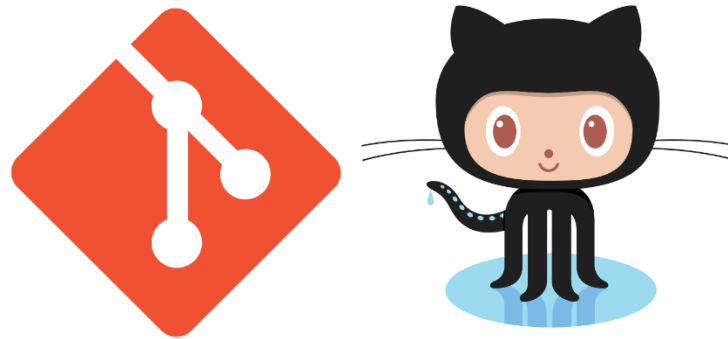


Projet C++ Atelier Git/Github



Hanen JABNOUN

- Introduction
- Centralized VCS vs. Distributed VCS
- Pourquoi git ?
- Création d'un compte sur Github
- Installation du git
- Les commandes sur Git Bash

Objectifs :

- ☐ Comprendre l'utilité de la gestion des versions (Eng: version control)
- ☐ Savoir utiliser git et github
- ☐ Créer une branche
- ☐ Pratiquer les commandes de git avec Git Bash

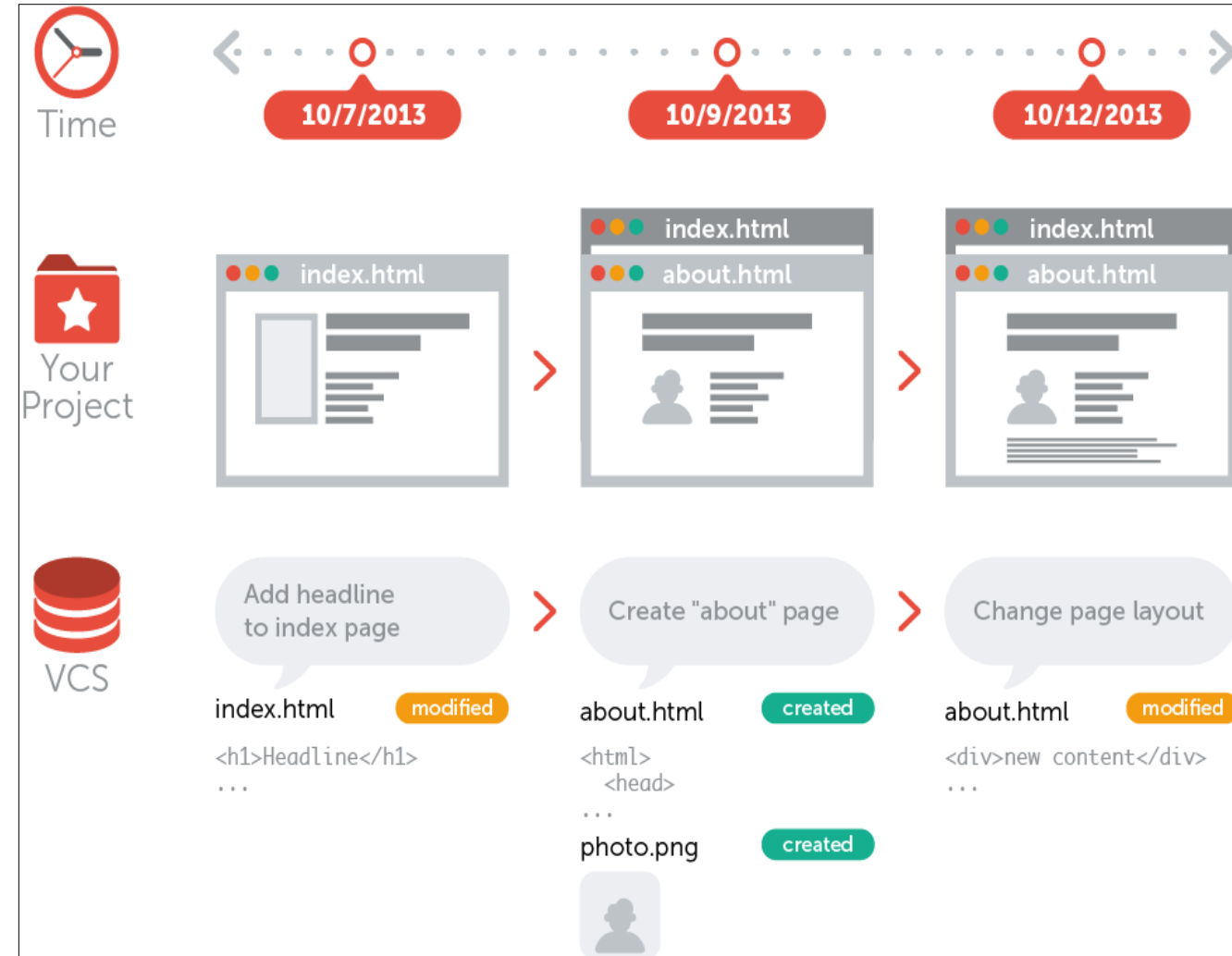
Introduction

C'est quoi la gestion des versions?

Gérer l'ensemble des versions d'un ou plusieurs fichiers

(code sources)

- ✓ Collaboration
- ✓ Sauvegarde des versions.
- ✓ Restaurer version antérieur.
- ✓ Garder trace des changements (messages indicatifs)
- ✓ Backup

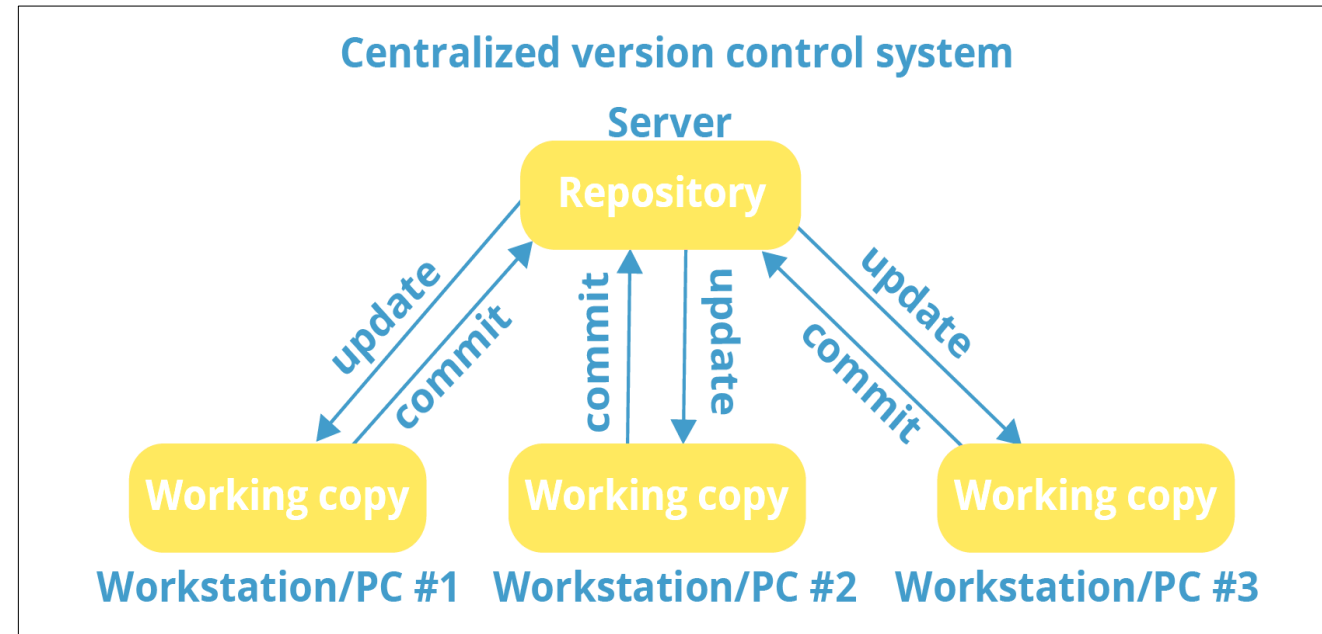


Centralized VCS vs. Distributed VCS

CVCS: Centralized Version Control System

Serveur central pour sauvegarder toutes les données.

- Collaboration entre les membres de l'équipe.
- Un seul dépôt de travail (repository)
- Chaque utilisateur possède une copie de travail.
- Commit des changements dans le dépôt sur le serveur
- Update: pour mettre à jour les copies de travail en local.



CVCS: Centralized Version Control System

- + Facile à apprendre et à gérer
 - + Fonctionne bien avec les fichiers binaires
 - + ++ Plus de contrôle sur les utilisateurs et leur accès.
-
- **Non disponible localement:** se connecter au réseau pour effectuer des opérations.
 - **Risque d'une perte de données** si le serveur tombe en panne.
 - **Pas de rapidité:** connexion obligatoire pour effectuer ou récupérer les mises à jour.

Exemple:



Centralized VCS vs. Distributed VCS

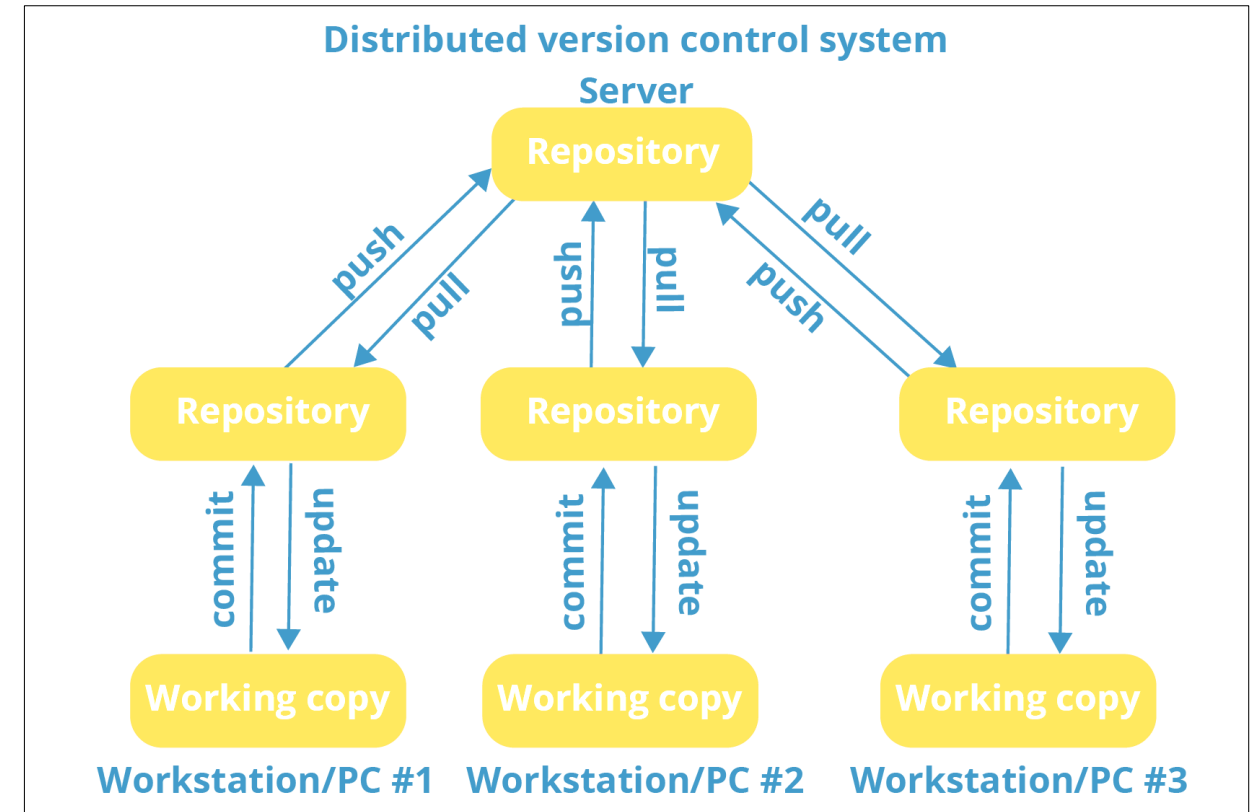
DVCS: Distributed Version Control System

Gérer l'ensemble des versions d'un ou plusieurs fichiers
(code source).

Mettre à jour les modifications dans le référentiel local.

Transmettre les modifications au dépôt central.

Pour vérifier les modifications, il faut extraire le dépôt
central mis à jour dans le référentiel local et mettre à jour
dans la copie locale.

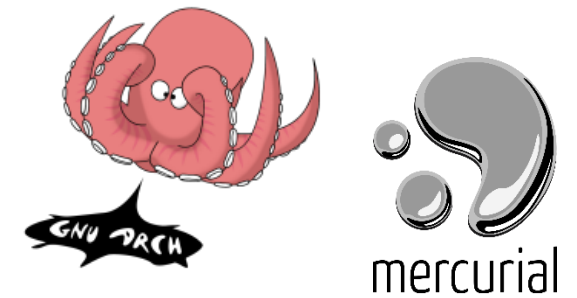


Centralized VCS vs. Distributed VCS

DVCS: Distributed Version Control System

Exemple:

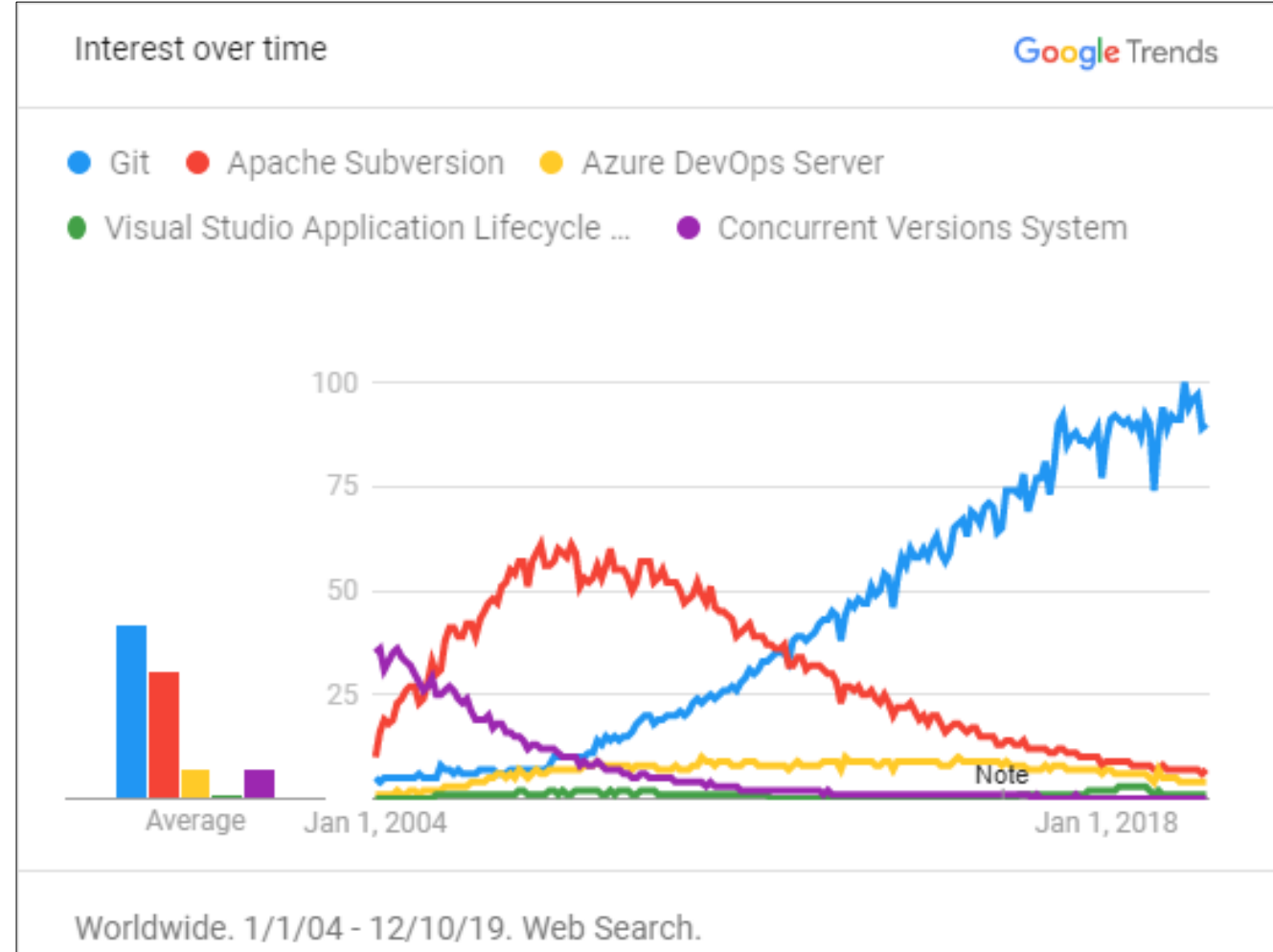
- + Travail hors ligne: l'utilisateur peut travailler hors ligne dans DVCS
 - + Rapidité: DVCS est rapide par rapport à CVCS
 - + Fusion et branchement des modifications très simples
 - + Pas de perte des données: le code sera stocké dans les systèmes locaux
-
- Verrouillage de fichiers limite l'accès à différents développeurs pour travailler simultanément sur le même morceau de code. → ralentit le développement
 - Cloner le référentiel → peut engendrer un problème de sécurité
 - Gestion des fichiers non fusionnables (exemple fichiers binaires) → nécessite une énorme quantité d'espace, et les développeurs ne peuvent pas faire de différences



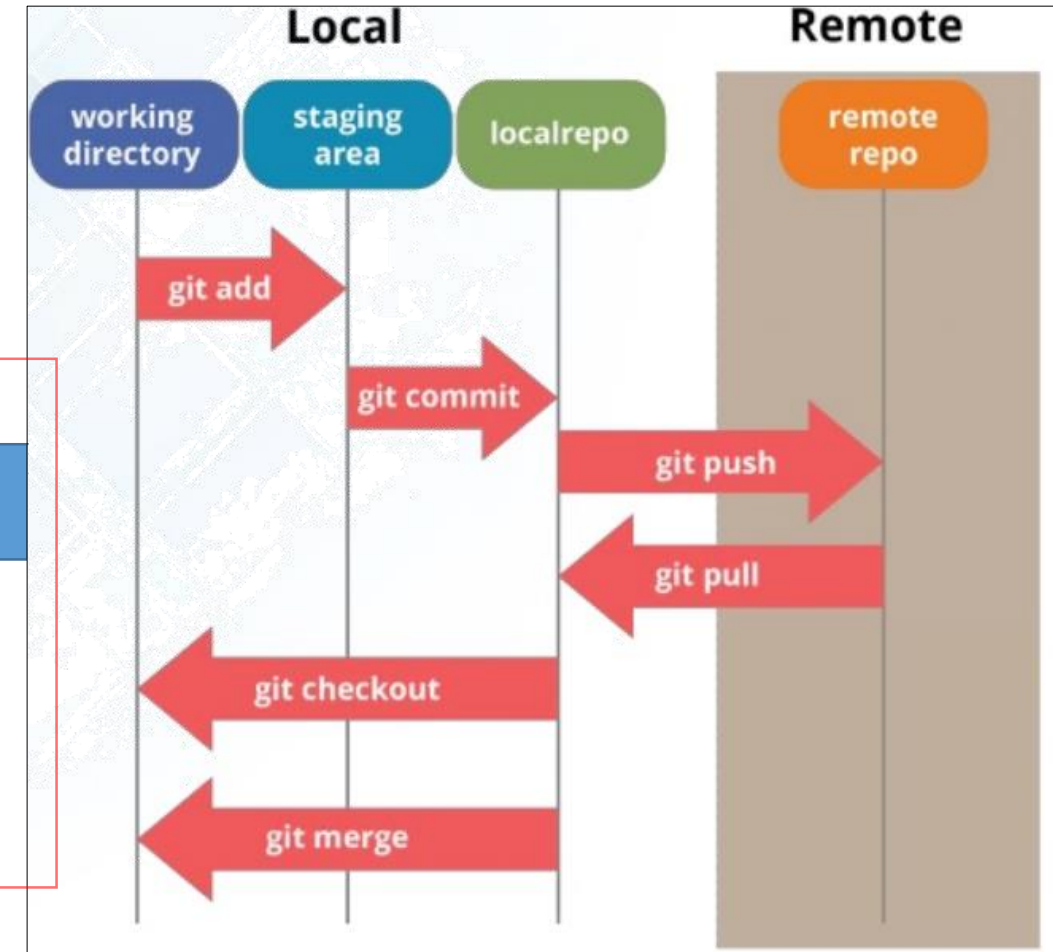
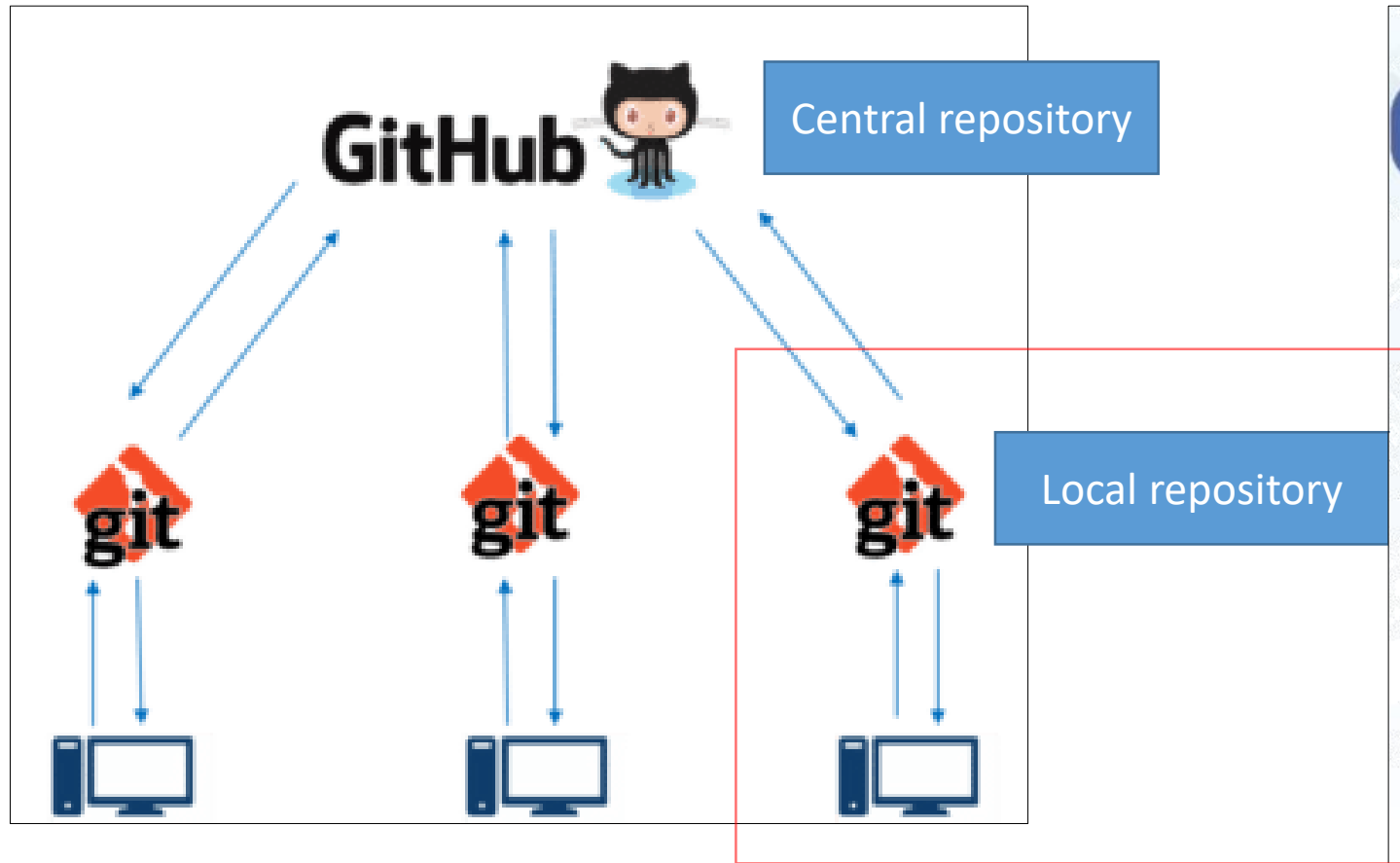
Pourquoi Git?

Git:

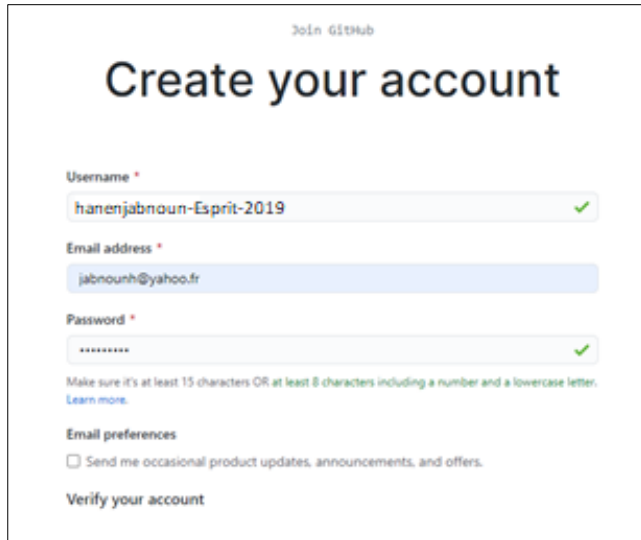
- ✓ Système de contrôle de versions décentralisé.
- ✓ SCM: source code management
- ✓ Gratuit
- ✓ Open Source
- ✓ Gérer la traçabilité et les modifications des petits aux grands projet.
- ✓ Rapide et efficace



Les commandes sur Git Bash

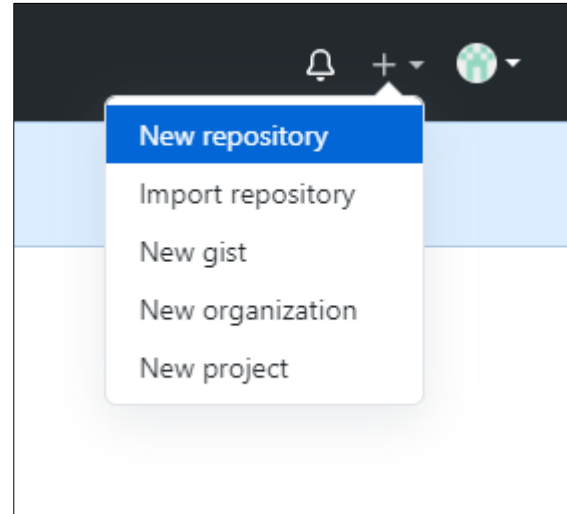


Création d'un compte sur Github

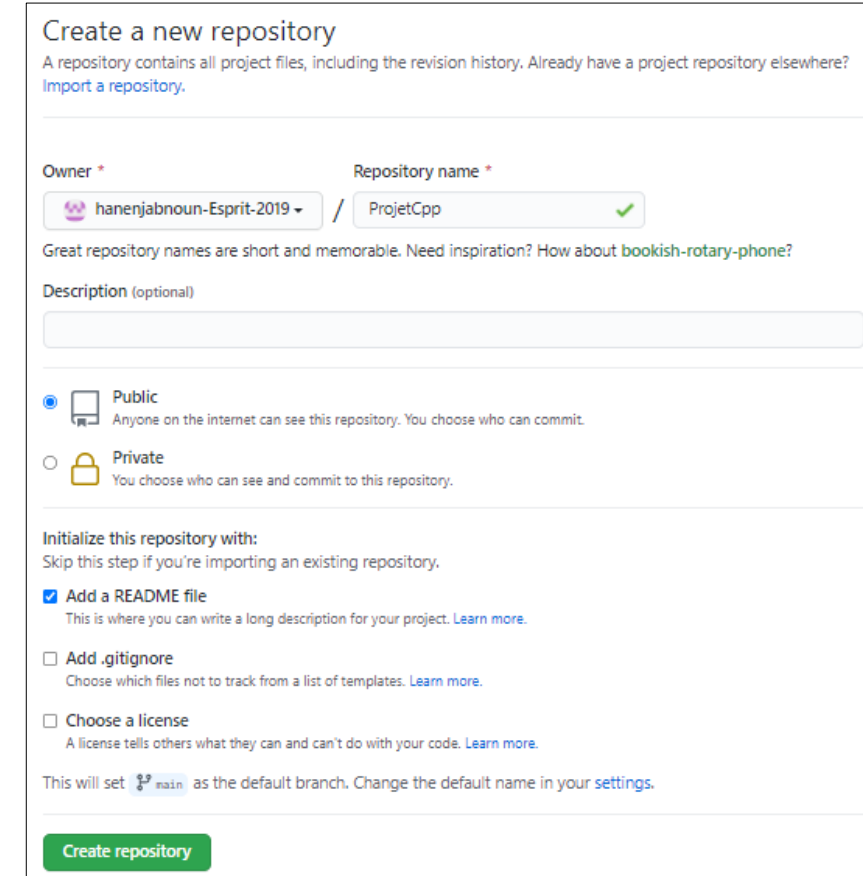


The screenshot shows the 'Create your account' page on GitHub. It includes fields for Username (hanenjabnoun-Esprit-2019), Email address (jabnounh@yahoo.fr), and Password. There are checkboxes for email preferences and a 'Verify your account' link.

Créer un compte sur Github



Cliquer sur New repository
et créer le dépôt distant
NB: Un seul membre de
l'équipe va créer le repo. Et
inviter ses collègues



The screenshot shows the 'Create a new repository' page on GitHub. It includes fields for Owner (hanenjabnoun-Esprit-2019) and Repository name (ProjetCpp). There are checkboxes for Public (selected) and Private. There are also checkboxes for 'Add a README file', 'Add .gitignore', and 'Choose a license'. A 'Create repository' button is at the bottom.

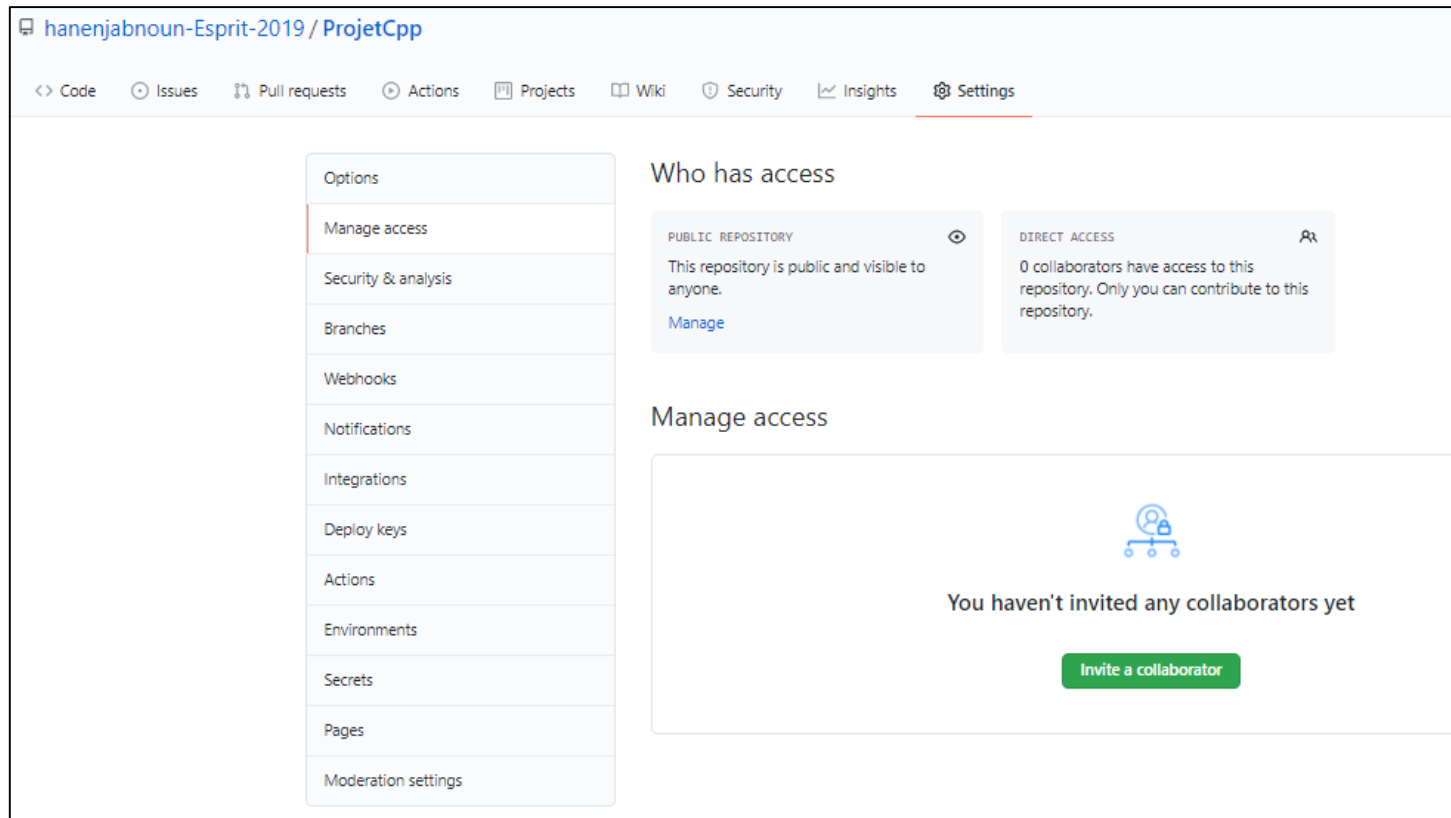
Remplir le formulaire et cliquer sur
Create repository

Création d'un compte sur Github

Ajouter le reste des membres de l'équipe:

NB: Chacun doit avoir un compte sur Github

Dans le repo que vous venez de créer aller sur **Settings** → **Manage access** → **invite collaborators**



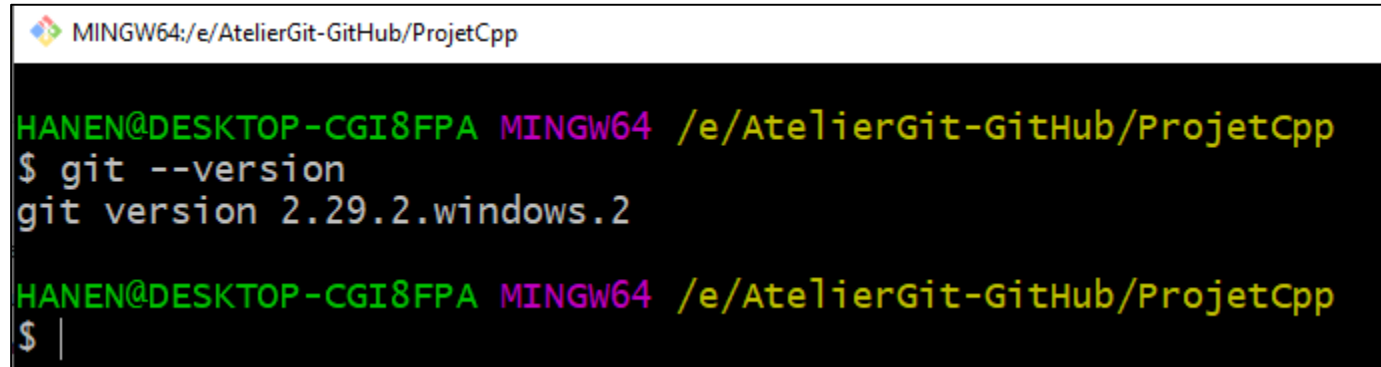
Installation du git

Télécharger et installer git:

<https://git-scm.com/download/win>

Créer un dossier « ProjetCpp »

Cliquer droit et choisir « **Git Bash here** »



```
MINGW64:/e/AtelierGit-GitHub/ProjetCpp  
  
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp  
$ git --version  
git version 2.29.2.windows.2  
  
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp  
$ |
```

git --version: donne la version du git installé sur la machine


Les commandes sur Git Bash (init)

Création du dépôt local:

git init : initialiser le dépôt local

Ou bien **git clone** : qui permet de télécharger et extraire le dépôt distant (sur github)

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp
$ git init
Initialized empty Git repository in E:/AtelierGit-GitHub/ProjetCpp/.git/
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
```

Ce PC > Disque local (E:) > AtelierGit-GitHub > ProjetCpp		
Nom	Modifié le	Type
 .git	26/06/2021 22:46	Dossier de fichiers

master: c'est la branche principale

Une branche: un pointeur léger et déplaçable vers un de ces commits.

La branche par défaut dans Git s'appelle **master**.

Les commandes sur Git Bash (config)

Configuration du git:

Spécifier un nom d'utilisateur et l'adresse mail (avec laquelle vous avez créer un compte sur github)

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp  
$ git config --global user.name "Hanen"  
  
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp  
$ git config --global user.email "hanene.jabnoun@esprit.tn"
```

`git config --global user.name "nom":`

`git config --global user.email "email":`

Les commandes sur Git Bash (status)

Création du dépôt local:

`git status`: visualiser le statuts des fichiers sur la branche courante

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp
$ git init
Initialized empty Git repository in E:/AtelierGit-GitHub/ProjetCpp/.git/

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git config --global user.name "Hanen"

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git config --global user.email "hanene.jabnoun@esprit.tn"

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git status
On branch master

No commits yet

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

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
```


Les commandes sur Git Bash (add, commit)

Le commit:

`git add <nom-fichier>` : ajouter les changements sur un fichier à l'index

`git add -A`: ajouter tous les changements à l'index

`git commit -m <message>`: enregistrer les modifications (dernière image du projet) dans le dépôt local

Dans le dossier du projet (dans lequel vous avez initialisé git) créer un nouveau fichier « main.cpp »

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  main.cpp

nothing added to commit but untracked files present (use "git add" to track)
```

Les commandes sur Git Bash (add, commit)

Le commit:

On va ajouter ces
modifications à l'index
Puis faire le commit pour
enregistrer ces modifications

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git add main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git commit -m "First commit main.cpp"
[master (root-commit) c349720] First commit main.cpp
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 main.cpp


HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Les commandes sur Git Bash (add, commit)

Le commit:

Pour chaque modification sur le fichier « main.cpp », on doit faire « add » suivi par « commit »

Essayons de modifier le fichier «main.cpp »



```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6
7
8  int main()
9  {
10     cout<<"Hello world!!"<<endl;
11
12     return 0;
13 }
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

no changes added to commit (use "git add" and/or "git commit -a")

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git add main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git commit -m "Second commit main.cpp"
[master 576f3ba] Second commit main.cpp
1 file changed, 19 insertions(+)
```

Les commandes sur Git Bash (log)

Historique de modifications :

`git log`: permet d'afficher l'historique de commit sur la branche courante (par défaut master)

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git log
commit 576f3ba0152362f729d262319b2bab17075cdeae (HEAD -> master)
Author: Hanen <hanene.jabnoun@esprit.tn>
Date: Sat Jun 26 23:06:07 2021 +0100

    Second commit main.cpp

commit c349720bd941cc1a0ac07f756e1d6170945770e9
Author: Hanen <hanene.jabnoun@esprit.tn>
Date: Sat Jun 26 23:00:58 2021 +0100

    First commit main.cpp
```

Création d'une branche

`git branch`: lister toutes les branches

`git branch <nouvelle-branche>` : créer une nouvelle branche

`git checkout -b <nouvelle-branche>.` : créer une nouvelle branche

`git checkout <nom-branche>`: basculer à une autre branche

Depuis Git version 2.23::

`git switch <nom-branche>`: basculer sur une branche existante : ,

`git switch -c <nouvelle-branche>`: créer une branche et basculer dessus ; le drapeau `-c` ou `--create` signifie créer

`git switch -`. revenir sur votre branche précédemment extraite :

Gestion des branches

On va essayer de créer une nouvelle branche

➔ Chaque membre de l'équipe doit créer une branche qui porte le nom de son module (exp: GestionProduits)
Jusqu'à l'instant tous les changements sont sauvegardés sous la branche master

ls : permet de lister les fichiers du projet.

Dans git branch: le * veut dire la branche courante

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ ls
main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git branch
* master

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git branch GestionProduits

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git branch
  GestionProduits
* master

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git checkout GestionProduits
Switched to branch 'GestionProduits'

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ ls
main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ git branch
* GestionProduits
  master
```

Gestion des branches

Depuis la branche

« GestionProduits » on va
modifier le fichier « main.cpp »

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ git status
On branch GestionProduits
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

no changes added to commit (use "git add" and/or "git commit -a")

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ git add main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ git status
On branch GestionProduits
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   main.cpp

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ git commit -m "Third commit from GestionProduits sur main.cpp"
[GestionProduits 3b16495] Third commit from GestionProduits sur main.cpp
1 file changed, 1 insertion(+), 2 deletions(-)
```

Les commandes sur Git Bash (merge)

Gestion des branches

Les modifications depuis « GestionProduit » ne sont pas visibles que pour cette branche!

Comment faire pour récupérer ce travail dans une autre branche (« master par défaut »)

0- toutes les modifications doivent être sauvegardées (commit)

1- basculer vers la branche destinataires (master dans notre cas)

2- faire un merge vers la branche courante

git merge <branche-source>: récupérer le travail depuis la source et l'enregistrer la branche courante.

Les commandes sur Git Bash (merge)

Gestion des branches

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ cat main.cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    cout<<"Hello world!!"<<endl;
    cout<<"Atelier Git/Github"<<endl;

    return 0;
}
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ git checkout master
Switched to branch 'master'
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ cat main.cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    cout<<"Hello world!!"<<endl;

    return 0;
}
```

Les commandes sur Git Bash (merge)

Gestion des branches

Récupérer le travail de « GestionProduits » dans « master »

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git merge GestionProduits
Updating 576f3ba..0366eab
Fast-forward
 main.cpp | 11 ++-----
 1 file changed, 2 insertions(+), 9 deletions(-)
```

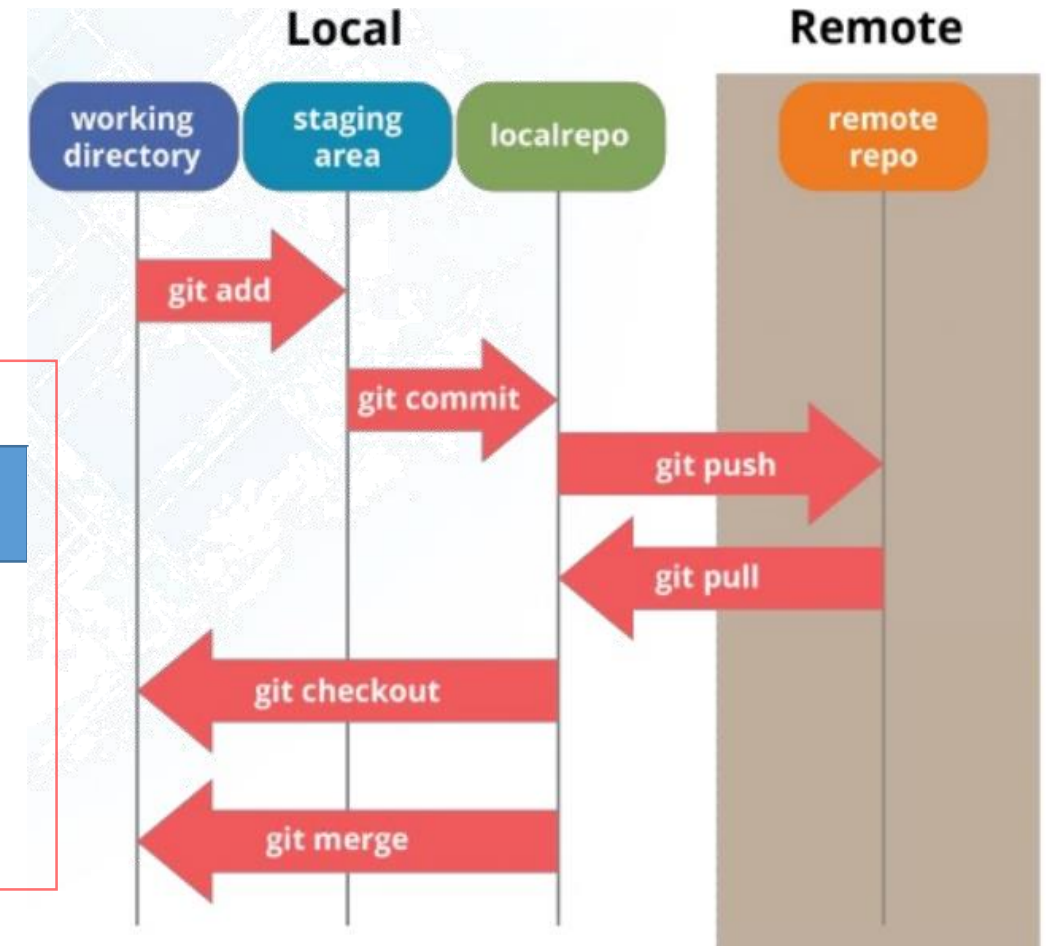
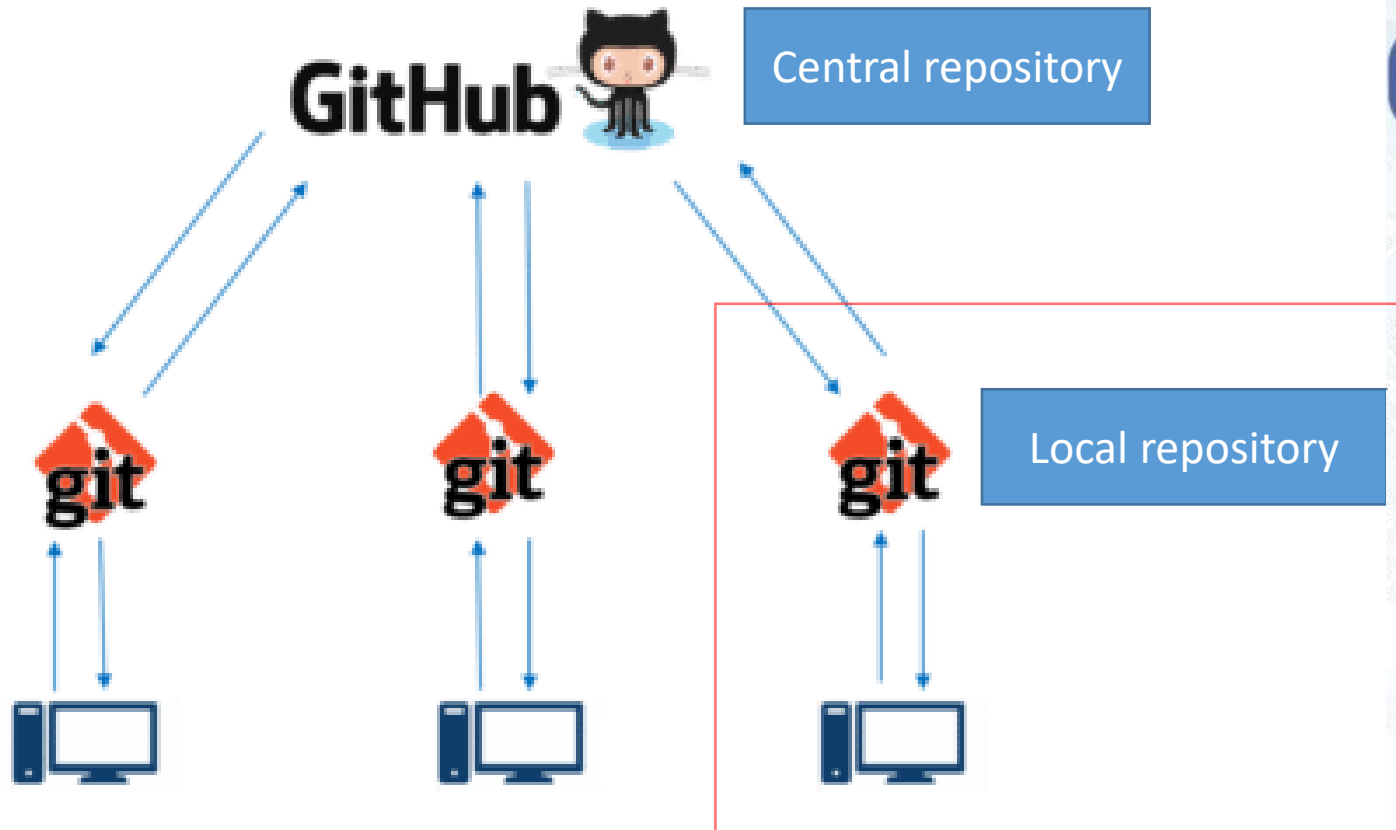
```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ cat main.cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    cout<<"Hello world!!"<<endl;
    cout<<"Atelier Git/Github"<<endl;

    return 0;
}
```

Les commandes sur Git Bash



Les commandes sur Git Bash (push)

Travail d'équipe:

➔ Tous les travaux doivent être centralisés ➔ dépôt distant sur GitHub

`git remote add oringin <url de dépôt distant>`: synchroniser les deux dépôts distant et local

`git push origin <nom-branche>`: pousser la dernière version du travail depuis le dépôt local vers le dépôt distant (de toute l'équipe)

Recommandations:

- Pas de push sur le master
- push depuis la branche personnelle (ici GestionProduits)

Source: Tutotial git/github @ edurica

Les commandes sur Git Bash (push)

Travail d'équipe:

Récupérer URL dépôt distant:

The screenshot shows the GitHub interface for the repository 'hanenjabnoun-Esprit-2019 / ProjetCpp'. The 'Code' button is highlighted with a red box. A dropdown menu is open, showing options to clone the repository. The 'HTTPS' option is circled in red, and the corresponding URL 'https://github.com/hanenjabnoun-Esprit-2019' is highlighted with a red box. The repository's README.md file is also visible, showing the title 'ProjetCpp'.

Les commandes sur Git Bash (push)

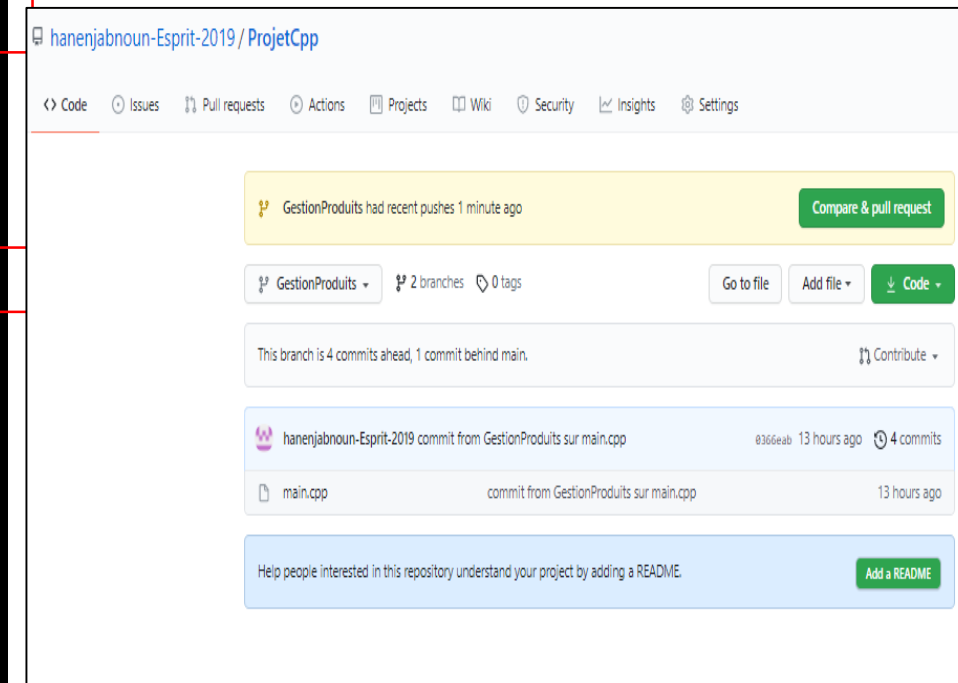
Travail d'équipe:

Pousser le travail dans le dépôt distant (Github)

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git remote add origin https://github.com/hanenjourn-Esprit-2019/ProjetCpp.git

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (master)
$ git checkout GestionProduits
Switched to branch 'GestionProduits'

HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCpp (GestionProduits)
$ git push origin GestionProduits
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (12/12), 1.03 KiB | 75.00 KiB/s, done.
Total 12 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'GestionProduits' on GitHub by visiting:
remote:   https://github.com/hanenjourn-Esprit-2019/ProjetCpp/pull/new/Gest
ionProduits
remote:
To https://github.com/hanenjourn-Esprit-2019/ProjetCpp.git
```



Les commandes sur Git Bash (pull)

Travail d'équipe:

Récupérer le travail depuis le dépôt distant (Github)

Exemple :

Ali et Mohamed deux développeurs travaillent sur le même projet

Ali a crée le dépôt distant et il a invité Mohamed comme un collaborateur sur Github.

Ali a développer une partie et il a poussé le travail sur Github.

Mohamed doit récupérer le travail sur son dépôt local

Comment faire??

`git init` : initialiser le dépôt local

`git remote add origin <URL-dépôt distant>`: synchroniser les deux dépôts local et distant

Ou bien en une seule commande:

`git clone <URL-depot distant>` : cloner le dépôt distant dans le dépôt local

Les commandes sur Git Bash (pull)

Travail d'équipe:

Depuis un autre dossier, on va se comporter comme un autre collaborateur

`git init` : initialiser le dépôt local

`git remote add origin <URL-dépôt distant>` : synchroniser les deux dépôts local et distant

`git pull origin <nom-branche>` : récupérer la dernière version du travail depuis le dépôt distant sous la branche « non-
branche »

Ou bien en une seule commande:

`git clone <URL-depot distant>` : cloner le dépôt distant dans le dépôt local

Les commandes sur Git Bash (pull)

Travail d'équipe:

MINGW64:/e/AtelierGit-GitHub/ProjetCPP-Hanen2

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCPP-Hanen2
$ git init
Initialized empty Git repository in E:/AtelierGit-GitHub/ProjetCPP-Hanen2/.git/
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCPP-Hanen2 (master)
$ git config --global user.name "HanenJ
abnoun"
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/Atelie
rGit-GitHub/ProjetCPP-Hanen2 (master)
$ git config --global user.email "jabnounhanen@gmail.com"
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCPP-Hanen2 (master)
$ git remote add origin https://github.com/hanenjabnoun-Esprit-2019/ProjetCpp.git
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCPP-Hanen2 (master)
$ git pull origin GestionProduits
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 12 (delta 2), reused 12 (delta 2), pack-reused 0
Unpacking objects: 100% (12/12), 1.01 KiB | 2.00 KiB/s, done.
From https://github.com/hanenjabnoun-Esprit-2019/ProjetCpp
* branch          GestionProduits -> FETCH_HEAD
* [new branch]     GestionProduits -> origin/GestionProduits
```

```
HANEN@DESKTOP-CGI8FPA MINGW64 /e/AtelierGit-GitHub/ProjetCPP-Hanen2 (master)
$ ls
main.cpp
```

Dans ce cas: on a récupérer le travail poussé avec la branche GestionProduit

A découvrir

Comment git gère les commits de plusieurs branches
en parallèle ??
Est-ce qu'il y a d'autres commandes avec Git Bash ??



**COMING TOGETHER
IS A BEGINNING.**

**KEEPING TOGETHER
IS PROGRESS.**

**WORKING TOGETHER IS
SUCCESS.**