The background of the slide is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes. Some droplets are large and prominent, while others are small and subtle. They are scattered across the slide, with a higher concentration in the top-left and bottom-right corners, and a few smaller ones near the center text.


INTÉGRATION CONTINUE (JAVA & JENKINS)

EPSI PARIS 2021

STEEVE PARDIN



SOMMAIRE

- LE MONDE D'INTÉGRATION CONTINUE
 - GESTION DES SOURCES
 - LES TESTS
 - AUTOMATISATION DES TÂCHES
 - LE SERVEUR D'INTÉGRATION CONTINUE
 - LA MISE EN PLACE DES MÉTRIQUES (KPI)
- 

GESTION DES SOURCES

DÉFINITION

- LA GESTION DES SOURCES EST UNE PRATIQUE D'INGÉNIERIE DONT L'ART EST DE GÉRER LES MODIFICATIONS D'INFORMATIONS
- APPLICABLES AU-DELÀ DE LA GESTION DES VERSION DE CODE D'UN LOGICIEL.
- ELLE CONCERNE:
 - GESTION DES FICHIERS D'UN SITE WEB
 - GESTION DE L'HISTORIQUE DE /ETC POUR LES INGÉNIEURS SYSTÈME UNIX
 - DOCUMENTATIONS, SUPPORTS DE COURS, ARTICLES
 - GESTION DES CONFIGURATIONS

HISTORIQUE

- A L 'ORIGINE (70-80)
 - GESTION MANUELLE DES ARTIFACTS
 - DÉMARCHE COMPLEXE, FASTIDIEUSE ET SOUVENT SOURCES D'ERREURS
 - NÉCESSITÉ D'UN LOGICIEL GÉRANT LE RÉFÉRENTIEL DES MODIFICATIONS
- LOGICIEL DE GESTION DE VERSIONS
 - ASSURE LE STOCKAGE CHRONOLOGIQUE DES FICHIERS
 - CONSERVE L'INTÉGRALITÉ DES VERSIONS D'UN FICHIER OU D'UN RÉPERTOIRE
 - RETROUVE LES RÉVISIONS D'UNE RESSOURCE
 - FICHIER
 - IMAGE
 - ...

PRINCIPE DE BASE

- LOGICIEL DE GESTION DE VERSIONS EST GÉNÉRALEMENT CONSTITUÉ D'UN DÉPÔT (LOCAL, DISTANT):
 - CONTENANT TOUTES LES VERSIONS,
 - DE COPIES DE TRAVAIL
 - CONTENANT LES MODIFICATIONS D'UN UTILISATEUR QUI SERONT ENSUITE INCLUSES DANS LE DÉPÔT.
- PERMETTRE UN TRAVAIL COLLABORATIF OU COOPÉRATIF DANS LES DOMAINES SUIVANTS
 - DÉVELOPPEMENT DE CODE SOURCE
 - OU L'ÉCRITURE DE DOCUMENTS
- DONNER UN DROIT À L'ERREUR EN PERMETTANT DE REVENIR À DES VERSIONS ANTÉRIEURES

ARCHITECTURE

- DISPOSER DE 2 (OU PLUSIEURS) VERSION DE TRAVAIL (BRANCHES)
 - CORRECTION DES BUGS EN PARALLÈLE ET DÉVELOPPEMENTS DE NOUVELLES FONCTIONNALITÉS
- DISPOSER D'UN (OU PLUSIEURS) DÉPÔT(S) DE RÉFÉRENCE
- DÉPÔT LOCAL OU DISTANT RÉPERTORIANANT TOUT L'ENSEMBLE DES MODIFICATIONS
- L'ARBORESCENCE EST CONSTRUITE SUR LA BASE DE :
 - BRANCHES
 - TAGS

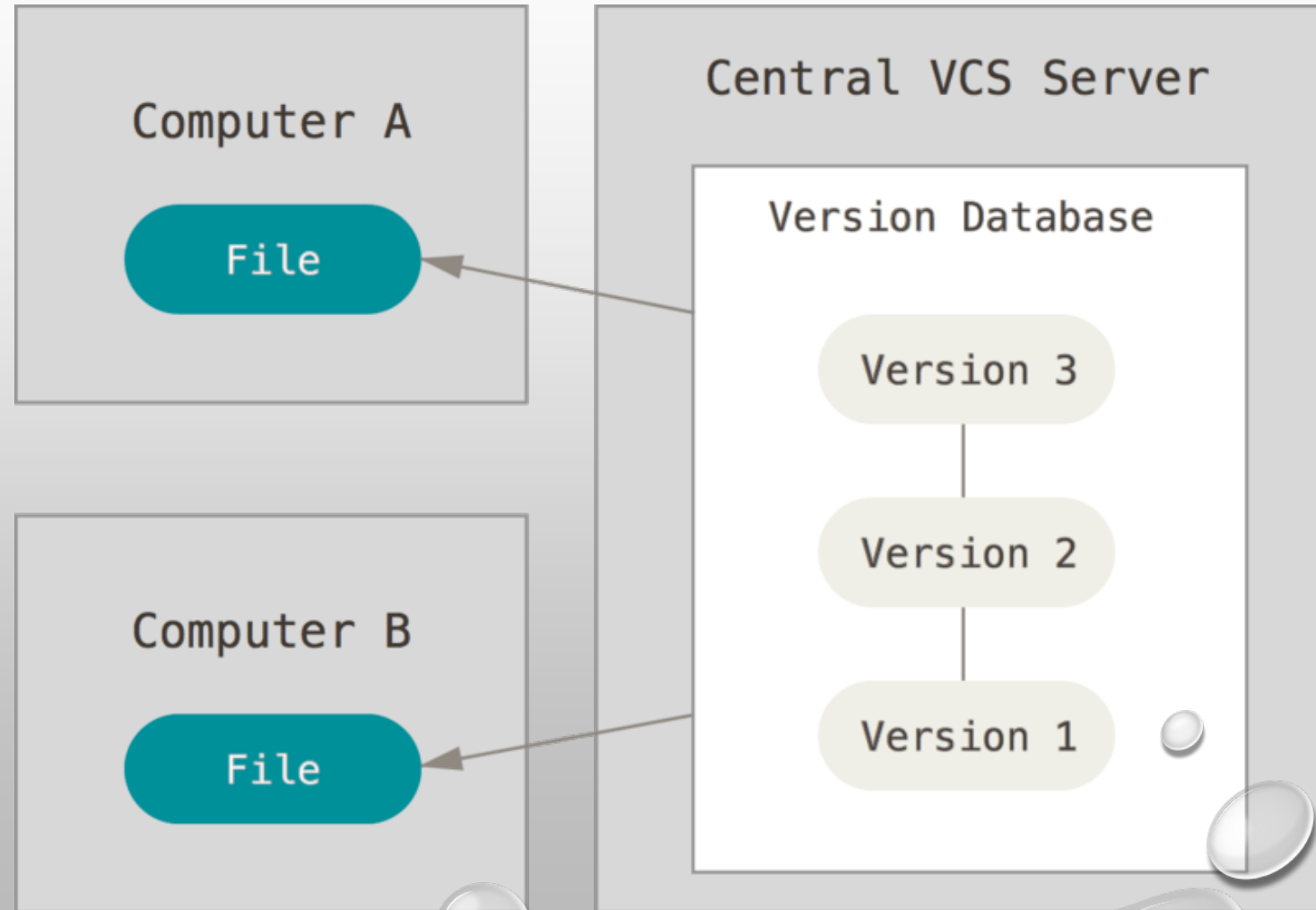


ARCHITECTURE

- LES BRANCHES SERVENT À
 - CORRIGER UN PROBLÈME SUR UNE ANCIENNE VERSION,
 - DÉVELOPPER 2 IDÉES EN PARALLÈLE, GÉRER SA PROPRE VERSION DU LOGICIEL, FUSIONNER APRÈS UNE DIVERGENCE
- LES TAGS SONT DES MARQUES SYMBOLIQUES
 - IL PERMETTENT DE DÉFINIR LES VERSIONS DU PROJET
 - ILS PERMETTENT DE MARQUER L'ÉTAT DES ARTIFACTS DU PROJET

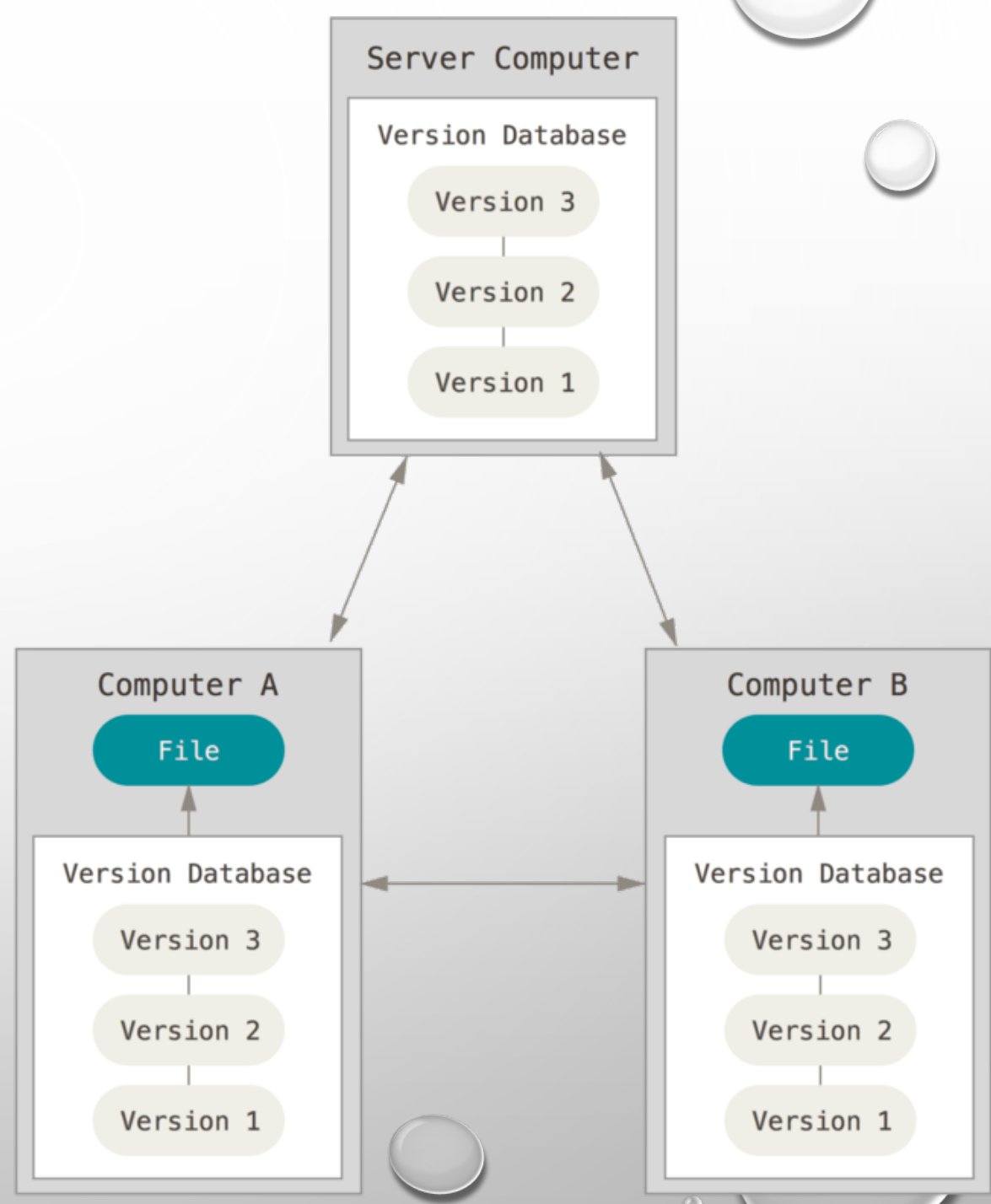
LES DIFFÉRENTES FAMILLES D'OUTILS

- VERSION CENTRALISÉE
 - FAMILLES D'OUTILS DE GESTION DE VERSION
 - UN SEUL DÉPÔT OU REPOSITORY CONTENANT TOUTES LES VERSIONS
 - SEULS DES PRIVILÉGIÉS DÉCLARÉS ONT LE DROIT DE « VALIDER » LES MODIFICATIONS PROPOSÉES
- EXEMPLE : CVS, SVN



LES DIFFÉRENTES FAMILLES D'OUTILS

- VERSION DÉCENTRALISÉE
 - PLUSIEURS DÉPÔTS COEXISTENT
 - NÉCESSITÉ DE SYNCHRONISER
 - NÉCESSITÉ POUR LES UTILISATEURS DE DÉFINIR DES RÈGLES DE FONCTIONNEMENT
- EXEMPLE : GIT, MERCURIAL, DARCS SVK



GIT CHEAT SHEET

presented by TOWER > Version control with Git - made easy



COMMANDES GIT

CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repo.git
```

Create a new local repository

```
$ git init
```

LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

Don't amend published commits!

```
$ git commit --amend
```

COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```

MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

Don't rebase published commits!

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
```

```
$ git rm <resolved-file>
```

UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit

...and discard all changes since then

```
$ git reset --hard <commit>
```

...and preserve all changes as unstaged changes

```
$ git reset <commit>
```

...and preserve uncommitted local changes

```
$ git reset --keep <commit>
```

LES OUTIL WEB DE GIT

- **GITHUB**

- GITHUB EST UN SERVICE POUR HÉBERGER DES DÉPÔTS ET GIT EST L'OUTIL QUI NOUS PERMET DE CRÉER UN DÉPÔT LOCAL ET DE GÉRER LES VERSIONS.
- GIT EST DIFFÉRENT DE GITHUB: GITHUB EST UNE ENTREPRISE QUI VOUS PERMET D'HÉBERGER VOS REPOSITORIES ET COLLABORER AVEC D'AUTRES PERSONNES, VOUS POUVEZ CONSIDÉRER GITHUB COMME UN RÉSEAU SOCIAL POUR DÉVELOPPEUR, IL PERMET AUSSI AUX DÉVELOPPEURS DE PARTAGER DES CODES SOURCES ET DES PROGRAMMES.

LES OUTIL WEB DE GIT

- **GITLAB**

- PREMIÈREMENT, ET SURTOUT, GITLAB PROPOSE UNE INTERFACE WEB COMPLÈTE ET ÉPURÉE. TOUTE SOLUTION D'HÉBERGEMENT WEB DE PROJETS GIT PERMET DE VISUALISER SES DIFFÉRENTS PROJETS, L'ÉTAT ET L'ÉVOLUTION DES BRANCHES ET L'HISTORIQUE DU PROJET, CHOSE QUI PEUT ÉGALEMENT ÊTRE FAITE AVEC UN GRAND NOMBRE D'OUTILS DE VISUALISATION DE REPOSITORIES GIT LOCAUX TELQUE GITK.

LES LIMITES DE GIT

- MALGRÉ QUE GIT SOIT UN OUTIL PUISSANT ET INCONTOURNABLE POUR LE TRAVAIL COLLABORATIF, ON PEUT LUI TROUVER QUELQUES PETITS DÉFAUTS, NOTAMMENT SUR WINDOWS OU GIT DOIT ÊTRE UTILISÉ À L'AIDE DE CYGWIN ET MSYSGIT ET RESTE LÉGÈREMENT PLUS LENT.
- L'ENCODAGE DES TEXTES PEUT ÊTRE CAPRICIEUX NOTAMMENT AVEC LES SAUTS DE LIGNE.
- LA COMPLEXITÉ D'UTILISATION ET D'APPRENTISSAGE RESTE ENCORE UN PETIT DÉFAUT MAIS GIT VIENT PALIER À CELA EN PROPOSANT UNE PETITE FORMATION EN LIGNE DE COMMANDE DIRECTEMENT SUR LEUR SITE

FONCTIONNEMENT DE GIT

Lorsque vous initialisez un projet avec Git ce dernier crée un répertoire .git, c'est ici que Git stocke et manipules les données de votre projet.



4 TYPES D'OBJETS UTILISÉS PAR GIT

Git utilise principalement 4 types d'objets qui vont permettre de référencer tout votre projet



4 TYPES D'OBJETS UTILISÉS PAR GIT

Git utilise principalement 4 types d'objets qui vont permettre de référencer tout votre projet



A diagram of a Blob object. It consists of a red rounded rectangle. Inside the rectangle, at the top, is a set of three binary strings: "1010", "0001", and "1100", each enclosed in its own curly braces and stacked vertically. Below these strings, the word "BLOB" is written in a bold, black, sans-serif font.

BLOB

Blob
(Binary Large Object)

L'objet Blob est utilisé pour
stocker le contenu d'un
fichier seul.

4 TYPES D'OBJETS UTILISÉS PAR GIT

Git utilise principalement 4 types d'objets qui vont permettre de référencer tout votre projet

A red square containing a black icon of a curly brace with three lines of binary code inside: 1010, 0001, and 1100.

BLOB

Blob
(Binary Large Object)

L'objet Blob est utilisé pour stocker le contenu d'un fichier seul.



Tree

Il contient la référence aux autres blobs et / ou aux autres arbres (tree)

4 TYPES D'OBJETS UTILISÉS PAR GIT

Git utilise principalement 4 types d'objets qui vont permettre de référencer tout votre projet



BLOB

Blob
(Binary Large Object)

L'objet Blob est utilisé pour stocker le contenu d'un fichier seul.



TREE

Tree

Il contient la référence aux autres blobs et / ou aux autres arbres (tree)



COMMIT

Commit

L'objet commit contient la référence à un objet arbre ainsi que d'autres informations complémentaires (auteur, commiter, date, etc.)

4 TYPES D'OBJETS UTILISÉS PAR GIT

Git utilise principalement 4 types d'objets qui vont permettre de référencer tout votre projet



BLOB

Blob
(Binary Large Object)

L'objet Blob est utilisé pour stocker le contenu d'un fichier seul.



TREE

Tree

Il contient la référence aux autres blobs et/ ou aux autres arbres (tree)



COMMIT

Commit

L'objet commit contient la référence à un objet arbre ainsi que d'autres informations complémentaires (auteur, commiter, date, etc.)



TAG

Tag

L'objet tag est simplement une autre référence à un commit et existe pour rendre le référencement plus facile

ORGANISATION D'UN PROJET GIT

Lorsque vous faites un «add» d'un fichier sur votre projet, un objet **Blob** est créé. Ce blob contient le contenu complet de votre fichier à un instant «T».



```
{1010}  
0001  
1100}
```

BLOB_1

ORGANISATION D'UN PROJET GIT

Lorsque vous faites un commit de ce fichier, cela crée 2 objets supplémentaires, un objet **Tree** et un objet **Commit**



COMMIT



TREE_1

$\begin{Bmatrix} 1010 \\ 0001 \\ 1100 \end{Bmatrix}$

BLOB_1

ORGANISATION D'UN PROJET GIT

A cette étape nous avons un objet **Commit** qui contient la référence à un objet **Tree** qui lui même référence le **Blob** que nous avons crée lors de l'ajout.



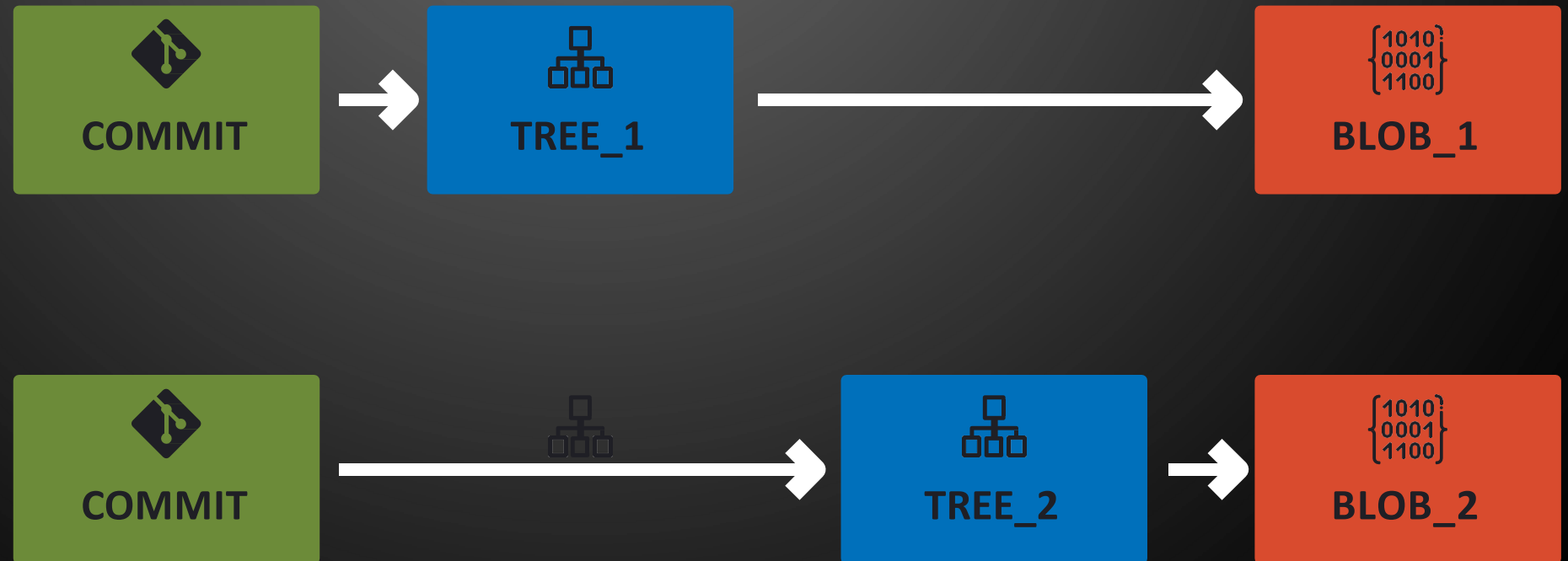
ORGANISATION D'UN PROJET GIT

Lorsque j'ajoute et «commit» un autre fichier, alors un nouveau blob est créé.



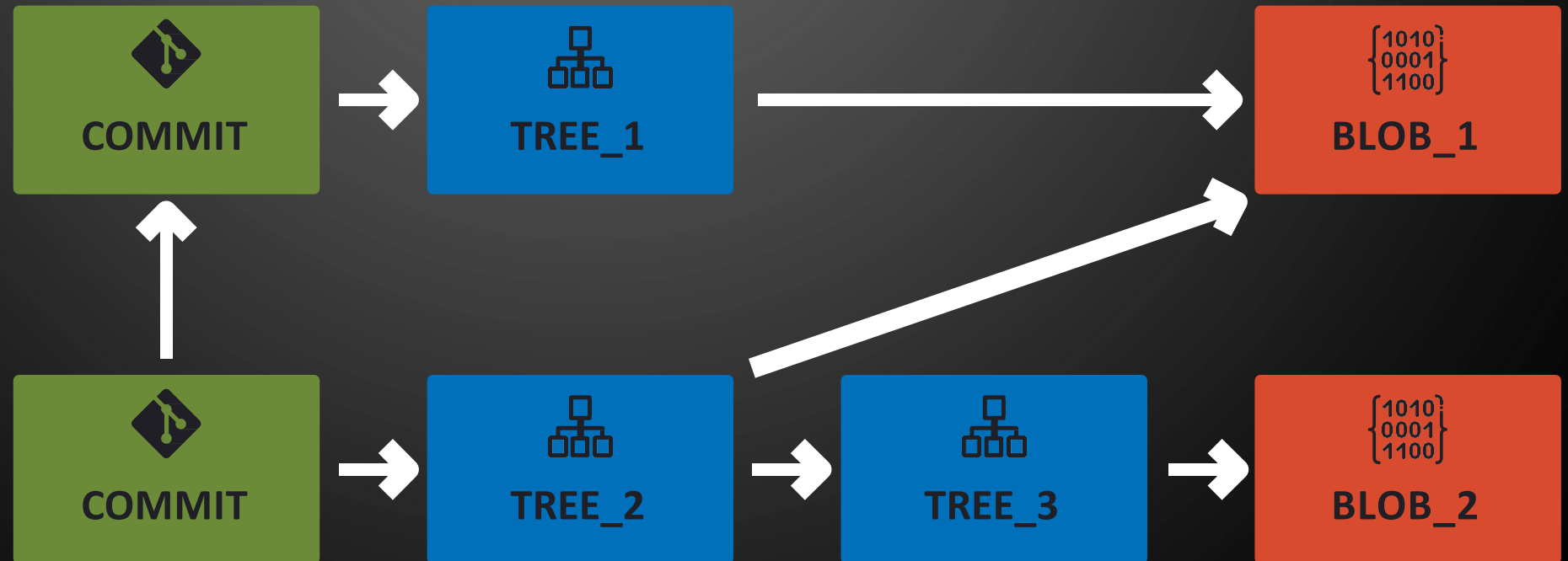
ORGANISATION D'UN PROJET GIT

De facto un nouvel objet **Tree** ainsi qu'un nouvel objet **Commit** sont créés, qui référencent ce nouveau **Blob** ...



ORGANISATION D'UN PROJET GIT

... mais en réalité cet objet **Tree** contiendra aussi la référence au premier **Blob**, et le nouvel objet **Commit** contiendra la référence à son parent



ORGANISATION D'UN PROJET GIT

L'objet Tag fait seulement référence à un Commit spécifique pour retrouver le projet à une étape donnée. Mais il est intéressant de noter que les branches fonctionnent exactement de la même façon.

