Table des matières

1) Utilisation basique

1) Je commence par créer un dossier créatures puis je me déplace dedans :

```
test@232-22:~$ mkdir creatures
test@232-22:~$ cd creatures/
```

Maintenant j'initialise un dépôt git dans mon dossier créatures

```
test@232-22:~/creatures$ git init
Dépôt Git vide initialisé dans /home/test/creatures/.git/
```

Je crée un fichier dwarf.txt contenant les informations d'un nain :

```
test@232-22:~/creatures$ touch dwarf.txt
test@232-22:~/creatures$ vim dwarf.txt
```

2) Je fais un git status pour vérifier l'état actuel de mon dépôt :

```
test@232-22:~/creatures$ git status
Sur la branche master

Aucun commit

Fichiers non suivis:
```

```
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        dwarf.txt

aucune modification ajoutée à la validation mais des fichiers non suivis sont
présents (utilisez "git add" pour les suivre)
```

Je vois qu'aucun commit n'a été fait.

3) Je prépare mon fichier dwarf.txt au commit avec uun git add :

```
test@232-22:~/creatures$ git add dwarf.txt
test@232-22:~/creatures$ git status
Sur la branche master

Aucun commit

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

        nouveau fichier : dwarf.txt
```

On voit bien que le fichier est prêt pour le commit.

4) Je fais un git diff :

```
test@232-22:~/creatures$ git diff --cached
diff --git a/dwarf.txt b/dwarf.txt
new file mode 100644
index 0000000..ebb881e
--- /dev/null
+++ b/dwarf.txt
@@ -0,0 +1,5 @@
+taille
+force
+barbe
+hache
+elfophobie
```

Je vois qu'il y a des différences par entre l'index et la dernière version présente. Je vois qu'il y a les 5 lignes de mon fichier qui ont été ajouté à l'index ce qui n'était pas le cas avant.

5) Je commit mon travail puis je vérifie avec un git status que cela a bien fonctionné :

```
test@232-22:~/creatures$ git commit -m "first commit"
[master (commit racine) 22ea3df] first commit
 Committer: test <test@244-0>
```

```
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant sur votre nom d'utilisateur et le nom de votre machine. Veuillez
vérifier qu'ils sont corrects. Vous pouvez supprimer ce message en les
paramétrant explicitement. Lancez les commandes suivantes et suivez les
instruction dans votre éditeur pour éditer votre fichier de configuration :

    git config --global --edit

Après ceci, vous pouvez corriger l'identité utilisée pour cette validation avec :

    git commit --amend --reset-author

 1 file changed, 5 insertions(+)
 create mode 100644 dwarf.txt
test@232-22:~/creatures$ git status
Sur la branche master
rien à valider, la copie de travail est propre
```

Je vois que le commit a bien fonctionné.

6) J'affiche la liste des changements effectués :

```
test@232-22:~/creatures$ git log
commit 22ea3df48791ea899a19c9d7afb12ce86de779c1 (HEAD -> master)
Author: test <test@244-0>
Date:   Tue Nov 10 13:29:42 2020 +0100

    first commit
```

Je vois qu'il y a un seul changement. Le hash du dernier commit effectué est surligné en bleu au dessus.

7) Je dois créer des nouvelles créatures en leur donnant chacune des descriptions différentes :

```
test@232-22:~/creatures$ touch hobbit.txt
test@232-22:~/creatures$ touch elf.txt
test@232-22:~/creatures$ touch dragon.txt
test@232-22:~/creatures$ touch humain.txt
```

Modification d'une description :

Je modifie la description de mon nain puis je fais un git diff pour voir ce qui a changé :

```
test@232-22:~/creatures$ git diff
diff --git a/dwarf.txt b/dwarf.txt
index ebb881e..7979917 100644
--- a/dwarf.txt
```

```
+++ b/dwarf.txt
@@ -1,5 +1,5 @@
-taille
-force
-barbe
-hache
-elfophobie
+0.60
+1
+2
+5
+8
```

Je vois que la description est bien changée. Par exemple il n'y a plus (taille, force, barbe…) et à la place d'autres informations ont été ajouté (0.60, 1…).

Maintenant je peux git add de nouveau mon nain et vérifier ce qui va être commité :

```
test@232-22:~/creatures$ git add dwarf.txt
test@232-22:~/creatures$ git diff --cached
diff --git a/dwarf.txt b/dwarf.txt
index ebb881e..7979917 100644
--- a/dwarf.txt
+++ b/dwarf.txt
@@ -1,5 +1,5 @@
-taille
-force
-barbe
-hache
-elfophobie
+0.60
+1
+2
+5
+8
```

Je vois que dans mon nouveau commit toutes les infos précédentes vont êtres supprimées et remplacées par les nouvelles.

Si je fais un git status je peux voir que seul mon fichier dwarf.txt va recevoir des modifications (les autres fichiers on ne s'en occupe pas) :

```
test@232-22:~/creatures$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)
```

```
        modifié :        dwarf.txt

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        dragon.txt
        elf.txt
        hobbit.txt
        humain.txt
```

Je peux maintenant commiter le fichier :

```
test@232-22:~/creatures$ git commit -m "commit 2"
[master f699ae3] commit 2
 Committer: test <test@244-0>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant sur votre nom d'utilisateur et le nom de votre machine. Veuillez
vérifier qu'ils sont corrects. Vous pouvez supprimer ce message en les
paramétrant explicitement. Lancez les commandes suivantes et suivez les
instruction dans votre éditeur pour éditer votre fichier de configuration :

    git config --global --edit

Après ceci, vous pouvez corriger l'identité utilisée pour cette validation avec :

    git commit --amend --reset-author

 1 file changed, 5 insertions(+), 5 deletions(-)
```

Il y a bien eu mes 10 modifications (5 insertions et 5 suppressions)

Maintenant pour finir on peut faire un git log et comparer le hash du fichier avec celui d'avant :

```
test@232-22:~/creatures$ git log
commit f699ae39ce6dc9d4862182925109e88ef0cde6c6 (HEAD -> master)
Author: test <test@244-0>
Date:   Tue Nov 10 13:54:08 2020 +0100

    commit 2

commit 22ea3df48791ea899a19c9d7afb12ce86de779c1
Author: test <test@244-0>
Date:   Tue Nov 10 13:29:42 2020 +0100

    first commit
```

Je vois bien que les hash sont différents entre les commit 1 et le commit 2.

Création d'une description :

Dans cette partie je crée une nouvelle créature (hobbit.txt) puis je fais un git diff :

```
test@232-22:~/creatures$ vim hobbit.txt
test@232-22:~/creatures$ git diff
```

Quand je fais le git diff je ne vois aucune différences ce qui est normal car le fichier vient juste d'être crée.

Je peux maintenant add le fichier et voir ce qui va être commit :

```
 test@232-22:~/creatures$ git add hobbit.txt
test@232-22:~/creatures$ git diff --cached
diff --git a/hobbit.txt b/hobbit.txt
new file mode 100644
index 0000000..bf4cba1
--- /dev/null
+++ b/hobbit.txt
@@ -0,0 +1,5 @@
+taille 1m20
+force 5/10
+barbe 0
+hache 2
+phobie 8
```

Je vois qu'il va y avoir 5 insertions pendant le commit.

Si je fais un git status je peux voir que des modifications sont apportées au fichier hobbit.txt et que les autres ne sont pas pris en compte. Le fichier dwarf.txt a disparu car il est déjà commit

```
test@232-22:~/creatures$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        nouveau fichier : hobbit.txt

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        dragon.txt
        elf.txt
        humain.txt
```

Je peux maintenant commit le fichier :

```
test@232-22:~/creatures$ git commit -m "commit 3"
[master b0ebc83] commit 3
 Committer: test <test@244-0>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant sur votre nom d'utilisateur et le nom de votre machine. Veuillez
vérifier qu'ils sont corrects. Vous pouvez supprimer ce message en les
paramétrant explicitement. Lancez les commandes suivantes et suivez les
instruction dans votre éditeur pour éditer votre fichier de configuration :

    git config --global --edit

Après ceci, vous pouvez corriger l'identité utilisée pour cette validation avec :

    git commit --amend --reset-author

 1 file changed, 5 insertions(+)
 create mode 100644 hobbit.txt
```

1 fichier a été changé et c'est bien le hobbit.txt.

Je peux maintenant faire un git log :

```
test@232-22:~/creatures$ git log
commit b0ebc83d4fcefcfb017845bfb444937e6bd06754 (HEAD -> master)
Author: test <test@244-0>
Date:   Tue Nov 10 14:06:53 2020 +0100

    commit 3

commit f699ae39ce6dc9d4862182925109e88ef0cde6c6
Author: test <test@244-0>
Date:   Tue Nov 10 13:54:08 2020 +0100

    commit 2

commit 22ea3df48791ea899a19c9d7afb12ce86de779c1
Author: test <test@244-0>
Date:   Tue Nov 10 13:29:42 2020 +0100

    first commit
```

Je vois en bleu que le hash est aussi différents des autres car le fichier ne contient pas exactement les mêmes caractères que les autres fichiers.

Si je faisais d'autres créations de créatures en les faisant une par une j'obtiendrais exactement les mêmes choses qu'au dessus avec les mêmes modifications et toujours un hash différent.

8) Je regarde à nouveau l'historique des modifications avec un git log et je vérifie avec git status que mes commit ont bien été effectué :

```
test@232-22:~/creatures$ git log
commit b0ebc83d4fcefcfb017845bfb444937e6bd06754 (HEAD -> master)
Author: test <test@244-0>
Date:   Tue Nov 10 14:06:53 2020 +0100

    commit 3

commit f699ae39ce6dc9d4862182925109e88ef0cde6c6
Author: test <test@244-0>
Date:   Tue Nov 10 13:54:08 2020 +0100

    commit 2

commit 22ea3df48791ea899a19c9d7afb12ce86de779c1
Author: test <test@244-0>
Date:   Tue Nov 10 13:29:42 2020 +0100

    first commit
```

Les hash sont différents car les fichiers n'ont pas le même contenu.

```
test@232-22:~/creatures$ git status
Sur la branche master
rien à valider, la copie de travail est propre
```

Avec un git status je vois que tous mes commit sont bien effectués.

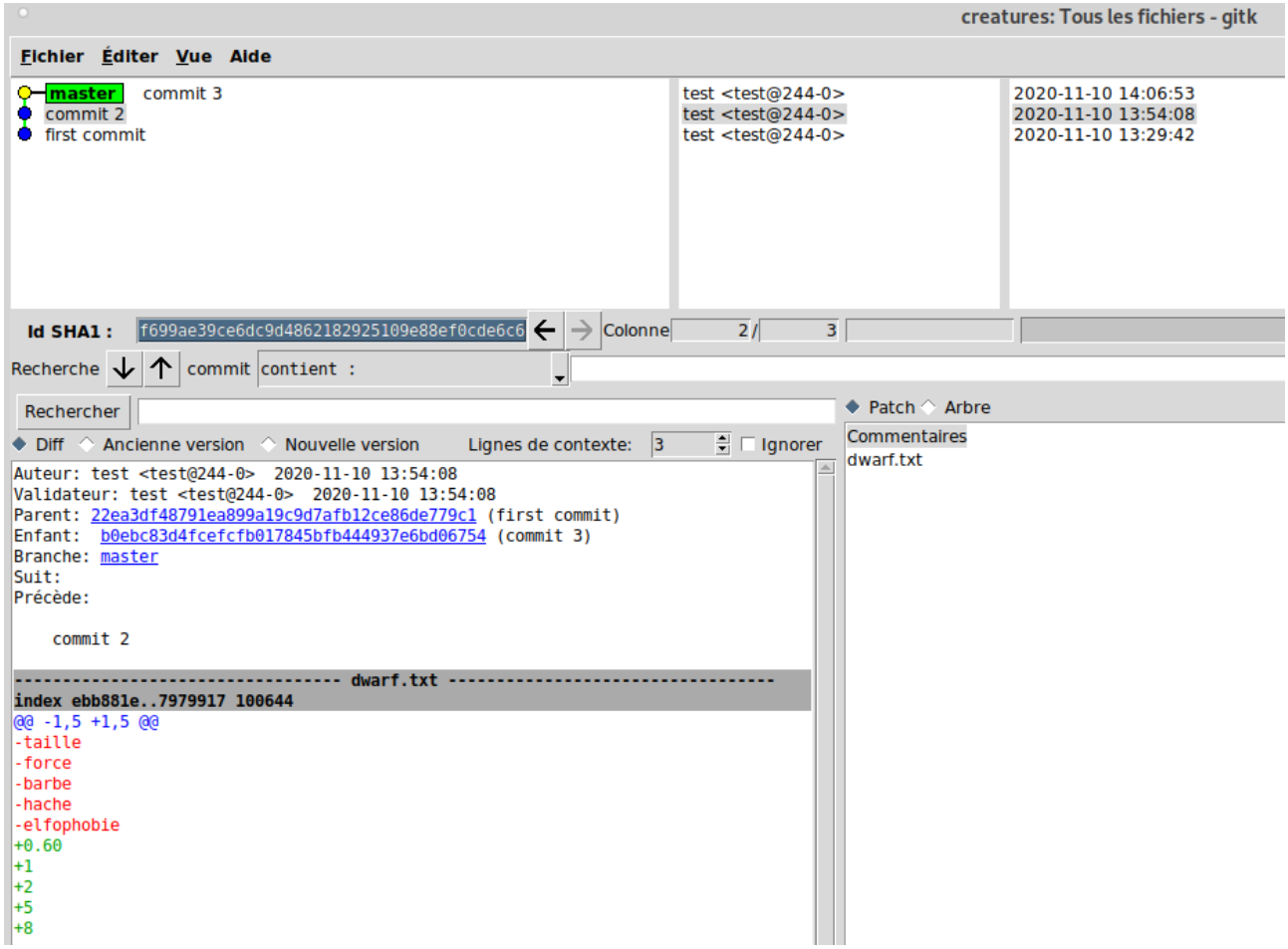
Je test maintenant différents git log pour voir les modifications :

```
test@232-22:~/creatures$ git log --graph --pretty=short
* commit b0ebc83d4fcefcfb017845bfb444937e6bd06754 (HEAD -> master)
| Author: test <test@244-0>
|
|     commit 3
|
* commit f699ae39ce6dc9d4862182925109e88ef0cde6c6
| Author: test <test@244-0>
|
|     commit 2
|
* commit 22ea3df48791ea899a19c9d7afb12ce86de779c1
  Author: test <test@244-0>

      first commit
```

Avec cette commande je vois les branches dans lesquelles se situent mes modifications

Commande gitk :



Je peux voir dans gitk ma branche master avec tous les commit qui ont suivis ainsi que tout ce qui a été modifié (en bas)

C’est la même chose dans gitg :

2) Voyage dans le temps

1) Je fais la modifications d'une de mes créatures (hobbit.txt par exemple) puis je l'ajoute à l'index avec git add. Je vérifie avec git status que cela a bien fonctionné.

```
test@232-22:~/creatures$ vim hobbit.txt
test@232-22:~/creatures$ git add hobbit.txt
test@232-22:~/creatures$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        modifié :         hobbit.txt
```

Tout est bon mais maintenant je me rend compte qu'en fait je ne veux plus de cette modification donc je l'annule :

```
test@232-22:~/creatures$ git reset HEAD hobbit.txt
Modifications non indexées après reset :
M       hobbit.txt
```

Mon git add a bien été annulé :

```
test@232-22:~/creatures$ git status
Sur la branche master
Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la
copie de travail)

        modifié :         hobbit.txt

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git
commit -a")
```

2) Je supprime ma modification :

```
test@232-22:~/creatures$ git checkout hobbit.txt
test@232-22:~/creatures$ git status
Sur la branche master
rien à valider, la copie de travail est propre
```

3) Je vérifie mes différents commit et je fais un git log d'un de mes commit (mais pas le dernier) :

```
test@232-22:~/creatures$ git log
commit b0ebc83d4fcefcfb017845bfb444937e6bd06754 (HEAD -> master)
Author: test <test@244-0>
Date:   Tue Nov 10 14:06:53 2020 +0100

    commit 3

commit f699ae39ce6dc9d4862182925109e88ef0cde6c6
Author: test <test@244-0>
Date:   Tue Nov 10 13:54:08 2020 +0100

    commit 2

commit 22ea3df48791ea899a19c9d7afb12ce86de779c1
Author: test <test@244-0>
Date:   Tue Nov 10 13:29:42 2020 +0100

    first commit
```

```
test@232-22:~/creatures$ git checkout
f699ae39ce6dc9d4862182925109e88ef0cde6c6
Note : extraction de 'f699ae39ce6dc9d4862182925109e88ef0cde6c6'.

Vous êtes dans l'état « HEAD détachée ». Vous pouvez visiter, faire des
modifications
expérimentales et les valider. Il vous suffit de faire une autre extraction pour
abandonner les commits que vous faites dans cet état sans impacter les autres
branches

Si vous voulez créer une nouvelle branche pour conserver les commits que
vous créez,
il vous suffit d'utiliser « checkout -b » (maintenant ou plus tard) comme ceci :

  git checkout -b <nom-de-la-nouvelle-branche>

HEAD est maintenant sur f699ae3 commit 2
```

Je suis maintenant revenu sur mon commit numéro deux et le 3ème a été supprimé donc mon fichier hobbit a disparu :

```
test@232-22:~/creatures$ ls
dwarf.txt
```

Je fais un git status :

```
test@232-22:~/creatures$ git status
```

```
HEAD détachée sur f699ae3
rien à valider, la copie de travail est propre
```

Je vois que la tête est détachée sur mon commit.

Je fais maintenant un git log avec et sans --all :

```
test@232-22:~/creatures$ git log
commit f699ae39ce6dc9d4862182925109e88ef0cde6c6 (HEAD)
Author: test <test@244-0>
Date:   Tue Nov 10 13:54:08 2020 +0100

    commit 2

commit 22ea3df48791ea899a19c9d7afb12ce86de779c1
Author: test <test@244-0>
Date:   Tue Nov 10 13:29:42 2020 +0100

    first commit
```

```
test@232-22:~/creatures$ git log --all
commit b0ebc83d4fcefcfb017845bfb444937e6bd06754 (master)
Author: test <test@244-0>
Date:   Tue Nov 10 14:06:53 2020 +0100

    commit 3

commit f699ae39ce6dc9d4862182925109e88ef0cde6c6 (HEAD)
Author: test <test@244-0>
Date:   Tue Nov 10 13:54:08 2020 +0100

    commit 2

commit 22ea3df48791ea899a19c9d7afb12ce86de779c1
Author: test <test@244-0>
Date:   Tue Nov 10 13:29:42 2020 +0100

    first commit
```

Sans le --all je ne vois que les commit jusqu'à celui auquel je me situe et avec --all je vosi même le commit dans lequel je ne suis pas.

4) Je reviens à la version la plus récente de mon projet :

```
test@232-22:~/creatures$ git checkout master
```

```
La position précédente de HEAD était sur f699ae3 commit 2
Basculement sur la branche 'master'
```

Le git status me dit que tout est bon sur mes commit :

```
test@232-22:~/creatures$ git status
Sur la branche master
rien à valider, la copie de travail est propre
```

5) Voici le contenu actuel de mon fichier dwarf.txt :

```
test@232-22:~/creatures$ cat dwarf.txt
0.60
1
2
5
8
```

J’ai fais une erreur et je veux annuler la modification donc je revert le commit contenant la modification du fichier :

```
test@232-22:~/creatures$ git revert
f699ae39ce6dc9d4862182925109e88ef0cde6c6
[master dcadf59] Revert "commit 2"
 Committer: test <test@244-0>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant sur votre nom d'utilisateur et le nom de votre machine. Veuillez
vérifier qu'ils sont corrects. Vous pouvez supprimer ce message en les
paramétrant explicitement. Lancez les commandes suivantes et suivez les
instruction dans votre éditeur pour éditer votre fichier de configuration :

    git config --global --edit

Après ceci, vous pouvez corriger l'identité utilisée pour cette validation avec :

    git commit --amend --reset-author

 1 file changed, 5 insertions(+), 5 deletions(-)
```

Je peux voir que je suis bien revenu au fichier de base sans la modification :

```
test@232-22:~/creatures$ cat dwarf.txt
taille
force
barbe
hache
```

```
elfophobie
```

3) Gestion des branches

1) Je crée une branche nommée dragon et je bascule dedans :

```
test@232-22:~/creatures$ git branch dragon
test@232-22:~/creatures$ git checkout dragon
Basculement sur la branche 'dragon'
```

Dedans j'y crée plusieurs fichiers :

```
test@232-22:~/creatures$ touch fred.txt
test@232-22:~/creatures$ touch seb.txt
test@232-22:~/creatures$ vim fred.txt
test@232-22:~/creatures$ vim seb.txt
```

Je fais un commit par fichier :

```
test@232-22:~/creatures$ git log
commit 5564a8e6fcd32720c8f021bec5d77ec37e66c231 (HEAD -> dragon)
Author: test <test@244-0>
Date:   Tue Nov 10 15:53:19 2020 +0100

    fichier seb

commit 2bfd85dd3bf7c59a94291f21f2b72f3deb22e9a2
Author: test <test@244-0>
Date:   Tue Nov 10 15:53:02 2020 +0100

    fichier fred
```

Je vois bien que les commit sont fait dans la branche dragon.

2) Je veux garder mon travail et donc le mettre dans la branche principale. Il faut que je commence par me déplacer dans la banche master :

```
test@232-22:~/creatures$ git checkout master
Basculement sur la branche 'master'
```

Maintenant je peux mettre mon travail dans la branche master :

```
test@232-22:~/creatures$ git merge dragon
Mise à jour dcadf59..5564a8e
Fast-forward
 fred.txt | 3 +++
 seb.txt  | 3 +++
```

```
 2 files changed, 6 insertions(+)
 create mode 100644 fred.txt
 create mode 100644 seb.txt
```

Je peux voir en dessous que mes fichiers sont bien arrivés dans la branche master :

```
test@232-22:~/creatures$ ls
dwarf.txt  fred.txt  hobbit.txt  seb.txt
```

3) Je peux supprimer la branch dragon avec la commande suivante :

```
test@232-22:~/creatures$ git branch -d dragon
Branche dragon supprimée (précédemment 5564a8e).
```

Il a été possible de la supprimer car nous l'avons mergé juste avant sur notre branche principal.

4) Travail avec un dépôt distant

1) Mon voisin crée un user nommé tp_git (non privilégié). Il le configure de façon à ce que je puisse le ssh avec ma clé rsa

Ensuite moi je crée une clé rsa (publique et privée) :

```
test@232-22:~/creatures$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/test/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/test/.ssh/id_rsa.
Your public key has been saved in /home/test/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Xt3RegTkLI0C8zBsqXthhzqbQO31UPuiU8lec2P6gys test@202-15
The key's randomart image is:
+---[RSA 2048]----+
|      .=.   .o  |
|       +*.  = o |
|    . o oo.o = o|
|   . o * oo o + |
|  . . =S*.o. o .|
|   . =...* + +. |
|    . =.+ o * . |
|     o o E o .  |
|       . ..o.. |
+----[SHA256]-----+
```

Je lui donne ma clé publique pour qu'il puisse me ssh :

```
test@232-22:~$ sudo scp /home/test/.ssh/id_rsa.pub
tp_git@10.202.16.1:/home/tp_git/
tp_git@10.202.16.1's password:
id_rsa.pub                                    100%  393   286.4KB/s   00:00
```

La clé publique il la copie dans /home/test/.ssh/authorized_keys

2) **Sur le serveur :**

Je me connecte au serveur via ssh.

Dans le serveur on crée un dossier .git :

```
mkdir test.git
```

Ensuite on se déplace dedans :

```
cd test.git
```

Maintenant j'initie un dépôt git dans le dossier :

```
git init --bare
```

Sur le client :

On se connecte au serveur avec la commande suivante :

```
git remote add [nom] login@IP:[chemin vers le repo git du serveur]
```

Une fois connecté on a plus qu'à push nos commit :

```
git push [nom] [branche]
```

Si je fais maintenant un git log sur le serveur je peux voir tous mes commit :

```
tp_git@202-16:~/tp.git$ git log
commit 99c2bab53324b11cb789564276cc204c765fd019 (HEAD -> master)
Author: test <test@244-0>
Date:   Tue Nov 10 17:08:21 2020 +0100

    commit serveur

commit 5564a8e6fcd32720c8f021bec5d77ec37e66c231
Author: test <test@244-0>
Date:   Tue Nov 10 15:53:19 2020 +0100
```

```
    fichier seb
```

3) Je fais une modification sur mon dépôt en cahngeant le contenu du fichier test.txt :

```
test@232-22:~/creatures$ vim test.txt
```

Maintenant je l'add et je le commit :

```
test@232-22:~/creatures$ git add test.txt
test@232-22:~/creatures$ git commit -m "nouveau test"
[master 3af76e2] nouveau test
 Committer: test <test@244-0>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant sur votre nom d'utilisateur et le nom de votre machine. Veuillez
vérifier qu'ils sont corrects. Vous pouvez supprimer ce message en les
paramétrant explicitement. Lancez les commandes suivantes et suivez les
instruction dans votre éditeur pour éditer votre fichier de configuration :

    git config --global --edit

Après ceci, vous pouvez corriger l'identité utilisée pour cette validation avec :

    git commit --amend --reset-author

 1 file changed, 1 insertion(+), 1 deletion(-)
```

Une fois commit je peux push le fichier vers le serveur :

```
test@232-22:~/creatures$ git push salut master
tp_git@10.202.16.1's password:
Énumération des objets: 5, fait.
Décompte des objets: 100% (5/5), fait.
Compression par delta en utilisant jusqu'à 8 fils d'exécution
Compression des objets: 100% (2/2), fait.
Écriture des objets: 100% (3/3), 273 bytes | 273.00 KiB/s, fait.
Total 3 (delta 1), réutilisés 0 (delta 0)
To 10.202.16.1:/home/tp_git/tp.git
   99c2bab..3af76e2  master -> master
```

Sur le serveur après avoir push :

```
tp_git@202-16:~/tp.git$ git log
commit 3af76e21fc96a2b38d7b88242ede6e9756024788 (HEAD -> master)
Author: test <test@244-0>
Date:   Tue Nov 10 17:33:09 2020 +0100

    nouveau test
```

On peut faire la manipulation plusieurs fois et on peut push plusieurs commit en même temps dans un seul push

4) Je me crée un nouveau dossier créature_copy sur le client puis dedans je clone le dépôt git du serveur :

```
test@232-22:~/creatures_copy$ git clone
tp_git@10.202.16.1:/home/tp_git/tp.git
Clonage dans 'tp'...
tp_git@10.202.16.1's password:
remote: Énumération des objets: 23, fait.
remote: Décompte des objets: 100% (23/23), fait.
remote: Compression des objets: 100% (14/14), fait.
remote: Total 23 (delta 3), réutilisés 0 (delta 0)
Réception d'objets: 100% (23/23), fait.
Résolution des deltas: 100% (3/3), fait.
```

Dans le dossier **créatures** je modifie le fichier **dwarf.txt** et dans le dossier **créatures_copy** je modifie le fichier **hobbit.txt**

Maintenant je push sur le serveur le commit du fichier dwarf.txt :

```
test@232-22:~/creatures$ git add dwarf.txt
test@232-22:~/creatures$ git commit -m "commit dwarf"
[master 9f4b1c2] commit dwarf
 Committer: test <test@244-0>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant sur votre nom d'utilisateur et le nom de votre machine. Veuillez
vérifier qu'ils sont corrects. Vous pouvez supprimer ce message en les
paramétrant explicitement. Lancez les commandes suivantes et suivez les
instruction dans votre éditeur pour éditer votre fichier de configuration :

    git config --global --edit

Après ceci, vous pouvez corriger l'identité utilisée pour cette validation avec :

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)

test@232-22:~/creatures$ git push salut master
tp_git@10.202.16.1's password:
Énumération des objets: 5, fait.
Décompte des objets: 100% (5/5), fait.
Compression par delta en utilisant jusqu'à 8 fils d'exécution
Compression des objets: 100% (3/3), fait.
```

```
Écriture des objets: 100% (3/3), 307 bytes | 307.00 KiB/s, fait.
Total 3 (delta 1), réutilisés 0 (delta 0)
To 10.202.16.1:/home/tp_git/tp.git
   3af76e2..9f4b1c2  master -> master
```

Maintenant j'essaye de push sur le serveur le commit du fichier hobbit.txt via le dossier créatures_copy :

```
test@232-22:~/creatures_copy/tp$ git remote add salut
tp_git@10.202.16.1:/home/tp_git/tp.git/
test@232-22:~/creatures_copy/tp$ git push salut master
tp_git@10.202.16.1's password:
To 10.202.16.1:/home/tp_git/tp.git/
 ! [rejected]        master -> master (fetch first)
error: impossible de pousser des références vers
'tp_git@10.202.16.1:/home/tp_git/tp.git/'
astuce: Les mises à jour ont été rejetées car la branche distante contient du
travail que
astuce: vous n'avez pas en local. Ceci est généralement causé par un autre
dépôt poussé
astuce: vers la même référence. Vous pourriez intégrer d'abord les
changements distants
astuce: (par exemple 'git pull ...') avant de pousser à nouveau.
astuce: Voir la 'Note à propos des avances rapides' dans 'git push --help' pour
plus d'information.
```

Je ne peux pas push sur les serveur mon commit car il y a un travail sur le serveur que je n'ai pas en local donc je dois d'abord me mettre à jour pour push.

5) Pour cette question les deux personnes travaillent sur le même fichier "seb.txt" et sur la même ligne. Je modifie exactement la même ligne dans le même fichier et je tente de push sur le serveur

Sur une personne je remplace hostile par "Gentil" tandis que sur l'autre je met "Mechant"

```
root@debian:~/creatures# vim seb.txt
root@debian:~/creatures# root@debian:~/creatures# git add seb.txt
root@debian:~/creatures# git commit -m "commit 2 seb"
[master f67296a] commit 2 seb
 Committer: root <root@debian>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant
sur votre nom d'utilisateur et le nom de votre machine. Veuillez vérifier qu'ils
sont corrects.
Vous pouvez supprimer ce message en les paramétrant explicitement :

    git config --global user.name "Votre Nom"
    git config --global user.email vous@exemple.com
```

```
Après ceci, vous pouvez corriger l'identité utilisée pour ce commit avec :

    git commit --amend --reset-author

 1 file changed, 1 insertion(+), 1 deletion(-)
root@debian:~/creatures# git push origin master
root@192.168.1.59's password:
Énumération des objets: 5, fait.
Décompte des objets: 100% (5/5), fait.
Compression des objets: 100% (2/2), fait.
Écriture des objets: 100% (3/3), 279 bytes | 279.00 KiB/s, fait.
Total 3 (delta 1), réutilisés 0 (delta 0)
To 192.168.1.59:/root/tp/
   3f0ae7a..f67296a  master -> master
```

Sur la 1ère personne cela fonctionne bien maintenant je test avec la 2ème personne :

```
root@debian:~/creatures_copy/tp# vim seb.txt
root@debian:~/creatures_copy/tp# git add seb.txt
root@debian:~/creatures_copy/tp# git commit -m "commit 4 seb"
[master 4f5dc2b] commit 4 seb
 Committer: root <root@debian>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant
sur votre nom d'utilisateur et le nom de votre machine. Veuillez vérifier qu'ils
sont corrects.
Vous pouvez supprimer ce message en les paramétrant explicitement :

    git config --global user.name "Votre Nom"
    git config --global user.email vous@exemple.com

Après ceci, vous pouvez corriger l'identité utilisée pour ce commit avec :

    git commit --amend --reset-author

 1 file changed, 1 insertion(+), 1 deletion(-)
root@debian:~/creatures_copy/tp# git push origin master
root@192.168.1.59's password:
To 192.168.1.59:/root/tp/
 ! [rejected]        master -> master (fetch first)
error: impossible de pousser des références vers 'root@192.168.1.59:/root/tp/'
astuce: Les mises à jour ont été rejetées car la branche distante contient du
travail que
astuce: vous n'avez pas en local. Ceci est généralement causé par un autre
dépôt poussé
```

```
astuce: vers la même référence. Vous pourriez intégrer d'abord les
changements distants
astuce: (par exemple 'git pull ...') avant de pousser à nouveau.
astuce: Voir la 'Note à propos des avances rapides' dans 'git push --help' pour
plus d'information.
```

Ça n'a pas fonctionné et nous avons eu le même problème qu'à la question précédente. Pour résoudre le problème il faut faire un git pull de ce qui se trouve sur le serveur pour mettre à jour.

5) Branches distantes

Vous décidez de gérer deux versions de votre projet sur votre serveur. Une version principale et une version de developpement («dev» pour les intimes). Montrez comment vous créez cette branche supplémentaire sur le serveur et comment vos deux utilisateurs peuvent travailler sur l'une ou l'autre branche. Vous donnerez le plus de détails possibles.

Pour commencer je dois aller sur le serveur et créer la branche dev (pour le développement) :

```
root@debian:~/tp# git branch dev
```

Une fois la branche crée je dois faire un git pull sur les deux personnes pour mettre à jour le dépôt :

```
root@debian:~/creatures# git pull origin master
root@192.168.1.59's password:
Depuis 192.168.1.59:/root/tp
 * branch            master     -> FETCH_HEAD
Déjà à jour.
```

Maintenant on peut se déplacer dans la branche dev pour travailler sur le développement avant de remettre sur la branche principale :

```
root@debian:~/creatures# git checkout dev
La branche 'dev' est paramétrée pour suivre la branche distante 'dev' depuis
'origin'.
Basculement sur la nouvelle branche 'dev'
```

```
root@debian:~/creatures# git branch
* dev
  master
```

Maintenant sur la branche dev je peux faire toutes les modifications que je veux (ici je crée un fichier test.txt) et je peux tout push dans le serveur sur la branche dev :

```
root@debian:~/creatures# vim test.txt
```

```
root@debian:~/creatures# root@debian:~/creatures# git add test.txt
root@debian:~/creatures# git commit -m "commit test branche dev"
[dev 08c5d2e] commit test branche dev
 Committer: root <root@debian>
Votre nom et votre adresse courriel ont été configurés automatiquement en se
fondant
sur votre nom d'utilisateur et le nom de votre machine. Veuillez vérifier qu'ils
sont corrects.
Vous pouvez supprimer ce message en les paramétrant explicitement :

    git config --global user.name "Votre Nom"
    git config --global user.email vous@exemple.com

Après ceci, vous pouvez corriger l'identité utilisée pour ce commit avec :

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
root@debian:~/creatures# git push origin dev
root@192.168.1.59's password:
Énumération des objets: 4, fait.
Décompte des objets: 100% (4/4), fait.
Compression des objets: 100% (2/2), fait.
Écriture des objets: 100% (3/3), 295 bytes | 295.00 KiB/s, fait.
Total 3 (delta 1), réutilisés 0 (delta 0)
To 192.168.1.59:/root/tp/
   f67296a..08c5d2e  dev -> dev
```

Si on regarde sur la branche master de mon client on ne doit pas voir le fichier test.txt :

```
root@debian:~/creatures# git checkout master
Basculement sur la branche 'master'
root@debian:~/creatures# ls
dwarf.txt  fred.txt  hobbit.txt  seb.txt
```

Sur le client toutes les modifications dans la branche dev ont été faites et si je veux le remettre sur la branche master je fais la commande suivante :

```
 root@debian:~/creatures# git branch
  dev
* master
root@debian:~/creatures# ls
dwarf.txt  fred.txt  hobbit.txt  seb.txt
root@debian:~/creatures# git merge dev
Mise à jour f67296a..08c5d2e
```

```
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode
root@debian:~/creatures# ls
dwarf.txt  fred.txt  hobbit.txt  seb.txt  test.txt 100644 test.txt
```

Je peux finir en mettant à jour les données sur le serveur :

```
root@debian:~/creatures# git push origin master
root@192.168.1.59's password:
Énumération des objets: 1, fait.
Décompte des objets: 100% (1/1), fait.
Écriture des objets: 100% (1/1), 227 bytes | 227.00 KiB/s, fait.
Total 1 (delta 0), réutilisés 0 (delta 0)
To 192.168.1.59:/root/tp/
   9554a45..0d7d796  master -> master
```

6) Le stash

1) Git stash est une commande de Git qui permet de "**mettre de côté**" des modifications. Cette fonctionnalité est très pratique, notamment lorsque le développeur a besoin de changer de branche rapidement.

Le stash est le container dans lequel les modifications sont sauvegardées.

Il existe plusieurs commandes pour le stash :

La sauvegarde de mes modifications :

```
test@232-22:~/creatures$ git stash save
Copie de travail et état de l'index sauvegardés dans WIP on master: 5564a8e
fichier seb
```

Pour reprendre mes modifications je fais la commande suivante :

```
test@232-22:~/creatures$ git stash pop
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        nouveau fichier : test.txt

refs/stash@{0} supprimé (142f6ad499a9f227a4951af7086b80b4b81a1ba0)
```

Ou alors si on a plusieurs save en attente :

```
test@232-22:~/creatures$ git stash pop stash@{1}
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        nouveau fichier : test.txt

stash@{1} supprimé (132a7de16cab5f1898a9116123b8357d2256ef6b)
```

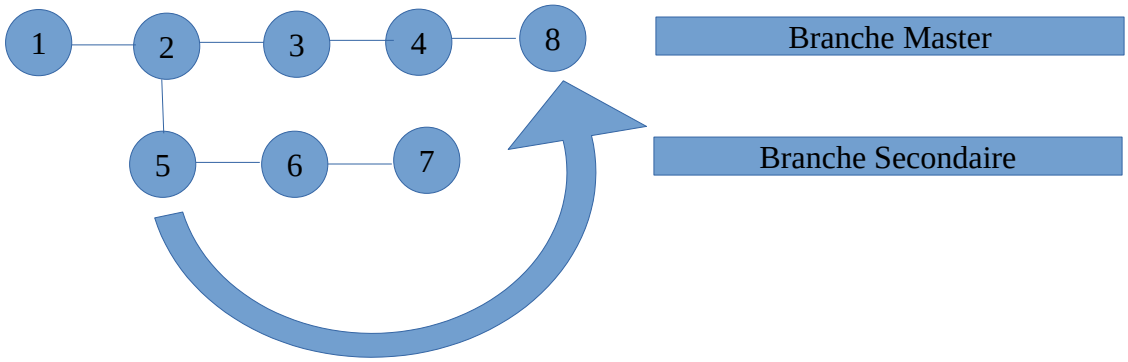
Le dernier stash sauvegardé prend la valeur du stash{0} et les autres montent tous de "1"

On peut aussi voir un stash en cours pour savoir ce qu'il y a dedans :

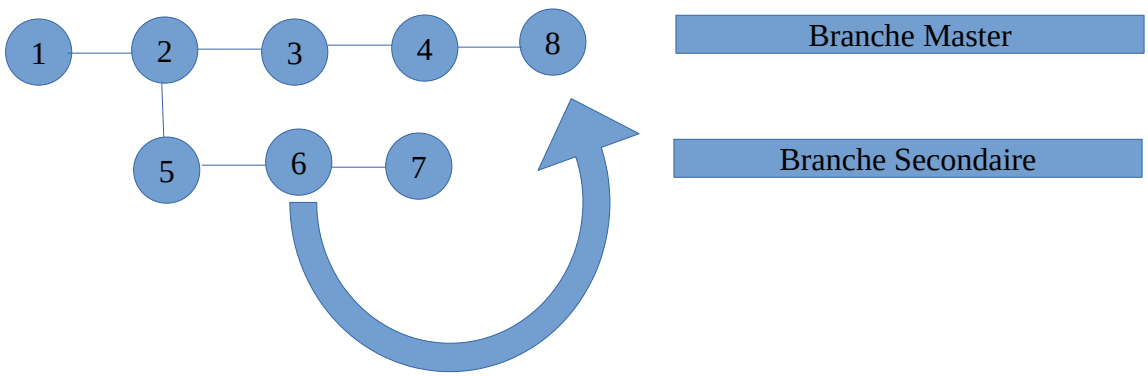
```
test@232-22:~/creatures$ git stash show stash@{0}
 coucou.txt | 1 +
 1 file changed, 1 insertion(+)
```

7) commandes avancées

1) git rebase : cette commande permet de récupérer tous les commit d'une branche et de les rajouter à la fin de la branche master



2) git cheery-pick : cette commande permet de ne récupérer qu'un seul commit et de le mettre au bout de la branche master



3) Voici le contenu de mon dossier .git contenu à la racine de mon dépôt :

```
root@debian:~/creatures# cd .git/
root@debian:~/creatures/.git# ls
branches  COMMIT_EDITMSG  config  description  FETCH_HEAD  HEAD  hooks
index  info  logs  objects  ORIG_HEAD  refs
```

Il contient toutes sortes d'infos comme les branches ou les commit da façon caché.

4) Pour ignorer les fichiers .bin il faut créer un répertoire .gitignore où l'on va mettre .bin dedans

5) La configuration se fait au niveau du serveur :

Il faut d'abord récupérer le fichier python git_multimail :

```
cd /root/tp.git
cd hooks/
curl https://raw.githubusercontent.com/git/git/master/contrib/hooks/multimail/
git_multimail.py >git_multimail.py
```

Dans le même répertoire il faut créer un fichier post-receive :

```
#!/bin/sh

exec python2 "$(dirname "$0")"/git_multimail.py
```

Il faut rendre le fichier exécutable sinon lors du push il y aura une erreur :

```
chmod +x post-receive
```

Il faut maintenant configurer le git-multimail

```
git config multimailhook.mailingList 'exemple@gmail.com,
mon.binome@gmail.com'
```

Maintenant quand on fera un git push sur le serveur on recevra un mail