Formation GIT

Table des matières

Présentation du cours

Enseignant

Lawrence Terpin

Entrepreneur, concepteur de sites web & d'applications mobiles

Dans les métiers du web depuis 2012

Résumé

Ce cours s'adresse à ceux qui souhaitent utiliser et comprendre la logique du versioning pour sauvegarder leurs fichiers à distance notamment et maitriser GIT.

A la fin de cette formation, vous saurez utiliser GIT, travailler en collaboration sur un même projet et sauvegarder vos fichiers et les commandes essentielles !

1. Qu’est-ce que le versioning ?

Le versioning, comme son nom l’indique signifie versionner ses fichiers.

Le fait de versionner ses fichiers fait que l’on sauvegarde une certaine version tandis qu’on en modifie une autre. Cela créé un historique de ses fichiers, sur lequel on peut naviguer pour revenir à une ancienne version.

Un logiciel de versioning, ou logiciel de gestion de version est un logiciel qui permet de conserver un historique des modifications effectuées sur un projet afin de pouvoir rapidement identifier les changements effectués et de revenir à une ancienne version en cas de problème.

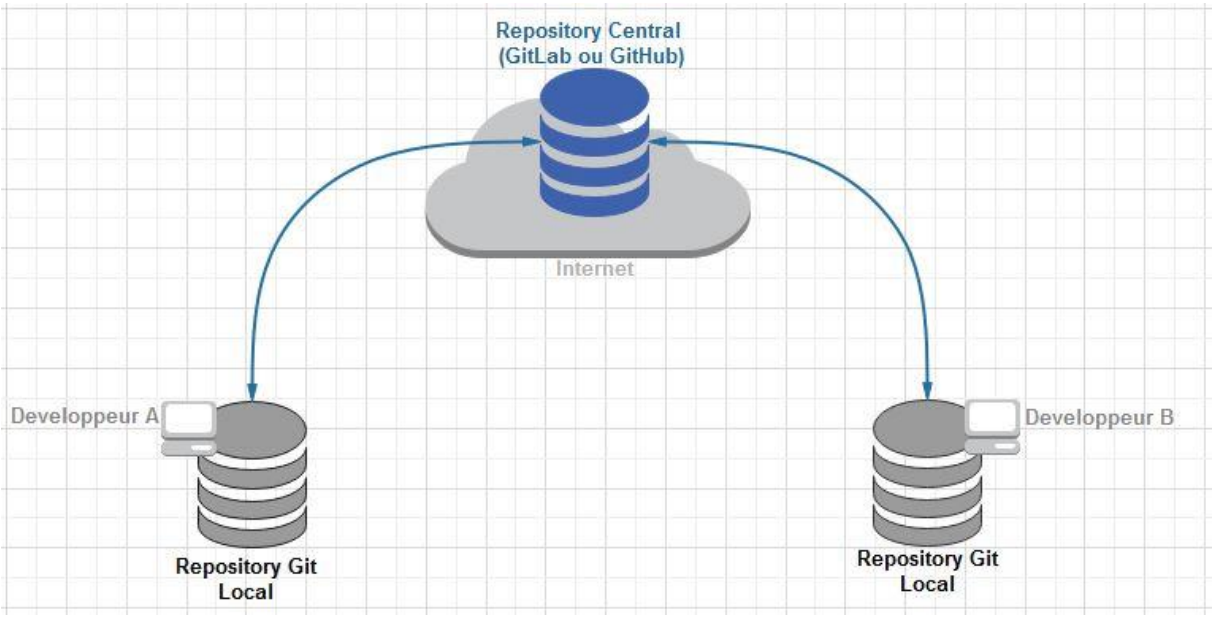
Les logiciels de gestion de versions sont quasiment incontournables aujourd’hui car ils facilitent grandement la gestion de projets et ils permettent de travailler en équipe de manière beaucoup plus efficace.

4. Présentation de GIT

Git est un logiciel de versioning créé en 2005 par Linus Torvalds, le créateur de Linux.

Parmi les logiciels de gestion de versions, Git est le leader incontesté et il est donc indispensable pour tout développeur de savoir utiliser Git.

Il suffit de l'installer sur sa machine et d'utiliser les commandes qu'il fournit.



Chaque personne récupère une première fois le dépôt distant ou repository du projet et travaille depuis sa version locale.

Chaque personne peut envoyer son travail sur la version distante à tout moment.

Cela permet d'avoir une sauvegarde permanente de notre projet.

5. Les repositories

Git est un logiciel de gestion de version. Git va nous permettre d'enregistrer les différentes modifications effectuées sur un projet et de pouvoir retourner à une version précédente du projet.

Dans le langage des systèmes de gestion de version, la copie de l'intégralité des fichiers d'un projet et de leur version située sur le serveur central est appelé un dépôt. Git appelle également cela "repository" ou "repo" en abrégé.

GitHub, Gitlab, Bitbucket par-exemple sont des services en ligne qui permettent d'héberger des dépôts ou repo Git.

Une grande partie des dépôts hébergés sont publics, ce qui signifie que n'importe qui peut télécharger le code de ces dépôts et contribuer à leur développement en proposant de nouvelles fonctionnalités.

Pour récapituler, et afin d'être bien clair sur ce point : Git est un logiciel de gestion de version tandis que GitHub notamment est un service en ligne d'hébergement de dépôts Git qui fait office de serveur central pour ces dépôts.

Nous verrons comment utiliser GitHub en détail dans cette formation après avoir appris à utiliser Git.

6. Installation de GIT

Avant d'installer GIT, il faut le télécharger.

Lien de téléchargement :

https://git-scm.com/downloads

GIT est téléchargeable sur Windows, MAC, Linux.

Voici un aperçu de la page de téléchargement :

Après avoir téléchargé GIT, on peut cliquer sur l'exécutable pour l'installer.

Suivez les étapes d'installation en laissant les paramètres par défaut.

Ensuite, vérifiez que l'installation s'est bien effectuée, en tapant cette commande dans le terminal :

- **git --version**

Si cette commande retourne le numéro de version de GIT installée sur votre système d'exploitation c'est que tout est prêt.

Sinon, redémarrer votre PC afin de bien finaliser l'installation.

7. Créer le repository GIT

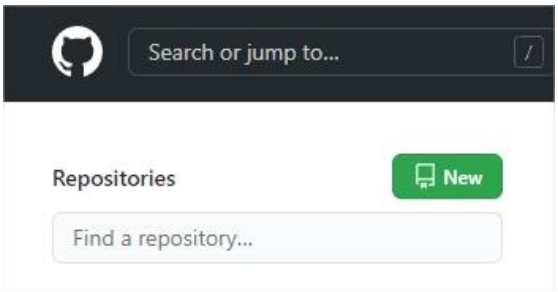
Avant d'utiliser GIT, il faut tout d'abord créer un repository, afin de déposer nos fichiers.

Nous allons créer ce repository sur GITHUB.

Rendez vous sur la page **https://github.com/** et connectez-vous.

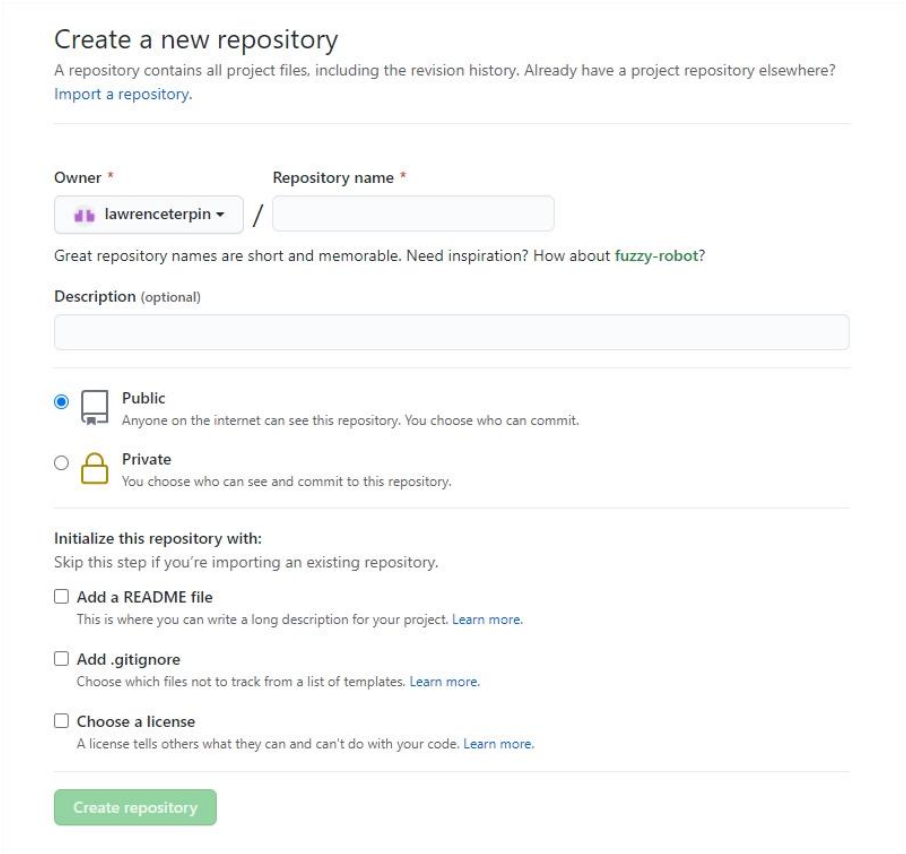
Après vous être connecté sur GITHUB, vous serez redirigé vers votre tableau de bord, la page d'accueil de vos projets.

Vous pouvez désormais créer votre repository depuis le volet de gauche.



Cliquez sur le bouton vert « **New** ».

Vous arrivez sur la page de création du repository.



Dans un premier temps, donnez un **nom** à votre repository (Ex : **formation-git**).

Le nom de votre repository ne doit comporter ni de caractères spéciaux, ni espaces, ni majuscules.

Dans un second temps, vous pouvez renseigner la **description**. Ce champ est facultatif.

Ensuite, il s'agit de choisir le **type de visibilité** de votre repository (public ou privé).

Nous allons choisir le type privé.

Un repository public sera vu par tout le monde.

Un repository privé sera visible uniquement des collaborateurs du projet. Au préalable invité par l'administrateur.

Enfin, nous pouvons choisir quelques **options par défaut** :

- **La création d'un fichier README.**

Le fichier **README ou lisez-moi**, permet de décrire notre projet et de donner des instructions d'installation notamment.

- **La création d'un fichier .gitignore**

Le fichier **.gitignore** permet d'ignorer des éléments dans notre projet. Comme par-exemple des fichiers, dossiers, ou extensions de fichiers.

- **La création d'une license**

Comme son nom l'indique le fichier de **license** permet de signer le projet et de protéger le code source notamment.

Nous pouvons désormais créer le repository, en cliquant sur le bouton vert « **Create repository** »

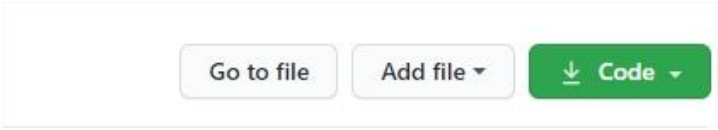
Après notre repository créé, nous sommes redirigés vers la page d'accueil du repository, vide par défaut.

8. Récupérer le repository GIT sur le pc

Il s'agit maintenant de récupérer ce repository sur notre ordinateur.

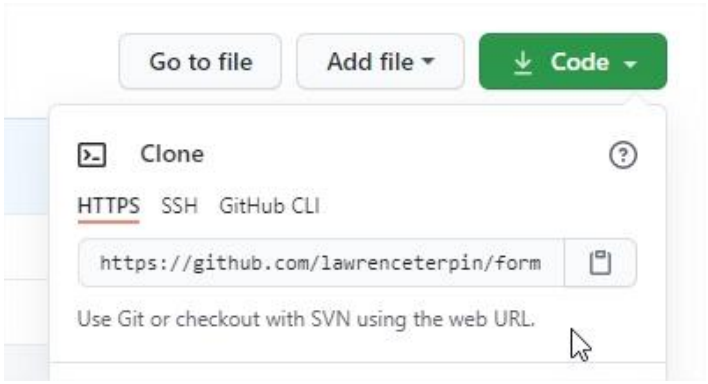
En haut de notre repository, se trouvent un menu proposant plusieurs possibilités, notamment de récupérer l'url de ce dernier.

Pour récupérer l'url du repository, cliquez sur le bouton vert « **Code** ».



En cliquant sur ce bouton, nous pouvons récupérer le repository.

Dans l'onglet **HTTPS**, copiez l'url :



Ensuite, ouvrez votre terminal de votre PC et rendez vous dans le dossier **Documents/** de votre ordinateur, en tapant cette commande par-exemple :

```
➢  cd ~/Documents/
```

Ou,

```
➢  cd /Users/nom_d_utilisateur/Documents/
```

Dans le dossier **Documents**, tapez la commande :

```
➢  git clone https://github.com/lawrenceterpin/formation-git.git
```

Cette commande va téléchargez le repository à l'emplacement où vous vous trouvez dans le terminal.

Le fait de cloner votre repository va créer un nouveau dossier, portant le même nom que votre repository, en l'occurrence **formation-git**.

C'est le dossier de notre repository local et c'est ici que l'on va travailler et ajouter nos fichiers.

Ce repository local est connecté à notre repository distant et peut désormais comparer toutes les modifications d'un côté ou de l'autre.

Après avoir cloné le repository, vous devriez donc retrouver cette arborescence :

- **Documents**
  - **formation-git**

Maintenant, placez-vous dans le dossier **formation-git**, en tapant cette commande dans votre terminal par-exemple :

```
➢  cd formation-git/
```

9. Les premières commandes GIT

9.1 Visualiser les modifications (git status)

Dans notre dossier **formation-git**, créez un fichier **bonjour.txt.**

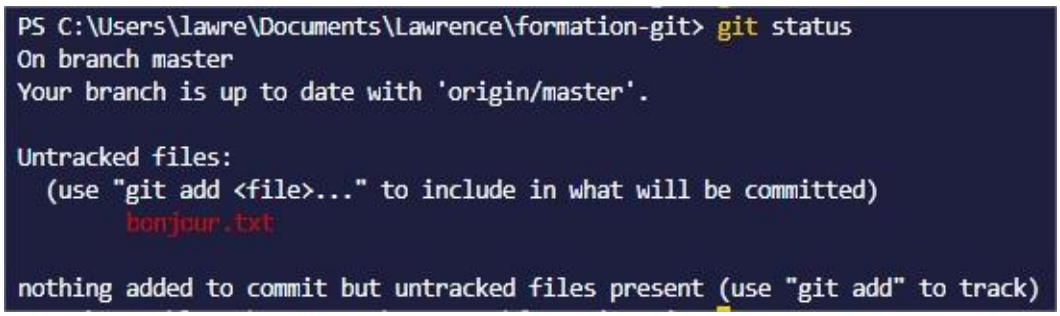
Ensuite, ouvrez votre terminal dans le dossier formation-git et tapez la commande :

➢ **git status**

Cette commande permet de visualiser le statut de notre branche locale et détecte les modifications à l'intérieur de celle-ci.

Nous avons précédemment créé un nouveau fichier **bonjour.txt.**

Ce nouveau fichier est donc visible en rouge :



La commande git status est à la première à taper systématiquement avant toute chose.

Cette commande **git status** donne aussi quelques indications, comme :

- Le nom de notre branche courante (master par défaut).
- La commande à tapez pour ajouter en mémoire notre fichier.
- Aider à finaliser des process GIT, comme des conflits, des commits ou des push.

Cette commande est à taper systématiquement avant toute autre commande GIT dans notre repository local, afin de visualiser toutes les modifications.

9.2 Ajouter des modifications (git add)

Après avoir visualiser nos modifications avec la commande **git status**, on peut désormais ajouter ces dernières en mémoire.

Pour ajouter ces modifications, tapez la commande :

```
➢ git add .
```

La commande **git add** peut être écrite de plusieurs façons :

- Soit en spécifiant un élément précis comme un fichier ou un dossier, (Ex : **git add bonjour.txt**)
- Soit en sélectionnant tous les éléments (Ex : **git add .**)

9.3 Valider les changements (git commit)

Après avoir ajouter nos modifications en mémoire, il s'agit maintenant de les valider en les capturant dans un instantané.

Pour valider ces modifications, tapez la commande :

```
➢ git commit -m "mon 1er commit"
```

Cette commande possède un paramètre **-m** pour le message et entre guillemets le texte du message.

Le message du commit permet de donner des indications sur les modifications apportées, il doit toujours être relativement précis.

Ce commit créé un instantané de notre repository.

C'est grâce à cette commande que l'on pourra revenir en arrière en cas de besoin, afin de retrouver notre repository dans un certain état bien précis.

Ce commit est identifiable par un code qui lui est propre.

Ce dernier conserve un historique des différents états de notre repository et peut donc être restauré en cas de besoin.

9.4 Soumettre les modifications (git push)

Après avoir ajouté et commité les modifications, il s'agit maintenant de les soumettre à notre repository distant.

En effet, toutes les actions que l'on a faites juste à maintenant n'étaient que locales.

Pour appliquer ces modifications en les envoyant à notre repository distant, tapez la commande :

```
➢ git push
```

La commande **git push** va envoyer les modifications sur notre repository distant.

Après avoir finalisé le processus, vous pouvez constater sur GITHUB que le fichier **bonjour.txt** est bien présent.

Pour résumer :

- 1. Nous avons cloné le repository sur notre PC avec la commande :
  git clone https://github.com/lawrenceterpin/formation-git.git
- 2. Nous avons modifié notre repository local, en ajoutant un fichier **bonjour.txt**
- 3. Nous avons visualisé ces modifications, avec la commande :
  git status
- 4. Nous avons ajouté ces modifications en mémoire, avec la commande :
  git add .
- 5. Nous avons capturé un instantané de ces modifications, avec la commande :
  git commit -m "mon 1er commit"
- 6. Nous avons envoyé ces modifications sur notre repository distant, avec la commande :
  git push

Nous avons vu ici le process GIT habituel à reproduire à chaque fois que l'on veut mettre à jour notre repository distant :

1. **git status**
2. **git add .**
3. **git commit -m "message du commit"**
4. **git push**

10. Les branches

Par-défaut, la seule branche existante est Master.

Les branches GIT sont un moyen de rassembler une tâche ou une de travailler sur une fonctionnalité spécifique par-exemple.

De plus, cela est une bonne pratique car cela évite de travailler directement dans la branche Master.

Créer une branche, c'est en quelque sorte comme créer une "copie" de votre projet pour développer et tester de nouvelles fonctionnalités sans impacter le projet de base.

Dans un premier temps, nous allons créer une branche héritée de la branche Master.

Dans votre terminal, pour créer une branche, tapez cette commande :

- **git checkout -b ma-premiere-branche**

Après avoir créer la branche, nous serons redirigés directement dans celle-ci.

Nous pouvons désormais travailler dans cette branche sans toucher à la branche Master.

Modifiez le fichier **bonjour.txt** en écrivant dedans : bonjour ma 1ère branche.

Ensuite, validez ces modifications avec le processus vu précédemment :

- 1.
- **git status**
- 2.
- **git add .**
- 3.
- **git commit -m "1er commit de ma 1ère branche"**
- 4.
- **git push**

Le fait de pousser la branche va la créer automatiquement sur le repository distant.

Vous pouvez constater sur votre repository Github que la branche est bien présente.

11. Synchroniser les branches locales avec les branches distantes

Avant toute modification sur votre branche, il est toujours nécessaire de la synchroniser avec le repository distant. **Cette démarche doit être systématique avant de travailler sur sa branche.**

11.1 Synchroniser la branche master

D'abord, pour fetcher la branche master, il faut s'y rendre, en tapant la commande :

- **git checkout master**

Sur la branche master, il s'agit maintenant de fetcher ses modifications, en tapant la commande :

- **git fetch**

Le fait de fetcher va comparer notre version locale avec la version distante de la branche courante.

Après avoir fetcher les modifications, nous avons besoin de les récupérer sur notre version locale.

Pour récupérer ces modifications, tapez la commande :

- **git pull**

11.2 Synchroniser une branche spécifique

Sur notre branche (par-exemple "ma-premiere-branche"), il s'agit de la synchroniser avec la branche master.

Pour ce faire, **après avoir au préalable récupérer la version distante de la branche master** (voir 11.1), il faut maintenant merger la branche master sur notre branche "ma-premiere-branche".

Pour merger la branche master sur notre branche, tapez la commande :

- **git merge master**

Désormais, toutes les modifications de la branche Master ont été fusionnées sur notre branche spécifique.

S'il n'y a pas de conflits, il ne reste plus qu'à pousser cette fusion sur notre branche distante, en tapant la commande :

- **git push**

Maintenant que notre branche est à jour avec le repository distant, on peut commencer à travailler.

Pour résumer :

- 1. Avant de travailler sur sa propre branche, toujours se rendre sur master afin de récupérer la dernière version du repository distant :

- **git checkout master**
- **git fetch**
- **git pull**

- 2. Retourner sur sa branche afin de merge master :

- **git checkout ma-premiere-branche**
- **git merge master**
- **git push**

12. Les merge (ou fusion)

Le fait de merger veut dire fusionner. Il s'agit de fusionner les modifications d'une certaine branche dans une autre.

Après avoir poussé nos modifications sur notre première branche, on peut les merger sur la branche Master.

Pour merger notre branche **ma-premiere-branche** dans la branche master, il faut tout d'abord se rendre sur la branche master.

Dans votre terminal, pour aller sur la branche master, tapez la commande :

- **git checkout master**

Désormais sur la branche master, nous devons tout d'abord mettre à jour notre repository local.

Pour récupérer la dernière version de la branche master, tapez la commande :

- **git fetch**

La commande git fetch va comparer les dernières modifications de la branche du repository distant sur notre branche du repository local.

Ensuite, il s'agit de récupérer ces modifications, avec la commande :

- **git pull**

Les modifications seront ainsi téléchargées vers notre version locale.

La branche de notre repository local à jour, pouvons maintenant merger la branche **ma-premiere-branche.**

Pour merger notre branche dans master, tapez la commande :

- **git merge ma-premiere-branche**

Ensuite, si tout est bon et qu'il n'y a aucune erreur ou conflit, nous pouvons pousser notre merge sur la branche distante, avec la commande :

- **git push**

Désormais, notre branche **ma-premiere-branche** est fusionnée dans master. Nous pouvons le vérifier en voyant le fichier bonjour.txt avec le texte : bonjour ma 1ère branche.

13. La gestion des conflits

Lors de récupération de modifications d'une branche distante à locale (lors d'un git merge notamment) il peut y avoir des conflits.

C'est-à-dire que les mêmes lignes dans un certain fichier en particulier peuvent être en conflits. Si deux personnes ont modifié la même ligne, ou qu'une personne a supprimé ce fichier auparavant, alors il peut y avoir conflit et la commande en cours est stoppée.

Il s'agit alors de résoudre ce conflit.

Pour résoudre un conflit, il faut se rendre sur le fichier impacté et modifier soi-même les lignes en prenant soin de prendre en compte à la fois le travail validé par l'autre personne et le nôtre.

Après avoir résolu tous les conflits, vous pouvez ajouter vos modifications, avec la commande :

- **git add .**

Ensuite, on commit ces modifications, avec la commande :

- **git commit -m "résolution des conflits"**

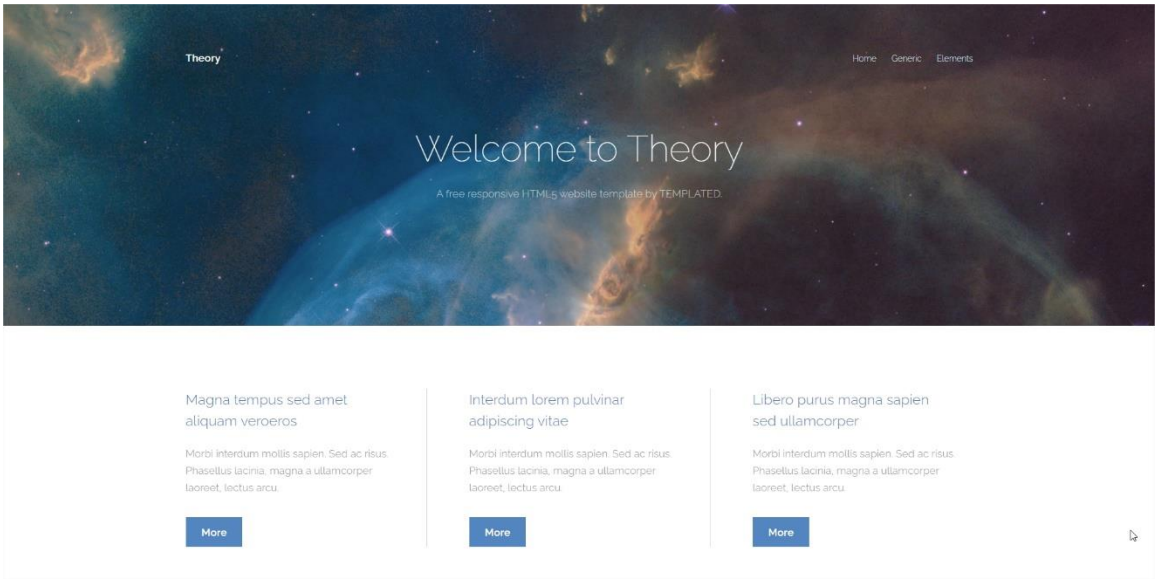
Enfin, on push ces modifications sur la branche distante, avec la commande :

- **git push**

14. Projet de formation à réaliser

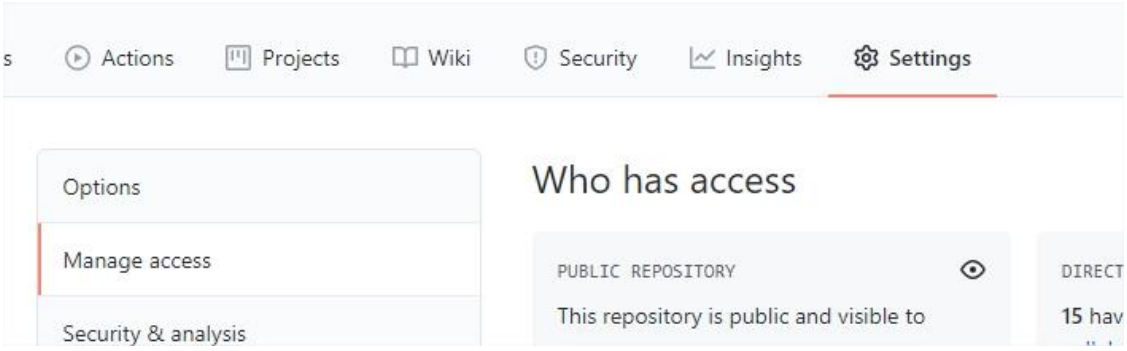
Pour cette formation, il s'agira de travailler en collaboration sur le même projet et sur le même repository.

Nous utiliserons le template **Theory by templated** fourni et présent sur la branche master.



Pour ce faire, chacun devra créer sa propre branche nommée par son prénom, à partir de master.

Il faudra donc ajouter chaque collaborateur au repository, dans l'onglet « **access** » :



Le but de l'exercice est :

- Dans un premier temps, de modifier le template par défaut, de travailler sur différentes zones en les attribuant à chaque collaborateur (ex : Léo sur le header, Alexandre sur le menu principal, …).
- Dans un second temps, d'ajouter sa propre vignette dans le trombinoscope afin de s'exercer sur les merge et la gestion de conflits.
- Communiquer entre collaborateurs pour gérer les conflits
- Respecter les process GIT vus au cours de la formation.