Pourquoi utiliser Git ?

Table des matières

Objectifs

- Comprendre l'intérêt de la gestion de version arborescente
- Maîtriser les différents concepts de la gestion de version arborescente
- Effectuer un premier commit simple

I Mise en contexte de l'activité

Le début d'une aventure palpitante

Alphonse, Boris et Céline, vos amis de toujours, ont choisi de rédiger leur rapport de CW01 en markdown en utilisant le logiciel de gestion de version git, et tout allait pour le mieux jusqu'à ce qu'Alphonse vous appelle en panique : tout a disparu.

Non connaisseur mais curieux, vous acceptez de vous lancer dans une passionnante aventure à la découverte de Git, cet animal extraordinaire, dans le but d'aider votre ami.

II Quelques notions de base

Récupérer le code

Pour récupérer l'entièreté du code de votre ami, ce sera très simple, il faudra entrer une et une seule commande Linux. Pour comprendre pourquoi c'est si simple, il faut comprendre comment on partage usuellement du code sur internet

Le code de votre ami est sur ce que l'on appelle un dépôt, c'est, à l'instar d'un dépôt logistique, un endroit où l'on met du code dans le but qu'il soit récupéré par d'autres personnes.

Dépôt de code source Az Définition

Un dépôt de code source est un espace de stockage de données qui permet leur accès à différents utilisateurs. Il stocke également un historique de toutes les modifications apportées au code depuis le début d'un projet.

Pour récupérer le code, nous allons devoir cloner le dépôt de notre ami, c'est à dire créer une copie en tout point identique à l'endroit où ont jusqu'à présent travaillé vos amis. Vous l'aurez compris : on va également cloner toutes les modifications apportées au code depuis le début du projet et savoir ce qui s'est passé.

III Exercice : Récupérer du code existant

Cloner le dépôt

Ça y est, on va cloner le dépôt git.

```
1 git clone https://gitlab.utc.fr/bwojtkow/rapport_cw01
```

C'est notre première commande git, elle dit simplement à Linux de récupérer le dépôt de code situé à `https://gitlab.utc.fr/bwojtkow/rapport_cw01`, de créer un dossier et d'y mettre l'entièreté des versions du code.

Si git n'est pas installé, souvenez vous du début d'init :

```
1 sudo apt install git
```

Question 1

[solution n°1 p. 18]

Rendez-vous dans le dossier créé et affichez-y la liste des fichiers cachés, que contient-il ?

Indice :

Quand git clone un dépôt, il crée un dossier ayant le nom du projet, généralement présent dans l'URL, et il y colle le code et l'historique

Indice :

```
1 cd rapport_cw01/
2 ls -a
```

Question 2

[solution n°2 p. 18]

À votre avis, que contient le fichier `.git` ?

IV Exercice : Exploration du dépot

Prise de premières informations

Ça y est, vous avez sur votre machine le code, avec tout son historique dans un fichier `.git`. Comme on l'a dit, on ne veut surtout pas manipuler directement le `.git`, c'est d'ailleurs pour cela qu'il est caché.

Bon, on va essayer d'obtenir quelques informations sur tout ça, et on va utiliser une commande importante : `git log`. Git log, c'est une commande qui permet d'afficher l'historique des modifications. Pour le moment, on va utiliser une version de `git log` un peu améliorée pour que ce soit joli et compréhensible. Mais cette version simplifiée n'est pas toujours satisfaisante.

```
1 git log --pretty=oneline --reverse
```

Ce qu'on voit, c'est une suite de changements. On peut voir ainsi en un clin d'œil que d'abord le plan a été ajouté, puis l'introduction, etc. Chaque modification s'appelle un commit, et a un identifiant unique et un nom, ce dernier lui a été donné par l'auteur.

Question 1

[solution n°3 p. 18]

Quelle a été la dernière modification effectuée sur le dépôt ?

Question 2

[solution n°4 p. 18]

Quel est l'identifiant du commit de la dernière modification effectuée sur le dépôt ?

Question 3

[solution n°5 p. 18]

Qui est l'auteur de ce commit ? Quelle est son adresse e-mail ?

Indice :

Comme dit, on utilise une version simplifiée de `git log`, pensez à plutôt utiliser sa version initial pour avoir plus d'informations

```
1 git log
```

Vous devriez avoir compris pourquoi l'option qui affiche les commits dans l'ordre s'appelle `--reverse` : les commits qui nous intéressent, quand on développe, ce sont les derniers, il est donc normal de les afficher au début par défaut.

V Historique des versions

On a vu que git enregistrait les modifications sous formes de commits accessibles via git log. Git log est, comme toutes les commandes git, très modulaire et dispose de nombreuses options.

Commit Az Définition

Un commit est une version du projet à un instant t. Elle a nécessairement

- Un message de commit, ce message est rédigé par l'auteur lors du commit
- Un identifiant unique.
- Un commit parent unique (la version précédente)
- Un auteur, défini par un nom, un prénom et une adresse e-mail

Identifiant unique Complément

L'identifiant unique est appelé SHA, c'est un *hashage* de beaucoup d'information, dont notamment :

- Le SHA du commit parent
- Les diverses modifications apportées par le commit
- Les méta-données du commit

Remarque

Il est possible de connaître les modifications introduites par un commit en utilisant la commande git show

Les bonnes pratiques du commit Conseil

Un bon commit, c'est un commit dit atomique, c'est à dire qu'il introduit une modification, la plus petite possible tout en ayant du sens.

Un bon commit, c'est un commit dont le message de commit est court et descriptif. On doit comprendre ce que fait le commit sans avoir à regarder dans le code.

Remarque

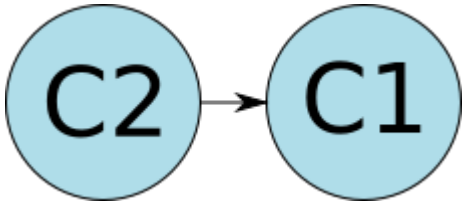
On a donc vu deux intérêts de git :

1. Il permet de stocker et partager son code de manière structurée ;
2. Il permet de tracer facilement les modifications faites au cours du temps.

Formalisation

Complément

Pour représenter un commit, on les représente souvent avec un cercle et une flèche, qui pointe vers son parent. Par exemple, dans l'image suivante, on a deux commit, C1 et C2, et C2 a été créé à partir de C1



VI Exercice : Exploration du dépôt

Question 1

[solution n°6 p. 18]

Utilisez la commande `git show` pour afficher le contenu du commit `be949407c8a5cb921470a78369ca2ccaa8adf6cb` (il s'agit de l'avant-dernier commit).

Quelle modification a été introduite par ce commit ?

Indice :

Pour accéder à la documentation d'une commande `git`, il faut utiliser la syntaxe "`man git-cmd`", dans notre cas :

```
1 man git-show
```

Indice :

dans notre cas, il faut utiliser `git show be949407c8a5cb921470a78369ca2ccaa8adf6cb.`

Essayez de comprendre par vous même le retour de cette commande.

Question 2

[solution n°7 p. 19]

SI vous avez de l'avance, utilisez le manuel pour trouver une option de `git diff` pour voir précisément ce qui a changé dans la ligne.

Question 3

[solution n°8 p. 19]

Trouvez ce qu'il s'est passé entre le commit "retrait ligne pas intéressantes" et le commit "modifications mineures pour améliorer la lecture".

Quand on veut connaître les modifications introduites par plusieurs commits, on utilise la commande git diff.

Indice :

git diff a des tonnes d'utilisations, dans notre cas, son usage est :

```
1 git diff 740df085f4bcbdaef86cfc35a5d927d2453733a1
  be949407c8a5cb921470a78369ca2ccaa8adf6cb --word-diff
```

VII Exercice : Navigation dans le dépôt

Maintenant qu'on a une bonne idée du commit coupable, on va se poser la question suivante : où est le rapport ?

De ce que l'on sait du git diff, le rapport est un fichier rapport.md, supposément à la racine du projet.

Question 1

[solution n°9 p. 19]

On l'a dit, un commit est une version du projet. Ce que l'on va faire, c'est se déplacer dans l'arborescence des commits pour charger la version qui nous intéresse

Utilisez la commande git checkout pour changer de version du rapport et charger le commit be949407c8a5cb921470a78369ca2ccaa8adf6cb

Constatez que le rapport est magiquement réapparu

Indice :

```
1 git checkout be949407c8a5cb921470a78369ca2ccaa8adf6cb
```

Question 2

[solution n°10 p. 19]

Affichez le rapport dans le terminal, constatez qu'il est très gros

Indice :

```
1 cat rapport.md
```

Question 3

[solution n°11 p. 19]

Déplacez vous au tout premier commit, constatez que le rapport est encore plus petit

Indice :

```
1 git checkout 348da1f810758f96b987b3e6269ad1897bc5a8b8
2 cat Rapport.md
```

Question 4

[solution n°12 p. 20]

Revenez à l'état initial

Indice :

Il faut utiliser l'option `--all` de la commande git log pour obtenir l'ensemble des commits.

Nous venons d'utiliser les hash de commit afin de naviguer dans l'historique. De manière générale nous préférons utiliser les étiquettes de branche directement.

VIII Se déplacer dans les versions

Questionner l'historique

Fondamental

Comme chaque commit a un identifiant, associé à beaucoup d'informations, il est donc possible de :

- Voir les changements introduits pas un commit en utilisant `git show` ;
- Comparer deux états du code en utilisant `git diff`.

Se déplacer dans les versions

Fondamental

Il est possible de charger différentes versions du projet en utilisant `git checkout`.

Quand cette commande est appelée avec pour argument une version du projet, elle remplace complètement le contenu à la racine du projet, et charge celui correspondant à la version demandée.

La version précédente du projet est désormais cachée quelque part dans le dossier `.git`.

Remarque

On a vu deux intérêts supplémentaires de la gestion de version :

- Naviguer entre les versions de manière totalement transparente ;
- Pouvoir comparer différentes versions pour voir s'il y a un bug.

IX Versionnage arborescent

Travailler à plusieurs : notion de branche

Pour travailler à plusieurs, il peut être intéressant de travailler chacun dans son coin à partir d'une version minimale du code et de mettre en commun à la fin.

Dans le cas qui nous occupe c'est ce qui a été fait, si vous exécutez la commande suivante, vous verrez une version arborescente :

```
1 git log --graph --oneline
```

Notion de branche — Fondamental

Une branche, c'est une divergence avec le travail principal, créée dans le but de ne pas déranger les autres membres du projet.

Branche master — Fondamental

La suite de commits qui contient l'ensemble du travail stable et fonctionnel est appelée *« branche master »*, c'est également la toute première branche existante lorsque vous créez un projet. Cependant, pour des raisons éthiques, certains hébergeurs (comme GitHub) préfèrent la nommer `main`.

Notion de fusion — Fondamental

Fusionner (ou par anglicisme, merger), c'est rassembler le travail produit sur deux branches en créant un commit spécial : le commit de fusion (`merge`).

- La plupart du temps, cela se passe bien et l'algorithme détermine sans problème comment fusionner ;
- Des fois les modifications apportées concernent la même zone du code, et git ne peut pas déterminer tout seul les modifications à apporter ; il y a conflit, et il faut gérer cela manuellement.

Commit de fusion — Az Définition

Le commit de fusion, c'est le seul commit de git qui peut avoir deux parents. Il contient l'arbitrage entre les deux branches, et détermine comment les modifications de chaque branche doivent être appliquées.

Remarque

Quand l'une des branches ne contient pas de commit depuis la séparation des deux branches, il n'y a pas d'arbitrage à faire, git fusionne donc les deux branches sans créer de commit.

Étiquette

Az Définition

Une étiquette, c'est une variable dans git qui permet d'identifier une version et qui peut être utilisée de la même manière que le numéro d'un commit.

Exemple

- L'étiquette HEAD peut être utilisée pour identifier le commit qui est actuellement chargé dans votre répertoire courant, vous l'avez normalement vu bouger.
- L'étiquette `master` identifie le commit au bout de la branche master
- L'étiquette `origin/master` identifie le bout de la branche master sur le dépôt distant
- Dans notre cas, l'étiquette alphonse identifie le bout de la branche alphonse.

Remarque

Cela nous amène à un nouvel intérêt de git : collaborer sur un même projet en ne gênant pas les autres dans leur travail.

X Exercice : Comprendre l'arborescence

Avant de corriger le problème causé par Boris, on va essayer de comprendre ce qu'il s'est passé au cours du projet en utilisant la notion de branche

Question 1

[solution n°13 p. 20]

En utilisant la commande ci-après essayez de comprendre ce qui s'est passé.

Indice :

```
1 git log --pretty=oneline --graph --all
```

Git branch est une commande qui permet de gérer tout ce qui a trait aux branches.

Question 2

[solution n°14 p. 20]

La commande `git branch` permet de lister les branches. Utilisez la avec l'option `-a`, puis sans `-a`. Combien y a-t-il de branche sur le dépôt original ? Combien y a-t-il de branches sur votre dépôt ?

Indice :

Consultez la documentation pour voir ce que fait l'option `-a.`

Indice :

`remotes/origin` correspond aux branches disponibles sur le dépôt `origin`. C'est le nom que prend par défaut celui depuis lequel vous avez cloné votre dépôt.

Pour vous en convaincre, vous pouvez entrer :

```
1 git remote -v
```

Question 3

[solution n°15 p. 20]

Entrez la commande

```
1 git checkout alphonse
```

À votre avis que s'est-il passé ?

XI Exercice : Revert un commit

Ça y est, maintenant qu'on a compris tout ce qu'il y avait à comprendre sur le projet, on peut passer à la fin : corriger l'erreur de Boris.

Question 1

[solution n°16 p. 20]

Pour corriger l'erreur de Boris, il faut annuler son commit. On verra plus tard pourquoi, mais, dans ce genre de cas, on veut ajouter un commit qui va complètement annuler l'erreur de Boris, et pas supprimer le commit. Disons pour le moment que c'est simplement pour garder une traçabilité. Pour cela, on va utiliser la commande `git revert`.

Indice :

Il faut utilisez la commande suivante. Lisez les instructions

```
1 git revert HEAD
```

Question 2

[solution n°17 p. 21]

On l'a dit, un message de commit contient un nom, un prénom, et un mail d'auteur. Normalement, si vous faites un git-revert, vous allez vous faire jeter. Une fois votre identité correctement configurée, essayez de faire un `git revert`, que se passe-t-il ? À votre avis, pourquoi ?

```
1 git config --global user.name "John Doe"
2 git config --global user.email "john@doe.org"
```

Indice :

Pensez aux composantes obligatoires d'un commit.

Question 3

[solution n°18 p. 21]

Vérifiez qu'il y a bien un nouveau commit, qu'il est à vous.

En théorie, il vous faudrait encore propager vos modifications sur le dépôt de votre ami, mais on va y revenir durant l'API.

XII Configurer son identité

Configurer son identité

Fondamental

Pour pouvoir sauvegarder des modifications, il est absolument nécessaire d'avoir configuré son identité.

Identité globale

Az Définition

Votre identité globale est propre à l'ordinateur sur lequel vous êtes. Par défaut, lorsque vous committez sur un projet, git utilise votre identité globale.

Complément

Pour configurer son identité globale, on utilise les commandes :

```
1 git config --global user.email "Vous@exemple.com"
2 git config --global user.name "Votre Nom"
```

Identité locale

Az Définition

Votre identité locale est propre au dépôt sur lequel vous êtes. C'est utile si vous voulez par exemple renseigner une adresse personnelle pour un projet particulier.

Complément

Pour configurer son identité locale, on utilise les commandes :

```
1 git config --local user.email "Vous@exemple.com"
2 git config --local user.name "Votre Nom"
```

Solutions des exercices

Solution n°1

[exercice p. 6]

```
1. .. .git readme.md
```

Solution n°2

[exercice p. 6]

Il contient énormément d'information dont le fameux historique des modifications, des copies des différentes versions des fichiers, etc.

> ⚠ Attention
>
> Il est très risqué de manipuler directement le fichier `.git`, mieux vaut toujours utiliser les commandes qui permettent de le modifier.

Solution n°3

[exercice p. 7]

D'après l'auteur de la modification, lors de ce commit, ce qui n'était pas intéressant a été retiré du projet.

Solution n°4

[exercice p. 7]

dd36dacaa2c6f576a590a6433bb3c31ed9710cd3

Solution n°5

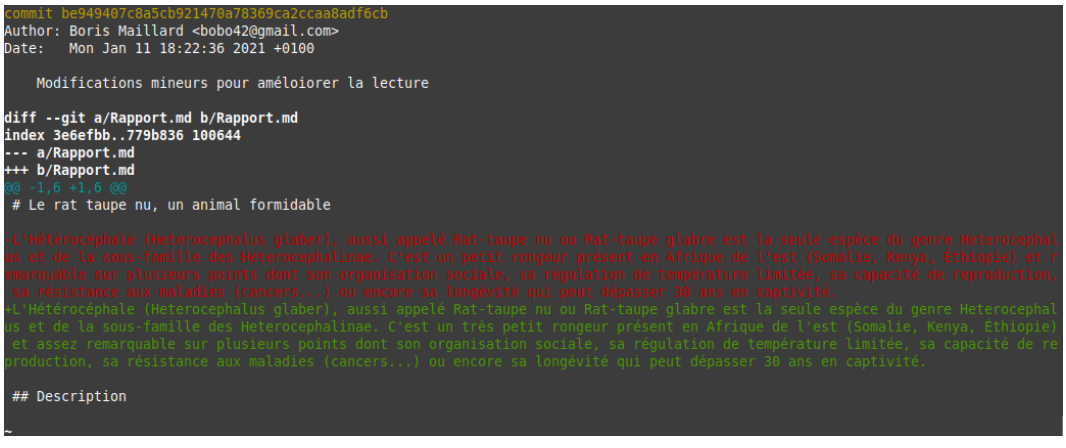
[exercice p. 7]

Alphonse Robichu, son adresse mail est <arobi@arobase.net>

Solution n°6

[exercice p. 10]

Quand on tape la commande précédente, on obtient l'image suivante :



De haut en bas, elle contient :

- Les mêmes informations que l'historique ;
- Une ligne qui nous indique les fichiers concernés par les modifications ;
- Une ligne indiquant l'index, en gros un identifiant de l'état de git au moment de la sauvegarde ;
- précédés de -----, les fichiers qui ont subi des suppressions de ligne ;
- précédés de ++++, les fichiers qui ont subi des ajouts de ligne ;
- En rouge, les lignes supprimées ;
- En vert, les lignes ajoutées ;
- En blanc, quelques lignes non modifiées pour permettre de comprendre où est le code.

On peut donc en déduire que ce commit a supprimé une ligne et en a ajouté une autre.

Solution n°7

[exercice p. 10]

il faut utiliser l'option `--word-diff`.

```
1 git show be949407c8a5cb921470a78369ca2ccaa8adf6cb --word-diff
```

Solution n°8

[exercice p. 10]

Entre ces deux commits, on a ajouté deux mots dans le rapport ainsi qu'un fichier readme.md.

Solution n°9

[exercice p. 11]

Solution n°10

[exercice p. 11]

Solution n°11

[exercice p. 11]

```
1 git log
2 #pour trouver l'identifiant du commit
3 git checkout 348da1f810758f96b987b3e6269ad1897bc5a8b8
4 cat Rapport.md
```

```
1 # Le rat taupe nu, un animal formidable
2
3 ## Description
4
5 ## Mode de vie
6
7 ## Particularités remarquables
8
9 ### Longévité exceptionnelle
10
11 ### Insensibilité à la douleur
12
13 ### Privation d'oxygène
14
15 ## Reproduction
16
17 ## Alimentation
18
```

```
## Systématique

## Le rat-taupe nu dans la fiction


```

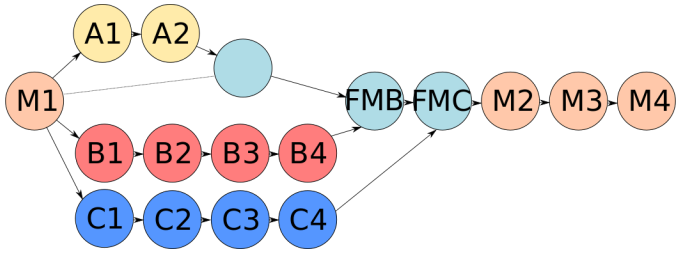
Solution n°12

[exercice p. 11]

```
git log --pretty=oneline --all
git checkout dd36dacaa2c6f576a590a6433bb3c31ed9710cd3
```

Solution n°13

[exercice p. 15]



Historique :

- un premier commit a été fait sur master ;
- 3 branches ont été créées à partir de master ;
- Alphonse a fait 2 commit, boris 4 et céline 4 également ;
- Alphonse a fusionné avec master, cas particulier, comme master n'a pas de commit, il n'y a pas de commit de merge à faire ;
- Boris a fusionné avec master ;
- Céline a fusionné avec master ;
- 3 commits ont été faits sur master.

Solution n°14

[exercice p. 15]

Sur votre machine, `git branch -a` affiche normalement ceci :

```
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/alphonse
  remotes/origin/boris
  remotes/origin/celine
  remotes/origin/master
```

L'astérisque matérialise la branche sur laquelle est placée HEAD.

On voit qu'il y a 4 branches sur le dépôt originel, que HEAD pointe sur master, et qu'il n'y a qu'une seule branche en local.

Solution n°15

[exercice p. 15]

Solution n°16

[exercice p. 16]

Solution n°17

[exercice p. 16]

Un éditeur de texte (nano normalement) s'ouvre pour que vous puissiez remplir le message du commit. Une fois que vous aurez sauvegardé et quitté, vous aurez désormais effectué un commit.

Solution n°18

[exercice p. 16]