Chapitre 3
Création d'un dépôt

1. Créer un dépôt local

Pour tout nouveau projet que l'on souhaite versionner, il est nécessaire de créer un nouveau dépôt. C'est ensuite dans ce dépôt que Git stockera toutes nos informations. Pour cela, il faut se placer dans le dossier racine du projet. Par exemple, si le projet est un site web simple, nous pouvons imaginer que le dossier qui contiendra le projet se nomme **www**. Dans notre cas, le dossier qui va contenir le code du projet se nomme **depot**. Il faut donc se placer dans ce dossier et exécuter la commande suivante :

```
git init
```

Git confirme la création du dépôt avec le message suivant :

```
Initialized empty Git repository in /Users/dauzon/Projets/depot/.git/
```

Cette action va créer un dossier nommé *.git* à la racine du projet. Sur la plupart des explorateurs de fichiers, ce dossier est par défaut invisible. Vous pouvez utiliser la ligne de commande pour visualiser ce dossier avec la commande `ls -la` sous les systèmes Linux ou macOS, et sous Windows en utilisant l'outil Cygwin.

Par défaut, la création d'un dépôt crée automatiquement une branche *master*. Pour changer le nom de cette branche, il est possible de définir l'option de configuration `init.defaultBranch` comme dans l'exemple ci-dessous.

```
git config --global init.defaultBranch main
touch README.md
git init
Dépôt Git vide initialisé dans /Users/dauzon/git3eme/.git/
git add README.md
git commit -m "README : add file"
[main (commit racine) e18c7a7] README : add file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
git branch
* main
```

2. Le contenu du dossier .git

Le dossier *.git* héberge tout le contenu du dépôt utilisé par Git. Nous allons visualiser les dossiers et fichiers présents dans ce dossier. Pour cela, il faut se placer dans le répertoire *.git* en utilisant la commande `cd` :

```
cd .git
ls -la
```

Le système d'exploitation affiche alors une liste de fichiers et de dossiers :

```
-rw-r--r--   1 dauzon  staff   23  8 jui 23:30 HEAD
drwxr-xr-x   2 dauzon  staff   68  8 jui 23:30 branches
-rw-r--r--   1 dauzon  staff  137  8 jui 23:30 config
-rw-r--r--   1 dauzon  staff   73  8 jui 23:30 description
drwxr-xr-x  11 dauzon  staff  374  8 jui 23:30 hooks
drwxr-xr-x   3 dauzon  staff  102  8 jui 23:30 info
drwxr-xr-x   4 dauzon  staff  136  8 jui 23:30 objects
drwxr-xr-x   4 dauzon  staff  136  8 jui 23:30 refs
```

Chacun de ces fichiers ou dossiers contient des éléments que Git utilise pour suivre notre code. Voici une liste des fichiers et dossiers et leur contenu (certains termes peuvent paraître obscurs, mais ils seront définis dans la suite de ce livre) :

– *HEAD* : contient la référence du commit à partir duquel on travaille. C'est une référence qui est dépendante de la branche sur laquelle nous sommes situés. Dans la suite de ce livre (et dans de nombreux cas), HEAD désignera le commit le plus récent d'une branche et par défaut le commit le plus récent de la branche courante.
– *branches* : ce dossier contient les branches du projet. Les branches sont des versions qui divergent du développement principal et sont particulièrement utiles pour développer de nouvelles fonctionnalités sans créer de conflit avec la version stable du projet. Les branches seront abordées plus en détail dans le chapitre Les branches et les tags.
– *config* : ce fichier contient les éléments de configuration et les alias propres au dépôt.
– *description* : ce fichier sera affiché lors de l'utilisation de l'interface web GitWeb.
– *hooks* : ce dossier contient les hooks du dépôt. Les hooks sont des mécanismes de contrôle utilisés par Git. Il est possible de créer ses propres hooks pour répondre à des besoins spécifiques.
– *info* : ce dossier contient un fichier *exclude* utilisé pour demander à Git d'ignorer les fichiers spécifiés. Une autre possibilité existe pour ignorer des fichiers, elle sera abordée dans le chapitre Manipulation des fichiers et commit, à la section Manipuler les fichiers/Ignorer des fichiers.
– *objects* : ce dossier contient tous les objets de notre projet, c'est-à-dire que c'est dans ce dossier que seront placées toutes les informations concernant nos fichiers, dossiers, commits, ou tout autre objet que Git a besoin de traiter.
– *refs* : ce dossier comprend les références qui pointent vers des commits. Ces références sont en réalité soit des branches, soit des tags qui seront abordés dans le chapitre Les branches et les tags.

Ces dossiers et fichiers forment le dépôt réel. En utilisant normalement Git sans chercher à en comprendre les mécanismes internes, un utilisateur n'a jamais besoin d'aller dans ces dossiers.

3. Le fichier README

Une bonne pratique lors de la création d'un dépôt consiste à expliquer le projet dans un fichier README à la racine du projet. Ce fichier est celui que vous consultez sur la page GitHub principale d'un projet. Comme exemple, il est possible de regarder les pages GitHub des projets de Django et de Bootstrap en consultant les liens suivants :

– https://github.com/django/django

– https://github.com/twbs/bootstrap

Sur ces pages se trouve un bloc contenant les dossiers et les fichiers à la racine du projet. En dessous de ce bloc se trouve le bloc nommé *README.md* qui contient la présentation du projet. Le fichier *README* de Bootstrap porte l'extension *md* (pour Markdown) et celui de Django porte l'extension *rst* (pour *reStructuredText*). Ce sont deux formats abordés dans la suite de ce chapitre.

La présentation d'un projet est très importante et ne doit pas être négligée. Cette présentation est surtout à destination des développeurs qui utiliseront ou participeront au projet. Elle doit être très claire et contenir plusieurs parties :

– Une partie présentant le projet : ses fonctionnalités, ses dépendances, etc. Cette partie doit être la plus compréhensible possible. Même un utilisateur débutant doit comprendre cette partie sans difficulté.

– Une partie expliquant très rapidement l'utilisation du projet. Si c'est une bibliothèque, il faut présenter en 5 à 10 lignes de code la manière de l'utiliser. Cette partie ne sert pas de documentation aux futurs utilisateurs, mais elle leur facilitera la prise en main.

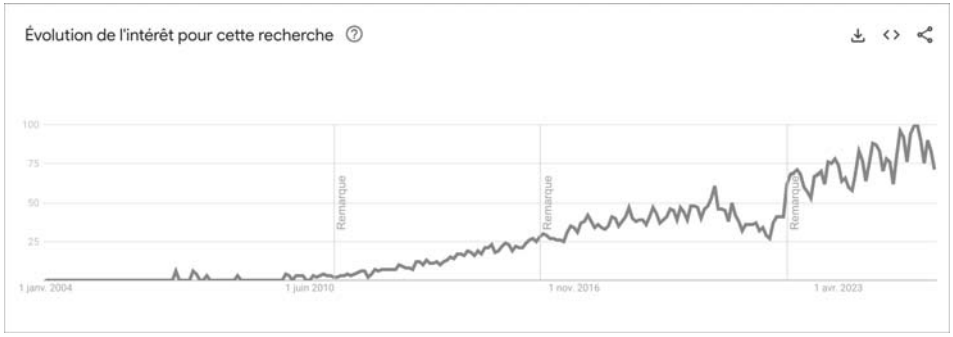
– Des liens ressources : vers une documentation plus complète, vers le site officiel, etc.

– Les informations à destination des potentiels contributeurs au projet. Les contributeurs sont des développeurs qui acceptent bénévolement de participer au projet, mais pour de nombreux dépôts ils doivent suivre une procédure précise pour soumettre leurs modifications.
– La liste des auteurs et contributeurs au projet. Cette partie peut également, selon les projets, se trouver dans un fichier *AUTHORS* à la racine du projet.
– La ou les licences que nécessite le projet (licences libres ou restrictives). Cette partie peut également, selon les projets, se trouver dans un fichier *LICENSE* à la racine du projet.
– Les moyens de contacter les mainteneurs du projet ou des informations diverses.

4. Markdown

4.1 Présentation

Markdown est un langage de présentation qui gagne en popularité depuis plusieurs années, comme le montre le graphique Google Trends ci-dessous :



Source : https://trends.google.fr/trends/explore?date=all&q=markdown

Ce langage n'est pas un langage de présentation avec une structure XML, comme l'est HTML. Ce langage a été conçu dans le but d'être lisible directement en mode texte, ce qui l'a rendu très populaire chez les développeurs. En effet, le format Markdown peut être consulté en mode texte et donc peut facilement être versionné. Il est d'ailleurs possible d'exporter des fichiers Markdown facilement vers du code HTML ou d'autres formats.

La syntaxe officielle est documentée sur le lien suivant :
https://daringfireball.net/projects/markdown/syntax

Certaines syntaxes ne sont pas officielles, mais sont généralement bien supportées par les logiciels et bibliothèques s'interfaçant avec Markdown. Elles permettent d'ajouter des fonctionnalités absentes du format officiel. Les fichiers Markdown portent généralement l'extension *.md*.

Voici un exemple de fichier Markdown très simple, présentant les caractéristiques d'un jeu fictif, et l'affichage formaté de ce fichier sur GitHub :

```
# Stratégiii : le meilleur de la stratégie

## Fonctionnalités

+ 64 unités différentes !
+ La **meilleure IA** jamais vue !
```

Voici la manière dont ce fichier sera affiché sous GitHub :

4.2 Éléments de syntaxe

4.2.1 Titres

En Markdown, il est possible d'utiliser les titres sur six niveaux comme en HTML. Ces titres sont définis par un nombre précis de dièses au début du titre.

Par exemple, un titre principal (équivalent à `h1` en HTML) se note donc ainsi :

```
# Titre de premier niveau
```

Un sous-titre (titre de deuxième niveau, équivalent à `h2` en HTML) se note ainsi :

```
## Titre de deuxième niveau
```

Le nombre de dièses devant le titre détermine ainsi son niveau.

4.2.2 Listes non ordonnées

Pour établir des listes à puces (équivalent de la balise `ul` en HTML), il faut utiliser un des caractères suivants `+ * -` au début de chaque ligne formant la liste.

Par exemple, voici une liste de trois éléments :

```
+ 64 unités différentes !
+ La meilleure IA jamais vue !
+ Un jeu compatible avec toutes les plateformes !
```

4.2.3 Listes ordonnées

Pour établir des listes ordonnées (équivalent de la balise `ol` en HTML), il faut commencer chaque ligne de cette liste par le numéro de l'élément dans cette liste.

Voici un exemple fictif de liste d'actions à effectuer pour contribuer au projet :

```
1. S'inscrire sur la liste de diffusion.
2. Lire la liste des règles de code.
3. Soumettre le patch aux administrateurs du projet.
```

4.2.4 Mettre en gras

Pour mettre en gras un mot ou une partie de phrase, il faut l'entourer de doubles étoiles **. Par exemple, dans la ligne suivante le mot « évolué » sera affiché en gras :

```
Le jeu le plus **évolué** de tous les temps.
```

4.2.5 Mettre en italique

Pour mettre en italique un mot ou une partie de phrase, il faut l'entourer d'étoiles *. Par exemple, dans la ligne suivante la partie « le meilleur jeu de tous les temps » sera affichée en italique :

```
Selon la presse Stratégiii est *le meilleur jeu de tous les temps*.
```

4.2.6 Ligne horizontale

Pour séparer différentes parties d'un document, une ligne horizontale (équivalente à <hr /> en HTML) pourra être utilisée. Il faut utiliser l'un des trois caractères suivants _ - * et le répéter au minimum trois fois sur une même ligne, par exemple :

```
---
```

Cette suite de caractères affichera une ligne horizontale.

4.2.7 Code

Pour insérer du code il faut l'entourer de trois backtick : ```. Il est également possible (en fonction des convertisseurs Markdown) de spécifier le langage après les premiers ```. Par exemple, voici une balise HTML simple :

````
```html
<button>Clique-moi !</button>
```
````

Pour insérer une petite ligne de code dans une phrase standard, il faut utiliser un seul backtick, par exemple :

```
Pour accéder à l'objet Utilisateur de Django il faut utiliser la
propriété `user` de l'objet `request`.
```

Pour insérer du code dans une liste, il faut l'indenter de huit espaces ou de deux tabulations.

4.2.8 Tableaux

Il est également possible d'insérer des tableaux en Markdown. Un exemple sera plus compréhensible et lisible qu'une longue explication :

```
|Prénom|Nom|
|---|--------------|
|Guido|Van Rossum|
|Dennis|Ritchie|
```

Voici le rendu sur GitHub :

Prénom	Nom
Guido	Van Rossum
Dennis	Ritchie

4.2.9 Liens

Des liens peuvent également être générés avec Markdown. Pour cela, il faut utiliser la syntaxe suivante :

```
[Texte du lien](URL du lien)
```

Par exemple, voici un lien qui redirigera vers le moteur de recherche Google :

```
[Lien de Google](https://google.fr)
```

4.2.10 Notes de bas de page

Insérer des notes de bas de page peut être utile pour ajouter des informations à la documentation sans surcharger la lecture. Voici un exemple de phrase qui contient une note de bas de page :

```
Git est un SCM[^1] très utilisé.
[^1]: Système de gestion de versions (Source Content Management)
```