

TUTORIEL D'UTILISATION DU LOGICIEL GIT

DESCRIPTION DU THÈME

Propriétés	Description
Intitulé long	Tutoriel d'utilisation du logiciel GIT
Formation(s) concernée(s)	<input type="checkbox"/> Classes de première Sciences et technologies du management et de la gestion (STMG) <input type="checkbox"/> Terminale STMG Système d'information de gestion (SIG) <input checked="" type="checkbox"/> BTS Services Informatiques aux Organisations
Matière(s)	<input type="checkbox"/> Sciences de gestion <input type="checkbox"/> SIG <input checked="" type="checkbox"/> Bloc 1 – Support et mise à disposition de services informatiques <input type="checkbox"/> Bloc 2 SISR – Administration des systèmes et des réseaux <input type="checkbox"/> Bloc 3 SLAM – Cybersécurité des services informatiques
Présentation	Ce document apporte la compréhension et l'utilisation des commandes GIT
Savoirs	Gestion de version
Compétences	Pour certains types de ressources : labo, exolab
Transversalité	SLAM/SISR
Prérequis	
Outils	git
Mots-clés	git, tutoriel, utilisation, commandes
Durée	Indicative et non obligatoire
Auteur·e·s	David ROUMANET
Version	v 9
Date de publication	15 Septembre 2020

SOMMAIRE

A Création d'un projet avec git.....	5
1 Objectifs.....	5
2 Compréhension du fonctionnement.....	5
2.1 Fonctionnement.....	5
2.2 Schéma de flux vers un serveur distant.....	5
2.3 Schéma de flux depuis un serveur distant.....	6
B Mise en pratique.....	6
1 Partie Anna.....	7
1.1 Préparation des fichiers.....	7
1.2 Préparation base locale GIT.....	7
1.3 Modifier les fichiers.....	8
1.4 Enregistrement des changements.....	9
1.5 Vérification des enregistrements.....	9
2 Partie Bob.....	11
2.1 Créer le projet (clonage).....	11
2.2 Mettre à jour le projet local.....	11
2.3 Modifier le projet.....	11
2.4 Évolutions du projet.....	12
C Annexes.....	13
1 Création projet github.....	13

A CRÉATION D'UN PROJET AVEC GIT

1 OBJECTIFS

L'installation de GIT étant terminée, nous allons prendre le temps de comprendre son fonctionnement, puis créer un projet mettant en œuvre quelques commandes.

2 COMPRÉHENSION DU FONCTIONNEMENT

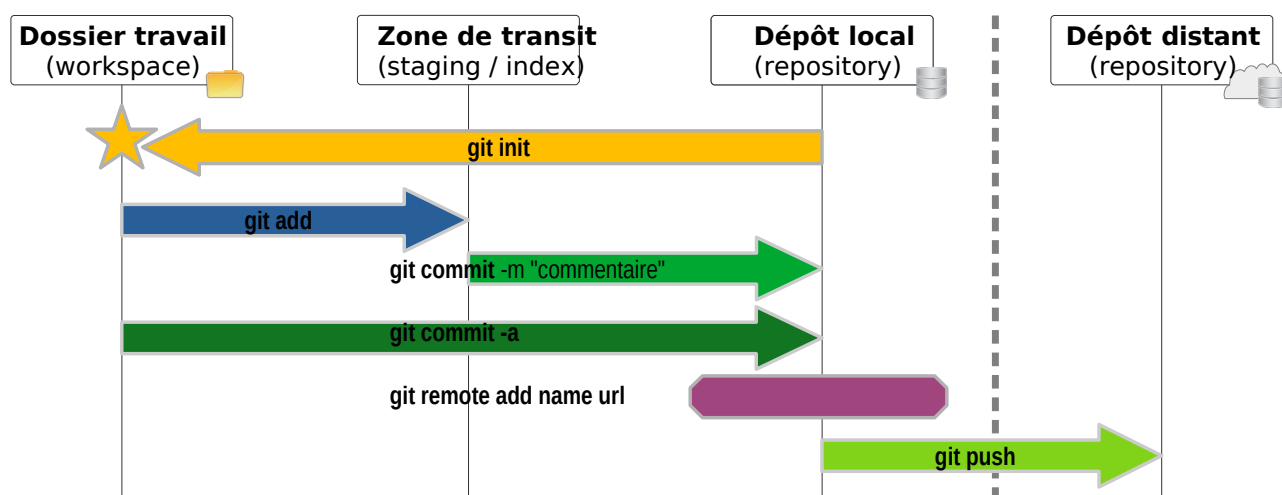
GIT est un outil capable de gérer les versions et les changements dans un projet, sur les fichiers et dossiers déclarés.

2.1 Fonctionnement

Pour cela, il utilise sa propre base de données, qu'il inclut dans les fichiers à synchroniser, dans une zone de transit. Cependant, GIT doit permettre à un développeur hors connexion de continuer à travailler sur le projet sans devoir accéder au réseau : GIT propose donc une étape supplémentaire de dépôt local, qui permet de gérer les changements en stockant les modifications localement. Une fois la connexion au serveur disponible, il suffit de synchroniser les dépôts (local et distant).

2.2 Schéma de flux vers un serveur distant

Lorsqu'un utilisateur a un projet sur son disque local, le schéma est le suivant :

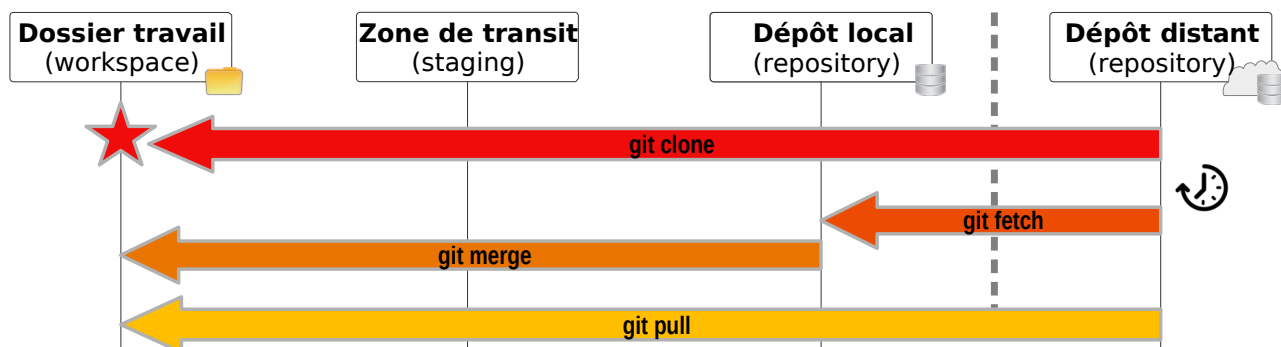


Voici l'ordre des étapes :

1. La commande **git init** permet de générer une base locale dans le répertoire de travail. La base est dans le répertoire caché **.git**.
2. La commande **git add** permet d'ajouter les fichiers et dossiers à synchroniser.
3. Lorsque les fichiers à synchroniser sont modifiés, il faut enregistrer une version, en utilisant la commande **git commit**.
4. La commande **git commit** avec l'option **-a** ajoute l'ensemble des fichiers et dossiers du répertoire de travail à la synchronisation et gère les modifications de la base de données.
5. Pour pouvoir envoyer le projet sur un serveur, il faut utiliser la commande de configuration **git remote add** qui associe un label à l'URL du projet hébergé.
6. L'envoi du projet se fait avec la commande **git push**.

2.3 Schéma de flux depuis un serveur distant

Lorsqu'un utilisateur veut récupérer un projet depuis Internet, le schéma est différent :

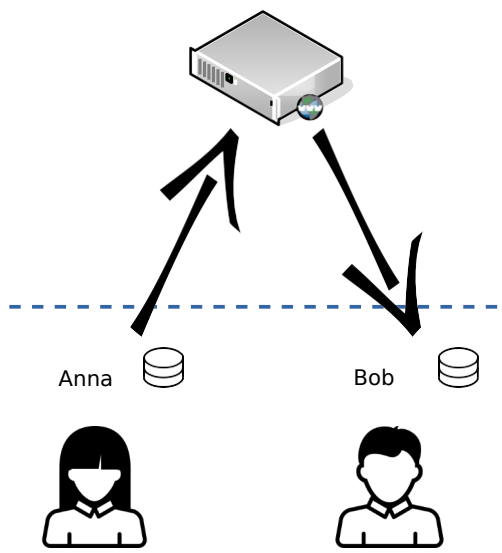


Les étapes sont les suivantes :

1. Il faut se placer dans le répertoire local de travail. La base sera téléchargée en même temps que les fichiers, avec la commande **git clone**.
2. Avant de commencer à travailler, un codeur doit toujours récupérer les dernières mises à jour du projet, par la commande **git fetch**.
3. La commande **git merge** tente de fusionner les différentes versions, pour ne pas écraser de modifications (du développeur local ou du projet distant).
4. La commande **git pull** effectue les deux commandes précédentes, comme une sorte de macro.

B MISE EN PRATIQUE

L'application de GIT consiste à synchroniser le projet d'Anna que vous créerez dans un répertoire, sur un serveur GIT, puis cloner ce projet vers le répertoire de Bob.



Le serveur GIT sera celui de votre choix : [github](#), [gitlab](#), [tuleap-campus](#)... l'important sera l'URL au format HTTPS du projet créé.

1 PARTIE ANNA

1.1 Préparation des fichiers

Anna souhaite donc créer un projet : dans son répertoire de travail où elle dépose tous ses projets, elle crée un répertoire Anna_Test.

```
mkdir Anna_Test
```

Pour gagner du temps, vous pouvez créer le répertoire de Bob.



```
mkdir Bob_Test
```

Dans le répertoire Anna_Test, elle va créer quelques fichiers (principalement de type texte) :

```
cd Anna_Test
copy NUL readme.md
copy NUL index.html
copy NUL index.js
```

Sous Linux, remplacez les commandes « copy NUL nomfichier » par « touch nomfichier ».



Les fichiers sont à 0 octet, ce qui signifie qu'ils ne contiennent aucun caractère.

1.2 Préparation base locale GIT

Anna veut avoir un historique de son projet. Pour le moment, elle n'envisage pas encore de le partager sur Internet, mais veut bénéficier de l'enregistrement des versions.

```
git init
```

La base étant créée, il faut maintenant y ajouter les fichiers à surveiller :

```
git add readme.md
git add index.html index.js
```

Elle vérifie que les fichiers ont bien été ajoutés :

```
git ls-files
```

```
D:\WorkSpaces\Anna_Test>git ls-files index.html
index.js readme.md
```



1.3 Modifier les fichiers

Éditez les trois fichiers en plaçant du code Emmet dans `readme.md`, du code HTML dans `index.html` et un peu de JavaScript dans `index.js`

readme.md

```
#Projet Anna
Voici le projet Anna_Test.
Il permet de tester le fonctionnement de git.

## Objectifs
Cette exploration doit permettre de réviser
- [x] Markdown
- [x] HTML
- [x] JavaScript
- [ ] Emmet

Mais également découvrir le fonctionnement de git.


```

Anna peut jeter un œil sur les fichiers modifiés, en utilisant la commande suivante :

```
git diff --name-only
```

Finissons l'édition des fichiers... la commande Emmet « **html:5** » peut faire gagner du temps sur le fichier ci-dessous :

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <H1>Projet d'Anna</H1>
</body>
</html>
```

Anna peut également vérifier le contenu des changements dans les fichiers, par la commande « `git diff` » sans option... testez-la !

Nous ne modifions pas le fichier `index.js` pour le moment.

1.4 Enregistrement des changements

Anna décide qu'elle a suffisamment travaillé et veut enregistrer son projet dans sa base locale.

GIT impose de mettre un commentaire (option -m) lors de la validation d'une version, cela permet de savoir la raison de cet enregistrement. L'option -a synchronise les fichiers ajoutés avec git add.

```
git commit -am "initialisation du projet sans JavaScript"
```

```
[master (root-commit) b191921] initialisation du projet sans JavaScript
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html
create mode 100644 index.js
create mode 100644 readme.md
```

Lors de l'enregistrement, GIT affecte un code d'index unique (ici b191921) et crée les fichiers dans la base.

Le mode de création doit évoquer les permissions Unix : 644 pour "Write-Read-Read"

1.5 Vérification des enregistrements

GIT dispose d'une journalisation des "commit". Anna vérifie que son enregistrement est conforme avec la commande suivante :

```
git log
```

```
commit b191921de70daf9adf51c6e0f7f37f456bdf0ea7 (HEAD -> master)
Author: anna <anna@exemple.fr>
Date:   Sat Aug 29 12:55:07 2020 +0200

    initialisation du projet sans JavaScript
```

Notez le texte dans la parenthèse, nous reparlerons de 'master' plus tard, au sujet des branches.



Finalement, Anna décide de partager ce projet sur Internet, via un serveur GIT. Elle doit donc commencer par créer le projet sur un serveur (voir [annexe 1](#)), pour récupérer un lien HTTPS.

Ce lien sera utilisé avec GIT. La commande git remote add accepte deux paramètres :

- **origin** est le nom de la connexion vers le serveur (une sorte d'alias)
- **https://** est le lien du serveur

```
git remote add origin https://github.com/#####/Anna_Test.git
```

Anna pourrait synchroniser son projet sur plusieurs serveurs GIT en ajoutant plusieurs des liens (et un no



Enfin, Anna envoie son projet avec la commande git push. Le premier paramètre est le nom du serveur (origin), le deuxième est la branche (ici, on reste sur la branche principale : master)

```
git push origin master
```

```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 258 bytes | 129.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/#####/Anna_Test.git
* [new branch]      master -> master
```


2 PARTIE BOB

Bob découvre le projet d'Anna sur le serveur GIT (si projet public), ou bien Anna lui envoie une invitation à participer à son projet (si projet privé).

2.1 Créer le projet (clonage)

Le travail de Bob est de récupérer le travail d'Anna avant de travailler dessus. Il doit donc se placer dans le répertoire de travail (workspace) de son choix (pour cette exploration, nous avons choisi 'Bob_Test') et va cloner le projet depuis le serveur Internet :

```
git clone https://github.com/roumanet/Anna_Test
```

```
Cloning into 'Anna_Test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), 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), 238 bytes | 2.00 KiB/s, done.
```

Désormais, il y a le projet 'Anna_Test' à l'intérieur du répertoire 'Bob_Test'.

2.2 Mettre à jour le projet local

Désormais, Bob peut travailler sur le même projet qu'Anna... mais s'il ne se connecte pas pendant une longue période, le projet sur Internet peut avancer et Bob pourrait travailler sur des fichiers anciens.

Il est donc recommandé de lancer une commande de mise à jour à chaque fois que vous voulez travailler sur un projet. Bob lance donc la commande

```
git pull
```

```
Already up to date.
```

2.3 Modifier le projet

Le projet est à jour, Bob peut modifier les fichiers. Anna, lui a demandé un petit script JavaScript pour la page index.html : il va donc modifier les deux fichiers suivants :

index.js

```
// Original Code By Webdevtrick ( https://webdevtrick.com )
function showTime(){
    var date = new Date();
    var h = date.getHours();
    var m = date.getMinutes();
    var s = date.getSeconds();

    h = (h < 10) ? "0" + h : h;
    m = (m < 10) ? "0" + m : m;
    s = (s < 10) ? "0" + s : s;

    var time = h + ":" + m + ":" + s + " " + session;
    document.getElementById("DigitalCLOCK").innerText = time;
    document.getElementById("DigitalCLOCK").textContent = time;

    setTimeout(showTime, 1000);
}
showTime();
```

Index.html

```
<body>
  <div id="DigitalCLOCK" class="clock" onload="showTime()"></div>
  <script src="index.js"></script>
</body>
```

Ce script doit afficher l'heure en temps réel sur la page web.

N'oubliez pas d'effacer la ligne contenant le titre <h1> !



Bob va utiliser les commandes vues dans la partie d'Anna pour [enregistrer les changements](#).

Le résultat devrait être celui-ci :

```
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 737 bytes | 737.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/droumanet/Anna_Test
95b139b..cf5c339 master -> master
```

Vous pouvez également vérifier directement sur le serveur GIT.

2.4 Évolutions du projet

En écrasant la balise <h1>, Bob est intervenu sur le code d'Anna : lorsqu'elle se [synchronisera](#) sur le serveur, elle devra insérer cette ligne à nouveau.

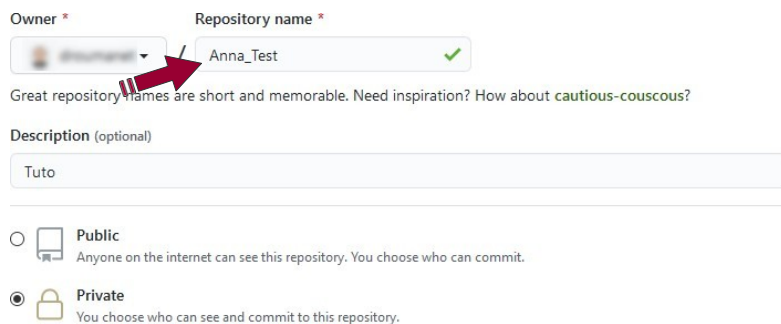
L'idéal serait que Bob fasse ses modifications de son côté, pendant qu'Anna travaille sur la mise en forme du site.

C'est ici que la notion de branche débute, mais ce sera l'objet du prochain tutoriel.

C ANNEXES

1 CRÉATION PROJET GITHUB

Une fois vos identifiants saisis sur le site, vous devez créer un nouveau "repository".



Owner * Repository name *

Anna_Test ✓

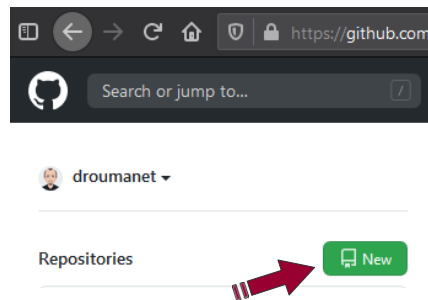
Great repository names are short and memorable. Need inspiration? How about [cautious-couscous](#)?

Description (optional)

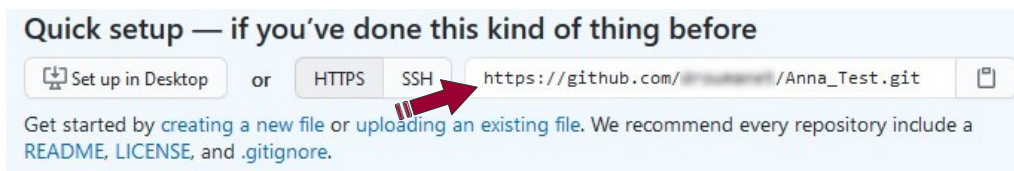
Tuto

☐ 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.



Une fois validé, vous pourrez utiliser le lien nécessaire pour GIT :



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

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) https://github.com/droumanet/Anna_Test.git

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