**Utilisation de GIT**Ghislain RABETRANO  
[2017-07-18](file:///D:\Switzernet\howto\git\index1.docx)

Utilisation de GIT

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Auteur** | **Date** | **Remarque** |
| 1.0 | Ghislain RABETRANO | 2017-07-18 | Création du document |
|  |  |  |  |
|  |  |  |  |

Table des matières

[1. Une histoire de bonnes pratiques 1](#_Toc488251017)

[1.1. La mauvaise façon de travailler 1](#_Toc488251018)

[1.2. La bonne façon de travailler 1](#_Toc488251019)

[2. GIT, approche globale 1](#_Toc488251020)

[2.1. Étapes classiques à suivre 1](#_Toc488251021)

[2.2. Exemple de communication client/serveur 1](#_Toc488251022)

[2.3. Les branches 1](#_Toc488251023)

[3. Les logiciels 1](#_Toc488251024)

[3.1. En ligne de commande 1](#_Toc488251025)

[3.1.1. Clé SSH 1](#_Toc488251026)

[3.1.2. Publier son travail 1](#_Toc488251027)

[3.1.3. Récupérer la dernière version sur le serveur 1](#_Toc488251028)

[3.2. Avec une interface graphique (Préparation) 1](#_Toc488251029)

[3.2.1. Installation de Git pour Windows 1](#_Toc488251030)

[3.2.2. Clé SSH 1](#_Toc488251031)

[3.3. Utilisation de « Git UI » 1](#_Toc488251032)

[3.3.1. Publier son travail 1](#_Toc488251033)

[3.3.2. Récupérer la dernière version sur le serveur 1](#_Toc488251034)

[3.3.3. L’historique des modifications 1](#_Toc488251035)

[4. Les branches 1](#_Toc488251036)

[4.1. En ligne de commande 1](#_Toc488251037)

[4.1.1. Publier son travail 1](#_Toc488251038)

[4.1.2. Changer de branche 1](#_Toc488251039)

[4.1.3. Fusionner 2 branches 1](#_Toc488251040)

[4.2. Utilisation de Git UI 1](#_Toc488251041)

[4.2.1. Publier son travail 1](#_Toc488251042)

[4.2.2. Changer de branche 1](#_Toc488251043)

[4.2.3. Fusionner 2 branches 1](#_Toc488251044)

# Une histoire de bonnes pratiques

## La mauvaise façon de travailler

Avant de voir une bonne façon de travailler, voyons la mauvaise manière avec l’exemple d’un site web :

* Quand l’application web évolue, les développeurs vont modifier les pages PHP, JS, CSS ...
* Pour cela, ils vont se connecter en SSH sur le serveur et faire ces modifications.
* Une fois qu’un des développeurs a quitté SSH, il n’y a aucune trace de ce qu’il a fait et encore pire : aucune trace du code passé.
* Le grand problème est que si l’équipe veut voir et comprendre l’historique des modifications, ou même faire une marche arrière… ils seront très embêtés.

## La bonne façon de travailler

Dans l’idéal et dans la mesure du possible :

* On rapatrie les fichiers dans un environnement de dev (ex : sa propre machine).
* On fait les modifications et on teste dans cet environnement.
* Quand les tests sont bons, on déploie les fichiers dans l’environnement de production.

Le grand principe à retenir est : on ne travaille jamais directement sur les fichiers en production.

Pour cela, il faut au moins 2 choses :

* Développer de sorte que le code ne soit pas prisonnier d’un environnement (avoir un fichier de configuration qui permet de changer les connexions aux répertoires, à la base de données, aux comptes Paypal …)
* Avoir un outil qui permette de faire passer les fichiers d’un environnement à l’autre facilement.

Ce document a donc pour but d’expliciter ce dernier point.  
L’un des bons outils qui permet de bien gérer les différentes versions de fichiers, de les rapatrier/déployer, de travailler à plusieurs personnes est actuellement GIT.

# GIT, approche globale

## Étapes classiques à suivre

* Le développeur s’assure auprès de l’administrateur qu’il a bien les droits suffisants pour travailler sur le projet (passage de clé SSH …)
* Le développeur rapatrie le dépôt, c’est-à-dire le projet et ses fichiers, sur son ordinateur (machine locale) en utilisant Git
* Le développeur effectue les changements
* Le développeur envoi les changements sur le serveur git
* L’administrateur déploie le code sur le(s) serveur(s) de test et de production (sur le serveur, il rapatrie les fichiers avec git).
* L’administrateur aura évidemment la perspicacité de créer des scripts pour tout cela …

## Exemple de communication client/serveur

GIT est un environnement client/serveur.

Cela signifie que le serveur stocke les fichiers avec ces versions que les uns et les autres auront publiées.

Ceci dit, GIT stocke beaucoup d’information en local, dans le répertoire « .git » qui se trouve à la racine du projet.  
Dans ce répertoire se trouve 2 entités :

* INDEX :  
  Au départ, lorsqu’on crée un fichier, il est simplement dans l’un des répertoires du projet, mais il n’est pas encore indexé par Git.  
  Lorsqu’on l’ajoute à Git, le fichier est dans l’index. Git le considère, mais il n’est pas dans une des versions du projet (il est hors version)
* HEAD  
  Lorsqu’on valide un fichier, celui est indexé dans le Head. C’est l’espace où les fichiers sont stockés de manière versionnée.

Cette architecture permet des opérations rapides comme la recherche de différence entre 2 versions ou la fusion de 2 versions.  
Si ça peut vous rassurer, sachez que vous n’aurez jamais à travailler dans le répertoire « .git ».

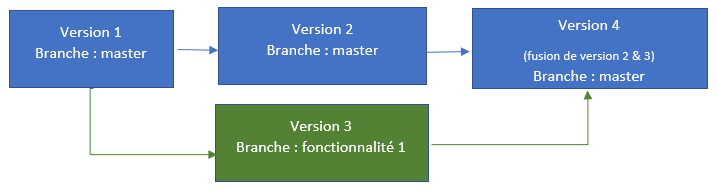
Voici un exemple simple de communication entre le client et le serveur.  
Nous sommes dans le cas où un dépôt existe déjà sur le serveur distant et l’utilisateur veut le récupérer pour travailler dessus :

|  |  |  |
| --- | --- | --- |
| **C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture39.png** | **git clone <URL>** | Récupération du dépôt : Tous les fichiers + le répertoire « .git ». |
| **git add -A** | Indexation de tous les fichiers |
| **git commit -m « my comment »** | Création d’une version en local avec un commentaire. Les fichiers indexés sont dans le HEAD. |
| **git push** | Envoi de la version auprès du serveur |

## Les branches

Comme nous l’avons vu, un fichier se trouve dans une version. Mais, il faut savoir qu’une version est elle-même dans une branche spécifique.  
La branche par défaut est se nomme « master » (mais on peut changer par la suite).

Par exemple, on peut vouloir créer une branche parallèle car un autre développeur doit programmer des fonctionnalités spécifiques.  
Plus tard, quand les différentes branches sont testées et validées, on peut les fusionner, c’est-à-dire fusionner les répertoires, les fichiers et le contenu des fichiers.



# Les logiciels

GIT est un outil qui se gère en ligne de commande. Étant devenu très populaire, on a vu apparaitre un bon nombre d’outils avec une interface graphique.

## En ligne de commande

Quel intérêt y a-t-il à utiliser les lignes de commande alors qu’il existe d’outils graphiques ?

Il y a 2 principales raisons :

* Git a été conçu pour être gérer en ligne de commande. Donc on a une grande maîtrise/compréhension d’actions à mener.
* Dans certains cas, il n’y a pas d’interface graphique (serveur Linux) ! Donc, il vaut mieux connaître les lignes de commande.

Dans mon exemple, j’utilise Cygwin. Si vous voulez faire comme moi, il faut au préalable installer les paquets « git » pour Cygwin.

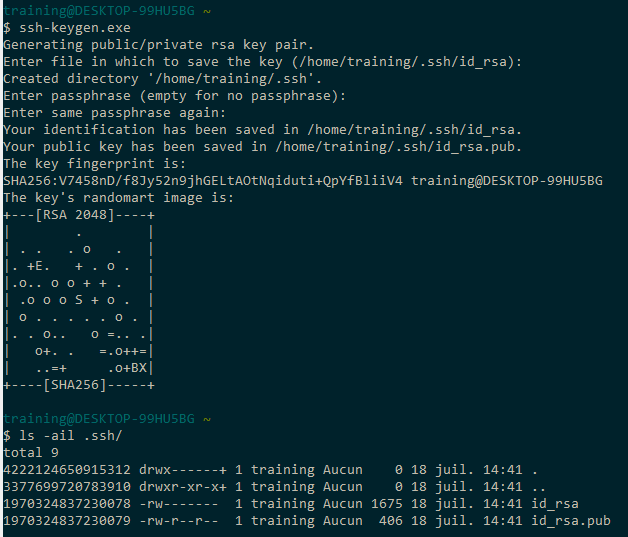
### Clé SSH

Il y a 2 manières d’attaquer un serveur git :

* Soit par HTTP
* Soit par SSH

Si on veut juste récupérer un code, la liaison par HTTP peut suffire. Mais si on veut faire des modifications, il vaut mieux passer par SSH.

Pour générer cette clé, dans Cygwin, il faut taper « ssh-keygen.exe » et l’envoyer à l’administrateur.  
Vous pouvez l’afficher en tapant « more ~/.ssh/id\_rsa.pub ».  
Copiez la clé et envoyez-la à l’administrateur.



### Publier son travail

Nous nous plaçons dans le cas où une personne veut récupérer la 1re fois le projet, travailler dessus et le publier sur le serveur.

|  |  |
| --- | --- |
| **~/trainings**  **$** git clone git.switzernet.com:training/howto.git  Clonage dans 'howto'...  remote: Counting objects: 3, done.  remote: Total 3 (delta 0), reused 0 (delta 0)  Réception d'objets: 100% (3/3), fait. | * Récupération du dépôt git.switzernet.com:training/howto.git * Git va créer dans le répertoire «  howto », avec tous les fichiers du projet |
| **~/trainings**  **$** cd howto/ | Pour travailler sur le projet, j’entre dans le nouveau répertoire créé |
| **~/trainings/howto**  **$** git config --global user.name "Ghislain RABETRANO"  **~/trainings/howto**  **$** git config --global user.email "ghislain.rabetrano@switzernet.com" | Avant de continuer, il est bon de décliner son nom et son adresse email |
| **~/trainings/howto**  **$** touch my\_first\_file.txt | À cette étape, on fait le travail sur nos fichiers (création, mise à jour, suppression …. de fichiers ou de répertoires)  En l’occurrence, nous créons un fichier vide qui s’appelle « my\_first\_file.txt ». |
| **~/trainings/howto**  **$** git status  Sur la branche master  Votre branche est à jour avec 'origin/master'.  Fichiers non suivis:  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)  my\_first\_file.txt  aucune modification ajoutée à la validation, mais des fichiers non suivis sont présents (utilisez "git add" pour les suivre) | À tout moment, je peux afficher le statut de mon répertoire.  Dans notre cas, nous constatons que :   * Nous sommes dans la branche par défaut : master * Le fichier my\_first\_file.txt n’est pas dans l’index donc encore moins dans une version   Remarque : « origin » est un alias local qui fait référence à l’URL. Ça évite de taper à chaque fois l’URL du dépôt ([git.switzernet.com:training/howto.git](mailto:git@git.switzernet.com:training/howto.git)). |
| **~/trainings/howto**  **$** git add -A | * Ajout de tous les fichiers du répertoire à l’index de git (-A signifie « all files ») * remarque : Nous aurions pu ajouter les fichiers manuellement en tapant :   git add my\_first\_file.txt |
| **~/trainings/howto**  **$** git status  Sur la branche master  Votre branche est à jour avec 'origin/master'.  Modifications qui seront validées :  (utilisez "git reset HEAD <fichier>..." pour désindexer)  nouveau fichier : my\_first\_file.txt | Si nous affichons de nouveau le statut de notre dépôt, nous constatons que :   * Le fichier my\_first\_file.txt est bien dans l’index * Par contre, le fichier n’est pas dans une version car il fait partie des nouveaux fichiers |
| **~/trainings/howto**  **$** git commit -m "my comment"  [master cf2d7eb] my comment  1 file changed, 0 insertions(+), 0 deletions(-)  create mode 100644 my\_first\_file.txt | * Création d’une version que je nomme/commente par « my comment » * On constate que Git a trouvé un fichier dans l’index pour cette version   Remarque : nous sommes bien dans la branche « master » |
| **~/trainings/howto**  **$** git status  Sur la branche master  Votre branche est en avance sur 'origin/master' de 1 commit.  (utilisez "git push" pour publier vos commits locaux)  rien à valider, la copie de travail est propre | maintenant, tous les fichiers sont indexés et sont dans une version |
| **~/trainings/howto**  **$** git diff master origin/master  diff --git a/my\_first\_file.txt b/my\_first\_file.txt  index e69de29..bac9c71 100644  +++ a/my\_first\_file.txt  @@ -0,0 +1 @@  + | En tapant « git diff », nous voyons la différence entre notre dépôt local (HEAD) et le serveur. |
| **~/trainings/howto**  **$** git push  Décompte des objets: 2, fait.  Delta compression using up to 8 threads.  Compression des objets: 100% (2/2), fait.  Écriture des objets: 100% (2/2), 242 bytes | 0 bytes/s, fait.  Total 2 (delta 1), reused 0 (delta 0)  To git.switzernet.com:training/howto.git  8e619cc..6e3a121 master -> master | « git push » permet l’envoi de la version du dépôt local (HEAD) au serveur |
| **~/trainings/howto**  **$** git diff master origin/master | Cette fois-ci, il n’y plus de différence entre le dépôt local et le serveur |

### Récupérer la dernière version sur le serveur

Si nous avons déjà une copie du projet et qu’une autre personne a travaillé dessus et a envoyé son travail au serveur, nous pouvons récupérer les modifications avec un « pull ». Ainsi, cette commande permet de récupérer la dernière version sur le serveur.

|  |  |
| --- | --- |
| **~/trainings/howto**  **$** git pull | Cette commande permet de récupérer la dernière version qui est sur le serveur. |

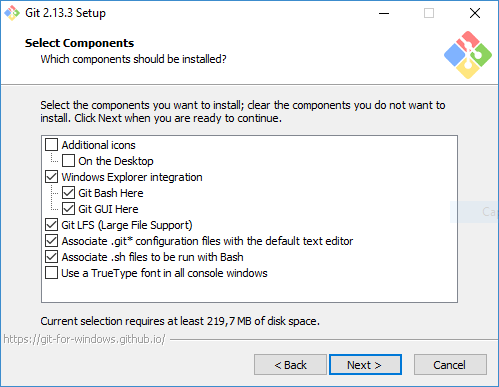
## Avec une interface graphique (Préparation)

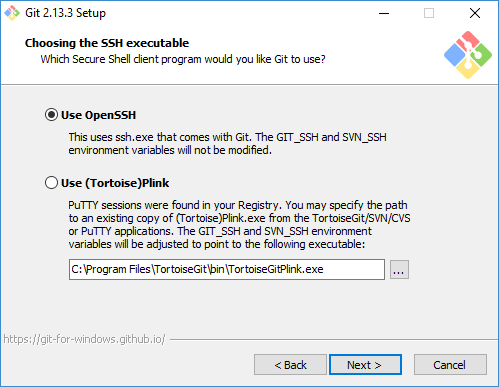
### Installation de Git pour Windows

Nous allons faire la même chose avec une interface graphique dans un environnement Windows 10.

Pour cela, nous aurons besoin du logiciel Git pour Windows (<https://git-scm.com/download/win>)

Pendant l’installation, gardez les paramètres par défaut et assurez-vous de bien prendre les utilitaires « Git Bash here » et « Git GUI here » ainsi que d’utiliser OpenSSH :



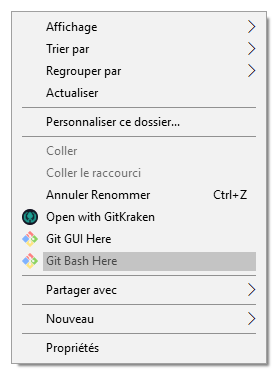


### Clé SSH

Si ce n’est pas encore fait, nous allons commencer par créer une clé SSH.

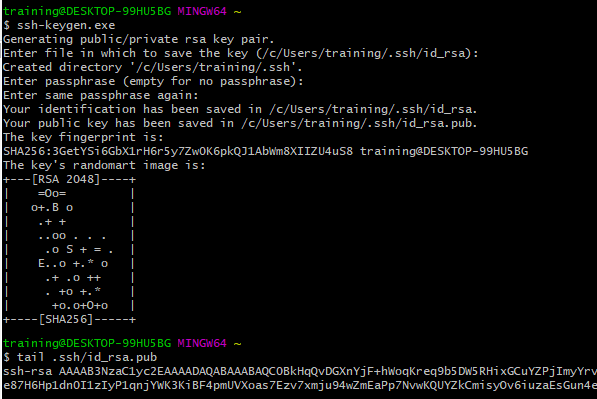
*Remarquez que cette clé n’est malheureusement pas la même que celle générée par Cygwin.*

Pour en créer une, allez dans l’Explorer de Windows (dans un n’importe quel répertoire) et faites un clic droit et cliquez sur « Git Bash here ».



Vous vous retrouverez dans un environnement en ligne de commande de type Linux. Le principe sera donc le même qu’avec Cygwin :

Il faut taper « ssh-keygen.exe » et envoyer la clé publique générée à l’administrateur.  
Vous pouvez l’afficher en tapant « tail ~/.ssh/id\_rsa.pub ». Copiez la clé et envoyez-la à l’administrateur.



## Utilisation de « Git UI »

### Publier son travail

Nous allons donc refaire le même travail que ce que nous avons fait en ligne de commande, mais cette fois-ci, avec une interface graphique.

Rappel : Avant d’attaquer cette étape, il est nécessaire que l’administrateur vous ait donné les droits !

|  |  |
| --- | --- |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture08.png | Après avoir installé le logiciel Git pour Windows :   * Placez-vous dans le répertoire où vous allez accueillir le nouveau projet * Faites un clic droit et cliquez sur « Git GUI Here » |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture09.png | * Une fois que vous avez cliqué sur « Git GUI Here », cette fenêtre va alors s’ouvrir. * Cliquez alors sur « Clone Existing Repository » |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture10.png | * Dans « Source location », saisissez l’URL du dépôt que l’administrateur vous a communiqué * Dans Target Directory, saisissez le répertoire où vous allez mettre le projet.   Remarque : Ce répertoire ne doit pas exister (ou il doit être vide) |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture11.png | Si c’est la première fois que vous vous connectez au serveur, vous aurez cette fenêtre.  Il suffit simplement de taper « yes »  Remarque : Si vous n’arrivez pas à vous connecter, c’est parce qu’il y a un problème avec la clé SSH. |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture12.png | Si tout s’est bien passé :   * Le répertoire que vous avez spécifié est créé. Dans mon cas, je l’ai nommé « howto ». * Si vous rentrez dans ce répertoire, vous verrez les fichiers du projet.   Modification sur le projet :   * Avec un éditeur de texte quelconque, j’ai créé le fichier « my\_second\_file.txt ».   Prise en compte des modifications :   * Si nous cliquons sur « Rescan », dans « Git GUI », nous verrons que ce fichier viendra s’inscrire dans la section « Unstaged changes ». Cela signifie qu’il n’est pas indexé et encore moins positionné dans une version |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture14.png | * En cliquant sur « Stage Changed », il va changer de catégorie et passer dans la section « Staged Changes ». * Si vous avez bien suivi, nous avons fait un « git add … » |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture15.png | Si nous tentons de faire un commit tout de suite, nous aurons cette erreur.  En effet, nous devons spécifier notre identité. |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture16.png | Pour corriger cela :   * Dans le menu de « Git GUI », allez dans « Edit » -> « Option ». Renseignez votre nom et votre email pour le dépôt courant et les options globales |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture17.0.png | Maintenant, nous pouvons faire un commit.  Remarque : Un message est obligatoire. |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture18.png | Git GUI demande de confirmer la branche en vue de faire un « push », c’est-à-dire la publication de notre version sur notre serveur. |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture19.png | Si tout s’est bien passé, nous avons envoyé notre version au serveur. |

### Récupérer la dernière version sur le serveur

Si vous avez déjà le dépôt sur votre machine et que vous voulez récupérer la dernière version mise sur le serveur, vous devez aussi faire « pull ».

Mais avec « Git GUI », on ne peut pas le faire directement. On doit faire un « fetch » et un « merge ».

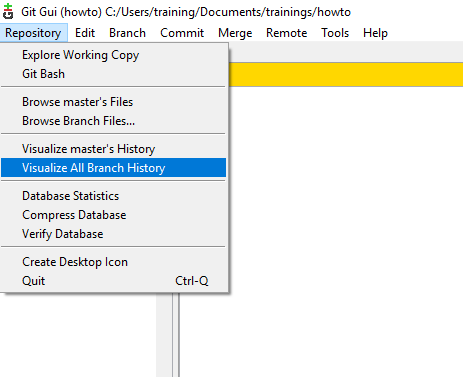
|  |  |
| --- | --- |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture20.png | **Git fetch :**  En ligne de commande, nous aurions tapé « git fetch origin ».  Cette commande récupère auprès du serveur toutes les données du projet dans le HEAD que nous ne possédons pas (fichiers, modifications dans un fichier, branches …).  Pour rappel, « origin » est un alias local qui fait référence à l’URL ([git.switzernet.com:training/howto.git](mailto:git@git.switzernet.com:training/howto.git)). |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture21.png | **Git merge :**  En ligne de commande, nous aurions tapé « git merge ».  Cette commande fusionne les données dans le HEAD et notre répertoire local.  C’est ainsi qu’on aura les dernières modifications visibles dans le répertoire local. |

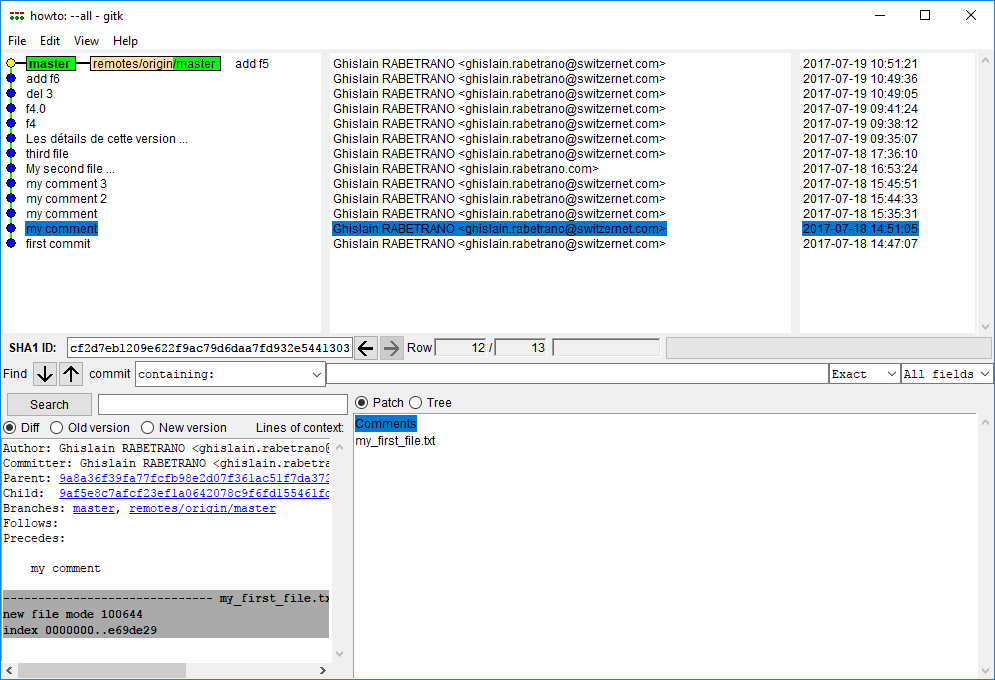
### L’historique des modifications

Quel est l’intérêt d’avoir une interface graphique ?

* Quand on a bien compris les lignes de commandes, il faut reconnaître que c’est plus « user-firendly ».
* Un vrai avantage est celui de mieux voir l’historique du projet

L’interface graphique permet donc de voir l’historique des modifications de manière très lisible.  
Dans Git GUI, nous trouvons cela dans Repository > Vizualize All Branch History





# Les branches

Comme nous l’avons vu, nous pouvons être amenés à faire 2 développements en parallèle.

* Exemple 1 :  
  L’équipe frontend travaille sur l’aspect graphique tandis que l’équipe backend travaille sur les API.  
  Si les 2 chantiers n’ont rien à voir l’un avec l’autre, il peut être judicieux que l’une des 2 équipes (voir les 2 en fait) crée une branche à part et quand elle aura validé son code, elle fusionnera cette branche avec la branche MASTER.
* Exemple 2 :  
  Une équipe effectue un travail à haut risque. Créer une branche permettrait de faire une validation par une autre équipe sans bousculer tout le code.

D’une manière générale, il faut créer une branche quand on veut tenter une évolution sans être certain de la garder ou non.

## En ligne de commande

### Publier son travail

|  |  |
| --- | --- |
| **~/trainings/howto**  **$** git branch -l  \* master | Avec « git branch -l », nous listons toutes les branches existantes. |
| **~/trainings/howto**  **$** git checkout -b a\_new\_branch  Basculement sur la nouvelle branche 'a\_new\_branch' | * Création d’une nouvelle branche qui se nomme « a\_new\_branch » * Bascule vers cette branch.   Tout notre travail va être gardé dans la branche « a\_new\_branch » |
| **~/trainings/howto**  **$** git branch -l  \* a\_new\_branch  master | Si nous relistons les branches existant, nous constatons que :   * Il y a 2 branches qui existent dans notre HEAD * L’étoile indique que nous travaillons pour l’heure avec la branche « a\_new\_branch »   Remarque : Tant que nous ne faisons pas un « push », personne ne peut voir et récupérer cette branche. |
| **~/trainings/howto**  **$** touch 7.txt | À cette étape, on fait le travail sur nos fichiers (création, mise à jour, suppression …. de fichiers ou de répertoires=  En l’occurrence, nous créons un fichier vide qui s’appelle « 7.txt ». |
| **~/trainings/howto**  **$** git add -A | (Voir git add -A) |
| **~/trainings/howto**  **$** git commit -m "add f7"  [a\_new\_branch 94273c1] add f7  1 file changed, 0 insertions(+), 0 deletions(-)  create mode 100644 7.txt | (Voir git commit )  Remarque : Nous sommes bien dans la branche « a\_new\_branch » |
| **~/trainings/howto**  **$** git push  fatal: La branche courante a\_new\_branch n'a pas de branche amont.  Pour pousser la branche courante et définir la distante comme amont, utilisez  git push --set-upstream origin a\_new\_branch | Si nous faisons un « git push », cela va remonter une erreur.  En effet, nous devons spécifier avant dans quelle branche sur le serveur nous ferons les « push ».  Pour remédier à cela, il suffit de taper ce qui est affiché :  git push --set-upstream origin a\_new\_branch |
| **~/trainings/howto**  **$** git push --set-upstream origin a\_new\_branch  Total 0 (delta 0), reused 0 (delta 0)  To git.switzernet.com:training/howto.git  \* [new branch] a\_new\_branch -> a\_new\_branch  La branche a\_new\_branch est paramétrée pour suivre la branche distante a\_new\_branch depuis origin. | Nous avons remédié au problème … |
| **~/trainings/howto**  **$** git push  Décompte des objets: 2, fait.  Delta compression using up to 8 threads.  Compression des objets: 100% (2/2), fait.  Écriture des objets: 100% (2/2), 253 bytes | 0 bytes/s, fait.  Total 2 (delta 1), reused 0 (delta 0)  To git.switzernet.com:training/howto.git  0c24db6..94273c1 a\_new\_branch -> a\_new\_branch | Maintenant, nous pouvons faire un git push (Voir git push) |

### Changer de branche

Pour une raison ou une autre, on peut vouloir basculer à tout moment de branche. Dans ce cas, les fichiers du répertoire local sera conforme à la branche à laquelle on vient de se raccorder.

|  |  |
| --- | --- |
| **~/trainings/howto**  **$** git fetch --all  Récupération de origin  remote: Counting objects: 2, done.  remote: Compressing objects: 100% (2/2), done.  remote: Total 2 (delta 1), reused 0 (delta 0)  Dépaquetage des objets: 100% (2/2), fait.  Depuis git.switzernet.com:training/howto  20a0eb4..86655ab second\_new\_breanch -> origin/second\_new\_breanch | Avant toute chose, il est important d’avoir toutes les infos du serveur (Voir **Git fetch**). |
| **~/trainings/howto**  **$** git checkout master  Basculement sur la branche 'master'  Votre branche est en retard sur 'origin/master' de 1 commit, et peut être mise à jour en avance rapide.  (utilisez "git pull" pour mettre à jour votre branche locale) | Bascule vers la branche master  Remarque : Lorsqu’on bascule vers une branche vers laquelle nous avions déjà travaillé, Git reprend là où nous en étions. Si d’autres personnes ont fait avancer le projet, il est naturel que nous soyons en retard.  Nous devons donc faire un git pull |
| **~/trainings/howto**  **$** git pull  Mise à jour 0c24db6..00b798a  Fast-forward  r1/r1\_f1.txt | 0  1 file changed, 0 insertions(+), 0 dele | Dans notre exemple, nous voyons que « git pull » a permis de récupérer le répertoire « r1 » et son contenu.  Maintenant, nous pouvons continuer à travailler dans la branche « master » |

### Fusionner 2 branches

Quand on est certain que la branche sur laquelle on travaille est stable, il faut la fusionner avec la branche principale.

|  |  |
| --- | --- |
| **~/trainings/howto**  **$** git branch  a\_new\_branch  \* master | On s’assure qu’on est bien dans la branche sur laquelle le travail va continuer, en l’occurrence la branche « master ».  Remarque : C’est souvent la branche master qui est la « vrai ligne de vie » des projets, mais ça pourrait très bien être une autre. |
| **~/trainings/howto**  **$** git merge a\_new\_branch  Merge made by the 'recursive' strategy.  7.txt | 1 +  f9.txt | 0  2 files changed, 1 insertion(+)  create mode 100644 7.txt  create mode 100644 f9.txt | Avec la commande « git merge [branch\_name] », on fusionne les 2 branches dans la branche où on est. |
| **~/trainings/howto**  **$** git push | Pour publier la version fusionnée et ainsi la rendre visible par les autres, il faut faire un « git push » |

## Utilisation de Git UI

### Publier son travail

Nous allons donc refaire le même travail que précédemment, mais cette fois-ci, avec une interface graphique.

|  |  |
| --- | --- |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture24.png | Pour créer une nouvelle branche avec Git GUI, il faut aller dans le menu -> Branch -> Create |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture25.png | Saisir le nouveau nom et cliquer sur « create ».  Cela revient à faire un git checkout -b |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture26.png | Une fois créer, nous constatons la bascule de branche par l’information en dessous du menu « Current branch » |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture27.png | Dans cette étape, nous faisons le travail en local.  En l’occurrence, nous avons ajouté le fichier « f8.txt » |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture28.png | Nous pouvons faire ensuite :  . Ajouter les fichiers à l’index (Voir Stage Changed)  . Ajouter les fichiers à une version (Voir commit)  . Envoyer la dernière version au serveur (Voir push)  Remarque : Lors du push, nous voyons que nous allons mettre à jour la branche « second\_new\_branch » |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture29.png | Si tout s’est bien passé, nous voyons cette fenêtre |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture38.png | Pour rendre la nouvelle branche visible par les autres, il est nécessaire de faire un « git push » |

### Changer de branche

|  |  |
| --- | --- |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture30.png | Pour basculer vers une branche avec Git GUI, il faut aller dans le menu -> Branch -> Checkout |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture31.png | Dans la fenêtre qui s’affiche, on sélectionne la branche qui nous intéresse ; en l’occurrence « master » |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture33.png | En validant, nous constatons :   * Nous sommes bien dans la branche « master » * Les fichiers se sont bien mis à jour |

### Fusionner 2 branches

|  |  |
| --- | --- |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture33.png | On s’assure qu’on est bien dans la bonne branche et qu’elle est à jour |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture21.png | Nous allons dans le menu -> Merge -> Local Merge, comme pour mettre à jour le répertoire |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture34.png | Cette fois-ci, on sélectionne « Local Branch », et la branche qu’on veut fusionner.  Remarque : Cela sous-entend qu’on ne peut fusionner une branche que si elle est dans notre HEAD (c’est-à-dire, sur notre machine) |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture35.png | Si tout s’est bien passé, nous avons la fenêtre de validation |
| C:\Users\training\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture18.png | Pour rendre la nouvelle version fusionnée visible par les autres, il est nécessaire de faire un « git push » |

\*\*\*