08/07/2022

Gît

Suite á la demande de certains collègues ,j'ai créé ce document comme une sorte de manuel

Mon but n'est pas de me substituer au prof

Patrice André Waechter-Ebling

Notez que ce document ne traite que Windows

Je ne parlerais pas des commandes pour

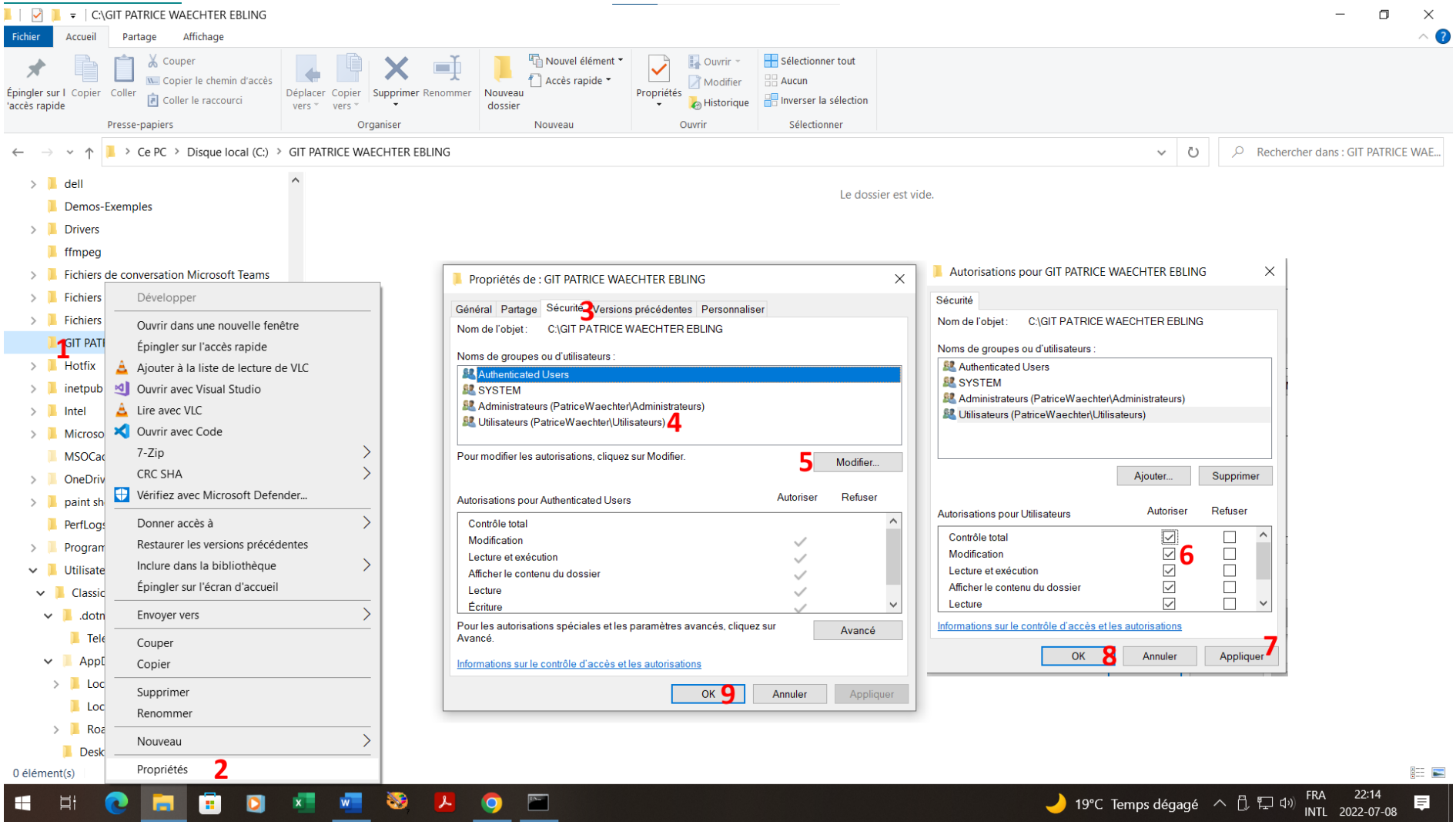
- **HTTPS**
- **SSH**

Car je ne les maitrise pas suffisamment bien

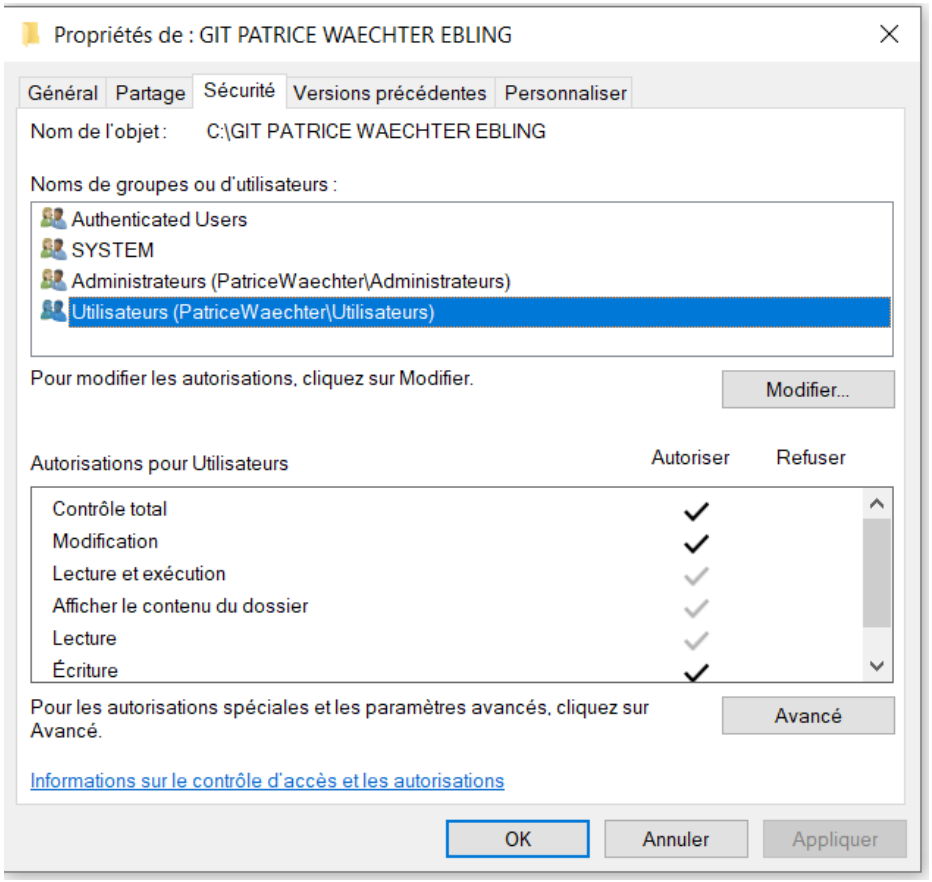
Je vous suggère de vous créer un dossier dans lequel vous déposerez vos projets à « Giter » et surtout pas dans un chemin qui se synchronise avec le cloud; vous réaliserez très vite que ce n'est pas une bonne idée

Ex : C:\GIT PATRICE

bien sûr il va falloir lui attribuer les permissions voici comment faire :



Ce qui donne comme résultat :



Paramétrage à la première utilisation de git

Personnaliser son environnement git.

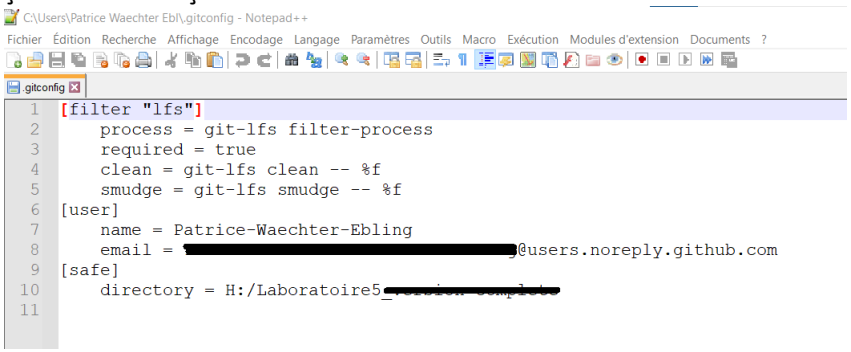
Avant de modifier un fichier config assurez-vous d'avoir créé une copie de sauvegarde avant de modifier les originaux //ça m'a sauvé des heures de debug après tout; une faute de frappe ça arrive à tout le monde

Git contient un outil appelé git config pour vous permettre de voir et modifier les variables de configuration qui contrôlent tous les aspects de l'apparence et du comportement de git.
Ces variables peuvent être stockées dans trois endroits différents :
[chemin]/etc/gitconfig : Contient les valeurs pour tous les utilisateurs et tous les dépôts du système. //Requière niveau Admin pour modifier
Fichier **~/.gitconfig** : Spécifique à votre utilisateur. Vous pouvez forcer git à lire et écrire ce fichier en passant l'option --global et cela affecte **tous** les dépôts avec lesquels vous travaillez sur ce système.
Fichier config dans le répertoire git (c'est-à-dire **.git/config**) du dépôt en cours d'utilisation : spécifique au seul dépôt en cours. Vous pouvez forcer git à lire et écrire dans ce fichier avec l'option **–local** //mais c'est en fait l'option par défaut.
Sans surprise, le répertoire courant doit être dans un dépôt git pour que cette option fonctionne correctement.
Chaque niveau surcharge le niveau précédent, donc les valeurs dans **.git/config** surchargent celles de **[chemin]/etc/gitconfig.**
Sur les systèmes Windows, git recherche le fichier .gitconfig dans le répertoire HOME (%USERPROFILE% dans l'environnement natif de Windows) qui est C:\Documents and Settings\USER ou C:\Users\USER depuis (Windows Vista) la plupart du temps, selon la version (USER devient %USERNAME% dans l'environnement de Windows).
Il recherche tout de même /etc/gitconfig, bien qu'il soit relatif à la racine qui se trouve où vous aurez décidé d'installer git sur votre Windows.

Si vous utilisez une version 2.x ou supérieure de git pour Windows, il y a aussi un fichier de configuration système à
C:\Documents and Settings\All Users\Application Data\Git\config sur Windows XP
C:\ProgramData\Git\config sur Windows, Vista Windows7,Windows8.x
%USERPROFILE%\AppData\Local\GitHubDesktop\app-3.0.1\resources\app\git\etc sous Windows10

Soyez vigilants quant la section dans laquelle vous modifiez les paramètres comme je disais plus il y a une hiérarchie de priorité :

%USERPROFILE%\.gitconfig
ça ressemble à ça comme contenu



%USERPROFILE%\AppData\Local\GitHubDesktop\app-3.0.1\resources\app\git\etc sous Windows10

Ce fichier de configuration **ne peut être modifié** qu'avec la commande `git config -f <fichier>` **en tant qu'administrateur.**
Vous pouvez voir tous vos paramétrages et d'où ils viennent en utilisant : `git config --list --show-origin`

Identité

La première chose à faire après l'installation de git est de renseigner votre nom et votre adresse de courriel.
C'est une information importante car toutes les validations dans git utilisent cette information et elle est indélébile dans toutes les validations que vous pourrez réaliser :

```
git config --global user.name "Patrice Waechter-Ebling"

git config --global user.email patrice@cegep.edu //adresse fictive bien sur
```

Cette étape n'est nécessaire qu'une fois si vous passez l'option **--global**, parce que git utilisera toujours cette information pour tout ce que votre utilisateur fera sur ce système.

Éditeur de texte

L'éditeur de texte qui sera utilisé quand git vous demande de saisir un message.
Par défaut, git utilise l'éditeur configuré au niveau système, qui est généralement Vi ou Vim.
Cependant il est possible d'en assigner un autre dans mon cas Notepad++

```
git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"
```

Nom de branche par défaut

Par défaut git crée une branche nommée `main` anciennement (`master`**)** quand vous créez un nouveau repository avec git `init`.
Depuis la version 2.28, vous pouvez définir un nom différent pour la branche initiale.
Pour définir **main** comme nom de branche par défaut
`git config --global init.defaultBranch main`

Vérification des paramètres

`git config --list`

- user.name= Patrice Waechter-Ebling
- user.email=patrice@cegep.edu //adresse fictive bien sur
- color.status=auto
- color.branch=auto
- color.interactive=auto
- color.diff=auto

Certains paramètres apparaissent plusieurs fois car git lit les mêmes paramètres depuis plusieurs fichiers git utilise la dernière valeur pour chaque paramètre unique qu'il lit.
Vous pouvez aussi vérifier la valeur effective d'un paramètre: `git config user.name`

Obtenir de l'aide

Commandes bien utiles pour l'aide

git help <commande>
git help config
git <commande> --help

De plus, si vous n'avez pas besoin de l'aide complète de la page de manuel, mais avez juste besoin d'un rappel rapide sur les options disponibles pour une commandegit, vous pouvez demander la sortie "help" plus concise avec l' option **-h** :

```
git add -h

usage:git add [<options>] [--] <pathspec>...

    -n, --dry-run         dry run
    -v, --verbose         be verbose
    -i, --interactive     interactive picking
    -p, --patch           select hunks interactively
    -e, --edit            edit current diff and apply
    -f, --force           allow adding otherwise ignored files
    -u, --update          update tracked files
    --renormalize         renormalize EOL of tracked files (implies -u)
    -N, --intent-to-add   record only the fact that the path will be added later
    -A, --all             add changes from all tracked and untracked files
    --ignore-removal      ignore paths removed in the working tree (same as --no-all)
    --refresh             don't add, only refresh the index
    --ignore-errors       just skip files which cannot be added because of errors
    --ignore-missing      check if - even missing - files are ignored in dry run
   --chmod (+|-)x         override the executable bit of the listed files
```

Démarrer un dépôt git

Vous pouvez principalement démarrer un dépôt git de deux manières.

- ✓ Vous pouvez prendre un répertoire existant et le transformer en dépôt git.
- ✓ Vous pouvez **cloner** un dépôt git existant sur un autre serveur.

Dans les deux cas, vous vous retrouvez avec un dépôt git sur votre machine locale, prêt pour y travailler.

Initialisation d'un dépôt git dans un répertoire existant

Exemple le projet **Laboratoire5_Zip_du_Prof**

```
cd C:\GIT PATRICE WAECHTER EBLING\Laboratoire5_Zip_du_Prof //dossier local
git init // vous avez un dépôt
git add *.* // des fichiers suivis
git add LICENSE
git commit -m ' Laboratoire5 Zip du Prof' // un commit initial.
```

Cloner un dépôt existant

Si vous souhaitez obtenir une copie d'un dépôt git
Par exemple, si vous voulez cloner la bibliothèque logicielle git appelée libgit2, vous pouvez le faire de la manière suivante :

```
git clone https://github.com/libgit2/libgit2
```

Ceci crée un répertoire nommé `libgit2`, initialise un répertoire `.git` à l'intérieur, récupère toutes les données de ce dépôt, et extrait une copie de travail de la dernière version.
Cette commande réalise la même chose que la précédente, mais le répertoire cible s'appelle `monlibgit2`.

```
git clone https://github.com/libgit2/libgit2 monlibgit2 //destination choisie par l'utilisateur
```

Visualiser l'historique des validations

Après avoir créé plusieurs commits ou cloné un dépôt ayant un historique de commits
Pour voir le fil des événements : **git log**

Démonstration :

1. git clone https://github.com/schacon/simplegit-progit DemoGit
2. cd c:\git patrice waechter ebling\demogit
3. git log **//voici les commandes des log les plus couramment utilisées**
   - ✓ git log –stat
   - ✓ git log --pretty=oneline
   - ✓ git log --pretty=format:"%h - %an, %ar : %s"
   - ✓ git log --pretty=format:"%h %s" –graph
   - ✓ git log --since=2.weeks
   - ✓ git log -S nom_de_fonction
4. git log –stat -- c:\git patrice waechter ebling\demogit\demogit.log**//généralement utilisé**
5. echo ' DemoGit ' > LISEZMOI **//se créer un «read.me »**
6. git status
7. git add LISEZMOI **//pour inclure dans ce qui sera validé**
8. git status
9. vim CONTRIBUTING.md
10. git status

Inspecter les modifications indexées et non indexées

1. git status
2. git reset HEAD demogit.log //pour désindexer
3. git add c:\git patrice waechter ebling\demogit\demogit.log **//pour mettre à jour ce qui sera validé**
4. git checkout -- c:\git patrice waechter ebling\demogit\demogit.log **//pour annuler les modifications dans la copie de travail**
5. git diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
6. git diff –staged**// le dernier instantané :**
7. diff --git a/LISEZMOI b/LISEZMOI
8. git add CONTRIBUTING.md
9. echo 'ligne de test' >> CONTRIBUTING.md **//nouvelle entrée**
10. git status
11. git reset HEAD CONTRIBUTING.md **//pour désindexer**
12. git add CONTRIBUTING.md **//pour mettre à jour ce qui sera validé)**
13. git checkout -- CONTRIBUTING.md **//pour annuler les modifications dans la copie de travail)**
14. git diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
15. git diff --cached **//pour visualiser ce qui a été indexé jusqu'à maintenant**

16. git commit -m "Version démo"

Passer l'étape de mise en index

Bien qu'il soit incroyablement utile de pouvoir organiser les **commits** exactement comme on l'entend, la gestion de la zone d'index est parfois plus complexe que nécessaire dans le cadre d'une utilisation normale déjà en suivi de version dans la zone d'index avant de réaliser la validation, évitant ainsi d'avoir à taper les commandes **git add :**

1. git status
2. git add CONTRIBUTING.md **//pour mettre à jour ce qui sera validé)**
3. git checkout -- CONTRIBUTING.md **//pour annuler les modifications dans la copie de travail)**
4. git commit -a -m 'Nouvel embranchement ajouté'

Notez bien que vous n'avez pas eu à lancer ***git add*** sur le fichier **CONTRIBUTING.md** avant de valider**.**

Effacer des fichiers

Pour effacer un fichier de Git, vous devez l'éliminer des fichiers en suivi de version (plus précisément, l'effacer dans la zone d'index) puis valider.
La commande git rm réalise cette action mais efface aussi ce fichier de votre copie de travail de telle sorte que vous ne le verrez pas réapparaître comme fichier non suivi en version à la prochaine validation, sinon le fichier dans le dossier de travail, apparaîtra sous la section « Modifications qui ne seront pas validées »
// non indexé dans le résultat de git status :

1. git status
2. git add/rm CONTRIBUTING.md **//pour mettre à jour ce qui sera validé)**
3. git checkout -- CONTRIBUTING.md **//pour annuler les modifications dans la copie de travail)**
4. git commit -a
5. git rm CONTRIBUTING.md**// l'effacement du fichier est indexé**
6. git reset HEAD CONTRIBUTING.md **//pour désindexer**

Façon forcée

git rm --cached LISEZMOI **//noms de fichiers ou de répertoires**
Cette commande efface tous les fichiers avec l'extension .logs présents dans le répertoire log.
git rm log/*.log // Notez bien la barre oblique inverse (\) devant *. Il est nécessaire d'échapper le caractère *
Cette commande élimine tous les fichiers se terminant par ~.
git rm *~

Déplacer des fichiers

À la différence des autres VCS, Git ne suit pas explicitement les mouvements des fichiers.

```
git mv LISEZMOI.txt LISEZMOI
git status
git reset HEAD LISEZMOI
```

=

```
mv LISEZMOI.txt LISEZMOI
git rm LISEZMOI.txt
git add LISEZMOI
```

Annuler des actions

À tout moment, vous pouvez désirer annuler une des dernières actions.

Il faut être très attentif car certaines de ces annulations sont définitives (elles ne peuvent pas être elles-mêmes annulées

Une des annulations les plus communes apparaît lorsqu'on valide une modification trop tôt en oubliant d'ajouter certains fichiers, ou si on se trompe dans le message de validation.

git commit --amend

Cette commande prend en compte la zone d'index et l'utilise pour le `commit`. Si aucune modification n'a été réalisée depuis la dernière validation (

- ✓ **git commit -m 'validation initiale'**
- ✓ **git add fichier_oublie**
- ✓ **git commit --amend**

Désindexer un fichier déjà indexé

Les deux sections suivantes démontrent comment bricoler les modifications dans votre zone d'index et votre zone de travail.

```
git add .
git status
git reset HEAD CONTRIBUTING.md
git status
```

Réinitialiser un fichier modifié

Important	Vous devriez aussi vous apercevoir que c'est une commande **dangereuse** : toutes les modifications que vous auriez réalisées sur ce fichier ont disparues Il vient tout juste de l'écraser par un autre fichier. **Ne jamais utiliser cette commande à moins d'être vraiment sûr** de ne pas vouloir de ces modifications.

Que faire pour ne pas conserver les modifications du fichier `CONTRIBUTING.md` ?
Comment le réinitialiser facilement, le ramener à son état du dernier .

```
git add LISEZMOI
git checkout -- LISEZMOI
```

Ce qui vous indique de façon explicite comment annuler des modifications que vous avez faites. Faisons comme indiqué :

```
git checkout – LISEZMOI
git status
git reset HEAD LISEZMOI //les modifications ont été annulées.
```

Réinitialiser les choses avec git restore

Git version 2.25.0 a introduit une nouvelle commande : `git restore`.
C'est fondamentalement une alternative à `git reset`
Vous avez accidentellement tapé **git add *** et avez indexé les deux.
Comment désindexer l'une de deux ?

```
git add *
git status
git restore --staged README.md
git restore --staged CONTRIBUTING.md
git status
```

Réinitialiser un fichier modifié avec git restore

Que faire si vous vous apercevez que vous ne voulez pas garder les modifications du fichier `CONTRIBUTING.md` ?
Comment le modifier simplement

```
git add/rm README.md
git restore README.md
git restore CONTRIBUTING.md
git status
```

Travailler avec des dépôts distants

Pour pouvoir collaborer sur un projet Git, il est nécessaire de savoir comment gérer les dépôts distants.
Les dépôts distants sont des versions de votre projet qui sont hébergées sur Internet ou le réseau d'entreprise.
Vous pouvez en avoir plusieurs, pour lesquels vous pouvez avoir des droits soit en lecture seule, soit en lecture/écriture.
Collaborer avec d'autres personnes consiste à gérer ces dépôts distants, en poussant ou tirant des données depuis et vers ces dépôts quand vous souhaitez partager votre travail.

Afficher les dépôts distants

Pour visualiser les serveurs distants que vous avez enregistrés, vous pouvez lancer la commande **git remote.** Elle liste les noms des différentes références distantes que vous avez spécifiées. Si vous avez cloné un dépôt, vous devriez au moins voir l'origine **origin** — c'est-à-dire le nom par défaut que Git donne au serveur à partir duquel vous avez cloné :

```
git clone https://github.com/schacon/ticgit
cd ticgit
git remote
git remote -v
cd grit
git remote -v
```

Ajouter des dépôts distants

Voici spécifiquement comment faire

```
git remote
git remote add pb https://github.com/paulboone/ticgit
git remote -v
git fetch pb
```

Récupérer et tirer depuis des dépôts distants

git fetch [remote-name]

Cette commande s'adresse au dépôt distant et récupère toutes les données de ce projet que vous ne possédez pas encore. Après cette action, vous possédez toutes les références à toutes les branches contenues dans ce dépôt, que vous pouvez fusionner ou inspecter à tout moment.

Habituellement c'est le gestionnaire de projet qui récupère le projet intégral ou pour une sauvegarde en externe ex clef USB pour le projet

Si vous clonez un dépôt, le dépôt distant est automatiquement ajouté sous le nom « origin ».

Pousser son travail sur un dépôt distant

Lorsque votre dépôt vous semble prêt à être partagé, il faut le pousser en amont.
La commande pour le faire

git push <nom-distant> <nom-de-branche>

Pousser votre branche `master` vers le serveur `origin`
Pousser votre travail vers le serveur amont :

git push origin master

Cette commande ne fonctionne que si vous avez cloné depuis un serveur sur lequel vous avez des droits d'accès en écriture et si personne n'a poussé dans l'intervalle. Si vous et quelqu'un d'autre clonez un dépôt au même moment et que cette autre personne pousse ses modifications et qu'après vous tentez de pousser les vôtres, **votre poussée sera rejetée à juste titre.**

Inspecter un dépôt distant

Si vous souhaitez visualiser plus d'informations à propos d'un dépôt distant particulier

git remote show [nom-distant].

Si vous lancez cette commande avec un nom court particulier, tel que `origin`

git remote show origin

git pull.

Retirer et renommer des dépôts distants

Par exemple, si vous souhaitez renommer `pb` en `paul`

git remote rename pb paul

git remote

git remote rm paul

git remote

Étiquetage

À l'instar de la plupart des VCS, Git donne la possibilité d'étiqueter un certain état dans l'historique comme important.

Lister vos étiquettes

Lister les étiquettes existantes dans Git est très simple. Tapez juste `git tag` :

git tag

Cette commande liste les étiquettes dans l'ordre alphabétique.
L'ordre dans lequel elles apparaissent n'a aucun rapport avec l'historique.
Vous pouvez aussi rechercher les étiquettes correspondant à un motif particulier.

git tag -l 'v1.8.5*'

Créer des étiquettes

Une **étiquette légère** ressemble beaucoup à une branche qui ne change pas, c'est juste un pointeur sur un **`commit`** spécifique.

Les **étiquettes annotées**, en revanche, sont stockées en tant qu'objets à part entière dans la base de données de Git.

Elles ont

- ✓ une somme de contrôle,
- ✓ contiennent le nom et l'adresse électronique du créateur,
- ✓ la date,
- ✓ un message d'étiquetage
- ✓ peuvent être signées et vérifiées

Il est généralement recommandé de créer des étiquettes annotées pour générer toute cette information

Mais si l'étiquette doit rester temporaire ou l'information supplémentaire n'est pas désirée, les étiquettes légères peuvent suffire.

Les étiquettes annotées

L'option `-m` permet de spécifier le message d'étiquetage qui sera stocké avec l'étiquette **git tag -a v1.4 -m 'ma version 1.4'**

Git lance votre éditeur pour pouvoir le saisir. **git tag**

Vous pouvez visualiser les données de l'étiquette à côté du **`commit`** qui a été marqué en utilisant la commande `git show` **git show v1.4**

Les étiquettes légères

Une autre manière d'étiqueter les commits est d'utiliser les étiquettes légères. Celles-ci se réduisent à stocker la somme de contrôle d'un commit dans un fichier, aucune autre information n'est conservée. Pour créer une étiquette légère, il suffit de n'utiliser aucune des options -a, -s ou -m :

git tag v1.4-lg

git tag

La commande ne montre que l'information de validation :**git show v1.4-lg**

Étiqueter après coup

Vous pouvez aussi étiqueter des **`commits`** plus anciens. Supposons que l'historique des **`commits`** ressemble à ceci :

git log --pretty=oneline

Supposons avoir oublié d'étiqueter le projet à la version `v1.2` qui correspondait au **`commit`** « mise à jour rakefile ».
git tag -a v1.2 9fceb02 //id du commit cible
git show v1.2

Partager les étiquettes

Par défaut, la commande `git push` ne transfère pas les étiquettes vers les serveurs distants.
Il faut explicitement pousser les étiquettes après les avoir créées localement.
Ce processus s'apparente à pousser des branches distantes **git push origin v1.5**
Si vous avez de nombreuses étiquettes que vous souhaitez pousser en une fois **git push origin --tags**

Supprimer les étiquettes

Pour supprimer une étiquette de votre dépôt local **git tag -d v1.4-lw**
Pour supprimer une étiquette d'un serveur distant. **git push origin :refs/tags/v1.4-lw**
La seconde manière (et la plus intuitive) pour supprimer une étiquette distante utilise l'option **git push origin --delete <nom-d-etiquette>**

Extraire une étiquette

Si vous souhaitez voir les versions de fichiers qu'une étiquette pointe **git checkout v2.29.2**
`Si vous voulez créer une nouvelle branche pour conserver les commits que vous créez` **git switch -c <nom-de-la-nouvelle-branche>** Ou annuler cette opération **git switch -**
git checkout v2.29.1

Dans l'état « HEAD détachée », si vous modifiez puis créez un commit, l'étiquette restera identique, mais votre nouveau commit n'appartiendra à aucune branche et sera non joignable, à part avec son empreinte de commit exacte. Ainsi, si vous avez besoin de faire des modifications — disons que vous corrigez un bogue d'une ancienne version, par exemple — vous voudrez généralement créer une branche : **git checkout -b v2.9.X**
Si vous faites ceci et que vous faites un commit, votre branche `v2.9.X` sera légèrement différente de votre étiquette `v2.9.1` puisqu'elle aura avancé avec les modifications que vous y aurez intégrées, donc faites attention.