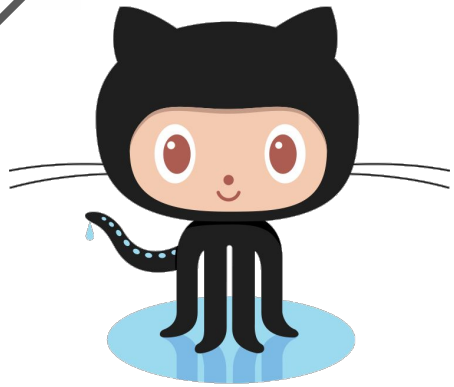
git



GitHub

COMPRENDRE GIT ET GITHUB

Qu’est-ce que Git ?

Git est un **logiciel de versionning**.

Grâce à lui, vous pouvez garder la trace de toutes les modifications faites sur votre code pour pouvoir vous y retrouver à tout moment. À chaque fois que vous faites une série de modifications (créer un fichier, supprimer un fichier, modifier un texte dans un fichier, etc.), vous allez pouvoir enregistrer ces modifications.

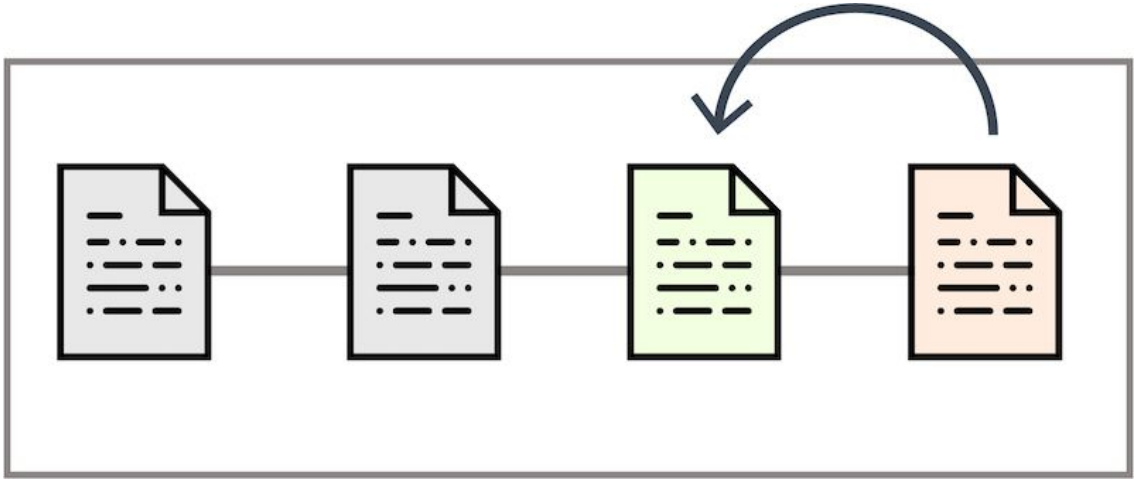
Qu’est-ce que GitHub ?

GitHub est un **service en ligne** qui permet d'héberger ses repositories de code. GitHub est un outil gratuit pour héberger du code open source, et propose également des plans payants pour des fonctionnalités supplémentaires. C'est le numéro 1 mondial et il héberge plus d'une dizaine de millions de repositories !
Il existe d’autres services en ligne de même type telle que GitLab ou BitBucket

Qu’est-ce que le versionning ?

Lorsque vous travaillez sur un projet de code, vous allez régulièrement y apporter des modifications, et par moments ces modifications vont provoquer des bugs. Lorsque vous revenez sur votre projet après quelques jours ou même quelques heures, il peut être difficile de vous souvenir des dernières modifications que vous avez effectuées et de retrouver vos repères dans votre code. vous allez pouvoir enregistrer ces modifs dans un commit.

Un **commit** correspond donc à une **version** de votre code à un instant t.
La somme de tous les commits constitue l'historique de votre projet. Et l'intérêt d'un logiciel de versionning comme Git, c'est que vous pouvez vous placer à n'importe quel endroit de cet historique.



Qu’est-ce qu’un dépôt local ?

Un **dépôt local** correspond aux versions de votre code sur votre ordinateur.

Qu’est-ce qu’un dépôt distant ?

Le **dépôt distant** est un peu différent. Il permet de stocker certaines versions qu'on lui aura envoyées, afin de garder un historique délocalisé (le code est stocké non pas sur votre ordinateur mais sur Github par exemple).

Imaginez que votre ordinateur brûle. Vous aurez toujours vos accès à votre code sur GitHub.

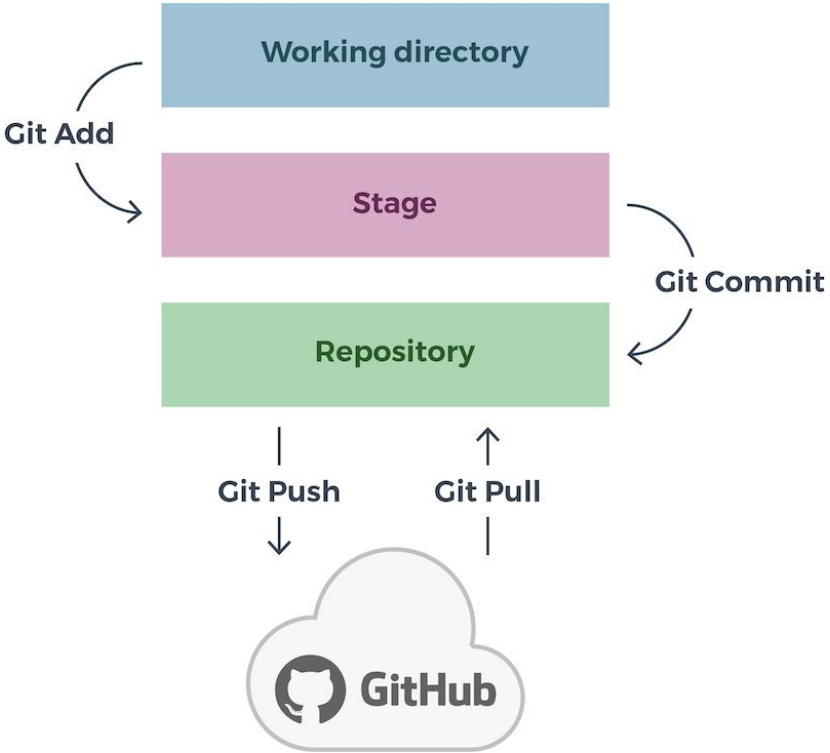
Mais en plus de les stocker, vous pouvez aussi les rendre publics, et chacun pourra alors venir y ajouter ses modifications et améliorations.

Git est l'outil qui nous permet de créer un **dépôt local** et de gérer les versions de nos fichiers

GitHub est un service en ligne qui va héberger notre dépôt, il sera du coup un **dépôt distant** (puisqu'il ne sera pas sur notre machine).

Git gère les versions de vos travaux locaux à travers 3 zones locales majeures :

- **le répertoire de travail** (working directory/WD) ;
- **l'index** (*stage)*;
- **le dépôt local** (Git directory/repository).



Télécharger Git

Mac-Linux : https://git-scm.com/downloads
Windows : https://gitforwindows.org/

Initialiser Git

La première chose à faire est de configurer son identité.

Nous allons commencer par renseigner votre nom et votre adresse e-mail. C'est une information importante car toutes les validations dans Git utilisent cette information et elle est indélébile dans toutes les validations que vous pourrez réaliser :

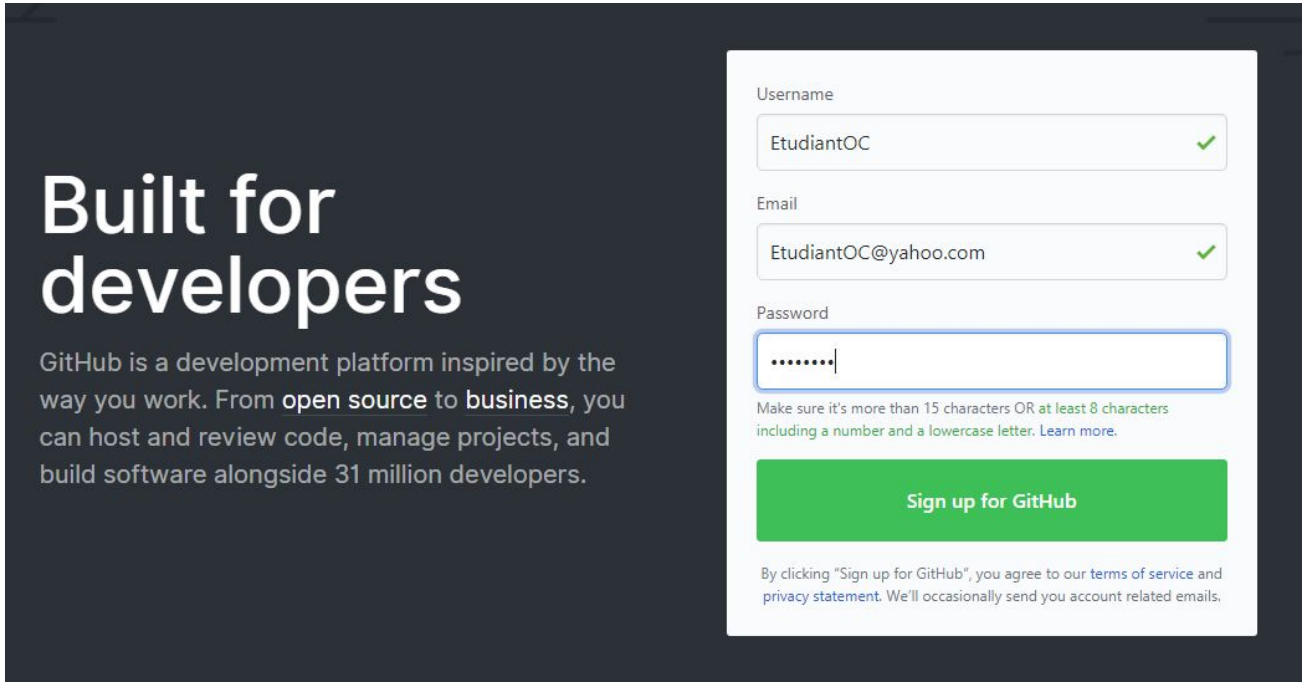
```
git config --global user.name "Votre nom ou pseudo"
git config --global user.email "votre@email.com"
```

Grâce à l'option --global, vous n'aurez besoin de le faire qu'une fois.

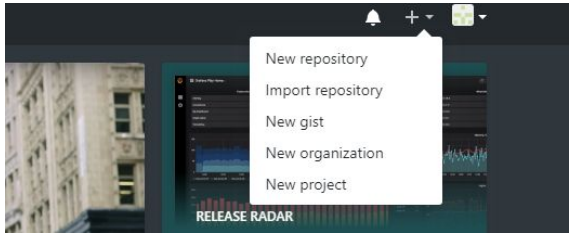
Pour vérifier que tout va bien, relancez votre console et tapez simplement 'git'. Si l'installation a fonctionné, vous devriez voir du texte en anglais expliquant l'utilisation de Git.
Afin de vérifier que vos paramètres aient bien été pris en compte, et vérifier les autres paramètres, il suffit de passer la commande **`git config --list`**

Démarrer un projet sur Github

1 - Créer un compte sur https://github.com/



2 - Créer un dépôt distant (un repository)



Owner * **Repository name** *

/ cours-git

Great repository names are short and memorable. Need inspiration? How about **probable-system**?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.

Choose a license
A license tells others what they can and can't do with your code. Learn more.

Create repository

GIT EN LIGNE DE COMMANDE

CODE 1

1. Créez un dossier: cours-git
2. Créez y un fichier index.html
3. Ajoutez y le code suivant:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <h1>Mon premier Repository !</h1>
</body>

</html>
```

Créer un nouveau repository sur la ligne de commande

Récupérer le lien de votre nouveau repository. Le lien utilisé dans cette démonstration sera: `https://github.com/mon-pseudo-github/cours-git.git`

Ouvrez votre terminal dans votre dossier `cours-git` préalablement créer

Exécutez les commandes suivantes :

```
git init
git add .
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mon-pseudo-github/cours-git.git
git push -u origin main
```

créé un fichier invisible .git

Rendez-vous sur github pour voir le repository distant à jour !

CODE 2

1. Modifier votre code par le code suivant:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <h1>Mon premier Repository !</h1>
  <p> Un nouveau texte ajouté ! </p>
</body>

</html>
```

Enregistrer une nouvelle version du code (Faire un commit ou commiter)

Exécutez les commandes suivantes :

```
git status
git add . ou git add index.html
git status
git commit -m "nouveau texte ajouté"
git status
git push origin main
```

Rendez-vous sur github pour voir le repository distant à jour !

Pour voir l'historique des commits : **`git log`**
Remarque: utiliser la touche Q pour sortir de l'historique

Pour supprimer le dernier commit: **`git reset --hard HEAD^`**
Remarque: Le HEAD^ indique que c'est bien le dernier commit que nous voulons supprimer.

Revenir à une ancienne sauvegarde (un ancien commit)

Exécutez les commandes suivantes :

```
git log
```

Je repère le commit que je souhaite et je récupère son numéro, dans l'exemple suivant il s'agit de `ca83a6dff817ec66f443420071545390a954664949`

```
$ git log
commit ca83a6dff817ec66f443420071545390a954664949
Author: Jean <jean-dupond@gmail.com>
Date: Mon Mar 19 21:52:11 2019 -0700
```

J'effectue la commande suivante

```
git reset --hard ca83a6dff817ec66f443420071545390a954664949
```

Je suis sur mon commit et donc sur l'ancienne version de mon code

ATTENTION: Si tu modifies et commites sur une ancienne sauvegarde, cela écrasera les sauvegardes plus récente !

Créer une branche dans un dépôt

Un élément que vous allez être souvent amenés à utiliser lorsque vous travaillez sur un repo, ce sont les branches. Les branches permettent de travailler sur des versions de code qui divergent de la branche principale contenant votre code courant.

Travailler sur plusieurs branches est très utile lorsque vous souhaitez tester une expérimentation sur votre projet, ou encore pour vous concentrer sur le développement d'une fonctionnalité spécifique.

Remarque: En règle général, on ne travaille jamais sur la branche master,
mais sur une branche DEV ou developpement

1- Il faut s'assurer d'être sur la branche master : **`git checkout master`**
2- Il faut s'assurer que la branche master soit à jour : **`git pull`**
3- Ensuite il faut créer ta nouvelle branche de fonctionnalité : **`git checkout -b ma-branche`**

Tu es désormais sur ta branche de fonctionnalité

Envoyer une nouvelle branche créée sur le dépôt distant

Vérifier que vous êtes sur la nouvelle branche avec :

```
git branch
```

```
* ma-branche
  master
```

Créer un fichier text.html avec le contenu suivant :

```
<p>Contenu ajouté depuis une nouvelle branche</p>
```

Faites un commit pour enregistrer vos modifications

```
git status
git add .
git commit -m "ma nouvelle branche"
```

Envoyer votre branche sur le repo distant

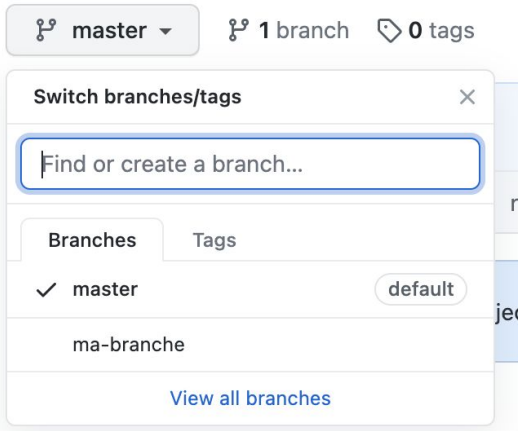
```
git push origin ma-branche
```

Retourner sur la branche master :

```
git checkout master
```

```
  ma-branche
* master
```

Votre branche apparaît à présent sur GitHub !



Fusionner 2 branches

Lorsque vous travaillez sur plusieurs branches, il va souvent vous arriver de vouloir ajouter dans une branche A les mises à jour que vous avez faites dans une autre branche B. Pour cela, on se place dans la branche A :

```
git checkout brancheA
```

Puis on utilise la commande **git merge** :

```
git merge brancheB
```

> Fusionner des branches est une pratique courante lorsque vous travaillez sur un projet : vous devez toujours chercher à remettre les modifications faites sur vos différentes branches dans la branche principale **master**.

Dans votre projet, placer vous sur ma branche master: **`git checkout master`**

Puis fusionner la avec votre branche ma-branche: **`git merge ma-branche`**

La branche master a récupéré ce qu'il y avait des la branche ma-branche

Ignorer des fichiers

Pour des raisons de sécurité et de clarté, il est important d'ignorer certains fichiers dans Git, tels que :

- Tous les fichiers de configuration (config.xml, databases.yml, .env...)
- Les fichiers et dossiers temporaires (tmp, temp/...)
- Les fichiers inutiles comme ceux créés par votre IDE ou votre OS (.DS_Store, .project...)

Le plus crucial est de ne **JAMAIS versionner une variable de configuration**, que ce soit un mot de passe, une clé secrète ou quoi que ce soit de ce type. Versionner une telle variable conduirait à une large faille de sécurité, surtout si vous mettez votre code en ligne sur GitHub !

Si le code a déjà été envoyé sur GitHub, partez du principe que quelqu'un a pu voir vos données de configuration et mettez-les à jour (changez votre mot de passe ou bien générez une nouvelle clé secrète).

Créez le fichier que vous nommerez **.gitignore** pour y lister les fichiers que vous ne voulez pas versionner dans Git. Listez ces fichiers ligne par ligne en indiquant leurs chemins complets, par exemple :

```
.DS_Store
config/application.yml
motsdepasse.txt
```

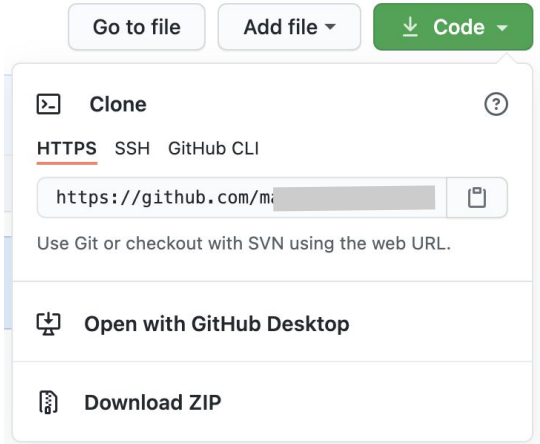
GITHUB ET TRAVAIL DE GROUPE

Cloner un dépôt existant

Nous allons à présent travailler sur le même repository !

Créer un dossier cours-git-2
Place toi dans ce dossier dans ton terminal

A présent, récupère le repository du compte d'un autre développeur en cliquant sur le bouton vert CODE, puis copier le lien HTTPS



Dans ton terminal, écris :

```
git clone https://github/lien-que-tu-as-recupere
```

Récupérez des modifications et mettre à jour son dépôt local avec les nouveautés du dépôt distant

Pour récupérer en local les dernières modifications du repo GitHub, il vous faut utiliser la commande git pull :

```
git pull origin master
```

Pensez à synchroniser régulièrement votre code local avec vos repos en ligne à l'aide des commandes git push et pull. C'est particulièrement important lorsque vous travaillez à plusieurs sur un projet, pour que tout le monde avance sur la même base !

Gérer les conflits

Il arrive très souvent qu'il y aie des conflits entre les deux branches qui empêchent de les fusionner, par exemple lorsque plusieurs personnes travaillent en même temps sur un même fichier.

Exemple :

- La branche ***master*** contient un fichier **index.html** avec à la **ligne 11** ce code : `<h1>Un titre </h1>`.
- La branche ***ma-branche*** contient un fichier **index.html** avec à la **ligne 11** ce code-ci : `<h1>Un autre titre </h1>`.

Si vous tentez de fusionner la branche ma-branche dans la branche master :

```
git checkout master
```

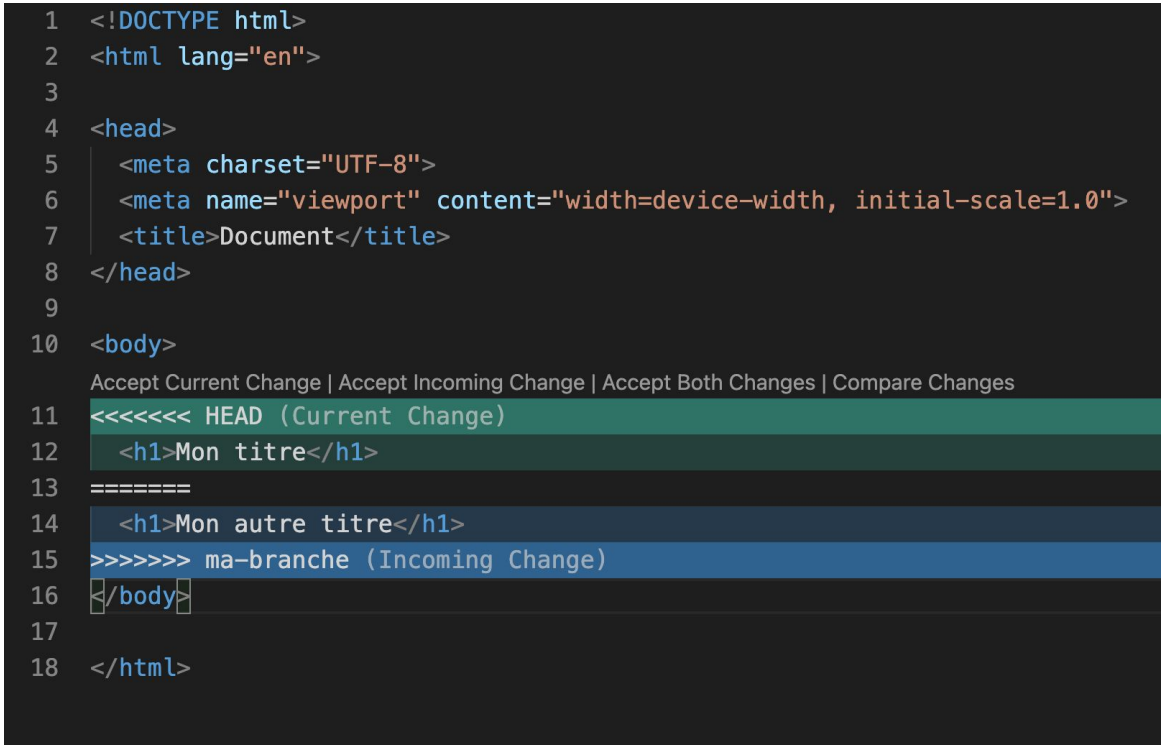
```
git merge ma-branche
```

Git va reconnaître qu'il existe un conflit entre les deux branches car la ligne 11 du fichier index.html est différente dans chacune des branches et afficher le message suivant :

```
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Vous allez donc devoir ouvrir le fichier index.html dans votre éditeur de texte.

Vous allez alors voir les différences de contenu du fichier index.html entre les deux branches, et vous pouvez choisir quel contenu garder pour la branche master dans laquelle vous faites le merge.



Accept Current Change
(Accepter le changement actuel -> celui de master)

Accept Incoming Change
(Accepter le changement entrant -> celui de ma-branche)

Accept Both Changes
(Accepter les 2 changements)

Compare Changes
(Comparer les changements)

Maintenant que vous avez résolu le conflit, il vous reste à le dire à Git !

Pour cela, faites un commit sans message : **`git commit`**

OUTIL FACULTATIF _ GITHUB Desktop

https://desktop.github.com/