

# UTILISATION DE GIT ET GITHUB

**Git** est un logiciel qui permet de gérer les différentes versions d'un projet. Il comporte des fonctionnalités intéressantes comme par exemple :

- La mémorisation des différentes versions et la possibilité de revenir à une version antérieure
- La possibilité de travailler sur plusieurs branches en parallèle puis de faire une synthèse à un moment donné du travail réalisé sur chacune des branches.
- La possibilité d'analyser précisément les différences entre les différentes versions
- ...

**GitHub** est un service Web d'hébergement en ligne de vos dépôts Git. Il va faciliter le travail communautaire sur le même projet, le partage de vos ressources avec des collaborateurs, etc.

Ce cours va vous faire découvrir les fonctionnalités de Git et GitHub à travers un exemple concret.

## Table des matières

1	Installation de Git .....	2
2	Création d'un espace de travail pour notre projet sur notre ordinateur personnel.....	2
3	Ajout d'un fichier dans notre répertoire .....	3
4	Modification d'un fichier .....	3
5	Réparer des erreurs.....	4
5.1	Revenir à la dernière version que l'on a commitée.....	4
5.2	Revenir à une version quelconque et perdre l'historique.....	4
5.3	Réparer une erreur commitée.....	5
5.4	Récupérer un fichier d'une vieille version sans perdre l'historique.....	5
6	Notion de branche.....	6
6.1	On souhaite tester la fonction 1.....	7
6.2	On souhaite tester la fonction 2.....	8
6.3	On valide les tests réalisés sur les 2 fonctions et on veut ramener tout le travail dans la branche principale.....	9
6.4	Supprimer les branches que vous n'utilisez plus.....	9
7	Travailler à plusieurs sur le même projet.....	10
7.1	Création d'un compte sur Github.....	10
7.2	Création d'une repository .....	10
7.3	Création d'un "token" d'identification .....	12
7.4	Associer ce dépôt distant à notre dépôt local.....	12
7.5	Copier tout notre projet local vers le projet distant .....	13
7.6	Partager mon travail avec des collaborateurs.....	13
7.7	Echanges entre collaborateurs.....	14
8	Cloner un projet déposé en public sur GitHub.....	15
9	Principales commandes Git : pensez à utiliser <code>git commande --help</code> pour avoir des infos sur une commande précise.....	16
10	Références.....	18

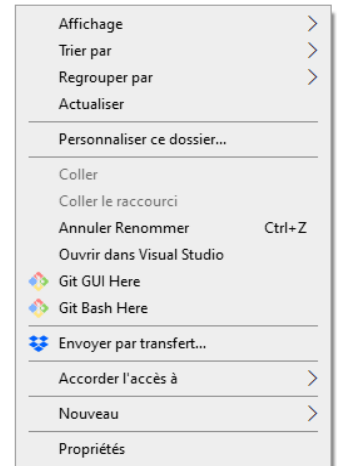
# 1 Installation de Git

Installer Git sur votre ordinateur en allant sur le site officiel <https://git-scm.com/>



## 2 Création d'un espace de travail pour notre projet sur notre ordinateur personnel

- Avec l'explorateur de fichiers, créer un répertoire qui va contenir votre projet sur votre ordinateur ou sur votre disque externe. Déplacez vous avec l'explorateur de fichiers dans ce répertoire puis faire un clic droit et choisir "Git Bash here". Une fenêtre console s'ouvre dans laquelle nous allons pouvoir écrire nos commandes Git.
- Ecrivez `git --version` dans cette fenêtre console pour vérifier la bonne installation de Git



git --version

```
MINGW64:/u/tmp/Prog/MesProjets_SNIR/Projet1
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1
$ git --version
git version 2.29.2.windows.3
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1
$ |
```

Vérifiez que le dossier est bien celui dans lequel vous allez placer votre projet

- Définir son nom et son email. Ecrivez les lignes suivantes (en remplaçant mes identifiants par les vôtres bien sûr !)

git config

```
$ git config --global user.name "Vincent ROBERT"
$ git config --global user.email "vincent.robert@cdfnancy.fr"
```

- Initialiser git

git init

```
$ git init
Initialized empty Git repository in U:/tmp/Prog/MesProjets_SNIR/Projet1/.git/
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (master)
$
```

- Vérifiez l'état de git

git status

```
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

### 3 Ajout d'un fichier dans notre répertoire

- Déplacez dans votre répertoire le fichier `squelette.cpp` que vous devriez avoir à disposition. Renommez le `exemple1.cpp`

```

/*****
*****
*****/
#include <iostream> // bibliothèque de gestion des E/S
#include <conio.h> // gestion de la console (ici _getch())
#include <windows.h>

using namespace std;

/*****
Fonction principale
*****/
int main() // Fonction principale
{
    SetConsoleOutputCP(1252); // pour les accents (il faut <windows.h>)

    _getch(); // attente d'appui sur une touche
    return 0; // fin du programme. Le code 0 est envoyé
}

```

- On choisit le(s) fichiers à ajouter à notre projet. Ici nous n'avons qu'un fichier, donc, nous allons écrire

git add

```
$ git add exemple1.cpp
```

Si on voulait ajouter tout, on écrirait `git add *`

- Si on veut prendre "une photo" de notre projet à cet instant, on va réaliser un "**commit**" en utilisant la commande

git commit

```
$ git commit -m "programme de départ"
[master (root-commit) 65af71e] programme de départ
1 file changed, 23 insertions(+)
create mode 100644 exemple1.cpp
```

Ici, c'est du blabla qui caractérise la version que vous "commitez".

- Les fois suivantes, si on veut traiter l'ajout et le commit des mêmes fichiers, il suffira d'écrire `git commit -a -m "nouvelle version"` pour faire à la fois la commande `add` et la commande `commit`

### 4 Modification d'un fichier

Maintenant, modifiez `exemple1.cpp` en ajoutant l'affichage de "**Bonjour le monde**"

On va prendre une nouvelle "photo" de notre projet.

```
$ git commit -a -m "version polie"
[master d345012] version polie
1 file changed, 2 insertions(+), 1 deletion(-)
```

Vérifions qu'on a bien les deux versions mémorisées avec la commande `git log`

git log

```
$ git log --oneline
617d678 (HEAD -> master) version polie
eb5ede6 (brancheDeDepart) Programme de départ
```

Ajoutons une troisième modification dans le fichier.

```
...
int main() // Fonction principale
{
    SetConsoleOutputCP(1252); // pour les accents (il faut <windows.h>
    int x,
    cout <<"Bonjour le monde"<<endl;

    _getch(); // attente d'appui sur une touche
    return 0; // fin du programme. Le code 0 est envoyé
}
```

Ajout qui conduira à une erreur de compilation car j'ai écrit une virgule à la place d'un point-virgule.

On peut déjà observer les modifications entre les fichiers sélectionnés (ceux choisis avec add) commités et leur état actuel.

git diff

```
$ git diff
diff --git a/exemple1.cpp b/exemple1.cpp
index 7465ed0..530947f 100644
--- a/exemple1.cpp
+++ b/exemple1.cpp
@@ -13,6 +13,7 @@ Fonction principale
 int main() // Fonction principale
 {
     SetConsoleOutputCP(1252); // pour les accents (il faut <windows.h>
+    int x,
     cout <<"Bonjour le monde"<<endl;
     _getch(); // attente d'appui sur une touche
     return 0; // fin du programme. Le code 0 est envoyé
}
```

## 5 Réparer des erreurs

### 5.1 Revenir à la dernière version que l'on a committée.

git reset

```
$ git reset --hard HEAD
HEAD is now at d345012 Version polie
```

les dernières  
modifications  
sont perdues

### 5.2 Revenir à une version quelconque et perdre l'historique

**Attention, dans ce cas, vous allez perdre toutes les modifications qui ont suivi la version vers laquelle vous voulez revenir.**

git reset

```
$ git reset --hard idDuCommitVersLequelvousRevenez

$ git log --oneline
617d678 (HEAD -> master) version polie
eb5ede6 (brancheDeDepart) Programme de départ

$ git reset --hard eb5ede6
HEAD is now at eb5ede6 Programme de départ

$ git log --oneline
eb5ede6 (HEAD -> master, brancheDeDepart) Programme de départ
```

Une partie de  
l'historique  
est perdue.

### 5.3 Réparer une erreur committée

Réécrire la ligne `int x,` dans votre programme, puis prenons une nouvelle photo de notre projet

```
$ git add exemple1.cpp
vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (master)
$ git commit -m "x est déclaré comme un entier"
[master 6068f57] x est déclaré comme un entier
1 file changed, 2 insertions(+), 4 deletions(-)
```


Si vous avez déjà "committé" ce que vous n'auriez pas dû, la manière la plus correcte de faire ça est d'utiliser `git revert HEAD` qui revient à la version précédente et fait un commit

git revert

```
$ git revert HEAD
[master 213d7f4] Revert "x est déclaré comme un entier"
1 file changed, 1 insertion(+), 1 deletion(-)
```

Vous pouvez aussi revenir sur des changements plus anciens, par exemple, sur l'avant-dernier changement, mais des conflits peuvent apparaître, donc c'est à éviter !

```
$ git revert HEAD^
```

Le nombre de  indique le nombre de fois que vous voulez retourner en arrière.

### 5.4 Récupérer un fichier d'une vieille version sans perdre l'historique

Avec la commande `git log --oneline`, identifiez les différentes versions que vous avez réalisées, puis utilisez la commande `git checkout idDuCommit nomDuFichierARecuperer` pour revenir à la version que vous souhaitez d'un fichier donné.

Dans mon cas, je veux revenir à la version initiale du fichier exemple1.cpp. Je repère d'abord son identifiant avec `git log --oneline`

```
$ git log --oneline
f133efd Revert "x est déclaré comme un entier"
6068f57 x est déclaré comme un entier
d345012 version polie
65af71e programme de départ
```

Je vois que l'ID du programme de départ est 65af71e

Pour revenir à la version de départ, j'écris donc :

```
$ git checkout 65af71e exemple1.cpp
Updated 1 path from 02ed845
```

Vous pouvez contrôler que on est bien revenu au programme de base.

Si finalement, vous voulez revenir à la version polie, écrivez

```
$ git checkout d345012 exemple1.cpp
Updated 1 path from ab04441
```

Dans le cas d'un projet comportant de multiples fichiers, si vous écrivez simplement `git checkout id`, donc **si vous n'indiquez pas le fichier**, git vous crée une branche parallèle contenant la version sur laquelle vous revenez afin de pouvoir l'observer. Vous allez pouvoir **visualiser** tous les fichiers de cette version, mais vous ne serez qu'observateur (comme si vous pouviez observer le passé). Si vous voulez vraiment repartir de cette version, il faudra créer une nouvelle branche avec cette version.

## 6 Notion de branche

Supposons que vous avez 2 fonctions à réaliser pour ce programme **exemple1.cpp**. Vous voulez d'abord faire des petits programmes de tests pour tester individuellement ces 2 fonctions, puis quand tout fonctionnera, vous rassemblez tout.

Actuellement, nous n'avons qu'une branche, la branche "**master**".

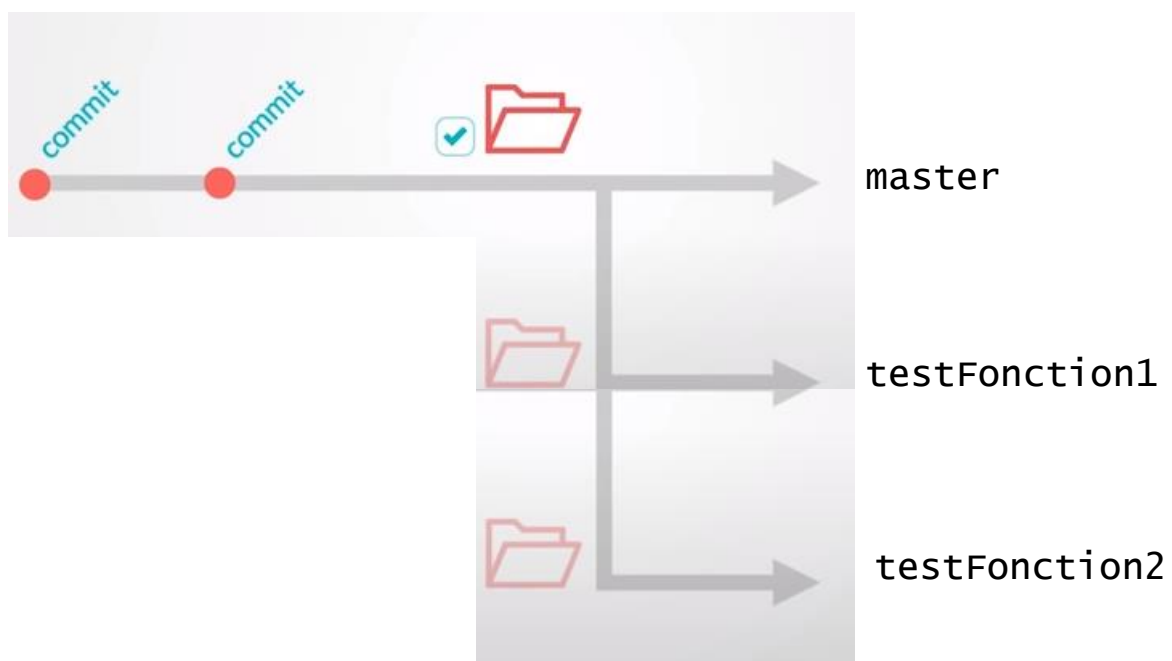
git branch

```
$ git branch
* master
```

Cette commande liste toutes les branches. La branche courante est marquée d'un **astérisque**.

Nous allons créer 2 branches pour chacune de nos fonctions.

```
$ git branch testFonction1
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (master)
$ git branch testFonction2
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (master)
$ git branch
* master
  testFonction1
  testFonction2
```



## 6.1 On souhaite tester la fonction 1

On sélectionne la branche **testFonction1**

git checkout

```
$ git checkout testFonction1
Switched to branch 'testFonction1'

vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (testFonction1)
$ git branch
  master
* testFonction1
  testFonction2
```

Je vois que la branche courante est maintenant testFonction1

Ajouter un fichier **fct1.cpp** contenant le code suivant

```
/******
Test de la fonction 1
*****/
#include <iostream>    // bibliothèque de gestion des E/S
#include <conio.h>      // gestion de la console (ici _getch())
#include <windows.h>

using namespace std;

/******
Fonction 1
*****/
void fonction1()
{
    //... ici se trouvera le code de la fonction 1 que je veux tester
}

/******
Fonction principale
*****/
int main()           // Fonction principale
{
    SetConsoleOutputCP(1252);

    fonction1(); // appel de la fonction1 pour vérifier son fonctionnement

    _getch();      // attente d'appui sur une touche
    return 0;      // fin du programme. Le code 0 est envoyé
}
```

Je vois ici la branche courante

On ajoute ce fichier à la nouvelle version que l'on veut créer et on fait un commit

```
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (testFonction1)
$ git add fct1.cpp

vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (testFonction1)
$ git commit -m "test de la fonction 1"
[testFonction1 d6ec890] test de la fonction 1
1 file changed, 29 insertions(+)
create mode 100644 fct1.cpp
```

## 6.2 On souhaite tester la fonction 2

On sélectionne la branche **testFonction2**

```
$ git checkout testFonction2
Switched to branch 'testFonction2'

Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (testFonction2)
$ git branch
  master
* testFonction1
* testFonction2
```

Je vois que la branche courante est maintenant testFonction2

Ajouter un fichier **fct2.cpp** contenant le code suivant

```
/******
Test de la fonction 2
*****/
#include <iostream>    // bibliothèque de gestion des E/S
#include <conio.h>     // gestion de la console (ici _getch())
#include <windows.h>

using namespace std;

/******
Fonction 2
*****/
int fonction2()
{
    //... ici se trouvera le code de la fonction 1 que je veux tester
}

/******
Fonction principale
*****/
int main()           // Fonction principale
{
    SetConsoleOutputCP(1252);

    cout << fonction2(); // appel de la fonction2 pour vérifier son fonctionnement

    _getch();         // attente d'appui sur une touche
    return 0;         // fin du programme. Le code 0 est envoyé
}
```

Je vois ici la branche courante

```
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (testFonction2)
$ git add fct2.cpp

Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (testFonction2)
$ git commit -m "test de la fonction 2"
[testFonction2 85fe570] test de la fonction 2
1 file changed, 29 insertions(+)
create mode 100644 fct2.cpp
```

***Vous pouvez remarquer que quand vous êtes sur la branche `testFonction2`, vous ne voyez pas les fichiers de la branche `testFonction1`***



### 6.3 On valide les tests réalisés sur les 2 fonctions et on veut ramener tout le travail dans la branche principale.

- On bascule sur la branche principale

```
$ git checkout master
Switched to branch 'master'
```

- On rapatrie le travail de la branche **testFonction1** et de la branche **testFonction2**

git merge

```
$ git merge testFonction1
Updating 803e5a0..d6ec890
Fast-forward
 fct1.cpp | 29 +++++++++++++++++++++++++++++++++++++
 1 file changed, 29 insertions(+)
 create mode 100644 fct1.cpp

Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (master)
$ git merge testFonction2
Merge made by the 'recursive' strategy.
 fct2.cpp | 29 +++++++++++++++++++++++++++++++++++++
 1 file changed, 29 insertions(+)
 create mode 100644 fct2.cpp
```

### 6.4 Supprimer les branches que vous n'utilisez plus

Bien sûr, vous ne devez pas supprimer la branche sur laquelle vous êtes, mais cela peut être intéressant de faire de temps en temps du nettoyage pour enlever les branches que vous êtes sûr de ne plus avoir besoin.

```
$ git branch -d testFonction1
Deleted branch testFonction1 (was d6ec890).

Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (master)
$ git branch -d testFonction2
Deleted branch testFonction2 (was 85fe570).
```

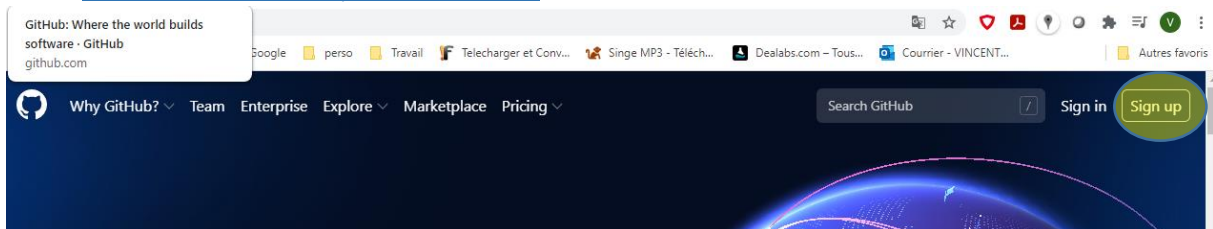
## 7 Travailler à plusieurs sur le même projet

Pour travailler en collaboration sur un même projet, il faut utiliser un dépôt distant. Je vous conseille d'utiliser **GitHub**.

Votre travail pourra être déposé sur le cloud et récupéré par d'autres personnes.



### 7.1 Création d'un compte sur Github



### 7.2 Création d'une repository

- Se connecter sur le compte que vous venez de créer
- Créer une "repository" pour y placer votre projet. Appelez cette "repository" : **TestGitEtGitHub**

The screenshot shows the 'Create a new repository' form on GitHub. The form includes fields for 'Owner' (VincentRobert54) and 'Repository name' (testGitEtGitHub). There is a description field with the text 'Découverte de l'utilisation de Git et GitHub'. The form also has options for 'Public' and 'Private' visibility, and checkboxes for 'Add a README file', 'Add .gitignore', and 'Choose a license'. A green 'Create repository' button is at the bottom.

Create your first project  
Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

Create repository

Import repository

- En mode **public**, tout le monde sur Internet pourra voir votre projet. Vous pourrez choisir les personnes qui pourront y faire des modifications.
- En mode **privé**, vous pourrez choisir qui pourra voir et modifier votre projet.


<sup>1</sup> La version gratuite de GitHub limitée à 500 Mo nous convient bien pour nos projets.

<https://github.com/pricing>

Une fois votre repository créée, GitHub vous donne un lien qu'il vous faut mémoriser.

The screenshot shows the GitHub interface for a repository named 'testGitEtGithub' by user 'VincentRobert54'. The repository is marked as 'Private'. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. The main content area provides instructions for setting up the repository. It starts with a 'Quick setup' section for users who have done this before, offering options to 'Set up in Desktop', 'HTTPS', or 'SSH', with the repository URL 'https://github.com/VincentRobert54/testGitEtGithub.git'. Below this, it provides a list of commands to create a new repository or push an existing one from the command line. The final section, '...or import code from another repository', mentions that code from Subversion, Mercurial, or TFS can be imported, with an 'Import code' button.

**Quick setup — if you've done this kind of thing before**

 Set up in Desktop or **HTTPS** **SSH** <https://github.com/VincentRobert54/testGitEtGithub.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

**...or create a new repository on the command line**

```
echo "# testGitEtGithub" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/VincentRobert54/testGitEtGithub.git
git push -u origin main
```

**...or push an existing repository from the command line**

```
git remote add origin https://github.com/VincentRobert54/testGitEtGithub.git
git branch -M main
git push -u origin main
```

**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

### 7.3 Création d'un "token" d'identification

Chaque collaborateur doit générer un "token" qui permettra de l'authentifier. Pour cela :

- cliquer en haut à droite et sélectionner **"Settings"**
- Dans la partie gauche, sélectionner **"Developer settings"**
- Choisir ensuite **"Personal access tokens"** puis **"Generate new token"**

Settings / Developer settings

Personal access tokens

Generate new token

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

New personal access token

Note

exampleToken

What's this token for?

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

☒ repo Full control of private repositories

☐ repository:status Access repository status

☐ repository:deployment Access deployment status

☐ public\_repo Access public repositories

☐ repository\_invitation Access repository invitations

☐ security\_events Read and write security events

☒ delete\_repo Delete repositories

☐ write:discussion Read and write team discussions

☐ read:discussion Read team discussions

☐ admin:enterprise Full control of enterprises

☐ manage\_billing:enterprise Read and write enterprise billing data

☐ read:enterprise Read enterprise profile data

☐ admin:gpg\_key Full control of public user gpg keys (Developer Preview)

☐ write:gpg\_key Write public user gpg keys

☐ read:gpg\_key Read public user gpg keys

Generate token Cancel

- Mémorisez l'identifiant du **"token"** créé

Settings / Developer settings

Personal access tokens

Generate new token Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ 091ber... b7c246 Delete

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### 7.4 Associer ce dépôt distant à notre dépôt local

```
git remote
```

```
$ git remote add origin
https://votreLoginGitHub:token@github.com/vincentRobert54/testGitEtGithub.git
```

Ajoutez votre identifiant et l'id du token

Si vous ne donnez pas de "token", mais juste [votreLoginGitHub@github.com/...](#) une page Web s'ouvrira quand vous voudrez déposer des fichiers sur GitHub et vous demandera de vous authentifier.

## 7.5 Copier tout notre projet local vers le projet distant

```
$ git branch -M main
```

Change le nom de la branche principale qui s'appelle **main** sous GitHub (et non **master**)

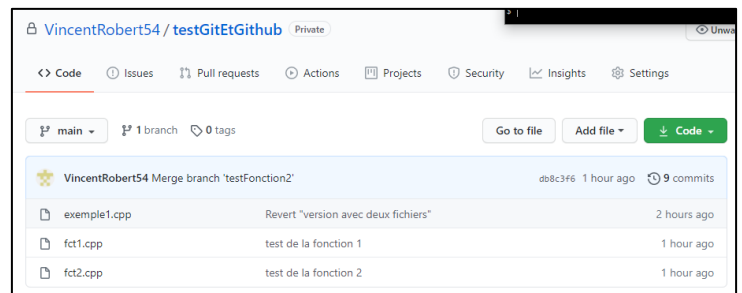
```
Vincent@MSI-VINCENT MINGW64 /u/tmp/Prog/MesProjets_SNIR/Projet1 (main)
```

```
$ git push -u origin main
```

```
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (23/23), 2.63 KiB | 673.00 KiB/s, done.
Total 23 (delta 10), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (10/10), done.
To https://github.com/vincentRobert54/testGitEtGithub.git
* [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

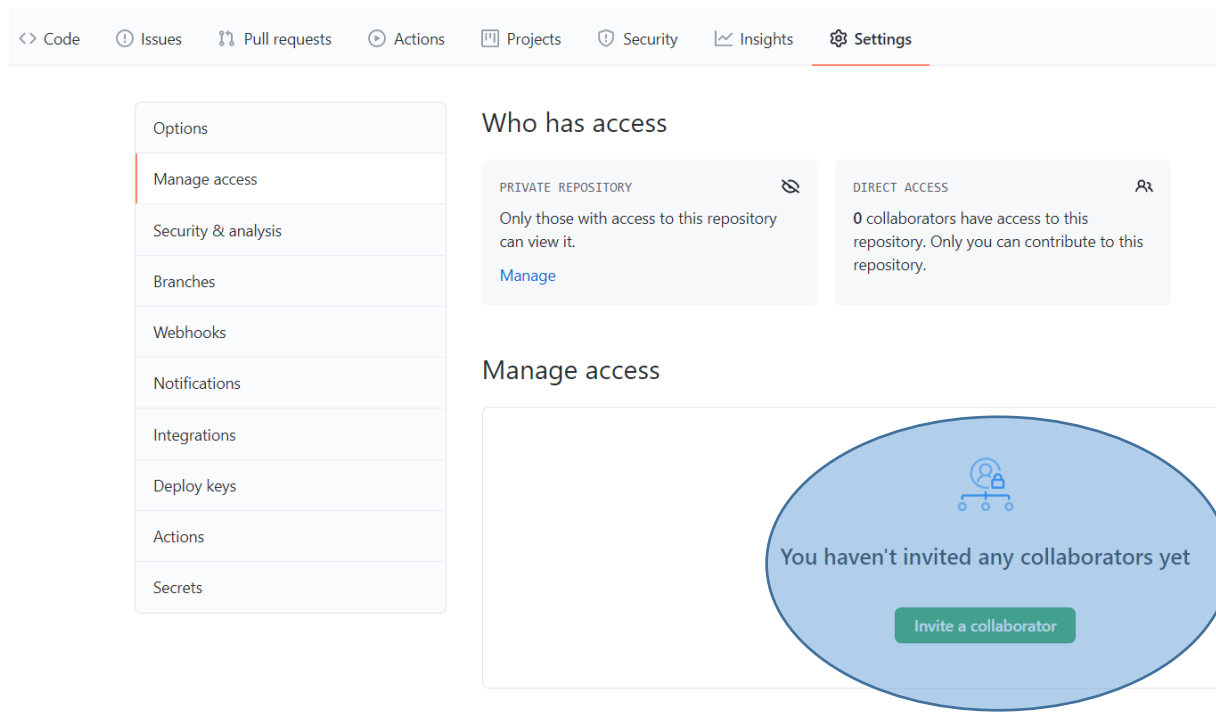
Copie la branche main du dépôt local vers le dépôt distant nommé origin

Vos fichiers apparaissent alors sur GitHub



## 7.6 Partager mon travail avec des collaborateurs

- Le chef de projet (celui qui vient précédemment de copier les fichiers sur GitHub) doit inviter des collaborateurs (qui doivent bien sûr avoir un compte GitHub).



- Les collaborateurs invités vont recevoir un email qui leur demande d'accepter ou non l'invitation.
- Si la repository est privée, les collaborateurs doivent créer un "token" (cf. [Création d'un "token" d'identification \(si la repository est privée\)](#))
- Les collaborateurs doivent se brancher sur le dépôt distant avec la commande

```
$ git remote add origin
https://leLoginGithubDuCollaborateur:token@github.com/VincentRobert54/testGitEtGithub.git
```

Chaque collaborateur ajoute son identifiant et l'identifiant du "token" créé précédemment

- Les collaborateurs peuvent ensuite récupérer les fichiers avec la commande **git pull<sup>2</sup>**

```
$ git pull origin main
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 23 (delta 10), reused 23 (delta 10), pack-reused 0
Unpacking objects: 100% (23/23), 2.61 KiB | 2.00 KiB/s, done.
From https://github.com/VincentRobert54/testGitEtGithub
* branch          main      -> FETCH_HEAD
* [new branch]    main      -> origin/main
```

Si l'erreur "**fatal : refusing to merge unrelated histories**" apparaît, cela signifie que les 2 dépôts ont des historiques différents et qu'il refuse de les fusionner. On peut alors forcer la fusion en complétant la commande avec l'option **--allow-unrelated-histories**

- En local, on peut ensuite renommer la branche **master** en **main**

```
$ git branch -M main
```

## 7.7 Echanges entre collaborateurs

### Quand un collaborateur modifie un ou plusieurs fichiers qu'il veut faire partager aux autres

```
$ git add --all
$ git commit -m "modif collaborateur1"
[main ad3105f] modif collaborateur1
1 file changed, 1 insertion(+), 1 deletion(-)

$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 362 bytes | 362.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/VincentRobert54/testGitEtGithub.git
db8c3f6..ad3105f main -> main
```

Choisissez éventuellement les fichiers

### Quand une autre personne veut récupérer le travail de ce collaborateur

```
$ git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), 342 bytes | 1024 bytes/s, done.
From https://github.com/VincentRobert54/testGitEtGithub
* branch          main      -> FETCH_HEAD
db8c3f6..ad3105f main      -> origin/main
Updating db8c3f6..ad3105f
Fast-forward
 fct1.cpp | 2 + -
1 file changed, 1 insertion(+), 1 deletion(-)
```

<sup>2</sup> Les commandes **git pull** et **git fetch** sont toutes les deux utilisées pour mettre à jour un répertoire de travail local avec les données d'un repository distant. Elles n'ont cependant pas le même fonctionnement.

La commande **git fetch** va récupérer toutes les données des commits effectués sur la branche courante qui n'existent pas encore dans votre version en local. Ces données seront stockées dans le répertoire de travail local mais ne seront pas fusionnées avec votre branche locale. Si vous souhaitez fusionner ces données pour que votre branche soit à jour, vous devez utiliser ensuite la commande **git merge**.

La commande **git pull** est en fait la commande qui regroupe les commandes **git fetch** suivie de **git merge**. Cette commande télécharge les données des commits qui n'ont pas encore été récupérées dans votre branche locale puis fusionne ensuite ces données.

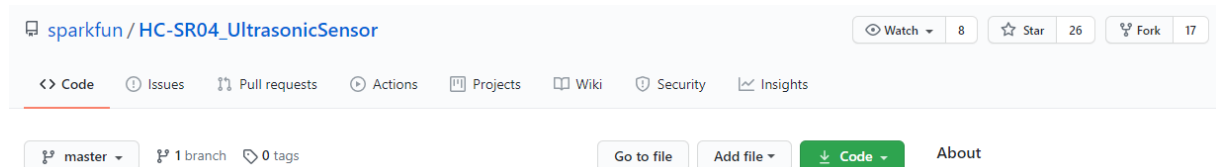
## 8 Cloner un projet déposé en public sur GitHub

Dans ce cas, il ne s'agit pas de travailler en collaboration, mais simplement de récupérer rapidement le travail de quelqu'un.

Prenons un exemple : vous voulez mettre en place un capteur ultrasonore HC-SR04 sur Arduino.

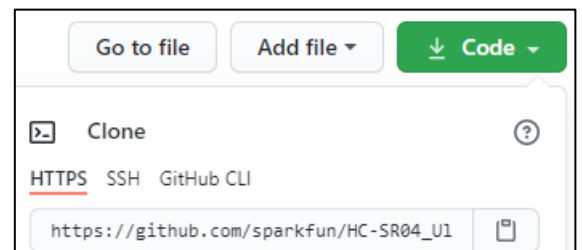
Cherchons dans notre moteur de recherche **arduino hc-sr04 github**

Une des pages me conduit vers la page de **sparkfun**



Le clonage est très simple :

- Déplacez-vous dans un dossier sur votre disque puis ouvrez la console Git (**Git bash**).
- Sur la page GitHub que vous avez choisie, cliquez sur Code. Un lien vous est fourni que vous devez copier dans le presse-papier.
- Dans votre console GitBash, écrivez



git clone

```
$ git clone https://github.com/sparkfun/HC-SR04\_UltrasonicSensor.git
```

⇒ Le dossier avec tous les codes associés est alors copié sur votre disque.

## 9 Principales commandes Git : pensez à utiliser `git commande --help` pour avoir des infos sur une commande précise

\$ git config	Configuration des préférences de l'utilisateur	git config --global user.email toto@titi.com	
		\$ git config --global user.name "nomUtilisateur"	
\$ git init	initialisation de Git. Cela crée un nouveau sous-répertoire nommé <b>.git</b> qui contient tous vos fichiers de référentiel nécessaires - un squelette de référentiel Git. À ce stade, rien dans votre projet n'est encore suivi.		
\$ git status	Vérification de l'état des fichiers. Vous indique quels changements ont eu lieu		
\$ git log	Affichage de l'historique des validations (commits)	\$ git log	Affiche l'historique des commits
		\$ git log -p	Donne les différences introduites par chaque commit
		\$ git log -p -2	N'affiche que les 2 dernières entrées
		\$ git log --oneline	Affichage court avec une ligne par commit
\$ git add	Ajoute les fichiers indiqués à l'index des fichiers suivis par Git	\$ git add --all	Ajoute tous les fichiers du répertoire courant
		\$ git add Documentation / \ *.txt	Ajoute le contenu de tous les fichiers <b>*.txt</b> sous le répertoire <b>Documentation</b> et ses sous-répertoires.
		git add toto*.cpp	Ajout de contenu à partir de tous les fichiers cpp commençant par toto.
\$ git commit	Enregistre les changements dans un dépôt	git commit -m "Nom qui caractérise le dépôt"	
		git commit -a -m "blabla"	Ajoute (add) et commit les fichiers qui sont mémorisés
\$ git diff	Lister les conflits	\$ git diff	Enumère les conflits en indiquant les différences trouvées
		\$ git diff <i>brancheSource brancheCible</i>	La commande suivante est utilisée pour afficher les conflits entre les branches à fusionner avant de les fusionner.
\$ git branch	Gestion des branches	\$ git branch	Liste les branches
		\$ git branch <i>nomNouvelleBranche</i>	Crée une nouvelle branche
		\$ git branch -d <i>nomDeLaBranche</i>	Efface une branche
\$ git checkout	Déplacement sur les branches ou restauration de fichiers	\$ git checkout <i>nomDeLaBranche</i>	Se déplace sur la branche
		\$ git checkout <i>idCommit nomDuFichier</i>	Restaure le fichier tel qu'il était au commit indiqué
		\$ git checkout <i>idCommit</i>	Récupère l'état du commit tel qu'il était au numéro indiqué. Ce n'est qu'une vision de l'état du projet. La création d'une nouvelle branche est nécessaire si on veut repartir de ce moment-là, mais garder en mémoire tout l'historique.
\$ git reset	Revenir à des versions antérieures <b>et perdre l'historique</b>	git reset --hard HEAD	Revenir à la dernière version committée.
		git reset --hard <i>idDuCommit</i>	Revenir à un commit

**Dernières  
modifications  
perdues !**

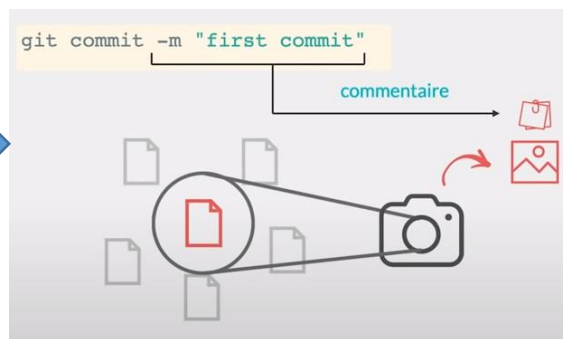


\$ git revert HEAD	Revient à l'ancien commit en conservant l'historique		
\$ git merge	Fusion de branches	\$ git merge <i>nomDeLaBranche</i>	Fusionne la branche indiquée dans la branche active
\$ git remote	Fait la connexion entre l'hébergement local et distant.	\$ git remote add origin https://github.com/toto/tutu/titiithub.git	
		\$ git remote set-url origin <i>nouveauLienDistantDeOrigin</i>	
\$ git clone	Cloner un référentiel dans un nouveau répertoire	\$ git clone https://github.com/libgit2/libgit2	Si vous voulez cloner la bibliothèque logicielle Git appelée <b>libgit2</b>
\$ git push	Met à jour les références distantes à l'aide des références locales.	\$ git push origin master	Envoie la branche <b>master</b> vers le dépôt distant nommé <b>origin</b>
		\$ git push	Envoie du dernier commit vers la branche courante du dépôt distant associé.
\$ git pull	Mise à jour des références locales à partir des références distantes	\$ git pull origin master	Envoie la branche master du dépôt distant nommé <b>origin</b> vers le dépôt local
		\$ git push	Envoie du dernier commit de la branche courante du dépôt distant vers le dépôt local

Sélection des fichiers



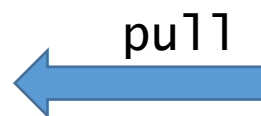
Photo des fichiers



push



pull



## 10 Références

### Comprendre Git et GitHub :

- [https://www.youtube.com/watch?v=gp\\_k0UVOYMW](https://www.youtube.com/watch?v=gp_k0UVOYMW)
- <https://www.youtube.com/watch?v=hPfgekYUKgk>
- <https://www.youtube.com/watch?v=no35TFWg0CU>

### Formation poussée sur Git :

<https://www.youtube.com/watch?v=rP3T0Ee6pLU&list=PLjwdMgw5TTLXuY5i7RW0QqGdW0NZntqiP>

<https://openclassrooms.com/fr/courses/5641721-utilisez-git-et-github-pour-vos-projets-de-developpement>

### Documentation officielle de Git :

<https://git-scm.com/docs>