

PRISE EN MAIN RAPIDE DE GIT

P/2 Contexte

P/3 1. Gérer les versions de ses fichiers

P/4 2. git init

P/5 3. git add

P/6 4. git status

P/7 5. git commit

P/8 6. git log

P/9 7. git commit -a

P/11 8. git checkout

P/12 9. Quand commiter ?

P/13 10. Travailler avec un dépôt distant

P/15 11. git clone

P/17 12. Autres commandes

P/18 13. En mode graphique ?

P/19 14. Le conseil du jour

P/20 15. En résumé

P/21 Crédits

Vidéo

[https://www.youtube.com/watch?](https://www.youtube.com/watch?v=rP3T0Ee6pLU)

[v=rP3T0Ee6pLU](https://www.youtube.com/watch?v=rP3T0Ee6pLU)

Contexte



Ce document propose de manipuler Git au travers d'exemples simples pour en saisir l'intérêt et le fonctionnement de base. Il résume l'utilisation des principales commandes.

Il est organisé en 2 parties :

- Gérer les versions de ses fichiers en local
- Travailler avec un dépôt distant

Une documentation très complète de Git se trouve ici :

<https://git-scm.com/book/fr/v2/>

● Prérequis

Vous savez ouvrir un terminal et utiliser la ligne de commande.

Git est déjà installé sur votre machine, sinon c'est ici :

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

Profitez-en pour vérifier votre version de Git avec la commande :

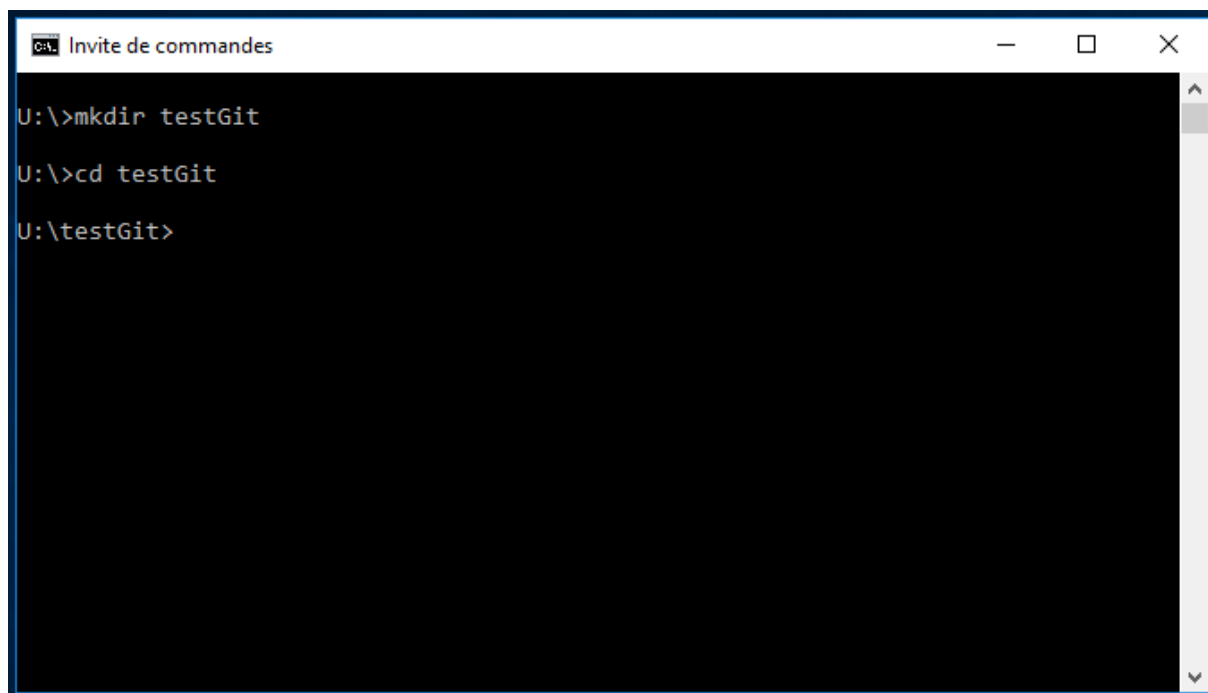
git --version (à lancer dans un terminal)

et si besoin vous mettre à jour.

1. Gérer les versions de ses fichiers

Nous allons créer un fichier contenant des lignes de code, le « versionner » et le modifier. Nous verrons également comment revenir facilement à une version précédente.

Lancez un terminal si ce n'est pas déjà fait, créez un nouveau répertoire et placez-vous dedans.



```
U:\>mkdir testGit
U:\>cd testGit
U:\testGit>
```

2. git init

La première chose à faire va être d'indiquer à Git que l'on souhaite qu'il suive notre travail dans ce répertoire. Nous allons donc créer un dépôt git local avec la commande :

git init

```
U:\testGit>git init
Initialized empty Git repository in U:/testGit/.git/

U:\testGit>
```

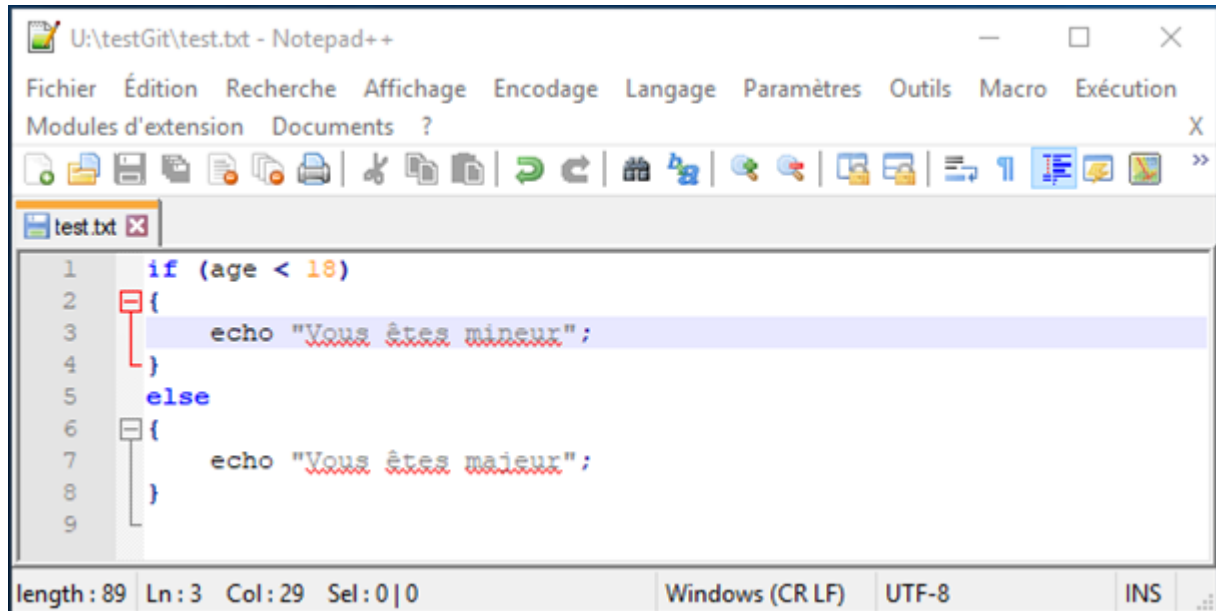


Remarque

On peut voir qu'il a créé un sous-répertoire caché (**.git**) dans lequel il va enregistrer ses fichiers de travail et les versions de nos fichiers. Nous n'interviendrons pas dans ce répertoire et laisserons à Git le soin de gérer son contenu.

3. git add

Créez maintenant un fichier « test.txt » avec votre éditeur préféré comme dans l'exemple ci-dessous et enregistrez.



```
1  if (age < 18)
2  {
3      echo "Vous êtes mineur";
4  }
5  else
6  {
7      echo "Vous êtes majeur";
8  }
9
```

Capturez le fichier pour le suivi de version (on dit aussi « indexé » ou « staged »), avec la commande :

git add test.txt

Il est possible d'utiliser les caractères joker pour ajouter plusieurs fichiers en même temps, exemple :

git add *.txt

Ou encore tout le contenu du répertoire avec la commande

git add .



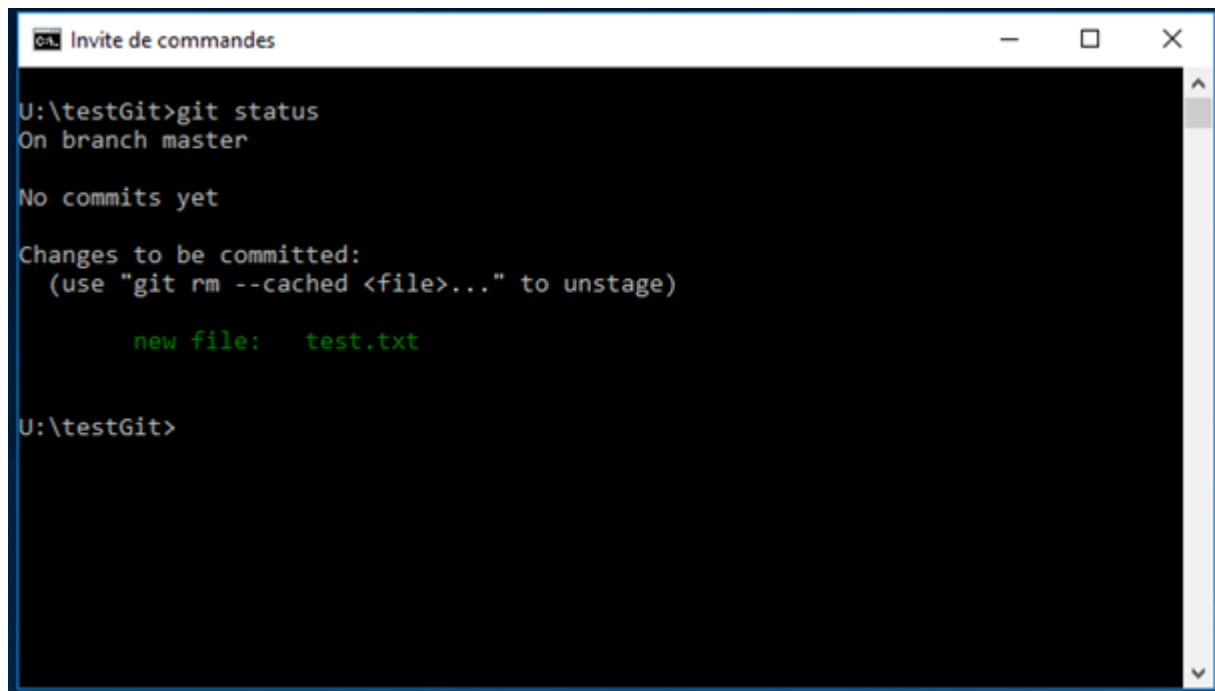
Remarque

Notez qu'une copie du fichier est immédiatement mise de côté pour être incluse dans la prochaine version. C'est pourquoi on attend en général le dernier moment pour faire les git add et on exécute juste après la commande qui va « versionner » (on en parle plus loin).

4. git status

Il est possible de consulter les fichiers concernés avec la commande

git status



```
U:\testGit>git status
On branch master

No commits yet

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

        new file:   test.txt

U:\testGit>
```

test.txt est un nouveau fichier prêt à être "commité".

5. git commit

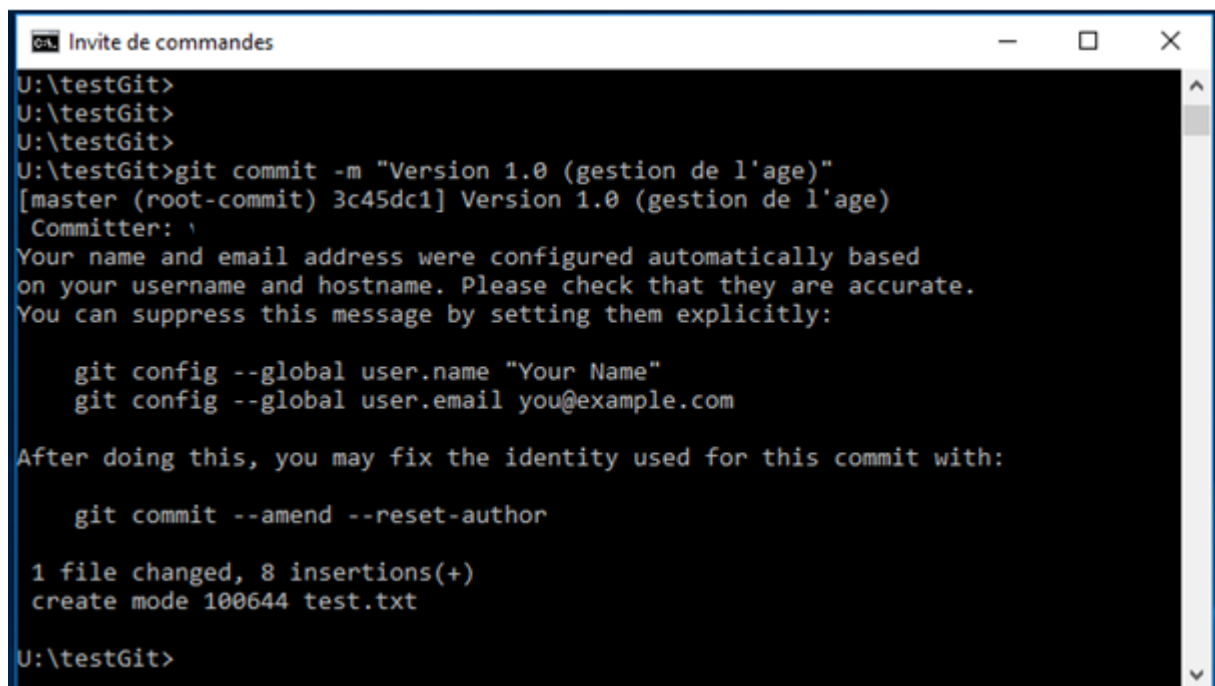
Nous allons maintenant imaginer qu'à ce stade, notre programme est stable et suffisamment abouti pour constituer la première version de notre logiciel ;-)

Il est donc temps de le « versionner » ou « commiter », avec la commande

git commit -m "Version 1.0 (gestion de l'age)"

-m ou **-message** utilise la chaîne de caractères qui suit comme libellé du commit

"Version 1.0 (gestion de l'age)" libellé **obligatoire** du commit. Utilisez un texte court, synthétique et descriptif.



```
U:\testGit>
U:\testGit>
U:\testGit>
U:\testGit>git commit -m "Version 1.0 (gestion de l'age)"
[master (root-commit) 3c45dc1] Version 1.0 (gestion de l'age)
Committer: \
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 8 insertions(+)
create mode 100644 test.txt
U:\testGit>
```

Le "commit" est maintenant effectif.



git config --global

Notez que Git nous informe qu'il n'a pas d'information sur le **"committer"** (l'auteur de l'action).

La commande **"git config"** permet de [re]définir les paramètres de votre projet (nom, mail, éditeur par défaut, ...)

Si vous ajoutez l'option **"--global"** ces changements seront appliqués à l'ensemble de vos projets.

6. git log

La commande « **git log** » permet de voir les « commits » déjà effectués



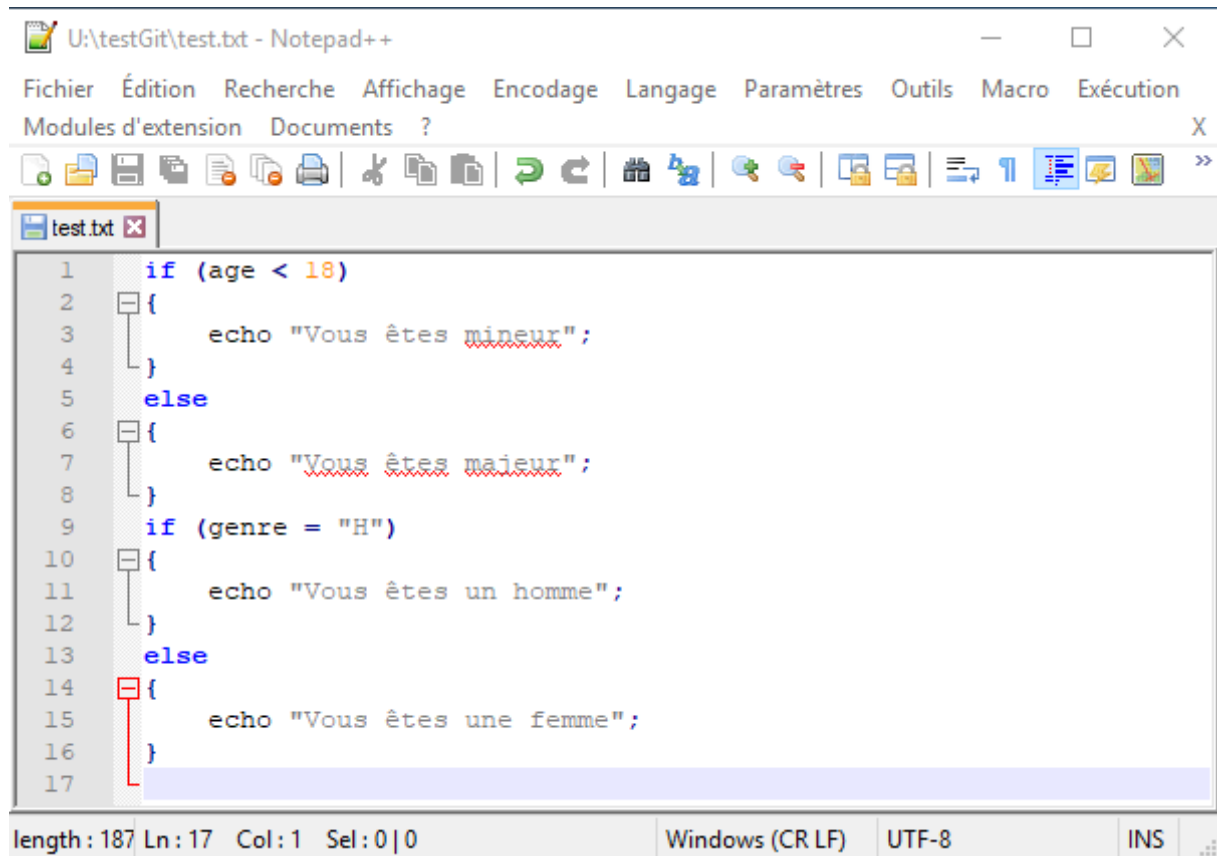
```
U:\testGit>git log
commit 3c45dc19325765a4297dc48d04277b372db78115 (HEAD -> master)
Author:
Date:   Mon Jul 8 14:41:36 2019 +0200

    Version 1.0 (gestion de l'age)

U:\testGit>
```


7. git commit -a

Continuons à travailler sur notre logiciel en ajoutant du code :



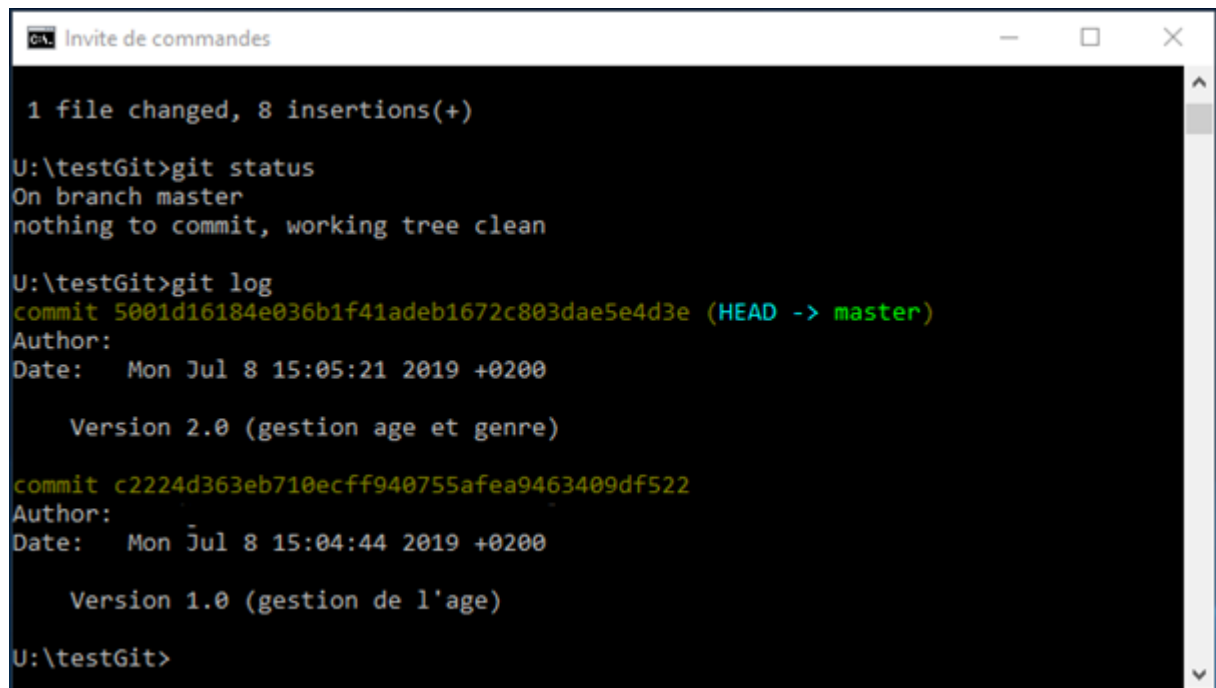
```
1  if (age < 18)
2  {
3      echo "Vous êtes mineur";
4  }
5  else
6  {
7      echo "Vous êtes majeur";
8  }
9  if (genre = "H")
10 {
11     echo "Vous êtes un homme";
12 }
13 else
14 {
15     echo "Vous êtes une femme";
16 }
17
```

Validons cette nouvelle version

git commit -am "Version 2.0 (gestion age et genre)"

-a ou -all capture les fichiers déjà suivis qui ont été modifiés ou supprimés, les nouveaux fichiers non suivis par Git ne sont pas pris en compte.

Regardons la liste des « commits »



```
1 file changed, 8 insertions(+)  
U:\testGit>git status  
On branch master  
nothing to commit, working tree clean  
  
U:\testGit>git log  
commit 5001d16184e036b1f41adeb1672c803dae5e4d3e (HEAD -> master)  
Author:  
Date:   Mon Jul 8 15:05:21 2019 +0200  
  
    Version 2.0 (gestion age et genre)  
  
commit c2224d363eb710ecff940755afea9463409df522  
Author:  
Date:   Mon Jul 8 15:04:44 2019 +0200  
  
    Version 1.0 (gestion de l'age)  
  
U:\testGit>
```

Parfait, nous avons maintenant l'historique de notre code. Vous pouvez continuer vos modifications sur le fichier test.txt, créer d'autres fichiers, les ajouter (ou pas) à la liste des fichiers à suivre et « commiter » si besoin. La liste des « commits » va ainsi s'allonger pour garder la trace de ces modifications.

8. git checkout

● Retour en arrière

Le client a changé d'avis, plus besoin de gérer le **genre** dans l'application.

Pas de problème !

Pour remettre votre code dans l'état d'un des « commits » précédents, on utilise la commande « **git checkout** » en précisant le début du numéro de commit (la valeur en hexadécimal).

git checkout c2224



```
U:\testGit>git checkout c2224
Note: checking out 'c2224'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b <new-branch-name>

HEAD is now at c2224d3 Version 1.0 (gestion de l'age)
U:\testGit>
```

Actualisez le contenu du fichier test.txt dans votre éditeur, vous constaterez que la gestion du genre n'est plus dans le code, le fichier est revenu dans l'état au moment du commit.

Vous pouvez continuer vos modifications sur le fichier et « commiter » à nouveau, ce ne sera qu'une version de plus dans la liste. Sans pour autant écraser celle avec la gestion du genre qui pourrait nous resservir si le client changeait encore d'avis !

bah oui, c'est le client ... et on est Agile, non ?

9. Quand commiter ?

Il n'y a pas de règles applicables à tous les cas, car les besoins varient selon la taille du projet, l'importance des modifications, la vitesse de production du code ou encore le nombre de développeurs.

Mais retenez qu'il est inutile de « commiter » toutes les 5 mn ou à la moindre modification de code. Une fois par jour suffit en général. « Committez » en plus lorsque vous avez terminé une fonctionnalité et « Committez » impérativement lorsque vous livrez une version.

10. Travailler avec un dépôt distant

« Versionner » sur son disque dur, c'est bien et cela vous « sauvera la vie » en cas de manipulation malencontreuse, en général la veille de la recette finale avec le client ;-)

Mais « versionner » en plus sur un dépôt distant, c'est la garantie de retrouver son code en cas de crache du disque dur (oui, comme tout le monde vous n'aviez pas fait de sauvegardes), la possibilité de récupérer facilement et rapidement le code sur une autre machine et même la possibilité de travailler à plusieurs sur le même projet. Ah ouais, carrément !

● Première étape, il nous faut un dépôt distant.



Remarque

Le plus « hype » est d'installer un « serveur Git » sur une machine connectée en permanence au web. Après avoir lu, compris et assimilé toute la documentation de Git et plusieurs excellents ouvrages de type : « comment je sécurise correctement mon serveur web perso », vous devriez y arriver. On se donne rendez-vous dans quelques semaines.

Si comme moi vous êtes pressés, il existe des sites qui font cela très bien et pour pas cher, voire gratuitement. Pour les plus connus on peut citer **GitHub** et **GitLab**.

Dans les exemples qui suivent, nous allons utiliser **GitLab** mais GitHub aurait très bien fait le « job » avec les mêmes fonctionnalités et les mêmes commandes, seule l'interface web des sites présente des différences, les options n'étant pas forcément au même endroit.

Commencer donc par vous créer un compte sur GitLab si vous voulez jouer la facilité ou ailleurs si vous savez ce que vous faites.



Remarque

Si vous avez déjà réussi à vous créer un compte sur FaceBook, Amazone, Tinder ou Instagram vous devriez y arriver sans plus d'explications. Le plus dur est de penser à noter quelque part le user, le full name, le mail et le password que vous allez fournir ;-)

A vous de jouer.

Maintenant que vous avez votre compte GitLab, Connectez-vous et cliquez sur le bouton de gestion



du compte en haut à droite :

Puis sur Paramètres/Compte et notez le **chemin d'accès**, vous en aurez besoin.

Changer le nom
d'utilisateur-ric

Chemin d'accès

https://gitlab.com/ yyy:yyy

Super, reste à pousser le dépôt local existant sur le serveur (ce qui va créer le dépôt distant).



Remarque

Dans les commandes qui suivent,
yyy.yyy est à remplacer par votre chemin d'accès
xxx est à remplacer par le nom de votre projet (testGit)

C'est parti :

git remote add origin https://gitlab.com/yyy.yyy/xxx.git

git push -u origin --all

Entrez vos identifiants quand Git vous les demande.

Allez vérifier dans l'interface web de GitLab que votre projet est bien là, notez au passage la présence d'un cadenas fermé à côté du nom du projet. Votre projet est privé, il n'est visible que de vous pour le moment. Vous en êtes le propriétaire et le seul à même d'autoriser d'autres contributeurs à en voir ou modifier le contenu, ou tout simplement le supprimer.

Faites des modifications en local, des « commits » et pousser à nouveau vers le dépôt distant avec

git push -u origin

Parfait, vous avez maintenant une copie privée externalisée du projet. A vous de la tenir à jour avec des « commits » et des « pushes » lorsque vous le jugez nécessaire.

11. git clone

● Et si je veux récupérer le projet sur un autre ordinateur ?

Pas de problème, vous n'avez peut-être pas un deuxième ordinateur sous la main mais on va faire « comme si » en clonant le projet dans un autre répertoire.

Placez-vous ailleurs que dans le répertoire parent de « testGit » et clonez le dépôt distant :

```
cd ..
```

```
mkdir temp
```

```
cd temp
```

```
git clone https://gitlab.com/yyy.yyy/xxx.git
```

```
cd xxx
```

Git a créé dans « **temp** » un répertoire « **testGit** » (c'est pour ça qu'il ne fallait pas rester dans le répertoire parent de « **testGit** ») et il y a copié tous les fichiers du projet (y compris le répertoire caché **.git**).

On peut modifier le code du nouveau répertoire et pousser les modifications sur le serveur :

Modifier le fichier « test.txt » (du nouveau répertoire « temp/testGit ») et « commiter »

```
git commit -am "Version nouveau repertoire"
```

Pousser votre modification sur le dépôt distant

```
git push -u origin
```

Retournez maintenant dans l'ancien répertoire « testGit » (pas celui qui est dans « temp »)

Et récupérez les modifications avec

```
git pull
```

Regardez le contenu de « test.txt » il contient les modifications faites dans le « test.txt » du nouveau répertoire.

Faites un « **git log** », la liste a été mis à jour avec le « commit » "Version nouveau repertoire".

Si vous ne travaillez pas toujours sur la même machine, ce n'est pas plus compliqué de tenir à jour ses fichiers.



Retenez toutefois une règle essentielle pour éviter les problèmes :

Si vous n'avez pas travaillé depuis quelques temps sur une machine, commencez **TOUJOURS** par faire un **git pull** avant toute modification de code.

Cela vous permettra de mettre à jour votre dépôt local par rapport à des modifications récentes faites par vous depuis une autre machine ou par d'autres si vous travaillez à plusieurs sur un projet.

Vous avez maintenant toutes les commandes pour gérer vos versions et les centraliser sur un dépôt.

● On a vu quoi déjà ?

- **git add + commit -m** pour capturer et versionner de nouveaux fichiers.
- **commit -am** pour versionner des fichiers modifiés (déjà suivis)
- Avec **push** vous envoyez vos « commits » sur le serveur distant
- Avec **pull** vous récupérez les « commits » du serveur distant.
- Avec **checkout** vous changez la version du code
- **git status** affiche le suivi des fichiers
- **git log** affiche la liste des « commits »

12. Autres commandes

- **D'autres commandes intéressantes :**

- **Gérer avec Git un projet en cours**

Créer d'un répertoire existant, un dépôt local et le pousser sur le serveur (créer le dépôt distant)

```
cd existing_folder
```

```
git init
```

```
git remote add origin https://gitlab.com/yyy.yyy/xxx.git
```

```
git add .
```

```
git commit -m "Initial commit"
```

```
git push -u origin master
```

- **Créer un dépôt local à partir d'un dépôt distant existant**

Cloner un dépôt

```
git clone https://gitlab.com/yyy.yyy/xxx.git
```

```
cd xxx
```

- **Créer un nouveau dépôt distant avec un dépôt local existant**

```
cd existing_repo
```

```
git remote rename origin old-origin
```

```
git remote add origin https://gitlab.com/yyy.yyy/xxx.git
```

```
git push -u origin --all
```

Il est aussi possible de créer un dépôt distant vide avec l'interface web du site puis de le cloner en local dans un répertoire. Bref, pleins de façons de faire, en fonction des cas et de vos besoins.

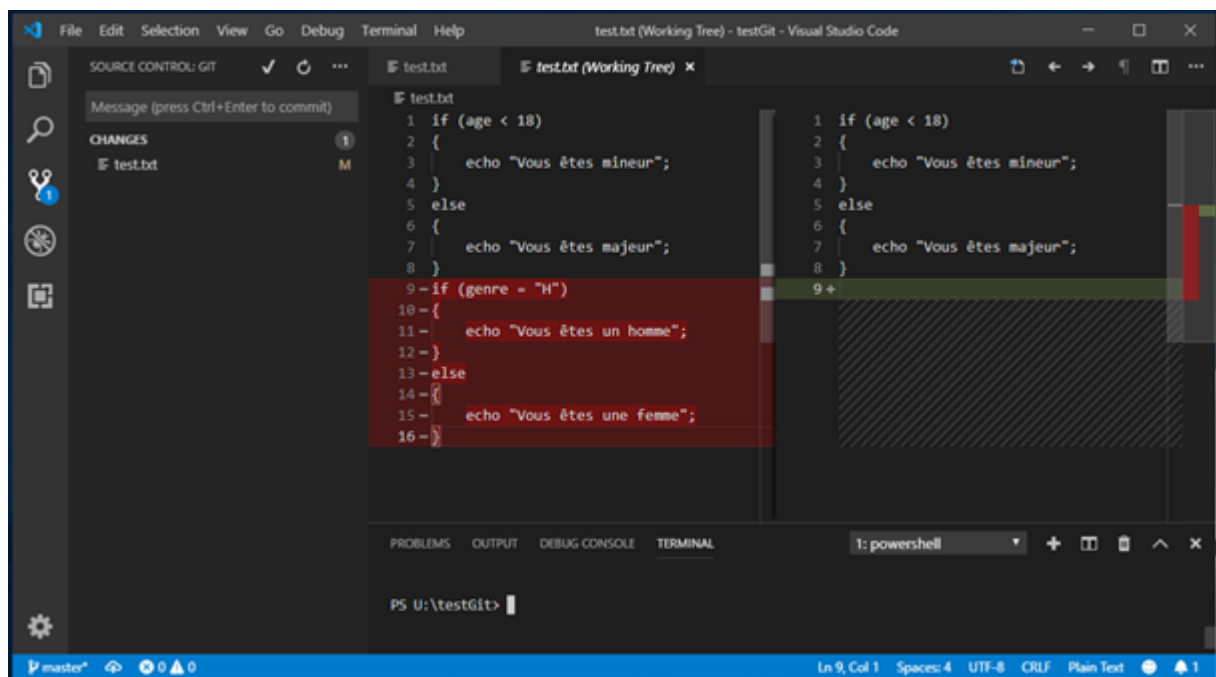
13. En mode graphique ?

Nous avons ici utilisé la ligne de commande pour piloter Git, l'avantage étant que ces commandes fonctionnent quel que soit le système d'exploitation que vous utilisez.

Sachez qu'il existe des outils graphiques pour faire la même chose avec la souris, exemple pour Windows « **GitHub Desktop** ».

La plupart des IDE actuels proposent également de gérer vos dépôts en mode graphique.

VS Code intègre la gestion des dépôts Git et permet de gérer facilement vos versions, vous ne seriez pas pardonnés de ne pas y jeter un coup d'œil, des tutoriels vidéo sur le web vous montreront en quelques minutes son fonctionnement.



Exemple de comparaison des modifications entre deux « commits »

14. Le conseil du jour

Commencez par travailler seul sur un projet pour vous familiariser avec les commandes de Git, et bien saisir ces notions de **staged**, **commit**, **indexé**, **push**, **pull**, **checkout** ...

Lorsque ces manipulations vous paraîtront simples, passez en mode multi-développeurs sur le même projet.

Il faudra **inviter/autoriser** des « **membres** » dans votre projet et avoir vu le concept des « **branches** » (notions non abordées dans ce support).

15. En résumé

● En début de projet :

Créer un nouveau dépôt dans un répertoire existant avec **git init**

● Régulièrement :

Capturer les fichiers à suivre avec **git add**

« Versionner » avec **git commit**

Pousser vers le dépôt distant les « commits » avec **git push**

● Au besoin :

Récupérer en local le contenu du dépôt distant avec **git pull**

Changer de version avec **git checkout**

git init	Initialise un nouveau dépôt local
git add [toto.txt,*.txt,.]	Capture les fichiers sélectionnés
git commit -m "libellé"	Enregistre une version des nouveaux fichiers du suivi
git commit -am "libellé"	Enregistre une version des fichiers modifiés ou supprimés du suivi
git checkout ffffff	Mets les fichiers dans l'état de la version désignée par le début de son numéro
git push -u origin master	Pousse le dépôt local vers le dépôt distant
git pull	Tire le dépôt distant vers le dépôt local
git clone https://url/chemin/projet.git	Clone (duplique) un dépôt distant
git remote add origin https://url/chemin/projet.git	Crée un dépôt distant

Crédits

© AFPA

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconques. »