Understanding Git / GitHub

1.1 Introduction to version control

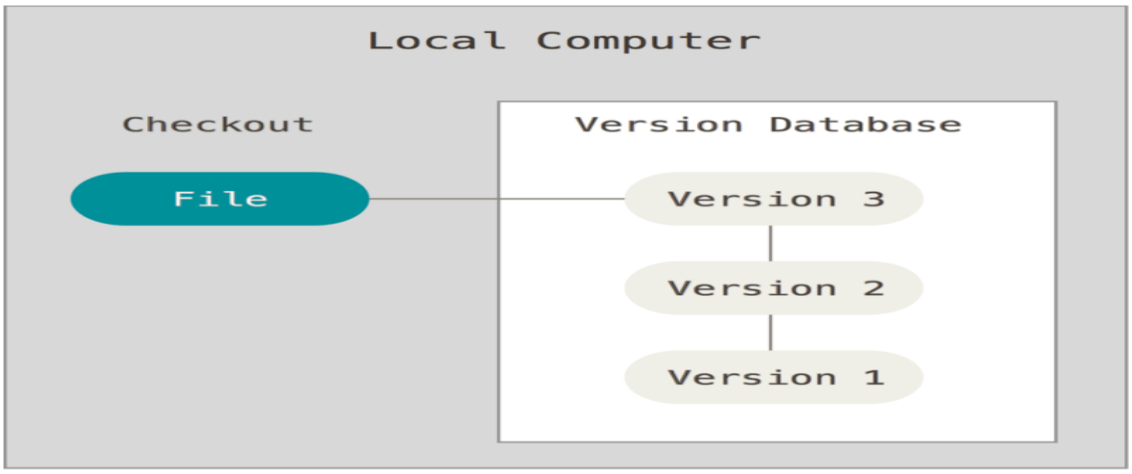
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

1.1.1 Responsibilities of Version Control System

1. Version control, also known as source control, is the practice of tracking and managing changes to software code.
2. Version control software keeps track of every modification to the code in a special kind of database- repository.
3. One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.
4. VCS are sometimes known as SCM (Source Code Management)
5. One of the most popular VCS tools in use today is called Git.
6. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source.

1.1.2 Local Version Control Systems

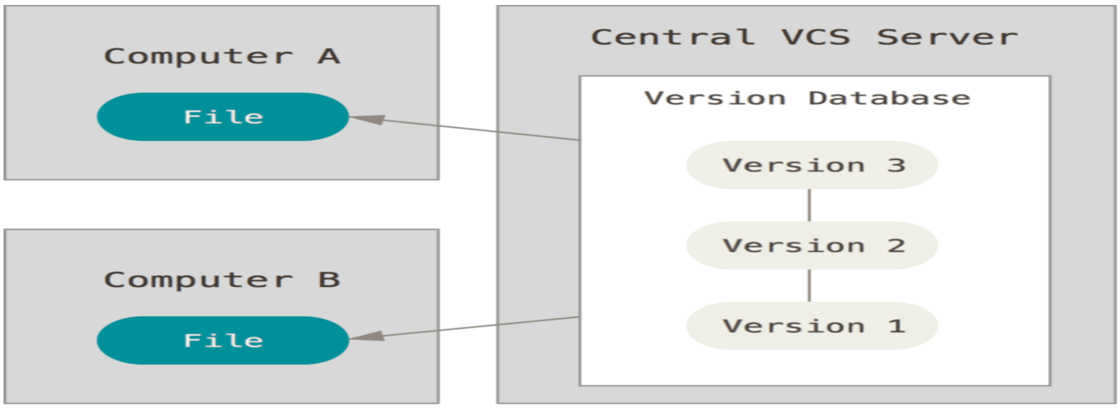
Many people’s version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they’re clever). This approach is very common because it is so simple, but it is also incredibly error prone.



1.1.3 Centralized Version Control Systems

SUB VERSION
TEAM FOUNDATION SERVER

The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.

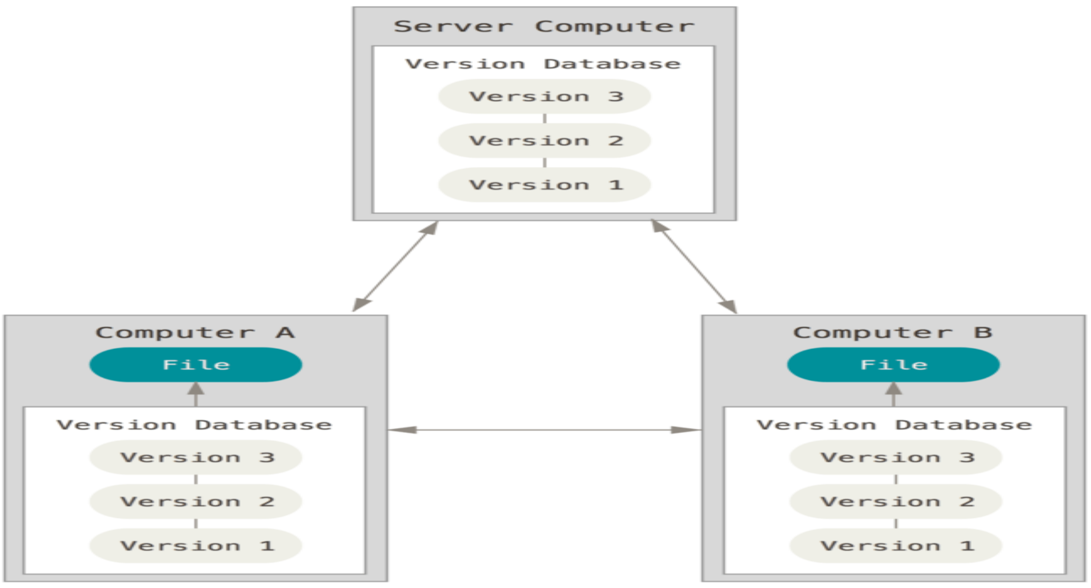


However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they’re working on. If the hard disk the central database is on becomes corrupted, and proper backups haven’t been kept, you lose absolutely everything.

1.1.4 Distributed Version Control Systems

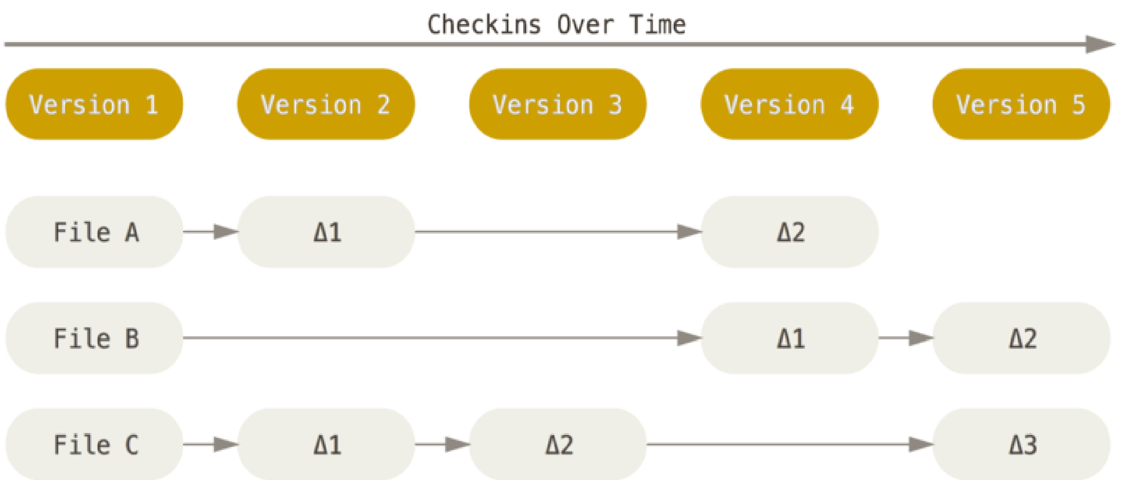
GIT
MERCURIAL

This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial or Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.



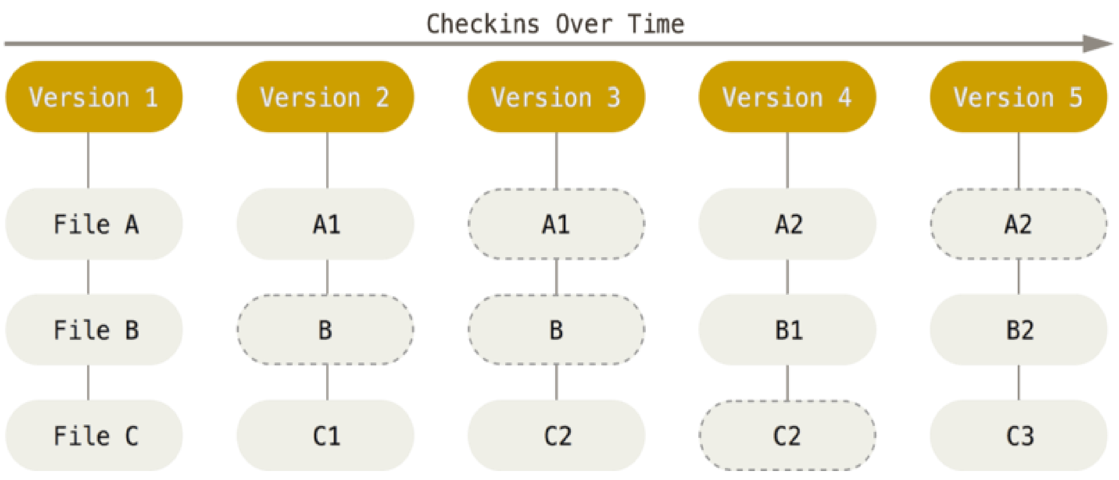
1.2 What Is GIT

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 other systems (CVS, Subversion, Perforce, and so on) think of the information they store as a set of files and the changes made to each file over time (this is commonly described as delta-based version control).



Storing 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 series of snapshots of a miniature filesystem. With Git, every time you commit, or save the state of your project, Git 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**.



Storing 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.

1.2.1 The Three States of GIT

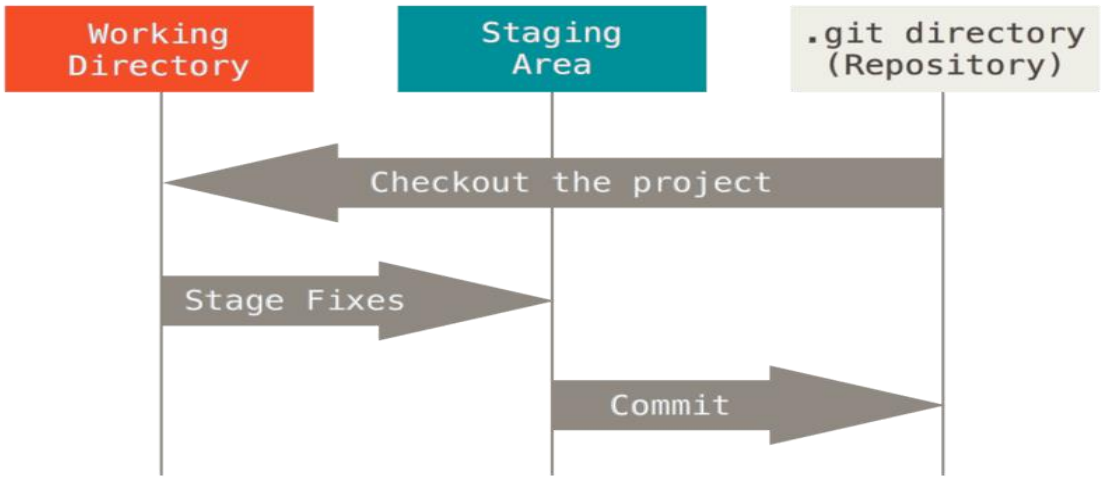
Git has three main states that your files can reside in: modified, staged, and committed:

1. Modified means that you have changed the file but have not committed it to your database yet.
2. Staged means that you have marked a modified file in its current version to go into your next commit snapshot.
3. Committed means that the data is safely stored in your local database.

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

1.2.2 Why GIT?

- Free
- Open Source
- Scalable
- Fast



1.2.3 Downloading and Installing Git

Please visit the following page and download the file based on the type of operating system you are using.

https://git-scm.com/downloads

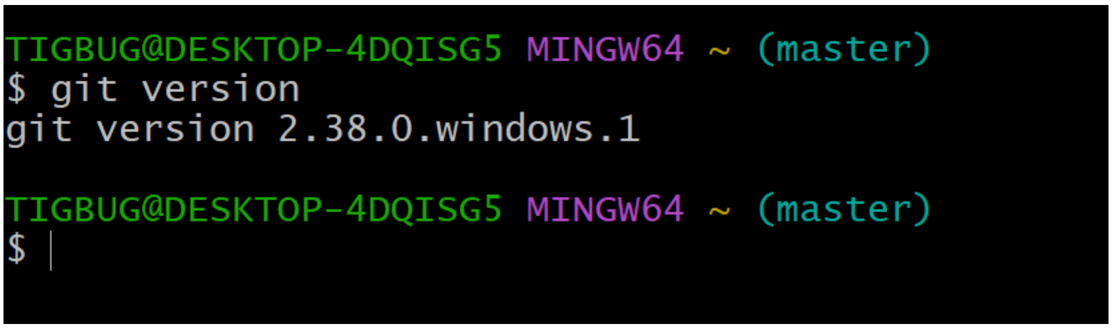
After downloading choose the default settings and install the application.

1.2.4 Using the shell

Git commands are typically performed using the Shell. Understanding some common shell commands allows you to perform more of your Git workflow in the shell without having to spend time navigating different programs.

Example: To check the version of the GIT installed type the following command at the prompt.

$ git version



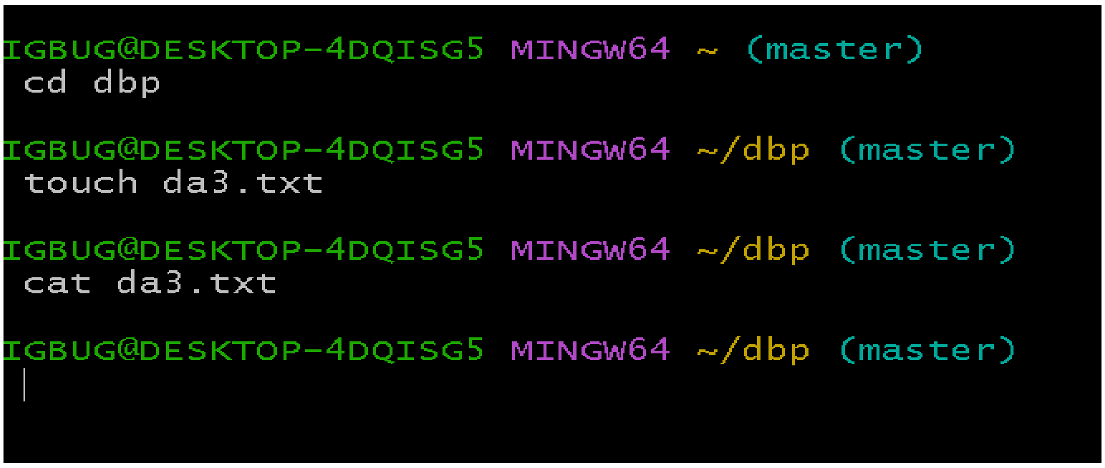
```
TIGBUG@DESKTOP-4DQISG5 MINGW64 ~ (master)
$ git version
git version 2.38.0.windows.1

TIGBUG@DESKTOP-4DQISG5 MINGW64 ~ (master)
$ |
```

1.2.5 Git Commands

1.2.5.1 *Making a directory, navigating into new directory, creating an empty file and checking the contents of the file.*

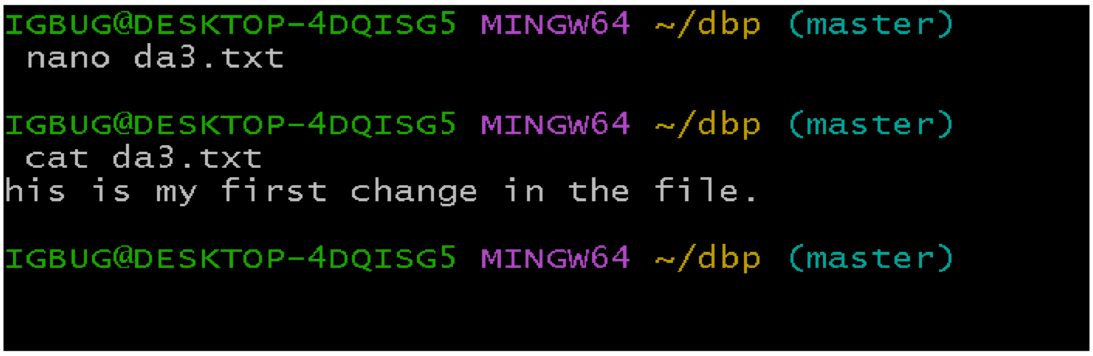
Below commands creates a new directory with the name specified.

$ mkdir dbp



```
IGBUG@DESKTOP-4DQISG5 MINGW64 ~ (master)
 cd dbp

IGBUG@DESKTOP-4DQISG5 MINGW64 ~/dbp (master)
 touch da3.txt

IGBUG@DESKTOP-4DQISG5 MINGW64 ~/dbp (master)
 cat da3.txt

IGBUG@DESKTOP-4DQISG5 MINGW64 ~/dbp (master)
 |
```

1.2.5.2 *Creating a file using nano command to add text into the file & Checking the contents of the file using cat command.*



```
IGBUG@DESKTOP-4DQISG5 MINGW64 ~/dbp (master)
 nano da3.txt

IGBUG@DESKTOP-4DQISG5 MINGW64 ~/dbp (master)
 cat da3.txt
his is my first change in the file.

IGBUG@DESKTOP-4DQISG5 MINGW64 ~/dbp (master)
```

We will be also learning more about the following steps in our session.

- Working directory - Initializing the folder into repository
- Creating a repository in GIT
- Checking the status of the repository
- Establishing connection to remote repository.
- Moving files to remote repository
- Check the web version of the repository for the updated version
- Checking the log.
- Cloning the repository
- Collaborating with other developers.