

Secteur Tertiaire Informatique Filière « Etude et développement »

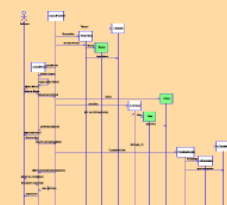
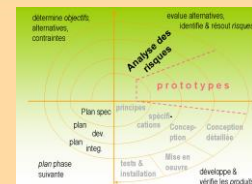
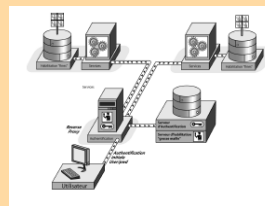
Partage de code

GitHub

Apprentissage

Mise en pratique

Evaluation



Version	Date	Auteur(s)	Action(s)
1.0	07/02/18	Ludovic Domenici	Création du document

Table des matières	4
1. Introduction	7
1.1 Le versioning.....	7
2. Installation.....	7
2.1 Windows	7
2.2 Linux	10
2.3 GitHub.....	10
3. Utilisation	11
3.1 Premiers pas avec Hello world	11
3.1.1 Étape 1 : Créer un dépôt.....	11
3.1.2 Étape 2 : Créer une branche.....	13
3.1.3 Étape 3 : Effectuer et valider des modifications	14
3.1.4 Étape 4 : Ouvrir un Pull request.....	15
3.1.5 Étape 5 :Fusionner votre demande d'extraction	17
3.2 gestion du dépôt	17
3.2.1 Dans GitHub.....	17
3.2.2 Via le shell	19
3.3 Gestion des fichiers.....	20
3.4 Gestion des <i>commits</i>	20
3.4.1 Sauvegarde provisoire	21
3.4.2 Annulations.....	21
3.5 gestion des branches	22
3.5.1 Fusion des sources.....	22
3.5.2 Liste des modifications	23
3.5.3 Récupération des changements	23
3.5.4 Exclusion de fichiers	23
4. Annexe : Aide mémoire	25
4.1 Configuration des outils.....	25
4.2 Créer des dépôts.....	25
4.3 Effectuer des changements.....	25
4.4 Grouper des changements.....	26
4.5 Changement au niveau des noms de fichier.....	26
4.6 Exclure du suivi de version.....	26

4.7	Enregistrer des fragments	27
4.8	Vérifier l'historique des versions	27
4.9	Refaire des commits	28
4.10	Synchroniser les changements	28

Objectifs

Utiliser un outil de partage de code.

Pré requis

Outils de développement

Méthodologie

Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

Ressources

Lectures conseillées

1. INTRODUCTION

GitHub est une plateforme communautaire de partage de code.

Elle permet de versionner son code (c'est-à-dire de lui donner des versions qui sont amenées à changer au fil du temps), de visualiser le code, de télécharger le code source, de partager des soucis à propos de l'application à son créateur, etc.

Git est un logiciel de gestion de version décentralisé, conçu pour être efficace tant avec les petits projets qu'avec les projets plus importants.

Git a spécialement été créé pour le développement du noyau linux en 2005.

Contrairement à des outils comme SVN ou CVS (autres logiciels de gestion de versions), Git fonctionne de façon décentralisée, c'est-à-dire que le développement ne se fait pas sur un serveur centralisé, mais chaque personne peut développer sur son propre dépôt. Git facilite ensuite la fusion des différents dépôts.

Nous allons étudier les bonnes pratiques à adapter pour se servir correctement de cet outil.

1.1 LE VERSIONING

Il est très important de versionner son code aussi bien sur un projet solo et encore plus sur un projet d'équipe.

C'est-à-dire de le sauvegarder après chaque modification. Ainsi, lorsque vous serez confronté à un bug, il vous suffira de revenir à la version précédente si vous ne souhaitez pas débayer.

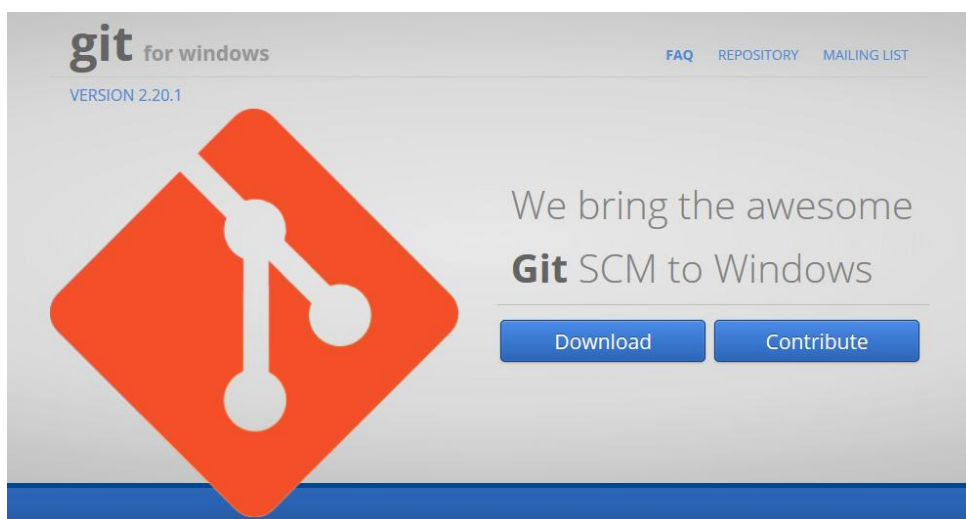
Les sauvegardes sur GitHub sont appelées des **commit**. Elles sont gérées par un gestionnaire, Git.

Avant toute chose, il nous faut le télécharger.

2. INSTALLATION

2.1 WINDOWS

Pour cela, rendez-vous sur <https://gitforwindows.org/> et choisissez la dernière version en date. Une fois téléchargé, installez le soft en laissant tous les paramètres par défaut, puis lancez l'application *git bash* (une émulation du shell bash).



GitHub

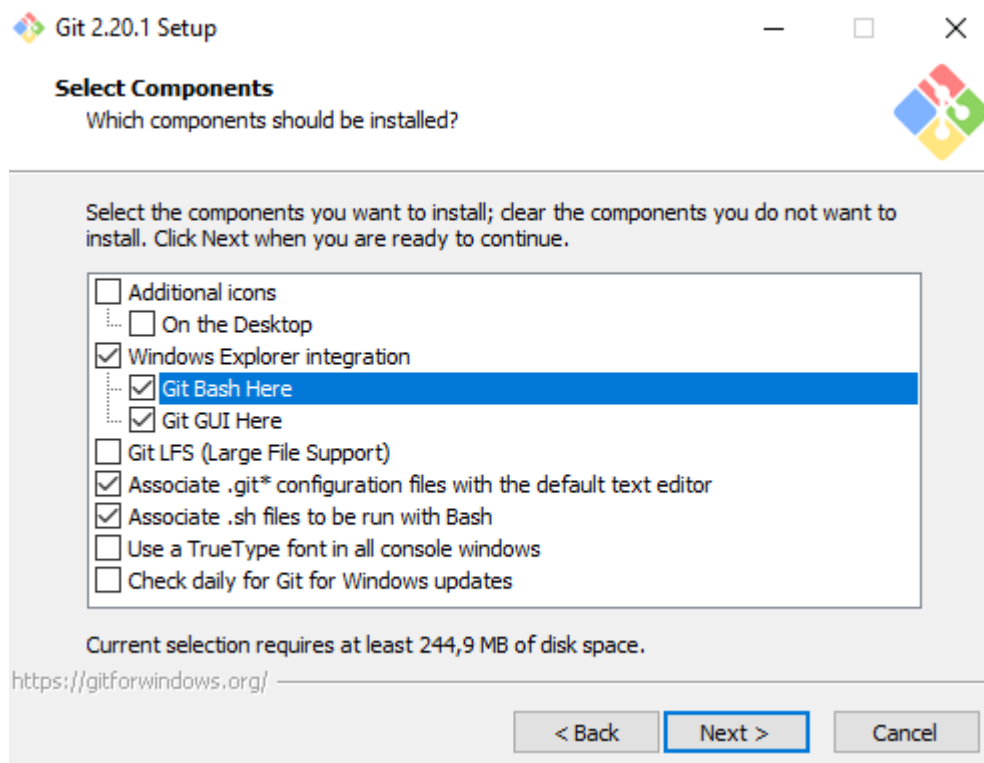


Figure 1 : Permettre l'exploration de Git via Git Bash et le GUI.

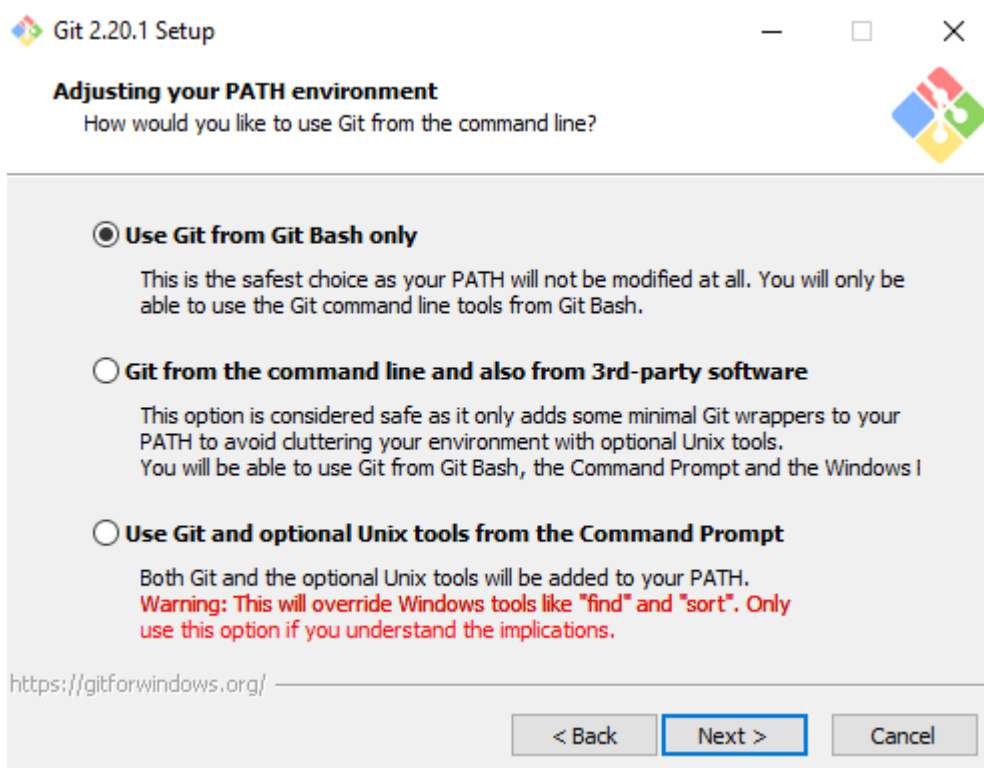
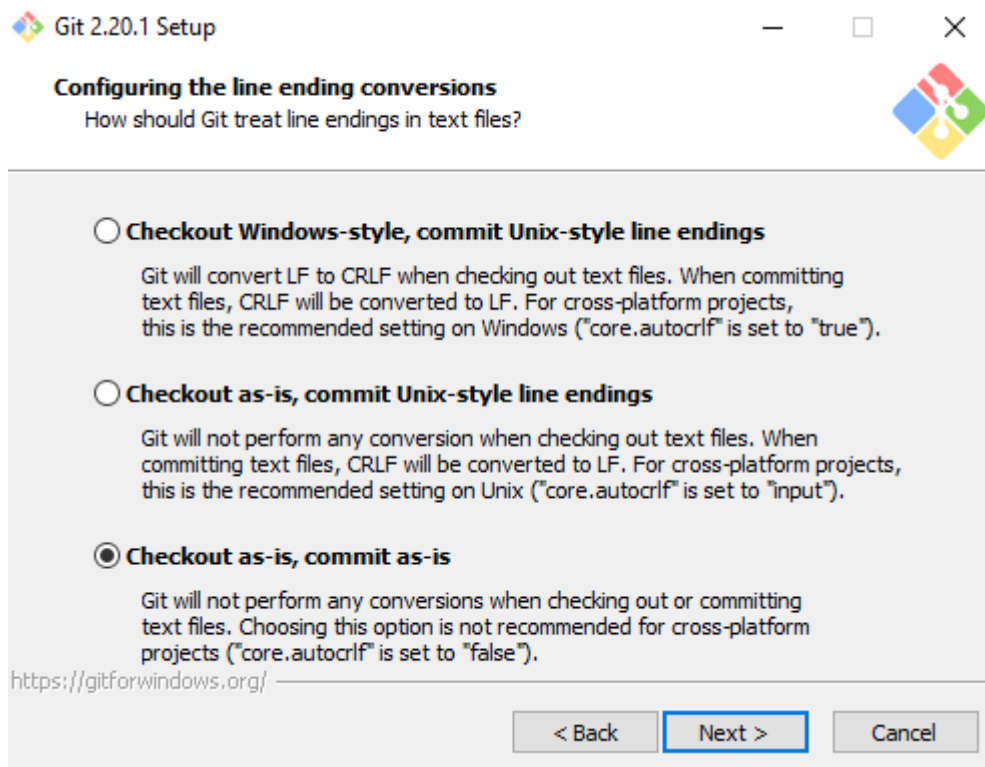


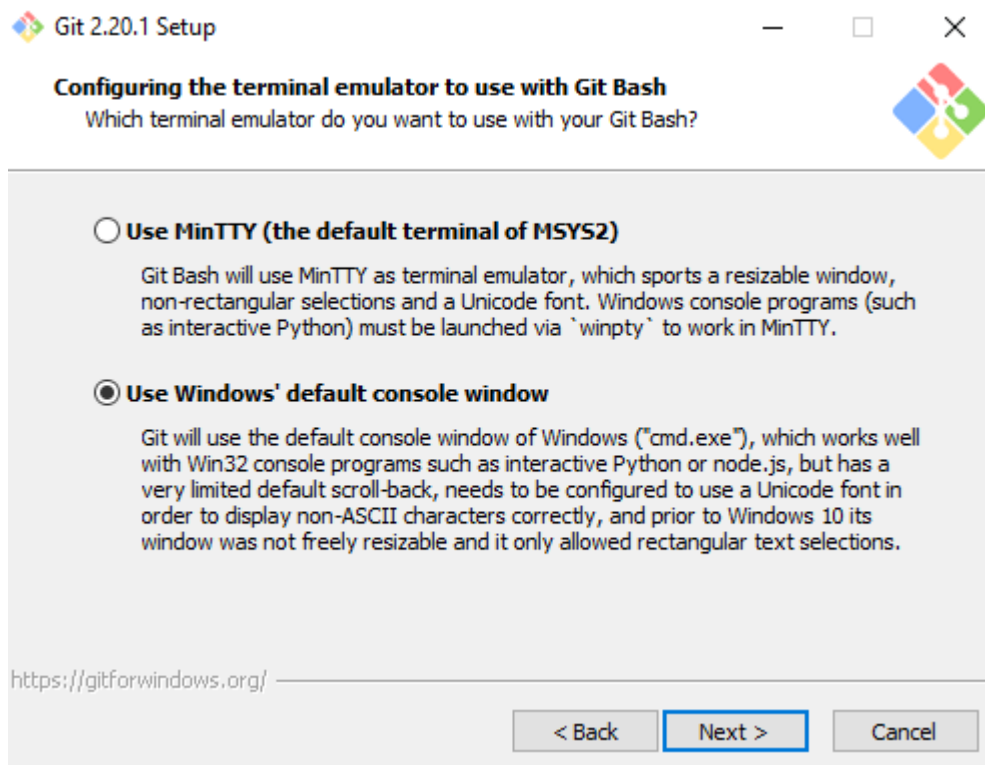
Figure 2 : Permettre l'utilisation de de Git seulement par le Git Bash.

Dans la fenêtre suivante choisissez l'utilisation d'*openSSL* pour la sécurisation des données.

Vous pouvez éviter que Git ne transforme les caractères de Fin de Ligne (LF), de retour à la ligne (CR), etc. Si vous travaillez tous dans le même environnement (Windows ou Linux) cela ne sert à rien.



Dans la fenêtre suivante, utilisez la console Windows pour la ligne de commande.



Puis laisser les autres paramètres par défaut et installez.

Afin de choisir le nom et le mail de votre compte Git, il vous faut à présent taper ces commandes :

```
$ git config --global user.name "Votre nom ou pseudo"  
$ git config --global user.email votre@email.com
```

2.2 LINUX

Il suffit d'installer le paquet git (`apt install git`).

Git dispose également de nombreux plugins, dont les plus utilisés sont dans les dépôts :

- git-svn : Gestion des dépôts SVN ;
- git-cvs : Gestion des dépôts CVS ;
- git-track ;
- etc.

Une fois l'installation réalisée, il faut impérativement définir le paramètre *user*. Pour cela, il vous faut à présent taper ces commandes :

```
$ git config --global user.name "Votre nom ou pseudo"  
$ git config --global user.email votre@email.com
```

Vous pouvez vérifier les données dans le fichier caché `.gitconfig` se trouvant dans votre dossier personnel :

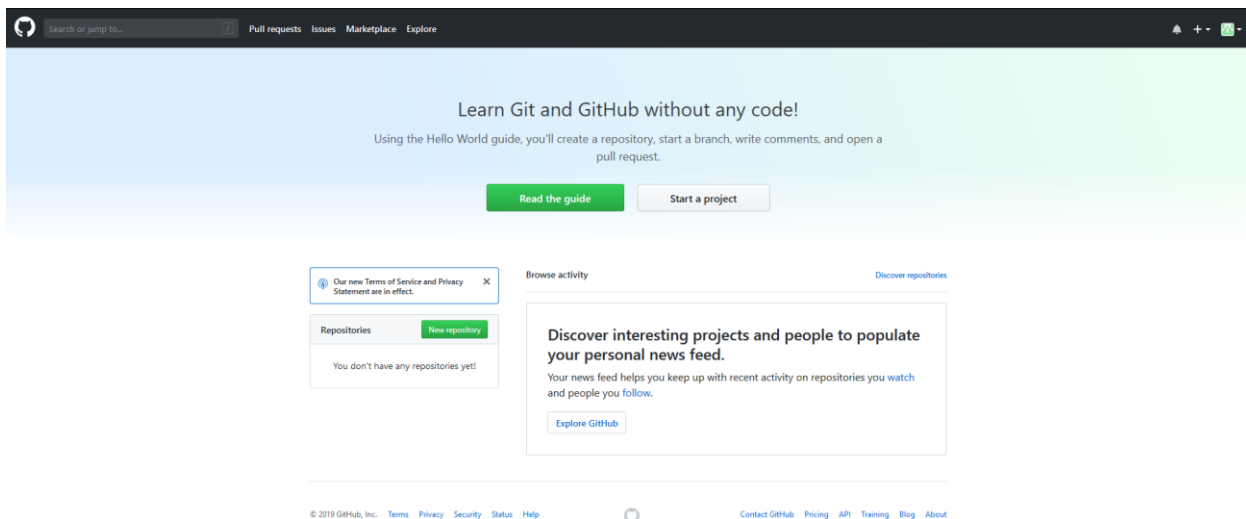
```
[user]  
  email = p.nom@ubuntu-fr.org  
  name = nom_programmeur  
  
[alias]  
  ci = commit  
  co = checkout  
  st = status  
  br = branch
```

Les alias quant à eux permettent de raccourcir les commandes, exemple :

`git st` au lieu de `git status`.

2.3 GITHUB

Rendez-vous sur le site (<https://github.com>), créer votre profil.



3. UTILISATION

La première des manipulations à effectuer via Git et de créer un dossier qui fera office de dépôt (*repository*), c'est-à-dire de dossier affilié à GitHub.

3.1 PREMIERS PAS AVEC HELLO WORLD

Cette section va vous apprendre les bases de GitHub telles que les *Repositories* (référentiels ou dépôts), les branches, les *Commits* (validations) et les *Pull Requests* (requêtes d'extraction). Vous allez créer votre propre référentiel *Hello World* et comprendre le fonctionnement du flux de travail Pull Request de GitHub.

3.1.1 Étape 1 : Créer un dépôt


Un repository (ou référentiel) est généralement utilisé pour organiser un seul projet. Les référentiels peuvent contenir des dossiers et des fichiers, des images, des vidéos, des feuilles de calcul et des ensembles de données, soit tout ce dont votre projet a besoin. Nous vous recommandons d'inclure un fichier README ou un fichier contenant des informations sur votre projet. GitHub permet l'ajout d'un fichier README en même temps que vous créez votre nouveau référentiel. Il offre également d'autres options courantes telles qu'un fichier de licence.

Votre référentiel hello-world peut être un endroit où vous stockez des idées, des ressources ou même partagez et discutez avec d'autres.

3.1.1.1 Pour créer un nouveau dépôt

1. Dans le coin supérieur droit, à côté de votre avatar, cliquez sur + puis sélectionnez New Repository.
2. Nommez votre référentiel `hello-world`.
3. Ecrire une courte description.
4. Sélectionnez Initialiser ce référentiel avec un fichier README.

Owner **Repository name**

PUBLIC  hubot / hello-world ✓

Great repository names are short and memorable. Need inspiration? How about **petulant-shame**.

Description (optional)

Just another repository

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

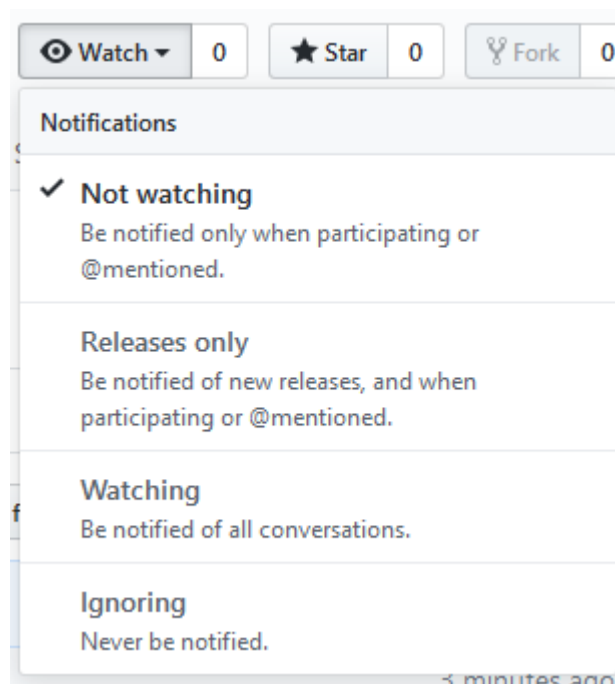
☒ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

Enfin cliquez sur *Create repository*.

Un pop-up vous signale à quel moment vous pouvez avoir des notifications.



Vous avez plusieurs choix possibles dont :

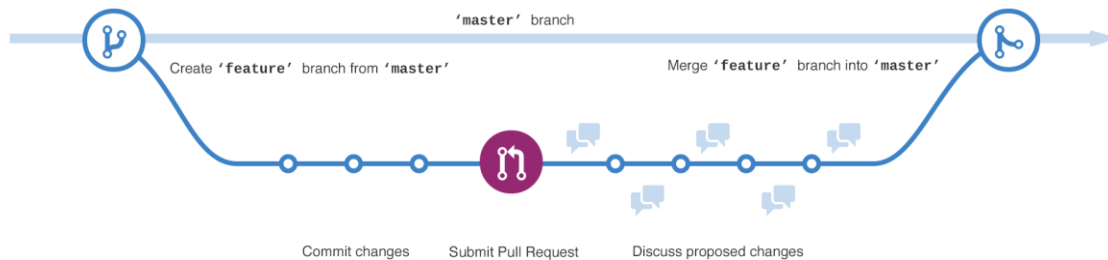
- le mentionner expressément (Not watching),
- être averti lors d'une mise à jour
- être averti tout le temps
- ignoré

3.1.2 Étape 2 : Créer une branche

Les branches sont le moyen de travailler sur différentes versions d'un référentiel à la fois.

Par défaut, votre référentiel a une branche appelée master qui est considérée comme la branche définitive. Nous utilisons des branches pour expérimenter et apporter des modifications avant de les engager à master.

Lorsque vous créez une branche à partir de la branche principale, vous créez une copie, ou un instantané, du master tel qu'il était à ce moment-là. Si quelqu'un d'autre apportait des modifications à la branche principale pendant que vous travailliez sur votre branche, vous pouvez récupérer ces mises à jour.



Ce diagramme montre :

- La branche master
- Une nouvelle branche appelée *feature* (car nous effectuons un "travail de fonctionnalité" sur cette branche)
- Le parcours de cette feature avant sa fusion en master

Avez-vous déjà créé plusieurs versions d'un fichier ? Quelque chose comme

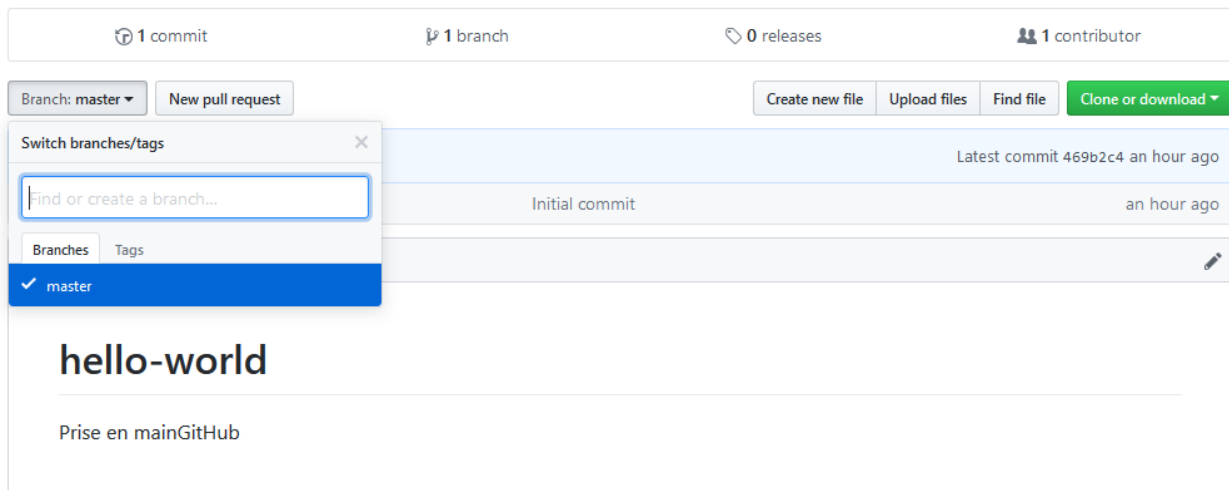
- Index.html
- Index.01.html
- Index-bak.html
- Etc...

Les branches réalisent des objectifs similaires dans les référentiels GitHub.

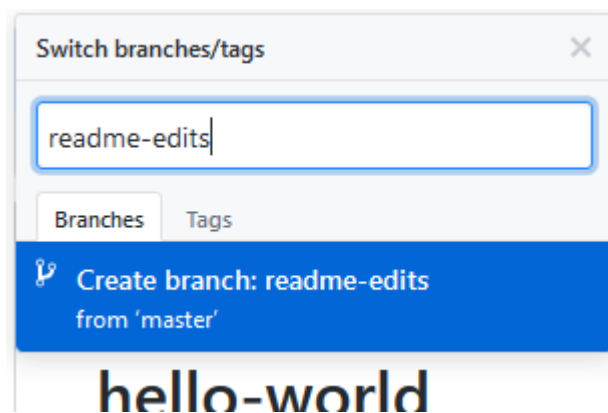
Ici, dans GitHub, les développeurs, rédacteurs et concepteurs utilisent des branches pour conserver les correctifs de bogues et le fonctionnement des fonctionnalités séparément de la branche master (qui est la branche de production). Lorsqu'un changement est prêt, ils fusionnent leur branche avec master.

3.1.2.1 Pour créer une nouvelle branche

- Accédez à votre dépôt, `hello-world`.
- Cliquez sur le menu déroulant en haut de la liste de fichiers mentionnant : *master*. Comme dans la figure suivante.



- Tapez un nom de branche, *readme-edits*, dans la zone de texte de la nouvelle branche.



- Sélectionnez la zone bleue *Create branch* ou appuyez sur la touche "Entrée" du clavier.


Vous avez maintenant deux branches: `master` et `readme-edits`. Elles se ressemblent, mais pas pour longtemps ! Nous ajouterons ensuite nos modifications à la nouvelle branche.

3.1.3 Étape 3 : Effectuer et valider des modifications

Vous êtes maintenant dans la vue code de votre branche `readme-edits` qui est une copie de `master`. Faisons quelques modifications.

Sur GitHub, les modifications enregistrées sont appelées *commits* (validations). Chaque commit a un message de validation associé, qui est une description expliquant pourquoi un changement particulier a été apporté. Les messages de commit capturent l'historique de vos modifications afin que les autres contributeurs puissent comprendre ce que vous avez fait et pourquoi.

3.1.3.1 Faire et valider des changements

1. Cliquez sur le fichier `README.md`.
2. Cliquez sur l'icône de crayon  dans le coin supérieur droit de la vue du fichier à modifier. Une infobulle averti que c'est pour éditer le fichier
3. Dans l'éditeur, écrivez un peu sur vous.

GitHub

4. Ecrivez un message de validation décrivant vos modifications.
5. Cliquez sur le bouton *Commit changes*.

hubot / hello-world

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Wiki Pulse Graphs Settings

hello-world / README.md or cancel

Edit file Preview changes Spaces 2 Soft wrap

```
1 # hello-world
2
3 Hi Humans!
4
5 Hubot here, I like Node.js and Coffeescript (that's what I'm made of!).
6 I've had tacos on the moon and find them far superior to Earth tacos.
7
```

Commit changes

Finish README

And mention moon tacos

☒ Commit directly to the `readme-edits` branch

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes Cancel

3.1.4 Étape 4 : Ouvrir un Pull request

Maintenant que vous avez des modifications dans une branche du `master`, vous pouvez ouvrir une demande d'extraction (*Pull Request*).

Les *Pull Request* sont au cœur de la collaboration sur GitHub. Lorsque vous ouvrez un Pull request, vous proposez vos modifications et demandez de réviser et d'extraire votre contribution et de la fusionner dans sa branche. Les demandes d'extraction affichent des différences entre le contenu des deux branches. Les modifications, additions et soustractions, entre les fichiers sont indiquées en vert et en rouge.

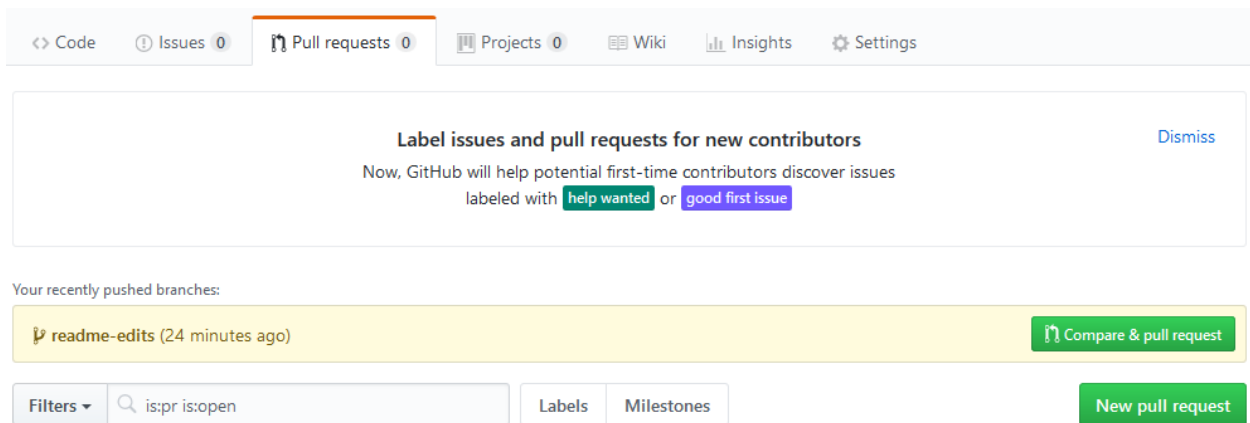
Dès que vous effectuez un commit, vous pouvez ouvrir un pull request et lancer une discussion entre collaborateurs.

En utilisant le système *@mention* de GitHub dans votre message de pull request, vous pouvez demander des informations à des personnes ou à des équipes spécifiques, où qu'elles soient.

Vous pouvez même ouvrir des demandes d'extraction dans votre propre référentiel et les fusionner vous-même. C'est une excellente façon d'apprendre le flux GitHub avant de travailler sur des projets plus importants.

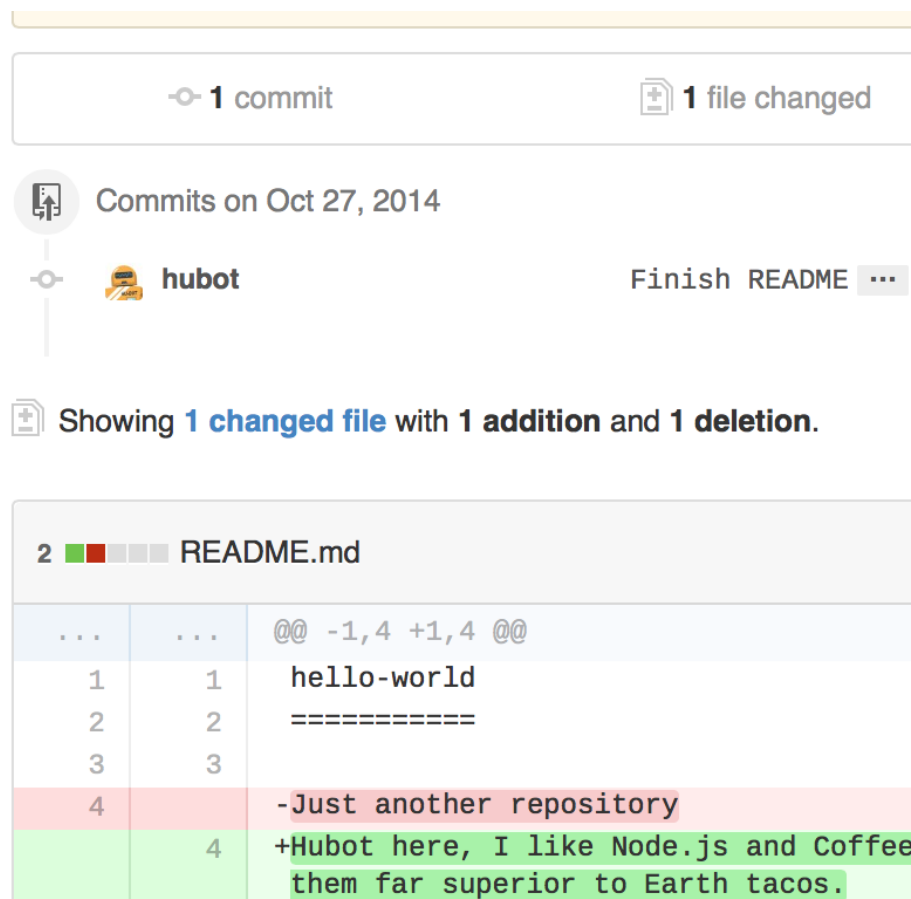
3.1.4.1 Ouvrir un pull request pour les changement du README

Cliquer sur l'onglet Pull request, puis dans la page Pull Request, cliquer sur le bouton vert New Pull Request.

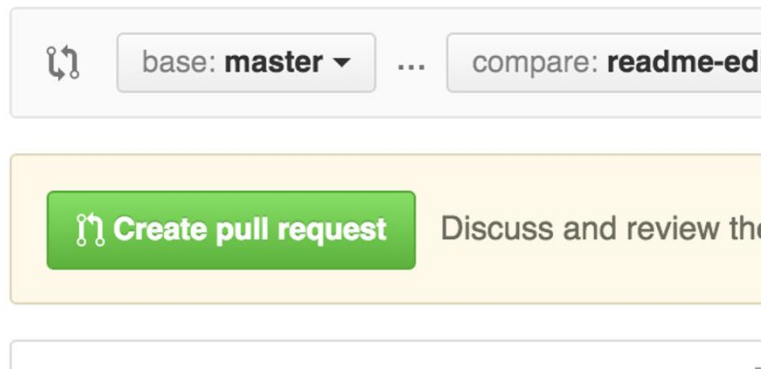


Dans la zone Example comparaisons, sélectionner la branche que vous avez créé (readme-edits) à comparer avec la branche master.

Examiner les modifications. Les différences manquantes sont sur fond rouge tandis que les ajouts sont sur fond vert.



Lorsque vous êtes satisfait des modifications que vous souhaitez soumettre, cliquez sur le bouton vert *Create a pull request*.



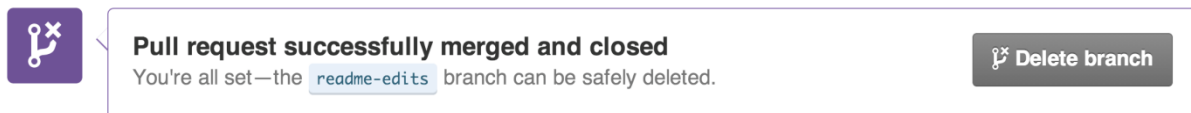
Créer un titre à votre demande d'extraction et décrivez brièvement les modifications que vous avez apporté.

Enfin cliquer sur *create a pull request*.

3.1.5 Étape 5 :Fusionner votre demande d'extraction

Dans cette dernière étape, il est temps de regrouper vos modifications en fusionnant votre branche de *readme-edits* avec la branche principale.

- Cliquez sur le bouton vert *Merge pull request* pour fusionner les modifications dans le master.
- Cliquez sur *Confirm merge*.
- Supprimez la branche en cliquant sur le bouton *Delete branch*, puisque ses modifications ont été incorporées.



3.2 GESTION DU DEPOT

3.2.1 Dans GitHub

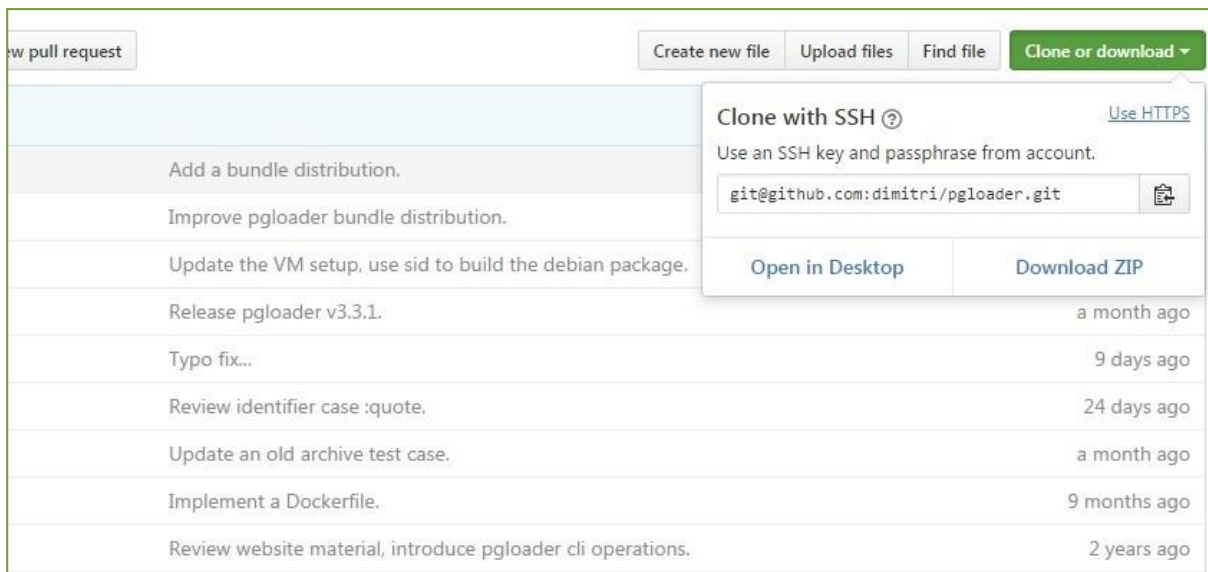
3.2.1.1 Copier un dépôt déjà existant

L'un des points intéressant de GitHub est de pouvoir copier un repository d'un autre développeur en local, sur son propre ordinateur.

Pour cela, rendez-vous sur le dépôt en ligne qui vous intéresse, et sélectionnez "*Clone or download*".

Sélectionnez "*Use HTTPS*" et copiez le lien fourni en-dessous.

Enfin, placez-vous dans un dossier dans la ligne de commande git, puis tapez `git clone <lien du repository>`. Cela créera localement une copie du dépôt qui vous intéresse.



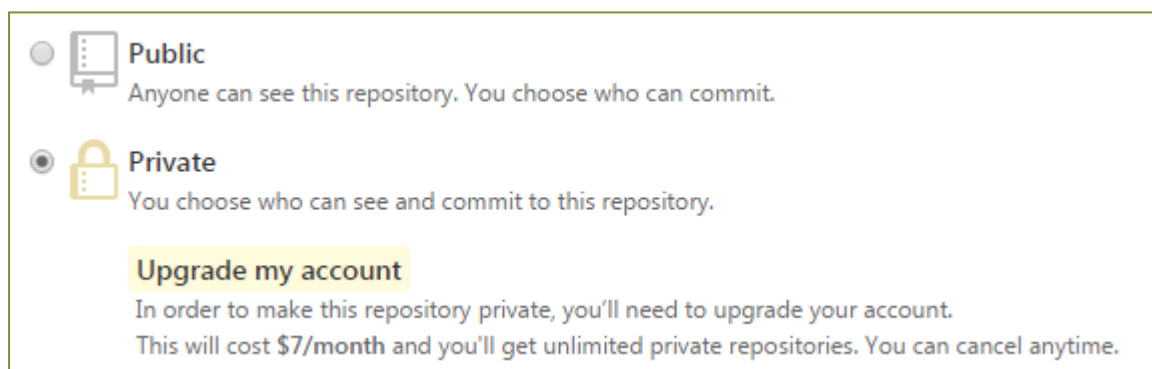
Si vous souhaitez télécharger le programme de façon plus classique, il vous est toujours possible d'obtenir l'archive ZIP du code source, et de l'installer sur votre machine.

3.2.1.2 Créer son dépôt privé

Pour créer votre propre repository, il suffit de cliquer sur la croix en haut à droite de la fenêtre gitHub, et de choisir "new repository". Vous pourrez ainsi choisir son nom, les autres propriétaires, et y inclure une description.

Ensuite, vous aurez 2 options :

- Soit votre dépôt sera public, donc consultable par tout le monde et de ce fait, clonable, mais vous pouvez toujours choisir qui pourra y effectuer un commit.
- Soit il sera privé, et vous choisirez ainsi qui y aura accès. Néanmoins, cette option est payante (7\$/mois) mais vous permettra de créer autant de dépôts privés que vous le souhaitez.



Cochez la case "Initialize this repository with a README" si vous n'avez pas encore créé le repository. Dans ce cas, GitHub le créera automatiquement pour vous. Enfin, vous avez la possibilité de commander au repo d'ignorer certains types de fichiers avec le *gitignore*, et de publier le repo avec une licence spécifique via les 2 menus déroulants.

Une fois le dépôt créé, vous pourrez mettre à jour celui-ci en lançant la commande `git push`. Cette commande comporte de nombreuses options, à l'instar de `git commit`.

- -4 pour n'utiliser que les adresses IPv4 et ainsi supprimer les adresses IPv6 (l'inverse est également réalisable).
- -v pour le mode verbeux
- -f ou --force pour forcer le push en cas de message d'avertissement

Ensuite, il faut spécifier :

- Là où on envoie le code source, c'est-à-dire dans quel appareil. *Origin* désigne GitHub, mais vous pouvez spécifier un autre ordinateur par exemple.
- La branche où on envoie le code source

Si le push s'est bien déroulé, un message de confirmation devrait apparaître.

Etant donné que le repository est en ligne et que d'autres utilisateurs sont susceptibles d'y effectuer des mises à jour, vous devez être capable de récupérer les dernières données sur votre machine, en local.

Pour se faire, utilisez la commande `git pull origin master`.

3.2.2 Via le shell

3.2.2.1 Création

Pour cela, placez-vous dans le dossier de votre choix, puis créez un nouveau dossier à l'aide de la commande `mkdir`. Placez-vous dans ce dernier avec la commande `cd`, puis lancez la commande `git init`. Comme son nom l'indique, cette commande va initier la gestion de Git pour ce dossier.

```
$ mkdir <dépôt>
$ cd <dépôt>
$ git init
```

Dans le répertoire <dépôt>, vous aurez alors un dossier caché `.git`, dans lequel Git stockera les différentes révisions et informations du projet.

3.2.2.2 Récupération

Vous pouvez aussi récupérer un dépôt déjà existant et travailler à partir de celui-ci en faisant :

```
$ git clone git://<dépôt>
```

Ou

```
$ git clone https://<dépôt>
```

Vous pourrez ensuite (si vous avez les droits suffisants sur le dépôt distant), envoyer vos changements avec `git push`.

3.2.2.3 État du dépôt

```
$ git diff
$ git diff <commit1> <commit2>
```

`git diff` permet de comparer 2 versions. Vous pourrez ainsi voir les changements effectués. Si vous avez des changements pas encore commités, la commande `git diff` affichera les modifications effectuées depuis le dernier commit.

```
$ git status
```

`git status` permet de savoir tout ce qui n'a pas encore été validé.

Tandis que la commande `git log` liste les mouvements effectués dans le dépôt.

```
$ git log
```

Vous pouvez le quitter en appuyant sur `Q`.

3.3 GESTION DES FICHIERS

Lorsque vous voudrez ajouter un fichier ou un répertoire à votre repository, vous devrez préciser à l'index Git le nom de ce fichier, avec la commande :

```
$ git add <fichier_ou_dossier>
```

Cette commande indique à Git que le fichier (ou dossier) nommé "*nom_fichier_ou_dossier*" devra être versionné. Cela permet en quelques sortes de le mettre dans une liste des fichiers reconnus.

Dans le même esprit,

```
$ git add --all *
```

Permet d'ajouter tout le contenu (fichier et dossier) du répertoire.

Pour supprimer un dossier, l'attribut `rm` le permet

```
$ git rm <nom_fichier>
```

Supprime `<nom_fichier>` de votre ordinateur, ainsi que du dépôt Git.

Et le paramètre `mv` permet de déplacer le fichier ou dossier

```
$ git mv <nom_fichier> <nouvelle_destination>
```

Déplace le fichier `<nom_fichier>` dans `<nouvelle_destination>`.

3.4 GESTION DES COMMITS

Le commit est l'action de placer les sources sur le dépôt GitHub afin de le mettre à disposition des autres membres. Cette opération s'effectue une fois que les modifications apportées au fichier sont validées.

"Il s'agit de la validation d'une transaction. Après avoir commis la transaction, les informations traitées par cette transaction seront disponibles pour les autres sessions, c'est-à-dire pour toute autre transaction éventuelle."

<http://fr.wikipedia.org/wiki/Commit>

Avant de commencer à modifier les fichiers, pour être sûr de ne travailler que sur les dernières versions, et avant tout commit, pour éviter les éventuels conflits avec des modifications effectuées par d'autres utilisateurs entre temps :

```
$ git pull
```

Met à jour votre dépôt local.

```
$ git commit <fichier1> <fichier2>
```

Crée un commit contenant <fichier1> et <fichier2>. Ces fichiers auront dû être au préalable ajoutés au dépôt avec la commande `git add`.

```
$ git commit -a
```

Crée un nouveau commit contenant tous les changements effectués sur les fichiers (`git add` n'est donc pas nécessaire avant un `commit -a`).

```
$ git push origin master
```

Envoie le commit dans la branche principale "master" du dépôt "origin".

`git commit` possède de nombreuses options intéressantes :

- `-v` pour activer le mode verbeux
- `--date=<date>` pour remplacer la date du commit
- `--author=<auteur>` pour remplacer le nom de l'auteur du commit
- `-m` pour attacher un message au commit (comme une description). Il est fortement conseillé d'utiliser cette option afin de s'y retrouver plus facilement parmi les commit.
-

3.4.1 Sauvegarde provisoire

Si vous désirez enregistrer temporairement vos modifications sans pour autant engendrer un autre commit, la commande `git stash` est appropriée. Si vous voulez appliquer les modifications apportées avant le `git stash`, il faudra taper `git stash apply` (ou `pop` à la place de `apply` pour vider votre `stash`).

3.4.2 Annulations

Git dispose de commandes permettant d'annuler des changements effectués. **Attention, ces annulations ne sont pas réversibles !**

Annuler les changements effectués depuis le dernier *commit*.

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

Supprime le dernier *commit*. Cette action peut être répétée autant de fois que vous le désirez.

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

Restaure le dépôt tel qu'il l'était lors du *commit* spécifié.

```
$ git revert commit
```

Pour que cette commande fonctionne, il faut que toutes les modifications soient *committées* ou annulées avec `git reset`.

Si vous avez repéré un commit spécifique et que vous voulez vous y déplacer, il faudra taper

```
$ git checkout <Commit>
```

Le commit dans lequel vous vous trouvez à la base s'appelle le *master*, donc pour y retourner, il suffira de taper

```
$ git checkout master.
```

3.5 GESTION DES BRANCHES

Les branches désignent des versions ultérieures du code. Ainsi, un développeur pourra modifier une version précédente, tandis qu'un autre pourra travailler sur la version la plus récente, généralement posée sur le tronc.

Cela est utile notamment pour la correction de bugs propres à certaines versions du logiciel, ou pour tester une fonctionnalité sans impacter le développement global vis-à-vis des autres utilisateurs.

Pour lister toutes les branches du dépôt :

```
$ git branch.
```

Le nom de la branche courante est précédé du caractère "*".

```
$ git branch <nouvelle_branche>
```

Créer la nouvelle branche `<nouvelle_branche>`, et

```
$ git checkout <nouvelle_branche>
```

Permet de s'y positionner.

La commande change de branche. Après cette commande, vous aurez alors accès aux fichiers présents dans la branche nommée `<nouvelle_branche>`. Avant de changer de branche, il faut impérativement que tous les changements effectués soient *sauvegardés (commités)* ou annulés, sinon Git refusera de changer de branche.

Plus simplement, ces 2 commandes peuvent être résumées en une seule :

```
$ git checkout -b <nouvelle_branche>
```

3.5.1 Fusion des sources

Il est possible de fusionner 2 branches, c'est-à-dire d'appliquer à une branche les modifications d'une autre branche.

Après s'être placé dans la branche que l'on veut mettre à jour avec un `git checkout`, on spécifie le nom de la branche avec laquelle on veut appliquer une fusion :

```
$ git merge <branche à fusionner>.
```

La fusion de branche est très pratique et vous serez amené à vous en servir souvent au cours du projet.

Néanmoins, elle s'accompagne d'une petite question : que se passe-t-il si on essaie de fusionner 2 branches dans lesquelles on a mis à jour un même élément ?

Dans le langage GitHub cette situation est appelée un conflit.

Une erreur va être soulevée lors de la tentative de fusion, contenant des informations sur le paramètre qui empêche ce dernier de se dérouler correctement.

Il vous suffira alors de modifier le paramètre convenablement grâce à un éditeur de texte tel que winmerge sous Windows ou vim sous linux, puis de faire un `git commit`.

Si la résolution du conflit est terminée, la console devrait retourner un message de confirmation.

Cependant si jamais vous décidez d'abandonner la fusion, alors exécutez la commande :

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

3.5.2 Liste des modifications

Si vous travaillez sur un projet avec de nombreux collaborateurs, il est intéressant de savoir qui a fait quoi comme modifications.

Pour cela, la commande `git blame` est très appropriée puisqu'elle va afficher sous forme de liste tous les changements que tous les utilisateurs ont effectué sur le fichier spécifié (sha, auteur, date).

Pensez donc à faire un `git show <sha>` pour avoir plus d'informations concernant une modification spécifique.

3.5.3 Récupération des changements

Imaginons que Bob ait implémenté une nouvelle fonctionnalité. Vous voulez naturellement l'intégrer à votre dépôt.

```
$ git remote add bob git://github.com/bob
$ git fetch bob
$ git merge bob/master
```

La première commande crée un alias qui fait pointer bob vers l'adresse du dépôt. Ça permet d'éviter d'avoir à taper l'adresse complète à chaque fois.

La deuxième commande récupère les changements que Bob a effectués.

La troisième commande fusionne les changements de Bob avec votre branche courante. Il existe une commande qui a le même effet que `git fetch` suivi de `git merge` :

```
$ git pull bob
```

3.5.4 Exclusion de fichiers

Un autre point important est le fait d'ignorer certains fichiers. En effet, certains fichiers comme les fichiers de configuration pourraient poser une faille de sécurité s'ils étaient disponibles à tous. GitHub gère très bien cela. En effet, il vous suffit de créer un fichier nommé `.gitignore` pour exclure des fichiers à tous les utilisateurs.

Parallèlement, le fichier `.git/info/exclude` permet également d'exclure des fichiers, mais uniquement pour soi-même car il n'est pas partagé avec les autres contributeurs du projet.

Enfin, il est possible de demander à Git d'ignorer des fichiers spécifiques sur tous les projets. Pour cela, créez un fichier (il est conseillé de le créer dans le répertoire de Git)

```
cd $HOME/.config/git  
touch ignore
```

Et lancez la commande

```
git config --global core.excludesfile ~/.config/git/ignore
```

Ce fichier est également personnel, et il ne faut pas indiquer de fichier spécifique à un projet dans celui-ci.

Dans chacune de ces 3 méthodes, le principe pour exclure des fichiers reste le même.

Par exemple :

- Pour ignorer tous les fichiers ou dossiers abc : abc
- Pour ignorer le répertoire abc à la racine du projet : /abc
- Pour ignorer les fichiers ayant l'extension mp3 ou mp4 : *.mp[34]
- Pour ignorer les fichiers commençant par test mais en acceptant le fichier test_final : test* !test_final.

4. ANNEXE : AIDE MEMOIRE

4.1 CONFIGURATION DES OUTILS

Configurer les informations de l'utilisateur pour tous les dépôts locaux

```
$ git config --global user.name "[nom]"
```

Définit le nom que vous voulez associer à toutes vos opérations de commit

```
$ git config --global user.email "[adresse email]"
```

Définit l'email que vous voulez associer à toutes vos opérations de commit

```
$ git config --global color.ui auto
```

Active la colorisation de la sortie en ligne de commande

4.2 CREER DES DEPOTS

Démarrer un nouveau dépôt ou en obtenir un depuis une URL existante

```
$ git init [nom-du-projet]
```

Crée un dépôt local à partir du nom spécifié

```
$ git clone [url]
```

Télécharge un projet et tout son historique de versions

4.3 EFFECTUER DES CHANGEMENTS

Consulter les modifications et effectuer une opération de commit

```
$ git status
```

Liste tous les nouveaux fichiers et les fichiers modifiés à commiter

```
$ git add [fichier]
```

Ajoute un instantané du fichier, en préparation pour le suivi de version

```
$ git reset [fichier]
```

Enleve le fichier de l'index, mais conserve son contenu

```
$ git diff
```

Montre les modifications de fichier qui ne sont pas encore indexées

```
$ git diff --staged
```

GitHub

Montre les différences de fichier entre la version indexée et la dernière version

```
$ git commit -m "[message descriptif]"
```

Enregistre des instantanés de fichiers de façon permanente dans l'historique des versions

4.4 GROUPER DES CHANGEMENTS

Nommer une série de commits et combiner les résultats de travaux terminés

```
$ git branch
```

Liste toutes les branches locales dans le dépôt courant

```
$ git branch [nom-de-branche]
```

Crée une nouvelle branche

```
$ git checkout [nom-de-branche]
```

Bascule sur la branche spécifiée et met à jour le répertoire de travail

```
$ git merge [nom-de-branche]
```

Combine dans la branche courante l'historique de la branche spécifiée

```
$ git branch -d [nom-de-branche]
```

Supprime la branche spécifiée

4.5 CHANGEMENT AU NIVEAU DES NOMS DE FICHIER

Déplacer et supprimer des fichiers sous suivi de version

```
$ git rm --cached [fichier]
```

Supprime le fichier du système de suivi de version mais le préserve localement

```
$ git rm [fichier]
```

Supprime le fichier du répertoire de travail et met à jour l'index

```
$ git mv [fichier-nom] [fichier-nouveau-nom]
```

Renomme le fichier et prépare le changement pour un commit

4.6 EXCLURE DU SUIVI DE VERSION

Exclure des fichiers et chemins temporaires

```
*.log  
Build/  
Temp-*
```

Un fichier texte nommé `.gitignore` permet d'éviter le suivi de version accidentel pour les fichiers et chemins correspondant aux patterns spécifiés

```
$ git ls-files --other --ignored --exclude-standard
```

Liste tous les fichiers exclus du suivi de version dans ce projet

4.7 ENREGISTRER DES FRAGMENTS

Mettre en suspens des modifications non finies pour y revenir plus tard

```
$ git stash
```

Enregistre de manière temporaire tous les fichiers sous suivi de version qui ont été modifiés ("remiser son travail").

```
$ git stash list
```

Liste toutes les remises

```
$ git stash pop
```

Applique une remise et la supprime immédiatement

```
$ git stash drop
```

Supprime la remise la plus récente

4.8 VERIFIER L'HISTORIQUE DES VERSIONS

Suivre et inspecter l'évolution des fichiers du projet

```
$ git log
```

Montre l'historique des versions pour la branche courante

```
$ git log --follow [fichier]
```

Montre l'historique des versions, y compris les actions de renommage, pour le fichier spécifié

```
$ git diff [premiere-branche]...[deuxieme-branche]
```

Montre les différences de contenu entre deux branches

```
$ git show [commit]
```

Montre les modifications de métadonnées et de contenu incluses dans le commit spécifié

4.9 REFAIRE DES COMMITS

Corriger des erreurs et gérer l'historique des corrections

```
$ git reset [commit]
```

Annule tous les commits après `[commit]`, en conservant les modifications localement

```
$ git reset --hard [commit]
```

Supprime tout l'historique et les modifications effectuées après le commit spécifié

4.10 SYNCHRONISER LES CHANGEMENTS

Référencer un dépôt distant et synchroniser l'historique de versions

```
$ git fetch [nom-de-depot]
```

Récupère tout l'historique du dépôt nommé

```
$ git merge [nom-de-depot]/[branche]
```

Fusionne la branche du dépôt dans la branche locale courante

```
$ git push [alias] [branche]
```

Envoie tous les commits de la branche locale vers GitHub

```
$ git pull
```

Récupère tout l'historique du dépôt nommé et incorpore les modifications

CREDITS

Les bases de Git et de GitHub : Antoine Sipesaque.

Utiliser Git et GitHub dans vos projets : Maxime Paloulack.

Git par Geenux.

GitHub

ŒUVRE COLLECTIVE DE l'AFPA

Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)

Ludovic Domenici – Formateur Développement Logiciel

Ch. Perrachon – Ingénieure de formation

Date de mise à jour : 15/01/19

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

GitHub

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »