--everything-is-local

Search entire site...

- About
- Documentation
  - Reference
  - Book
  - Videos
  - External Links
- Blog
- Downloads
  - GUI Clients
  - Logos
- Community

Download this book in PDF, mobi, or ePub form for free.

This book is translated into Deutsch, 简体中文, 正體中文, Français, 日本語, Nederlands, Русский, 한국어, Português (Brasil) and Čeština.

Partial translations available in Arabic, Español, Indonesian, Italiano, Suomi, Македонски, Polski and Türkçe.

Translations started for Azərbaycan dili, Беларуская, Català, Esperanto, Español (Nicaragua), हिन्दी, Magyar, Norwegian Bokmål, Română, Српски, ภาษาไทย, Tiếng Việt and Українська.

The source of this book and the translations are hosted on GitHub.
Patches, suggestions, and comments are welcome.

Related StackOverflow Questions

- About Git's mer
- git rebase vs git
- How to do a re
- recovering from
- git pull VS git fe
- Undoing a git re
- How to know if

UP NEXT: T-Pain: Five Minutes to the Stage - Myspace

Chapters ▾

1.3 Getting Started - Git Basics

Git Basics

So, what is Git in a nutshell? This is an important section to absorb, because if you understand what Git is and the fundamentals of how it works, then using Git effectively will probably be much easier for you. As you learn Git, try to clear your mind of the things you may know about other VCSs, such as Subversion and Perforce; doing so will help you avoid subtle confusion when using the tool. Git stores and thinks about information much differently than these other systems, even though the user interface is fairly similar; understanding those differences will help prevent you from becoming confused while using it.

Snapshots, Not Differences

The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data. Conceptually, most other systems store information as a list of file-based changes. These systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they keep as a set of files and the changes made to each file over time, as illustrated in Figure 1-4.

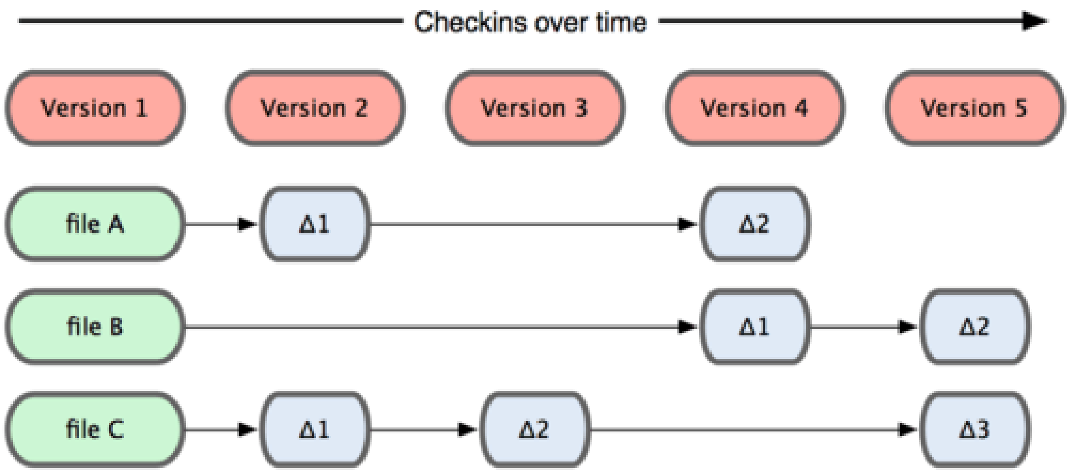

Figure 1-4. Other systems tend to store data as changes to a base version of each file.

Git doesn’t think of or store its data this way. Instead, Git thinks of its data more like a set of snapshots of a mini filesystem. Every time you commit, or save the state of your project in Git, it 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 Figure 1-5.

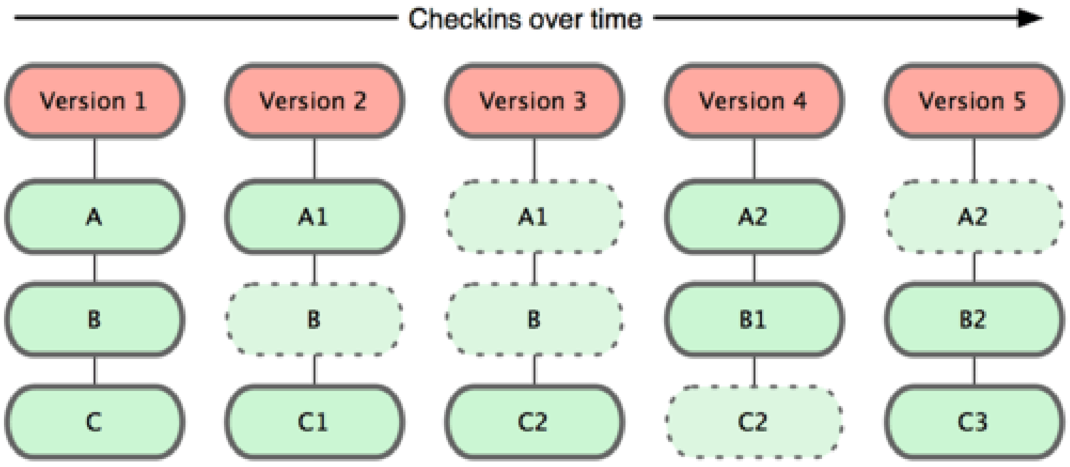

Figure 1-5. Git stores data as snapshots of the project over time.

This is an important distinction between Git and nearly all other VCSs. It makes Git reconsider almost every aspect of version control that most other systems copied from the previous generation. This makes Git more like a mini filesystem with some incredibly powerful tools built on top of it, rather than simply a VCS. We'll explore some of the benefits you gain by thinking of your data this way when we cover Git branching in Chapter 3.

Nearly Every Operation Is Local

Most operations in Git only need local files and resources to operate — generally no information is needed from another computer on your network. If you're used to a CVCS where most operations have that network latency overhead, this aspect of Git will make you think that the gods of speed have blessed Git with unworldly powers. Because you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.

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. If you want to see the changes introduced between the current version of a file and the file a month ago, Git can look up the file a month ago and do a local difference calculation, instead of having to either ask a remote server to do it or pull an older version of the file from the remote server to do it locally.

This also means that there is very little you can't do if you're offline or off VPN. If you get on an airplane or a train and want to do a little work, you can commit happily until you get to a network connection to upload. If you go home and can't get your VPN client working properly, you can still work. In many other systems, doing so is either impossible or painful. In Perforce, for example, you can't do much when you aren't connected to the server; and in Subversion and CVS, you can edit files, but you can't commit changes to your database (because your database is offline). This may not seem like a huge deal, but you may be surprised what a big difference it can make.

Git Has Integrity

Everything in Git is check-summed 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.

The mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git. A SHA-1 hash looks something like this:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

You will see these hash values all over the place in Git because it uses them so much. In fact, Git stores everything not by file name but in the Git database addressable by the hash value of its contents.

Git Generally Only Adds Data

When you do actions in Git, nearly all of them only add data to the Git database. It is very difficult to get the system to do anything that is not undoable or to make it erase data in any way. As in 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.

This makes using Git a joy because we know we can experiment without the danger of severely screwing things up. For a more in-depth look at how Git stores its data and how you can recover data that seems lost, see Chapter 9.

The Three States

Now, pay attention. This is the main thing to remember about Git if you want the rest of your learning process to go smoothly. Git has three main states that your files can reside in: committed, modified, and staged. Committed means that the data is safely stored in your local database. 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.

This leads us to the three main sections of a Git project: the Git directory, the working directory, and the staging area.

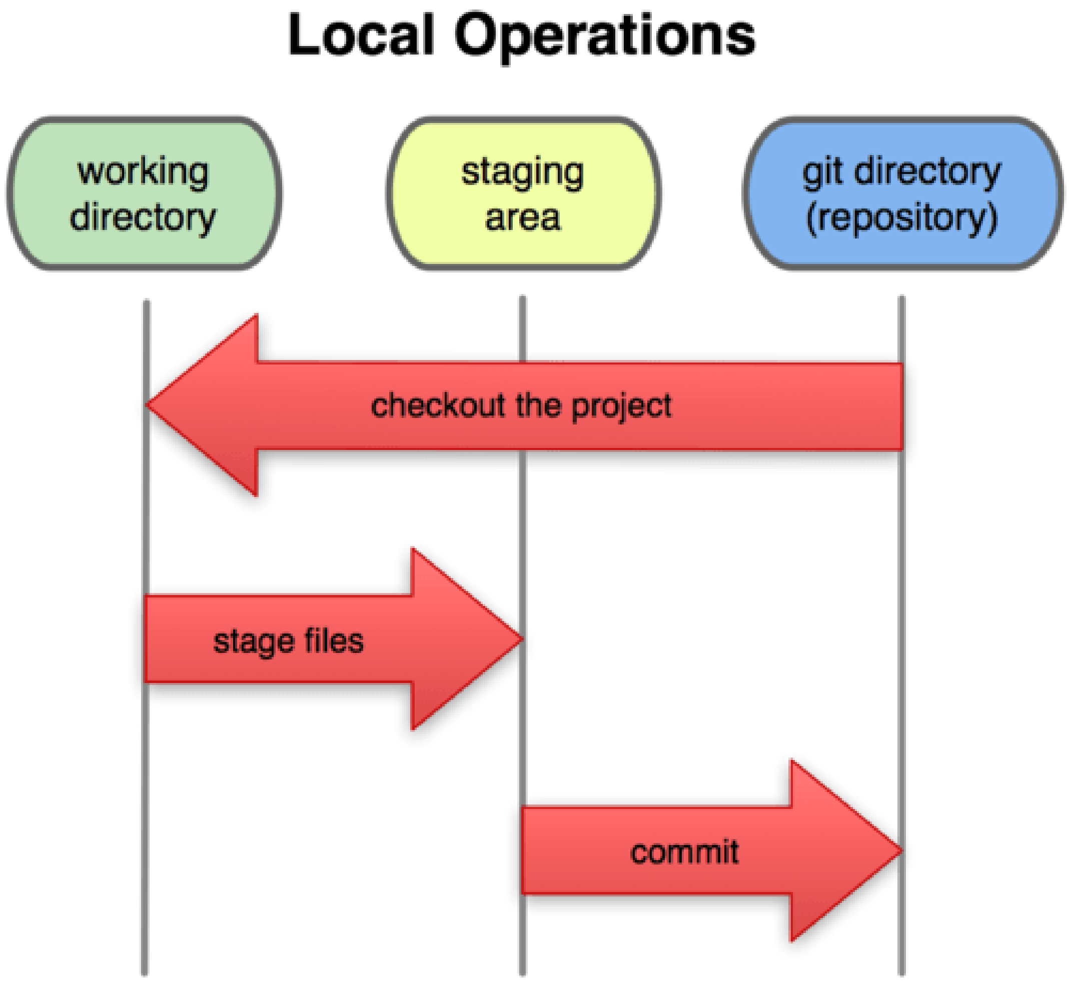

Figure 1-6. Working directory, staging area, and Git directory.

The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

The working directory is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.

The staging area is a simple file, generally contained in your Git directory, that stores information about what will go into your next commit. It's sometimes referred to as the index, but it's becoming standard to refer to it as the staging area.

The basic Git workflow goes something like this:

1. You modify files in your working directory.
2. You stage the files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

If a particular version of a file is in the Git directory, it's considered committed. If it's modified but has been added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified. In Chapter 2, you'll learn more about these states and how you can either take advantage of them or skip the staged part entirely.

prev | next