Logiciel de versionning : git

1) Qu'est-ce qu'un logiciel de versionning

Un logiciel de versionning (VCS en anglais pour Version Constrol System) est un outil très utile dans le milieu du développement Web. Il permet de garder une trace des versions des fichiers que l'on modifie, avec une chronologie. On peut donc reprendre la version d'un fichier d'il y a plusieurs jours, à condition que le logiciel de version était déjà présent et configuré.

2) Présentation de git

On télécharge le paquet git, disponible via apt-get ou même Synaptic.

```
root@poste61:/home/hugo/public_html/site_perso# apt-get install git
```

Une fois installé, on doit configurer l'utilisateur avec une adresse mail et un nom

```
hugo@poste61:~/public_html/site_perso$ git config --global user.email "hugo.nitard@gmail.com"
```

```
hugo@poste61:~/public_html/site_perso$ git config --global user.name "Hugo"
```

Une fois configuré, on peut créer un nouveau dépôt. Pour cela, on se met dans un nouveau répertoire, puis on tape la commande git init. Les fichiers nécessaires au fonctionnement de git seront créés lorsque la commande sera effectuée.

Une fois le travail terminé, il faut dire à Git d'ajouter les fichiers créés.

```
hugo@poste61:~/public_html/test_git$ git add test.txt
```

Lorsque tous les fichiers que vous voulez versionner sont ajoutés (on peut tout ajouter d'un coup avec git add –all), il faut commit.

Commit peut être traduit par la création d'une version à laquelle tous les fichiers que l'on vient d'ajouter feront parti. On lui transmet un message, afin de pouvoir garder un meilleur suivi :

```
hugo@poste61:~/public_html/test_git$ git commit -m "v1"
[master (commit racine) 1bb4a1c] v1
 1 file changed, 2 insertions(+)
 create mode 100644 test.txt
```

Notre fichier test.txt est maintenant versionné.

On va remodifier le fichier, refaire un commit, et revenir en arrière, en supposant que l'on ait fait une erreur, et que l'on ne puisse pas faire Ctrl + Z.

Avec la commande git status, on peut voir quels fichiers ont été changés, ajoutés ou supprimés depuis le dernier commit :

```
hugo@poste61:~/public_html/test_git$ 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 :         test.txt

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

Avec la commande git log, on voit toutes les versions.

```
hugo@poste61:~/public_html/test_git$ git log
commit 680b21013853eeba8252193b5a4aa4b99419476a
Author: Hugo <hugo.nitard@gmail.com>
Date:   Mon Mar 26 12:45:40 2018 +0200

    v2

commit 1bb4a1c1742e64f61b4940a0388ec5a6d8da2e56
Author: Hugo <hugo.nitard@gmail.com>
Date:   Mon Mar 26 12:40:45 2018 +0200

    v1
```

On souhaite revenir à la première version.

```
hugo@poste61:~/public_html/test_git$ git revert 680b21013853eeba8252193b5a4aa4b99419476a
[master ebc55f6] Revert "v2"
 1 file changed, 1 insertion(+), 1 deletion(-)
```

On retrouve alors notre fichier comme au début.

3) Gestion simple d'un projet à plusieurs collaborateurs grâce à GitHub

GitHub est un service Web permettant de décentraliser son dépôt Git.

Pour cela, on se rend sur le site de GitHub, on se connecte, et on créé un nouveau dépôt.

Ici, j'en ai créé un appelé test.

Afin de fusionner mon répertoire local avec celui de GitHub, nous allons taper :

```
git remote add origin https://github.com/Alyndel/test.git
git push -u origin master
```

Git remote va initialiser la connexion entre le dépôt GitHub et le dépôt local.

La branche origin sera créée sur le dépôt GitHub.

Git push permet d'envoyer le contenu d'un commit au serveur de GitHub.

Pour qu'un collaborateur récupère le travail déjà créé, il aura besoin de la commande suivante :

```
hugo@poste61:~/public_html/test_clonage$ git clone https://github.com/Alyndel/test.git
Clonage dans 'test'...
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 7 (delta 1), reused 7 (delta 1), pack-reused 0
Dépaquetage des objets: 100% (7/7), fait._
```

Le répertoire clone est maintenant opérationnel. Tout changement effectué devra faire l'objet d'un commit, qui sauvegardera les changements au niveau local, que l'on accompagnera d'un git push, afin que toutes les personnes puisse disposer des changements.

Dans le cas où un fichier a été modifié puis push par deux personnes en même temps, la deuxième personne à l'avoir push devra effectué une gestion de conflits :

```
hugo@poste61:~/public_html/test_git$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Dépaquetage des objets: 100% (3/3), fait.
Depuis https://github.com/Alyndel/test
   ebc55f6..81c65d9  master     -> origin/master
Fusion automatique de test.txt
CONFLIT (contenu) : Conflit de fusion dans test.txt
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

Lors de l'édition du fichier, il ressemble désormais à ceci :

```
<<<<<<< HEAD
il a aussi été modifié par la premiere personne entre temps
=======
fichier modifié avec le clone
>>>>>>> 81c65d9a2862d35b617fec6c48ec6a31f9b4ac46
gdhfjjghdfkgjhdkjg
```

Dans la partie HEAD, c'est ce que la personne a édité, dans la partie entourée des « = » et du commit, on voit la partie modifiée par l'autre personne. Dans le cas où les deux modifications n'influent pas sur l'autre, nous allons simplement retirer HEAD, ====== et >>>>>>.

On doit simplement recommit la gestio nde conflits, et le push.

```
hugo@poste61:~/public_html/test_git$ git add --all
hugo@poste61:~/public_html/test_git$ git commit -m "gestion des conflits"
[master 216d918] gestion des conflits
hugo@poste61:~/public_html/test_git$ git push
Username for 'https://github.com': Alyndel
Password for 'https://Alyndel@github.com':
Décompte des objets: 6, fait.
Delta compression using up to 4 threads.
Compression des objets: 100% (4/4), fait.
Écriture des objets: 100% (6/6), 679 bytes | 0 bytes/s, fait.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/Alyndel/test.git
   81c65d9..216d918  master -> master_
```

Maintenant, allons voir sur GitHub le résultat :

```
1
2  il a aussi été modifié par la premiere personne entre temps
3
4  fichier modifié avec le clone
5
6  gdhfjjghdfkgjhdkjg
```

Lors d'un travail collaboratif, il peut être intéressant que deux équipes travaillent sur des branches différentes, afin que le travail de l'un ne puisse pas altérer l'autre.

La commande git branch permet de créer une nouvelle branche, et donc une arborescence et des commits indépendants de la première branche.

On se place dans la branche créée grâce à Git Checkout :

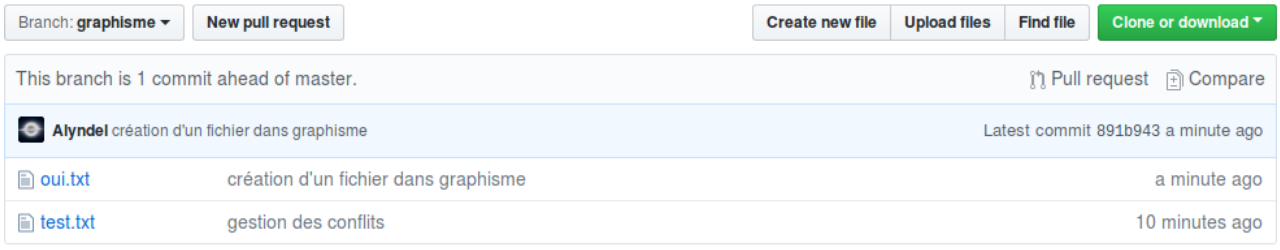
```
hugo@poste61:~/public_html/test_git$ git branch graphisme
hugo@poste61:~/public_html/test_git$ git checkout graphisme
```

On doit spécifier de quelle branche découle graphisme avec :

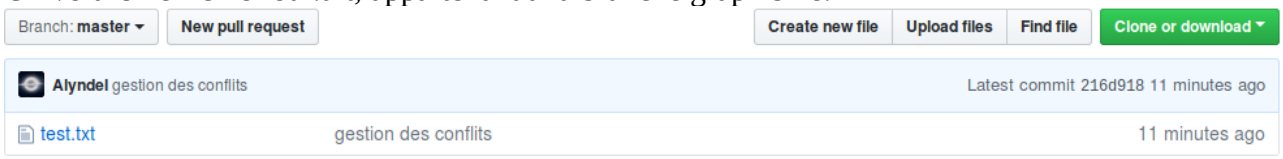
```
hugo@poste61:~/public_html/test_git$ git push --set-upstream origin graphisme
Username for 'https://github.com': Alyndel
Password for 'https://Alyndel@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/Alyndel/test.git
 * [new branch]      graphisme -> graphisme
La branche graphisme est paramétrée pour suivre la branche distante graphisme depuis origin.
```

On y créé un fichier appelé oui.txt, on commit et on push.

Sur github, on regarde le résultat :



On voit ici le fichier oui.txt, appartenant à la branche graphisme.



Dans la branche master, le fichier oui.txt n'apparaît pas.

Les branches sont donc bien indépendantes.

Lors de la réunion de deux branches, il peut y avoir des conflits. Ils seront gérés de la même manière qu'une gestion de conflits classique.

```
hugo@poste61:~/public_html/test_git$ git checkout master
Basculement sur la branche 'master'
Votre branche est à jour avec 'origin/master'.
hugo@poste61:~/public_html/test_git$ git merge graphisme
Mise à jour 216d918..891b943
Fast-forward
 oui.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 oui.txt
hugo@poste61:~/public_html/test_git$ ls
oui.txt  test.txt
```

Le fichier oui.txt se trouve maintenant dans les branches master et graphisme.