GIT

Architecture & Deployment

WHAT IS GIT?

Git is a **version control system (VCS)** originally developed by Linus Torvalds to work on Linux.

Speaker notes

Linus Torvals is the creator of Linux. He created Git to manage the source code of the Linux kernel, because the alternatives were proprietary and did not support the kind of workflow he wanted to develop Linux with.

DESIGN GOALS

- **Speed**
- **Simple** design
- Strong support for **non-linear development**
- Fully **distributed**
- Able to handle **large projects**

Speaker notes

The Linux is a large and complex piece of code, and it has thousands of contributors. A version control system was needed that could handle thousands of parallel branches and handle a project of that size with speed and efficiency (data size).

WHAT IS A VERSION CONTROL SYSTEM?

> *A system that **records changes** to a file or set of files over time so that you can **recall specific versions later**.*

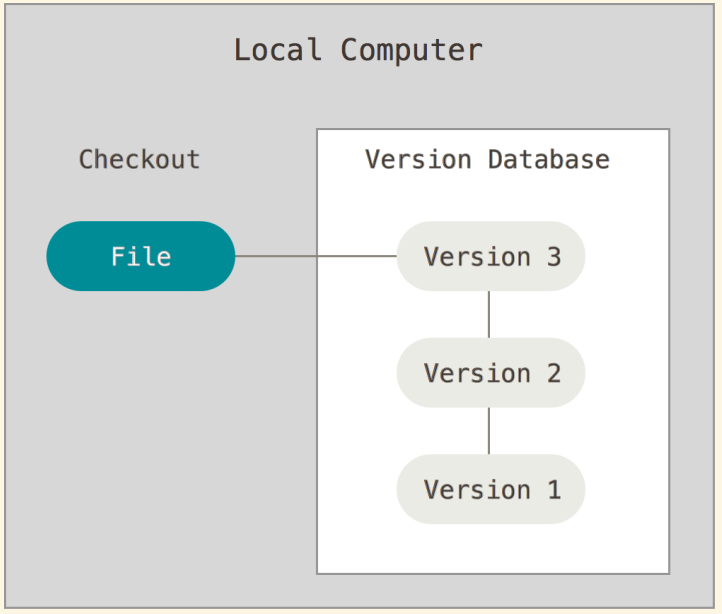
WHAT CAN I DO WITH IT?

- **Revert** files back to a previous state
- **Compare** changes over time
- See **who last modified something**
- **Recover** if you screw things up
- **Collaborate** as a distributed team

A SHORT HISTORY

LOCAL VERSION CONTROL SYSTEMS (1980S)

- Easy to **accidentally edit the wrong files**
- **Hard to collaborate** on different versions with other people



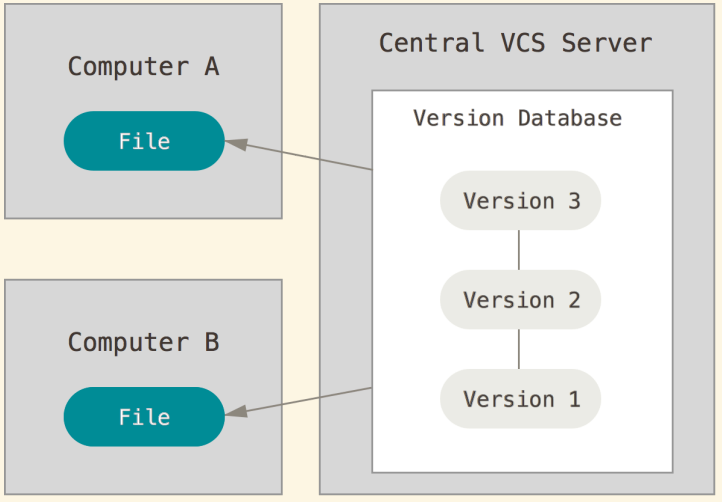
Speaker notes

Basically, you **manually** copy your files into other directories to keep old versions.

Revision **C**ontrol **S**ystem (RCS), first released in 1982, automates this process.

CENTRALIZED VERSION CONTROL SYSTEMS (1990S)

- Fine-grained administrator control
- Slow (many network operations)
- Single point of failure
- History can be lost without backups



Speaker notes

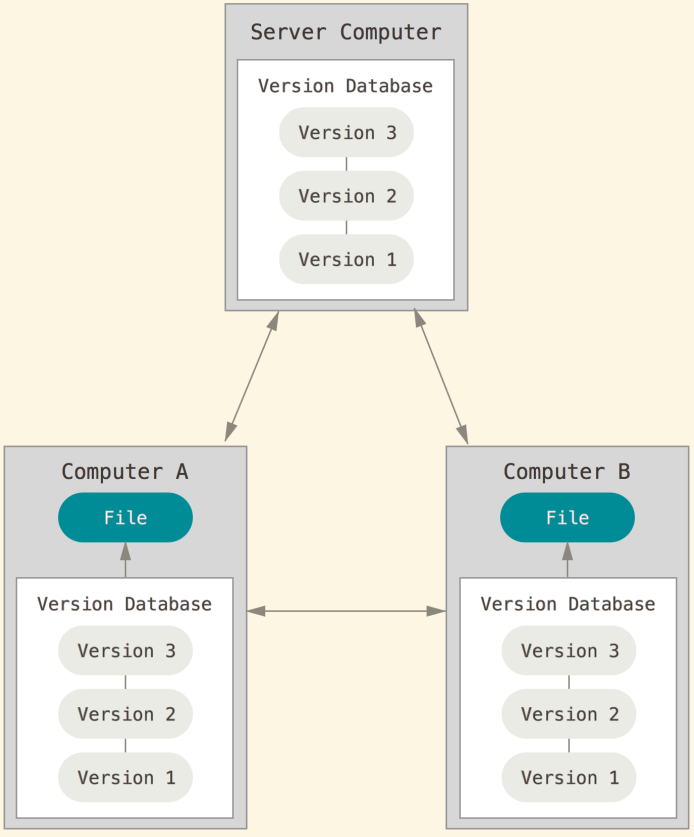
Centralized version control systems are based on a **single central server** that keeps all the versioned files.

Concurrent **V**ersion **S**ystems (CVS) and **S**ub**v**ersio**n** (SVN) are such systems that were first released in 1990 and 2000, respectively.

You could also consider storing your files in a shared Dropbox, Google Drive, etc. to be a kind of centralized version control system. However, it doesn't have as many tools for **consulting and manipulating the history** of your project, or to **collaborate on source code**.

DISTRIBUTED VERSION CONTROL SYSTEMS (2000+)

- Each client has a **full copy** of the project, so operations are local and **fast**.
- Basically any **collaborative workflow** is possible.
- **Administrators still have control** over their servers (but not collaborators' machines).



Speaker notes

Systems such as Git and Mercurial are **distributed**. This enables various types of collaborative workflows, since the team can organize itself however it wants.

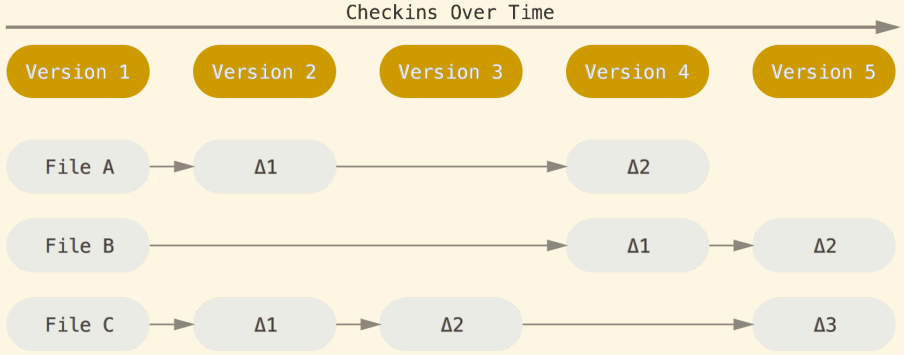
Clients **fully mirror** the repository, not just the latest snapshot. Because Git stores all versions of all files **locally**, most Git operations are almost instantaneous and do not require a connection to a server:

- Browsing the history
- Checking a file's changes from a month ago
- Committing

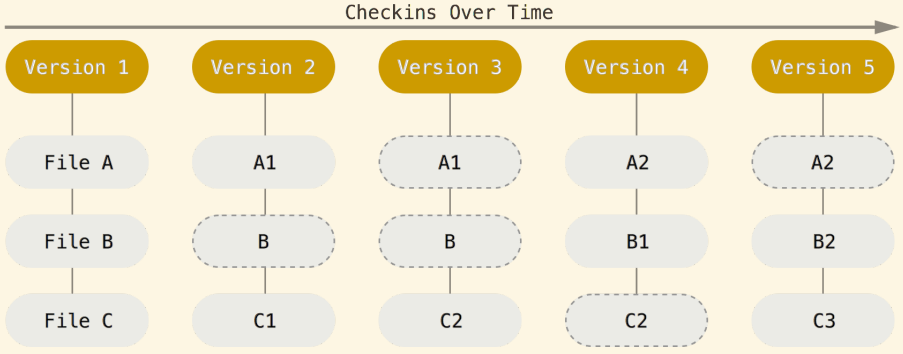
Git and Mercurial were first released in 2005.

GIT BASICS

SNAPSHOTS, NOT DIFFERENCES



Subversion



Git

Speaker notes

Unlike other version control systems, Git stores its data as **snapshots** instead of file-based changes. Git thinks of its data like a set of **snapshots** of a miniature filesystem.

Every time you 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 a stream of snapshots.

GIT HAS INTEGRITY

All Git objects are identified by a SHA-1 digest:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

You will see them all over the place in Git. Often you will only see a prefix (the first 6-7 characters):

```
24b9da6
```

Speaker notes

SHA-1 is a [hash function][cryptographic-hashfunction] and provides integrity. Because all content is hashed, it's virtually impossible for files to be lost or corrupted without Git knowing about it. This functionality is built into Git at the lowest levels and is integral to its philosophy.

WHAT'S IN A GIT PROJECT?

```
my-project:          # the working directory
├── .git:            # the git directory
│   ├── HEAD
│   ├── config
│   ├── hooks
│   ├── index        # the staging area
│   ├── objects
│   └── ...
├── file1.txt
├── file2.txt
└── dir:
    └── file3.txt
```

Speaker notes

A Git project has three main parts:

- The **Git directory**: this is where Git stores all the **snapshots** of the different **versions** of your files. This is the most important part of Git, and it is what is copied when you clone a repository from another computer or a server.

  You should never modify any of the files in this directory yourself; you could easily corrupt the Git repository. It is hidden by default, but you can see it on the command line.

- The **working directory**: it contains the **files you are currently working on**; that is, **one specific version** of your project. These files are pulled out of the compressed database in the Git directory and placed in your project's directory for you to use or modify:

- The **staging area** (also called the **index**), that stores information about **what will go into the next commit (or version)**.

  Before file snapshots are **committed** in the Git directory, they must go through the *staging area*.

WHAT'S IN A GIT PROJECT?

```
my-project:          # the working directory
├── .git:            # the git directory
│   ├── HEAD
│   ├── config
│   ├── hooks
│   ├── index        # the staging area
│   ├── objects
│   └── ...
├── file1.txt
├── file2.txt
└── dir:
    └── file3.txt
```

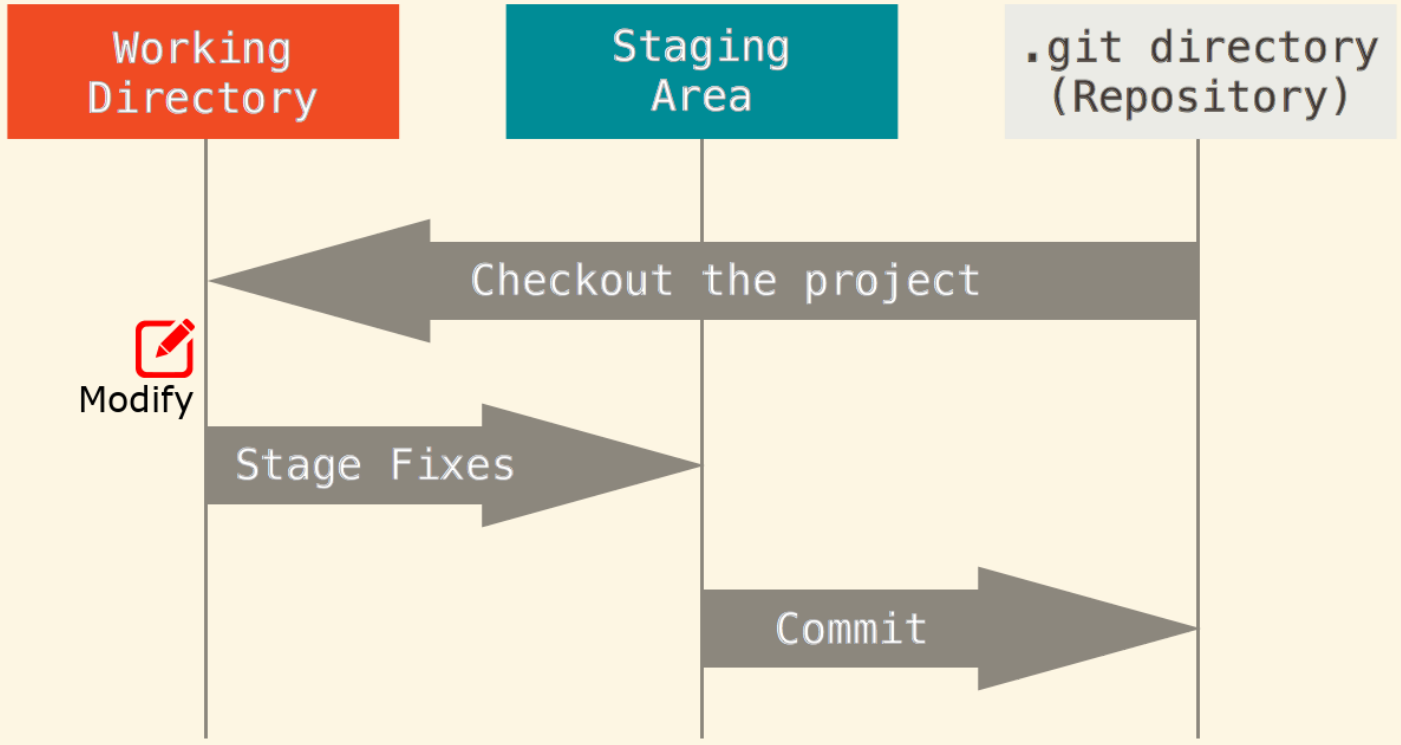
WHAT'S IN A GIT PROJECT?

```
my-project:          # the working directory
├── .git:            # the git directory
│   ├── HEAD
│   ├── config
│   ├── hooks
│   ├── index        # the staging area
│   ├── objects
│   └── ...
├── file1.txt
├── file2.txt
└── dir:
    └── file3.txt
```

WHAT'S IN A GIT PROJECT?

```
my-project:          # the working directory
├── .git:            # the git directory
│   ├── HEAD
│   ├── config
│   ├── hooks
│   ├── index        # the staging area
│   ├── objects
│   └── ...
├── file1.txt
├── file2.txt
└── dir:
    └── file3.txt
```

THE BASIC GIT WORKFLOW



Speaker notes

This is one of the **most important things to remember about Git**:

- You **check out** (or **switch to**) a specific version of your files into the *working directory*.
- You **modify** files (or add new files) in your *working directory*.
- You **stage** the files, adding snapshots of them to your *staging area*.
- You make a **commit**, which takes the files as they are in the *staging area* and stores these snapshots permanently to your *Git directory*.

USING THE STAGING AREA

Speaker notes

New snapshots of files **MUST go through the staging area** to be **committed** into the Git directory.