Introduction

1WEBD – Javascript Web Development

Sommaire

1. Histoire des VCS
2. Introduction à GIT
3. Introduction à Github

1. Histoire des VCS

1. Histoire des VCS

VCS ?

- Version Control System (Logiciel de gestion de versions
- Système de stockage chronologique de fichier
  - Stocke différentes versions du fichier dans le temps
  - Permet de remonter vers des anciennes versions

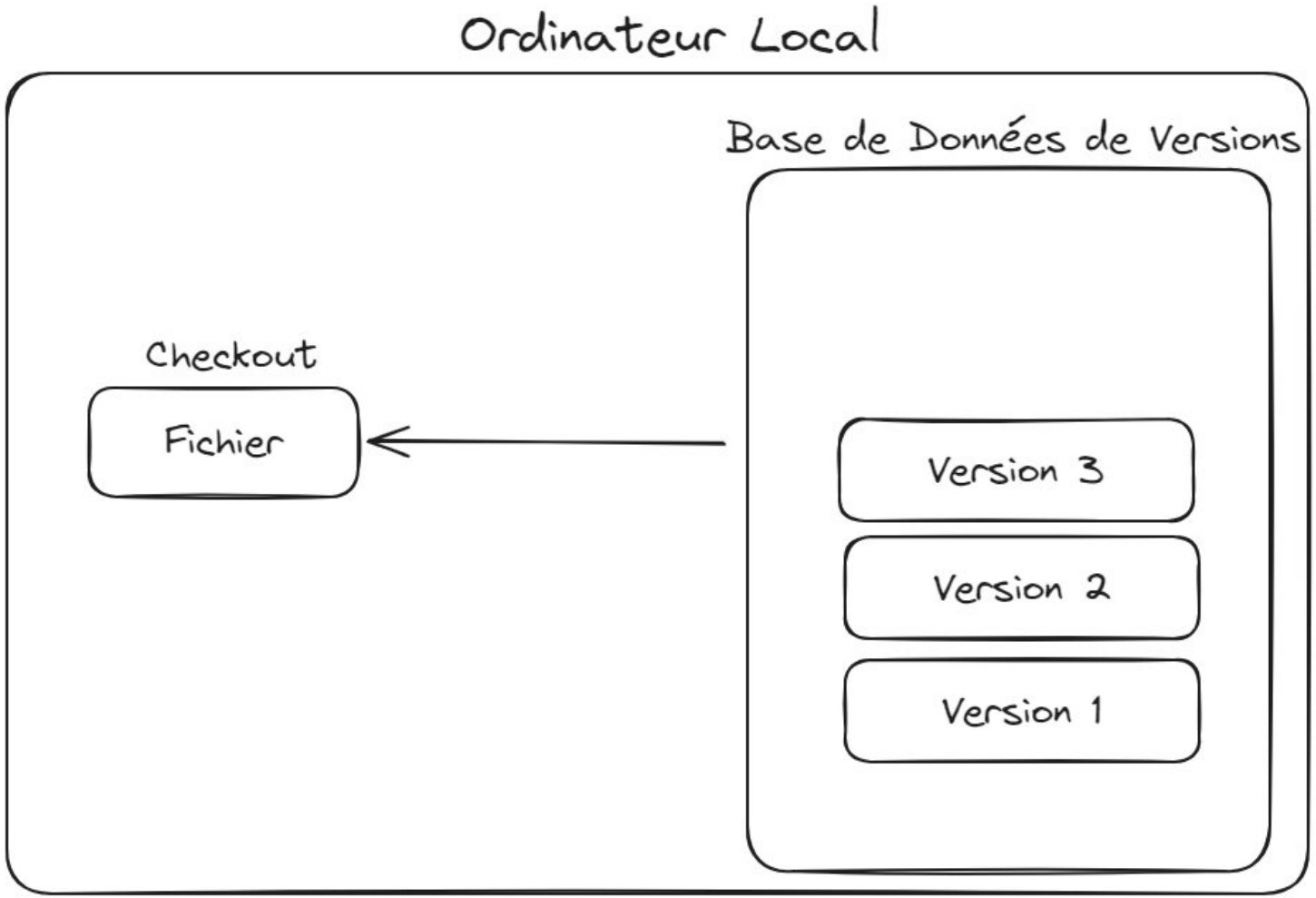
1. Histoire des VCS

L’histoire des VCS (Version Control System

- 1962: The Librarian
- 1972: SCCS (Source Code Control System)
- 1982: RCS (Revision Control System)
- 1990: CVS (Concurrent Version System)
- 2000: SVN (Apache Subversion)
- 2005: GIT

1. Histoire des VCS

VCS Local



1. Histoire des VCS

VCS Local

- On stocke les versions du fichiers dans une base de données en local
- Path Sets (on ne garde que les modifications entre versions)

1. Histoire des VCS

VCS Local The Librarian et SCSS

- Développé par IBM
- SCCS est considéré comme le premier vrai VCS

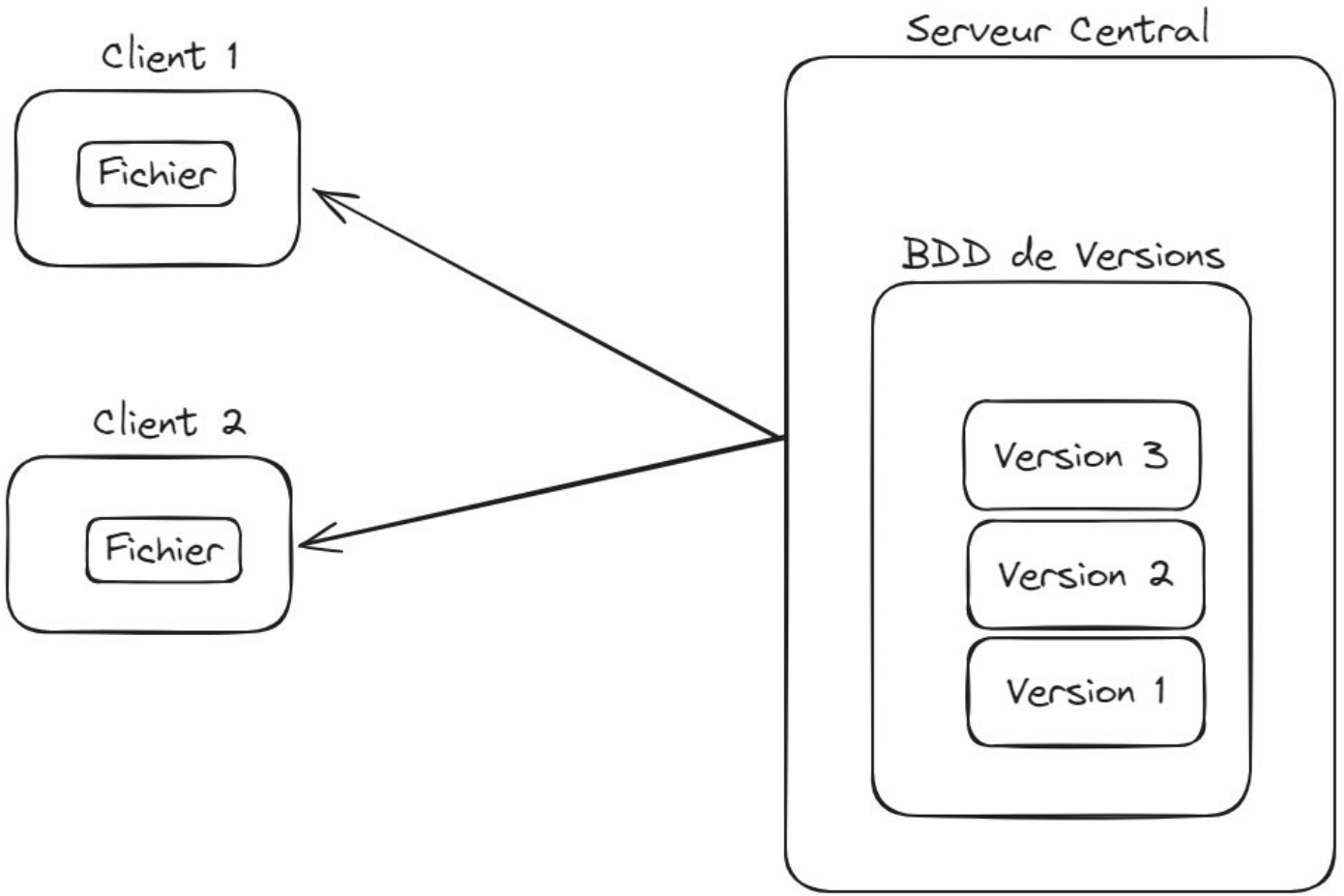
1. Histoire des VCS

VCS Local RCS

- Open Source (Projet GNU)
- Née comme alternative à SCCS

1. Histoire des VCS

VCS Centralisé



1. Histoire des VCS

VCS Centralisé

- La BDD (Base de Données) des versions est stocké dans un serveur central
- Utile quand plusieurs développeurs veulent travailler ensemble

1. Histoire des VCS

VCS Centralisé — Avantages

- Contrôle d’accès
- Gestion de tâches
- Synchronisation entre plusieurs ordinateurs simplifié

1. Histoire des VCS

VCS Centralisé — Inconvénients

- Single Point of Failure
  - Si le serveur tombe, personne n’a accès au projet
  - Si le disque dur tombe en panne, le projet est perdu à jamais

1. Histoire des VCS

VCS Centralisé CVS

- Introduit le concept de dépôt distant

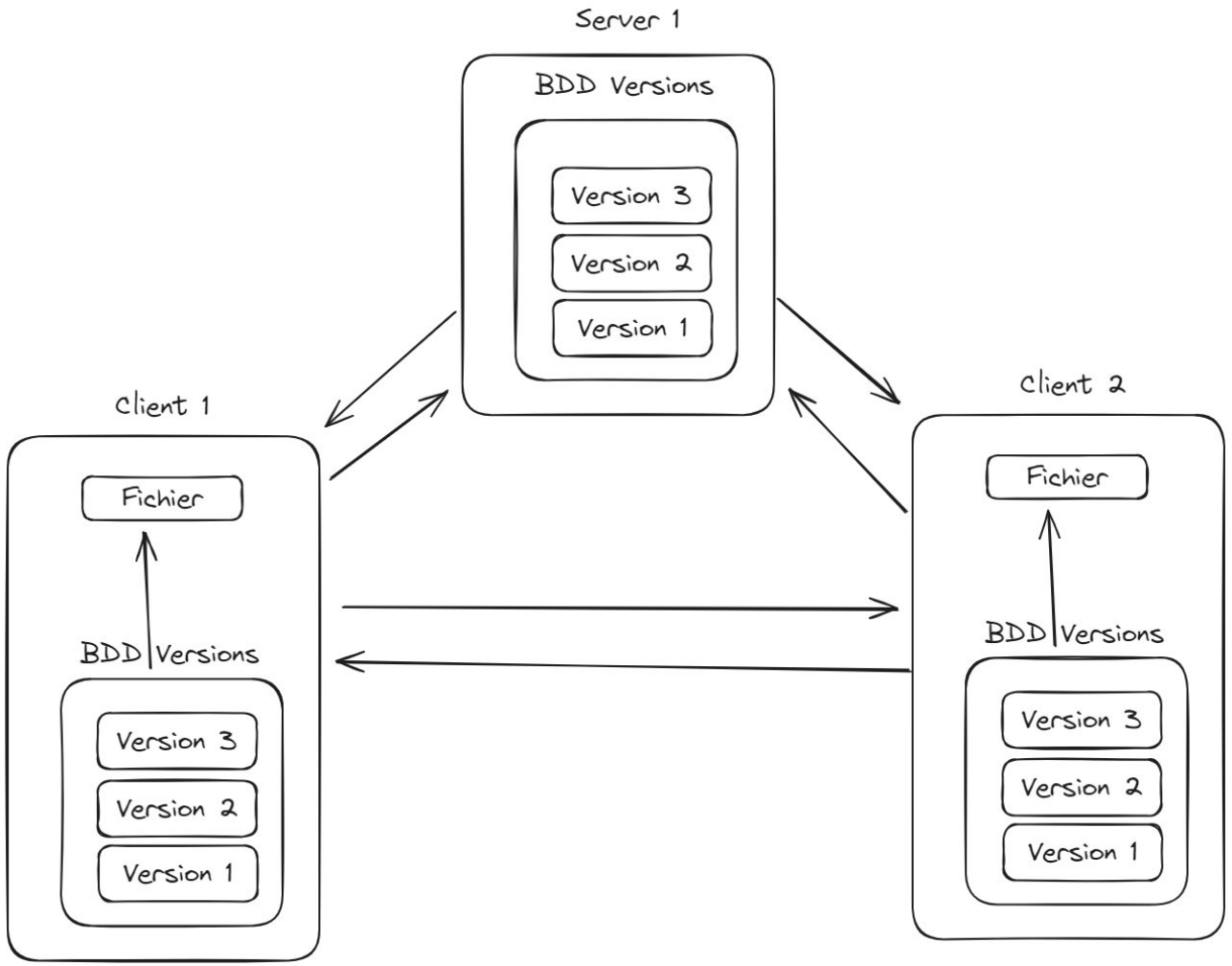
1. Histoire des VCS

VCS Centralisé SVN

- Alternative Open Source à CVS

1. Histoire des VCS

VCS Distribué



1. Histoire des VCS

VCS Distribué

- Chaque copie (clone) contient tout le projet et son historique
- Chaque copie devient un backup
  - Plus de Single Point of Failure
- Il faut quand-même faire des vraies sauvegardes si le projet est important

1. Histoire des VCS

Résumé des VCS

VCS Local	VCS Centralisé	VCS Distribué
The Librarian	CVS	Git
SCCS	SVN	Mercurial
RCS		BitKeeper

1. Histoire des VCS

2. GIT

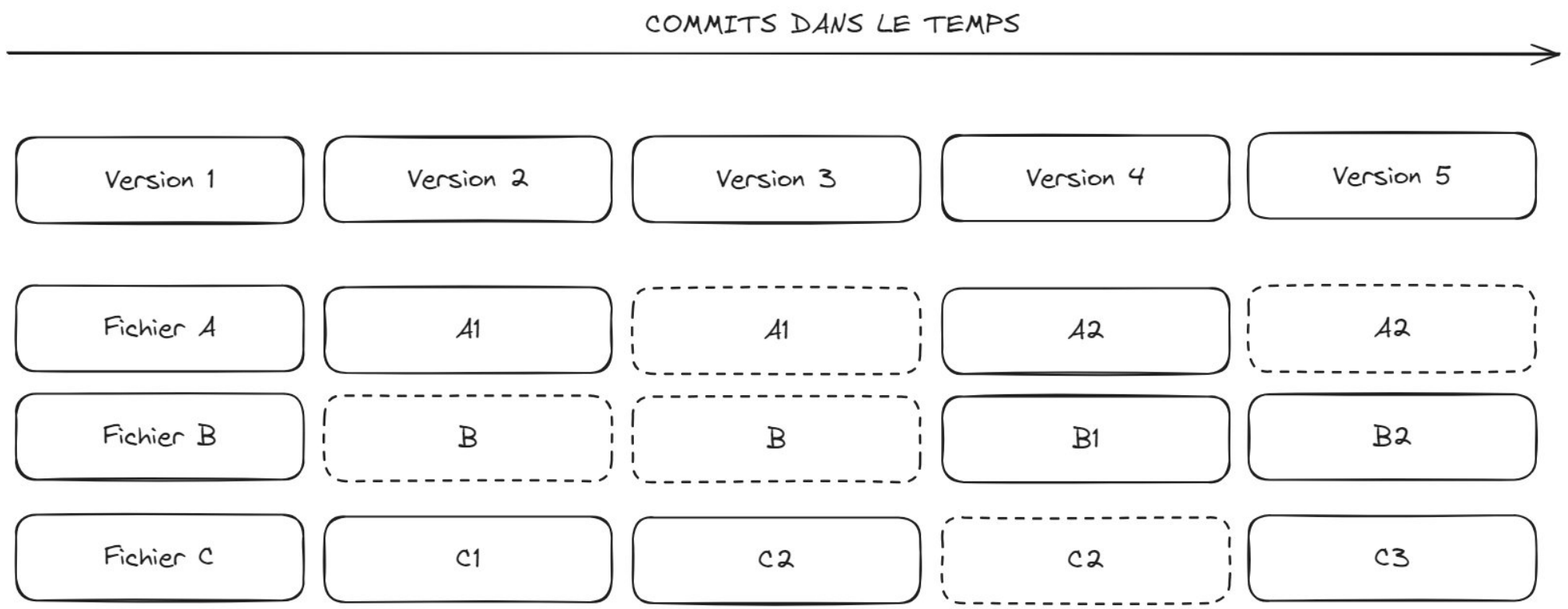
2. GIT

GIT

- Crée en 2005 par Linus Torvalds
- Pour le développement du noyau Linux
- VCS le plus connu aujourd’hui

2. GIT

Sauvegarde de fichiers



2. GIT

Etats des Fichiers

- Les fichiers peuvent être « modifié », « staged » ou « commited »
- Un fichier ne peut être « staged » que s'il a été « modifié »
- Un fichier ne peut être « commited » que s'il a été « staged »
- Donc seul les fichiers « commited » sont sauvegardés

2. GIT

Sauvegarde de fichiers

- Les données sont représentées par des snapshots (état du projet à un instant)
  - Si le fichier n’a pas changé depuis le dernier « commit » (sauvegarde), git stocke une référence vers la dernière modification du fichier
  - Si le fichier a changé, git prend un nouveau snapshot du fichier

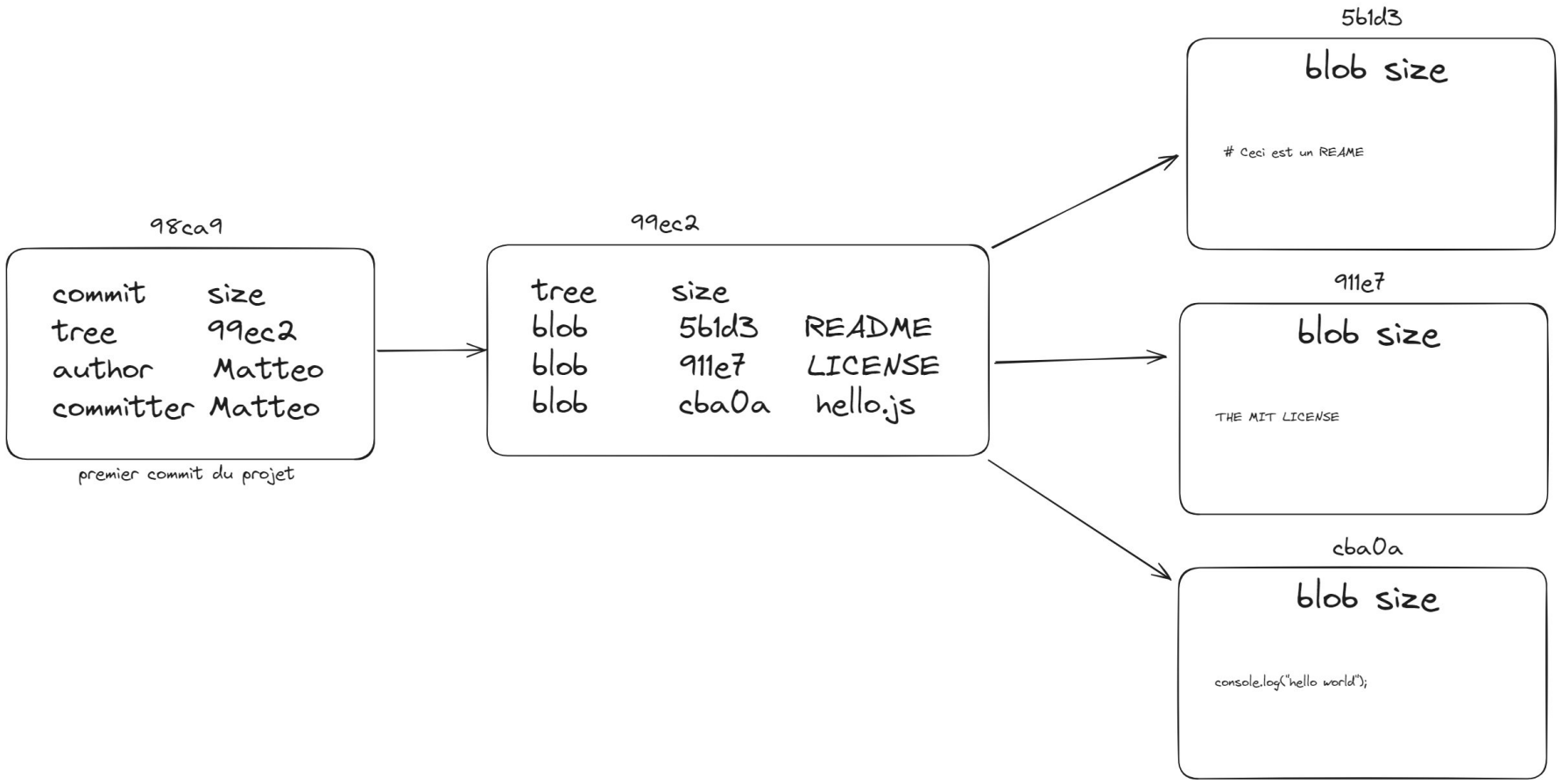
2. GIT

Commit

- Un commit est un snapshot
- Identifié par un hash SHA1 des informations du commit
- Il contient
  - Une référence au commit précédant s'il existe
  - Un message (de préférence compréhensible par des humains)
  - Un auteur et une date d'édition
  - Un « commiter » (la personne qui crée le commit) et la date de commit
  - Les fichiers « staged » qu'on veut rajouter au commit

2. GIT

Stockage D’information



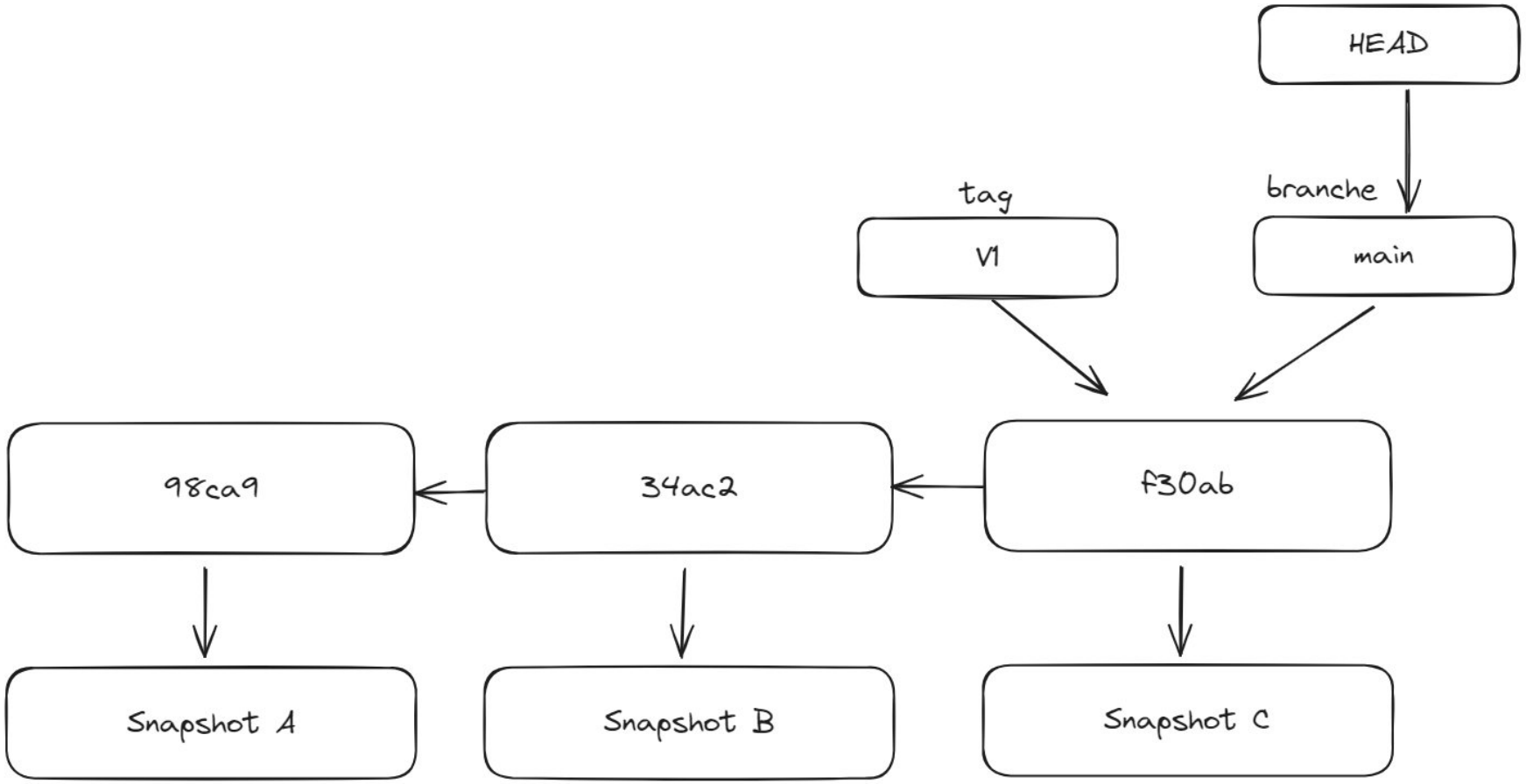
2. GIT

Stockage D’information

- Un fichier est stocké comme Blob (Binary Large Object)
- Un dossier est stocké comme Tree
  - Contient la liste des blob du dossier
  - Contient la liste des trees des sous-dossiers

2. GIT

Références



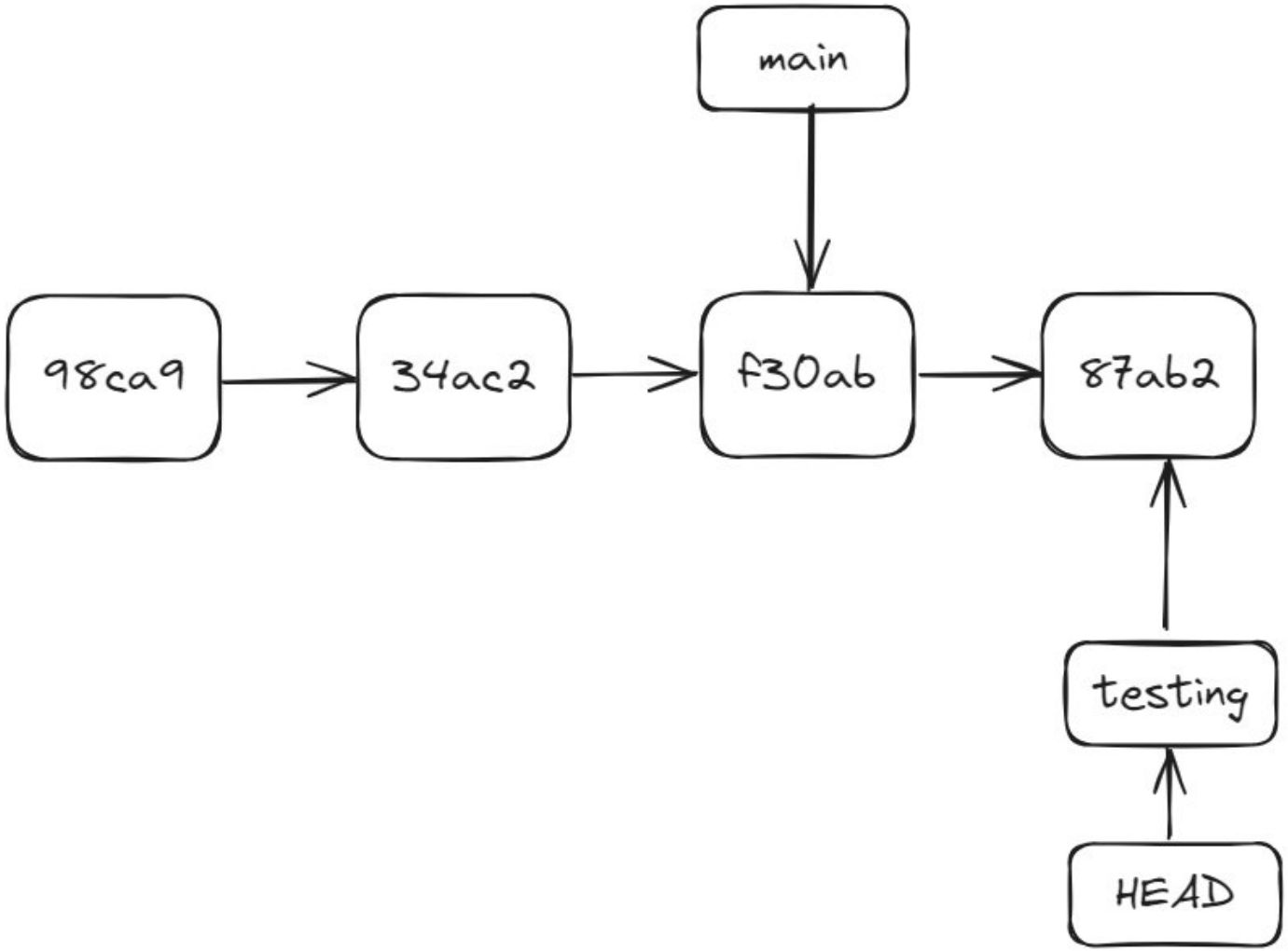
2. GIT

Références

- Compliqué de se souvenir d'un hash
- Git peut avoir des labels compréhensibles
  - Branches: références movibles (peuvent changer de destinataire)
  - Tags: immutables (pointent toujours vers un seul commit)
  - HEAD: pointe **généralement** vers le dernier commit de la branche
- La création d'un commit bouge le HEAD vers ce dernier

2. GIT

Branches



2. GIT

Branches

- Façon de travailler en parallèle sur un projet
- Quand on change de branche, le HEAD se déplace vers le dernier commit de la branche
- Conventionnellement la branche principale s’appelle « master » ou « main »
- Si plusieurs branches pointent vers un même commit et qu’on crée un nouveau commit dans une branche, les branches « divergent »

2. GIT

Merge

- Sert à rapatrier les commits d'une branche avec celle dont elle a divergé
- Différentes méthodes:
  - Merge
  - Cherry pick
  - Rebase (méthode par default)
- Github appelle ça des « Pull Requests »

2. GIT

Merge Conflict

- Apparait quand deux commits de deux branches modifient le même fichier
- Il est conseillé de ne pas travailler dans le même fichier sur deux branches différentes

2. GIT

Remotes

- Git est un VCS distribué
- Un « remote » est un dépôt distant pour un projet

2. GIT

Glossaire

- Commit: image (snapshot) du projet à un instant T
- Origin: dépôt distant
- Dépôt: projet / dossier du projet
- Pull: téléchargement de commits du dépôt distant
- Push: envoie de commits vers un dépôt distant
- Branche: version d’une arborescence de commits

2. GIT

Commandes de base

- **git init** : initialise un projet git dans un dossier
- **git status** : donne un sommaire du repository git: fichier modifié, staged et non suivi
- **git diff** : liste les différences entre l'état courant du dossier, le HEAD, un commit
- **git log** : liste les commits crée dans le temps
- **git blame** : montre le dernier commit qui a changé chaque ligne du fichier sélectionné (utile pour savoir qui fait planter une application)
- **git stash** : mets de côtés les changements ne faisant pas partie d'un commit, qu'ils soient "staged" ou "suivis"
- **git pull** : récupère les commits d'un remote et les applique en local
- **git push** : envoie les commits vers un remote

2. GIT

Commandes pour Commits

- **git add** : prépare des fichiers pour le commit (ils deviennent "staged")
- git commit : crée le commit avec les fichiers "staged" (donc une fois le commit crée, il n'y aura plus de fichiers staged)

2. GIT

Commandes pour Branches

- **git branch** : liste les branches du repo
- **git checkout** : se déplacer sur une branche (« git checkout –b » pour créer une nouvelle branche)

2. GIT

3. Github

3. Github

Github

- Utilise Git
- Peut servir de dépôt distant (Git est un VCS distribué)
- Très utilisé pour le développement open source

3. Github

Github - Fonctionnalités

- Gestion d’accès (dépôt privé/public, gestion contributeurs)
- Wiki par dépôt
- Système de suivi de bugs (issues)
- Pages web par dépôt (Github Pages)
- Intégration Continue
- Distribution d’application
- Gestion de versions (avec les tags de Git)

Ressources

1. Pro Git
2. Git Visual Reference (plus très à jour, mais des notions intéressantes)
3. How git cherry-pick and revert use 3-way merge
4. Git Overview - Computerphile

3. Github