Git (Global information tracker)

Version control

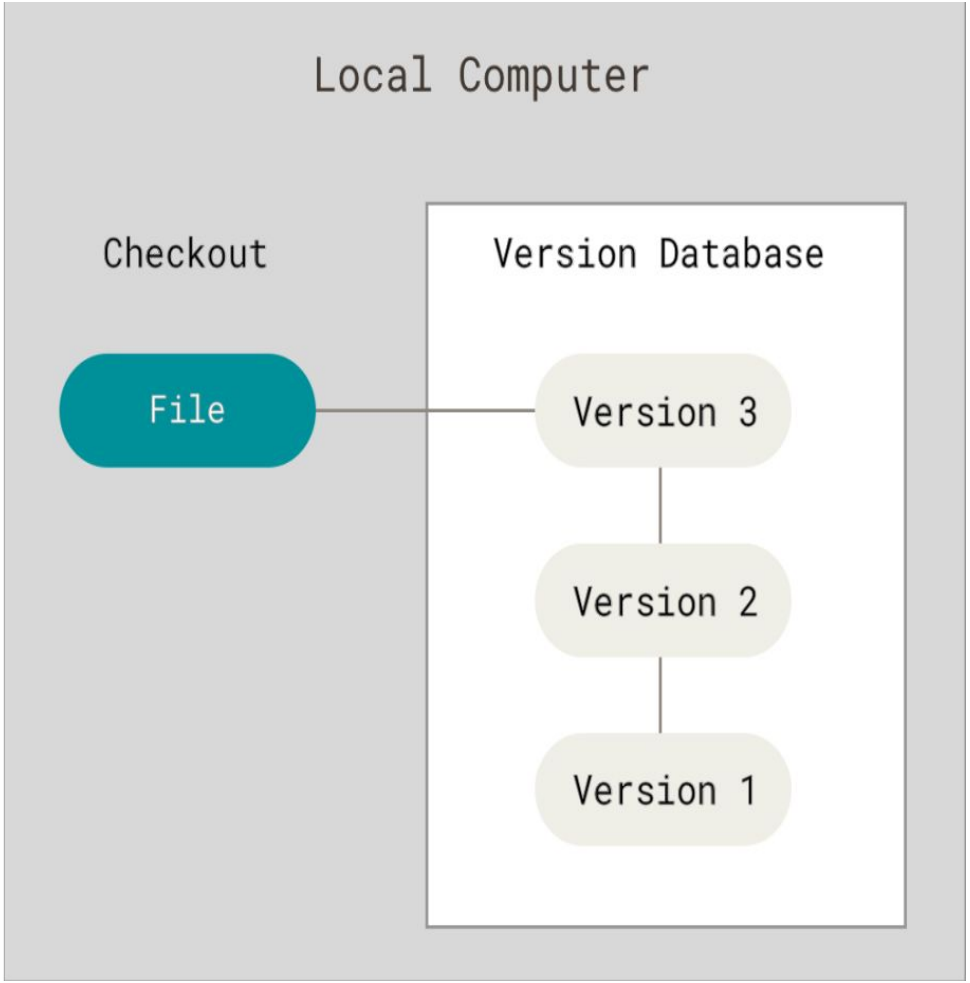
Version control is a system that records changes to a file or set of files over time so that you can revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something .

Why version control?.

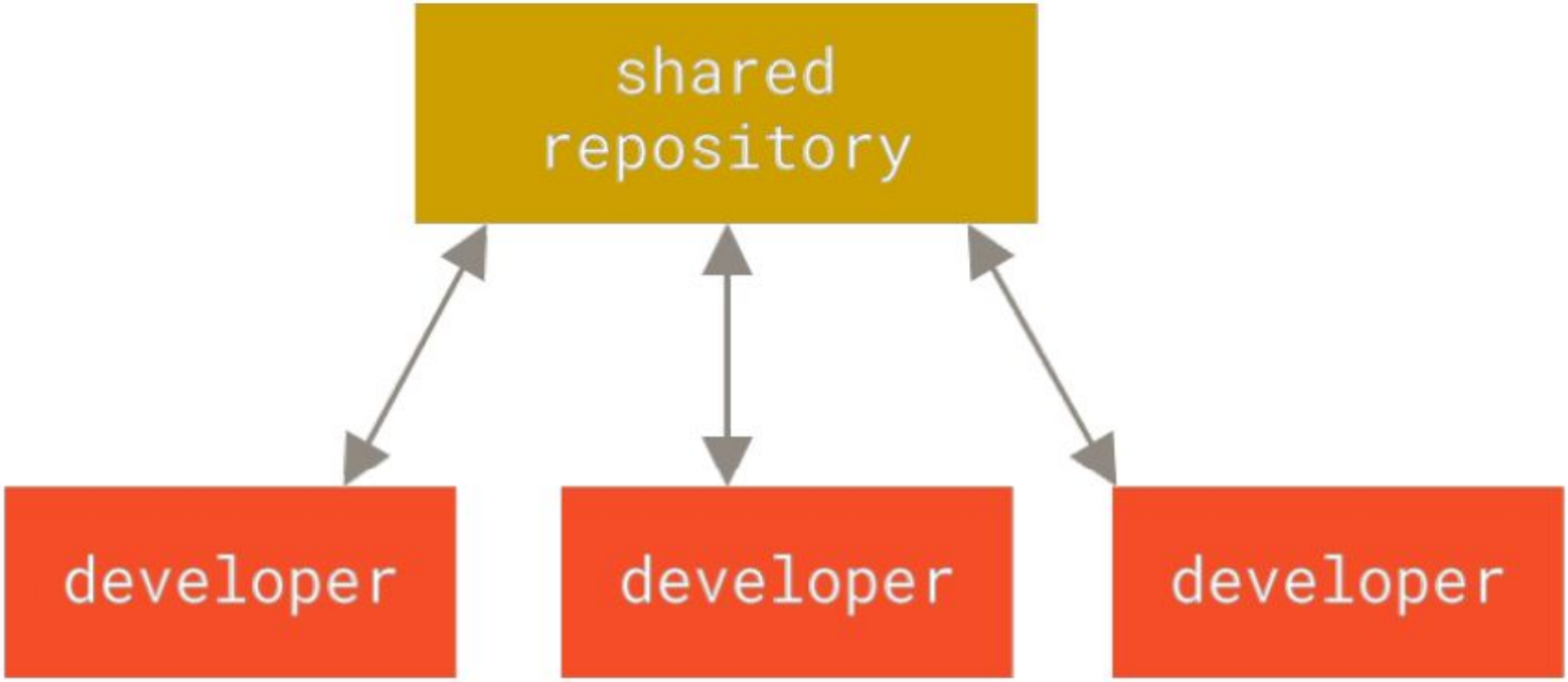
Collaboration
Version control
Rollback (if any problem arise move back to older version)

Local version control

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. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

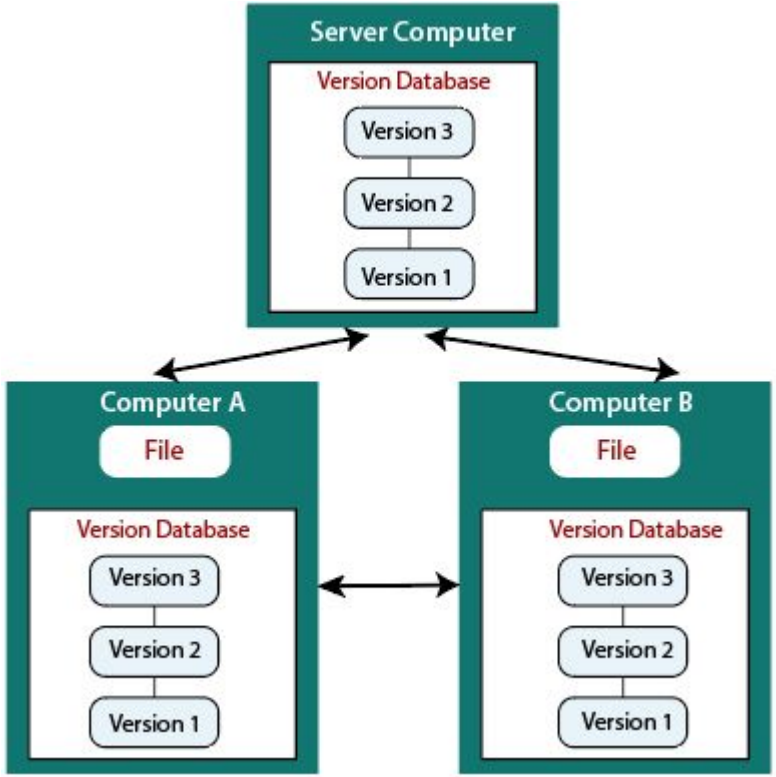


Centralized version control systems (CVCSs)



Distributed version control systems

Here not completely reliable on the server, as everyone have local copy of clone of the center project file in their system.
Only push and pull command is on the cloud as everything present run on local environment.



What is Git?. Why 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).

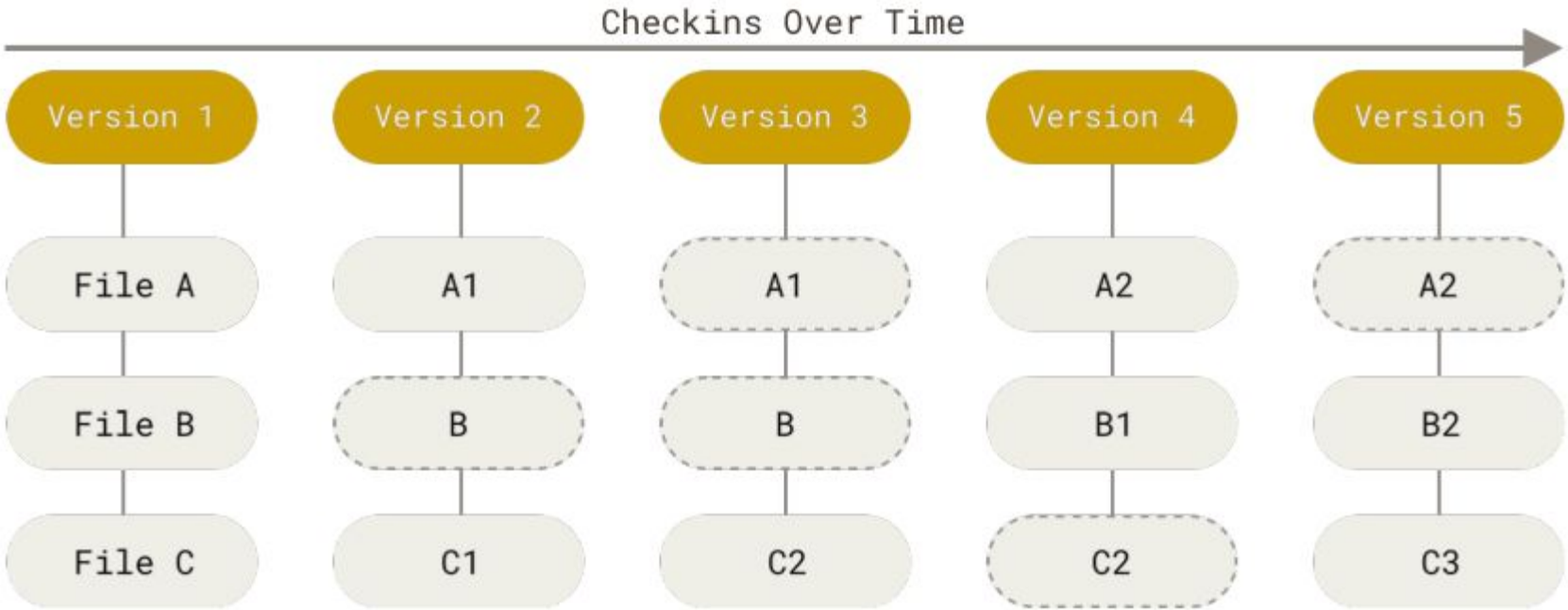


Figure 5. Storing data as snapshots of the project over time

Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network.

Everything in Git is checksummed before it is stored and is then referred to by that checksum
Checksum is a value that represents the number of bits in a transmission message.
Git use SHA-1 hash for checksumming.
It has 40 character, hexadecimal characters(0-9 and a-f)
Example:

24b9da6552252987aa493b52f8696cd6d3b00373

It only add data to it Database. It doesn't render the file or repo

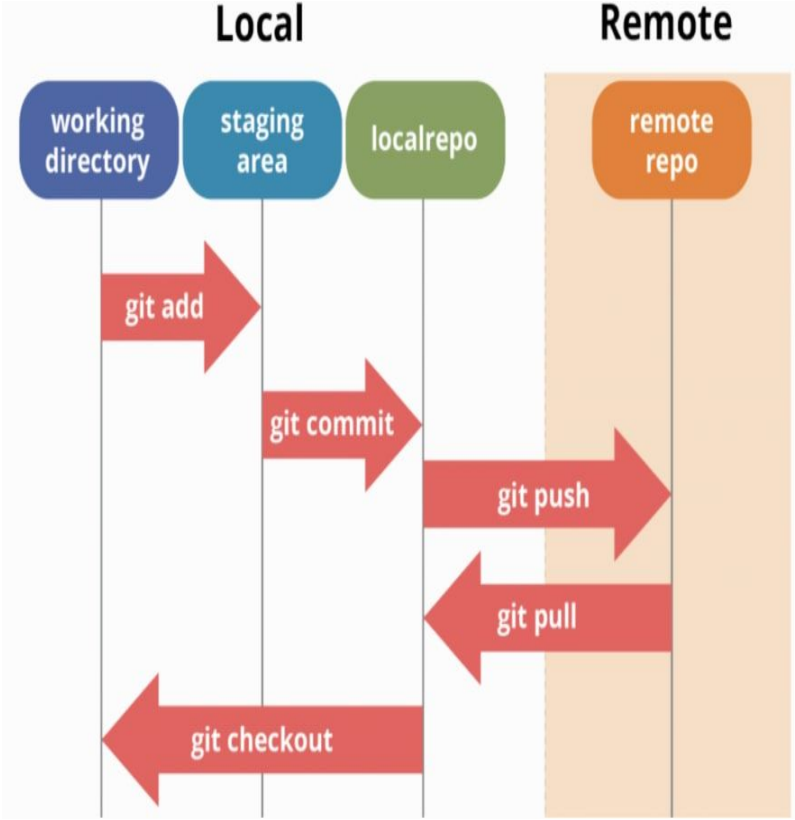
Git workflow

The basic Git workflow goes something like this:

1. You modify files in your working directory.

2. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.

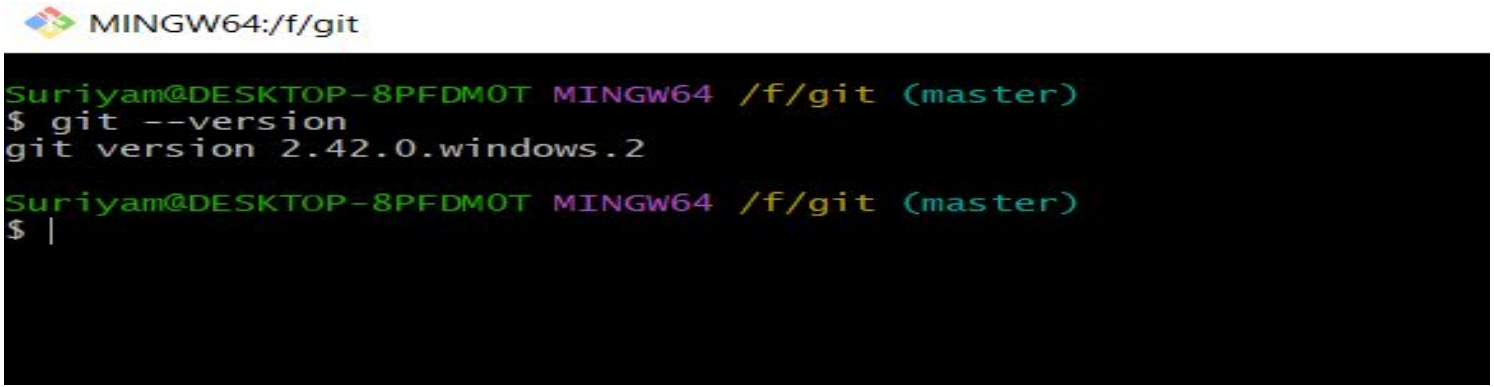
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.



Git Setup

1. Install git from Git-download and install the exe file and automatically setup every thing for you.
2. Open git bash terminal and run the command git --version
3. You can view all of your settings and where they are coming from using:
   $ git config --list --show-origin
4. Then set your user name and email
   $ git config --global user.name ‘your user name’
   $ git config --global user.email ‘your email address’
5. Check that user name and email is correct by
   $ git config --list - -show-origin

- If you want to use a different text editor, such as Emacs, you can do the following:
  $ git config --global core.editor emacs
- How to change or set the default branch that initial
  $ git config --global init.defaultBranch main
- You can check the your setting :
  $ git config --list
- If you were not remember some command then you can use git help:
  $ git help config
- If you only want know available option in git rather referring manual
  $git add -h




```
Suriyam@DESKTOP-8PFDMOT MINGW64 /f/git (master)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.cr
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=suryam-s
user.email=suryamsourya8@gmail.com
color.ui=auto
init.defaultbranch=main
init.dfaultbranch=main
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
....skipping...
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
```

```
Suriyam@DESKTOP-8PFDM0T MINGW64 /f/git (master)
$ git config --global user.name "S-sourya"

Suriyam@DESKTOP-8PFDM0T MINGW64 /f/git (master)
$ git config --global user.email "suryamsourya4567@gmail.com"

Suriyam@DESKTOP-8PFDM0T MINGW64 /f/git (master)
$ git config --global user.name
S-sourya

Suriyam@DESKTOP-8PFDM0T MINGW64 /f/git (master)
$ git config --global user.email
suryamsourya4567@gmail.com

Suriyam@DESKTOP-8PFDM0T MINGW64 /f/git (master)
$
```

Git Basic command

- Git init
- Git clone
- Git add
- Git commit
- Git status
- Git branch
- Git push
- Git pull
- Git remote add origin
- Git checkout

Setup and init

- Git init
  - The git init command create a new repository. It can be used to convert an existing unversioned project to git repository.
  - Used to implement git functionality in normal (new) folder.
- Gti clone
  - Used to download the repository files in the system from the github.
  - Used create a copy in the local machine

- Git config
  - It is function that help in set up of the git configuration values on a global level and local.
  - Git comes with a tool called git config that lets you get and set configuration variables that control all aspects of how Git looks and operates.
- Git add
  - Used to add desired file to staging area (to track)
  - The staging area gives you a chance to decide which files you really want to track and which ones you don’t want.
  - Git add -A =>is used to add all files in the current directory to the staging area.
  - To unstage it use $git reset HEAD <fileName >
- Git commit
  - Used to commit / finalize the desired files to be tracked by git.
  - Git commit always require a commit message
  - $git commit -m “your message”

Git remote add origin

- Used to link your remote repository (GitHub repository) with your local system repository.
- Usage: git remote add origin [remote repo link]

Git push and git pull

Git push

> Transmit local branch commits to the remote repository branch
> Usage: git push

Git pull

> fetch and merge any commits from the tracking remote branch
> Usage: git pull

Stage and snapshot

git status
show modified files in working directory, staged for your next commit

git add [file]
add a file as it looks now to your next commit (stage)

git reset [file]
unstage a file while retaining the changes in working directory

git diff
diff of what is changed but not staged

git diff --staged
diff of what is staged but not yet committed

Branch and merge

Isolating work in branches, changing context, and integrating changes

git branch
list your branches. a * will appear next to the currently active branch

git branch [branch-name]
create a new branch at the current commit

git checkout
switch to another branch and check it out into your working directory

git merge [branch]
merge the specified branch’s history into the current one

git log
show all commits in the current branch’s history

Inspect and compare

Examining logs, diffs and object information

git log
show the commit history for the currently active branch

git log branchB..branchA
show the commits on branchA that are not on branchB

git log --follow [file]
show the commits that changed file, even across renames

git diff branchB...branchA
show the diff of what is in branchA that is not in branchB

git show [SHA]
show any object in Git in human-readable format

End