

BUT Info R2.03 - TP2 : Gestion de versions

Mise en place

Il existe de nombreuses [interfaces graphiques pour GIT](#) et de nombreux développeurs les emploient dans leur pratique quotidienne. Cependant, il est indispensable de comprendre ce qui se passe quand on appuie sur un bouton. C'est pourquoi ce TP se fera en ligne de commande uniquement.

Pour Linux ou MacOS, employez le terminal. GIT est probablement installé sur votre distribution. Si ce n'est pas le cas, appuyez-vous sur les [instructions officielles](#) pour l'installation.

Pour Windows, la meilleure chose à faire est d'utiliser le [terminal MS Windows](#) ainsi que le [sous-système WSL \(Linux\)](#). Après l'installation de WSL, si les choses ne se déroulent pas comme attendu, il sera peut-être tout de même nécessaire d'installer une distribution (comme Ubuntu par exemple) à partir du [Windows Store](#). Les captures d'écran de ce TP sont issues de cette configuration.

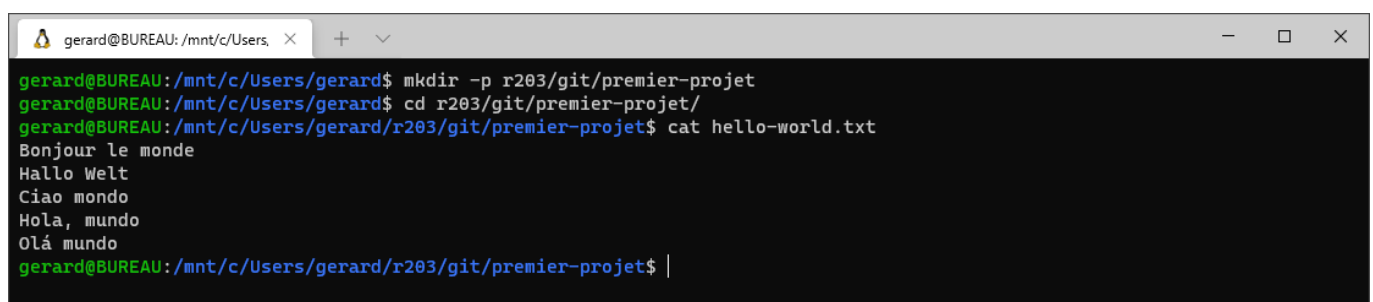
Initialisation d'un dépôt

Initialisation locale

Il existe deux grandes manières de mettre en place la gestion de version pour un projet avec GIT : soit en initialisant directement un dépôt local, soit en clonant un projet distant.

Sur votre système de fichiers, créez un dossier pour ce TP GIT et à l'intérieur, un dossier premier-projet pour votre premier projet. Dans ce dernier dossier, créez un fichier `hello-world.txt` avec le contenu suivant :

```
Bonjour le monde  
Hallo Welt  
Ciao mondo  
Hola, mundo  
Olá mundo
```



```
gerard@BUREAU: /mnt/c/Users/gerard$ mkdir -p r203/git/premier-projet  
gerard@BUREAU: /mnt/c/Users/gerard$ cd r203/git/premier-projet/  
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ cat hello-world.txt  
Bonjour le monde  
Hallo Welt  
Ciao mondo  
Hola, mundo  
Olá mundo  
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Ce projet est pour l'instant strictement local, aucunement sous contrôle de version. Pour transformer le dossier du projet en dépôt GIT, il suffit de lancer la commande `git init` à partir du dossier courant `premier-projet`.

```
gerard@BUREAU: /mnt/c/Users, X + v
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ git init
Initialized empty Git repository in /mnt/c/Users/gerard/r203/git/premier-projet/.git/
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ ls -al
total 0
drwxrwxrwx 1 gerard gerard 4096 Dec 22 11:15 .
drwxrwxrwx 1 gerard gerard 4096 Dec 22 11:02 ..
drwxrwxrwx 1 gerard gerard 4096 Dec 22 11:15 .git
-rwxrwxrwx 1 gerard gerard 67 Dec 22 11:08 hello-world.txt
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Cette commande crée un dossier `.git` (le `.` rend le dossier caché sous Linux/MacOS) avec à l'intérieur tous les fichiers nécessaires à la gestion de version avec GIT. En particulier, on y trouve - sous une forme cryptolabyrinthique - tout l'historique des versions du projet. Ce que vous venez de créer, c'est un dépôt GIT à part entière. Avec un accès à votre poste de travail à partir d'une autre machine, il est par exemple tout à fait possible de synchroniser le projet sur la nouvelle machine avec le projet actuel, de manière décentralisée.

```
gerard@BUREAU: /mnt/c/Users, X + v
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ git init
Initialized empty Git repository in /mnt/c/Users/gerard/r203/git/premier-projet/.git/
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ ls -al
total 0
drwxrwxrwx 1 gerard gerard 4096 Dec 22 11:15 .
drwxrwxrwx 1 gerard gerard 4096 Dec 22 11:02 ..
drwxrwxrwx 1 gerard gerard 4096 Dec 22 11:15 .git
-rwxrwxrwx 1 gerard gerard 67 Dec 22 11:08 hello-world.txt
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Pour obtenir l'état de votre dépôt, lancez la commande `git status`.

Ici, la sous-commande `status` indique que le fichier `hello-world.txt` n'est pas sous contrôle de version. Pour y remédier, on emploie la sous-commande `add` en précisant quel(s) fichier(s) ou dossier(s) on souhaite mettre sous contrôle de version. Ici, ne faisons pas de quartier et mettons le dossier complet sous contrôle de version : `git add ..`

```
gerard@BUREAU: /mnt/c/Users, X + v
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello-world.txt

nothing added to commit but untracked files present (use "git add" to track)
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ git add .
gerard@BUREAU:/mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Ceci fait, le fichier `hello-world.txt` est sous contrôle de version mais il n'a pas encore fait l'objet d'une version. Pour créer une nouvelle version en prenant en compte les modifications locales, c'est la sous-commande `commit` qui doit être employée. La première fois, vous devez juste renseigner votre identité pour pouvoir faire des commits (ainsi, GIT sait qui poste des nouvelles versions). Pour ce faire, on emploie la sous-commande `config`.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git config user.email "pierre.gerard@univ-paris13.fr"
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git config user.name "Pierre Gérard"
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ |
```

(avec votre adresse et votre nom à vous)

Ceci fait, vous pouvez créer la première version dans le dépôt en lançant `git commit -m "premier commit"`.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   hello-world.txt

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git commit -m "premier commit"
[master (root-commit) afd1ddb] premier commit
1 file changed, 5 insertions(+)
create mode 100644 hello-world.txt
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Le statut de votre projet devrait désormais être *clean*.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git status
On branch master
nothing to commit, working tree clean
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Publication sur GitHub

Bien que GIT soit décentralisé (à l'inverse de [SVN](#) typiquement), on emploie souvent GIT de manière centralisée, avec un dépôt principal en ligne, sur une plateforme garantissant une disponibilité constante. Parmi ces plateformes, on peut citer [GitLab](#) et [GitHub](#).

Vous allez maintenant [publier sur GitHub](#) votre projet premier-projet. Pour ce faire, créez un compte sur GitHub et ceci fait, créez un dépôt nommé premier-projet.


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *

Repository name *

 pierre-gerard ▾

/

premier-projet



Great repository names are short and memorable. Need inspiration? How about [fluffy-telegram?](#)

Description (optional)

Dans le cadre du TP GIT du module R203 en BUT Informatique



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

L'écran suivant vous indique les commandes à exécuter ensuite pour publier sur GitHub le projet sur votre poste de travail.

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

 Set up in Desktop or **HTTPS** **SSH**

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 "# premier-projet" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/pierre-gerard/premier-projet.git
git push -u origin main
```

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

```
git remote add origin https://github.com/pierre-gerard/premier-projet.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

Le première commande `git remote add origin`

<https://github.com/pierre-gerard/premier-projet.git> (remplacez par l'URL de votre propre dépôt sur GitHub) permet de lier votre projet local au dépôt nouvellement créé sur GitHub. Exécutez la. Si vous avez fait une erreur, il est possible de modifier l'adresse d'un dépôt distant :

<https://docs.github.com/en/get-started/git-basics/managing-remote-repositories>.

Si vous exécutez directement les commandes suivantes (ne le faites pas), vous devriez obtenir une erreur à cause d'un renforcement de la méthode d'authentification.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git branch -M main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git push -u origin main
Username for 'https://github.com': pierre.gerard@univ-paris13.fr
Password for 'https://pierre.gerard@univ-paris13.fr@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/pierre-gerard/premier-projet.git/'
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$
```

Désormais, en effet, un mot de passe ne suffit plus, il faut définir un [token d'authentification](#) et l'utiliser à la place du mot de passe. Faites le nécessaire pour obtenir un token d'identification personnel. Concernant le champ d'application du token, cochez rep. Attention : les “developer settings” mentionnées dans la documentation sont accessibles depuis votre page de profil GitHub, pas depuis la page d'un projet en particulier.

- GitHub Apps
- OAuth Apps
- Personal access tokens

New personal access token

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](#).

Note

TP R203

What's this token for?

Expiration *

30 days

The token will expire on Fri, Jan 21 2022

Select scopes

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

- | | |
|---|--------------------------------------|
| <input checked="" type="checkbox"/> repo | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status | Access commit status |
| <input checked="" type="checkbox"/> repo_deployment | Access deployment status |
| <input checked="" type="checkbox"/> public_repo | Access public repositories |
| <input checked="" type="checkbox"/> repo:invite | Access repository invitations |
| <input checked="" type="checkbox"/> security_events | Read and write security events |

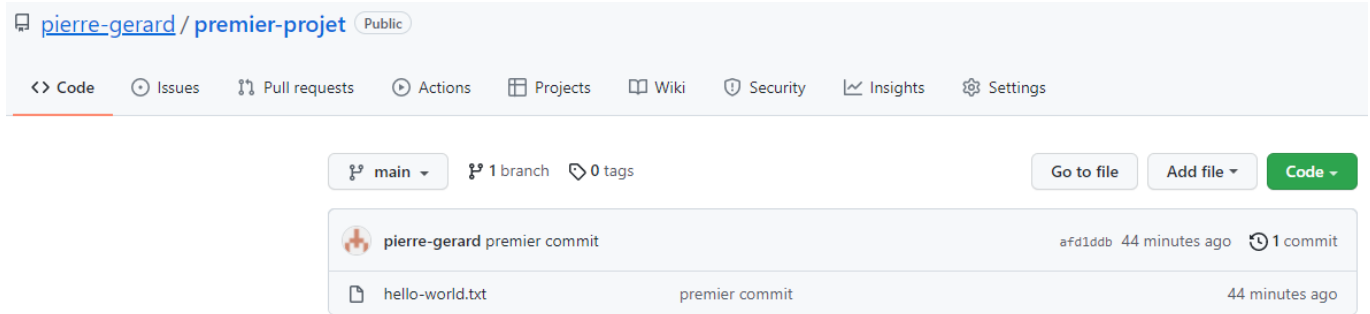
Ne le faites pas pour ne pas alourdir le TP mais il existe des solutions pour que le token ne vous soit pas demandé à chaque commande git : <https://git-scm.com/docs/git-credential-store>.

Ceci fait, lancez les commandes prévues et remplacez votre mot de passe par le token obtenu. Ici, on positionne le projet sur une branche main. Cette branche main est la branche principale habituelle avec GitHub, en remplacement de master. Les deux dénominations jouent en fait le même rôle.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git branch -M main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git push -u origin main
Username for 'https://github.com': pierre.gerard@univ-paris13.fr
Password for 'https://pierre.gerard@univ-paris13.fr@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 285 bytes | 285.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/pierre-gerard/premier-projet.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Désormais, votre projet local est publié sur GitHub. Remarquez que le commit premier commit que vous aviez fait en local apparaît sur GitHub. C'est bien tout le projet (avec toutes ses versions) qui a été publié sur GitHub, pas uniquement la version courante.

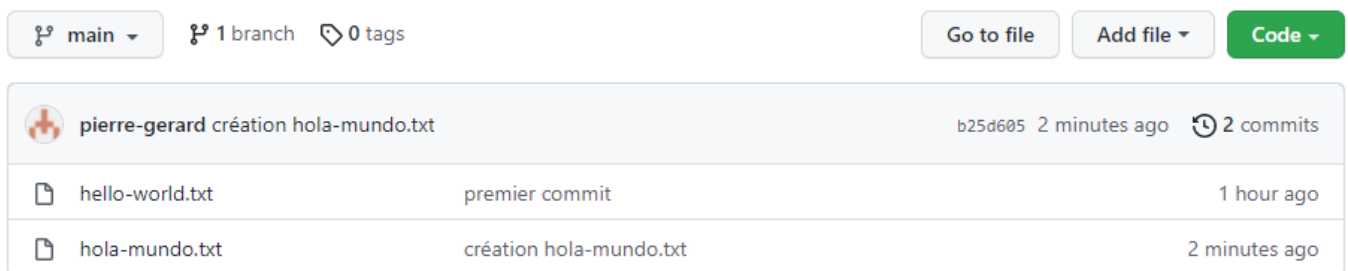


Faites le nécessaire pour :

1. Créer un fichier `hola-mundo.txt` localement
2. Le mettre sous contrôle de version
3. Créer une nouvelle version sur le dépôt local avec ce fichier
4. Envoyer le tout sur GitHub avec la sous-commande `push` employée cette fois sans paramètre

Selon les images installées à l'IUT, vous pourriez rencontrer un problème lié à un proxy dysfonctionnel (`fatal : unable to access 'http://...'`). Dans ce cas, supprimez le fichier `.gitconfig` à la racine de votre compte (commande `rm ~/.gitconfig`).

Vous devriez obtenir le résultat suivant sur GitHub :



Correction (mais essayez de faire vous même) :

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ cp hello-world.txt hola-mundo.txt
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ # modifier hola-mundo.txt
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git add hola-mundo.txt
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git commit -m "création hola-mundo.txt"
[main b25d605] création hola-mundo.txt
1 file changed, 5 insertions(+)
create mode 100644 hola-mundo.txt
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git push
Username for 'https://github.com': pierre.gerard@univ-paris13.fr
Password for 'https://pierre.gerard@univ-paris13.fr@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 278 bytes | 278.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To https://github.com/pierre-gerard/premier-projet.git
afdddb..b25d605 main -> main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ |
```

Clonage d'un dépôt distant

Sur votre poste de travail, supprimez purement et simplement le dossier `premier-projet` avec la commande `rm -rf premier-projet` lancée à partir du dossier courant `git`. A ce point, vous n'avez plus de

copie locale du dépôt, mais il est toujours sur GitHub.

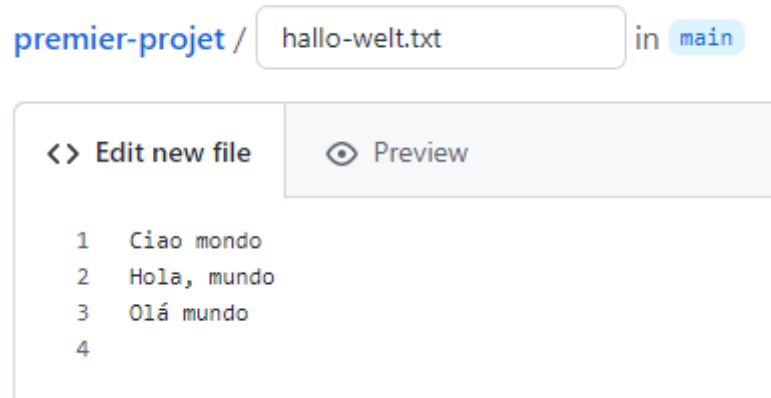
```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git$ rm -rf premier-projet/
gerard@BUREAU: /mnt/c/Users/gerard/r203/git$ ls
gerard@BUREAU: /mnt/c/Users/gerard/r203/git$ |
```

Pour instancier une copie locale du dépôt distant, on emploie la sous-commande `clone` qui a pour effet de cloner localement le dépôt distant complet, avec toutes les versions. La page principale d'un projet permet de trouver l'URL à partir de laquelle cloner le projet du dépôt. Dans le cas du rédacteur de ce TP, l'URL en question est <https://github.com/pierre-gerard/premier-projet.git> et la commande à lancer est la suivante (à adapter selon votre propre URL) :

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git$ git clone https://github.com/pierre-gerard/premier-projet.git
Cloning into 'premier-projet'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 5 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), 511 bytes | 39.00 KiB/s, done.
gerard@BUREAU: /mnt/c/Users/gerard/r203/git$ ls
premier-projet
gerard@BUREAU: /mnt/c/Users/gerard/r203/git$ |
```

Vous disposez à nouveau d'une copie locale de votre projet, avec tous les *commits* qui avaient été fait auparavant. Si d'autres personnes font de même et clonent également le projet, ils pourront créer de nouvelles versions sur GitHub, que vous pourrez récupérer avec la sous-commande `pull`.

Pour illustrer rapidement cette situation, utilisez l'interface web de GitHub pour créer un nouveau fichier `hallo-welt.txt`. Si un autre utilisateur avait poussé (push) un tel fichier à partir de son propre dépôt local, l'effet aurait été semblable.



En bas de la page, il n'y a pas de bouton Enregistrer mais un bouton Commit à la place, évidemment.



Commit new file

Create hallo-welt.txt

Add an optional extended description...

☒ Commit directly to the `main` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit new file

Cancel

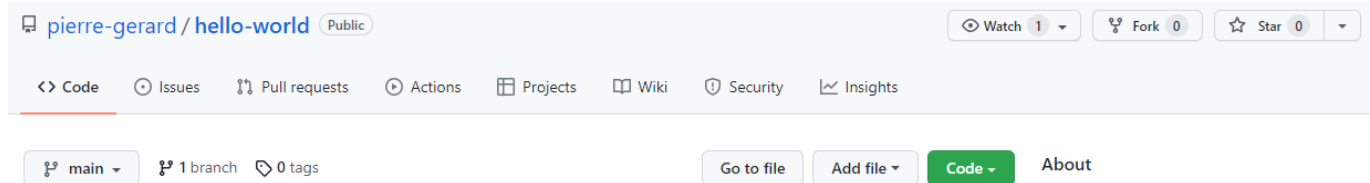
Pour récupérer la nouvelle version du projet sur votre poste de travail, employez la sous-commande `pull`.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 709 bytes | 28.00 KiB/s, done.
From https://github.com/pierre-gerard/premier-projet
   b25d605..b35d698  main       -> origin/main
Updating b25d605..b35d698
Fast-forward
 hallo-welt.txt | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 hallo-welt.txt
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ ls
hallo-welt.txt  hello-world.txt  hola-mundo.txt
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ cat hallo-welt.txt
Ciao mondo
Hola, mundo
Olá mundo
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/premier-projet$ |
```

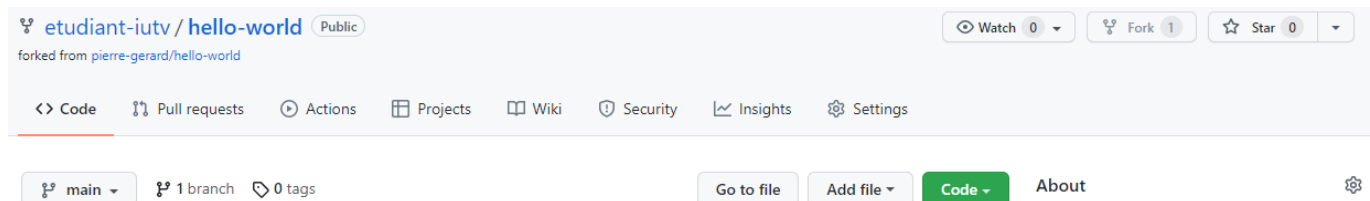
Fork d'un projet

Quand on veut expérimenter des modifications importantes dans un projet, plutôt que de travailler sur une copie complètement séparée, il est possible de créer un *fork* d'un dépôt existant. Le dépôt résultant de ce *fork* est un nouveau dépôt : les *commits* et créations de *branches* n'auront aucun impact sur le dépôt original.

Assurez-vous d'être bien logués sur votre compte GitHub. Rendez-vous sur la page du dépôt suivant : <https://github.com/pierre-gerard/hello-world>.



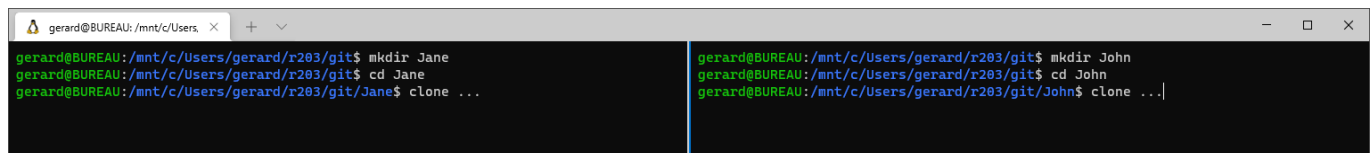
A partir de la page du dépôt hello-world de pierre-gerard, appuyez sur le bouton fork en haut à droite. Vous devriez obtenir un nouveau dépôt hello-world mais dans votre compte (pour l'exemple, etudiant-iutv). Ce dépôt est le vôtre, et ce que vous y ferez n'affectera pas le dépôt original de pierre-gerard.



Gestion des versions

Plusieurs dépôts locaux

Dans votre dossier r203/git, créez deux dossiers Jane et John. Ouvrez deux terminaux côté à côté, chacun avec un des dossiers créés comme dossier courant. Dans chacun des terminaux/dossiers Jane et John, clonez le projet etudiant-iutv/hello-world résultant du fork effectué plus haut.



Vous disposez maintenant de deux dépôts locaux issus de la même origine, et avec laquelle chacun peut rester synchronisé. Dans cette configuration, on a en réalité **un seul** utilisateur GitHub travaillant sur plusieurs dépôts locaux (comme sur plusieurs postes de travail) et pas vraiment plusieurs utilisateurs. On aurait pu le faire en a) créant plusieurs comptes GitHub et en b) gardant un seul dépôt sur GitHub mais c) plusieurs contributeurs déclarés sur GitHub et d) dans chaque terminal, un identifiant différents pour les push et les add. Mais dans tous les cas, les mécanismes expérimentés dans la suite restent semblables, avec le même utilisateur ou des différents, pourvu qu'on travaille sur plusieurs dépôts locaux distincts. Dans la suite, il y aura en conséquence des abus de langage de type *“du point de vue de Jane”* signifiant plus rigoureusement *“à partir du dépôt hello-world dans le dossier Jane”*.

Remarque : si vous ne souhaitez pas taper votre mot de passe et le token à chaque fois que vous faites un add ou un push, il est possible de configurer GIT en ce sens. Vous pouvez trouver les différentes procédures possibles dans la [documentation](#). La méthode la plus simple et raisonnablement sûre est de stocker l'authentification en cache avec la commande `git config --global credential.helper cache` (il y a deux - devant global). L'option `--global` indique que c'est valable pour tous les dépôts sur le poste de travail. Si on veut des contributeurs différents pour chaque dépôt, on doit enlever cette option et la remplacer par `--local` (ou rien) dans des commandes lancées pour chaque dépôt local. Le cache a une durée par défaut de 15 min. La méthode `store` est plus permanente mais elle stocke des informations sensibles en clair dans vos fichiers. Elle n'est pas recommandée en général, et surtout pas si vous n'êtes pas le seul à pouvoir accéder au dossier local du dépôt. La manière recommandée est d'employer des clés SSH.

Modifications en séquence

Les fichiers Jane/hello-world/src/HelloWord.java et John/hello-world/src/HelloWord.java contiennent le code suivant :

```
public class HelloWord {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World !");  
    }  
  
}
```

Il manque un `l` à HelloWord dans le nom de la classe pour en faire HelloWorld.

Du point de vue de **Jane**

Modifiez le fichier Jane/hello-world/src/HelloWord.java pour corriger l'erreur. Dans le terminal correspondant au dépôt Jane, propagez la modification jusqu'au dépôt GitHub (commit puis push).

Vérifiez dans le navigateur que le fichier a bien été modifié sur le dépôt distant.

The screenshot shows the GitHub interface for the repository 'etudiant-iutv / hello-world'. At the top, it indicates the repository is 'Public' and 'forked from pierre-gerard/hello-world'. Below this are navigation tabs: 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Code' tab is selected, showing the file path 'main / hello-world / src / HelloWorld.java'. A commit history section shows a recent commit by 'etudiant-iutv' titled 'Ajout d'un l à word' with the latest commit hash 'b2f545d' made '1 minute ago'. Below this, the file content is displayed, showing the corrected Java code for 'HelloWorld'.

```
1  
2 public class HelloWorld {  
3  
4     public static void main(String[] args) {  
5         System.out.println("Hello World !");  
6     }  
7  
8 }
```

Du point de vue de **John**

Une modification de Jane a corrigé le contenu du fichier. Pour récupérer la nouvelle version dans le dépôt de John, il faut employer la sous-commande `pull`.

```

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git commit -a -m "Ajout d'un l à word"
[main b2f545d] Ajout d'un l à word
1 file changed, 1 insertion(+), 1 deletion(-)
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git push
Username for 'https://github.com': gerard.up13@gmail.com
Password for 'https://gerard.up13@gmail.com@github.com':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 363 bytes | 181.00 KiB/s, done.
Total 4 (delta 1), reused 4 (delta 1), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/etudiant-iutv/hello-world.git
 4917692..b2f545d main -> main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 4 (delta 1), pack-reused 0
Unpacking objects: 100% (4/4), 343 bytes | 15.00 KiB/s, done.
From https://github.com/etudiant-iutv/hello-world
 4917692..b2f545d main -> origin/main
Updating 4917692..b2f545d
Fast-forward
 src/HelloWorld.java | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$

```

Vérifiez que le fichier a bien été modifié sur votre système de fichier local.

Une autre erreur n'a pas été corrigée : le `l` manque également au nom du fichier `HelloWorld.java`. Ce fichier étant sous contrôle de version, il ne suffit pas d'employer la commande `mv HelloWorld.java HelloWorld.java`. Il est effectivement nécessaire d'employer `mv` mais en tant que sous-commande de GIT.

```
$ git mv HelloWorld.java HelloWorld.java
```

Il en va de même lorsque l'on souhaite supprimer un fichier à la fois du disque dur et du dépôt : on emploie alors la commande `git rm`

Propagez les modifications jusque sur le dépôt GitHub.

Du point de vue de **Jane**

Récupérez les modifications de John et vérifiez sur votre copie de travail locale.

```

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 293 bytes | 26.00 KiB/s, done.
From https://github.com/etudiant-iutv/hello-world
 b2f545d..dc8bc17 main -> origin/main
Updating b2f545d..dc8bc17
Fast-forward
 src/{HelloWorld.java => HelloWorld.java} | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename src/{HelloWorld.java => HelloWorld.java} (100%)
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ ls
README.md  HelloWorld.java
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ ls src
HelloWorld.java

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git push
Username for 'https://github.com': gerard.up13@gmail.com
Password for 'https://gerard.up13@gmail.com@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 156.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/etudiant-iutv/hello-world.git
 b2f545d..dc8bc17 main -> main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$

```

Modifications simultanées

A ce point, les deux dépôts locaux sont synchronisés sur le dépôt distant.

Du point de vue de **Jane**

Modifiez le code en ajoutant une nouvelle ligne après le `System.out.println` existant.

```

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World !");
        System.out.println("Hola Mundo !");
    }

}

```

Faites un commit et restez en là pour l'instant.

Du point de vue de **John**

Modifiez le code en ajoutant une nouvelle ligne avant le `System.out.println` existant.

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Ciao mondo !");
        System.out.println("Hello World !");
    }

}
```

Faites un commit. A ce point, chaque dépôt local est en avance d'un commit sur le dépôt distant, mais avec des modifications différentes.

Propagez les modifications de John avec un push. Jusqu'ici, tout va bien parce que le dépôt distant ne sait pas qu'il y a un conflit avec le dépôt de Jane.

Du point de vue de **Jane**

Essayez de propager les modifications vers le dépôt distant.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git push
Username for 'https://github.com': 'C
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git config credential.helper cache
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git push
Username for 'https://github.com': gerard.up13@gmail.com
Password for 'https://github.com': gerard.up13@gmail.com
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 363 bytes | 363.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/etudiant-iutv/hello-world.git
dc8bc17..9954861 main -> main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$

On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git push
Username for 'https://github.com': gerard.up13@gmail.com
Password for 'https://github.com': gerard.up13@gmail.com
To https://github.com/etudiant-iutv/hello-world.git
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/etudiant-iutv/hello-world.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$
```

Ca ne fonctionne pas : la comparaison entre l'état du dépôt distant et celui du dépôt local de Jane permet de craindre un éventuel conflit. Ce type de conflit doit toujours être réglé localement. Le message d'erreur préconise de faire un `pull` pour intégrer les modifications distantes au dépôt local. Un éditeur s'ouvrira peut-être, où il est question de merge. Vous pouvez faire des commentaires ou/puis appuyer simultanément sur `Ctrl+X` pour fermer l'éditeur nano.

Il est aussi question de merge (*fusion* en français) dans le retour de git sur la console de Jane. Ici, comme les modifications de Jane et de John ne portaient pas sur la même ligne de code, GIT a réussi à fusionner automatiquement les deux versions. Le code de Jane est maintenant le suivant, et la version de travail de Jane est en avance sur le dépôt distant.

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Ciao mondo !");
        System.out.println("Hello World !");
        System.out.println("Hola Mundo !");
    }

}
```

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 4 (delta 1), pack-reused 0
Unpacking objects: 100% (4/4), 343 bytes | 34.00 KiB/s, done.
From https://github.com/etudiant-iutv/hello-world
dc8bc17..9954861 main --> origin/main
Auto-merging src/HelloWorld.java
Merge made by the 'recursive' strategy.
src/HelloWorld.java | 1 +
1 file changed, 1 insertion(+)
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ |

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git push
Username for 'https://github.com': gerard.up13@gmail.com
Password for 'https://gerard.up13@gmail.com@github.com':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 363 bytes | 363.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/etudiant-iutv/hello-world.git
dc8bc17..9954861 main --> main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$
```

Propagez la version de Jane sur le dépôt distant.

Du point de vue de **John**

Intégrez les modifications de Jane pour que tous les dépôts soient synchronisés.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git push
Username for 'https://github.com': gerard.up13@gmail.com
Password for 'https://gerard.up13@gmail.com@github.com':
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 750 bytes | 375.00 KiB/s, done.
Total 8 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 1 local object.
To https://github.com/etudiant-iutv/hello-world.git
9954861..e2141ec main --> main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git pull
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 8 (delta 2), reused 8 (delta 2), pack-reused 0
Unpacking objects: 100% (8/8), 730 bytes | 34.00 KiB/s, done.
From https://github.com/etudiant-iutv/hello-world
9954861..e2141ec main --> origin/main
Updating 9954861..e2141ec
Fast-forward
src/HelloWorld.java | 1 +
1 file changed, 1 insertion(+)
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$
```

Conflits au moment de la fusion

Dans l'immense majorité des cas, des développeurs interviennent dans la même journée sur des portions de code différentes. Mais il peut néanmoins arriver que la même ligne soit modifiée simultanément sur plusieurs dépôts locaux. Dans ce cas, la fusion automatique (merge) ne fonctionne pas et il est nécessaire de traiter le conflit manuellement.

Du point de vue de **John**

Modifiez le code de la manière suivante, faites un commit et un push.

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Ciao mondo !");
        System.out.println("Hallo Welt !");
        System.out.println("Hola Mundo !");
    }

}
```

Du point de vue de **Jane**

Modifiez le code de la manière suivante, faites un commit et un push.

```
public class HelloWorld {
```

```

public static void main(String[] args) {
    System.out.println("Ciao mondo !");
    System.out.println("Olá mundo !");
    System.out.println("Hola Mundo !");
}
}

```

Evidemment, le push pose problème pour les mêmes raisons que précédemment mais comme la même ligne a été modifiée deux deux côtés, la fusion automatique ne fonctionne pas.

Pour résoudre le conflit, il convient d'éditer le fichier HelloWorld.java.

```

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Ciao mondo !");
<<<<<<< HEAD
        System.out.println("Olá mundo !");
=====
        System.out.println("Hallo Welt !");
>>>>>>> 1a2a6ff8034f79346f5e296400f520b086e9d9ea
        System.out.println("Hola Mundo !");
    }
}

```

GIT a ajouté des annotations pour indiquer où est le conflit. Pour mettre tout le monde d'accord, écrivez la ligne centrale en français et sauvegardez.

```

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Ciao mondo !");
        System.out.println("Bonjour le monde !");
        System.out.println("Hola Mundo !");
    }
}

```


On aurait pu arbitrer le conflit autrement mais en tout état de cause, c'est un humain qui doit le faire, et sur une copie de travail locale. Une fois le conflit arbitré, on doit faire un add pour forcer le fichier en conflit à être repris en considération, puis on peut faire un commit et enfin un push.

```
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git add .
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git commit -m "Arbitré"
[main a250825] Arbitré
1 file changed, 1 insertion(+), 5 deletions(-)
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$ git push
Username for 'https://github.com': gerard.up13@gmail.com
Password for 'https://gerard.up13@gmail.com@github.com':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 383 bytes | 383.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/etudiant-iutv/hello-world.git
6645094..a250825 main -> main
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/Jane/hello-world$

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 4 (delta 1), pack-reused 0
Unpacking objects: 100% (4/4), 363 bytes | 33.00 KiB/s, done.
From https://github.com/etudiant-iutv/hello-world
6645094..a250825 main -> origin/main
Updating 6645094..a250825
Fast-forward
 src/HelloWorld.java | 6 +-----
1 file changed, 1 insertion(+), 5 deletions(-)
gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$ cat src/HelloWorld.java


public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Ciao mondo !");
        System.out.println("Bonjour le Monde !");
        System.out.println("Hola Mundo !");
    }
}

gerard@BUREAU: /mnt/c/Users/gerard/r203/git/John/hello-world$
```

Pour aller plus loin

Sur le dépôt GitHub, vous pouvez voir l'historique de toutes les versions du dépôt. Chaque dépôt local les possède toutes également et il est possible de revenir à des versions antérieures si on le souhaite.

 **etudiant-iutv / hello-world** Public

[forked from pierre-gerard/hello-world](#)

[Code](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

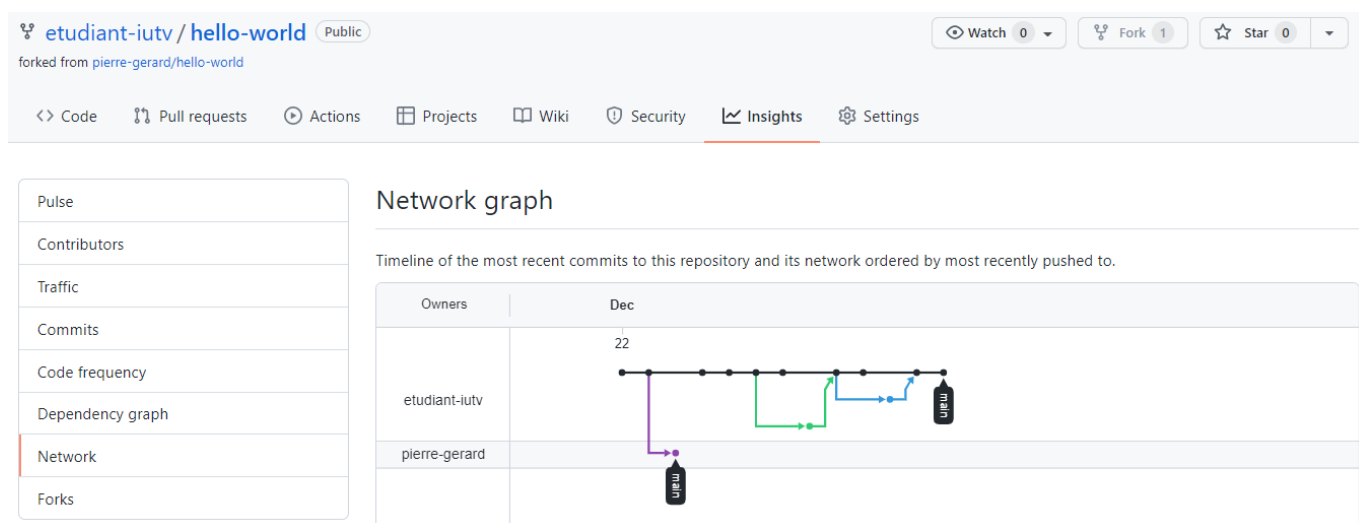
[main](#)

Commits on Dec 22, 2021

Arbitré	etudiant-iutv committed 6 minutes ago	a250825	View
Résolu	etudiant-iutv committed 10 minutes ago	6645094	View
DE	etudiant-iutv committed 26 minutes ago	1a2a6ff	View
PT	etudiant-iutv committed 26 minutes ago	5893183	View
Merge branch 'main' of https://github.com/etudiant-iutv/hello-world i...	etudiant-iutv committed 1 hour ago	e2141ec	View
IT	etudiant-iutv committed 1 hour ago	9954861	View
ES	etudiant-iutv committed 1 hour ago	fea0185	View
Fichier	etudiant-iutv committed 1 hour ago	dc8bc17	View
Ajout d'un l à word	etudiant-iutv committed 2 hours ago	b2f545d	View
Ajout src	etudiant-iutv committed 2 hours ago	4917692	View
Update README.md	pierre-gerard committed 9 hours ago	7dd00a6	Verified View
Initial commit	pierre-gerard committed 9 hours ago	a6ce980	Verified View

[Newer](#) [Older](#)

On peut également représenter l'historique du projet sous la forme d'un graphe, chaque noeud figurant une version (un commit).



Il est également possible de créer des branches de développement avec la sous-commande `branch` et de passer d'une branche à l'autre avec la sous-commande `checkout`. Fusionner des branches fonctionne de manière similaire aux fusions de versions, et avec les mêmes limitations. Ces fusions s'effectuent avec la sous-commande `merge`.

La gestion des branches est un des atouts majeurs de GIT et mérite d'être maîtrisée. A ce sujet, vous pouvez consulter [la documentation de GIT](https://git-scm.com/book/en/v2). Cette page est une partie d'un livre publié en ligne, très complet et facile à lire, ouvrage de référence sur GIT traduit en français : <https://git-scm.com/book/en/v2>. Pour une documentation technique sous-commande par sous-commande, on peut se référer à la documentation suivante : <https://git-scm.com/docs>.

Votre propre dépôt `hello-world` avait été créé par un *fork* de `pierre-gerard/hello-world`. Un *fork* est indépendant de son dépôt d'origine (avec ses propres versions en branches) mais il conserve néanmoins un lien avec lui. Après un *fork*, il est possible de demander aux développeurs du dépôt original un [pull request](#). Si les modifications sont acceptées, alors le *fork* peut être intégré au projet d'origine. Là encore, les mécanismes de fusion automatique sont précieux.

From:

<https://www-info.iutv.univ-paris13.fr/dokuwiki/> - infoWiki

Permanent link:

<https://www-info.iutv.univ-paris13.fr/dokuwiki/doku.php?id=but-info-r203:git-exos>

Last update: **15/05/2025 13:11**

