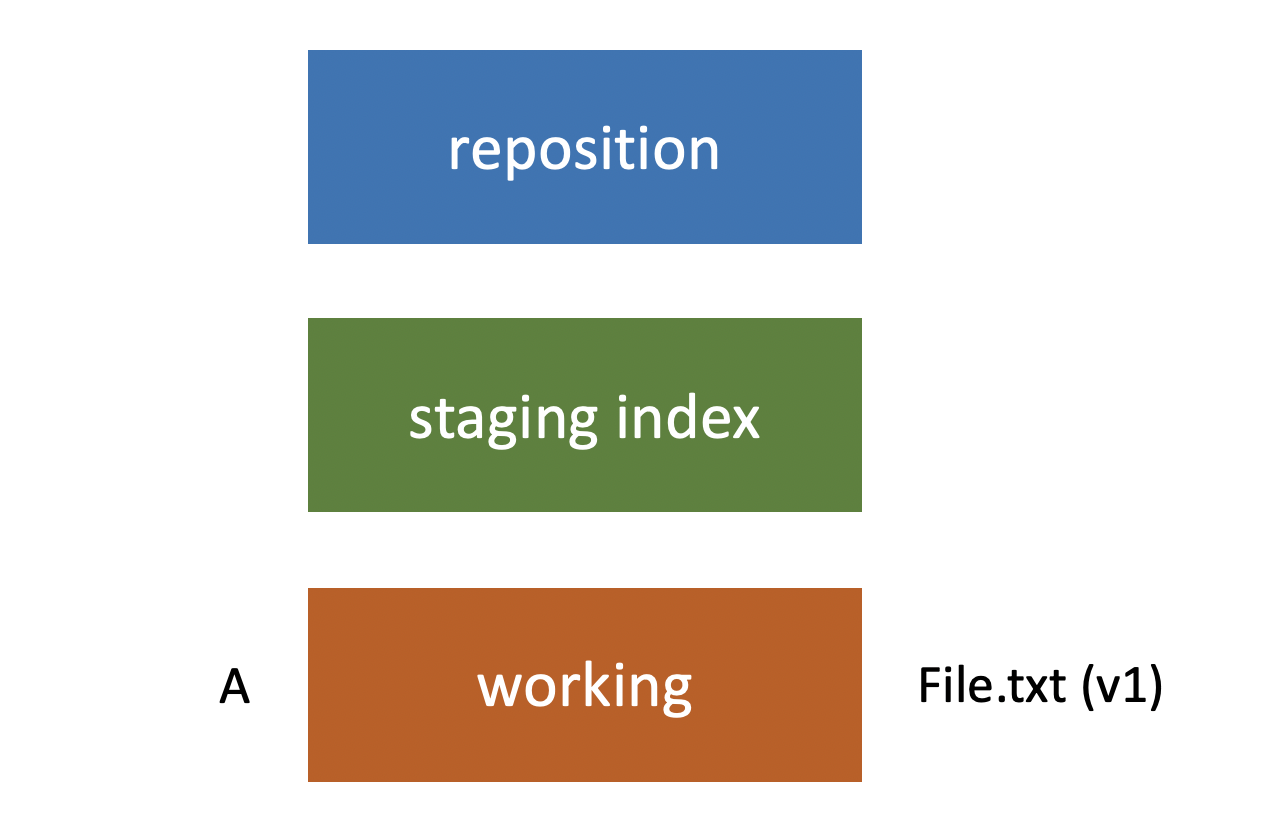
It still has the repository and the working copy, but in between is another tree, which is the *staging* index. When we made our first commit, we didn’t just perform a commit. First, we used the *add* command. We added, then we committed. It was a two-step process (added our files to the staging index, and then from there we committed to the repository).



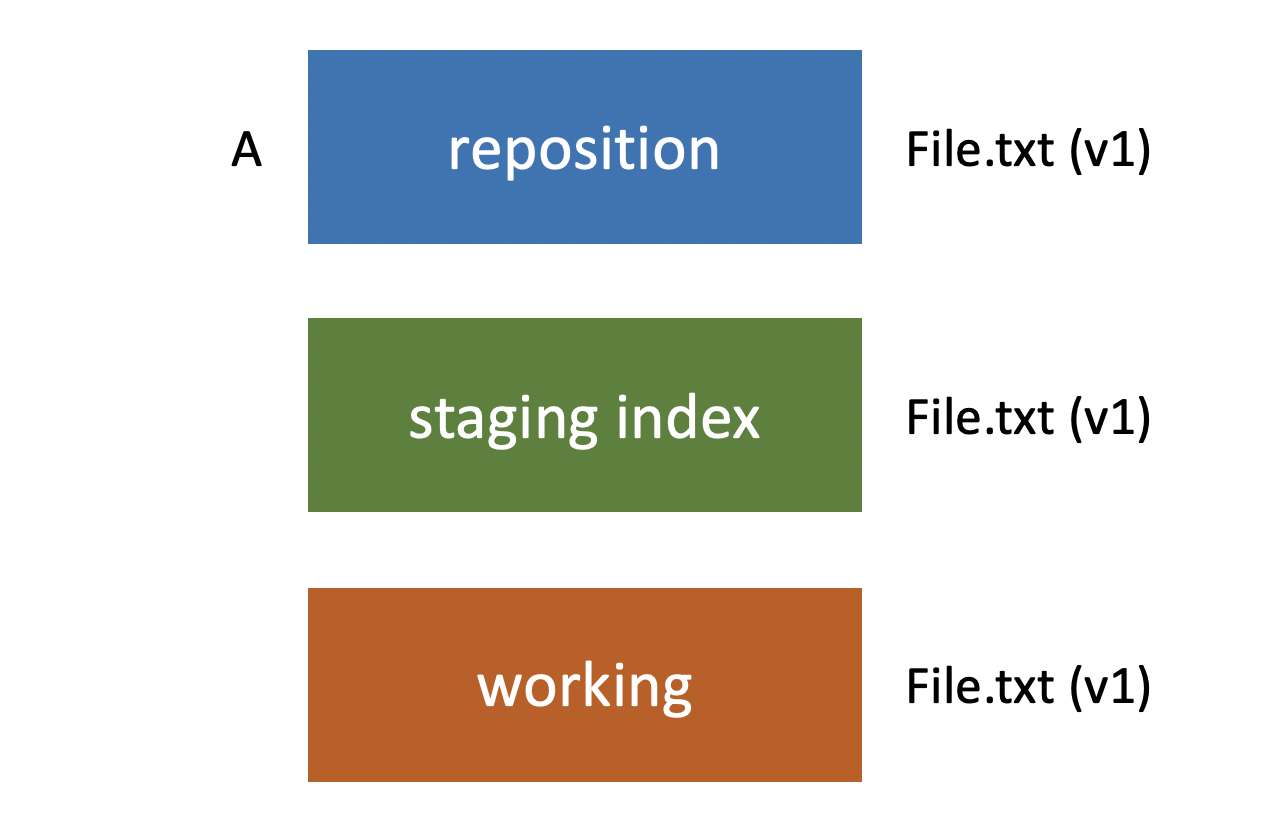
It is possible to just commit directly to the repository and skip adding them to the staging index. But it’s important to understand that this is part of the architecture of Git. It’s a really nice feature because we can make changes to ten different files in our working copy, and then selectively commit five of the changes as one set. That’s why it’s called staging. We have the chance to add the changes that we want to the staging, and then get them ready before we commit them. And we can check out changes from the repository the same way.

It’s also possible to pull them from the repository into the staging index, and then from the staging index to the working directory, but most of the time, that’s not what we do. Usually, we go ahead and pull them straight from the repository, down into the working directory. In the process, the staging index will also be updated too.

As we’re working with Git, it’s useful to keep these three different trees in mind. There’s our working directory, which contains changes that may not be tracked by Git yet, there’s the staging index, which contains changes that we’re about to commit into the repository, and then there’s the repository, and that’s what’s actually being tracked by Git.

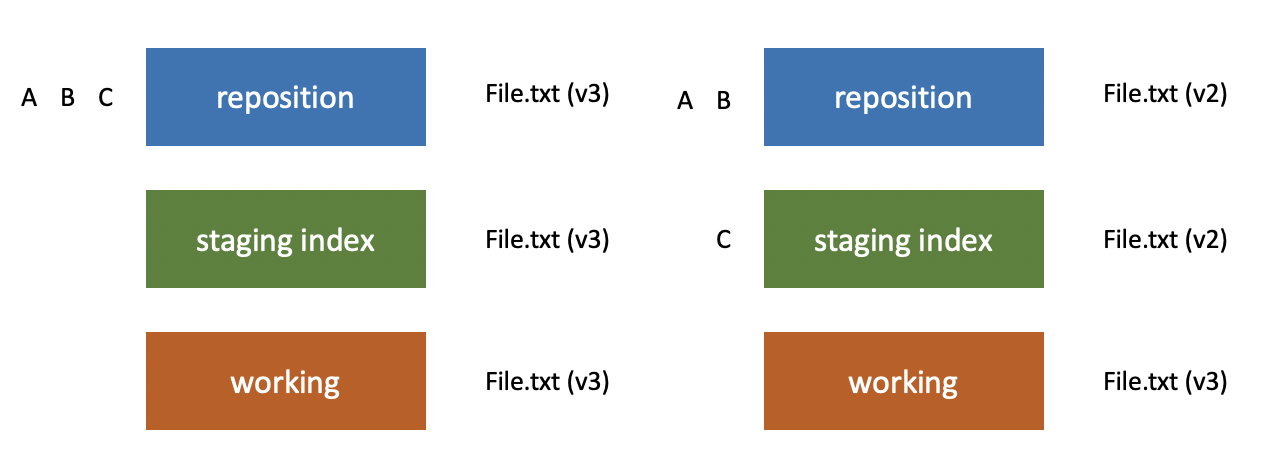


Let’s refer to this set of changes that we’ve made as A. So it’s in the working directory right now, not in the staging index or the repository. After that, we use git commit in order to push that change set into the repository. Now the repository has the same file and it’s the same version as what’s in our staging index and our working directory.



When we’re ready to add it to our staging index we use git add file.txt again, it adds it to our staging index, and then we’re ready to add it to the repository we use git commit, and it’s added to our repository. Now our repository has two sets of changes in it. Set A, and set B.

And then the same thing would be true if we made the third set of changes called c. We would use git add to push those to the staging index and then git commit to push them into the repository. This is the typical workflow you’re going to be using to make commits. And then you could use*git log* in order to view those commits (A, B & C) and to see what changed between each one.



Now, of course, Git doesn’t refer to them as changes a, b and c. It has a unique ID number that applies to each one. In the next post, let’s talk more about that ID.

**Conclusion**

Three-tree architecture in Git: The working directory (containing changes that may not be tracked by Git), the staging index (containing changes that are about to be committed into the repository), and the repository (being tracked by Git).

When we finish working with our files in the project, commit them to the staging stage and then push them to the repository.