Introduction à Git

20 septembre 2021

Table des matières

Première partie
Introduction

1 Les gestionnaires de versions

Un gestionnaire de version est un outil permettant de maintenir plusieurs versions, c'est-à-dire qu'il est capable d'enregistrer les différents états d'un ou de plusieurs fichiers au cours du temps et de naviguer facilement entres ses états.

Il en existe plusieurs, les plus connus étant *mercurial*, *SVN* et *Git*. Nous nous intéresserons uniquement à ce dernier qui est notamment utilisé pour la réalisation de Linux.

2 Git, GitLab, GitHub : fonctionnement et différences

Vous qui lisez ce tutoriel, vous avez sûrement déjà entendu parler de GitHub et/ou GitLab alors que dans ce poly, nous allons uniquement vous détailler git.

Pourquoi ? Tout simplement parce-que git est en fait le coeur du fonctionnement de GitHub et GitLab *(ce n'est pas rien qu'ils s'appelent tous les deux Gitquelque-chose...)*. En effet, tout le fonctionnement détaillé dans ce poly est le même pour ces trois outils, alors quelle sont les différences ?

Pour faire court git a un fonctionnement qui ne nécessite aucune centralisation, ou autrement dit aucun serveur ou tout le monde rassemble son travail. Mais la plupart du temps, les équipes préférent avoir un dépôt cental où l'on peut retrouver le projet avec les dernières modifications. Et c'est avec cet esprit que GitHub est né. En plus de cela, sans compte payant, les projets qui utilisent GitHub, sont entièrement accessible par tous (Open Source), par contre, pour avoir un dépôt privé, il faut avoir un compte payant.

Mais dans ce cas là, comment peut faire une petite équipe de développeurs ou même une très grosse entreprise qui désire utiliser Git avec un outil clés en main mais avec des dépôts privés ET sans payer ? C'est là où est arrivé GitLab. Ce dernier outil propose un outil avec autant (ou presque ou plus selon les années...) que GitHub mais de façon entièrement Open Source. Ainsi, n'importe qui peut installer un serveur GitLab pour lui ou son équipe

et en faire ce qu'il veut (public et/ ou privé). GitLab possède aussi des fonctionnalités payantes non Open-Source.

Dernier *détail*, le logiciel Git est Open Source (c'est grâce à cela que les deux autres outils peuvent se baser dessus) et n'appartient pas à une entreprise, contrairement à GitLab qui est une société et GitHub qui appartient à Microsoft depuis 2018.

Deuxième partie
Installation

L'installation de Git diffère en fonction de votre système d'exploitation :

3 Linux

Pour installer Git sur un système Linux il suffit d'exécuter la commande suivante :

```
$ sudo apt-get install git
```

A savoir que Git est installé par défaut sur certains systèmes Linux. Si vous voulez aussi une interface graphique, plutôt que d'utiliser git exclusivement en ligne de commande vous pouvez aussi installer "gitk" en exécutant :

```
$ sudo apt-get install gitk
```

4 Windows

Pour les systèmes Windaub, il faut installer *msysgit* qui installera un terminal Unix permettant d'utiliser les commandes Linux de Git (et donc de suivre la suite du tuto) ainsi que les paquets Git essentiels à son bon fonctionnement, pour installer msysgit reportez-vous au tutoriel fourni sur http://msysgit.github.io

5 MacOS

Il y a plusieurs façons d'installer Git sous Mac OS X. Le plus simple est de se baser sur cet installeur pour Mac OS X :
https://code.google.com/archive/p/git-osx-installer/.

MacOS utilisant un terminal Linux nous pourrons utiliser les commandes données dans ce tuto sous MacOS.

Troisième partie

Configuration de Git

Avant de commencer à utiliser Git il vous faudra créer une configuration basique de Git, pour cela il vous faudra seulement exécuter quelques commandes dans un terminal Linux afin de configurer *votre* Git en ajoutant votre nom et votre mail :

```
$ git config --global user.name "VotrePseudo"
$ git config --global user.email moi@email.com
```

Quatrième partie
Création de dépôt local

6 Clonage d'un dépôt distant

La manière la plus fréquente de créer un dépôt (*repository*) Git est le "clonage", c'est-à-dire la copie entière d'un projet géré avec Git. Si l'on prend par exemple le projet de ce cours même qui est stocké sur GitHub à l'adresse https://github.com/ClubNix/tuto-git, il suffit de taper la commande suivante pour copier le projet entier dans le dossier courant :

```
$ git clone https://github.com/ClubNix/tuto-git.git
```

Il est aussi possible de manière facultative de spécifier dans quel dépôt le clonage va se faire :

```
$ git clone https://github.com/ClubNix/tuto-git.git
   dossier
```

L'URL de clonage vous est normalement fournie par le service que vous utilisez pour héberger votre code (GitHub, GitLab ou autre). Attention : il est possible que l'URL de clonage ne soit pas la même que l'URL utilisée pour voire votre projet sur votre navigateur internet.

7 Création d'un dépôt vide

Il est bien sûr possible avec Git de créer un dépôt vide. Pour cela, il suffit de faire la commande suivante dans le dossier créé pour stocker le dépôt Git :

```
$ git init
```

Cela va tout simplement ajouter les fichiers propres à Git dans le répertoire courant ainsi qu'initialiser la gestion de version.

Notez qu'il est souvent plus simple de créer un dépôt vide directement sur GitHub ou GitLab, puis de le cloner, car Git sera à ce moment là assez malin pour "lier" votre dépôt local à celui distant.

Cinquième partie

Exemple d'un workflow avec Git

Avec Git, un workflow courant, est constitué de 4 étapes : la modification, la validation, le commit et l'envoi. Nous allons voir chacune des étapes dans cette partie.

8 Modification

La modification est tout simplement le moment pendant lequel les fichiers (de code, etc...) sont modifiés. C'est la partie qui semble la plus simple dans la gestion de version. Cependant, afin d'exploiter la gestion de version à son plein potentiel, certaines règles doivent être respectées.

Premièrement, Git analyse les différents changement en faisant des "diff", qui consistent à regarder le différences entre un même fichier à des versions antérieures. Ainsi, afin de clarifier un peu le tout, il est conseillé de préférer du texte sur plusieurs lignes que du texte sur une seule ligne.

Ensuite, afin de pouvoir mieux gérer son dépôt mais aussi de mieux gérer la stabilité et la fonctionnalité du projet, il est aussi conseillé de maximiser le nombre de versions et d'en valider une aussitôt que le projet "fonctionne". Cela facilitera grandement la détection et la résolution d'erreurs mais aussi le travail avec les branches que l'on verra plus tard.

Sixième partie

Quelle procédure suivre pour enregistrer vos fichiers

La procédure pour utiliser Git va être la suivante :

- On ajoute tous les fichiers qui seront à enregistrer sur le dépôt distant grâce à la commande "add"
- On "commit" tous les changements de fichier sur notre dépôt local, cela aura pour effet de les enregistrer en créant une version locale du projet en attente de l'envoi sur le dépôt distant,
- On "push" tout notre commit sur le dépôt distant afin d'enregistrer nos modifications sur celui-ci, la version locale précédemment crée est envoyée et devient similaire à la version distante.

Nous verrons en détail toutes ces commandes dans la suite de ce tuto, il est à noter que certaines commandes de vérifications que nous verrons peuvent être utilisées entre chaque étape afin de vérifier les fichiers que vous envoyez ou qu'il n'y a tout simplement pas d'erreur.

Septième partie

Les commandes Git de base

Arrivé à ce stade, vous savez créer un nouveau dépôt ou en cloner un existant afin de travailler dessus, nous allons donc maintenant voir les commandes de base pour effectuer des "sauvegardes" de votre dossier Git :

9 Obtenir de l'aide

Si vous voulez des informations sur une commande particulière, vous pouvez utiliser la commande "git help". Par exemple, si vous voulez de l'aide sur la commande "git status" (que nous allons voir juste après), vous pouvez exécuter dans un terminal :

```
$ git help status
```

Git est très bien documenté, alors n'hésitez pas.

10 Modifier le code et effectuer des commits

10.1 git status

Lorsque vous modifiez les fichiers contenus dans votre dépôt Git vous devez les envoyer sur le serveur afin de les sauvegarder pour l'ensemble des développeurs, pour cela vous pouvez utiliser la commande suivante afin de voir les fichiers qui ont été modifiés mais dont les modifications ne sont pas enregistrées sur le dépôt :

```
$ git status
```

Pour aller plus loin nous pouvons même voir en détail les modifications non enregistrées grâce à la commande suivante :

```
$ git diff
```

10.2 git commit

Maintenant que vous savez quelle modification sont a enregistrer nous pouvons passer au commit de ces changements. Pour cela plusieurs methodes sont possible :

La première va consister à ajouter les fichiers qui ont été modifiés dans le tas des fichiers à commit :

```
$ git add nomfichier1 nomfichier2
```

vous pouvez aussi ajouter tous les fichiers en même temps en utilisant :

```
$ git add .
```

Puis de commit ces fichiers grâce à la commande :

```
$ git commit
```

Cette commande va tout d'abord vous demander d'écrire un message pour votre commit. Nous ne nous étendrons pas dessus ici, mais, un message de commit doit être clair et concis. Quelqu'un qui n'était pas avec vous quand vous avez fait un certain commit doit pouvoir comprendre le but et l'intérêt du commit uniquement grâce à son message.

Par défaut git commit ouvrira un éditeur de texte intégré au terminal pour que vous le tapiez. Pour éviter cela vous pouvez passer le message en paramètre de la commande commit ainsi :

```
$ git commit -m "Votre message clair et concis."
```

Le commit aura pour effet de créer une sauvegarde/version du dépôt local sur votre ordinateur, sauvegarde que vous pourrez envoyer sur le dépôt distant par la suite.

NB : Après un *git add ...* si vous faites un *git status* vous verrez les fichiers ajoutés apparaître en vert.

La deuxième méthode consiste à "commiter" tous les fichiers qui étaient listés dans *git status* dans les colonnes "Changes to be committed" et "Changed but not updated" (qu'ils soient en vert ou en rouge) grâce à la commande :

```
$ git commit -a
```

Cette dernière ligne est donc équivalente à la suite suivante :

```
$ git add .
$ git commit
```

La troisième solution va consister à indiquer à Git quels fichiers doivent être commit :

```
$ git commit nomFichier1 nomFichier2
```

10.3 Annuler un commit

Dans le cas où vous auriez commité des erreurs il est encore possible d'annuler le dernier commit effectué.

Pour cela nous allons commencer par vérifier les logs :

```
$ git log
```

Vous devriez normalement voir tous les derniers commit effectués. Chaque commit est numéroté grâce à un long numéro hexadécimal comme *12328a1bc...* Cela permet de les identifiers.

Pour aller plus loin vous pouvez voir les lignes modifiées par le commit en utilisant la commande suivante :

```
$ git log -p
```

Vous pouvez aussi avoir un résumé plus court des commits avec :

```
$ git log --stat
```

Dans le cas où vous auriez fait une faute dans le message de votre dernier commit il existe une commande permettant d'ouvrir un éditeur de texte pour le modifier :

```
$ git commit --amend
```

Mais attention, il est en effet impossible de modifier le message d'un commit lorsque celui-ci a été transmis à d'autres personnes. Elle s'utilise daonc avant le push.

Pour annuler votre dernier commit vous pouvez utiliser la commande suivante :

```
$ git reset HEAD^
```

Cela annule le dernier commit et revient à l'avant-dernier.

Pour indiquer à quel commit on souhaite revenir, il existe plusieurs notations :

```
-HEAD : dernier commit ;
-HEAD^ : avant-dernier commit ;
-HEAD^^ : avant-avant-dernier commit ;
-HEAD~2 : avant-avant-dernier commit (notation é
   quivalente) ;
-d6d98923868578a7f38dea79833b56d0326fcba1 : indique un
   numéro de commit précis.
```

Seul le commit est retiré de Git ; vos fichiers, eux, restent modifiés. Vous pouvez alors à nouveau changer vos fichiers si besoin et refaire un commit.

Vous pouvez aussi annuler un commit et *toutes* les modifications qu'il implique grâce à la commande :

```
$ git reset --hard HEAD^
```

10.4 Restaurer un fichier à son état antérieur avant le commit

Si vous avez modifié plusieurs fichiers mais que vous n'avez pas encore envoyé le commit et que vous voulez restaurer un fichier tel qu'il était au dernier commit, utilisez :

```
$ git checkout nomfichier
```

Il est possible de retirer un fichier qui avait été ajouté pour être "commité" en procédant comme suit :

```
$ git reset HEAD -- fichier_a_supprimer
```

Ces deux dernières commandes peuvent se retrouver dans le commentaires affichés par la commande git status.

11 Partager votre travail et télécharger les nouveautés

Sous Git, vous travaillez sur un dépôt local, mais tous les fichiers du projet sont aussi contenus sur un dépôt distant dans lequel tous les contribueurs peuvent faire des modifications. Il sera donc nécessaire de savoir télécharger les nouveautés apportées par d'autres mais aussi de savoir envoyer vos nouveautés sur le dépôt distant.

11.1 Télécharger les nouveautés

Pour télécharger les nouveautés du dépôt il vous suffit d'utiliser la commande suivante :

```
$ git pull
```

Deux cas sont possibles :

- Soit vous n'avez effectuée aucune modification depuis le dernier pull, dans ce cas la mise à jour est simple (on parle de mise à jour fast-forward) ;
- Soit vous avez fait des commits en même temps que d'autres personnes. Les changements qu'ils ont effectués sont alors fusionnés aux vôtres automatiquement.

Si deux personnes modifient en même temps deux endroits distincts d'un même fichier, les changements sont intelligemment fusionnés par Git.

Parfois, mais cela arrive normalement rarement, deux personnes modifient la même zone de code en même temps. Dans ce cas, Git dit qu'il y a un conflit car il ne peut décider quelle modification doit être conservée ; il vous indique alors le nom des fichiers en conflit. Ouvrez-les avec un éditeur et recherchez une ligne contenant "`<<<<<<<`", "`=======`" ou "`>>>>>>>`". Ces symboles délimitent vos changements et ceux des autres personnes. Supprimez ces symboles et gardez uniquement les changements nécessaires, puis faites un nouveau commit pour enregistrer tout cela.Si jamais vous oubliez quoi faire lors d'une résolution de conflit, la commande git status vous donnera des indices.

11.2 Envoyer vos commit

Avant d'envoyer quoi que ce soit nous vous conseillons fortement de vérifier ce que vous allez envoyer grâce à un *git log -p*, mais aussi de télécharger ce

qui se trouve sur le dépôt distant car le serveur ne peut régler les conflits à votre place s'il y en a. Personne ne doit avoir fait un push avant vous depuis votre dernier pull. Au pire des cas, le push peut tout simplement refusé sans rien casser.

Une fois cela fait vous pouvez utiliser la commande suivante pour publier vos commit :

```
$ git push
```

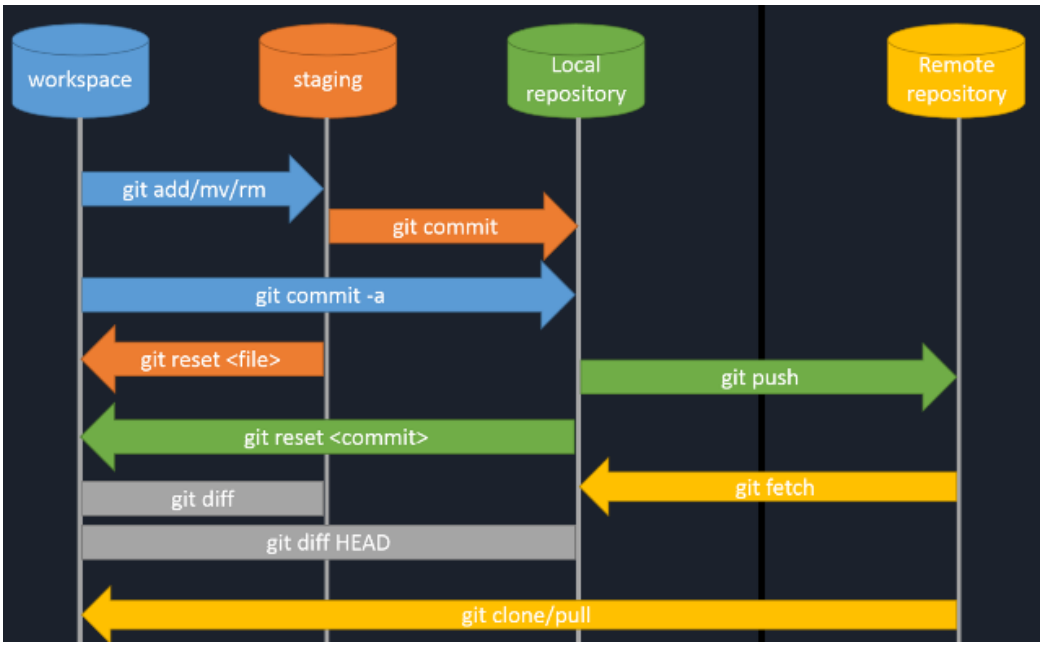
11.3 Annuler un commit publié

Si par mégarde vous avez publié des erreurs vous pouvez toujours annuler le dernier commit publié, pour cela vous devez connaitre le numéro hexadécimal de votre commit grâce à *git log*, ensuite vous pouvez utiliser la commande suivante pour annuler votre commit :

```
$ git revert numeroHexadecimal
```

Cette commande va simplement réaliser un nouveau commit, inverse à celui que vous passez en commentaire. Exemple : si le premier commit ajoute une ligne contenant "bonjour", le "revert" sera un commit supprimant cette ligne, et le résultat sera comme si rien ne c'était passé.

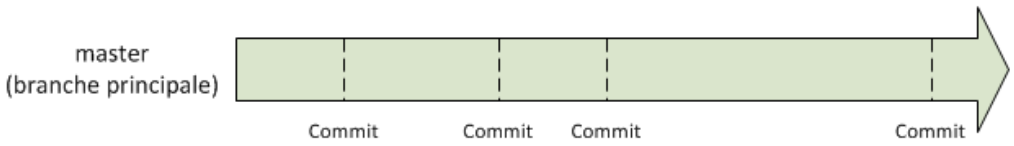
Voici un petit schéma résumant le fonctionnement des commandes vues :



12 Travailler avec des branches

Les branches font parties du cœur même de Git et constituent un de ses principaux atouts. C'est un moyen de travailler en parallèle sur d'autres fonctionnalités. C'est comme si vous aviez quelque part une "copie" du code source du projet qui vous permettrait de tester vos idées les plus folles et de vérifier si elles fonctionnent avant de les intégrer au véritable code source de votre projet.

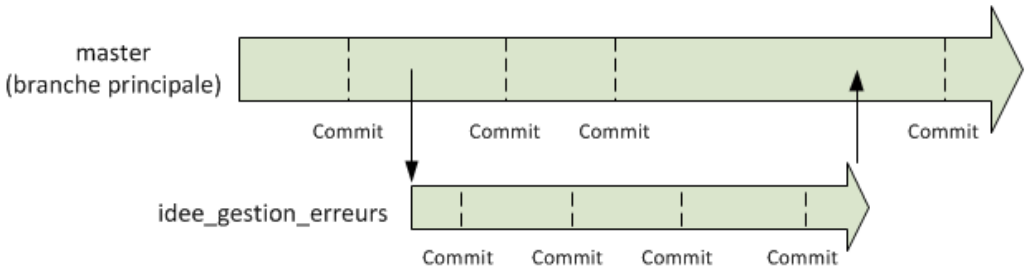
Dans Git, toutes les modifications que vous faites au fil du temps sont par défaut considérées comme appartenant à la branche principale appelée "master" :



Supposons que vous ayez une idée pour améliorer la gestion des erreurs dans votre programme mais que vous ne soyez pas sûrs qu'elle va fonctionner :

vous voulez faire des tests, ça va vous prendre du temps, donc vous ne voulez pas que votre projet incorpore ces changements dans l'immédiat.

Il suffit de créer une branche, que vous nommerez par exemple "idee_gestion_erreurs", dans laquelle vous allez pouvoir travailler en parallèle :



À un moment donné, nous avons décidés de créer une nouvelle branche. Nous avons pu y faire des commits, mais cela ne nous a pas empêché de continuer à travailler sur la branche principale et d'y faire des commits aussi.

Git gère tous ces problèmes pour vous. Au lieu de créer une copie des fichiers, il créé juste une branche "virtuelle" dans laquelle il retient vos changements en parallèle. Lorsque vous décidez de fusionner une branche (et donc de ramener vos changements dans "master" pour les valider), Git vérifie si vos modifications n'entrent pas en conflit avec des commits effectués en parallèle. S'il y a des conflits, il essaie de les résoudre tout seul ou vous avertit s'il a besoin de votre avis (c'est le cas si deux personnes ont modifiées la même ligne d'un même fichier par exemple).

Ce concept de branches très légères qui ne nécessitent pas de copier les fichiers est d'une grande puissance. Cela vous encourage à créer des branches tout le temps, pour toutes les modifications qui pourraient prendre du temps avant d'être terminées.

Vous pouvez même créer une sous-branche à partir d'une branche !

12.1 Les branches locales

Tout le monde commence avec une seule branche "master" : c'est la branche principale. Jusqu'ici, vous avez donc travaillé dans la branche "master", sur le "vrai" code source de votre projet.

Pour voir toutes vos branches, tapez ceci :

```
$ git branch
```

La branche sur laquelle vous vous trouvez est indiquée par une étoile *.

12.1.1 Pourquoi créer une branche et quand la créer ?

Lorsque vous vous apprêtez à faire des modifications sur le code source, posez-vous les questions suivantes :

— Ma modification sera-t-elle rapide ?
— Ma modification est-elle simple ?
— Ma modification nécessite-t-elle un seul commit ?
— Est-ce que je vois précisément comment faire ma modification d'un seul coup ?

Si la réponse à l'une de ces questions est "non", vous devriez probablement créer une branche. Créer une branche est très simple, très rapide et très efficace. Il ne faut donc pas s'en priver.

12.1.2 Créer une branche et changer de branche

Imaginons que vous vouliez créer un tutoriel Web sur Git et l'insérer sur votre site internet. Vous n'êtes pas sûrs du temps que cela va prendre, ce n'est pas un changement simple qui consiste à modifier deux-trois liens et vous risquez de faire plusieurs commits. Bref, il faut créer une branche pour cela.

```
$ git branch Tuto_Git
```

Cette commande va créer une branche du nom de *Tuto_Git*, il est à noter que cette branche est locale (vous seul y avez accès), il est aussi possible de la partager sur le dépôt distant.

Vous devriez maintenant voir votre nouvelle branche en faisant un *git branch*, mais vous n'êtes pas encore sur votre nouvelle branche. Pour changer de branche il vous faudra taper la commande suivante :

```
$ git checkout Tuto_Git
```

Qu'est-ce qui se passe lorsque l'on change de branche ? En fait, vous ne changez pas de dossier sur votre disque dur, mais Git change vos fichiers pour qu'il reflète l'état de la branche dans laquelle vous vous rendez. Imaginez que les branches dans Git sont comme des dossiers virtuels : vous "sautez" de l'un à l'autre avec la commande *git checkout*. Vous restez dans le même dossier, mais Git modifie les fichiers qui ont changés entre la branche où vous étiez et celle où vous allez.

Sachant que si vous avez fait des modifications non-enregistrées, Git va être gentil et ne va pas écrire par dessus lors d'un checkout.

12.1.3 Fusionner les changements

Lorsque vous avez fini de travailler sur une branche et que celle-ci est concluante, il faut "fusionner" cette branche vers "master". Pour cela commencer par vous rendre sur votre branche "master", puis exécutez la commande suivante :

```
$ git merge Tuto_Git
```

Cette commande aura pour effet de fusionner les changements effectués dans votre branche "Tuto_Git" dans votre branche "master", mais pas de supprimer cette branche qui ne sert plus a rien. Pour supprimer vos anciennes branches il vous suffira d'exécuter la commande suivante :

```
$ git branch -d Tuto_Git
```

Git vérifie que votre travail dans la branche "Tuto_Git" a bien été fusionné dans "master". Sinon, il vous en avertit et vous interdit de supprimer la branche (vous risqueriez sinon de perdre tout votre travail dans cette branche !).

Si vous souhaitez supprimer une branche sans en sauvegarder les changements vous pouvez utiliser cette commande à la place :

```
$ git branch -D Tuto_Git
```

12.2 Les branches partagées

Il est possible de travailler à plusieurs sur une même branche. En fait, c'est déjà ce que vous faisiez en travaillant sur la branche "master".

Pour voir toutes les branches partagées du dépôt distant il faut utiliser la commande suivante :

```
$ git branch -r
```

Si le serveur possède une autre branche, par exemple "origin/brancheserveur", et que vous souhaitez travailler dessus, il suffit d'utiliser la commande suivante :

```
$ git checkout brancheserveur
```

Lorsque vous ferez un pull depuis la branche "brancheserveur", les changements seront fusionnés dans votre "brancheserveur" local. Il est donc important de savoir dans quelle branche vous vous trouvez avant de faire un pull. Un pull depuis la branche "master" met à jour votre branche "master" locale en fonction de ce qui a changé sur le serveur, et il en va de même pour n'importe quelle autre branche.

Voyons maintenant comment créer et supprimer vos propres branches partagées. Si vous avez créé une branche localement, et que vous voulez "push" les modifications sur le serveur il vous suffit de faire :

```
$ git push --set-upstream origin ma_nouvelle_branche
```

Les options de "git push" signifient que votre nouvelle branche sera liée à votre dépôt distant, représenté par le mot "origin". Une fois cette liaison faite, les commandes "git push" et "git pull" fonctionneront automatiquement sans options, pourvu que vous soyez sur la bonne branche.

Enfin voyons comment supprimer une branche partagée du serveur, pour cela il suffit d'utiliser la commande suivante :

```
$ git push --delete origin nom_branche_a_supprimer
```

À noter que cette commande supprime uniquement la branche sur le serveur. Il vous faudra tout de même supprimer votre branche locale comme vu à la section précédente.

Huitième partie

Conclusion

Maintenant vous devriez avoir une idée de ce qu'est Git et de comment il va pouvoir vous être utile !

Vous devriez être capable de connaître les opérations basiques : créer ou cloner un dépôt, effectuer un commit ou des modifications et être capable de visualiser les modifications de chaque version dans un même projet. Nous avons enfin couvert la création de branches, le changement de branche et la fusion de branches locales. Vous devriez aussi être en mesure de partager vos branches sur un serveur et de travailler à plusieurs sur des branches partagées.

Le Club*Nix reste à votre disposition si vous avez besoin d'aide ou désirez poser une question.

Bibliographie

— LE site de l'origine avec les documentations originelles : http://git-scm.com

Et de la lecture en anglais

Si nous n'avons pas été clair, deux résumés :

— http://rogerdudler.github.io/git-guide/
— https://www.atlassian.com/git/

Un guide pour écrire de bons messages de commit :
https://juffalow.com/other/write-good-git-commit-message

Sachant que vous pouvez TOUT trouver sur Git, du bon comme du mauvais, amusez vous bien \o/