Les fondamentaux

Romain THERRAT

POCKOST

14 décembre 2020

Sommaire

Fonctionnement interne

- Git est avant tout un système de fichier
- Au-dessus de celui-ci a été construit le SCM
- Avant la version 1.5 la commande `git` était centrée sur le système de fichier
- git était peut-être complexe . . . avant !
- Pas trop peur ? On y va !

Le dossier ".git"

Après un `git init` un dossier `.git` est créé dans le dossier parent du projet. Certain fichier et dossier sont assez explicites

```
romain@laptop:~/myproject$ ls .git
branches  config  description  HEAD  hooks  info  object
```

config Ce qu'on a configuré avec `git config`

hooks Dossier contenant les hooks git pour ce projet

info Diverses informations (exclusion propre à l'utilisateur par exemple)

Fonctionnement interne

Les éléments les plus importants sont

HEAD	La référence vers l'état actuel
objects	Réel stockage des données
refs	Dossier des différentes branches (wait and see)
index	Éléments en attentes de versionnement

Stockage de données interne

A l'initialisation le dossier `objects` est vide. Nous pouvons ajouter des données via `git hash-objects`.

```
$ echo "Hello ;)" | git hash-object -w --stdin
2183f45093dfa506941dffc62ab9cfee352c2915
```

- Retourne un id (hash)
- Présent dans le dossier `objects` (chiffré)
- Récupérable via `git cat-file -p <id>`

Note : On utilise jamais cette commande !

Stockage interne et version

- Pourquoi faire ça ?
  - Avec un simple référence on pointe un fichier
  - Possibilité de gérer plusieurs versions

```
$ echo "version 1" > test.txt
$ git hash-object -w test.txt
83baae61804e65cc73a7201a7252750c76066a30
$ git hash-object -w test.txt
$ echo "version 2" > test.txt
$ git hash-object -w test.txt
1f7a7a472abf3dd9643fd615f6da379c4acb3e3a
$ git cat-file -p 83baae61804e65cc73a7201a7252750c76066a3
version 2
```

Gestion d'arbre

- `objects` = Contenue d'un fichier
- `blob` : `Binarie Large OBject`
- Aucune notion de nom de fichier
- Aucune notion d'arbre (Hiérarchie)
- Comment grouper les `object` entre eux
- Solution :

Les arbres (`tree`)

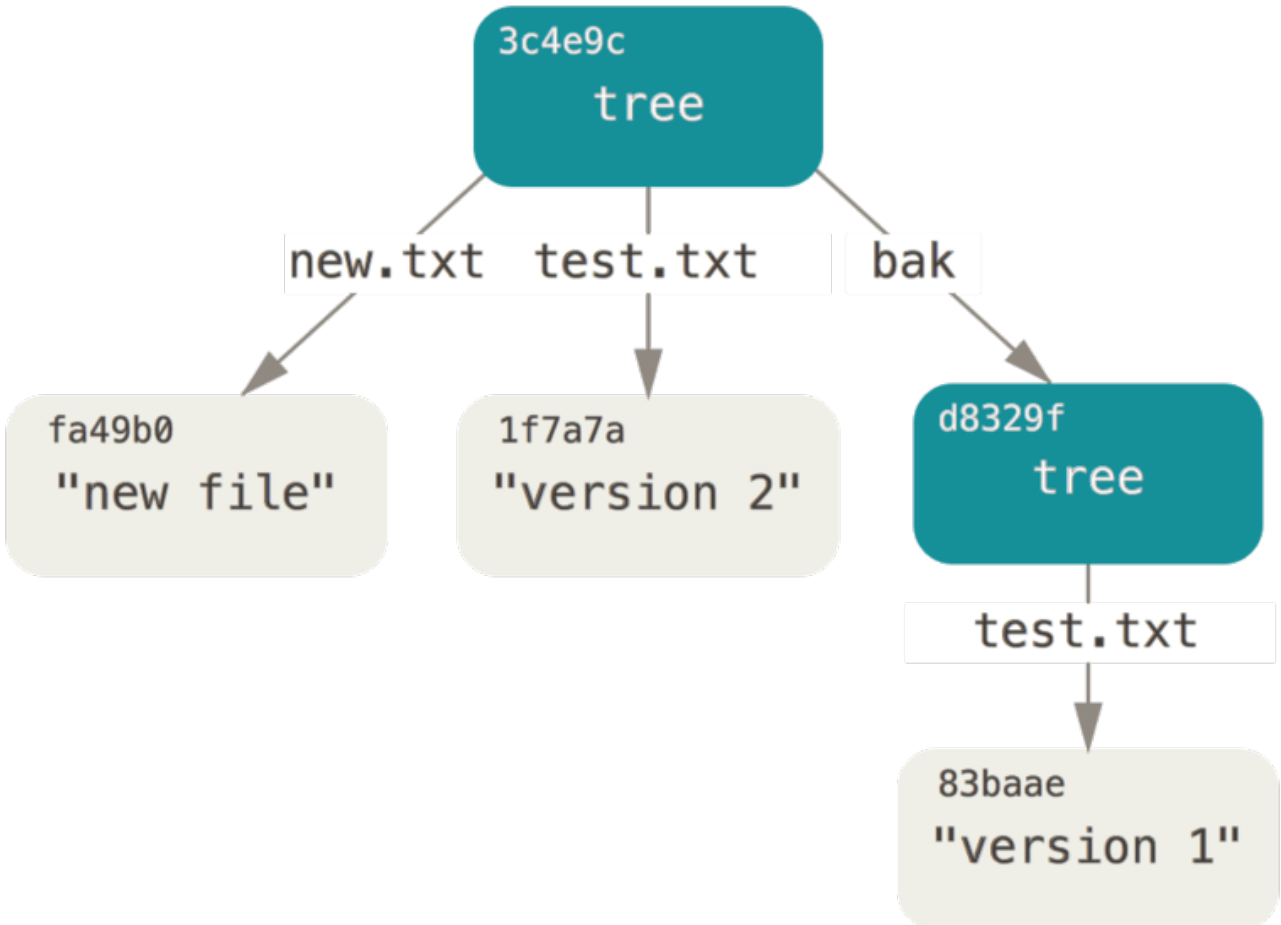
Les arbres

Les arbres :

- Correspondance entre un `object` et un nom
- Contient le mode (droits)

```
$ git update-index --add --cacheinfo 100644 83baae61804e
$ git write-tree
c1113de8ce1a603f7fdff0149c4ccfcdbfc823c3
git cat-file -p c1113de8ce1a603f7fdff0149c4ccfcdbfc823c3
100644 blob 83baae61804e65cc73a7201a7252750c76066a30
```

Les arbres


3c4e9c
tree
new.txt
test.txt
bak
fa49b0
"new file"
1f7a7a
"version 2"
d8329f
tree
test.txt
83baae
"version 1"

Les commit

Avec les `object` et les `tree` nous avons encore un problème

- Comment avoir un référentiel des différentes versions d'un fichier ?

Solution

- Les commits

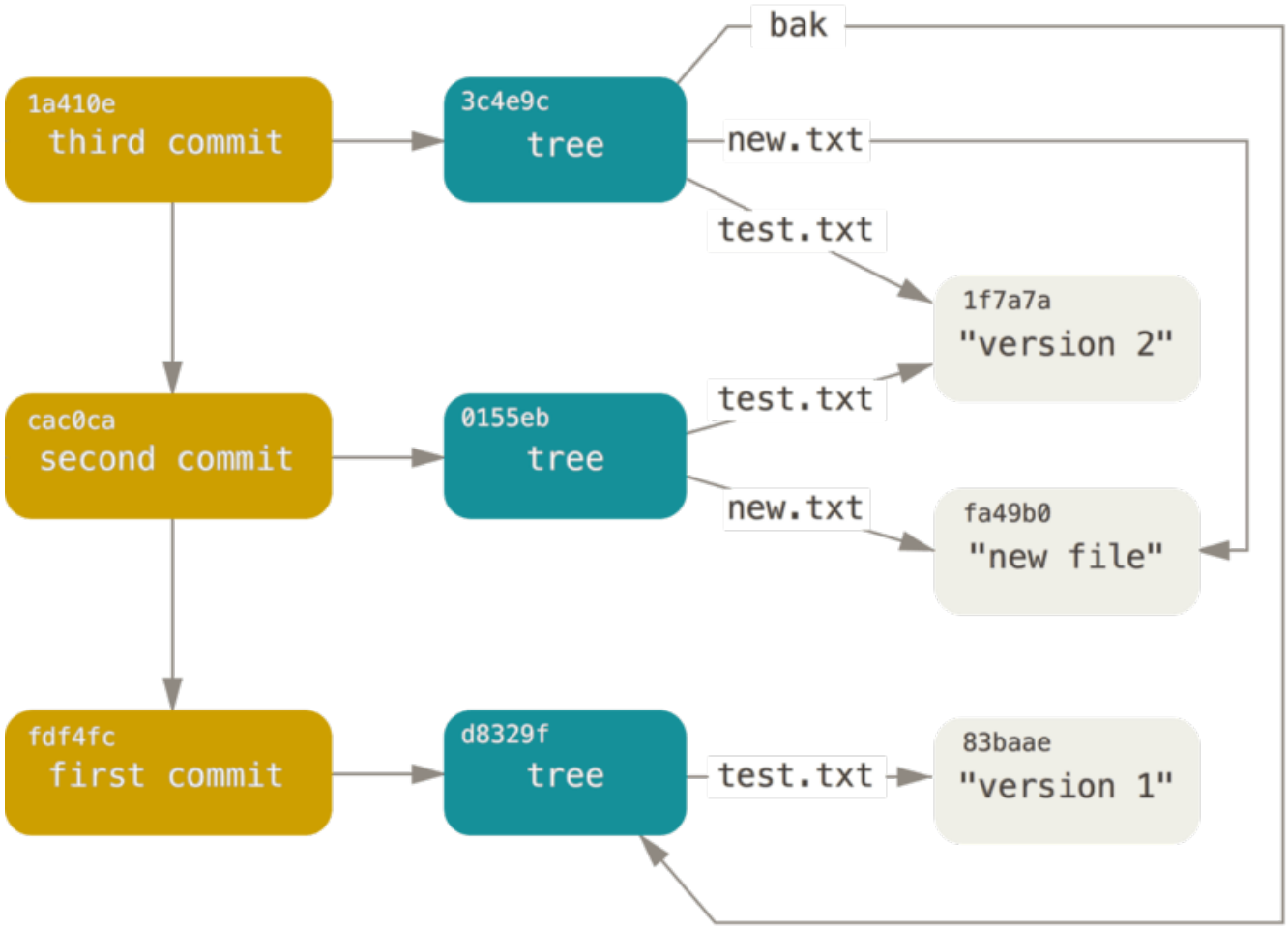
Les commit

Qu'est-ce qu'un commit ?

- Un `object`
- Référence un arbre
- Et un ancien commit

```
$ echo 'first commit' | git commit-tree c1113de8ce1a603f
f5c427827e1cb89ad7de521448cad48de6c2d264
$ git cat-file -p f5c427827e1cb89ad7de521448cad48de6c2d2
tree c1113de8ce1a603f7fdff0149c4ccfcdbfc823c3
author Romain THERRAT <romain@pockost.com> 1579270598 +0
committer Romain THERRAT <romain@pockost.com> 157927059

first commit
$ echo "second commit" | git commit-tree <new-id> -p f5
```

Les commits


bak
1a410e
third commit
3c4e9c
tree
new.txt
test.txt
1f7a7a
"version 2"
cac0ca
second commit
0155eb
tree
test.txt
new.txt
fa49b0
"new file"
fdf4fc
first commit
d8329f
tree
test.txt
83baae
"version 1"

Sommaire

Utilisation standard

On utilise jamais ces commandes vues précédemment

- Trop complexe
- Trop de risque d'erreur
- Pas de fonctions avancés

On va utiliser des commandes de plus haut niveau mais le cœur de git reste le même.

Créer un commit

Pour créer un commit il est nécessaire de

- Créer des fichiers (=votre code source)
- Ajouter ces fichiers pour le prochain commit (=staging) via la commande `git add <files>`
- Créer un commit via `git commit -m "comment"`

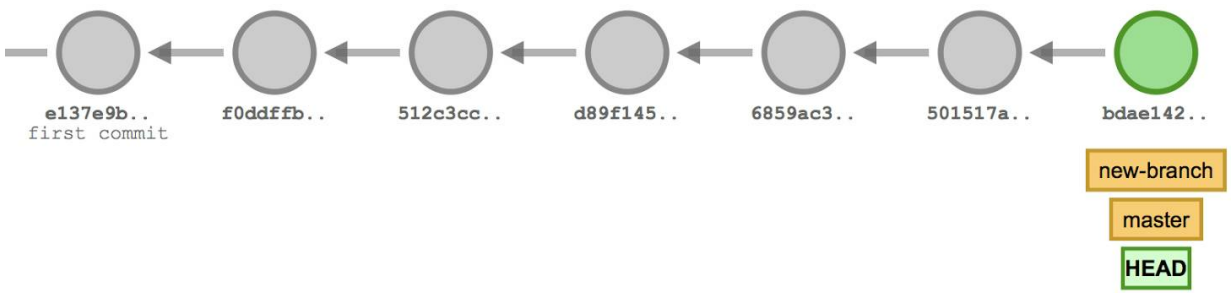
Note : Vous pouvez, à tout moment, voir l'état de votre dépôt via la commande `git status`.

Créer un commit

```
$ touch fichier1.txt
$ git status
...
Untracked files:
        fichier1.txt
...
$ git add fichier1.txt
$ git status
...
Changes to be committed:
        new file:   fichier1.txt
...
$ git commit -m "First commit"
[master (root-commit) 9a3e68e] First commit
 1 file changed, 0 insertions(+), 0 deletions(-)
```

Créer un commit

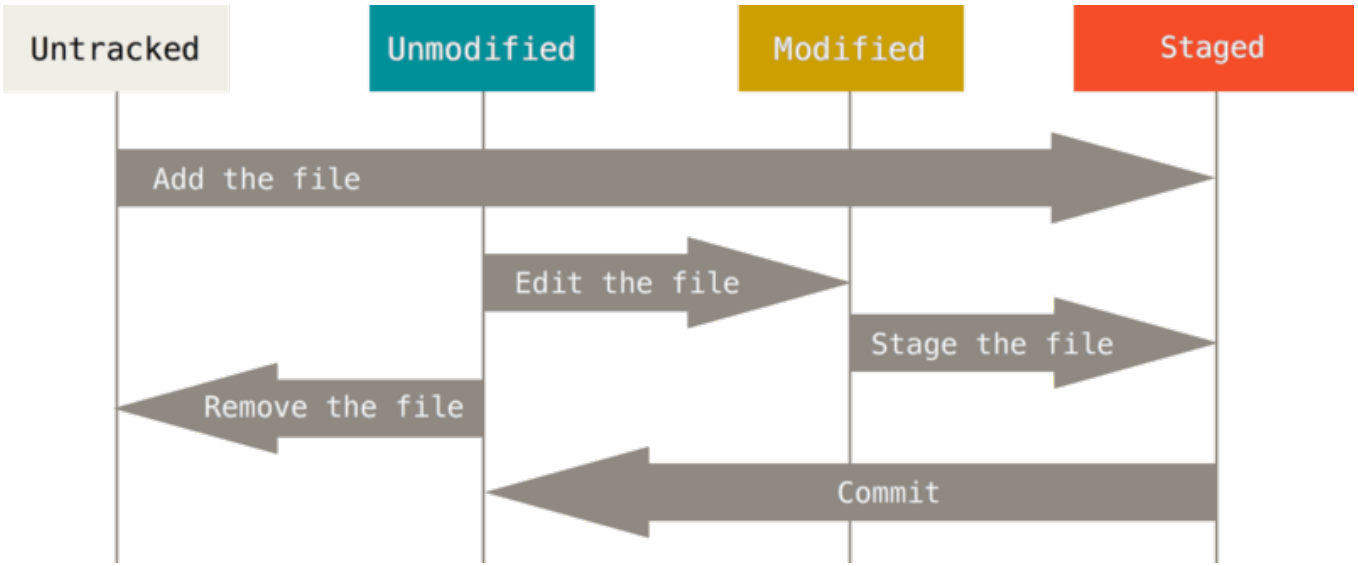
Les commits se suivent



Créer un commit

- Idem pour les
  - Modifications de source
  - Suppressions de fichier

État des fichiers



Sommaire

Visualisation

- Que pouvons-nous voir ?
  - Historique des commits
  - `git log`
  - Fichier modifié et/ou à commiter
  - `git status`
  - Changement à commiter
  - `git diff`
  - Changement réalisé par un commit
  - `git show` et `git blame`

git log

```
$ git log
commit d515e5c2b0b2b2ba5797cbfd996f0da2a4f72c6d
Author: Romain THERRAT <romain@pockost.com>
Date:   Sun Jan 19 16:14:28 2020 +0100
    Second commit

commit 9a3e68e69a077711bb1465967f1f771bd3846633
Author: Romain THERRAT <romain@pockost.com>
Date:   Fri Jan 17 15:26:05 2020 +0100
    First commit
```

- Option `--stat`
- Option `-p -2`

git status

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in wor

        modified:   fichier2.txt

Untracked files:
  (use "git add <file>..." to include in what will be commit

        fichier3.txt

no changes added to commit (use "git add" and/or "git commit
```

git diff

```
$ git diff fichier2.txt
diff --git a/fichier2.txt b/fichier2.txt
index e69de29..e965047 100644
--- a/fichier2.txt
+++ b/fichier2.txt
@@ -0,0 +1 @@
+Hello
```

git show

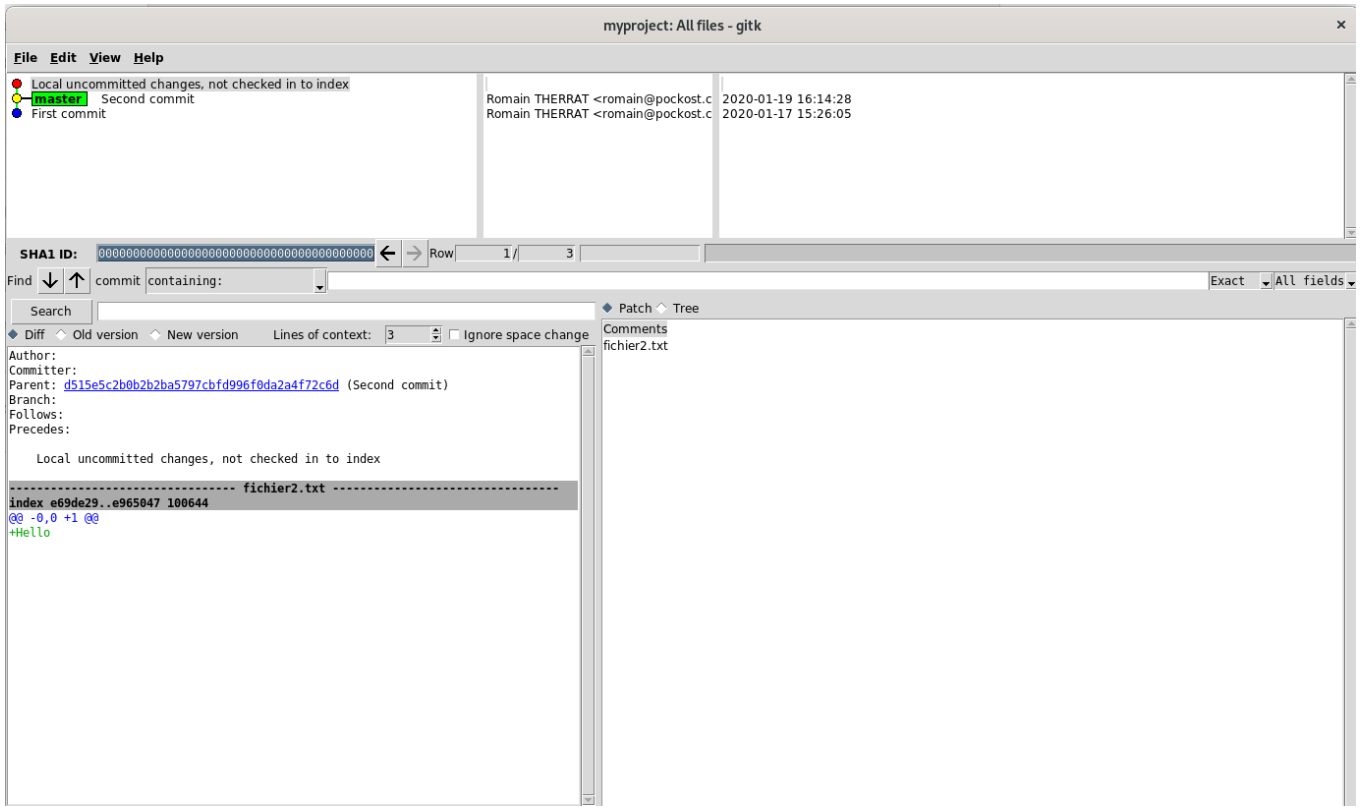
```
git show d515e5c2b0b2b2ba5797cbfd996f0da2a4f72c6d
commit d515e5c2b0b2b2ba5797cbfd996f0da2a4f72c6d
Author: Romain THERRAT <romain@pockost.com>
Date:   Sun Jan 19 16:14:28 2020 +0100

    Second commit

diff --git a/fichier1.txt b/fichier1.txt
index e69de29..8bdf806 100644
--- a/fichier1.txt
+++ b/fichier1.txt
@@ -0,0 +1 @@
+changement
diff --git a/fichier2.txt b/fichier2.txt
new file mode 100644
```

git blame

```
git blame fichier1.txt
d515e5c2 (Romain THERRAT 2020-01-19 16:14:28 +0100 1) change
bebc26f9 (Romain THERRAT 2020-01-19 16:32:54 +0100 2) change
```

Visualisation gitk

Nous pouvons aussi utiliser l'utilitaire `gitk`



Sommaire

Labs

Nous allons réaliser un labs pour mettre en œuvre toutes ces nouvelles connaissances. L'idée de celui-ci est de présenter la création d'un projet git local, l'ajout et la modification de fichier ainsi que la gestion des commits.

- Placer vous dans un nouveau dossier vide. Celui-ci contiendra à terme le code source de votre projet.
- Créer un nouveau projet git via la commande `git init`
- Vous pouvez constater via la commande `git status` que votre dépôt a bien été créé et que celui-ci ne contient aucune donnée.
- Ajoute un fichier nommé `index.php` vide
- Constater le changement d'état de la commande `git status`
- Utiliser les commandes `git add` et `git commit` pour versionner ce premier fichier.
- Exécuter les commandes `git status` et `git log` pour constater les changements.

Labs

- Ajouter dans votre fichier `index.php` du code source (`<?php echo "hello world";` par exemple)
- Créer un nouveau dossier vide nommé `static`
- Créer un nouveau fichier vide nommé `.htaccess`
- Exécuter les commandes `git status` et `git diff` pour constater les changements.
- Créer un nouveau commit avec ces modifications
- Ajouter une image dans le dossier `static` et utiliser la commande `git rm` pour supprimer le fichier `.htaccess`. Commiter le tout
- Exécuter les commandes `git status` et `git log` pour constater les changements.
- Visualiser le résultat en GUI avec l'utilitaire `gitk`
- Réaliser un `git blame` sur le fichier `index.php`

Labs

Si vous souhaitez aller plus loin vous pouvez

- Réaliser quelques commit sur le site ▸ git-school.github.io/visualizing-git/ pour constater l'évolution de votre arbre
- Vous pourrez utiliser ce site par la suite
- Analyser les fichiers créés dans l'exercices dans le dossier `.git`
- Récupérer le contenu d'un fichier à l'aide de la commande `git cat-file`