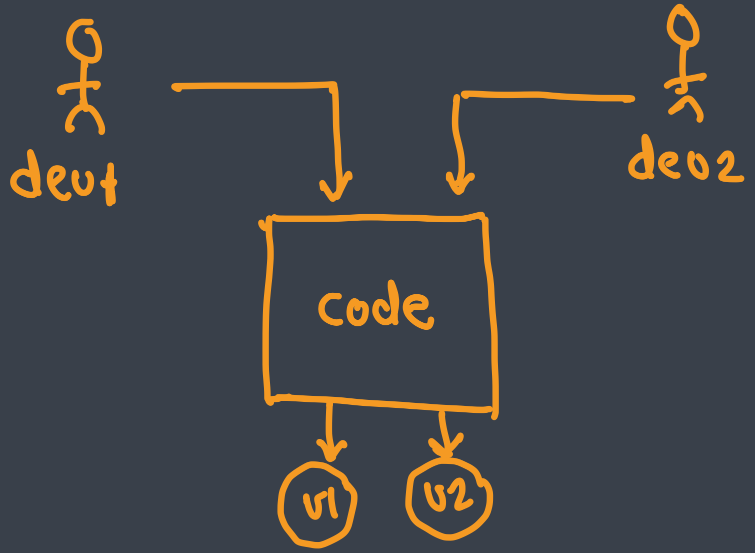
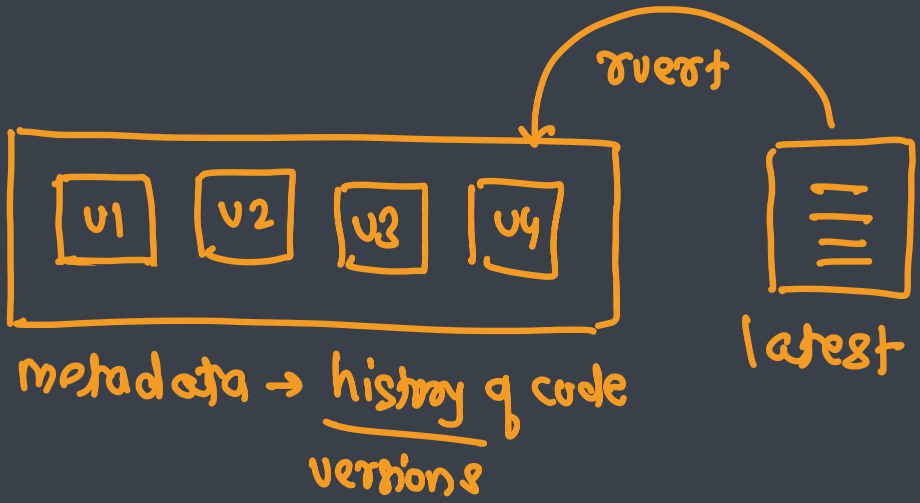
Source Code Management / Version Control System

Version Control System

- Version control is a system that allows the software team to manage changes to the source code over time
- This software tool makes it easier for developers to collaborate on different projects separating their tasks through branches
- It also gives the possibility to turn back to earlier versions for comparing and fixing the mistakes if needed
- Version Control Systems (VCS) also known as SCM (Source Code Management) or RCS (Revision Control System) are software tools for keeping track of changes to the source code over time





Benefits

- Long-term change history
  - The changes made by developers, including the creating, modification, and deletion of files over the years, can be seen in history
  - It will allow going back to the previous version for analyzing bugs and fixing problems
- Branching and merging → parallel development
  - Branching helps work in an independent manner and not interfere with each other's work
  - Merging brings the works together and allows seeing if there are conflicts between those works
- Traceability → track the changes → blame
  - Ability to trace each change and connect it to project management and bug tracking software, as well as to annotate each change with a message describing the purpose of the change
- Synchronization → different systems
  - The up-to-date codes can be fetched from the repository
- Backup and Restore
  - Files are saved at any time and restored from the last saved one
- Undoing → revert back to the last version
  - You can undo both the last known version and the last one created a long time ago
- ~~Branching and Merging~~
  - Changes are made on a branch and after being approved, they can be merged with the master branch

Types

$$\begin{tikzcd}
 & \text{Types} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\text{LVCS} & \text{CVCS} & \text{DVCS}
\end{tikzcd}$$

deprecated $\left\{ \frac{\text{LVCS}}{\text{local}} \right.$

$\frac{\text{CVCS}}{\text{centralized}}$

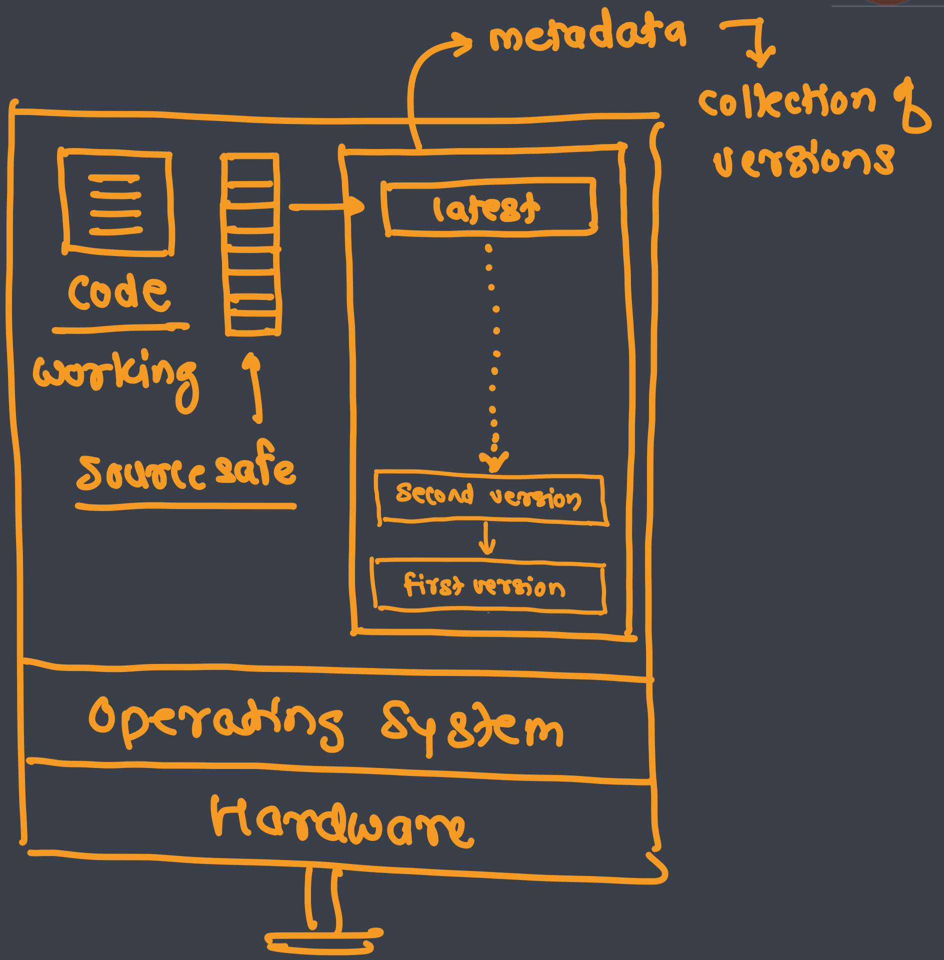
$\frac{\text{DVCS}}{\text{Distributed}}$

Local Version Control System

- Local VCSs were created to prevent issues like confusing the directories and accidentally writing or copying to the wrong file
- It is a simple database that keeps all the changes to files under revision control
- One of the most popular VCS tools was a system called Revision Control System (RCS), which is still distributed today, although being an earlier version control system
- It allows users to make their revisions of a document, commit changes, and merge them. RCS was originally developed for programs but is also useful for text documents or configuration files that are frequently revised

Cons

- No collarboration
- single point of failure



Centralized Version Control System

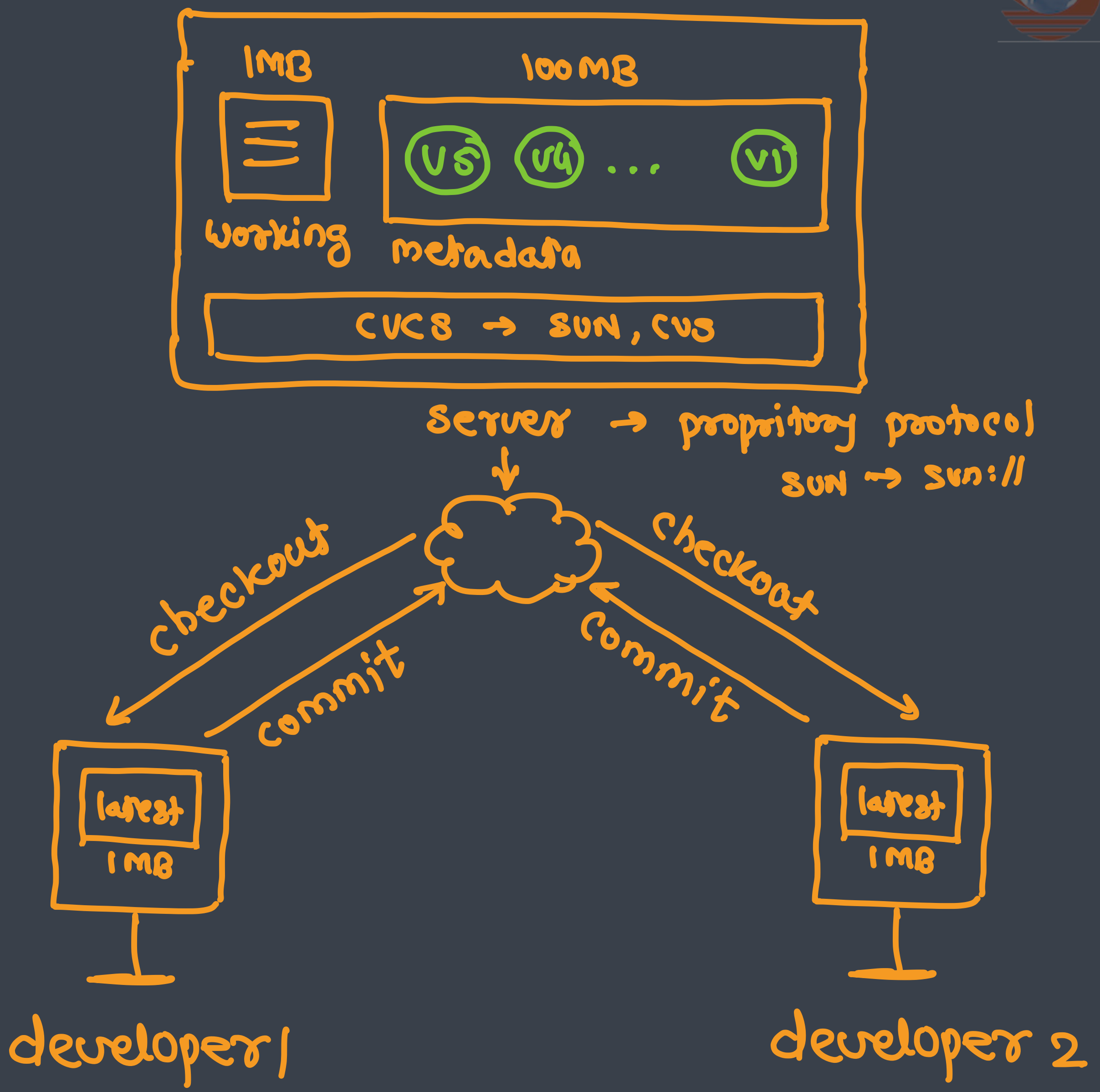
- In centralized systems, all the versioned files, as well as a number of clients that check out files from that central place, are included in a single server
- For many years, this has been the standard for version control
- Centralized Version Control Systems are CVS, Subversion, and Perforce

pros

- collaboration with others
- saving network bandwidth and disk storage of client machines

cons

- server has single point of failure
- no scaling



Distributed Version Control System

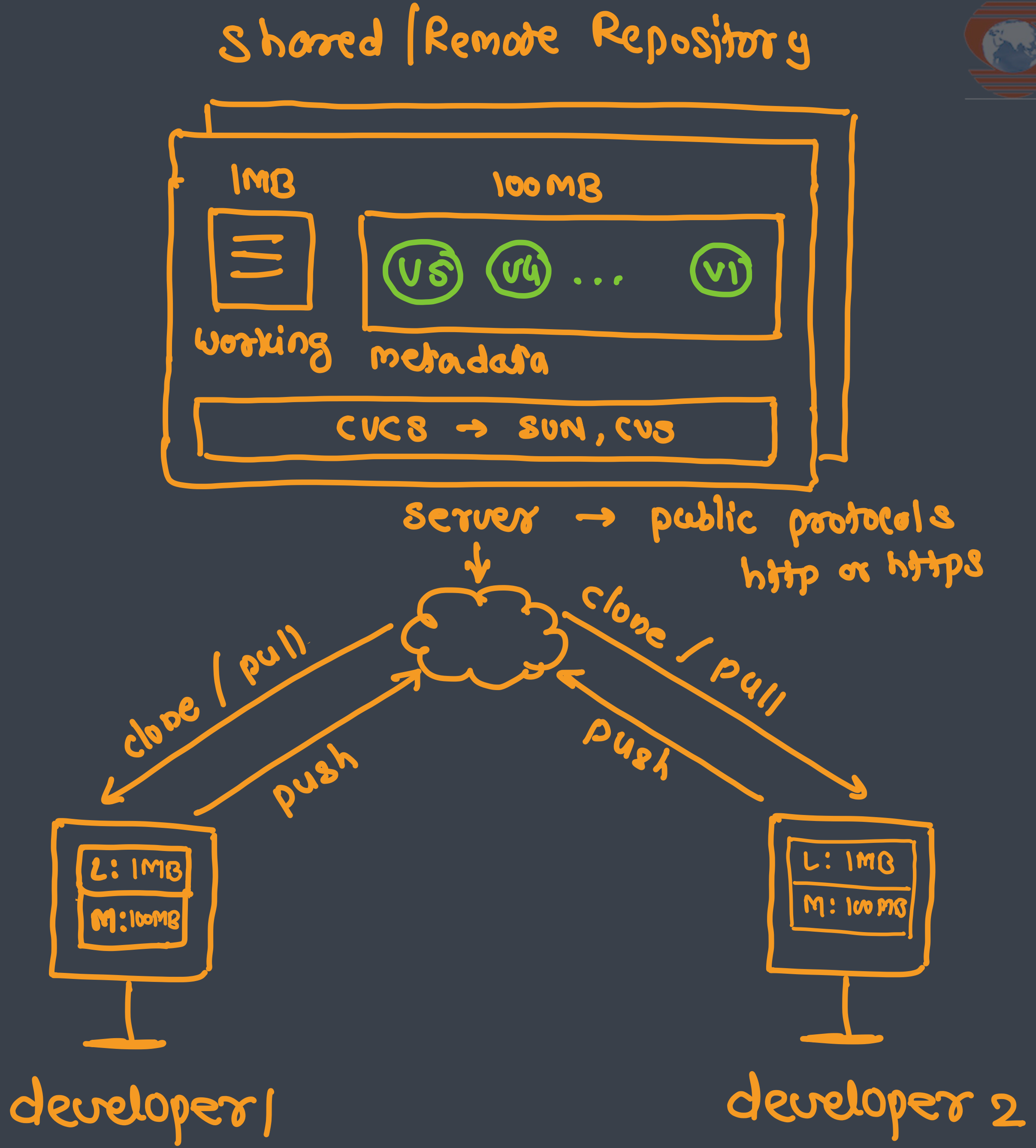
- In Distributed Version Control Systems (DVCS), clients fully mirror the repository, including its full history
- If the server that these systems were collaborating with dies, the client repositories can be copied back up to the server to restore it
- Distributed Version Control Systems are Git, Mercurial, Bazaar or Darcs

pros

- No single point of failure
- horizontally scaled server

CONS

- increases network bandwidth and disk storage
- redundancy



Git

What is Git ?

- Git is a distributed revision control and source code management system
- Git was initially designed and developed by Linus Torvalds for Linux kernel development
- Git is a free software distributed under the terms of the GNU General Public License version 2

History

- The Linux kernel is an open source software project of very large scope
- From 1991–2002, changes to the software were passed around as patches and archived files
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper
- In 2005, the relationship with BitKeeper broken down and tool's free-of-charge status was revoked
- tool's free-of-charge status was revoked (and in particular Linus Torvalds) to develop their own tool based on some of the lessons they learned while using BitKeeper
- Some of the goals of the new system were
  - Speed → better performance
  - Simple design → onion architecture
  - Strong support for non-linear development (thousands of parallel branches) → branching
  - Fully distributed → No sigle point of failure
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

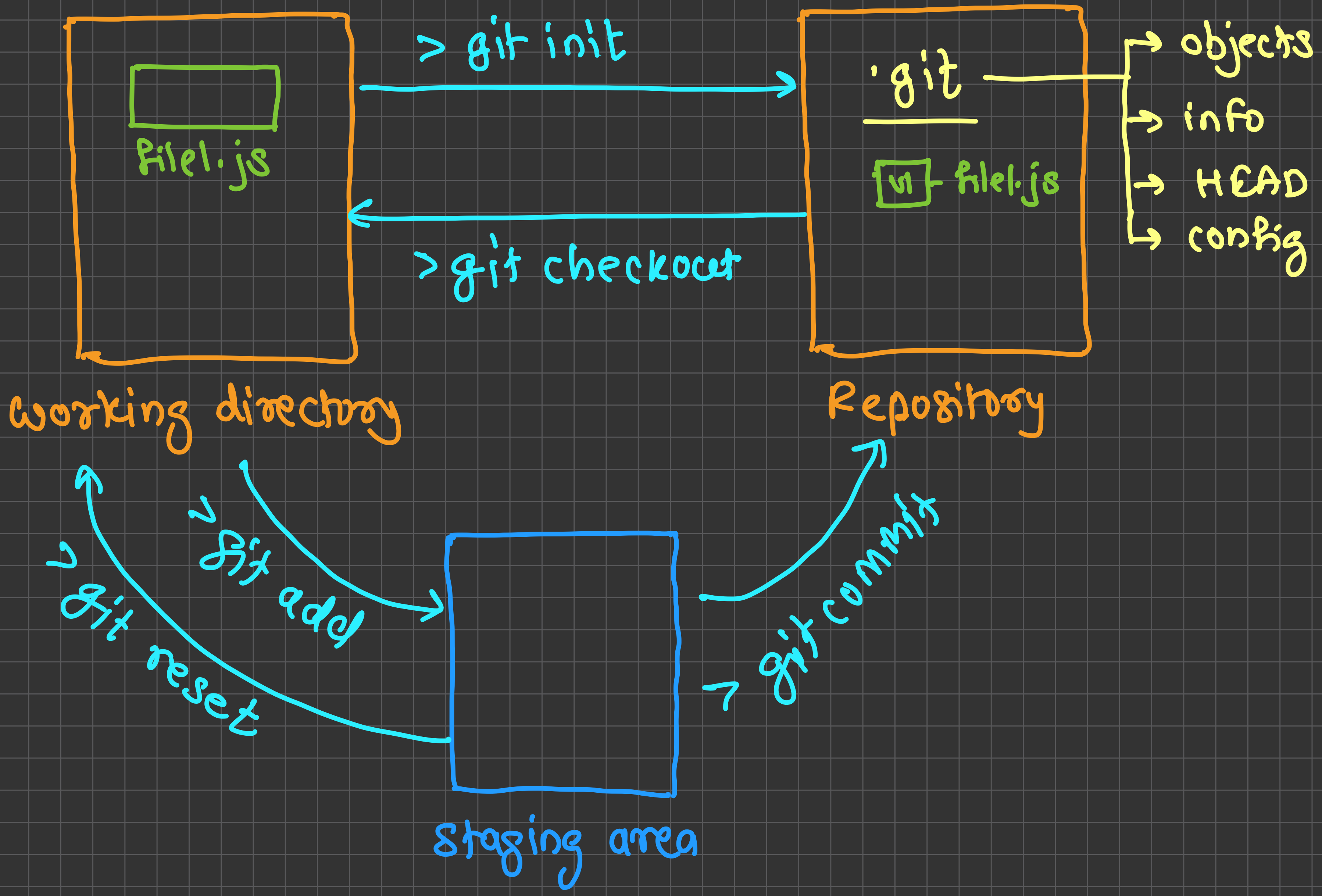
Characteristics

- Strong support for non-linear development → branching
- Distributed development
- Compatibility with existent systems and protocols → http or https
- Efficient handling of large projects → No restriction on size
- Cryptographic authentication of history → objects → entity to store the versions
- Toolkit-based design → porcelain commands and client commands
- Pluggable merge strategies → branching

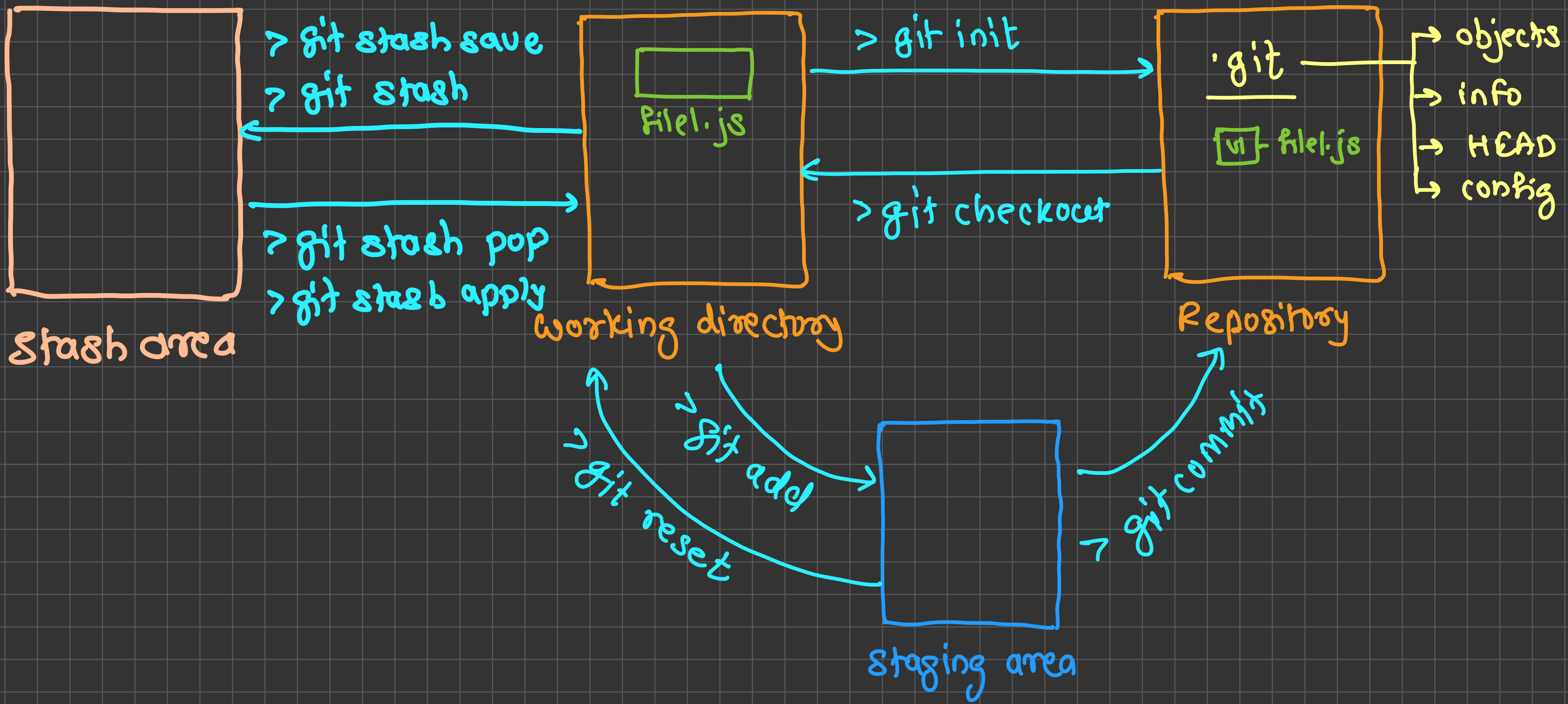
Advantages

- Free and open source
- Fast and small
- Implicit backup
- Security
- No need of powerful hardware
- Easier branching ✱ ✱

Basic Workflow



> git init
> git checkout
file1.js
.git
v1
file1.js
objects
info
HEAD
config
working directory
Repository
> git add
> git reset
> git commit
staging area


> git stash save
> git stash
> git stash pop
> git stash apply
stash area
file1.js
working directory
> git init
> git checkout
.git
v1
file1.js
Repository
objects
info
HEAD
config
> git add
> git reset
> git commit
staging area

git init

- The git init command is used to generate a new, empty Git repository or to reinitialize an existing one
- With the help of this command, a .git subdirectory is created, which includes the metadata, like subdirectories for objects and template files, needed for generating a new Git repository

git config

- The git config command is a function that sets configuration variables
- It controls git look and operation
- Levels
  - Local (--local)
    - When no configuration option is passed git config writes to a local level, by default
    - The repository of the .git directory has a file that stores local configuration values
  - Global (--global)
    - The application of the global level configuration includes the operating system user
    - Global configuration values can be found in a file placed in a user's home directory
  - System (--system)
    - The System-level configuration includes all users on an operating system and all repositories
    - System-level configuration file is located in a git config file of the system root path

git add

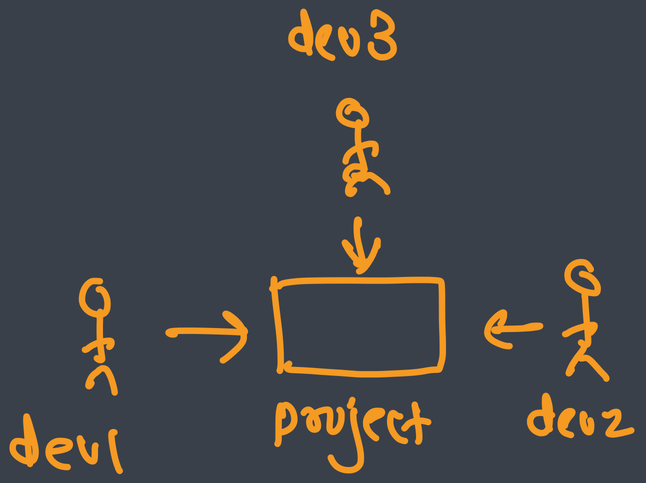
- The git add is a command, which adds changes in the working directory to the staging area
- With the help of this command, you tell Git that you want to add updates to a certain file in the next commit
- But in order to record changes, you need to run git commit too
- In combination with the commands mentioned above, git status command is also needed to see which state the working directory and the staging area are in

git commit

- The git commit command saves all currently staged changes of the project
- Commits are created to capture the current state of a project
- Committed snapshots are considered safe versions of a project because Git asks before changing them
- Before running git commit command, git add command is used to promote changes to the project that will be then stored in a commit
- Working of commit
  - Git snapshots are committed to the local repository
  - Git creates an opportunity to gather the commits in the local repository, rather than making a change and commit it immediately to the central repository
  - This has many advantages splitting up a feature into commits, grouping the related commits, and cleaning up local history before committing it to the central repository
  - This also gives the developers an opportunity to work in an isolated manner

git log

- The git log command shows committed snapshots
- It is used for listing and filtering the project history, and searching for particular changes
- The git log only works on the committed history in comparison with git status controlling the working directory and the staging area



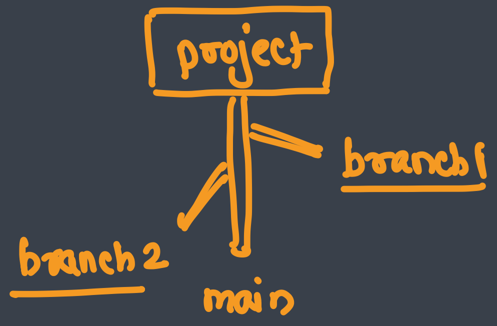
Branching

parallel development

Branching

- Branching allows developers to branch out from the original code base and work separately
- Allows another line of development → parallel development
- A way to write code without affecting the rest of your team
- Generally used for feature development
- Once confirmed the feature is working you can merge the branch in the master branch and release the build to customers

  main
- Why is it required ?
  - So that you can work independently / parallel
  - There will not be any conflicts with main code
  - You can keep unstable code separated from stable code
  - You can manage different features keeping away the main line code and there wont be any impact of the features on the main code



• main / master brach has
- well-tested code
- latest (with latest features)
- crash proof
- bug free
- stable code

main branch is mainly used to build and deploy the code using CI/CD pipeline

git branch

- The git branch command creates, lists and deletes branches
- It doesn’t allow switching between branches or putting a forked history back together again
- Git branches are a pointer to a snapshot of the changes you have made
- A new branch is created to encapsulate the changes when you want to fix bugs or add new features
- This helps you to clean up the future's history before merging it
- Git branches are an essential part of everyday workflow
- Git does not copy files from one directory to another, it stores the branch as a reference to a commit