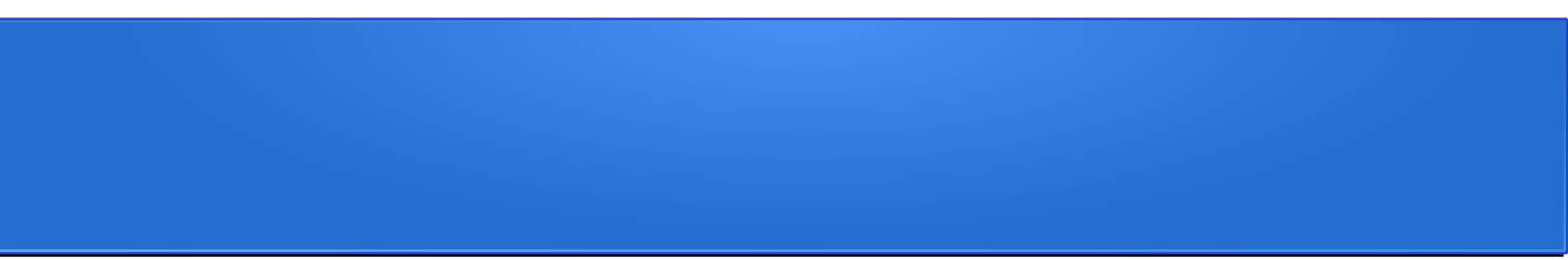


GIT : la gestion de version

Qu'est-ce que la gestion de version et pourquoi devriez-vous vous en soucier ?



Un gestionnaire de version enregistre l'évolution d'un fichier ou d'un ensemble de fichiers dans le temps. Il vous permet :

- De ramener un fichier à un état précédent,
- De ramener le projet complet à un état précédent,
- De visualiser les changements au cours du temps,
- De voir qui a modifié quelque chose qui pourrait causer un problème,
- De voir qui a introduit un problème et quand,
- De revenir facilement à un état stable
- Et plus encore.

vous obtenez tous ces avantages avec peu de travail additionnel.

3 systèmes principaux

- Les systèmes de gestion de version locaux
- VCS locaux (Version Control Systems)
- Les systèmes de gestion de version centralisés

Les systèmes de gestion de version locaux

La méthode courante pour la gestion de version est généralement de recopier les fichiers dans un autre répertoire (peut-être avec un nom incluant la date dans le meilleur des cas). Cette méthode est la plus courante parce que c'est la plus simple, mais c'est aussi la moins fiable.

Il est facile d'oublier le répertoire dans lequel vous êtes et d'écrire accidentellement dans le mauvais fichier ou d'écraser des fichiers que vous vouliez conserver.

VCS locaux

Pour traiter ce problème, les programmeurs ont développé il y a longtemps des systèmes qui utilisaient une base de données simple pour conserver les modifications d'un fichier.

Un des systèmes les plus populaires était RCS. Cet outil fonctionne en conservant des ensembles de patches (c'est-à-dire la différence entre les fichiers) d'une version à l'autre dans un format spécial sur disque ; il peut alors restituer l'état de n'importe quel fichier à n'importe quel instant en ajoutant toutes les différences.

Local Computer

Checkout

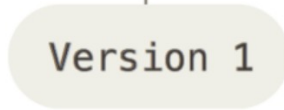
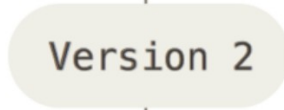
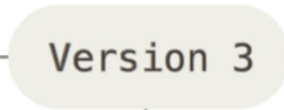
File

Version Database

Version 3

Version 2

Version 1



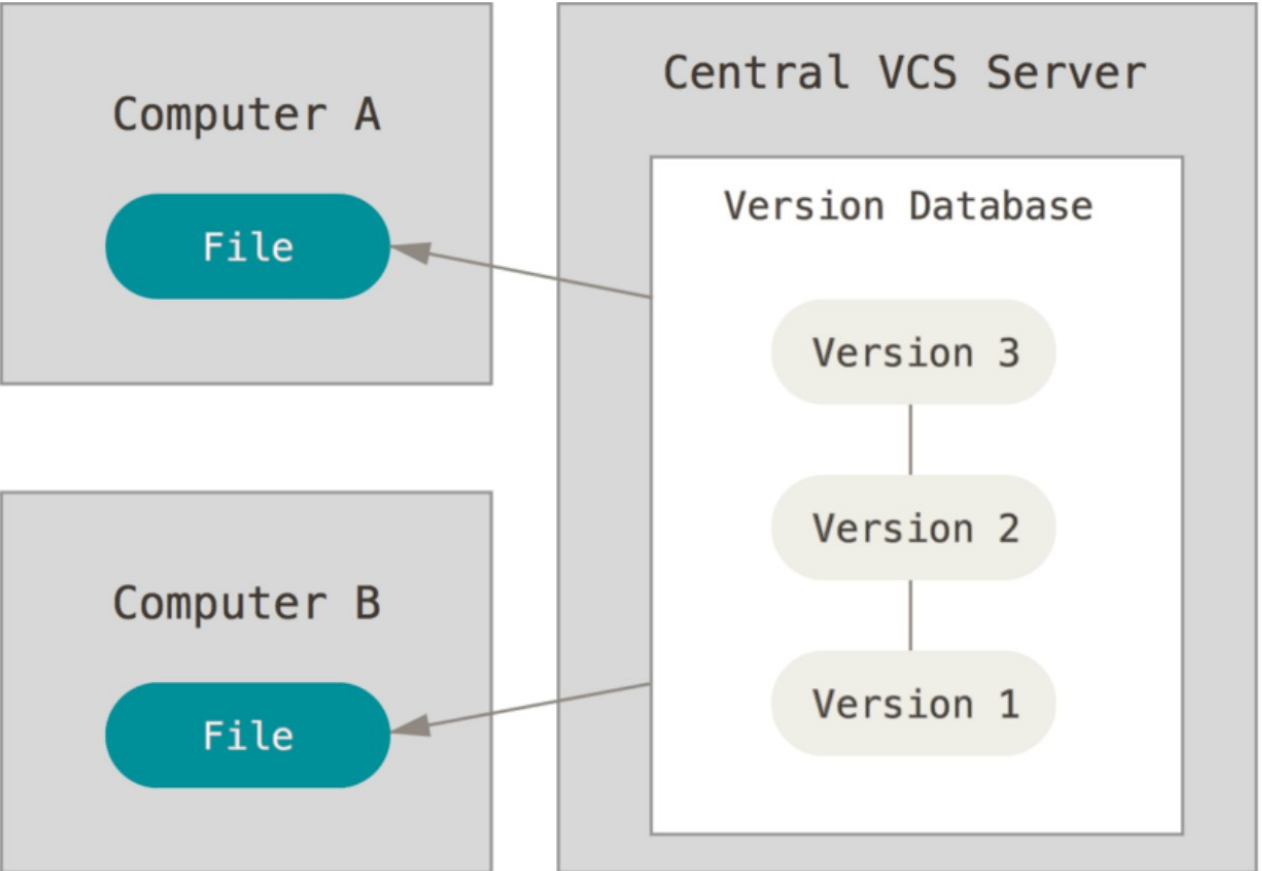
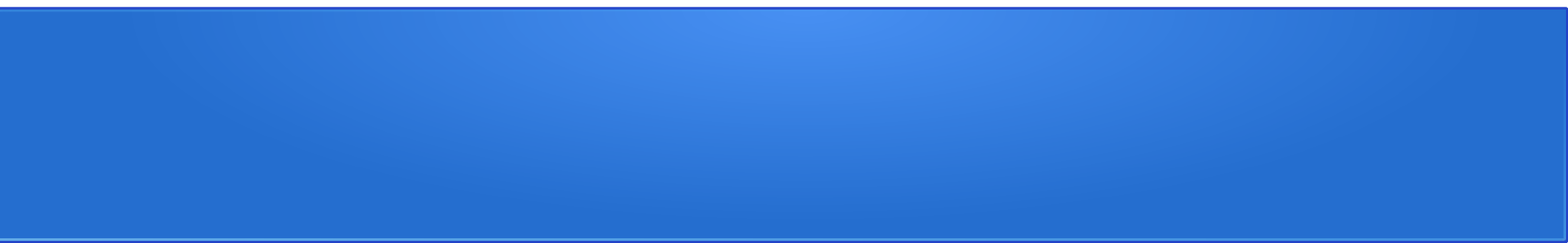
Les systèmes de gestion de version centralisés

Le problème majeur que les gens rencontrent est qu'ils ont besoin de collaborer avec des développeurs sur d'autres ordinateurs.

Pour traiter ce problème, les systèmes de gestion de version centralisés (CVCS en anglais pour Centralized Version Control Systems) furent développés.

Ces systèmes tels que CVS, Subversion, et Perforce, mettent en place un serveur central qui contient tous les fichiers sous gestion de version, et des clients qui peuvent extraire les fichiers de ce dépôt central.

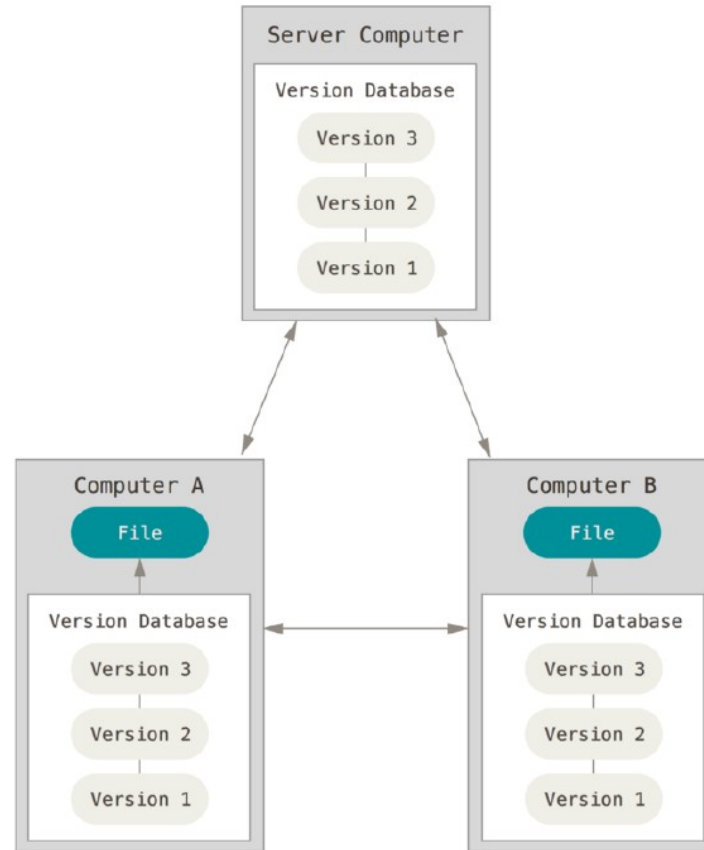
Pendant de nombreuses années, cela a été le standard pour la gestion de version.



Les systèmes de gestion de version distribués

Les systèmes de gestion de version distribués (DVCS en anglais pour Distributed Version Control Systems).

Dans un DVCS (tel que Git, Mercurial ou Darcs), les clients n'extraient plus seulement la dernière version d'un fichier, mais ils dupliquent complètement le dépôt. Ainsi, si le serveur disparaît et si les systèmes collaboraient via ce serveur, n'importe quel dépôt d'un des clients peut être copié sur le serveur pour le restaurer. Chaque extraction devient une sauvegarde complète de toutes les données.

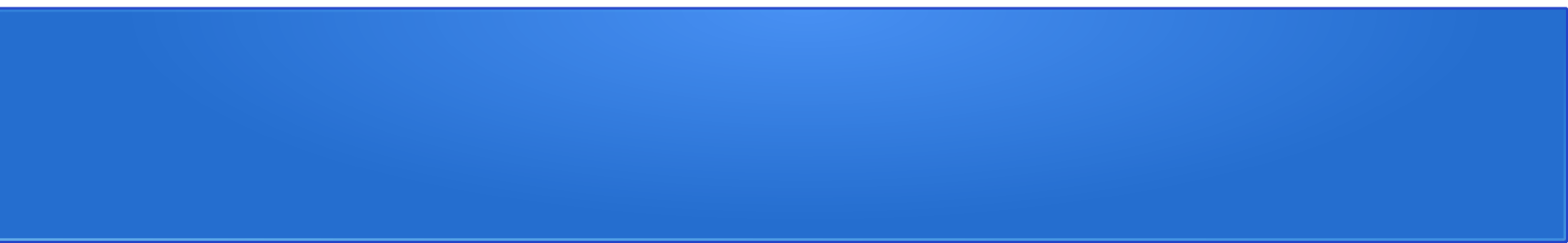


Rudiments de Git

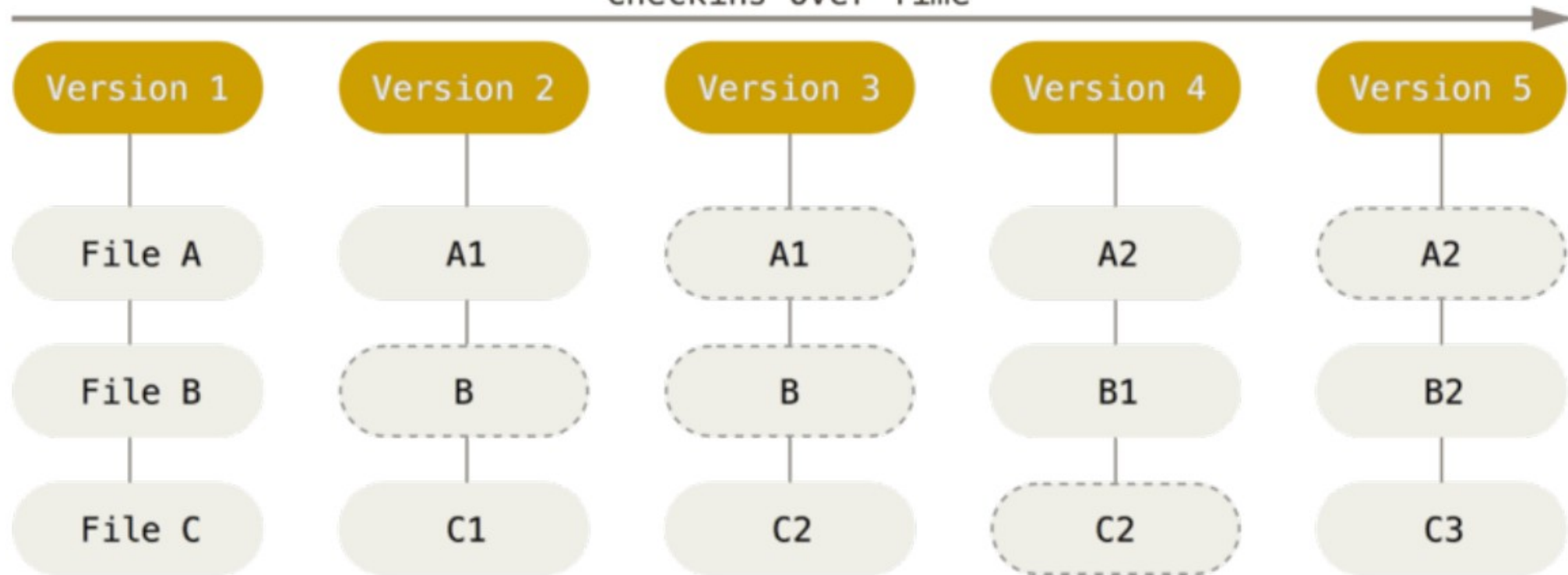
Il est important de bien comprendre la nature de GIT, parce que si on comprend cela et les principes sur lesquels il repose, alors utiliser efficacement Git devient simple.

Des instantanés, pas des différences

Git ne gère pas et ne stocke pas les informations sous la forme d'une liste de différence entre 2 versions. À la place, Git pense ses données plus comme un instantané d'un mini système de fichiers. À chaque fois que vous validez ou enregistrez l'état du projet dans Git, il prend effectivement un instantané du contenu de votre espace de travail à ce moment et enregistre une référence à cet instantané. Pour être efficace, si les fichiers n'ont pas changé, Git ne stocke pas le fichier à nouveau, juste une référence vers le fichier original qu'il a déjà enregistré. Git pense ses données plus à la manière d'un flux d'instantanés.



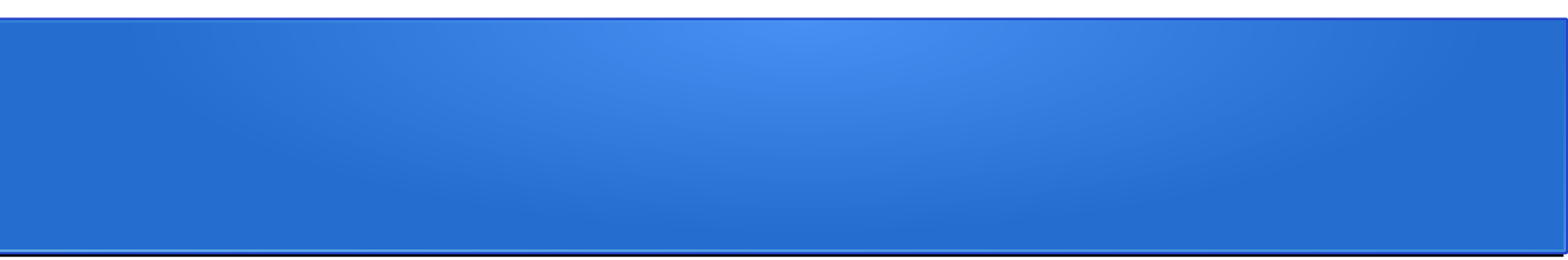
Checkins Over Time



Presque toutes les opérations sont locales

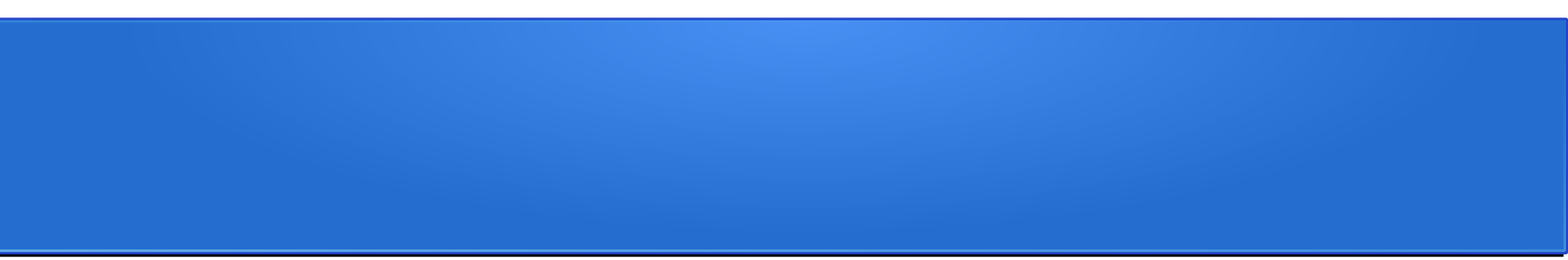
La plupart des opérations de Git ne nécessitent que des fichiers et ressources locaux généralement aucune information venant d'un autre ordinateur du réseau n'est nécessaire.

Comme vous disposez de l'historique complet du projet localement sur votre disque dur, la plupart des opérations semblent instantanées.



Par exemple, pour parcourir l'historique d'un projet, Git n'a pas besoin d'aller le chercher sur un serveur pour vous l'afficher ; il n'a qu'à simplement le lire directement dans votre base de données locale.

Cela signifie que vous avez quasi-instantanément accès à l'historique du projet. Si vous souhaitez connaître les modifications introduites entre la version actuelle d'un fichier et son état un mois auparavant, Git peut rechercher l'état du fichier un mois auparavant et réaliser le calcul de différence, au lieu d'avoir à demander cette différence à un serveur ou de devoir récupérer l'ancienne version sur le serveur pour calculer la différence localement.



Cela signifie aussi qu'il y a très peu de choses que vous ne puissiez réaliser si vous n'êtes pas connecté ou hors VPN. Si vous voyagez en train ou en avion et voulez avancer votre travail, vous pouvez continuer à gérer vos versions sans soucis en attendant de pouvoir de nouveau vous connecter pour partager votre travail. Si vous êtes chez vous et ne pouvez avoir une liaison VPN avec votre entreprise, vous pouvez tout de même travailler.

Git gère l'intégrité

Dans Git, tout est vérifié par une somme de contrôle avant d'être stocké et par la suite cette somme de contrôle, signature unique, sert de référence. Cela signifie qu'il est impossible de modifier le contenu d'un fichier ou d'un répertoire sans que Git ne s'en aperçoive. Cette fonctionnalité est ancrée dans les fondations de Git et fait partie intégrante de sa philosophie.

Vous ne pouvez pas perdre des données en cours de transfert ou corrompre un fichier sans que Git ne puisse le détecter !

Comment Ca marche

Le mécanisme que Git utilise pour réaliser les sommes de contrôle est appelé une empreinte SHA-1.

C'est une chaîne de caractères composée de 40 caractères hexadécimaux (de '0' à '9' et de 'a' à 'f') calculée en fonction du contenu du fichier ou de la structure du répertoire considéré.

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

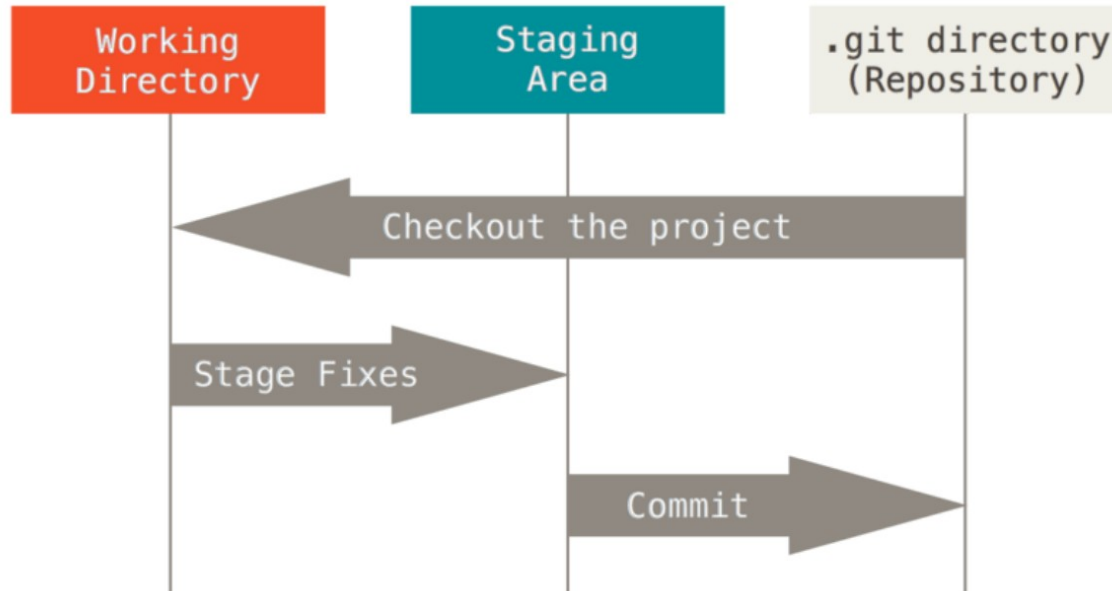
Vous trouverez ces valeurs à peu près partout dans Git

Les trois états

Git gère trois états dans lesquels les fichiers peuvent résider : modifié, indexé et validé.

- **Modifié** signifie que vous avez modifié le fichier mais qu'il n'a pas encore été validé en base.
- **Indexé** signifie que vous avez marqué un fichier modifié dans sa version actuelle pour qu'il fasse partie du prochain instantané du projet.
- **Validé** signifie que les données sont stockées en sécurité dans votre base de données **locale**.

Ceci nous mène aux trois sections principales d'un projet Git : le répertoire Git, **le répertoire de travail** et **la zone d'index**.



Le répertoire Git

C'est l'endroit où Git stocke les méta-données et la base de données des objets de votre projet.

C'est la partie la plus importante de Git, et c'est ce qui est copié lorsque vous clonez un dépôt depuis un autre ordinateur.

Le répertoire de travail

Le répertoire de travail est une extraction unique d'une version du projet. Ces fichiers sont extraits depuis la base de données compressée dans le répertoire Git et placés sur le disque pour pouvoir être utilisés ou modifiés.

C'est ce que vous voyez dans le dossier actif.



La zone d'index est un simple fichier, situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané.

On l'appelle aussi des fois la zone de préparation.

En pratique

L'utilisation standard de Git se passe comme suit :

1. vous modifiez des fichiers dans votre répertoire de travail ;
2. vous indexez les fichiers modifiés, ce qui ajoute des instantanés de ces fichiers dans la zone d'index ;
3. vous validez, ce qui a pour effet de basculer les instantanés des fichiers de l'index dans la base de données du répertoire Git.

Working
Directory

Staging
Area

.git directory
(Repository)

Checkout the project

Stage Fixes

Commit



Installation de Git

Si des interfaces graphiques existent, dans le cadre de ce cours cela ne sera qu'en ligne de commande.

- Linux : `sudo apt install git-all`
- Mac : `git --version` > si pas installé il devrait le proposer...
ou alors <https://git-scm.com/download/mac>.
- Windows : <https://git-scm.com/downloads/win>

Configuration minimum

Voir la config > `git config --list --show-origin`

Modifier son nom et email (important, cette donnée étant l'identité affichée en cas de travail en groupe)

```
git config --global user.name "John Doe"
```

```
git config --global user.email johndoe@example.com
```

Global permet de changer pour tous les projets.

A l'aide

Si vous avez besoin d'aide pour utiliser Git, il y a trois moyens d'obtenir les pages de manuel pour toutes les commandes de Git :

- `git help <commande>`
- `git <commande> --help`
- `man git-<commande>`

Par exemple, vous pouvez obtenir la page de manuel pour la commande config en lançant :

- `git help config`

Démarrer un dépôt Git

Vous pouvez principalement démarrer un dépôt Git de deux manières.

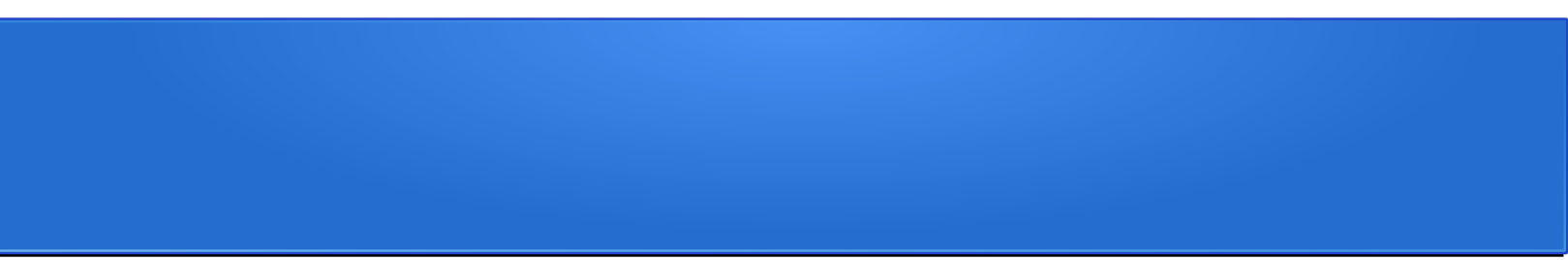
1. Vous pouvez prendre un répertoire existant et le transformer en dépôt Git.
2. Vous pouvez cloner un dépôt Git existant sur un autre serveur.

Cloner un dépôt existant

Si vous souhaitez obtenir une copie d'un dépôt Git existant, par exemple, un projet auquel vous aimeriez contribuer, la commande dont vous avez besoin s'appelle **git clone**.

Essayez de cloner dans un repertoire

<https://github.com/libgit2/libgit2> (google github libgit2)



Ceci crée un répertoire nommé libgit2, initialise un répertoire .git à l'intérieur, récupère toutes les données de ce dépôt, et extrait une copie de travail de la dernière version. Si vous examinez le nouveau répertoire libgit2, vous y verrez les fichiers du projet, prêts à être modifiés ou utilisés. Un petit **git log** vous permet de voir les divers commit (validation)

Vous pouvez supprimer le répertoire libgit2

Initialisation d'un dépôt Git dans un répertoire existant

Aller dans le répertoire en question et lancer la commande : `git init`

Cela crée un nouveau sous-répertoire nommé `.git` qui contient tous les fichiers nécessaires au dépôt — un squelette de dépôt Git. Pour l'instant, le répertoire de travail est vide donc aucun fichier n'est encore versionné.

Enregistrer des modifications dans le dépôt

Vous avez à présent un dépôt Git valide et une extraction ou copie de travail du projet. Vous devez faire quelques modifications et valider des instantanés de ces modifications dans votre dépôt chaque fois que votre projet atteint un état que vous souhaitez enregistrer.

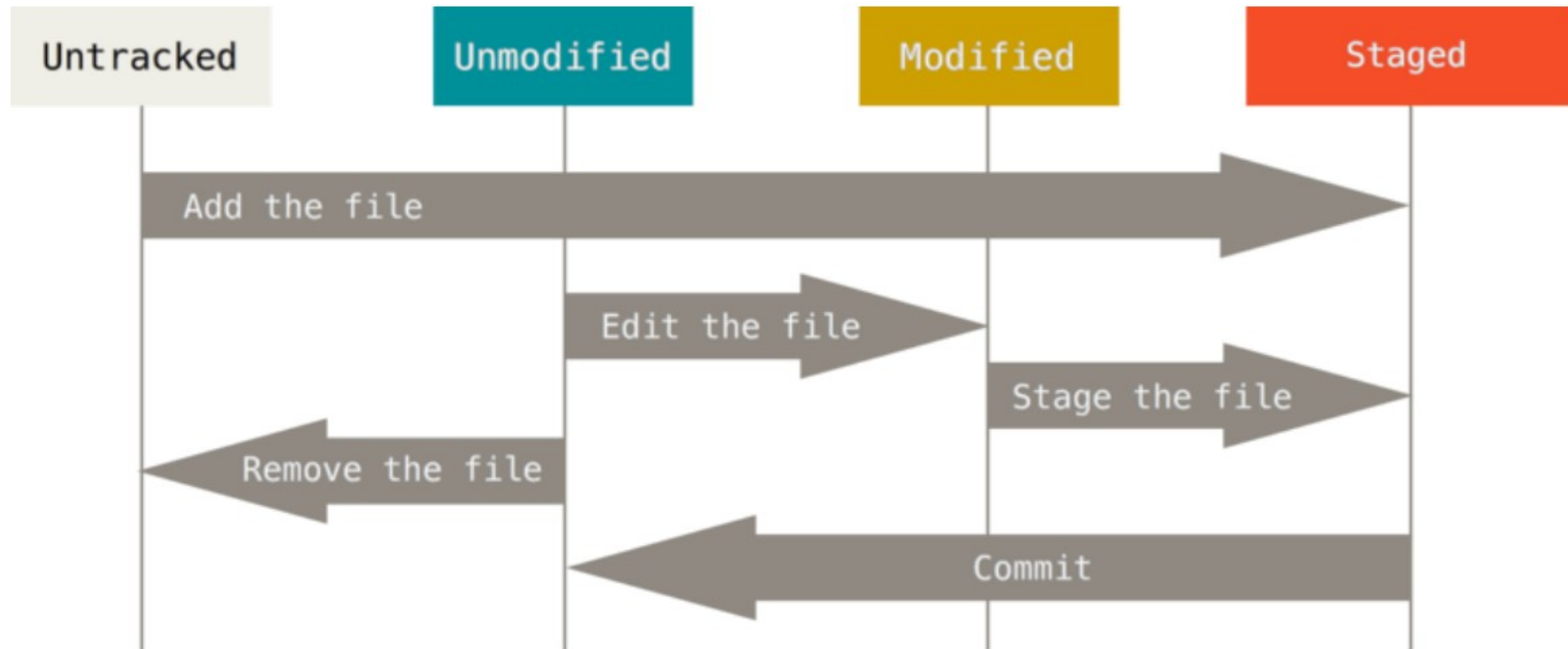
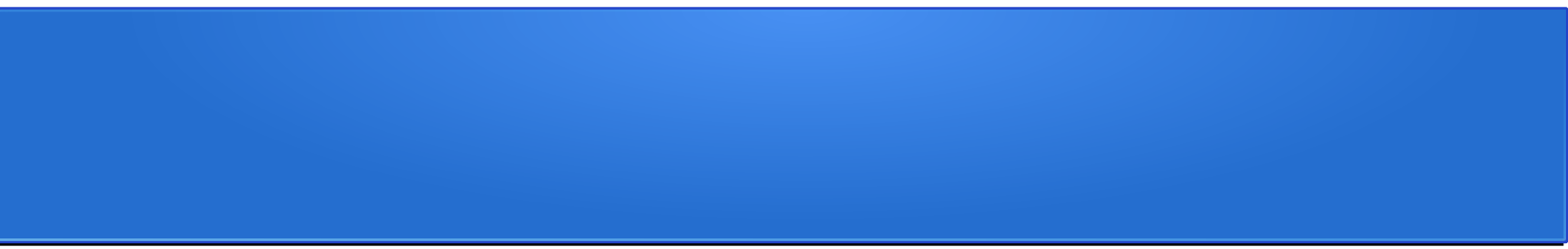
- La correction d'un bug
- Une amélioration (ou partie)
- ...

Suivi ou pas

Souvenez-vous que chaque fichier de votre copie de travail peut avoir deux états :

- **Sous suivi de version** : Les fichiers qui appartenaient déjà au dernier instantané ; ils peuvent être inchangés, modifiés ou indexés. En résumé,
- **non suivi**. tout fichier de votre copie de travail qui n'appartenait pas à votre dernier instantané et n'a pas été indexé.

Les fichiers non suivis sont par exemples des fichiers de configuration, des clés privées,...



Action !

Vérifions l'état des fichiers de notre répertoire vide

Pour cela aller dans le répertoire en question et **git status**

```
cedric@0siriris:~/textGit$ git status
Sur la branche master

Aucun commit

rien à valider (créez/copiez des fichiers et utilisez "git add" pour les suivre)
```

Logique nous n'avons pas de fichier...

Ajout d'un fichier

```
cedric@Osiris:~/textGit$ echo "Hello" > LISEZMOI
```

```
cedric@Osiris:~/textGit$ git status
```

```
Sur la branche master
```

```
Aucun commit
```

```
Fichiers non suivis:
```

```
(utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
```

```
LISEZMOI
```

```
aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez "git add" pour les suivre)
```

Vous pouvez constater que votre nouveau fichier LISEZMOI n'est pas en suivi de version, car il apparaît dans la section « Fichiers non suivis » de l'état de la copie de travail. « non suivi » signifie simplement que Git détecte un fichier qui n'était pas présent dans le dernier instantané ; Git ne le placera sous suivi de version que quand vous lui indiquerez de le faire.

Placer de nouveaux fichiers sous suivi de version

Placer de nouveaux fichiers sous suivi de version

`git add <fichier>`

```
cedric@0siriris:~/textGit$ git add LISEZMOI
cedric@0siriris:~/textGit$ git status
Sur la branche master

Aucun commit

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

nouveau fichier : LISEZMOI
```

Le fichier est bien indexé

Validons les modifications avec `git commit -m « lisezmoi »` (on y reviendra...)

A nouveau un petit git status :

```
cedric@0siriris:~/textGit$ git status  
Sur la branche master  
rien à valider, la copie de travail est propre
```

Git ne détecte pas la moindre modification entre le repertoire de travail et le dernier commit.

Modification d'un fichier suivi

Modifions le fichier LISEZMOI, et rajoutons un autre fichier maPage.html :

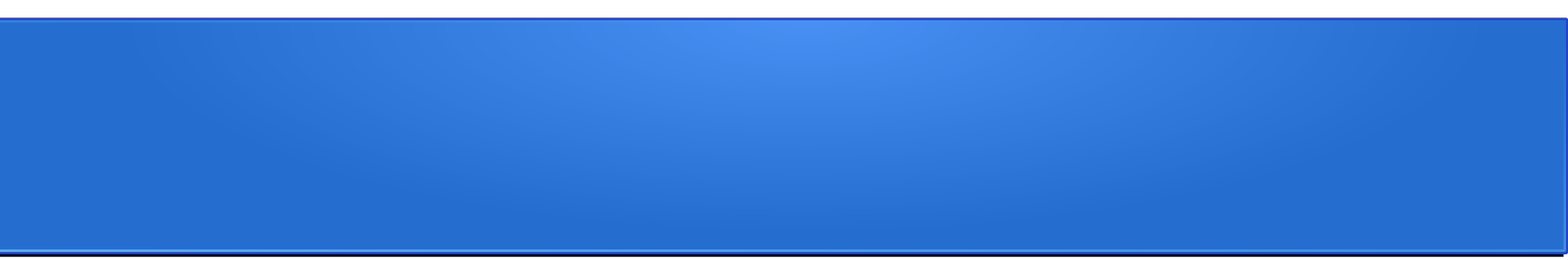
```
cedric@Osiris:~/textGit$ echo "World" >> LISEZMOI
cedric@Osiris:~/textGit$ echo "Ma super Page" > maPage.html
cedric@Osiris:~/textGit$ git status
Sur la branche master
Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

    modifié :      LISEZMOI

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

    maPage.html

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git commit -a")
```


- 
- Le fichier LISEZMOI apparaît sous la section nommée « Modifications qui ne seront pas validées » ce qui signifie que le fichier sous suivi de version a été modifié dans la copie de travail mais n'est pas encore indexé.

Pour l'indexer, il faut lancer la commande `git add`.

`git add` est une commande multi-usage

- elle peut être utilisée pour placer un fichier sous suivi de version,
- Pour indexer un fichier
- pour d'autres actions telles que marquer comme résolus des conflits de fusion de fichiers.

Git add = « ajouter ce contenu pour la prochaine validation »

Lançons `git add maPage.html` et `git add LISEZMOI`

```
cedric@0siriris:~/textGit$ git add maPage.html
cedric@0siriris:~/textGit$ git add LISEZMOI
cedric@0siriris:~/textGit$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    modifié :      LISEZMOI
    nouveau fichier : maPage.html
```

Nous voyons que l'ensemble sera validé lors du prochain commit

Modifions à nouveau LisezMoi

```
cedric@0siriris:~/textGit$ echo "kiss" >> LISEZMOI
cedric@0siriris:~/textGit$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    modifié :      LISEZMOI
    nouveau fichier : maPage.html

Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

    modifié :      LISEZMOI
```

LISEZMOI est dans les modifications validées et non validées ??? Pourquoi ?



Que s'est-il donc passé ? À présent, LISEZMOI apparaît à la fois comme indexé et non indexé.

En fait, Git indexe un fichier dans son état au moment où la commande `git add` est lancée. Si on valide les modifications maintenant, la version de LISEZMOI qui fera partie de l'instantané est celle correspondant au moment où la commande `git add LISEZ MOI` a été lancée, et non la version actuellement présente dans la copie de travail au moment où la commande `git commit` est lancée. Si le fichier est modifié après un `git add`, il faut relancer `git add` pour prendre en compte l'état actuel de la copie de travail.

Ignorer des fichiers

Il apparaît souvent qu'un type de fichiers présent dans la copie de travail ne doit pas être ajouté automatiquement ou même ne doit pas apparaître comme fichier potentiel pour le suivi de version. Ce sont par exemple :

- Des fichiers générés automatiquement tels que les fichiers de journaux ou de sauvegardes.
- Des fichiers binaires ou système
- Les répertoires de contenu sur un site web dynamique (images, fichiers, xml,...)

Dans un tel cas, on peut énumérer les patrons de noms de fichiers à ignorer dans un fichier `.gitignore`.

Créons un .gitignore qui n'indexe pas les fichiers qui finissent par .xyz et créons un fichier de la sorte

```
cedric@0siriris:~/textGit$ echo "*.xyz" > .gitignore
cedric@0siriris:~/textGit$ touch text.xyz
cedric@0siriris:~/textGit$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    modifié :      LISEZMOI
    nouveau fichier : maPage.html

Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

    modifié :      LISEZMOI

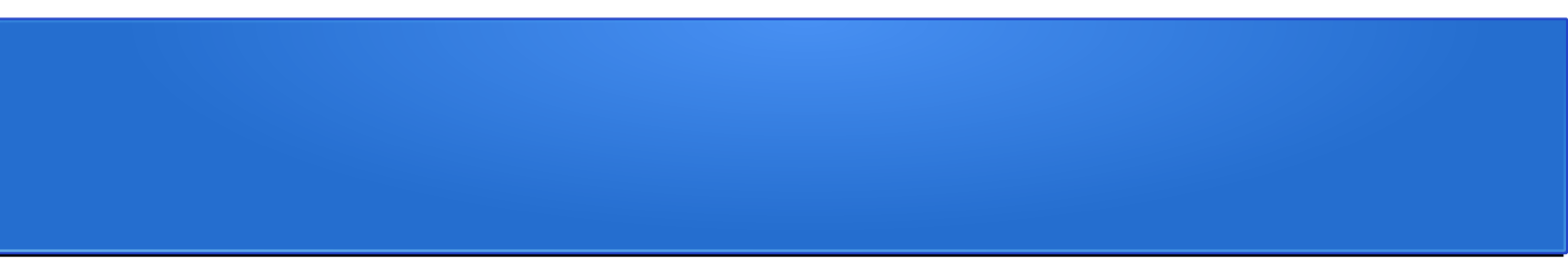
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

    .gitignore
```

le fichier .gitignore est bien détecté par git mais pas test.xyz

Vérifions le contenu du répertoire :

```
cedric@0siriris:~/textGit$ ls -lah
total 24K
drwxr-xr-x  3 cedric cedric 4,0K oct  6 17:22 .
drwxr-xr-x 60 cedric cedric 4,0K oct  6 16:40 ..
drwxr-xr-x  8 cedric cedric 4,0K oct  6 17:22 .git
-rw-r--r--  1 cedric cedric   6 oct  6 17:21 .gitignore
-rw-r--r--  1 cedric cedric  29 oct  6 17:15 LISEZMOI
-rw-r--r--  1 cedric cedric  14 oct  6 17:06 maPage.html
-rw-r--r--  1 cedric cedric   0 oct  6 17:22 text.xyz
```



Lorsqu'on désire savoir non seulement quels fichiers ont changé mais aussi ce qui a changé dans ces fichiers, on peut utiliser la commande `git diff`. Cette commande sera utilisée le plus souvent pour répondre aux questions suivantes :

- qu'est-ce qui a été modifié mais pas encore indexé ?
- Quelle modification a été indexée et est prête pour la validation ?
- Là où `git status` répond de manière générale à ces questions, `git diff` montre les lignes exactes qui ont été ajoutées, modifiées ou effacées

Le patch en somme.

Le fichier contenait « Hello World » et j'ai rajouté kiss
(depuis le dernier commit!)

```
cedric@0siriris:~/textGit$ git diff
diff --git a/LISEZMOI b/LISEZMOI
index edaee6f..f5bd66c 100644
--- a/LISEZMOI
+++ b/LISEZMOI
@@ -2,3 +2,4 @@ Hello
World
+kiss
```

Valider vos modifications

Maintenant que votre zone d'index est dans l'état désiré, vous pouvez valider vos modifications. Souvenez-vous que tout ce qui est encore non indexé — tous les fichiers qui ont été créés ou modifiés mais n'ont pas subi de `git add` depuis que vous les avez modifiés ne feront pas partie de la prochaine validation. Ils resteront en tant que fichiers modifiés sur votre disque.

Dans notre cas, nous allons indexer l'intégralité des modifications pour pouvoir valider l'ensemble :

```
git add .
```

Notez le point qui permet d'indexer tous les fichiers en attente

On est prêt à valider !

```
cedric@0siriris:~/textGit$ git add .
cedric@0siriris:~/textGit$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    nouveau fichier : .gitignore
    modifié :         LISEZMOI
    nouveau fichier : maPage.html
```

Valider

git commit

Cette commande ouvre votre éditeur de texte en console pour vous permettre d'indiquer un message explicite de ce commit (ex : résolution problème d'envoi de mails à l'administrateur).

```
# Veuillez saisir le message de validation pour vos modifications. Les lignes
# commençant par '#' seront ignorées, et un message vide abandonne la validation.
#
# Sur la branche master
# Modifications qui seront validées :
#     nouveau fichier : .gitignore
#     modifié :          LISEZMOI
#     nouveau fichier : maPage.html
#
```

```
cedric@0siriris:~/textGit$ git commit
[master 12d1e6d] résolution problème d'envoi de mails à l'administrateur
3 files changed, 6 insertions(+)
create mode 100644 .gitignore
create mode 100644 maPage.html
cedric@0siriris:~/textGit$ git status
Sur la branche master
rien à valider, la copie de travail est propre
cedric@0siriris:~/textGit$
```

Notez que nous aurions pu utiliser la notation rapide au lieu de **git commit** (qui ouvre l'editeur)

git commit -m"résolution problème..."

le m indique que la suite est le message à inclure

Passer l'étape de mise en index

De la meme facon nous aurions pu éviter le `git add` . Suivi du `git commit` de la sorte :

`git commit -a` (cela crée un `git add` . Avant le commit)

et pour ne pas avoir l'éditeur qui s'ouvre :

`git commit -am «message»`

Effacement de fichier

Effacer de quoi ?

- Supprimer le fichier de l'espace de travail
- Supprimer des fichiers suivis

Supprimer un fichier inutile

```
cedric@0siriris:~/textGit$ rm LISEZMOI
cedric@0siriris:~/textGit$ git status
Sur la branche master
Modifications qui ne seront pas validées :
  (utilisez "git add/rm <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

    supprimé :      LISEZMOI

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git commit -a")
```

Si vous effacez simplement le fichier dans votre copie de travail, il apparaîtra sous la section « Modifications qui ne seront pas validées » (c'est-à-dire, non indexé) dans le résultat de git status

Ensuite, si vous lancez `git rm <fichier>`, l'effacement du fichier est indexé. `Git add .` Aurait fait également le job

```
cedric@Osiris:~/textGit$ git rm LISEZMOI
rm 'LISEZMOI'
cedric@Osiris:~/textGit$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

       supprimé :      LISEZMOI

cedric@Osiris:~/textGit$ ls
maPage.html  test2  text.xyz
```

Exercice simple

Recréez le fichier LISEZMOI et validez cette nouvelle version

Solution :

```
cedric@0siriris:~/textGit$ echo "Hello World" > LISEZMOI
cedric@0siriris:~/textGit$ git commit -am"nouveau fichier"
[master ea0d459] nouveau fichier
 1 file changed, 5 deletions(-)
 delete mode 100644 LISEZMOI
cedric@0siriris:~/textGit$ git █
```

Supprimer de l'index

Un autre scénario serait de vouloir abandonner le suivi de version d'un fichier tout en le conservant dans la copie de travail. Ceci est particulièrement utile lorsqu'on a accidentellement indexé un fichier pour cela l'option `--cached` sera utilisée

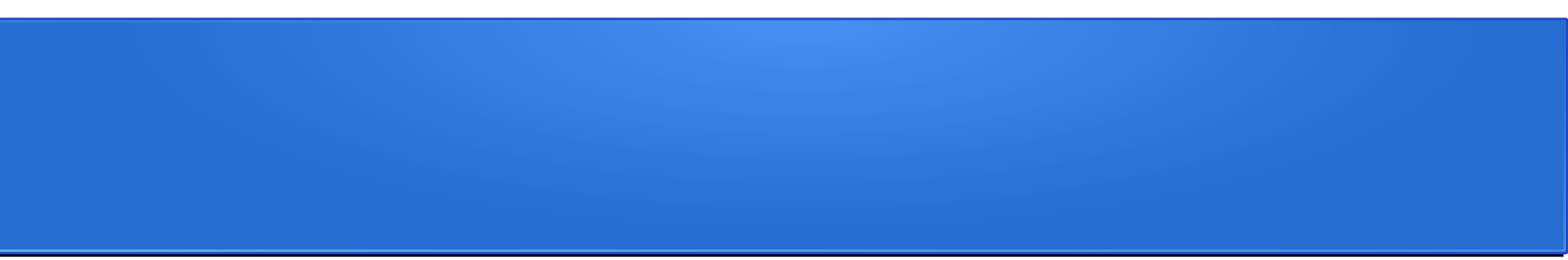
Il est donc supprimé de l'index de git, reste physiquement dans le repertoire. Il st fort probable qu'il devrait etre indiqué dans le .gitignore

```
cedric@Osiris:~/textGit$ git rm --cached LISEZMOI
rm 'LISEZMOI'
cedric@Osiris:~/textGit$ ls
LISEZMOI  maPage.html  test2  text.xyz
cedric@Osiris:~/textGit$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    supprimé :      LISEZMOI

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

    LISEZMOI
```

- 
- déplacement/renommage
 - push/pull
 - Voir les Les versions
 - Checkout
 - Branches
 -

Creation d'un compte en ligne

Avec moins de restriction que gitHub, creez un compte sur <https://bitbucket.org/>

Merci d'utiliser vos noms et adresses mail ifapme pour que je m'y retrouve...