Formation
Prise en main de GIT

Novembre 12, 2019

Dr. L. BAATI

Audience

- Développeur Mobile / Front et back
- Chefs de projet de développement
- steakholders

Objectifs

• Étudier Git pour pouvoir l'utiliser pour des projets personnels ou professionnels.

• Git surclasse les autres outils SCM (Source Code Management) par sa performance, la taille des dépôts et ses fonctionnalités uniques.

• Étudier en profondeur les fondamentaux de Git. Bien comprendre les bases, pour voir des notions plus complexes lors de la formation git avancée.

• A l'issue de la formation, vous aurez appris à configurer et utiliser GIT dans un contexte de gestion quotidienne des sources d'un projet professionnel.

Plan global

Formation GIT

Partie I :

1. Présentation de GIT
2. Les bases de git
3. Les branches avec Git

Plan

- Présentation de GIT
- Les bases de GIT
- Les branches avec GIT
- GITHUB / BITBUCKET

Introduction

Le nom GIT

Torvalds sarcastically quipped about the name *git* (which means *unpleasant person* in British English slang): **"I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'."The man page describes Git as "the stupid content tracker"**. The read-me file of the source code elaborates further:

> The name "git" was given by Linus Torvalds when he wrote the very first version.
>
> He described the tool as "**the stupid content tracker**" and the name as (depending on your way):
>
> - **random three-letter combination** that is pronounceable, and not actually used by any common UNIX command.
> - stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
> - "**global information tracker**": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
> - etc.

Les possibilités de git

- Git permet :
  - De versionner ses fichiers (code, images, documents, ...)
  - De mettre en attente une version et de travailler sur une autre
  - De pouvoir fusionner un même fichier sur lequel plusieurs personnes ont travaillé
  - Organiser son travail par version
  - Publier en production son code à partir d'une version donnée
  - Avoir une historique précise de son projet
  - De pouvoir blâmer quelqu'un !
  - ...

1. Présentation de GIT

1.1. Gestion des versions
1.2. Rudiments de Git
1.3. Installation de Git
1.4. Paramétrage de Git

1. Présentation de GIT

1.1 Gestion des versions

Plan

- Qu’est-ce qu’un gestionnaire de version ?
- Les systèmes de gestion de version local
- Les systèmes de gestion de version centralisé
- Les systèmes de gestion de version distribué

Qu’est-ce qu’un gestionnaire de version ?

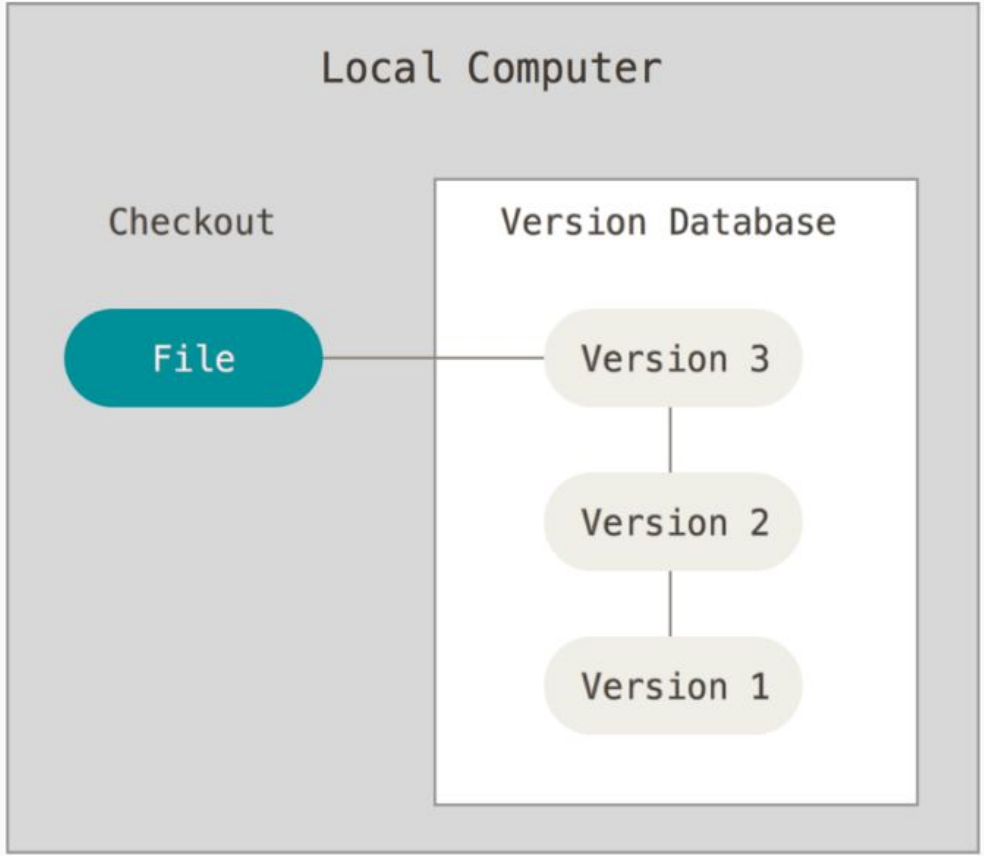
- Un gestionnaire de version :
  - Enregistre les évolutions d’un fichier
  - Permets de revenir à une version précédente
  - Fonctionne avec tout type de fichier (.txt, .php, .java, .jpg, .exe, ...)
  - Permets de retrouver un fichier supprimé

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 (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.

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

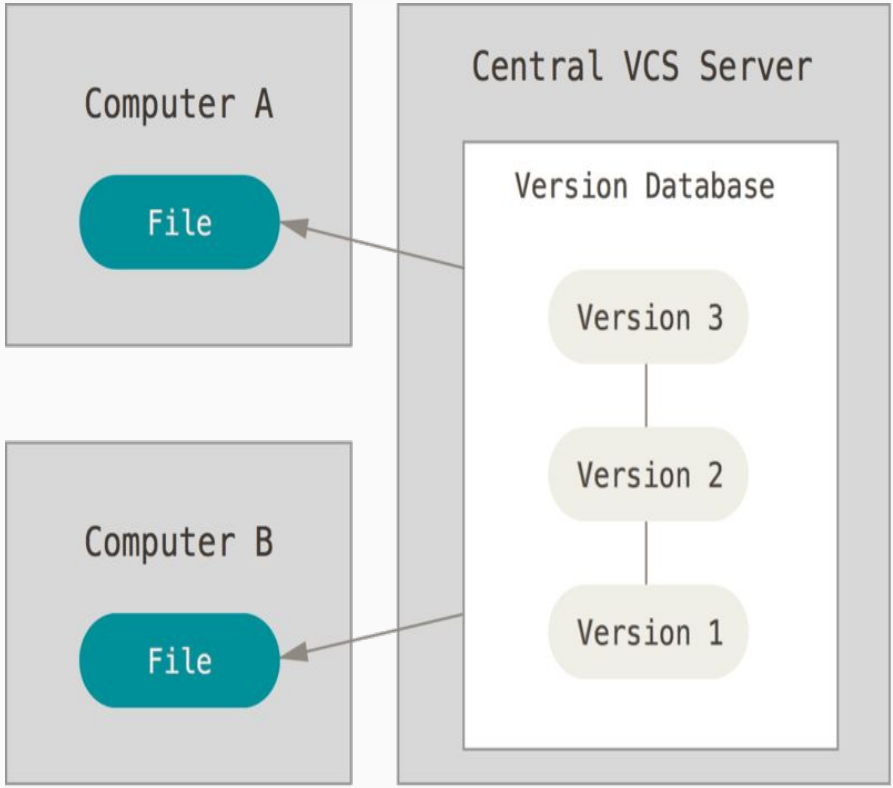
Un des systèmes les plus populaires était RCS, qui est encore distribué avec de nombreux OS. Même Mac OS X inclut rcs lorsqu'on installe les outils de développement logiciel.



Les systèmes de gestion de version centralisés

Ce schéma offre de nombreux avantages par rapport à la gestion de version locale. Par exemple, chacun sait jusqu'à un certain point ce que tous les autres sont en train de faire sur le projet. Les administrateurs ont un contrôle fin des permissions et il est beaucoup plus facile d'administrer un CVCS que de gérer des bases de données locales.

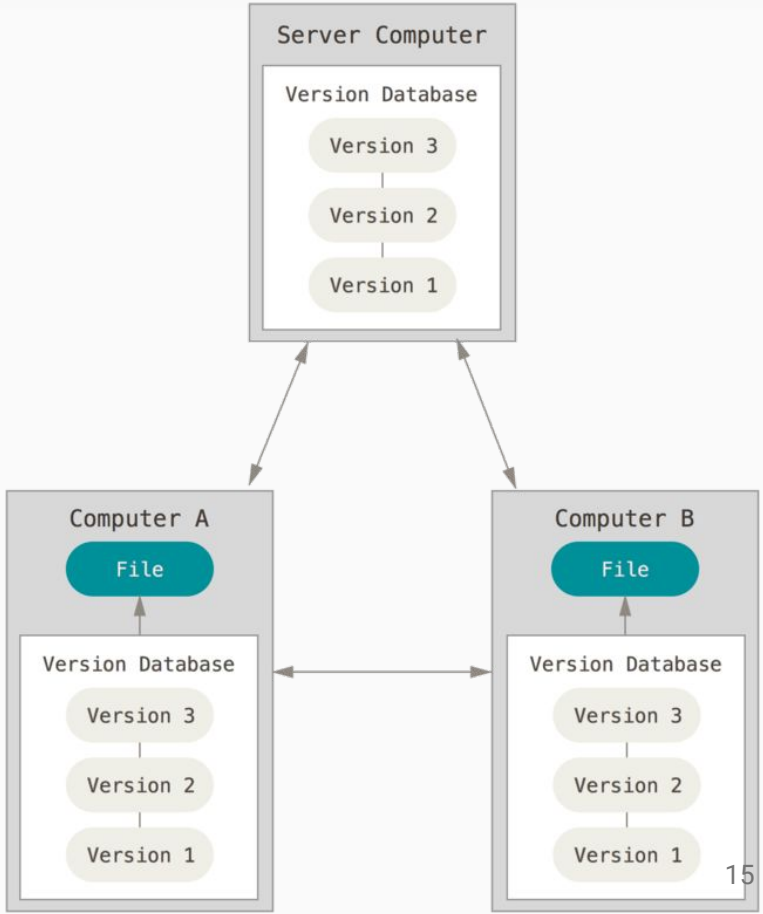
Cependant ce système a aussi de nombreux défauts. Le plus visible est le point unique de panne que le serveur centralisé représente. Si ce serveur est en panne pendant une heure, alors durant cette heure, aucun client ne peut collaborer ou enregistrer les modifications issues de son travail.



Les systèmes de gestion de version distribués

DVCS **Distributed Version Control Systems**.
Dans un DVCS tel que (**Git, Mercurial, Bazaar 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.

De plus, un grand nombre de ces systèmes gère particulièrement bien le fait d'avoir plusieurs dépôts avec lesquels travailler, vous permettant de collaborer avec différents groupes de personnes de manières différentes simultanément dans le même projet. ⇒ Cela **permet la mise en place de différentes chaînes de traitement** qui ne sont pas réalisables avec les systèmes centralisés, tels que les modèles hiérarchiques.



1. Présentation de GIT

1.2. Rudiments de git

Plan

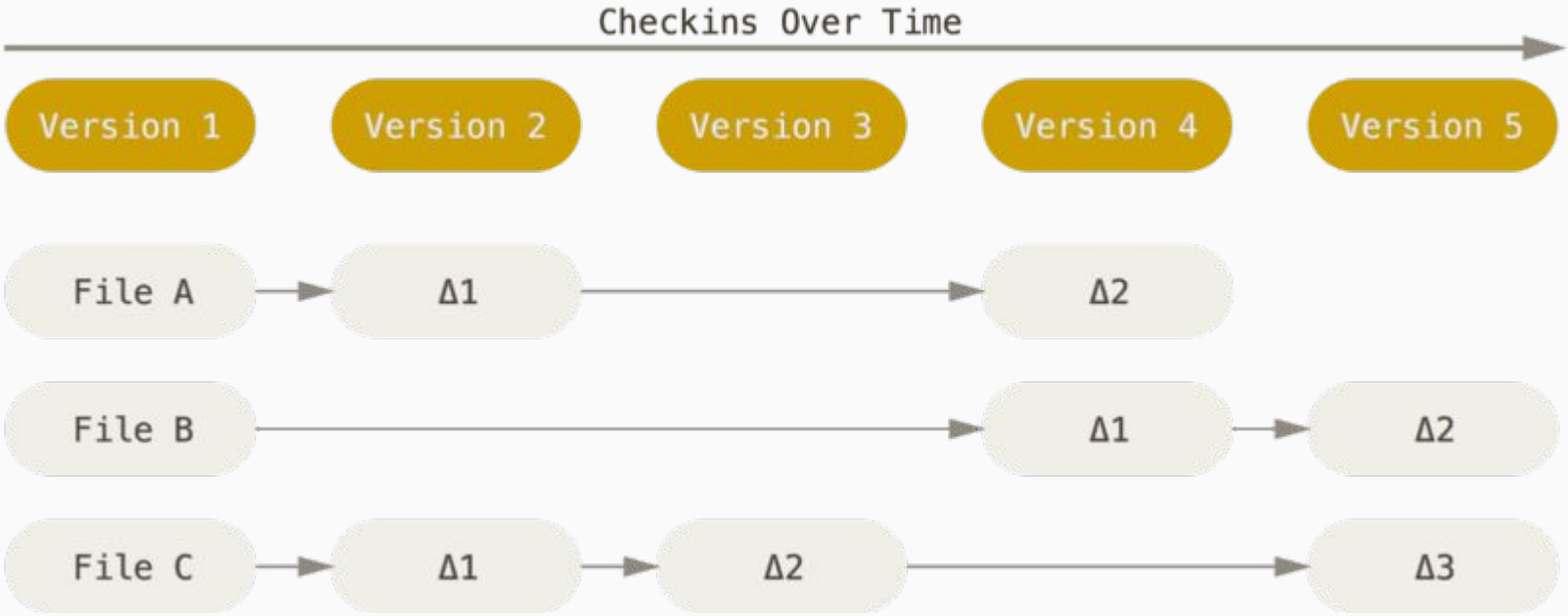
- L’histoire de GIT
- Des instantanés, pas des différences
- Presque toutes les opérations sont locales
- Git gère l’intégrité
- Les trois états

L’histoire de GIT

- Le noyau Linux est un projet libre de grande envergure. De 1991 à 2002, les modifications étaient transmises sous forme de **patchs et d’archives de fichiers**. En 2002, le projet du noyau Linux commença à utiliser un **DVCS propriétaire appelé BitKeeper**.
- En 2005, les relations entre la communauté développant le noyau Linux et la société en charge du développement **de BitKeeper furent rompues, et le statut de gratuité de l’outil fut révoqué**. Cela poussa la communauté du développement de Linux à développer son propre outil en se basant sur les leçons apprises lors de l’utilisation de BitKeeper. Certains des objectifs sont :
  - Vitesse
  - Conception simple
  - Support pour les développements non linéaires (milliers de branches parallèles)
  - Complètement distribué
  - Capacité à gérer efficacement des projets d’envergure tels que le noyau Linux (vitesse et compacité des données).

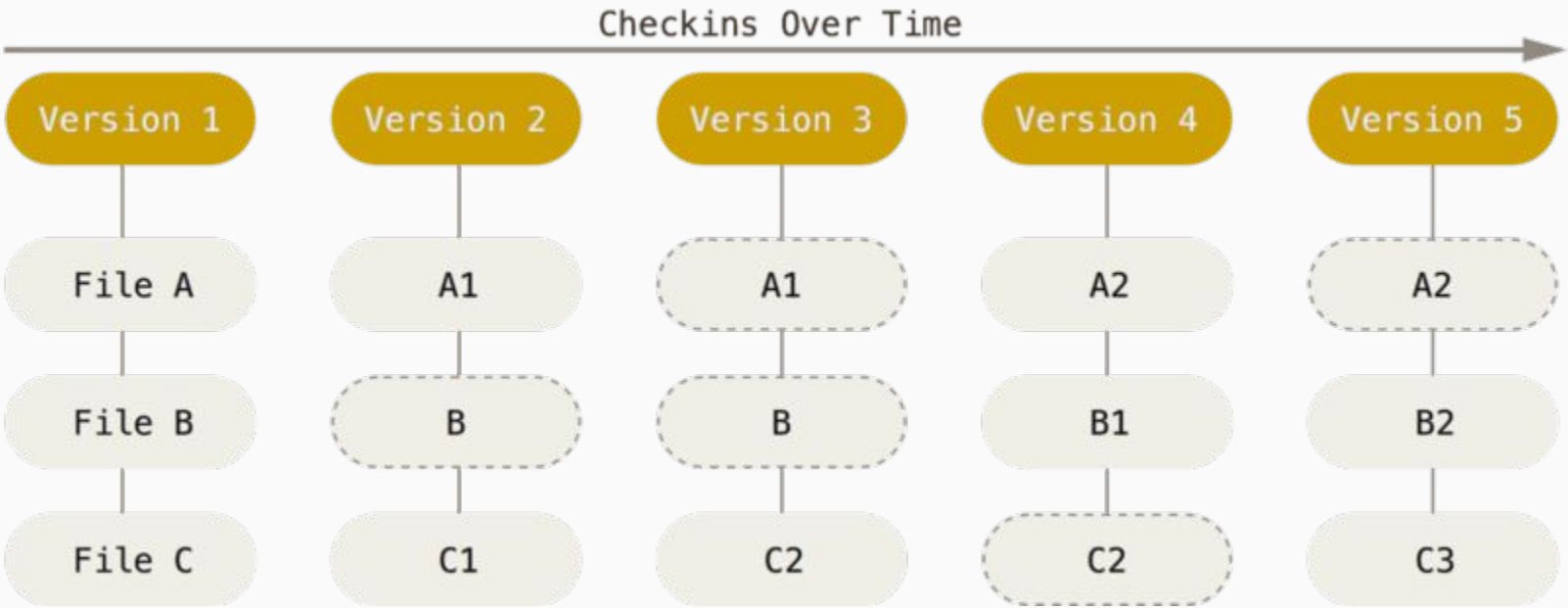
Des instantanés, pas des différences

Beaucoup de VCS considèrent l'information qu'ils gèrent comme une liste de fichiers et les modifications effectuées sur chaque fichier dans le temps.



Des instantanés, pas des différences

Avec Git, à chaque fois que vous validez ou enregistrez l'état du projet, il prend un instantané du contenu de votre espace de travail.



Presque toutes les opérations sont locales

- La plupart des opérations de Git se font en local.
- Même pour chercher dans l’historique du projet.
- L’intégralité de la base de données est en locale.

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.

• Impossible de perdre un fichier dans Git sans que Git s’en aperçoive.

•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.

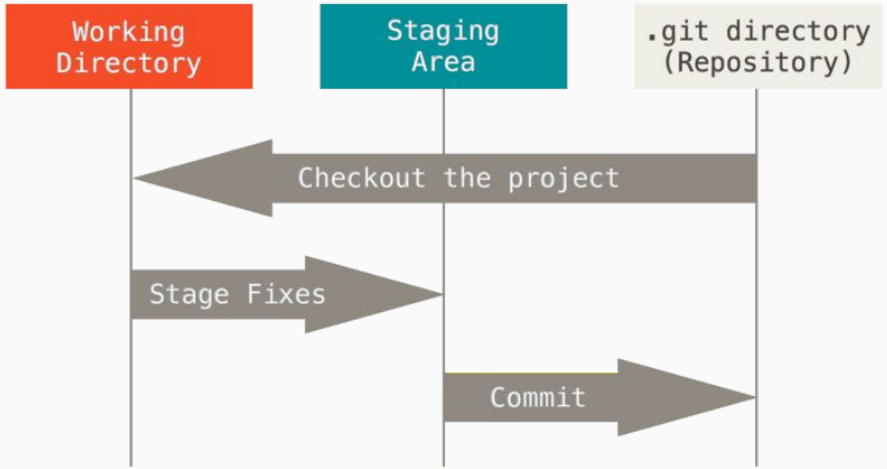
Une empreinte SHA-1.

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

⇒ On trouve ces valeurs à peu près partout dans Git car il les utilise pour tout. En fait, Git stocke tout non pas avec des noms de fichiers, mais dans **la base de données Git indexée par ces valeurs**.

Les trois états

- Un fichier versionné sous Git peut être sous 3 états :
  - Validé, le fichier est sauvegardé en base.
  - Modifié, le fichier est modifié, la modification n'est pas en base.
  - Indexé, le fichier est modifié, et est prêt à être envoyé en base.



Les trois états

- Ce qui nous donne trois sections principales d'un projet GIT
  - Validé = le répertoire .git
  - Modifié = Le répertoire de travail courant
  - Indexé = La zone virtuelle qui liste les fichiers qui seront dans le prochain instantané

Le **répertoire Git** est l'endroit où Git stocke les métadonnées 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 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.
La zone d'index est un simple fichier, généralement 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**.
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**.

1. Présentation de GIT

1.3. Installation de git

Plan

- Installation sous Linux
- Installation sous MacOS
- Installation sous Windows
- Installation à partir de source

Installation sous linux

Généralement, présent sur paquet de votre distribution sous le nom de ‘git’

- Sur fedora : `dnf install git`
- Sur Redhat : `yum install git`
- Sur Debian / Ubuntu : `apt-get install git`

```
# git --version
```

- La liste des distributions est disponible à cette adresse : https://git-scm.com/download/linux

Installation sous MACOS

• Il existe plusieurs méthodes d'installation de Git sur un Mac. La plus facile est probablement d'installer les Xcode Command Line Tools.

• Si vous voulez une version plus récente :

https://git-scm.com/download/mac

Installation depuis les sources

Pour installer Git, vous avez besoin des bibliothèques suivantes : autotools, curl, zlib, openssl, expat, libiconv. (Fedora, Redhat et Ubuntu/Debian)

```
$ dnf install curl-devel expat-devel gettext-devel openssl-devel zlib-devel
```

```
$ yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel
```

```
$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev
```

• Les sources sont disponibles sur ce site (miroir) :

https://github.com/git/git/releases

Installation depuis les sources

- Procédure d'installation :

```
$ tar -zxf git-1.9.1.tar.gz

$ cd git-1.9.1

$ make configure

$ ./configure --prefix=/usr

$ make all doc info

$ sudo make install install-doc install-html install-info
```

1. Présentation de GIT

1.4. Paramétrage de git

Plan

- Les fichiers de configuration
- Identité
- Éditeur de texte
- Paramètres

Les fichiers de configuration

- Git contient un outils `git config` pour modifier les paramètres de configuration
- Git a 3 niveaux de configuration
  - Global au système : `/etc/gitconfig`, Contient les valeurs pour tous les utilisateurs et tous les dépôts du système. Si vous passez l'option --system à git config, il lit et écrit ce fichier spécifiquement. `git config --system`
  - Global à l'utilisateur : `~/.gitconfig`, Spécifique à votre utilisateur. Vous pouvez forcer Git à lire et écrire ce fichier en passant l'option --global `git config --global`
  - Global au projet : .git/config, modifiable avec 'git config' du dépôt en cours d'utilisation : spécifique au seul dépôt en cours.
- Chaque niveau surcharge le précédent

Votre identité & Editeur de texte

La première chose à faire après l'installation de Git est de renseigner votre nom et votre adresse email.

```
$ git config --global user.name "lassaad BAATI"
$ git config --global user.email lassaad@lacksys.com
```

Vu que nous configurons ces options avec –global, nous n'aurons pas besoin de les reconfigurer à chaque projet.

• Nous faisons la même chose avec notre éditeur de texte.

```
$ git config --global core.editor emacs
```

Vos paramètres

- il est possible de vérifier nos paramètres.

```
$ git config --list
```

```
lassaad@my-leeds-ws:~/git_formation$ git config --global user.name "Lassaad BAATI"
lassaad@my-leeds-ws:~/git_formation$ git config --global user.email lassaad.baati@gmail.com
lassaad@my-leeds-ws:~/git_formation$ git config --list
user.name=Lassaad BAATI
user.email=lassaad.baati@gmail.com
lassaad@my-leeds-ws:~/git_formation$
```

Paramètres

Vous pouvez aussi vérifier la valeur effective d'un paramètre particulier en tapant git config <paramètre> :

```
$ git config user.name
Lassaad BAATI
```

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
```

2. Les bases de GIT

2.1. Démarrer un dépôt Git
2.2. Enregistrer des modifications dans le dépôt
2.3. Visualiser l'historique des validations
2.4. Annuler les actions
2.5. Travailler avec des dépôts distants
2.6. Etiquetage

2. Les bases de GIT

2.1 Démarrer un dépôt git

Plan

- Initialisation d’un dépôt Git
- Cloner un dépôt existant

Démarrer avec Git

- Vous pouvez principalement démarrer un dépôt Git de deux manières. La première consiste à prendre un projet ou un répertoire existant et à l’importer dans Git. La seconde consiste à cloner un dépôt Git existant sur un autre serveur.

Initialisation d'un dépôt Git dans un répertoire existant

⇒ Pour un projet existant, on peut se positionner dans le répertoire du projet et saisir :

```
$ 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, aucun fichier n'est encore versionné.

Si on démarre le contrôle de version sur des fichiers existants (par opposition à un répertoire vide), on peut suivre ces fichiers et faire un commit initial. On peut e faire avec add suivi par un git commit :

```
$ git add *.c
```

```
$ git add LICENSE
```

```
$ git commit -m 'initial project version'
```

git status / git init

```
lassaad@my-leeds-ws:~/git_formation/proj02$ ls -all
total 64
drwxr-xr-x  7 lassaad lassaad  4096 nov.  11 10:33 .
drwxr-xr-x  5 lassaad lassaad  4096 nov.  11 10:31 ..
-rw-r--r--  1 lassaad lassaad  1379 nov.  11 10:32 app.js
drwxr-xr-x  2 lassaad lassaad  4096 nov.  11 10:32 bin
drwxr-xr-x 86 lassaad lassaad  4096 nov.  11 10:33 node_modules
-rw-r--r--  1 lassaad lassaad   338 nov.  11 10:33 package.json
-rw-r--r--  1 lassaad lassaad 25153 nov.  11 10:33 package-lock.json
drwxr-xr-x  5 lassaad lassaad  4096 nov.  11 10:32 public
drwxr-xr-x  2 lassaad lassaad  4096 nov.  11 10:32 routes
drwxr-xr-x  2 lassaad lassaad  4096 nov.  11 10:32 views
lassaad@my-leeds-ws:~/git_formation/proj02$
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
fatal: ni ceci ni aucun de ses répertoires parents n'est un dépôt git : .git
lassaad@my-leeds-ws:~/git_formation/proj02$
lassaad@my-leeds-ws:~/git_formation/proj02$ git init
Dépôt Git vide initialisé dans /home/lassaad/git_formation/proj02/.git/
lassaad@my-leeds-ws:~/git_formation/proj02$ ls -all
total 68
drwxr-xr-x  8 lassaad lassaad  4096 nov.  11 10:36 .
drwxr-xr-x  5 lassaad lassaad  4096 nov.  11 10:31 ..
-rw-r--r--  1 lassaad lassaad  1379 nov.  11 10:32 app.js
drwxr-xr-x  2 lassaad lassaad  4096 nov.  11 10:32 bin
drwxr-xr-x  7 lassaad lassaad  4096 nov.  11 10:36 .git
drwxr-xr-x 86 lassaad lassaad  4096 nov.  11 10:33 node_modules
-rw-r--r--  1 lassaad lassaad   338 nov.  11 10:33 package.json
-rw-r--r--  1 lassaad lassaad 25153 nov.  11 10:33 package-lock.json
drwxr-xr-x  5 lassaad lassaad  4096 nov.  11 10:32 public
drwxr-xr-x  2 lassaad lassaad  4096 nov.  11 10:32 routes
drwxr-xr-x  2 lassaad lassaad  4096 nov.  11 10:32 views
lassaad@my-leeds-ws:~/git_formation/proj02$
```

Git status aprés avoir créé les dossier et fichier via express

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
Sur la branche master

Aucun commit

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

        app.js
        bin/
        node_modules/
        package-lock.json
        package.json
        public/
        routes/
        views/

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez "git add" pour les suivre)
lassaad@my-leeds-ws:~/git_formation/proj02$
```

Git status -s

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git status -s
?? app.js
?? bin/
?? node_modules/
?? package-lock.json
?? package.json
?? public/
?? routes/
?? views/
lassaad@my-leeds-ws:~/git_formation/proj02$
```

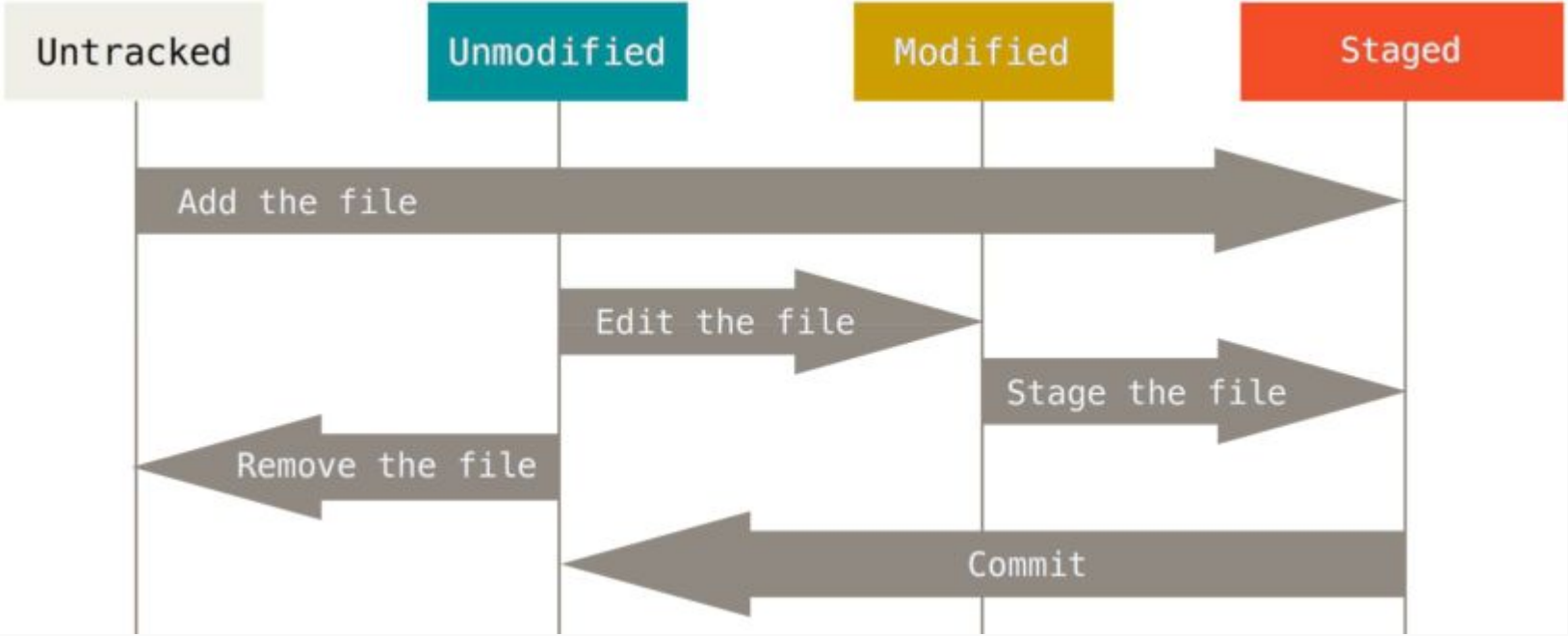
2. Les bases de GIT

2.2. Enregistrer des modifications dans le dépôt

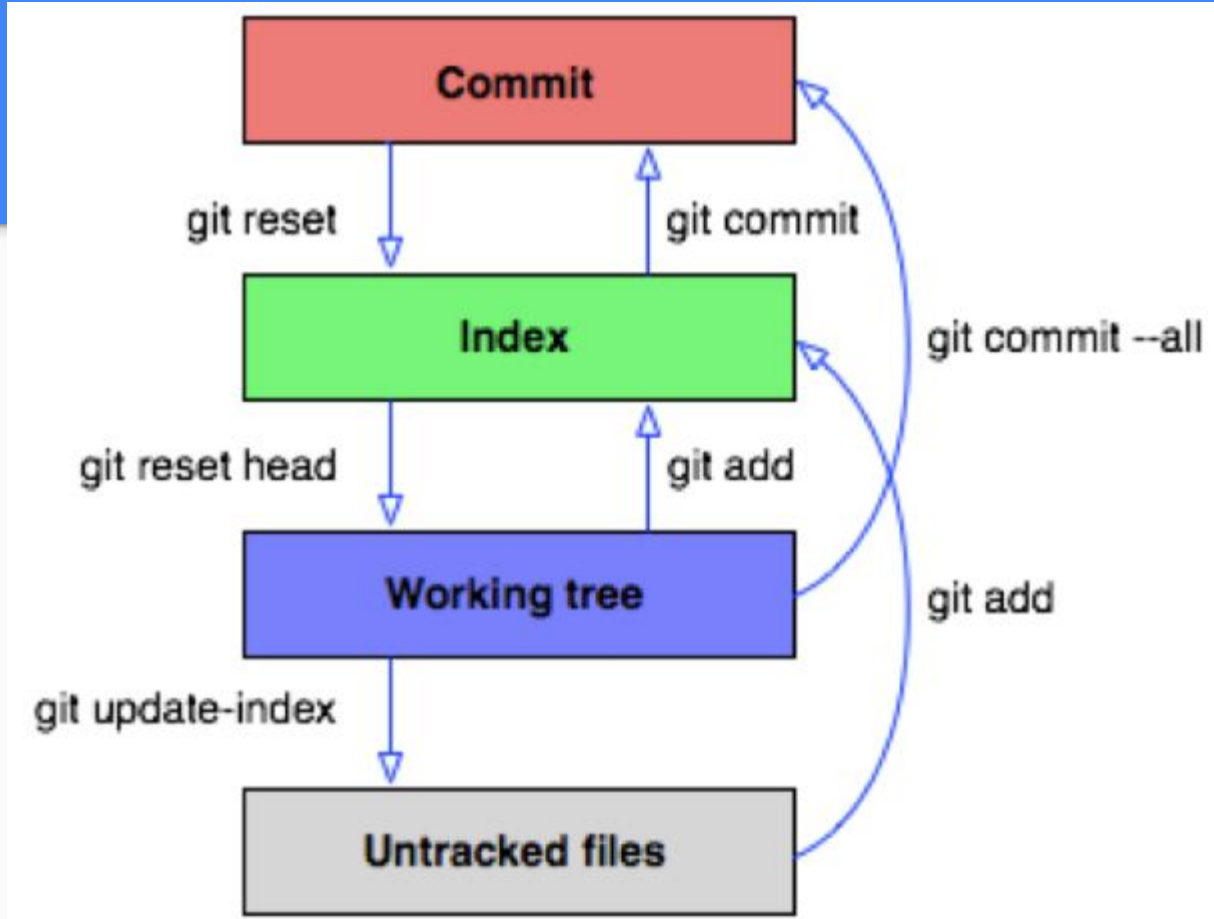
Plan

- Le fonctionnement
- Vérifier l’état des fichiers
- Placer de nouveaux fichiers sous suivi de version
- Indexer des fichiers modifiés
- Ignorer des fichiers
- Inspecter les modifications indexées et non indexées
- Valider vos modifications
- Passer l’étape de mise en index
- Effacer ou déplacer des fichiers

Le fonctionnement



Le fonctionnement



Clonage d’un nouveau projet à partir d’un dépôt distant (githubb, bitbuck, gitlab, etc.).
Le clonage obtient l’ensemble de la base de données avec toutes les actions et les instantanés enregistrées.

```
$ git status
```

La branche est master est celle par défaut
Rien à valider ⇒ Aucune modification en instance de validation, tout a été commité (Validé).

- Si on ajoute un nouveau fichier dans le projet, git status nous donne son nouvel état.
- Pour avoir un affichage plus concis :

```
$ git status -s
```

Placer de nouveaux fichiers sous suivi de version

• Pour commencer à suivre un nouveau fichier, il faut utiliser la commande :

```
$ git add <fichier>
```

• En faisant un `git status`, nous voyons que le fichier a changé d'état, le fichier est indexé dans Git.

• La version du fichier à l'instant où vous l'enregistrer avec `git add` sera celle qui sera dans l'historique des instantanés.

• On peut ajouter des répertoires, git ajoutera à l'index tout le répertoire récursivement.

Indexer des fichiers modifiés

- Maintenant, modifions un fichier qui est déjà sous suivi de version.
- Puis observons le status avec ‘`git status`’.
- Le fichier modifié a le status “`Modifications qui ne seront pas validées`”, le fichier n’est pas indexé.
- Pour indexer le fichier :

```
$ git add <fichier>
```

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
Sur la branche master

Aucun commit

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

        app.js
        bin/
        node_modules/
        package-lock.json
        package.json
        public/
        routes/
        views/

aucune modification ajoutée à la
lassaad@my-leeds-ws:~/git_formati
```

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git add app.js
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
Sur la branche master

Aucun commit

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

        nouveau fichier : app.js

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

        bin/
        node_modules/
        package-lock.json
        package.json
        public/
        routes/
        views/
```

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
Sur la branche master

Aucun commit

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

        nouveau fichier : app.js

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

        bin/
        node_modules/
        package-lock.json
        package.json
        public/
        routes/
        views/

lassaad@my-leeds-ws:~/git_formation/proj02$ git add bin
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
Sur la branche master

Aucun commit

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

        nouveau fichier : app.js
        nouveau fichier : bin/www

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

        node_modules/
        package-lock.json
        package.json
        public/
        routes/
        views/
```

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git status -s
A  app.js
?? bin/
?? node_modules/
?? package-lock.json
?? package.json
?? public/
?? routes/
?? views/
lassaad@my-leeds-ws:~/git_formation/proj02$
```

Ignorer des fichiers

- Dans les projets, on a souvent des fichiers que nous ne voulons pas ajouter à Git.
- Nous avons la possibilité de ne jamais les indexer avec ‘git add’.
- Sinon, nous pouvons indiquer à Git de les ignorer.
- Il faut rentrer la liste des fichiers dans le fichier .gitignore
- Les fichiers n'apparaîtront plus dans ‘git status’.

```
$ cat .gitignore
*.[oa]
*~
```

Ignorer les fichiers

• Les règles de construction des patrons à placer dans le fichier .gitignore sont les suivantes :
Les lignes vides ou commençant par `#` sont ignorées.
Les patrons standards de fichiers sont utilisables.
Si le patron se termine par une barre oblique (`/`), il indique un répertoire.
Un patron commençant par un point d'exclamation `(!)` indique des fichiers à inclure malgré les autres règles.
• Exemple de fichiers gitignore :
https://github.com/github/gitignore

gitignore - exemple

```
# pas de fichier .a
*.a

# mais suivre lib.a malgré la règle précédente
!lib.a

# ignorer uniquement le fichier TODO à la racine du projet
/TODO

# ignorer tous les fichiers dans le répertoire build
build/

# ignorer doc/notes.txt, mais pas doc/server/arch.txt
doc/*.txt

# ignorer tous les fichiers .txt sous le répertoire doc/
doc/**/*.txt
```

Inspecter les modifications indexées et non indexées

• Pour une vision précise des changements faits sur les fichiers, on utilise ‘`git diff`’.
• Cette commande répond aux questions :
Qu’est-ce qui a été modifié, mais pas encore indexé ?
Quelle modification a été indexée et est prête pour la validation ?
• `git diff` montre les changements ligne par ligne
• Pour voir les modifications qui font partie de la prochaine validation, vous pouvez utiliser ‘`git diff --cached`’

Valider les modifications

- Une fois que votre index est dans l'état désiré, nous pouvons valider les modifications.
- Rappel, ce qui n'est pas passé dans '`git add`' ne fera pas partie de la prochaine validation. Ils resteront en modifier sur votre disque.
- La commande pour valider est '`git commit`'
- Cette action lance l'éditeur de texte par défaut.
- Les options :
  - **-v** : ajoute le diff dans le commentaire
  - **-m 'my comment'** : Pour ajouter directement un commentaire sans passer par l'éditeur de texte.
- Une fois la validation faite, le commit a un id (SHA-1)

```
lassaad@my-leeds-ws:~/git_formation/proj02$ vim file2.txt
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        nouveau fichier : file2.txt

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 :         file2.txt

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

        file1.txt

lassaad@my-leeds-ws:~/git_formation/proj02$ git commit -a -m "commit all pour tester le file1 et file2"
[master dbf699a] commit all pour tester le file1 et file2
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt
lassaad@my-leeds-ws:~/git_formation/proj02$ git status
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        file1.txt

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez "git a
lassaad@my-leeds-ws:~/git_formation/proj02$
```

Passer l’étape de mise en index

- La gestion de la zone d’index peut être longue si on ajoute tous les fichiers un par un.
- `git commit -a` dit à git de placer tous les fichiers **déjà suivis** dans la zone d’index.

Effacer ou déplacer des fichiers

• Pour effacer un fichier de Git, vous devez l'éliminer des fichiers en suivi de version (plus précisément, l'effacer dans la zone d'index) puis valider.

• Si vous effacez le fichier avec '`rm <fichier>`', il apparaît sous la section « `Modifications qui ne seront pas validées` » (c'est-à-dire, non indexé).

• Avec un '`git rm <fichier>`', l'effacement du fichier est indexé.

Effacer ou déplacer des fichiers

• À la différence des autres VCS, Git ne suit pas explicitement les mouvements des fichiers, même s'il sera capable de voir qu'un fichier a été déplacé.

• Il existe la commande '`git mv <source> <destination>`' qui revient exactement à :

```
$> mv <source> <destination>'
```

```
$> git rm <source>
```

```
$> git add <destination>'
```

2. Les bases de GIT

2.3. Visualiser l’historique des validations

Plan

- Visualiser l’historique
- Les options principales
- L’option ‘pretty’
- Limiter la longueur de l’historique

Git clone || git log

- Pour voir l'historique, il existe la commande 'git log'
- Pour tester :

```
$ git clone --branch v6.0.0 https://github.com/laravel/laravel proj_laravelv6.0.0
```

- Par défaut, git log invoqué sans argument énumère en ordre chronologique inversé les commits réalisés.

Git log - exemple

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git log -2
commit dbf699a3bc6833e2448c1c6632c48d3a7043f658 (HEAD -> master)
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:57:57 2019 +0100

    commit all pour tester le file1 et file2

commit 616c36d7e4f337a8301e1602887c6370c5170059
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:53:02 2019 +0100

    commit pour le port 500
lassaad@my-leeds-ws:~/git_formation/proj02$
```

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git log -2
commit dbf699a3bc6833e2448c1c6632c48d3a7043f658 (HEAD -> master)
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:57:57 2019 +0100

    commit all pour tester le file1 et file2

commit 616c36d7e4f337a8301e1602887c6370c5170059
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:53:02 2019 +0100

    commit pour le port 500
lassaad@my-leeds-ws:~/git_formation/proj02$ git log -2 -p
commit dbf699a3bc6833e2448c1c6632c48d3a7043f658 (HEAD -> master)
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:57:57 2019 +0100

    commit all pour tester le file1 et file2

diff --git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..550eb6e
--- /dev/null
+++ b/file2.txt
@@ -0,0 +1 @@
+fichier 2 modifier test

commit 616c36d7e4f337a8301e1602887c6370c5170059
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:53:02 2019 +0100

    commit pour le port 500

diff --git a/bin/www b/bin/www
index 134f338..c93dd59 100755
--- a/bin/www
+++ b/bin/www
@@ -2,7 +2,7 @@
 var debug = require('debug')('my-application');
 var app = require('../app');

-app.set('port', process.env.PORT || 4000);
+app.set('port', process.env.PORT || 5000);

 var server = app.listen(app.get('port'), function() {
   debug('Express server listening on port ' + server.address().port);
lassaad@my-leeds-ws:~/git_formation/proj02$
```

```
lassaad@my-leeds-ws:~/git_formation/proj02$ git log --stat
commit dbf699a3bc6833e2448c1c6632c48d3a7043f658 (HEAD -> master)
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:57:57 2019 +0100

    commit all pour tester le file1 et file2

 file2.txt | 1 +
 1 file changed, 1 insertion(+)

commit 616c36d7e4f337a8301e1602887c6370c5170059
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:53:02 2019 +0100

    commit pour le port 500

 bin/www | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 65b1ab1a565e77e374901aef805d2af855137251
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 11:42:47 2019 +0100

    valider gitignore

 .gitignore                                         |   104 +
 app.js                                             |    59 +
 bin/www                                            |     9 +
 node_modules/.bin/jade                             |     1 +
```

```
lassaad@my-leeds-ws:~/repos/simplegit-progit$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master, origin/HEAD)
Author: Scott Chacon <schacon@gmail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the verison number

 Rakefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

 lib/simplegit.rb | 5 -----
 1 file changed, 5 deletions(-)

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit

 README           |  6 ++++++
 Rakefile         | 23 +++++++++++++++++++++++
 lib/simplegit.rb | 25 +++++++++++++++++++++++++
 3 files changed, 54 insertions(+)
lassaad@my-leeds-ws:~/repos/simplegit-progit$
```

l'option `--stat` affiche sous chaque entrée de validation :

- \+ une liste des fichiers modifiés,
- \+ combien de fichiers ont été changés
- \+ combien de lignes ont été ajoutées ou retirées dans ces fichiers.

Elle ajoute un résumé des informations en fin de sortie.

Les options principales de git log

- `git log` dispose d'un très grand nombre d'options permettant de paramétrer exactement ce que l'on cherche à voir :

Option	Description
`-p`	Affiche le patch appliqué par chaque commit
`--stat`	Affiche les statistiques de chaque fichier pour chaque commit
`--shortstat`	N'affiche que les ligne modifiées/insérées/effacées de l'option --stat
`--name-only`	Affiche la liste des fichiers modifiés après les informations du commit
`--name-status`	Affiche la liste des fichiers affectés accompagnés des informations d'ajout/modification/suppression
`--abbrev-commit`	N'affiche que les premiers caractères de la somme de contrôle SHA-1
`--relative-date`	Affiche la date en format relatif (par exemple "2 weeks ago" : il y a deux semaines) au lieu du format de date complet
`--graph`	Affiche en caractères ASCII le graphe de branches et fusions en vis-à-vis de l'historique
`--pretty`	Affiche les *commits* dans un format alternatif. Les formats incluent `oneline`, `short`, `full`, `fuller`, et `format` (où on peut spécifier son propre format)
`--oneline`	Option de convenance correspondant à `--pretty=oneline --abbrev-commit`

git log -4 ⇒ Appliqué au dépôt laravel/laravel dans github

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -4
commit bfd4b1e92f7c6b4e6b74cfdde995a5afad648d96 (HEAD -> master, origin/master, origin/HEAD)
Author: Graham Campbell <graham@alt-three.com>
Date:   Fri Nov 1 11:51:17 2019 +0000

    Update .styleci.yml

commit 953b488b8bb681d4d6e12227645c7c1b7ac26935 (tag: v6.4.0)
Author: Taylor Otwell <taylor@laravel.com>
Date:   Mon Oct 21 13:47:27 2019 -0500

    fix key

commit ba2f2abe830f5d03c52fd9c88411859cf863abd6
Author: Taylor Otwell <taylor@laravel.com>
Date:   Mon Oct 21 13:42:31 2019 -0500

    tweak formatting

commit ace38c133f3d8088fc7477f56b9db6fdc0098d06
Author: Michael Chernyshev <chmv-git@allnetic.com>
Date:   Fri Oct 18 13:57:19 2019 +0300

    Security fix: Waiting before retrying password reset
lassaad@my-leeds-ws:~/repos/laravel$
```

Git log -4 --pretty=oneline/short

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -4 --pretty=oneline
bfd4b1e92f7c6b4e6b74cfdde995a5afad648d96 (HEAD -> master, origin/master, origin/HEAD) Update .styleci.yml
953b488b8bb681d4d6e12227645c7c1b7ac26935 (tag: v6.4.0) fix key
ba2f2abe830f5d03c52fd9c88411859cf863abd6 tweak formatting
ace38c133f3d8088fc7477f56b9db6fdc0098d06 Security fix: Waiting before retrying password reset
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -4 --pretty=short
commit bfd4b1e92f7c6b4e6b74cfdde995a5afad648d96 (HEAD -> master, origin/master, origin/HEAD)
Author: Graham Campbell <graham@alt-three.com>

    Update .styleci.yml

commit 953b488b8bb681d4d6e12227645c7c1b7ac26935 (tag: v6.4.0)
Author: Taylor Otwell <taylor@laravel.com>

    fix key

commit ba2f2abe830f5d03c52fd9c88411859cf863abd6
Author: Taylor Otwell <taylor@laravel.com>

    tweak formatting

commit ace38c133f3d8088fc7477f56b9db6fdc0098d06
Author: Michael Chernyshev <chmv-git@allnetic.com>

    Security fix: Waiting before retrying password reset
lassaad@my-leeds-ws:~/repos/laravel$
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -10 --pretty=full
commit bfd4b1e92f7c6b4e6b74cfdde995a5afad648d96 (HEAD -> master, origin/master, origin/HEAD)
Author: Graham Campbell <graham@alt-three.com>
Commit: Graham Campbell <graham@alt-three.com>

    Update .styleci.yml

commit 953b488b8bb681d4d6e12227645c7c1b7ac26935 (tag: v6.4.0)
Author: Taylor Otwell <taylor@laravel.com>
Commit: Taylor Otwell <taylor@laravel.com>

    fix key

commit ba2f2abe830f5d03c52fd9c88411859cf863abd6
Author: Taylor Otwell <taylor@laravel.com>
Commit: Taylor Otwell <taylor@laravel.com>

    tweak formatting
commit 400df0b02bcc0e3fc8bc1c75ea494242c3f392af
Author: Gert de Pagter <BackEndTea@users.noreply.github.com>
Commit: Taylor Otwell <taylor@laravel.com>

    Add xml schema to phpunit (#5139)

    This allows an IDE to do auto completion, and show any errors in the configuration

commit bb969c61d41ec479adbe4a6da797831002b75092
Author: Nuno Maduro <enunomaduro@gmail.com>
Commit: Taylor Otwell <taylor@laravel.com>

    Fixes required version of the framework within `composer.json` (#5130)
```

L'option pretty

L'option `--pretty` permet d'avoir des affichages différents :

`$ git log -pretty=oneline` :

Affichage d'un `commit` par ligne

`$ git log -pretty=format` : '...' :

Permet de créer un format spécifique

Option	Description du formatage
%H	Somme de contrôle du commit
%h	Somme de contrôle abrégée du commit
%T	Somme de contrôle de l'arborescence
%t	Somme de contrôle abrégée de l'arborescence
%P	Sommes de contrôle des parents
%p	Sommes de contrôle abrégées des parents
%an	Nom de l'auteur
%ae	E-mail de l'auteur
%ad	Date de l'auteur (au format de l'option -date=)
%ar	Date relative de l'auteur
%cn	Nom du validateur
%ce	E-mail du validateur
%cd	Date du validateur
%cr	Date relative du validateur
%s	Sujet

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -10 --pretty=oneline
bfd4b1e92f7c6b4e6b74cfdde995a5afad648d96 (HEAD -> master, origin/master, origin/HEAD) Update .styleci.yml
953b488b8bb681d4d6e12227645c7c1b7ac26935 (tag: v6.4.0) fix key
ba2f2abe830f5d03c52fd9c88411859cf863abd6 tweak formatting
ace38c133f3d8088fc7477f56b9db6fdc0098d06 Security fix: Waiting before retrying password reset
400df0b02bcc0e3fc8bc1c75ea494242c3f392af Add xml schema to phpunit (#5139)
bb969c61d41ec479adbe4a6da797831002b75092 Fixes required version of the framework within `composer.json` (#5130)
39c28801e8d8a8cfc99c3eed4756c6acc7367e0c Update CHANGELOG.md
9bc23ee468e1fb3e5b4efccdc35f1fcee5a8b6bc (tag: v6.2.0) formatting
d1f7a5a886039e28a434905447865ca952032284 formatting
bc82317a02e96105ad9f3cc0616d491cb5585c62 Merge branch 'password-confirmation' of https://github.com/driesvints/laravel into driesvints-password-c
onfirmation
lassaad@my-leeds-ws:~/repos/laravel$
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -10 --pretty=format:'%h || %an || %s :: %cn'
bfd4b1e9 || Graham Campbell || Update .styleci.yml :: Graham Campbell
953b488b || Taylor Otwell || fix key :: Taylor Otwell
ba2f2abe || Taylor Otwell || tweak formatting :: Taylor Otwell
ace38c13 || Michael Chernyshev || Security fix: Waiting before retrying password reset :: Michael Chernyshev
400df0b0 || Gert de Pagter || Add xml schema to phpunit (#5139) :: Taylor Otwell
bb969c61 || Nuno Maduro || Fixes required version of the framework within `composer.json` (#5130) :: Taylor Otwell
39c28801 || Dries Vints || Update CHANGELOG.md :: Dries Vints
9bc23ee4 || Taylor Otwell || formatting :: Taylor Otwell
d1f7a5a8 || Taylor Otwell || formatting :: Taylor Otwell
bc82317a || Taylor Otwell || Merge branch 'password-confirmation' of https://github.com/driesvints/laravel into driesvints-password-confirmation
:: Taylor Otwell
lassaad@my-leeds-ws:~/repos/laravel$
```

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Added a method for getting the current branch.
* | 30e367c timeout code and tests
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -50 --pretty=format:'%h || %an || %s ' --graph
* bfd4b1e9 || Graham Campbell || Update .styleci.yml
* 953b488b || Taylor Otwell || fix key
* ba2f2abe || Taylor Otwell || tweak formatting
* ace38c13 || Michael Chernyshev || Security fix: Waiting before retrying password reset
* 400df0b0 || Gert de Pagter || Add xml schema to phpunit (#5139)
* bb969c61 || Nuno Maduro || Fixes required version of the framework within `composer.json` (#5130)
* 39c28801 || Dries Vints || Update CHANGELOG.md
* 9bc23ee4 || Taylor Otwell || formatting
* d1f7a5a8 || Taylor Otwell || formatting
*   bc82317a || Taylor Otwell || Merge branch 'password-confirmation' of https://github.com/driesvints/laravel into d
|\
| * ba3aae6c || Dries Vints || Implement password confirmation
| * 4036f174 || Dries Vints || Remove middleware from password reset
| * 050c1d88 || Dries Vints || Add new password rule language line
* | 51a1297a || Sangrak Choi || [6.x] Added OP.GG sponsor (#5121)
* | c70c986e || Roger Vilà || [6.x] Add 'null' logging channel (#5106)
* | 42e864f3 || Tim MacDonald || remove testing bootstrap extension (#5107)
* | cba8d19f || James Merrix || Added Appoly sponsor (#5105)
* | 56157b9c || Graham Campbell || Revert "According to PHP Bug 78516 Argon2 requires at least 8KB (#5097)" (#5102)
|/
* 7d728506 || Dries Vints || Update CHANGELOG.md
```

Git log

Option	Description
`-(n)`	N'affiche que les n derniers *commits*
`--since`, `--after`	Limite l'affichage aux *commits* réalisés après la date spécifiée
`--until`, `--before`	Limite l'affichage aux *commits* réalisés avant la date spécifiée
`--author`	Ne montre que les *commits* dont le champ auteur correspond à la chaîne passée en argument
`--committer`	Ne montre que les *commits* dont le champ validateur correspond à la chaîne passée en argument
`--grep`	Ne montre que les *commits* dont le message de validation contient la chaîne de caractères
`-S`	Ne montre que les *commits* dont les ajouts ou retraits contient la chaîne de caractères

2. Les bases de GIT

2.4. Annuler des actions

Plan

- Modifier un commit
- Désindexer un fichier déjà indexé
- Réinitialiser un fichier modifié

Modifier un commit

- ⇒ **Attention, certaines modifications sont permanentes.**
- Il peut arriver que l'on valide une modification trop tôt en oubliant des fichiers.
- Pour modifier un commit :

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

- Cette commande va reprendre l'index du commit précédent.
- L'éditeur s'ouvre avec les modifications et le message.
- Il est possible de modifier uniquement le message du commit.

Désindexer un fichier déja indexé

- Imaginons que vous avez indexé trop de fichiers.

```
$ git reset HEAD <file>
```

Réinitialiser un fichier modifié

• Pour faire revenir un fichier au niveau du précédent checkout :

```
$> git checkout -- <fichier>
```

• Attention, vous perdrez définitivement les modifications apportées au fichier depuis le dernier checkout.

• Si vous souhaitez seulement écarter momentanément cette modification, nous verrons comment mettre de côté et créer des branches

2. Les bases de GIT

2.4. Travailler avec des dépôts distants

Plan

- Introduction
- Afficher les dépôts distants
- Afficher les dépôts distants
- Récupérer et tirer depuis des dépôts distants
- Pousser son travail sur un dépôt distant
- Inspecter un dépôt distant
- Retirer et renommer des dépôts distants

Introduction

- Pour pouvoir collaborer sur un projet Git, il est nécessaire de savoir comment gérer les dépôts distants.
- Les dépôts distants sont des versions de votre projet qui sont hébergées sur Internet ou le réseau d'entreprise.
- Vous pouvez en avoir plusieurs, pour lesquels vous pouvez avoir des droits soit en lecture seule, soit en lecture/écriture.
- Nous allons voir comment :
    - Ajouter des dépôts distants.
    - Effacer des dépôts distants.

Afficher les dépôts distants

• Pour visualiser les serveurs distants que vous avez enregistré, vous pouvez lancer la commande ‘`git remote`’.

• Si vous avez cloné notre projet, vous devez avoir au moins un dépôt distant : ‘`origin`’.

• Vous pouvez aussi spécifier ‘`-v`’, qui vous montre l’URL que Git a stocké pour chaque nom court.

Ajouter des dépôts distants

```
lassaad@my-leeds-ws:~/repos/laravel$ git remote -v
lassaadBaatiAhmed       https://github.com/lassaadBaatiAhmed/projet.git (fetch)
lassaadBaatiAhmed       https://github.com/lassaadBaatiAhmed/projet.git (push)
origin  https://github.com/laravel/laravel.git (fetch)
origin  https://github.com/laravel/laravel.git (push)
lassaad@my-leeds-ws:~/repos/laravel$
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git remote add myorigin https://github.com/lassaadbaati/laravel.git
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$ git remote
lassaadBaatiAhmed
myorigin
origin
lassaad@my-leeds-ws:~/repos/laravel$ git remote -v
lassaadBaatiAhmed       https://github.com/lassaadBaatiAhmed/projet.git (fetch)
lassaadBaatiAhmed       https://github.com/lassaadBaatiAhmed/projet.git (push)
myorigin        https://github.com/lassaadbaati/laravel.git (fetch)
myorigin        https://github.com/lassaadbaati/laravel.git (push)
origin  https://github.com/laravel/laravel.git (fetch)
origin  https://github.com/laravel/laravel.git (push)
lassaad@my-leeds-ws:~/repos/laravel$
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git push myorigin master
Username for 'https://github.com': lassaad.baati@gmail.com
Password for 'https://lassaad.baati@gmail.com@github.com':
Décompte des objets: 30751, fait.
Delta compression using up to 8 threads.
Compression des objets: 100% (11870/11870), fait.
Écriture des objets: 100% (30751/30751), 9.61 MiB | 74.00 KiB/s, fait.
Total 30751 (delta 18199), reused 30751 (delta 18199)
remote: Resolving deltas: 100% (18199/18199), done.
To https://github.com/lassaadbaati/laravel.git
 * [new branch]        master -> master
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$ git branch
* master
```

Comparaison entre le dépôt officiel laravel/laravel & celui lassaadbaati/laravel

```
lassaad@my-leeds-ws:~/repos/laravel$ git remote show origin
* distante origin
  URL de rapatriement : https://github.com/laravel/laravel.git
  URL push : https://github.com/laravel/laravel.git
  Branche HEAD : master
  Branches distantes :
    3.0     suivi
    5.0     suivi
    5.1     suivi
    5.2     suivi
    5.3     suivi
    5.4     suivi
    5.5     suivi
    5.6     suivi
    5.7     suivi
    5.8     suivi
    develop suivi
    master  suivi
  Branche locale configurée pour 'git pull' :
    master fusionne avec la distante master
  Référence locale configurée pour 'git push' :
    master pousse vers master (peut être mis à jour en avance rapide)
lassaad@my-leeds-ws:~/repos/laravel$ git remote show myorigin
* distante myorigin
  URL de rapatriement : https://github.com/lassaadbaati/laravel.git
  URL push : https://github.com/lassaadbaati/laravel.git
  Branche HEAD : master
  Branche distante :
    master suivi
  Référence locale configurée pour 'git push' :
    master pousse vers master (à jour)
lassaad@my-leeds-ws:~/repos/laravel$
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git status
Sur la branche master
Votre branche est à jour avec 'origin/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 :         readme.md

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

```
lassaad@my-leeds-ws:~/repos/laravel$ git commit -a -m "Ajouter from lassaad au fichier readme.md"
[master 68727e2c] Ajouter from lassaad au fichier readme.md
 1 file changed, 1 insertion(+), 1 deletion(-)
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$ git status
Sur la branche master
Votre branche est en avance sur 'origin/master' de 1 commit.
  (utilisez "git push" pour publier vos commits locaux)

rien à valider, la copie de travail est propre
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -1 -p
commit 68727e2cd17003e8f82f77471ae678af0ca76e43 (HEAD -> master)
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 16:09:36 2019 +0100

    Ajouter from lassaad au fichier readme.md

diff --git a/readme.md b/readme.md
index 73dddea2..8aed06b4 100644
--- a/readme.md
+++ b/readme.md
@@ -7,7 +7,7 @@
 <a href="https://packagist.org/packages/laravel/framework"><img src="https://poser.pugx.org/laravel/framework/license.svg" alt="License"></a>
 </p>

-## About Laravel
+## About Laravel from Lassaad

 Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience t
ng. Laravel takes the pain out of development by easing common tasks used in many web projects, such as:

lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$ git push myorigin master
Username for 'https://github.com': lassaad.baati@gmail.com
Password for 'https://lassaad.baati@gmail.com@github.com':
Décompte des objets: 3, fait.
Delta compression using up to 8 threads.
Compression des objets: 100% (3/3), fait.
Écriture des objets: 100% (3/3), 330 bytes | 330.00 KiB/s, fait.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/lassaadbaati/laravel.git
   bfd4b1e9..68727e2c  master -> master
```

Récupérer et tirer depuis des dépôts distants

• Pour récupérer les données d'un dépôt distant :

```
$ git fetch [nom du repos distant]
```

• Cette commande s'adresse au dépôt distant et récupère toutes les données de ce projet que vous ne possédez pas déjà.

• Si vous clonez un dépôt, le dépôt distant est automatiquement ajouté sous le nom « `origin` ».

• Si le dépôt est différent que celui ajouté par défaut, les branches seront ajoutées dans '`<nom du dépôt distant>/master`'.

Pousser son travail sur un dépôt distant

• Lorsque votre dépôt vous semble prêt à être partagé, vous pouvez le pousser à votre dépôt distant :

```
$ git push [dépôt distant] [nom-de-la-branche]
```

```
$ git push origin master
```

Inspecter un dépôt distant

• Pour avoir plus d'informations sur un dépôt distant :

```
$ git remote show [dépôt distant]
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git remote show origin
* distante origin
  URL de rapatriement : https://github.com/laravel/laravel.git
  URL push : https://github.com/laravel/laravel.git
  Branche HEAD : master
  Branches distantes :
    3.0     suivi
    5.0     suivi
    5.1     suivi
    5.2     suivi
    5.3     suivi
    5.4     suivi
    5.5     suivi
    5.6     suivi
    5.7     suivi
    5.8     suivi
    develop suivi
    master  suivi
  Branche locale configurée pour 'git pull' :
    master fusionne avec la distante master
  Référence locale configurée pour 'git push' :
    master pousse vers master (peut être mis à jour en avance rapide)
lassaad@my-leeds-ws:~/repos/laravel$ git remote show myorigin
* distante myorigin
  URL de rapatriement : https://github.com/lassaadbaati/laravel.git
  URL push : https://github.com/lassaadbaati/laravel.git
  Branche HEAD : master
  Branche distante :
    master suivi
  Référence locale configurée pour 'git push' :
    master pousse vers master (à jour)
lassaad@my-leeds-ws:~/repos/laravel$
```

Retirer et renommer des dépôts distants

- Pour renommer une référence :

```
$ git remote rename myOrigin extOrigin
```

- Pour supprimer une référence :

```
$ git remote rm extOrigin
```

2. Les bases de GIT

2.5. Etiquetage

Plan

- Introduction
- Lister vos étiquettes
- Créer des étiquettes
- Les étiquettes annotées
- Les étiquettes légères
- Étiqueter après coup
- Partager les étiquettes
- Extraire une étiquette

Introduction

- À l'instar de la plupart des VCS, Git donne la possibilité d'étiqueter un certain état dans l'historique comme important.
- Généralement, on utilise cette fonctionnalité pour marquer les versions d'un projet (v1.0.0, v1.0.42, v2.0.1, ...)
- Nous verrons comment :
    - Lister les différentes étiquettes
    - Comment créer de nouvelles étiquettes
    - Voir tous les types d'étiquettes

Lister les étiquettes

- Pour lister les étiquettes :

```
$ git tag
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git tag
v3.0.0
v3.0.0-beta-2
v3.0.0-rc-2
v3.0.1
```

Les étiquettes sont listées par ordre alphabétique

- Pour faire une recherche :

```
$ git tag -l 'v6.0.*'
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git tag -l 'v6.0.*'
v6.0.0
v6.0.1
v6.0.2
lassaad@my-leeds-ws:~/repos/laravel$
```

Créer des étiquettes

- Git utilise deux types d’étiquettes :
    - Étiquette légère
    - Étiquette annotée
- Une étiquette légère ressemble beaucoup à une branche qui ne change pas, c’est juste un pointeur sur un commit spécifique.
- Les étiquettes annotées, par contre, sont stockées en tant qu’objets à part entière dans la base de données de Git

Les étiquettes annotées

- Créer des étiquettes annotées est simple avec Git :

```
$  git tag -a v1.1.0 -m 'Ma version 1.1.0'
```

- Puis pour afficher en détails votre nouveau tag :

```
$ git show v1.1.0
```

Les étiquettes légères

Une autre manière d'étiqueter les commits est d'utiliser les étiquettes légères. Celles-ci se réduisent à stocker la somme de contrôle d'un commit dans un fichier, aucune autre information n'est conservée. Pour créer une étiquette légère, il suffit de **n'utiliser aucune des options -a, -s ou -m** :

```
$  git tag v1.5.0.Test
```

```
$ git tag v1.4-lg
$ git tag
v0.1
v1.3
v1.4
v1.4-lg
v1.5
```

```
$ git show v1.4-lg
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git log -1 -p
commit 68727e2cd17003e8f82f77471ae678af0ca76e43 (HEAD -> master, myorigin/master)
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 16:09:36 2019 +0100

    Ajouter from lassaad au fichier readme.md

diff --git a/readme.md b/readme.md
index 73dddea2..8aed06b4 100644
--- a/readme.md
+++ b/readme.md
@@ -7,7 +7,7 @@
 <a href="https://packagist.org/packages/laravel/framework"><img src="https://poser.pugx.org/laravel/framework/license.svg" alt="License"></a>
 </p>

-## About Laravel
+## About Laravel from Lassaad

 Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to b
e truly fulfilling. Laravel takes the pain out of development by easing common tasks used in many web projects, such as:
```

```
lassaad@my-leeds-ws:~/repos/laravel$ git log --pretty=oneline -5
68727e2cd17003e8f82f77471ae678af0ca76e43 (HEAD -> master, myorigin/master) Ajouter from lassaad au fichier readme.md
bfd4b1e92f7c6b4e6b74cfdde995a5afad648d96 (origin/master, origin/HEAD) Update .styleci.yml
953b488b8bb681d4d6e12227645c7c1b7ac26935 (tag: v6.4.0) fix key
ba2f2abe830f5d03c52fd9c88411859cf863abd6 tweak formatting
ace38c133f3d8088fc7477f56b9db6fdc0098d06 Security fix: Waiting before retrying password reset
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$ git tag -a v3.3 68727e2cd17003e8f82f77471ae678af0ca76e43
lassaad@my-leeds-ws:~/repos/laravel$
lassaad@my-leeds-ws:~/repos/laravel$ git show v3.3
tag v3.3
Tagger: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 22:05:54 2019 +0100

Etiquette depuis Lassaad vers readme.md

commit 68727e2cd17003e8f82f77471ae678af0ca76e43 (HEAD -> master, tag: v3.3, myorigin/master)
Author: Lassaad BAATI <lassaad.baati@gmail.com>
Date:   Mon Nov 11 16:09:36 2019 +0100

    Ajouter from lassaad au fichier readme.md

diff --git a/readme.md b/readme.md
index 73dddea2..8aed06b4 100644
--- a/readme.md
+++ b/readme.md
@@ -7,7 +7,7 @@
 <a href="https://packagist.org/packages/laravel/framework"><img src="https://poser.pugx.org/
 </p>

-## About Laravel
+## About Laravel from Lassaad

 Laravel is a web application framework with expressive, elegant syntax. We believe developme
e truly fulfilling. Laravel takes the pain out of development by easing common tasks used in
```

Étiqueter un commit spécifique

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Fusion branche 'experimental'
a6b4c97498bd301d84096da251c98a07c7723e65 Début de l'écriture support
0d52aaab4479697da7686c15f77a3d64d9165190 Un truc de plus
6d52a271eda8725415634dd79daabbc4d9b6008e Fusion branche 'experimental'
0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc ajout d'une fonction de validatn
4682c3261057305bdd616e23b64b0857d832627b ajout fichier afaire
166ae0c4d3f420721acbb115cc33848dfcc2121a début de l'ecriture support
9fceb02d0ae598e95dc970b74767f19372d61af8 mise à jour rakefile
964f16d36dfccde844893cac5b347e7b3d44abbc validation afaire
8a5cbc430f1a9c3d00faaeffd07798508422908a mise à jour lisezmoi
```

Maintenant, supposons que vous avez oublié d'étiqueter le projet à la version v1.2 qui correspondait au commit « **mise à jour rakefile** ». Vous pouvez toujours le faire après l'événement. Pour étiqueter ce commit, vous spécifiez la somme de contrôle du commit (ou une partie) en fin de commande :

Partager les étiquettes

• Par défaut, la commande git push ne transfère pas les étiquettes vers les serveurs distants.

• Il faut explicitement pousser les étiquettes après les avoir créées localement.

```
$  git push origin v1.1.0
```

• Pour pouvoir pousser toutes les étiquettes que nous avons créé localement :

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

Extraire une étiquette

Il n'est pas vraiment possible d'extraire une étiquette avec Git, puisque les étiquettes ne peuvent pas être modifiées. Si vous souhaitez ressortir dans votre copie de travail une version de votre dépôt correspondant à une étiquette spécifique, le plus simple consiste à créer une branche à partir de cette étiquette :

```
$ git checkout -b version2 v2.0.0
Extraction des fichiers: 100% (602/602), fait.
Basculement sur la nouvelle branche 'version2'
```

Toute validation modifiera la **branche version2** par rapport à **l'étiquette v2.0.0** puisqu'elle avancera avec les nouvelles modifications. Il faut donc être prudent sur l'identification de cette branche.

Exercice 1.

Now that you have learned the basics of Git workflow, try running through this a couple of times on your own:

1. Créer un dossier nommé work_repo.
2. Placer vous dans le dossier work_repo.
3. Créer un fichier nommé “third.txt”.
4. Initialiser un dossier git vide.
5. Ajouter le fichier “third.txt” au “staging area”.
6. Appliquer un Commit avec le message "adding third.txt".
7. Vérifier le commit effectué avec git log.
8. Créer un autre fichier nommé “fourth.txt”.
9. Ajouter le fichier “fourth.txt” au “staging area”.
10. Appliquer un Commit avec le message "adding fourth.txt"
11. Supprimer le fichier “third.txt”
12. Ajouter ce changement au “staging area”.
13. Appliquer un Commit avec le message "removing third.txt".
14. Vérifier le commit effectué avec git log et ses différents paramètres.
15. Changer les paramètres globaux selon votre choix
16. Lister les paramètres de configuration git.

Les alias dans git

Git ne complète pas votre commande si vous ne la tapez que partiellement. Si vous ne voulez pas avoir à taper l'intégralité du texte de chaque commande, vous pouvez facilement définir un alias pour chaque commande en utilisant git config.

Ceci signifie que, par exemple, au lieu de taper git commit, vous n'avez plus qu'à taper git ci. Au fur et à mesure de votre utilisation de Git, vous utiliserez probablement d'autres commandes plus fréquemment.

```
lassaad@my-leeds-ws:~/repos/wc1$ git config --global alias.co checkout
lassaad@my-leeds-ws:~/repos/wc1$ git config --global alias.ci commit
lassaad@my-leeds-ws:~/repos/wc1$ git config --global alias.br branch
lassaad@my-leeds-ws:~/repos/wc1$ git config --global alias.st status
lassaad@my-leeds-ws:~/repos/wc1$
```

3. Les branches avec git

3.1. Introduction aux branches

Plan

- Introduction
- Créer une nouvelle branche
- Basculer entre les branches

Introduction

Git ne stocke pas ses données comme une série de modifications ou de différences successives mais plutôt comme une série d'instantanés (appelés **snapshots**).

Lorsque vous faites un commit, Git stocke un objet **commit** qui contient un pointeur vers l'instantané (**snapshot**) du contenu que vous avez indexé. Cet objet contient également les noms et prénoms de l'auteur, le message que vous avez renseigné ainsi que des pointeurs vers le ou les **commits** qui précèdent directement ce **commit** : aucun parent pour le **commit** initial, un parent pour un **commit** normal et de multiples parents pour un **commit** qui résulte de la fusion d'une ou plusieurs branches.

• Presque tous les VCS proposent une certaine forme de gestion de branches.

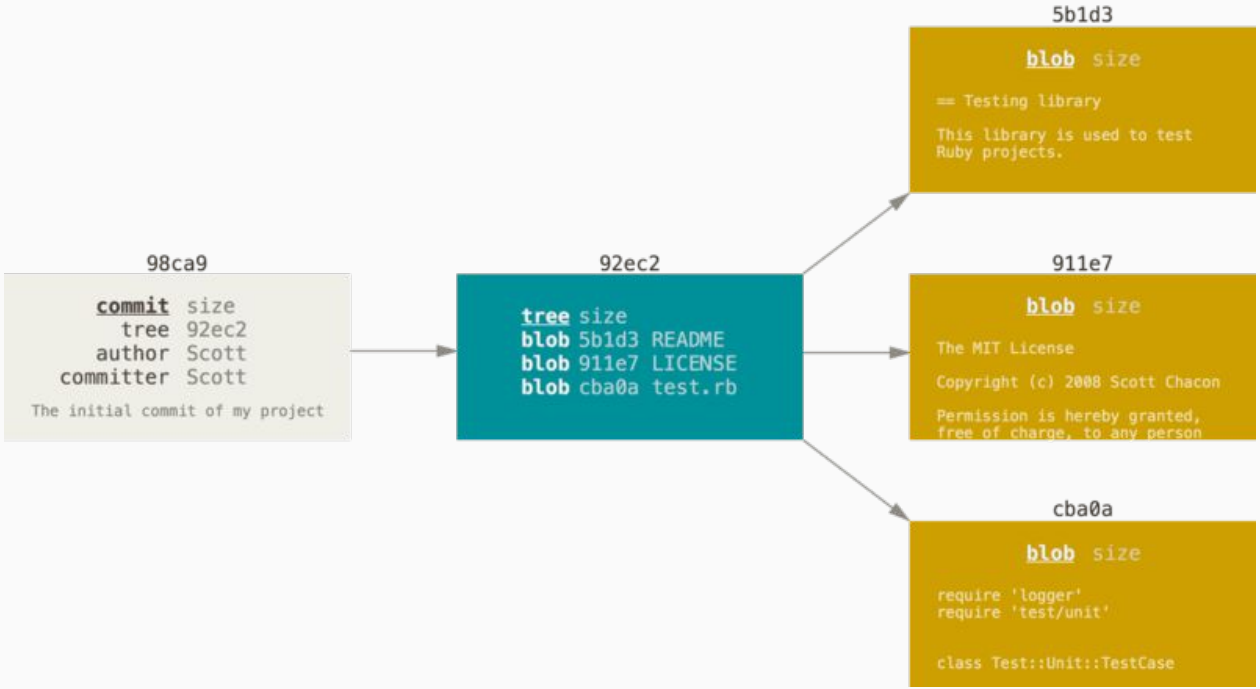
• Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne.

```
$  git add README test.rb LICENSE
```

```
$  git commit -m 'initial commit of my project'
```

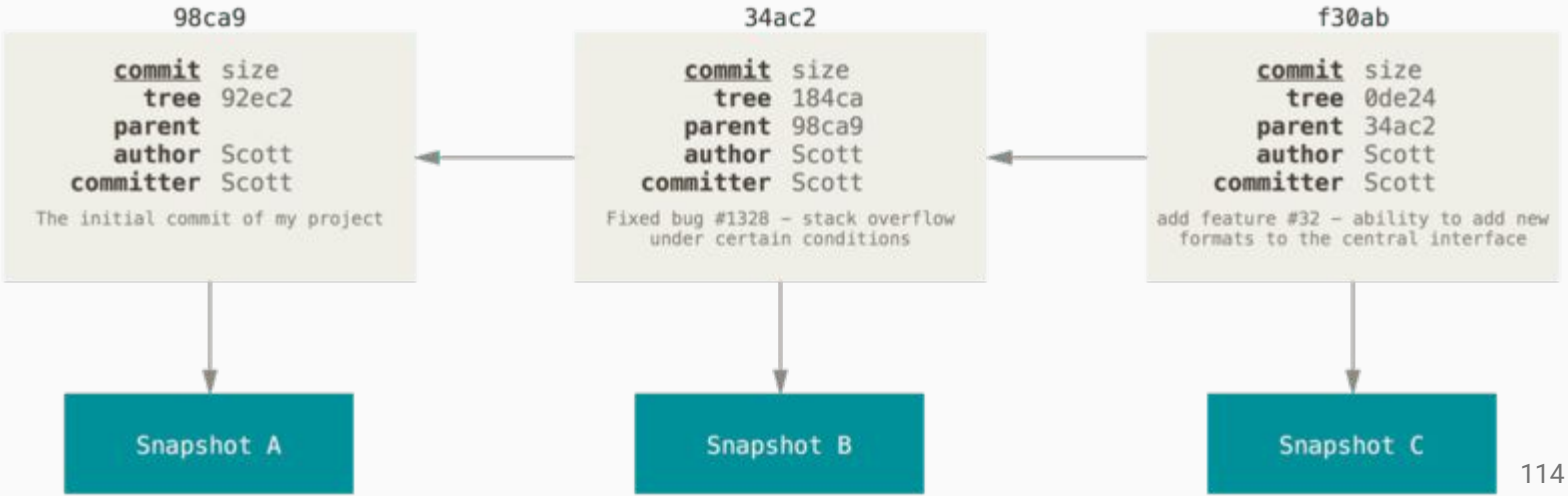
Un commit et son arbre

Votre dépôt Git contient à présent cinq objets :
un **blob** pour le contenu de chacun de vos trois fichiers,
un arbre (**tree**) qui liste le contenu du répertoire et spécifie quels noms de fichiers sont attachés à quels **blobs**
et enfin un objet **commit** portant le pointeur vers l'arbre de la racine ainsi que toutes les métadonnées attachées au **commit**.



Commit et leurs parents

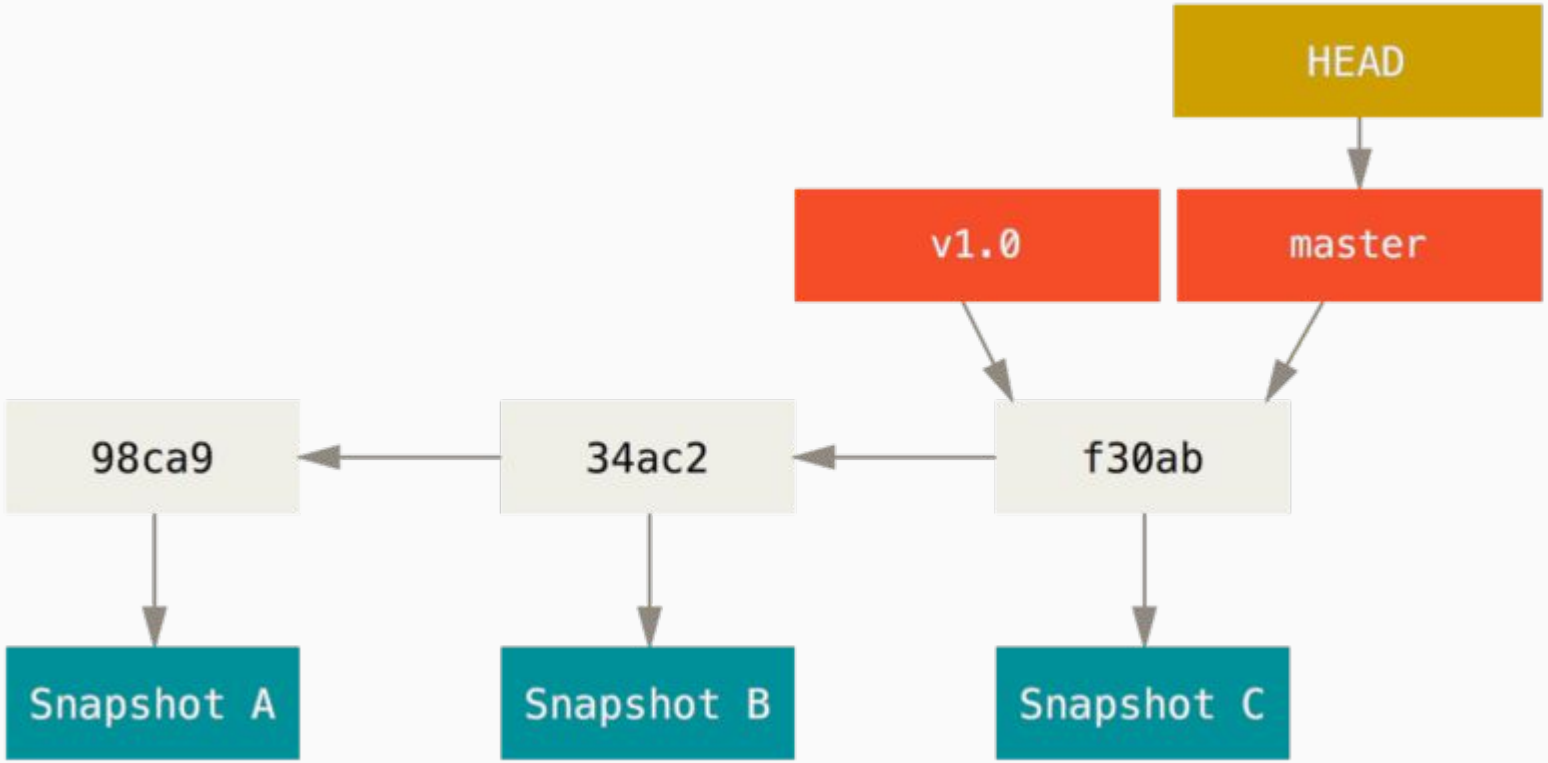
Si vous faites des modifications et validez à nouveau, le prochain commit stocke un pointeur vers le commit le précédant immédiatement.
⇒ Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits. La branche par défaut dans Git s'appelle master. Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés. À chaque validation, le pointeur de la branche master avance automatiquement.



Remarque

La branche **master** n’est pas une branche spéciale. Elle est identique à toutes les autres branches. La seule raison pour laquelle chaque dépôt en a une est que la commande `git init` la crée par défaut et que la plupart des gens ne s’embêtent pas à la changer.

Une branche et l'historique de ses commits



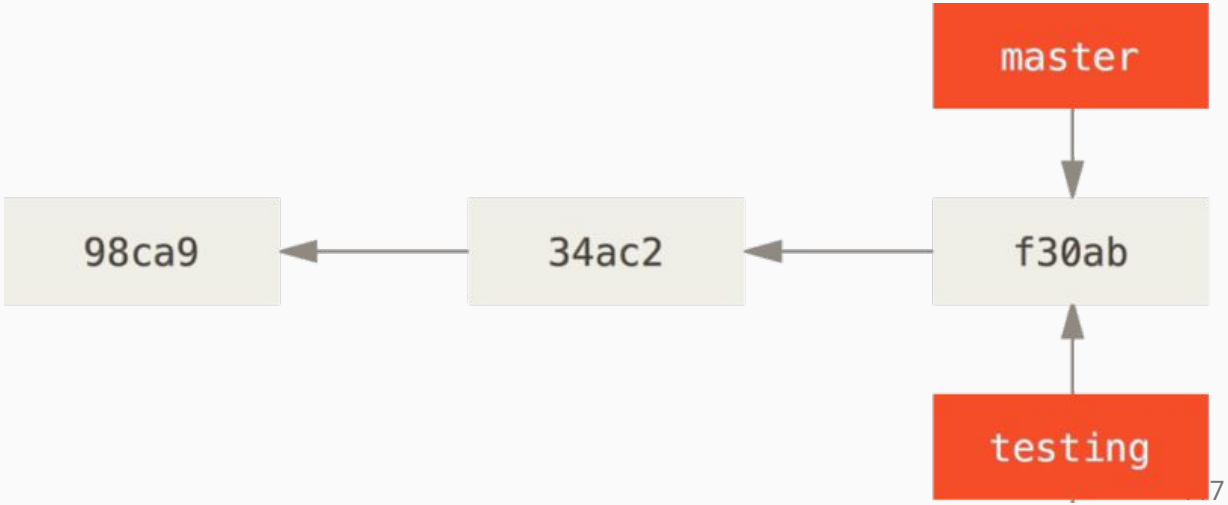
Créer une nouvelle branche

Créer une nouvelle branche

Que se passe-t-il si vous créez une nouvelle branche ? Eh bien, cela crée un nouveau pointeur pour vous. Supposons que vous créez une nouvelle branche nommée `test`. Vous utilisez pour cela la commande `git branch` :

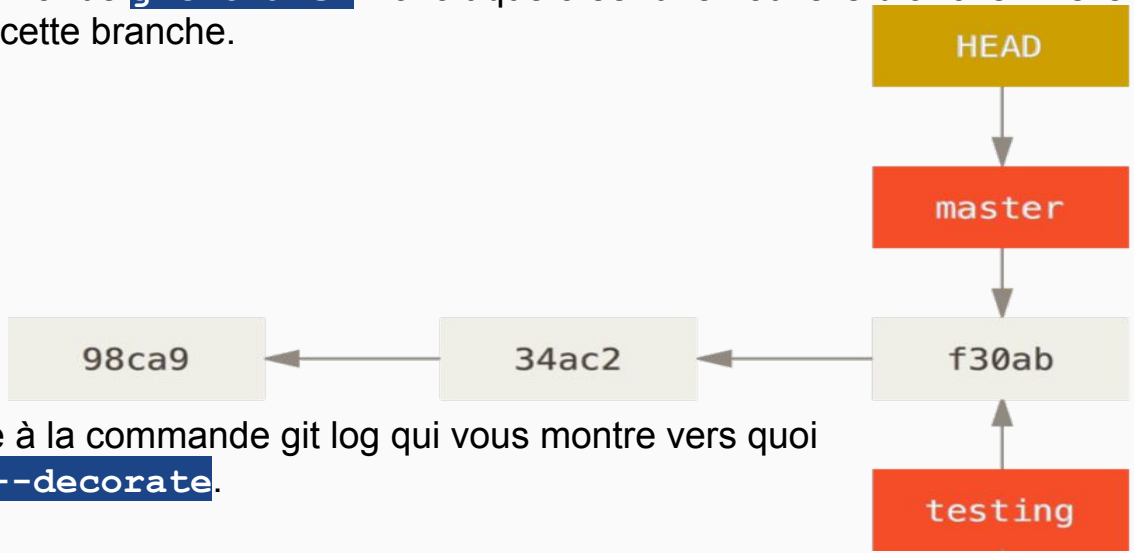
```
$ git branch testing
```

Cela crée un nouveau pointeur vers le **commit** courant.



HEAD pointant vers une branche

Comment Git connaît-il alors la branche sur laquelle vous vous trouvez ?
Il conserve à cet effet un pointeur spécial appelé HEAD.
==>il s'agit simplement d'un pointeur sur la branche locale où vous vous trouvez. Dans ce cas, vous vous trouvez toujours sur master. En effet, la commande `git branch` n'a fait que créer une nouvelle branche — elle n'a pas fait basculer la copie de travail vers cette branche.



Vous pouvez vérifier cela facilement grâce à la commande git log qui vous montre vers quoi les branches pointent. Il s'agit de l'option `--decorate`.

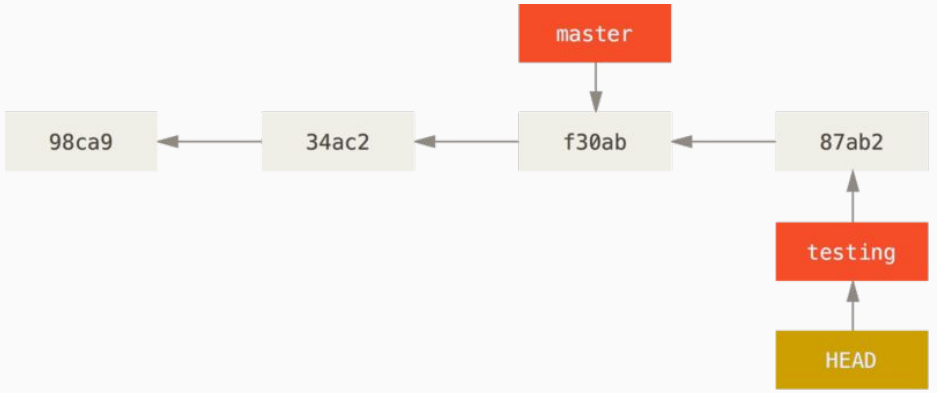
```
$ git log --oneline --decorate
f30ab (HEAD, master, test) add feature #32 - ability to add new
34ac2 fixed bug #ch1328 - stack overflow under certain conditions
98ca9 initial commit of my project
```

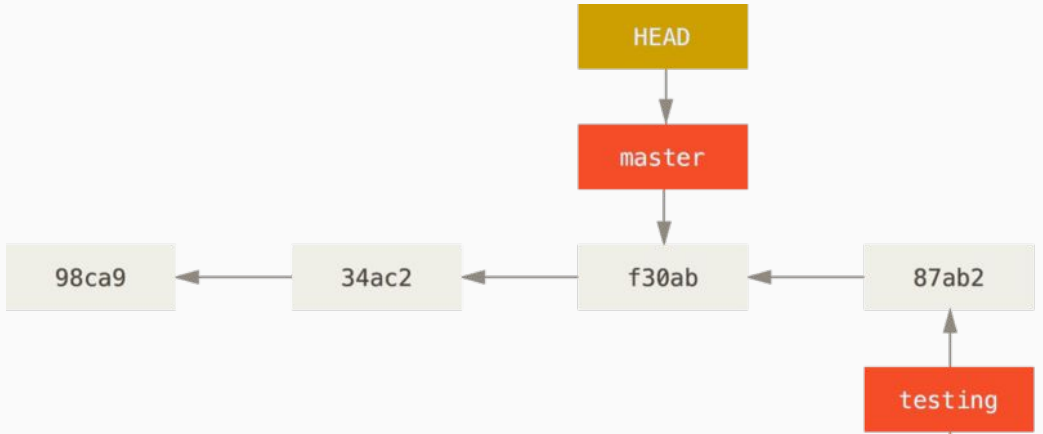
```
$ vim test.rb
$ git commit -a -m 'made a change'
```

La branche test a avancé tandis que la branche master pointe toujours sur le **commit** sur lequel vous étiez lorsque vous avez lancé la commande `git checkout` pour changer de branche.

Retournons sur la branche master :

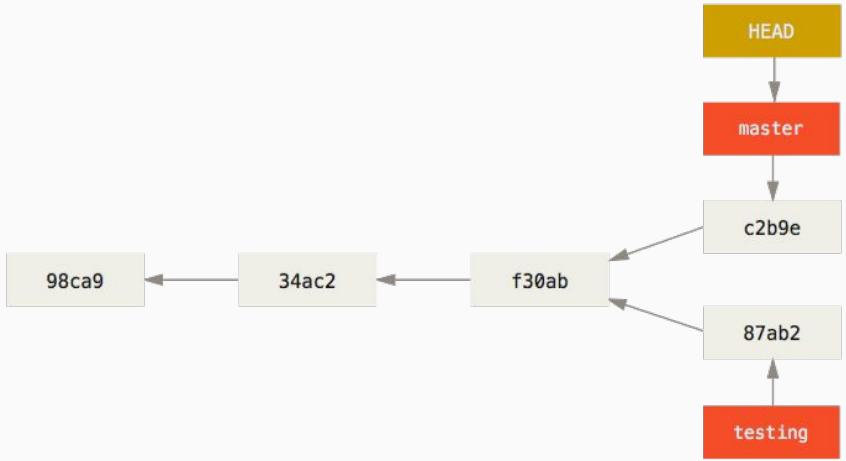
```
$ git checkout master
```



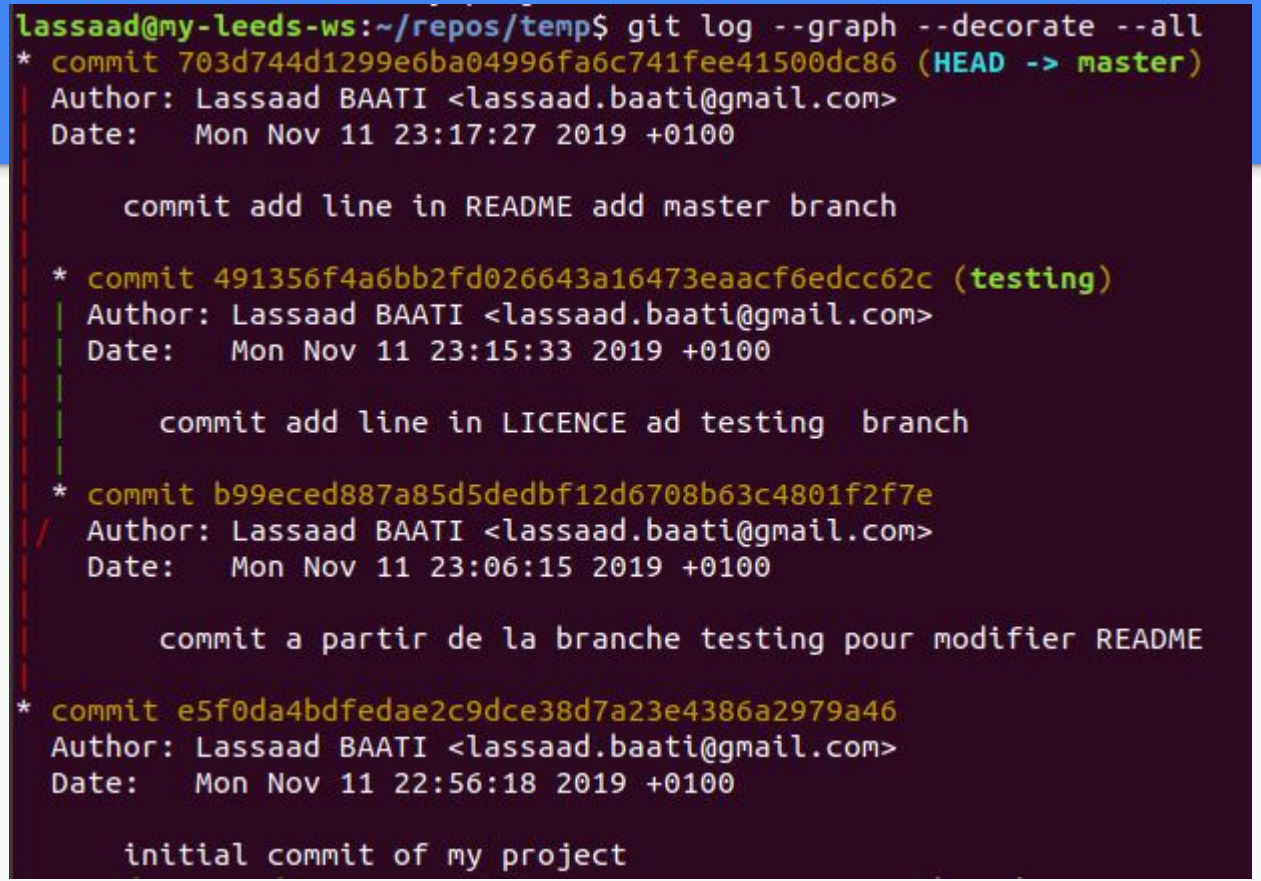


```
$ vim test.rb
$ git commit -a -m 'made other changes'
```

l'historique du projet a divergé. Vous avez créé une branche et basculé dessus, y avez réalisé des modifications, puis vous avez rebasculé sur la branche principale et réalisé d'autres modifications. Ces deux modifications sont isolées dans des branches séparées : vous pouvez basculer d'une branche à l'autre et les fusionner quand vous êtes prêt. Et vous avez fait tout ceci avec de simples commandes : `branch`, `checkout` et commit.



Introduction aux branches



Exercice

1. Créer un dossier temp
2. Ajouter 3 fichiers README LICENCE et test.php
3. Faire un git init
4. Modifier un fichier
5. Ajouter ce fichier à l'index
6. Faire un commit de la dernière modification sur la branche master (Pointé par HEAD)
7. Créer une nouvelle branche testing
8. Basculer sur la nouvelle branche testing
9. Ajouter une ligne au fichier LICENCE
10. Ajouter le fichier modifié à l'index (staged)
11. Faire un commit sur la branche testing (pointé par HEAD).
12. Vérifier le log des commit
13. Basculer sur la branche master
14. Ajouter une ligne au fichier test.rb
15. Visualiser l'état des commit par branche avec la commande suivante

```
$ git log --graph --decorate --all
```

Exercice 1 ⇒ solution.

Now that you have learned the basics of Git workflow, try running through this a couple of times on your own:

1. Create a folder called learn_git_again. $ mkdir learn_git_again
2. cd into the learn_git_again folder. $ cd learn_git_again
3. Create a file called third.txt. $ touch third.txt
4. Initialize an empty git repository. $ git init
5. Add third.txt to the staging area. $ git add third.txt
6. Commit with the message "adding third.txt". $ git commit -m "adding third.txt"
7. Check out your commit with git log. $ git log
8. Create another file called fourth.txt. $ touch fourth.txt
9. Add fourth.txt to the staging area. $ git add fourth.txt
10. Commit with the message "adding fourth.txt". $ git commit -m "adding fourth.txt"
11. Remove the third.txt file. $ rm third.txt
12. Add this change to the staging area. $ git add third.txt
13. Commit with the message "removing third.txt". $ git commit -m "removing third.txt"
14. Check out your commits using git log. $ git log
15. Change your global setting to core.pager=cat - you can read more about that here. $ git config --global core.pager "cat"
16. Write the command to list all of the global configurations for git on your machine. You can type $ git config --global to find out how to do this - $ git config --global --list