Installer et configurer Git

1. Installation

Nous allons voir ici comment installer Git sous Linux, Windows et Mac OS X. Comme je vous le disais plus tôt, Git est plus agréable à utiliser sous Linux et sensiblement plus rapide, mais il reste néanmoins utilisable sous Windows.

Installer Git sous Linux

Avec un gestionnaire de paquets, c'est très simple :

```
[user@localhost ~]$ sudo yum install git-all.noarch

[user@localhost ~]$ git --version
git version 1.8.3.1
```

Cela installe 2 paquets :

- git: c'est git, tout simplement. C'est le seul paquet vraiment indispensable ;

2. Configurer Git

Maintenant que Git est installé, vous configurer une console dans laquelle vous allez pouvoir utiliser des commandes de Git.

```
$ sudo dnf install zsh
$ curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-
zsh/master/tools/install.sh | sh; zsh
```

Elles activeront la couleur dans Git. Il ne faut le faire qu'une fois, et ça aide à la lisibilité des messages dans la console.

De même, il faut configurer votre nom (ou pseudo) :

```
$ git config --global user.name "elies_jebri"
```

Puis votre e-mail :

```
$ git config --global user.email elies.jebri@gmail.com
```

Vous pouvez aussi éditer votre fichier de configuration `.gitconfig` situé dans votre répertoire personnel pour y ajouter une section alias à la fin :

```
$ vim ~/.gitconfig
```

```
[color]
        diff = auto
        status = auto
        branch = auto
[user]
        name = elies_jebri
        email = elies.jebri@gmail.com
[alias]
        ci = commit
        co = checkout
        st = status
        br = branch
```

```
[user@localhost ~]$ git config --global user.name
elies_jebri
```

```
[user@localhost ~]$ git config --list
color.diff=auto
color.status=auto
color.branch=auto
user.name=elies_jebri
user.email=elies.jebri@gmail.com
alias.ci=commit
alias.co=checkout
alias.st=status
alias.br=branch
```

Ces alias permettent de raccourcir certaines commandes de Git. Ainsi, au lieu d'écrire `git status`, vous pourrez écrire si vous le désirez `git st`, ce qui est plus court.

Modifier la branche par défaut de « master » à « main »

```
$ git config --global init.defaultBranch main
```

3. Créer un nouveau dépôt local

Commencez par créer un dossier du nom de votre projet sur votre disque. Par exemple, je vais créer`/home/user/LabsGit/FirstProject`

```
$ mkdir -p LabsGit/FirstProject
```

```
$ cd LabsGit/FirstProject/
```

Ensuite, initialisez un dépôt Git tout neuf dans ce dossier avec la commande :

```
$ git init
```

Vous venez de créer un nouveau projet Git dans le dossier où vous vous trouvez. Un dossier caché *.git* vient tout simplement d'être créé.

Il faudra ensuite créer les fichiers source de votre projet et les faire connaître à Git en faisant des commits.

4. Cloner un dépôt existant

Cloner un dépôt existant consiste à récupérer tout l'historique et tous les codes source d'un projet avec Git.

On se connecte au dépôt en HTTPS, mais il existe d'autres méthodes : les protocoles «git://» et «ssh://».

Pour cloner le dépôt de Gnu-Hello, il suffit de lancer la commande suivante :

```
[user@localhost ~]$ cd ; cd LabsGit/
LabsGit]$ git clone https://git.savannah.gnu.org/git/hello.git Gnu-Hello
Cloning into ' Gnu-Hello'...
remote: Counting objects: 4528, done.
remote: Compressing objects: 100% (1123/1123), done.
Receiving objects:  69% (3125/4528), 3.09 MiB | 682.00 KiB/s
```

```
[user@localhost LabsGit]$ ls
FirstProject  Gnu-Hello
```

5. Modifier le code et effectuer des commits

Placez-vous dans le répertoire de base du code, par exemple :

```
[user@localhost ~]$ cd /home/user/LabsGit/FirstProject/
```

La commande `git status` vous indique les fichiers que vous avez modifiés récemment :

```
[user@localhost FirstProject]$ git status
# On branch main
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

Ce message nous informe que rien n'a été modifié (*nothing to commit*).

Vous allez créer deux fichiers : (notez que l'erreur « Prject » est volontaire)

```
[user@localhost FirstProject]$ touch file1.txt file2.txt
```

```
[user@localhost FirstProject]$ echo "My first Git Prject" > file1.txt
```

```
[user@localhost FirstProject]$ echo "Hello World" > file2.txt
```

```
[user@localhost FirstProject]$ git status
# On branch main
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file1.txt
#       file2.txt
nothing added to commit but untracked files present (use "git add" to
track)
```

Vous allez ajouter le fichier « file1.txt » au Staging Area :

```
[user@localhost FirstProject]$ git add file1.txt
```

```
[user@localhost FirstProject]$ git status
# On branch main
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file2.txt
```

C’est maintenant que vous vous apercevez que le mot « Project » est écrit avec erreur dans «file1.txt» et que vous décidez de le corriger :

```
[root@localhost FirstProject]# echo "My first Git Project" > file1.txt
```

```
[root@localhost FirstProject]# git status
# On branch main
## Initial commit
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   file1.txt
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#       file2.txt
```

Vérifiez quelles sont les différences entre les deux fichiers :

```
[root@localhost FirstProject]# git diff file1.txt
diff --git a/file1.txt b/file1.txt
index ff5c5ba..57ecbbd 100644
--- a/file1.txt
+++ b/file1.txt
@@ -1 +1 @@
-My first Git Prject
+My first Git Project
```

Cette commande compare le contenu du répertoire de travail avec la zone d'index. Le résultat vous indique les modifications réalisées mais non indexées.

Vous allez ajouter maintenant tous les fichiers au Staging Area avec la màj de file1.txt, mais aussi sans vous en rendre compte un fichier avec des données privées :

```
[user@localhost FirstProject]$ touch private.txt

[user@localhost FirstProject]$ git add -A

[user@localhost FirstProject]$ git status
# On branch main
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1.txt
#       new file:   file2.txt
#       new file:   private.txt
#
```

Pour supprimer le fichier « private.txt » du Staging Area :

```
[user@localhost FirstProject]$ git rm --cached private.txt
rm 'file1.txt'

[user@localhost FirstProject]$ git status
# On branch main
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1.txt
#       new file:   file2.txt
```

Pour ignorer ce fichier lors des prochains ajouts vous allez l'ajouter à ***.gitignore*** :

```
[user@localhost FirstProject]$ echo private.txt >> .gitignore
```

```
[user@localhost FirstProject]$  git status
# On branch main
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1.txt
#       new file:   file2.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#       .gitignore
```

```
[user@localhost FirstProject]$ git add -A
```

```
[user@localhost FirstProject]$ git status
# On branch main
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   .gitignore
#       new file:   file1.txt
#       new file:   file2.txt
```

Avant de valider par un commit nous allons effectuer un changement au niveau de « file1.txt » (supprimer le contenu par inadvertance) :

```
[user@localhost FirstProject]$ echo '' > file1.txt
```

```
[user@localhost FirstProject]$ git status
# On branch main
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   .gitignore
#       new file:   file1.txt
#       new file:   file2.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   file1.txt
```

Vous pouvez alors récupérer la version déjà indexée dans le Staging Area comme suit :

```
[user@localhost FirstProject]$ git checkout file1.txt
```

```
[user@localhost FirstProject]$ cat file1.txt
My first Git Project
```

Maintenant que votre zone d'index (Staging) est dans l'état désiré, vous pouvez valider vos modifications (commit).

```
[user@localhost FirstProject]$ git commit
Aborting commit due to empty commit message.
```

```
[user@localhost FirstProject]$ git commit -m "First commit"

[main (root-commit) 4fc0072] First commit
 3 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 file1.txt
 create mode 100644 file2.txt
```

```
[user@localhost FirstProject]$ git status
# On branch main
nothing to commit, working directory clean
```

6. Modifier le dernier message de commit

Si vous avez fait une faute d'orthographe dans votre dernier message de commit ou que vous voulez tout simplement le modifier, vous pouvez le faire facilement grâce à la commande suivante :

```
[user@localhost FirstProject]$ echo 'Ligne erreur' >> file2.txt
```

```
[user@localhost FirstProject]$ git status
# On branch main
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   file2.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
[user@localhost FirstProject]$ git add file2.txt

[user@localhost FirstProject]$ git commit -m "First commit"
[main 6e7b594] First commit
 1 file changed, 1 insertion(+)
```

```
[user@localhost FirstProject]$ git log
commit 6e7b594ab92024b1e50954bd35e231ce7e970f9d
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Thu Mar 5 17:35:28 2020 +0100
```

```
    First commit

commit 4fc007257fac301ad03c5883da76a76380c9669e
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Wed Mar 4 22:59:46 2020 +0100

    First commit
```

```
[user@localhost FirstProject]$ git add file2.txt
```

```
[user@localhost FirstProject]$ git commit --amend -m "j'ai ajouté une ligne
erreur"
[main ec787f2] j'ai ajouté une ligne erreur
 1 file changed, 1 insertion(+)
```

```
[user@localhost FirstProject]$ git log
commit ec787f2b0ad157256642ae9afb780dc277ac26b9
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Thu Mar 5 17:35:28 2020 +0100

    j'ai ajouté une ligne erreur

commit 4fc007257fac301ad03c5883da76a76380c9669e
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Wed Mar 4 22:59:46 2020 +0100

    First commit
```

7. Annuler un commit effectué par erreur ou contenant des erreurs

Il est fréquent de chercher à comprendre ce qui s'est passé récemment, pourquoi une erreur a été introduite et comment annuler ce changement qui pose problème.

Vous avez glissé une ligne d'erreur dans le dernier commit et vous devez l'annuler :

```
[user@localhost FirstProject]$ sed -i 's/Ligne erreur/Ligne corrigée/' \
file2.txt
```

```
[user@localhost FirstProject]$ git status
# On branch main
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   file2.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
[user@localhost FirstProject]$ git reset --soft 4fc007

[user@localhost FirstProject]$ git log
commit 4fc007257fac301ad03c5883da76a76380c9669e
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Thu Mar 5 22:36:54 2020 +0100

    First commit
```

```
[user@localhost FirstProject]$ git status
# On branch main
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   file2.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   file2.txt
#
```

```
[user@localhost FirstProject]$ git add file2.txt
```

```
[user@localhost FirstProject]$ git status
# On branch main
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   file2.txt
#
```

```
[user@localhost FirstProject]$ git commit -m "Ajout ligne corrigée"

[main 44f4c51] Ajout ligne corrigée
 1 file changed, 1 insertion(+)
```

```
[user@localhost FirstProject]$ git log
commit 44f4c51540f1d646f79db8317f0b9bcabaf11316
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Thu Mar 5 23:23:25 2020 +0100

    Ajout ligne corrigée

commit 8ff11f55b68ff539ca74587641989c29e54db4b5
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Thu Mar 5 22:36:54 2020 +0100

    First commit
```

```
[user@localhost FirstProject]$ git show 44f4c5
commit 44f4c51540f1d646f79db8317f0b9bcabaf11316
Author: elies_jebri <elies.jebri@gmail.com>
Date:   Thu Mar 5 23:23:25 2020 +0100

    Ajout ligne corrigée

diff --git a/file2.txt b/file2.txt
index 557db03..0f2276f 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1 +1,2 @@
 Hello World
+Ligne corrigée
```