

Introduction à Git

Table des matières

I. Contexte	3
II. Les systèmes de gestion de versions	3
III. Exercice : Appliquez la notion	7
IV. Les avantages de Git	7
V. Exercice : Appliquez la notion	10
VI. Installation et configuration	10
VII. Exercice : Appliquez la notion	12
VIII. Premier projet Git	12
IX. Exercice : Appliquez la notion	14
X. Essentiel	14
XI. Auto-évaluation	14
A. Exercice final	14
B. Exercice : Défi.....	15
Solutions des exercices	16

I. Contexte

Durée : 1 h

Environnement de travail : GitBash / Terminal

Pré-requis : Aucun

Contexte

En tant que développeur web, nous aurons besoin de gérer un code informatique plus ou moins conséquent, composé de fichiers texte, d'images, etc. Nous allons très rapidement avoir envie de pouvoir revenir en arrière sur l'historique de notre projet, avoir une sauvegarde distante de nos développements, et même travailler de façon collaborative avec d'autres développeurs, souvent en parallèle.

Pour effectuer toutes ces tâches, nous aurons besoin d'adopter un outil que l'on appelle un **système de gestion de versions** (*version control system* ou **VCS** en anglais). Il existe plusieurs de différents types, mais nous nous focaliserons ici sur l'utilisation de Git, le VCS le plus utilisé dans le monde.

II. Les systèmes de gestion de versions

Objectif

- Faire la différence entre les différents types de VCS

Mise en situation

Les VCS sont indispensables dans le développement informatique et vous les utiliserez au quotidien. Pour faire simple, ils permettent de stocker l'historique des modifications d'un fichier ou d'un ensemble de fichiers afin de pouvoir être interrogés et récupérer certaines informations. Il existe plusieurs familles de VCS, que nous allons voir ici.

Les VCS locaux

Pour garder une copie d'un projet, la méthode courante consiste à créer une copie des fichiers de ce projet, en changeant le nom (par exemple en ajoutant un numéro de version) ou bien en l'archivant dans un ZIP.

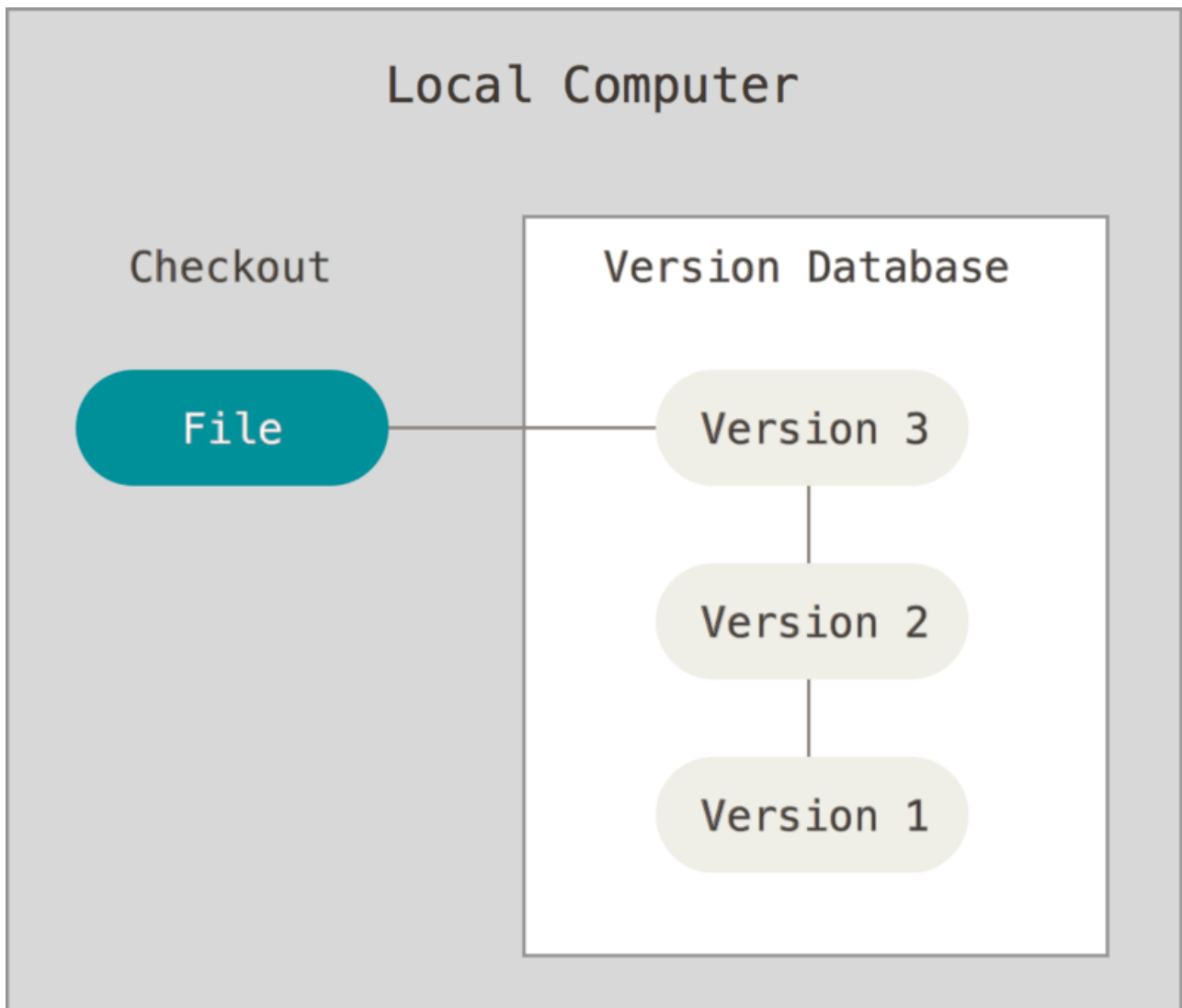
Mais cette méthode ne se révèle pas très performante à l'usage, en particulier pour :

- Garder des copies très régulières (chaque jour, par exemple),
- Travailler à plusieurs,
- Travailler en parallèle sur plusieurs morceaux du projet,
- Etc.

Le **VCS local** permet de gérer l'historique des versions de tous les fichiers du projet sur la machine locale.

Pour chaque fichier, le VCS local gère :

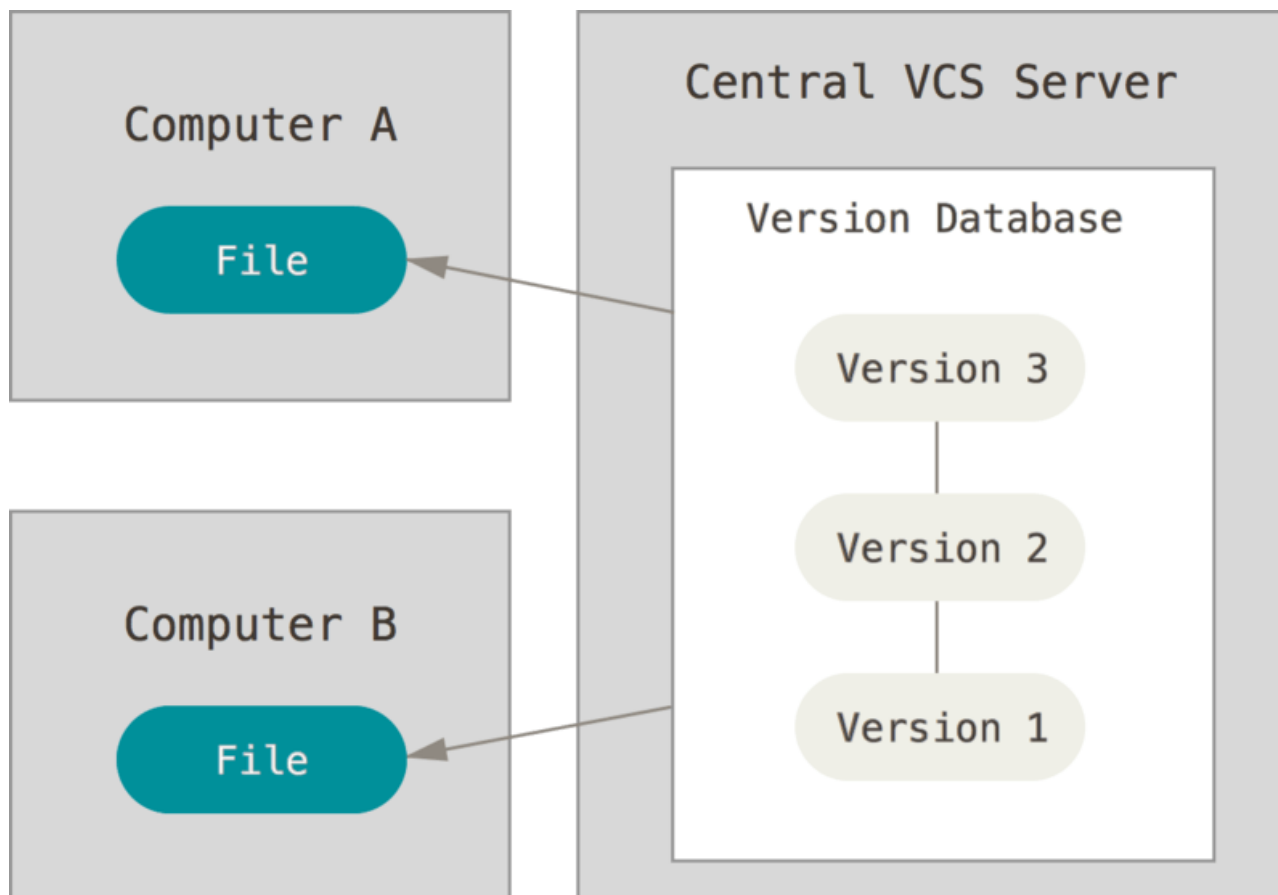
- Le fichier courant (celui qui est en cours de modification),
- Une base de données contenant l'ensemble des versions antérieures de ce fichier.



Ce VCS nous permet d'avoir un historique local de nos modifications, mais il ne permet pas de partager notre travail ou de collaborer avec d'autres personnes.

Les VCS centralisés

Dans la configuration des **VCS centralisés** (ou CVCS pour *Centralized Version Control System*), la base de données est hébergée sur un serveur distant. C'est le cas pour des CVCS tels que CVS ou Subversion (SVN), par exemple.

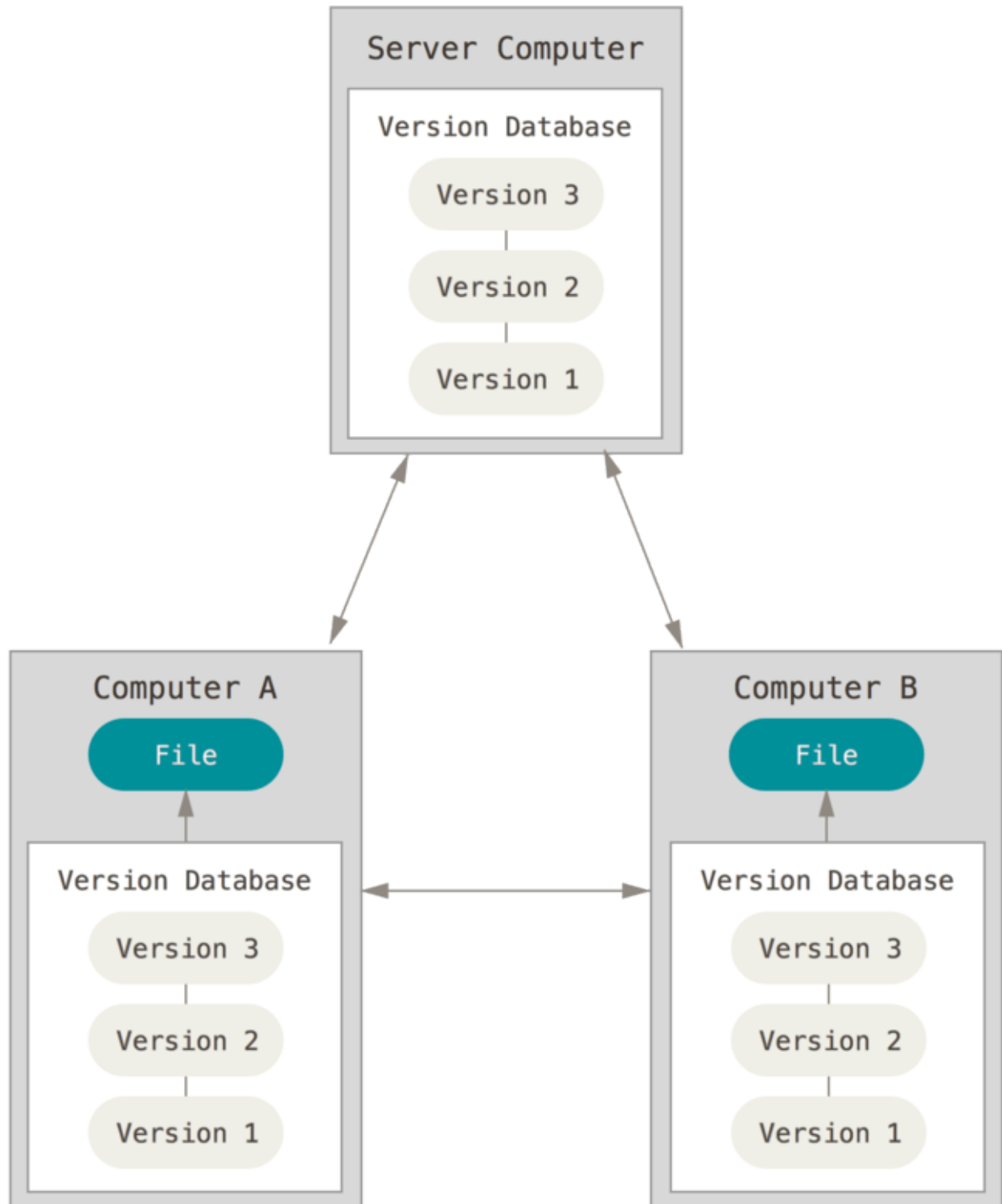


L'avantage de ce système est que nous ne possédons qu'une seule source de données : l'historique de nos modifications. Tous les ordinateurs y sont connectés et nous sommes sûrs qu'ils possèdent la dernière version du code.

Le principal inconvénient ici est que nous ne pouvons plus travailler sans avoir accès au serveur contenant notre historique (Central VCS Server). De ce fait, nous ne pouvons plus travailler sans connexion Internet ou lorsque le serveur tombe en panne.

Les VCS décentralisés

Si nous ne souhaitons pas nous baser sur une seule source de données, pour éviter les problèmes de centralisation de la base de données, nous pouvons nous orienter vers un type d'architecture : les **VCS décentralisés** (ou DVCS pour *Distributed Version Control System*). Dans ce cas, la base de données hébergée sur le serveur distant est distribuée sur les ordinateurs aux côtés du code. C'est le cas pour des DVCS tels que Git ou Mercurial, par exemple.



Ce système possède tous les avantages d'un CVCS, tout en garantissant que l'information est redondée et disponible localement.

Cependant, il est aussi plus complexe à administrer, puisque plusieurs sources de données existent : il peut être difficile de savoir laquelle est la plus à jour, et fusionner les changements de différentes bases de données peut s'avérer plus compliqué.

Syntaxe **À retenir**

Un système de gestion de versions, ou VCS, permet de gérer les versions de nos fichiers en permettant d'accéder à leur historique à un instant T, de sauvegarder notre travail sur un serveur distant ou même de collaborer avec d'autres développeurs.

Il existe plusieurs familles de VCS :

- Les VCS locaux (que nous n'utiliserons pratiquement jamais, puisque nous aurons besoin de stocker une copie de notre base de données de versions sur un serveur distant),
- Les CVCS ou CVS centralisés, qui nous permettent de sauvegarder notre base de données sur un serveur distant et de travailler collaborativement,
- Les DVCS, ou CVS distribués, qui nous permettent d'avoir une copie locale de la base de données en cas de corruption du serveur distant et de pouvoir travailler en local sans connexion.

Complément

Démarrage rapide - À propos de la gestion de version¹

Exercice : Appliquez la notion

[solution n°1 p.17]

Exercice

Quel est l'intérêt d'un système de gestion de versions ?

- ☐ Garder une copie de son travail à un instant T
- ☐ Savoir qui a modifié un fichier et quand
- ☐ Récupérer du travail perdu
- ☐ Travailler de façon collaborative

Exercice

Quelles architectures impliquent l'utilisation d'un serveur distant pour stocker la base de données du VCS ?

- ☐ VCS local
- ☐ CVCS
- ☐ DVCS

IV. Les avantages de Git**Objectif**

- Connaître le DVCS Git et ses avantages

Mise en situation

Lorsqu'il s'agit de choisir un VCS, il est important de connaître les avantages et inconvénients de chacun d'entre eux, ainsi qu'au type auquel ils appartiennent. De plus, il sera sans doute utilisé durant toute la vie du projet. C'est pourquoi nous allons voir ensemble les avantages de l'utilisation de Git par rapport aux autres VCS.

1 <https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version>

Un DVCS

Git est un DVCS, pour *Distributed Version Control System*. Cela veut dire qu'il peut être utilisé aussi bien en local qu'avec un serveur distant, puisque sa base de données est décentralisée. Par conséquent, il est très différent des CVCS tels que CVS ou Subversion (SVN).

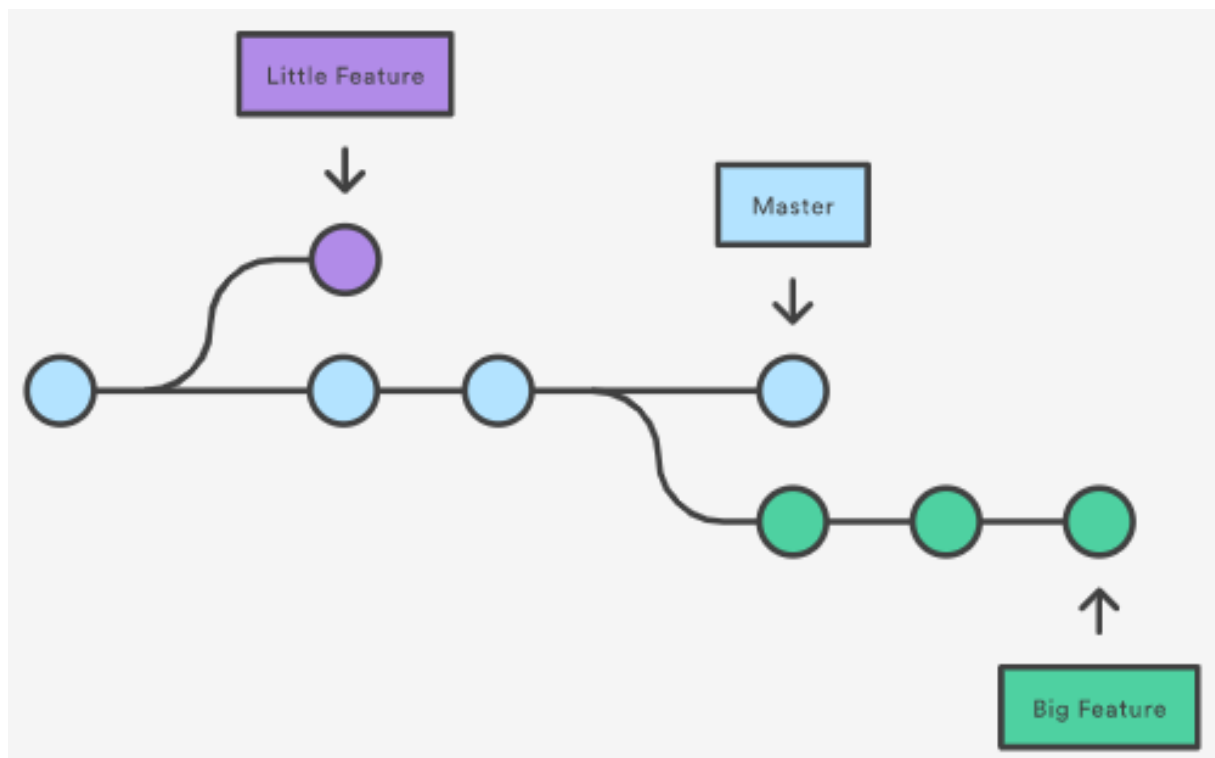
Un outil rapide

Étant donné son architecture distribuée, Git dispose d'une grande rapidité, puisque la majorité des opérations se font en local sur notre machine. C'est-à-dire que nous ne sommes pas tributaire d'une bonne connexion Internet ou d'un serveur centralisé puissant capable de gérer des milliers de requêtes simultanées.

Un des autres systèmes qui rend Git rapide est le fait qu'il travaille avec des instantanés plutôt qu'avec des différences. Là où d'autres VCS stockent la différence d'un fichier entre deux de ses états, Git stocke le fichier en entier lorsqu'il a été modifié, et simplement une référence vers l'état précédent s'il n'a pas changé. Cela le rend très rapide lorsqu'il faut effectuer des recherches dans l'historique ou faire des différences entre deux versions éloignées dans le temps.

Travail en parallèle

Grâce à son système de branches, Git permet à plusieurs développeurs de travailler en simultané sur plusieurs fonctionnalités. La gestion de la fusion de ces branches, le fait d'intégrer une fonctionnalité développée dans un code existant, est largement simplifiée par rapport à d'autres VCS.



Performant pour de grands projets

Git est capable de gérer des projets de grande envergure, tels que le développement du noyau Linux. Celui-ci a initialement été conçu en grande partie par Linus Torvald en 2005 pour gérer ce projet. Il n'a cessé depuis ce jour d'être amélioré et trouve parmi ses utilisateurs des entreprises telles que Google, Facebook ou Microsoft.

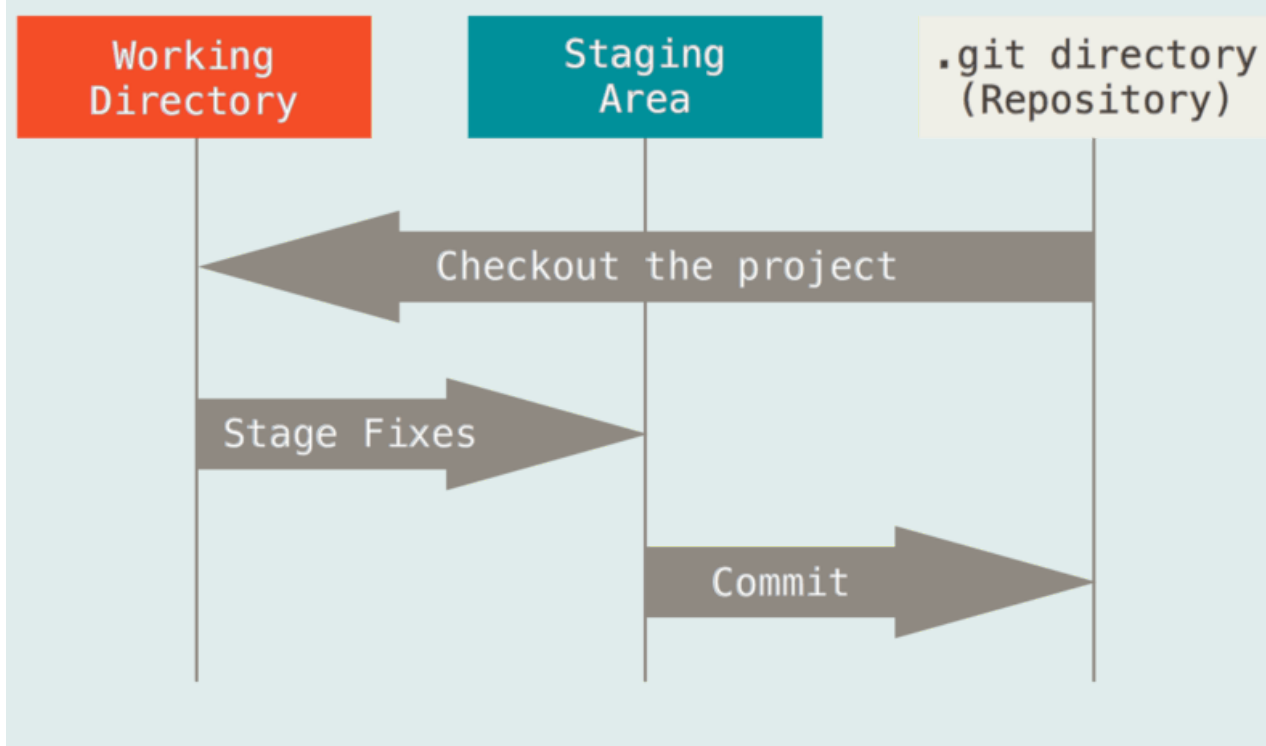
Intégrité des données

En général, Git ne fait qu'ajouter des données. De ce fait, il est assez simple de revenir en arrière lorsque vous pensez avoir cassé quelque chose. Git possède de nombreux garde-fous, qui vous avertiront la plupart du temps lorsque vous êtes sur le point de faire des opérations qui peuvent être dangereuses.

Fondamental Les trois états de fichiers

Git gère les modifications apportées aux fichiers selon trois états : modifié, validé et indexé.

- **Modifié** signifie que le fichier a été modifié en local, mais que ses modifications n'ont pas encore été ajoutées à la base de données locale.
- **Indexé** signifie que la version du fichier a été ajoutée au prochain lot de modifications qui seront intégrées à la base de données locale.
- **Validé** signifie que la version du fichier est présente dans la base de données locale.



Syntaxe À retenir

- Git est un système de gestion de version distribué, développé il y a 15 ans pour le projet du noyau Linux. Il a été depuis continuellement amélioré et convient à tous types de projet, même les plus importants.
- Sa base de données locale et son système de gestion des instantanés le rendent très performant.
- Son système de branches et les trois états de fichiers lui permettent d'être robuste lorsqu'il s'agit de travailler sur un projet, que ce soit seul ou à plusieurs.

Complément

Une rapide histoire de Git¹

Rudiments de Git²

Exercice : Appliquez la notion

[solution n°2 p.17]

Exercice

Sur quelles architectures se base Git ?

- ☐ VCS local
- ☐ CVCS
- ☐ DVCS

Exercice

Git n'est pas très performant pour de très gros projets.

- ☐ Vrai
- ☐ Faux

VI. Installation et configuration

Objectifs

- Installer Git sur son environnement de travail
- Configurer Git pour une première utilisation

Mise en situation

Nous allons voir comment installer Git en local afin de pouvoir l'utiliser sur notre environnement.

Installation

Même s'il existe des outils en interface graphique, Git est à l'origine disponible sous la forme d'une ligne de commande. Il peut être installé sur Windows, Linux ou Mac.

Pour l'installer, il suffit de se rendre sur la page de téléchargement de Git³ et de cliquer sur le lien de téléchargement qui correspond à notre système d'exploitation.

Remarque

Si vous l'avez installé sous Windows, le programme aura ajouté un logiciel du nom **GitBash**. C'est celui que nous utiliserons dans ce cours, puisqu'il ajoute de nombreuses fonctionnalités qui vont nous faciliter la vie.

1 <https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Une-rapide-histoire-de-Git>

2 <https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git>

3 <https://git-scm.com/downloads>

Configuration

Une fois Git installé, que nous l'utilisons avec GitBash, l'invite de commande Windows ou le terminal Linux ou Mac, il est conseillé (même si ce n'est pas indispensable) de procéder à un peu de configuration.

Tout d'abord, ouvrons l'invite de commande de notre choix, puis tapons cette commande pour vérifier que Git est bien disponible et pour afficher sa version :

```
1 git --version
```

Sous Windows, le résultat devrait ressembler à ceci (le résultat ne doit pas être très différent avec les autres invites de commandes) :

```
1 git version 2.26.1.windows.1
```

À partir de là, nous pouvons configurer Git pour qu'il utilise notre nom et notre adresse e-mail (nous remplacerons bien sûr les valeurs des arguments avec les nôtres) :

```
1 git config --global user.name "John Doe"
2 git config --global user.email johndoe@example.com
```

La commande `git config` permet de personnaliser l'installation de Git. Il existe un nombre important de paramètres disponibles, c'est pour cela que nous ne les verrons pas tous ici. Ils sont disponibles sur la documentation de la commande `git config`¹. Cette configuration est appliquée de façon globale grâce à son option `--global` et sera donc disponible pour tous nos projets.

La commande `git config --global --list` permet d'afficher facilement la configuration globale de Git. Si on voulait afficher la configuration de notre projet, il suffirait d'omettre l'option `--global`.

Syntaxe	À retenir
---------	-----------

- Git est disponible sous tous les systèmes d'exploitation. Sous Windows, Git met à notre disposition un outil du nom de GitBash qui nous facilite la vie.
- Git est entièrement personnalisable grâce à la commande `git config`, et son option `--global` permet de définir la configuration une fois pour tous les projets.

Complément

Paramétrage à la première utilisation de Git²

Configuration de Git³

1 <https://git-scm.com/docs/git-config>

2 <https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Param%C3%A9trage-%C3%A0-la-premi%C3%A8re-utilisation-de-Git>

3 <https://git-scm.com/book/fr/v2/Personnalisation-de-Git-Configuration-de-Git>

VII. Exercice : Appliquez la notion

Question

[solution n°3 p.17]

Après avoir procédé à l'installation et la configuration globale de Git, rendez-vous dans la documentation de la commande `git config` et cherchez comment activer l'auto-correction.

Vous devriez ensuite pouvoir taper la commande `git cnofig --global --list` (la faute de frappe est volontaire) et voir Git vous corriger :

```
1$ git config --global --list
2WARNING: You called a Git command named 'cnofig', which does not exist.
3Continuing in 0.1 seconds, assuming that you meant 'config'.
4user.email=dorfpowa@hotmail.com
5user.name=Guillaume Sarra megna
6...
```

Indice :

La configuration s'appelle `help.autocorrect`.

VIII. Premier projet Git

Objectifs

- Initialiser un projet Git
- Apprendre à faire un commit

Mise en situation

Afin de démystifier un peu l'utilisation de cet outil, nous allons voir comment créer notre première version de fichier avec Git.

Un projet Git peut se situer dans n'importe quel répertoire ou sous-répertoire de notre disque. Pour le créer, nous allons donc nous placer dans le répertoire de notre choix en ligne de commande et en créer un qui contiendra notre projet.

Exemple

```
1mkdir exemple-git
2cd $_ # équivaut à "cd exemple-git"
```

Attention Erreur "Filename too long" sous Windows

Git ne peut pas utiliser de chemin de fichier de plus de 4096 caractères, sauf sous Windows, où cette limite est de 260 caractères due au fait que Git utilise une ancienne version de l'API de Windows. Il est donc déconseillé de créer le dossier à une profondeur trop importante mais plutôt proche de la racine de votre lecteur, par exemple dans `C:\workspace\exemple-git` ou `D:\workspace\exemple-git`.

De plus, si vous créez votre dossier dans un répertoire protégé par Windows, tel que `C:\Windows` ou `C:\Program Files`, il se peut que vous rencontriez d'autres problèmes dus aux paramètres de sécurité du système d'exploitation. Évitez donc ces dossiers autant que possible.

Depuis ce répertoire, nous allons initialiser notre projet Git grâce à la commande `git init`.

Exemple

```
1 git init
```

Si on exécute la commande `ls -al`, on s'aperçoit qu'un répertoire `.git` a été créé. Il contiendra notre base de données locale.

Maintenant, créons un fichier `README.md` qui va contenir le contenu d'un fichier **Lisez-moi** :

Exemple

```
1 echo -e "# Fichier README.md\n\nIl ne contient pas grand chose." > README.md
```

Complément

La commande `echo` permet d'afficher le texte passer en paramètre dans la sortie standard, le plus souvent dans la ligne de commande. L'option `-e` indique que les séquences de caractères précédées d'un backslash `\` doivent être interprétées et non affichées telles quelles. `\n` équivaudra donc à un retour à la ligne.

Le caractère `>` signifie que le résultat de la commande, ici l'affichage du texte, sera redirigé vers le fichier `README.md`. Si le fichier n'existe pas, il sera créé, et, s'il existe, le contenu sera écrasé. Si nous avions utilisé `>>`, le contenu aurait été ajouté à la fin du fichier, ce qui permet d'ajouter du contenu avec plusieurs commandes `echo` par exemple.

Ajoutons maintenant le fichier à la liste des fichiers de notre nouvelle version, puis créons notre commit. Un commit correspond à un état de notre projet à un instant T, celui auquel le commit a été créé. On lui donne un nom qui va représenter, sous forme d'un texte humainement lisible, les modifications apportées.

Exemple

```
1 git add README.md
2 git commit -m"Mon premier commit"
```

Résultat :

```
1 [master (root-commit) 0c15bb5] Mon premier commit
2 1 file changed, 3 insertions(+)
3 create mode 100644 README.md
```

Syntaxe **À retenir**

- Pour pouvoir gérer les versions de nos fichiers, nous avons besoin de travailler dans un dépôt Git. On peut en créer un avec la commande `git init`.
- Pour créer une nouvelle version de nos fichiers, nous devons ajouter nos fichiers, puis faire un commit. On peut le faire facilement avec les commandes `git add` et `git commit`.

Remarque

Il est à noter que pour se déplacer vers un dossier contenant un ou plusieurs espaces, il est nécessaire d'écrire le nom du dossier entre guillemets.

Complément

Démarrer un dépôt Git¹

IX. Exercice : Appliquez la notion

Question

[solution n°4 p.17]

À votre tour de créer votre premier commit :

- Créez un répertoire que vous appellerez `exemple-git` et rendez-vous dans ce répertoire
- Initialisez-y un dépôt Git
- Créez un fichier `README.md`, ajoutez-y du contenu
- Ajoutez et commitez ce fichier
- Modifiez le fichier
- Ajoutez de nouveau ce fichier et effectuez un commit

X. Essentiel

XI. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°5 p.18]

Exercice

Il est indispensable d'utiliser un système de gestion de version lorsqu'on fait du développement informatique.

- ☐ Vrai
- ☐ Faux

Exercice

Je peux utiliser un CVCS sans serveur distant.

- ☐ Vrai
- ☐ Faux

Exercice

Quelle architecture implique l'utilisation d'un serveur distant pour stocker la base de données du VCS, ainsi que d'une copie locale ?

- ☐ VCS local
- ☐ CVCS
- ☐ DVCS

Exercice

¹ <https://git-scm.com/book/fr/v2/Les-bases-de-Git-D%C3%A9marrer-un-d%C3%A9p%C3%B4t-Git>

Combien d'états de fichiers Git utilise-t-il pour gérer les modifications de fichiers ?

- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Exercice

Quels sont les avantages à utiliser Git ?

- ☐ Il est très performant, même pour de gros projets
- ☐ Il permet de travailler à plusieurs facilement
- ☐ Il permet de n'avoir qu'une seule et unique base de données, ce qui garantit l'intégrité des données
- ☐ C'est un VCS mature qui a été créé en 2005

Exercice

Quelle commande Git permet de gérer sa configuration ?

Exercice

Quelle option permet d'appliquer une configuration à tous les projets Git ?

Exercice

Quelle commande permet d'initialiser un projet Git ?

Exercice

Quelle commande permet d'ajouter des modifications en prévision de la prochaine version ?

Exercice

Quelle commande permet de créer une nouvelle version ?

B. Exercice : Défi

L'objectif de cet exercice est d'appliquer des modifications basiques à nos fichiers et de créer de nouvelles versions.

Question

[solution n°6 p.19]

- Initialisez un nouveau projet Git intitulé `exercice-git`,
- Créez un fichier `README.md`, puis ajoutez vos modifications et faites un commit avec un message,
- Continuez à modifier votre fichier `README.md` et créez un nouveau commit,
- Ajoutez un nouveau fichier `sources.md`, modifiez ces deux fichiers, puis ajoutez-les et créez un nouveau commit.

Indice :

Vous pouvez ajouter plusieurs fichiers à une nouvelle version en les ajoutant un par un, ou en les spécifiant tous dans la commande `git add fichier1 fichier2`.

Solutions des exercices

Exercice p. 7 Solution n°1**Exercice**

Quel est l'intérêt d'un système de gestion de versions ?

- ☒ Garder une copie de son travail à un instant T
- ☒ Savoir qui a modifié un fichier et quand
- ☒ Récupérer du travail perdu
- ☒ Travailler de façon collaborative

Exercice

Quelles architectures impliquent l'utilisation d'un serveur distant pour stocker la base de données du VCS ?

- ☐ VCS local
- ☒ CVCS
- ☒ DVCS

Exercice p. 10 Solution n°2**Exercice**

Sur quelles architectures se base Git ?

- ☒ VCS local
Si utilisé en local pur, il sert de VCS standard (juste pour sauvegarder le code et en suivre les évolutions, sans le partager).
- ☐ CVCS
- ☒ DVCS
Si utilisé en collaboration (via github ou des dépôts d'entreprise).

Exercice

Git n'est pas très performant pour de très gros projets.

- ☐ Vrai
- ☒ Faux

p. 12 Solution n°3

```
1 git config --global help.autocorrect 1
```

p. 14 Solution n°4

```
1 mkdir exemple-git && cd $_
2
3 git init
4 echo -e "# Fichier README.md\n\nIl ne contient pas grand chose." > README.md
5 git add README.md
6 git commit -m"Mon premier commit"
7
8 # On remplace "choze" par "chose" dans le fichier README.md
9 sed -i "s/choze/chose/" README.md # Sur Linux/Git Bash
10 # sed -i "" -e "s/choze/chose/" README.md # Sur macOS
11 git add README.md
12 git commit -m"Je corrige la faute de frappe"
```


Exercice p. 14 Solution n°5

Exercice

Il est indispensable d'utiliser un système de gestion de version lorsqu'on fait du développement informatique.

☐ Vrai

☒ Faux

 Un VCS n'est pas obligatoire, mais il est fortement conseillé pour tous les avantages qu'il peut apporter (historique des versions, copie distante, travail collaboratif).

Exercice

Je peux utiliser un CVCS sans serveur distant.

☐ Vrai

☒ Faux

Exercice

Quelle architecture implique l'utilisation d'un serveur distant pour stocker la base de données du VCS, ainsi que d'une copie locale ?

☐ VCS local

☐ CVCS

☒ DVCS

Exercice

Combien d'états de fichiers Git utilise-t-il pour gérer les modifications de fichiers ?

☐ 2

☒ 3

☐ 4

☐ 5

Exercice

Quels sont les avantages à utiliser Git ?

- ☒ Il est très performant, même pour de gros projets
- ☒ Il permet de travailler à plusieurs facilement
- ☐ Il permet de n'avoir qu'une seule et unique base de données, ce qui garantit l'intégrité des données
Non, Git, comme tout DVCS, se repose sur des bases de données décentralisées.
- ☒ C'est un VCS mature qui a été créé en 2005

Exercice

Quelle commande Git permet de gérer sa configuration ?

git config

Exercice

Quelle option permet d'appliquer une configuration à tous les projets Git ?

--global

Exercice

Quelle commande permet d'initialiser un projet Git ?

git init

Exercice

Quelle commande permet d'ajouter des modifications en prévision de la prochaine version ?

git add

Exercice

Quelle commande permet de créer une nouvelle version ?

git commit

p. 15 Solution n°6

```
1 mkdir exercice-git && cd $_
2
3 git init
4 echo -e "# Fichier README.md\n" > README.md
5 git add README.md
6 git commit -m"Création du fichier README.md"
7
8 echo -e "Pour créer un projet Git, exécutez la commande `git init`. Puis pour ajouter un
  commit, faites `git add` et `git commit -m\"Mon message\"`." >> README.md
9 git add README.md
10 git commit -m"Ajout de l'explication sur git init"
11
12 echo -e "https://git-scm.com/book/fr/v2/Les-bases-de-Git-D%C3%A9marrer-un-d%C3%A9p%C3%B4t-
  Git" > sources.md
13 echo -e "[Sources](./sources.md)\n" >> README.md
14 git add README.md sources.md
15 git commit -m"Ajout des sources"
```