Introduction à la gestion de versions avec Git

Table des matières

I Installer Git

Fondamental

Git est un **logiciel** de **gestion de version**. Il est **open source** et publié sous **licence libre**

git-scm.com

Fonctions

Pourquoi la gestion de version ?

- Sauvegarde **incrémentale** du travail
- **Suivi** des modifications
- **Retour en arrière**
- **Partage** des modifications
- **Centralisation** des sources
- **Collaboration** contrôlée
- Possibilité de **maintenir plusieurs versions** simultanées

Installation

Méthode

Git est disponible sur les distributions **GNU/Linux**, sur **MacOS** et sur **Windows**. On trouve également des applications Git pour **Android**.

git-scm.com/book/en/v2/Getting-Started-Installing-Git

Documentation

Complément

git-scm.com/book/en/v2 [fr][1]

Vidéo pour démarrer avec Git

Complément

git-scm.com/video/get-going

1. https://git-scm.com/book/fr/v2

Exercice : Application

[solution n°1 p. 28]

Installez Git sur votre machine et exécutez la commande suivante :

```
1 git help
```

En utilisant cette commande trouver la commande qui permet de "Afficher l'état de la copie de travail".

III Configurer une identité (git config)

Syntaxe

Une fois Git installé la première chose à faire est de le configurer avec les informations qui permettront de signer les futurs *commits* (ce sont les opérations consistant à enregistrer des modifications dans Git).

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

```
git config -l
```

git-scm.com/book/fr/v2/Démarrage-rapide-Paramétrage-à-la-première-utilisation-de-Git[2]

Attention

Deux champs sont obligatoires pour Git :

- le **nom**,
- l'**email**.

Configuration locale vs. configuration globale

Complément

Pour la configuration de Git, on peut choisir entre l'option `--local` (option par défaut) ou `--global` :

- `--global` permet de spécifier que la configuration est vraie quelque soit le dépôt pour l'utilisateur qui fait la configuration ;
- `--local` permet de dire que la configuration n'est valable que pour le dépôt courant.

La configuration locale est **prioritaire** sur la configuration globale.

Complément

Il existe beaucoup d'options configurables dans Git, dont par exemple l'éditeur par défaut, les couleurs de sortie, les politiques de gestion...

git-scm.com/book/fr/v2/Personnalisation-de-Git-Configuration-de-Git

Ajouter une signature GPG à tous ses commits

Complément

L'identité nom/email pouvant être facilement usurpé, il est conseillé d'ajouter une signature via votre clef GPG personnelle. Il faut alors utiliser l'option `-S [IDCLEF]` avec la commande `git commit`.

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

Pour rendre automatique l'utilisation de la clef GPG dans les commandes de git, il faut mettre-à-jour sa configuration git :

```
1 git config --global commit.gpgsign true
2 git config --global user.signingkey IDCLEF
```

IV Exercice : Application

Question 1

[solution n°2 p. 28]

Initialiser votre identité Git.

Indice :

Initialiser Git *(cf. p.5)*

Question 2

[solution n°3 p. 28]

Vérifiez votre identité.

V Créer un dépôt (git init)

Tout dossier du système de fichier peut être suivi par Git, cela signifie que l'on va pouvoir gérer ce dossier (c'est à dire les fichiers et les sous-dossiers qu'il contient) avec Git.

Fondamental

On dit qu'on crée un **dépôt** Git.

Le dépôt Git

Syntaxe

Pour que Git suive un dossier il faut **initialiser** celui-ci.

On se positionne dans le dossier à suivre, puis on exécute :

```
1 git init
```

Exemple

```
1 mkdir example
2 cd example
3 git init
```

À partir de maintenant, Git prend en charge la gestion de version du répertoire *example*.

Remarque

Git ne suit donc pas toutes les modifications sur tout le système, mais uniquement dans les dossiers dans lequel on a fait un `git init`.

Le dossier .git

Attention

Lors du *git init*, Git a créé un dossier .git, dans lequel il stocke tout ce dont il a besoin.

Il **ne faut pas** toucher à ce dossier à moins d'être absolument sûr de ce qu'on fait.

VI Exercice : Application

Question

[solution n°4 p. 28]

Créer un dossier `git/we01` sur votre ordinateur.

Initialisez un dépôt Git dans ce répertoire.

VII Les trois espaces de Git : working directory, staging area, repository

Espaces principaux

Fondamental

Git s'organise en trois espaces (logiques) principaux :

- Le **working directory** (ou répertoire de travail)
- Le **staging area** (zone de préparation)
- Le **repository** (ou dépôt)

Working directory

Az Définition

Le *working directory* correspond à l'état actuel du répertoire Git :

- les nouveaux fichiers qui ne sont pas encore suivis,
- les fichiers modifiés depuis la dernière version.

C'est ce que l'on voit dans le système de fichier à un instant t.

Staging area

Az Définition

La *staging area* est la zone intermédiaire entre le *working directory* et le *repository*.

Elle contient les modifications effectuées dans le *working directory* que Git va ajouter au *repository* lors du prochain commit.

Repository

Az Définition

Le *repository* (ou dépôt) correspond aux fichiers dans l'état de la dernière validation effectuée (commit).

Complément

https://git-scm.com/book/fr/v2/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-dépôt[3]

3. https://git-scm.com/book/fr/v2/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-d%C3%A9p%C3%B4t

Exercice : Application

[solution n°5 p. 28]

Vous créez un nouveau fichier dans un dossier suivi par Git, ce fichier sera présent dans :

A Le *working directory*

B La *staging area*

C Le *repository*

IX Visualiser les changements dans le working directory

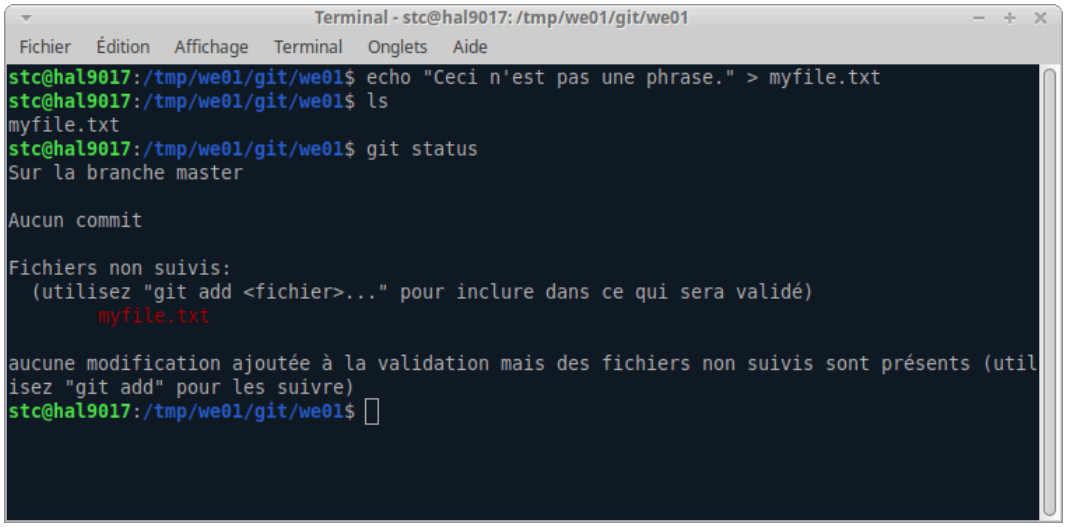
Working directory Rappel

Les trois espaces de Git : working directory, staging area, repository (cf. p.10)

Syntaxe

```
1 git status
```

Exemple



Complément

https://git-scm.com/book/fr/v2/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-dépôt[4]

4. https://git-scm.com/book/fr/v2/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-d%C3%A9p%C3%B4t

X Exercice : Application

Question 1

[solution n°6 p. 28]

Créez un fichier README.md à la racine d'un dépôt Git, ce fichier contient :

- votre nom,
- la licence de votre projet (par exemple "Licence Art Libre — https://artlibre.org").

Question 2

[solution n°7 p. 29]

Visualisez l'état de votre suivi Git avec `git status`.

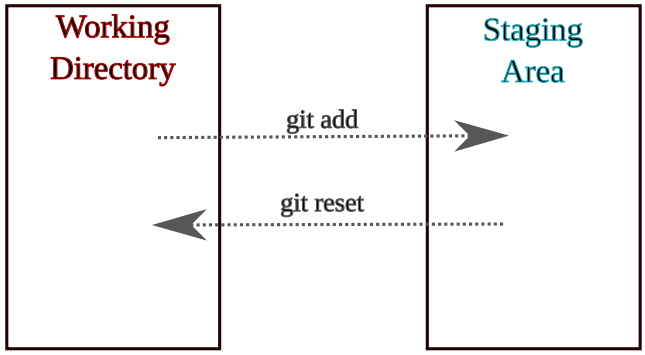
XI Suivre les fichiers (git add)

Rappel

Les trois espaces de Git : working directory, staging area, repository (cf. p.10)

Fondamental

La commande *git add* permet de déplacer un fichier depuis le *working directory* vers la *staging area*.



git add

Méthode

La commande `git add` est utilisée pour :

- préparer la validation d'un fichier qui a été créé récemment et qui n'est pas encore suivi ,
- préparer la validation des modifications apportées à un fichier déjà suivi.

git-scm.com/book/fr/v2/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-dépôt[5]

Syntaxe

```
1 git add myfile.txt
```

Méthode

La commande `git reset` permet d'annuler un `git add`.

5. https://git-scm.com/book/fr/v2/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-d%C3%A9p%C3%B4t

⚠ Attention

La commande `git reset` n'annule pas les modifications qui ont été faites, elle annule simplement le fait que le fichier est prêt à être validé.

Exercice : Application

[solution n°8 p. 29]

Exercice

Soit un fichier README.md à la racine d'un dépôt Git.

Ajouter ce fichier à la *staging area*.

Exercice

Vérifiez que le fichier a bien été ajouté à la *staging area*.

XIII Versionner les fichiers (git commit)

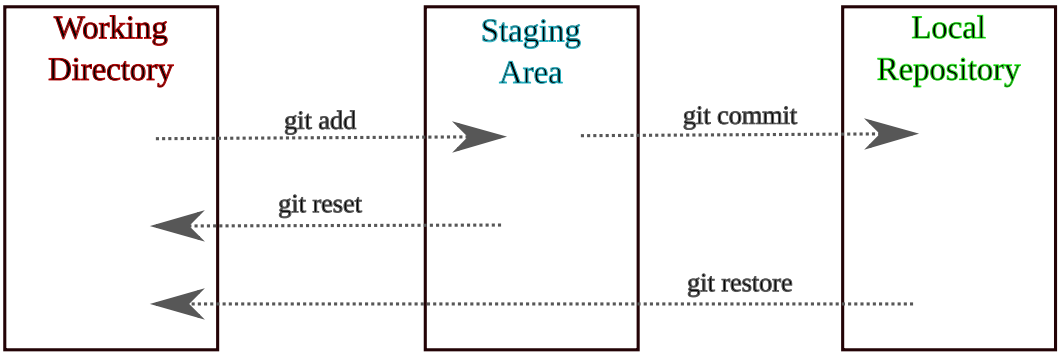
Rappel

Les trois espaces de Git : working directory, staging area, repository (cf. p.10)

Fondamental

La commande *git commit* permet de déplacer un fichier depuis la *staging area* vers le *repository* afin d'en créer une version permanente.

Une fois dans le *repository* la copie du fichier est figée, elle ne peut plus être modifiée (ni facilement supprimée), elle devient une archive que l'on pourra retrouver dans le futur telle quelle.



Méthode

- Pour effectuer un commit il faut que le ou les fichiers concernés aient été préalablement placés dans la *staging area*. L'instruction `git commit` permet donc de valider les changements qui ont été ajoutés à la *staging area*.
- Lorsque l'on effectue un commit, on doit associer un message qui résume le contenu des modifications de l'étape de validation.

Syntaxe

```
1 git commit
```

```
1 git commit -m "message"
```

Commit

Az Définition

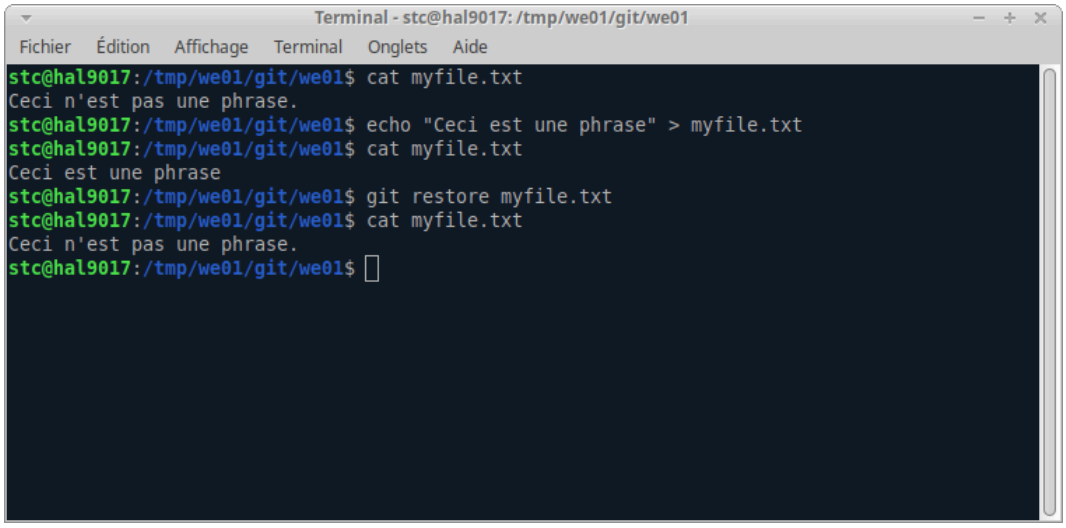
Un commit est **un point de sauvegarde** du travail.

- Chaque commit possède un identifiant unique ;
- Un commit est associé à une unique personne ;

- L'historique des commits est incrémental, out commit (excepté le premier) a un commit « père » ;
- Un commit correspond à une version figée du projet ;
- On peut naviguer dans les commits (et donc revenir en arrière).

Méthode

Ma commande `git restore` permet de remplacer la version actuelle d'un fichier par une version préalablement commitée.



Attention

La version actuelle du fichier sera définitivement remplacée.

Conseil

Dans Git on peut considérer que les fichiers qui sont dans le *working directory* et la *staging area* sont des fichiers temporaires qui peuvent facilement être altérés, et que ce qui sont dans le *repository* sont des fichiers protégés en écriture.

XIV Exercice : Application

Soit un fichier README.md à la racine d'un dépôt Git, ce fichier contient :

- votre nom,
- la licence de votre projet (par exemple "Licence Art Libre — https://artlibre.org").

Ce fichier a été ajouté à la *staging area* à l'aide de la commande `git add README.md`.

Question 1

[solution n°9 p. 29]

Valider définitivement les modifications apportées à ce fichier avec le message "Adding README".

Question 2

[solution n°10 p. 30]

Visualisez l'état de votre suivi Git avec `git status`.

Question 3

[solution n°11 p. 30]

Ajouter votre email au fichier README.md et vérifiez que le fichier a bien été détecté comme modifié par Git.

Question 4

[solution n°12 p. 30]

Validez cette nouvelle modification avec le message "Adding email to REAME".

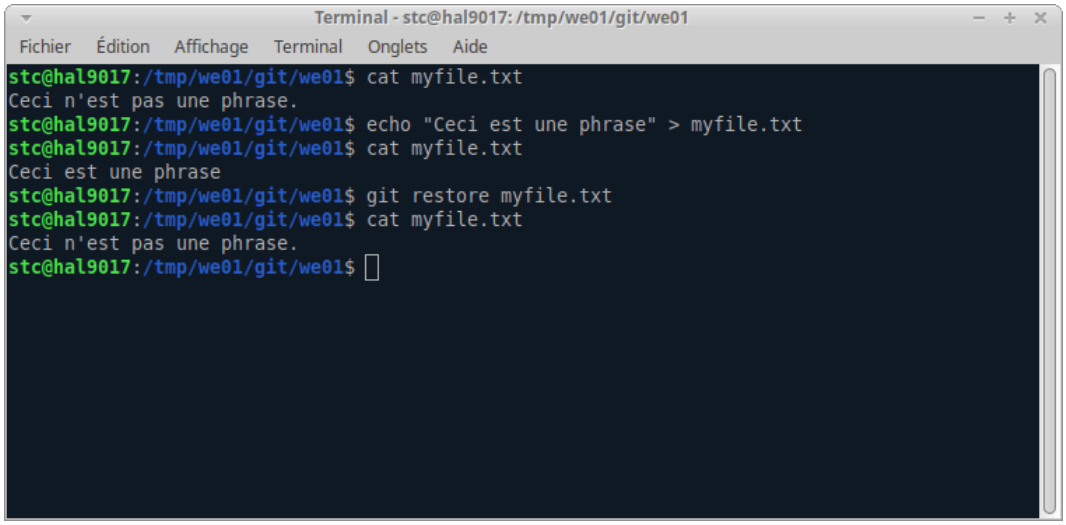
XV Afficher l'historique des commits (git log)

Syntaxe

Le `git log` permet de voir l'historique de tous les commits effectués sur un *repository*.

```
1 git log
```

Exemple



On peut voir ici :

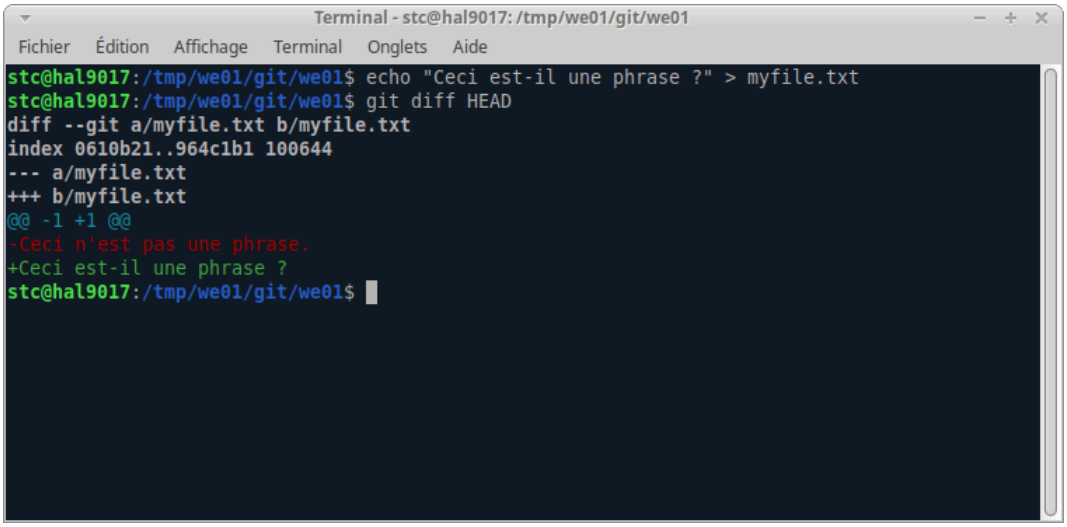
- L'identifiant unique des commits ;
- Les auteurs ;
- Les dates des commit ;
- Les messages qui ont été entrés lors des commits.

Syntaxe

Le `git diff` permet de voir les modifications apportées au *working directory* :

- depuis l'état du *staging area* : `git diff`
- depuis le dernier commit : `git diff HEAD`
- depuis un commit quelconque : `git diff id_commit`

Exemple



On peut observer sur cet exemple que entre le *working directory* et le dernier commit :

- le `myfile.txt` a été modifié,
- la ligne `Ceci n'est pas une phrase.` a été supprimée de ce fichier,
- la ligne `Ceci est-il une phrase ?` a été ajoutée à ce fichier.

Méthode

Les instruction `git log` et `git diff` peuvent être suivies du nom d'un fichier afin de n'afficher l'historique ou les différences qui concernent ce fichier.

Exemple

On peut observer sur cet exemple qu'entre *working directory* et le commit `30676ede3ab1aaedc3cd8a6f8abc749efa552858`, à propos du fichier README.md, la ligne `stephane.crozat@utc.fr` a été ajoutée.

Exercice : Application

[solution n°13 p. 30]

Exercice

Quelle est la commande qui a permis d'obtenir les informations suivantes ?

```
commit 51bbf4b54c4c853c13bd23057deebb958c12e1a2 (HEAD -> master)
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Mon Nov 16 01:40:24 2020 +0100

    Renommage des personnages et villes

commit 85cca47699b1eecc18417aad2a8ca47ddd03c766
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Sat Nov 14 01:04:20 2020 +0100

    Recalage temporel des chapitres 11, 12, 13 et 14

commit e2e7e37abee61dc8186a6ee645951da54eb951e6
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Fri Nov 13 00:32:31 2020 +0100

    Début du chapitre 15 (à finir)

commit 0aae619cd23c85647f1f72dade4045091b0c0a7b
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Thu Nov 12 02:42:21 2020 +0100

    Refonte du chapitre 11 (Ajout d'Ada)
```

Exercice

Le fichier `chap11.md` a été modifié sur le disque local.

Comment afficher ce qui a été modifié dans ce fichier depuis le dernier commit ?

XVII Restaurer des versions (git checkout)

Fondamental

Le `git checkout` permet de se déplacer sur un commit de l'historique.

Cette commande va restaurer l'ensemble du projet dans l'état dans lequel il était au moment de ce commit :

- les fichiers créés depuis seront supprimés,
- les fichiers supprimés depuis seront restaurés,
- les modifications faites depuis seront annulées.

Cette fonction permet de donc de remonter dans le passé de n'importe quel état du projet.

Syntaxe

```
1 git checkout identifiant
```

Remarque

La commande `git checkout master` permet de revenir à l'état du dernier commit (donc à l'état présent du présent).

Exemple

```
1 git checkout 30676ede3ab1aaedc3cd8a6f8abc749efa552858
```

Cette commande va restaurer l'ensemble du projet dans l'état dans lequel il était au moment du commit `30676ede3ab1aaedc3cd8a6f8abc749efa552858`.

Attention

Pour que cette commande soit inoffensive, il est nécessaire que le toutes les modifications courantes du projet aient été commitées.

En effet :

- la commande `git checkout` utilisée pour retourner à un état antérieur, va supprimer l'état présent,
- la commande `git checkout` pourra ensuite être utilisée pour revenir au dernier état commité.

Mais ce qui n'a pas été commité ne pourra plus jamais être retrouvé.

Remarque

Si la commande `git checkout` est lancée sur un projet qui comporte des modifications en cours non validées, Git imposera d'annuler ces modifications ou de les commiter avant de pouvoir effectuer le *checkout* (afin de protéger l'utilisateur d'une perte d'information).

HEAD~1, HEAD~2...

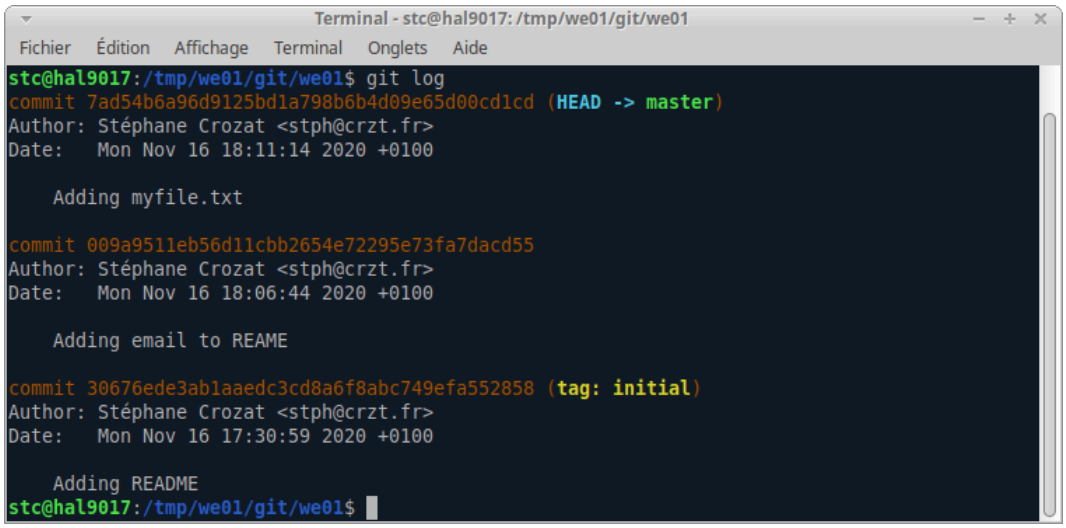
Complément

La commande `git checkout HEAD~1` permet de retourner à la dernière version validée avant celle actuelle (version actuelle "moins une"). De même `git checkout HEAD~2` pour retourner deux versions en arrière, etc.

git tag

Complément

La commande `git tag montag identifiant` permet d'associer un tag à une version afin de la retrouver plus facilement ensuite.



Exercice : Application

[solution n°14 p. 31]

Exercice

Soit l'historique suivant :

```
commit 51bbf4b54c4c853c13bd23057deebb958c12e1a2 (HEAD -> master)
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Mon Nov 16 01:40:24 2020 +0100

    Renommage des personnages et villes

commit 85cca47699b1eecc18417aad2a8ca47ddd03c766
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Sat Nov 14 01:04:20 2020 +0100

    Recalage temporel des chapitres 11, 12, 13 et 14

commit e2e7e37abee61dc8186a6ee645951da54eb951e6
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Fri Nov 13 00:32:31 2020 +0100

    Début du chapitre 15 (à finir)

commit 0aae619cd23c85647f1f72dade4045091b0c0a7b
Author: Stéphane Crozat <stph@crzt.fr>
Date:   Thu Nov 12 02:42:21 2020 +0100

    Refonte du chapitre 11 (Ajout d'Ada)
```

Écrivez la commande qui permet de revenir à l'état antérieur au renommage des personnages et des villes.

Exercice

Écrivez la commande qui permet de revenir à l'état actuel du projet, après le renommage des personnages et des villes.

XIX Mettre du code de côté

Git stash

On a vu précédemment que Git est constitué de 3 espaces :

- Le *Working Directory*
- Le *Staging Area*
- Le *Local Repository*

En réalité, il en existe un autre moins important : la *Stash*.

Git Stash Syntaxe

La commande `git stash [push]` permet de mettre de côté des modifications du *Working Directory*, puis revenir à l'état du dernier commit (HEAD). Cela est utile, par exemple, avant de se déplacer dans l'arborescence des modifications avec `git checkout`.

Pour rétablir ces modifications, il suffit d'utiliser la commande `git stash pop`.

Pour visualiser à tout moment le contenu de la *Stash*, on utilise la commande `git stash show`.

Remarque

Il est possible d'enchaîner plusieurs `git stash` pour sauvegarder différents états «sales» ; `git stash pop` restaurera toujours le **dernier état sauvegardé** : C'est le principe d'une pile.

Solutions des exercices

Solution n°1

[exercice p. 4]

Installez Git sur votre machine et exécutez la commande suivante :

```
1 git help
```

En utilisant cette commande trouver la commande qui permet de "Afficher l'état de la copie de travail".

git status

Solution n°2

[exercice p. 7]

Solution n°3

[exercice p. 7]

```
1 stc@hal9017:~$ git config -l
2 user.name=Stéphane Crozat
3 user.email=stph@crzt.fr
4 core.editor=nano
```

Solution n°4

[exercice p. 9]

```
1 mkdir git
2 cd git
3 mkdir we01
4 cd we01
5 git init
```

```
1 Dépôt Git vide initialisé dans /tmp/we01/git/we01/.git/
```

Solution n°5

[exercice p. 12]

Vous créez un nouveau fichier dans un dossier suivi par Git, ce fichier sera présent dans :

- **A** Le *working directory*
- **B** La *staging area*
- **C** Le *repository*

Solution n°6

[exercice p. 14]

```
1 nano README.md
```

Solution n°7

[exercice p. 14]

Solution n°8

[exercice p. 17]

Exercice

Soit un fichier README.md à la racine d'un dépôt Git.

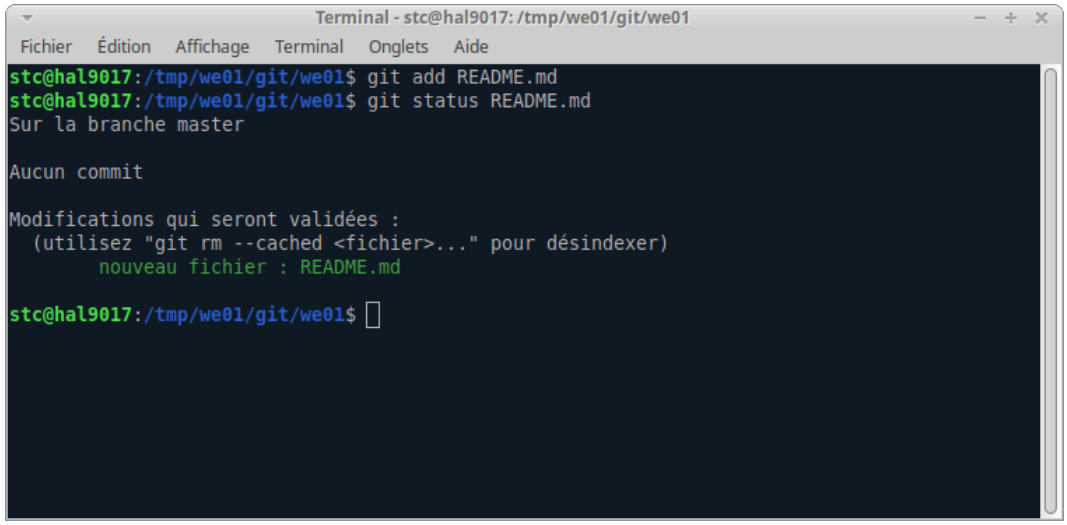
Ajouter ce fichier à la *staging area*.

git add README.md

Exercice

Vérifiez que le fichier a bien été ajouté à la *staging area*.

git status README.md



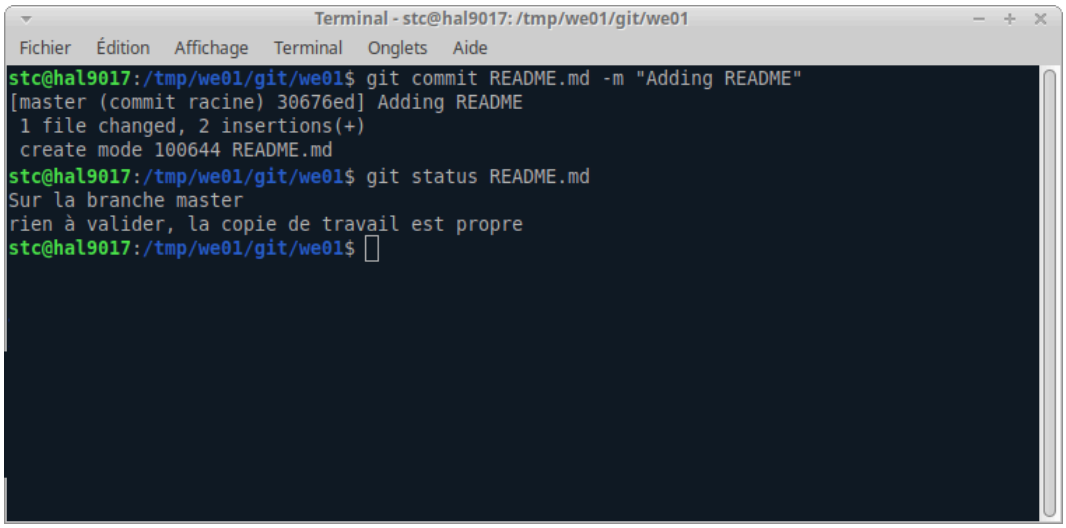
Solution n°9

[exercice p. 20]

```
1 git commit -m "Adding README"
```

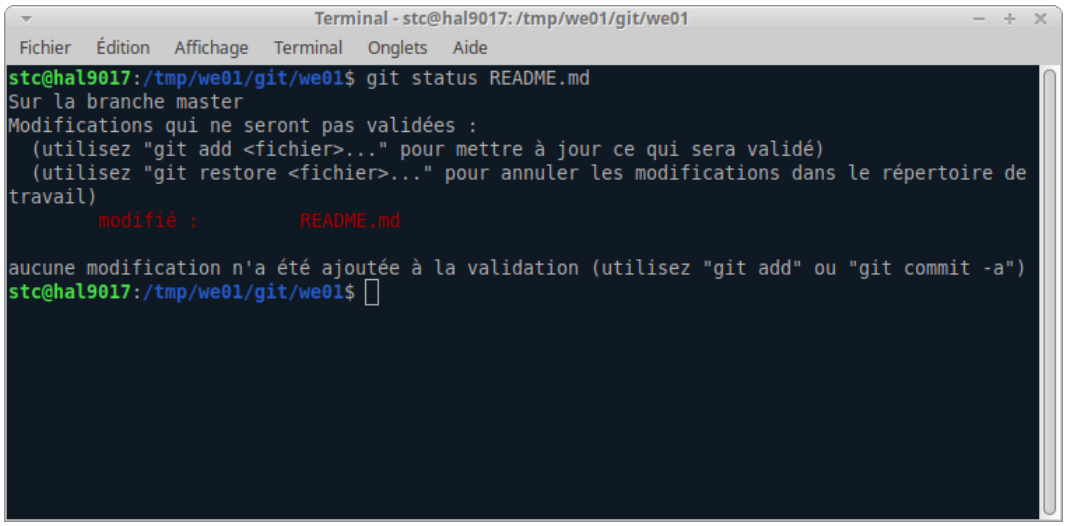
Solution n°10

[exercice p. 20]



Solution n°11

[exercice p. 20]



Solution n°12

[exercice p. 20]

```
1 git add README.md
2 git commit -m "Adding email to REAME"
```

Solution n°13

[exercice p. 23]

Exercice

Quelle est la commande qui a permis d'obtenir les informations suivantes ?

```
1 commit 51bbf4b54c4c853c13bd23057deebb958c12e1a2 (HEAD -> master)
2 Author: Stéphane Crozat <stph@crzt.fr>
3 Date:   Mon Nov 16 01:40:24 2020 +0100
4
5     Renommage des personnages et villes
6
7 commit 85cca47699b1eecc18417aad2a8ca47ddd03c766
8 Author: Stéphane Crozat <stph@crzt.fr>
9 Date:   Sat Nov 14 01:04:20 2020 +0100
```

```
10
11      Recalage temporel des chapitres 11, 12, 13 et 14
12
13  commit e2e7e37abee61dc8186a6ee645951da54eb951e6
14  Author: Stéphane Crozat <stph@crzt.fr>
15  Date:   Fri Nov 13 00:32:31 2020 +0100
16
17      Début du chapitre 15 (à finir)
18
19  commit 0aae619cd23c85647f1f72dade4045091b0c0a7b
20  Author: Stéphane Crozat <stph@crzt.fr>
21  Date:   Thu Nov 12 02:42:21 2020 +0100
22
23      Refonte du chapitre 11 (Ajout d'Ada)
```

git log

Exercice

Le fichier `chap11.md` a été modifié sur le disque local.

Comment afficher ce qui a été modifié dans ce fichier depuis le dernier commit ?

git diff HEAD

Solution n°14

[exercice p. 26]

Exercice

Soit l'historique suivant :

```
1  commit 51bbf4b54c4c853c13bd23057deebb958c12e1a2 (HEAD -> master)
2  Author: Stéphane Crozat <stph@crzt.fr>
3  Date:   Mon Nov 16 01:40:24 2020 +0100
4
5      Renommage des personnages et villes
6
7  commit 85cca47699b1eecc18417aad2a8ca47ddd03c766
8  Author: Stéphane Crozat <stph@crzt.fr>
9  Date:   Sat Nov 14 01:04:20 2020 +0100
10
11     Recalage temporel des chapitres 11, 12, 13 et 14
12
13 commit e2e7e37abee61dc8186a6ee645951da54eb951e6
14 Author: Stéphane Crozat <stph@crzt.fr>
15 Date:   Fri Nov 13 00:32:31 2020 +0100
16
17     Début du chapitre 15 (à finir)
18
19 commit 0aae619cd23c85647f1f72dade4045091b0c0a7b
20 Author: Stéphane Crozat <stph@crzt.fr>
21 Date:   Thu Nov 12 02:42:21 2020 +0100
22
23     Refonte du chapitre 11 (Ajout d'Ada)
```

Écrivez la commande qui permet de revenir à l'état antérieur au renommage des personnages et des villes.

git checkout 85cca47699b1eecc18417aad2a8ca47ddd03c766

Exercice

Écrivez la commande qui permet de revenir à l'état actuel du projet, après le renommage des personnages et des villes.

```
git checkout 51bbf4b54c4c853c13bd23057deebb958c12e1a2
```