**GIT**

**Dure totale du module : 21H00**

Table des matières

[Objectifs pédagogiques : 3](#_Toc68804482)

[Présentation et installation de GIT 4](#_Toc68804483)

[Présentation de GIT : 4](#_Toc68804484)

[Installation de GIT : 4](#_Toc68804485)

[Création d’un compte github et du premier repository (dépôt distant) : 7](#_Toc68804486)

[Création d’un repository sur github : 8](#_Toc68804487)

[Cloner un dépôt github (dépôt distant) sur votre repository (local) 9](#_Toc68804488)

[Le système de sauvegarde (commit) : 10](#_Toc68804489)

[Présentation du système de sauvegarde de git : 10](#_Toc68804490)

[Gestion des fichiers et commits (sauvegarde) : 11](#_Toc68804491)

[Les Remises (Stash) : 14](#_Toc68804492)

[Interaction avec un repository distant (github) : 16](#_Toc68804493)

[Téléchargement des données distantes vers votre dépôt local (dossier) 16](#_Toc68804494)

[Envoi des données locales vers votre dépôt distant (repository github) 17](#_Toc68804495)

[Le système de branches : 18](#_Toc68804496)

[Présentation du système de branches : 18](#_Toc68804497)

[Gestion des branches : 18](#_Toc68804498)

[Correction de problèmes Git : 20](#_Toc68804499)

[Les 3 types de réinitialisation de GIT : 23](#_Toc68804500)

[Scénarios d’erreurs : 25](#_Toc68804501)

[Fusion (merge): 32](#_Toc68804502)

# Objectifs pédagogiques :

Etre capable de comprendre les enjeux du système de branches (versionning, collaboration)

Etre capable de comprendre et utiliser le workflow Git Flow

Etre capable d'initialiser et de restaurer un dépôt local ou distant

Etre capable de réaliser des commandes commit, push et pull sur des dépôts distants (github)

# Présentation et installation de GIT

## Présentation de GIT :



**Qu’est-ce que GIT ?**

Git est un outil de gestion de projets et de versions collaboratif.

**Git** est un [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) [décentralisé](https://fr.wikipedia.org/wiki/Gestion_de_versions#Gestion_de_versions_d.C3.A9centralis.C3.A9e). C'est un [logiciel libre](https://fr.wikipedia.org/wiki/Logiciel_libre) créé par [Linus Torvalds](https://fr.wikipedia.org/wiki/Linus_Torvalds), auteur du [noyau Linux](https://fr.wikipedia.org/wiki/Noyau_Linux), et distribué selon les termes de la [licence publique générale GNU](https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU) version 2. En 2016, il s’agit du [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) le plus populaire qui est utilisé par plus de douze millions de personnes.

## Installation de GIT :

Télécharger l’outil à cette adresse :

<https://git-scm.com/downloads>

1 Lancer Git Bash :

2 Enregistrer le login :

$ git config --global user.name "***nom\_utilisateur***"

3 Enregistrer son adresse mail :

$ git config --global user.email **mail\_utilisateur**

"***nom\_utilisateur***"

« «  « «  «« « "***nom\_utilisateur***"

4 Pour vérifier sa configuration :

$ git config --list

"***nom\_utilisateur***"

« «  « «  «« « "***nom\_utilisateur***"

5 Activation de la coloration (optionnel):

$ git config --global color.diff auto

$ git config --global color.status auto

$ git config --global color.branch auto

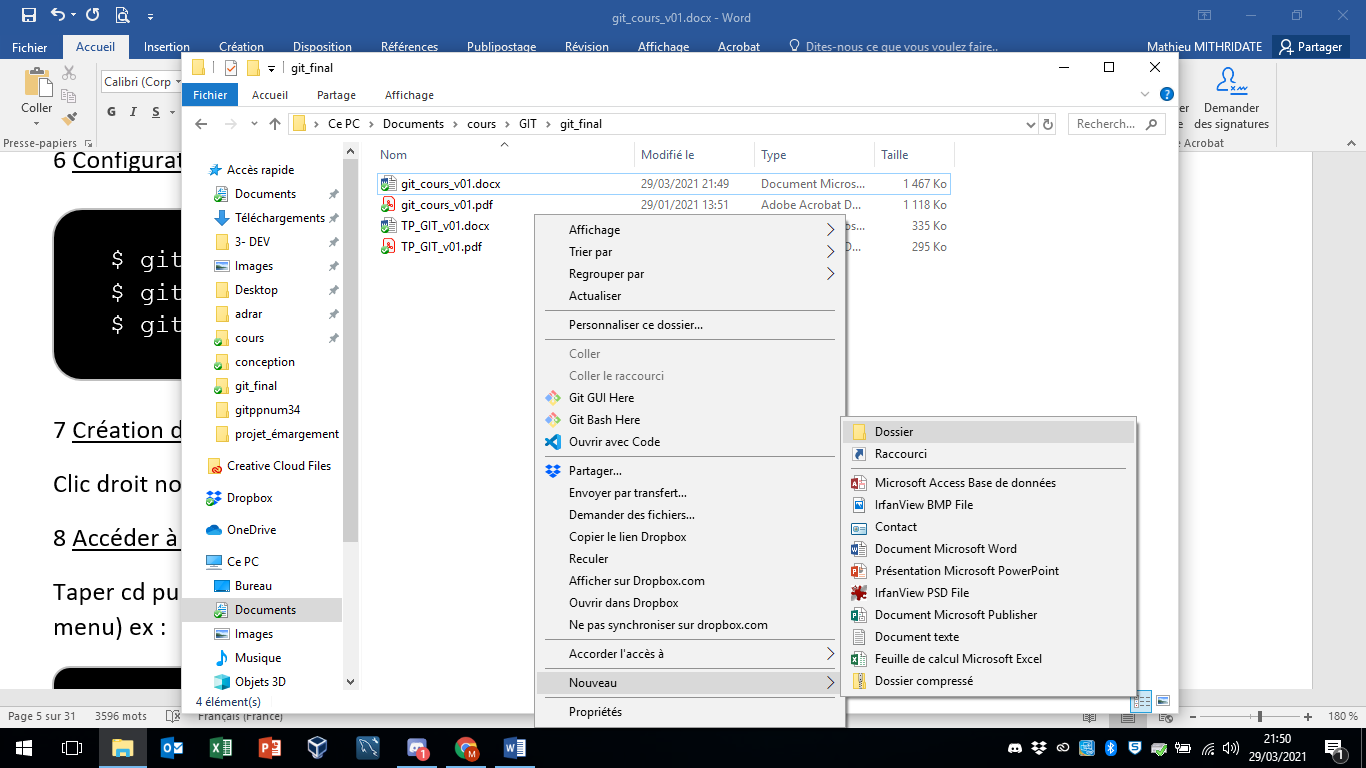
6 Configuration de son éditeur de texte par défaut et Vimdiff comme outil de merge (optionnel):

$ git config --global core.editor ***notepad++*** $ git config --global core.editor ***code***

$ git config --global merge.tool ***vimdiff***

7 Création d’un répertoire sur votre ordinateur :

Clic droit nouveau dossier.



8 Accéder à votre répertoire (dépôt local) :

Taper cd puis l’url de votre répertoire ou clic droit sur votre dossier (ou bien git bash here dans le menu) ex :

$ cd ***C:/Users/nom\_utilisateur/Desktop/git***

9 Initialisation du dépôt (local) :

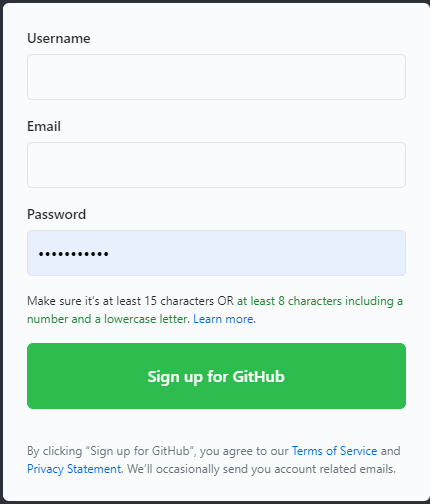
$ git init

## Création d’un compte github et du premier repository (dépôt distant) :

1 Se connecter sur github.com :

<https://github.com>

2 Cliquer sur Sign up en haut à droite :

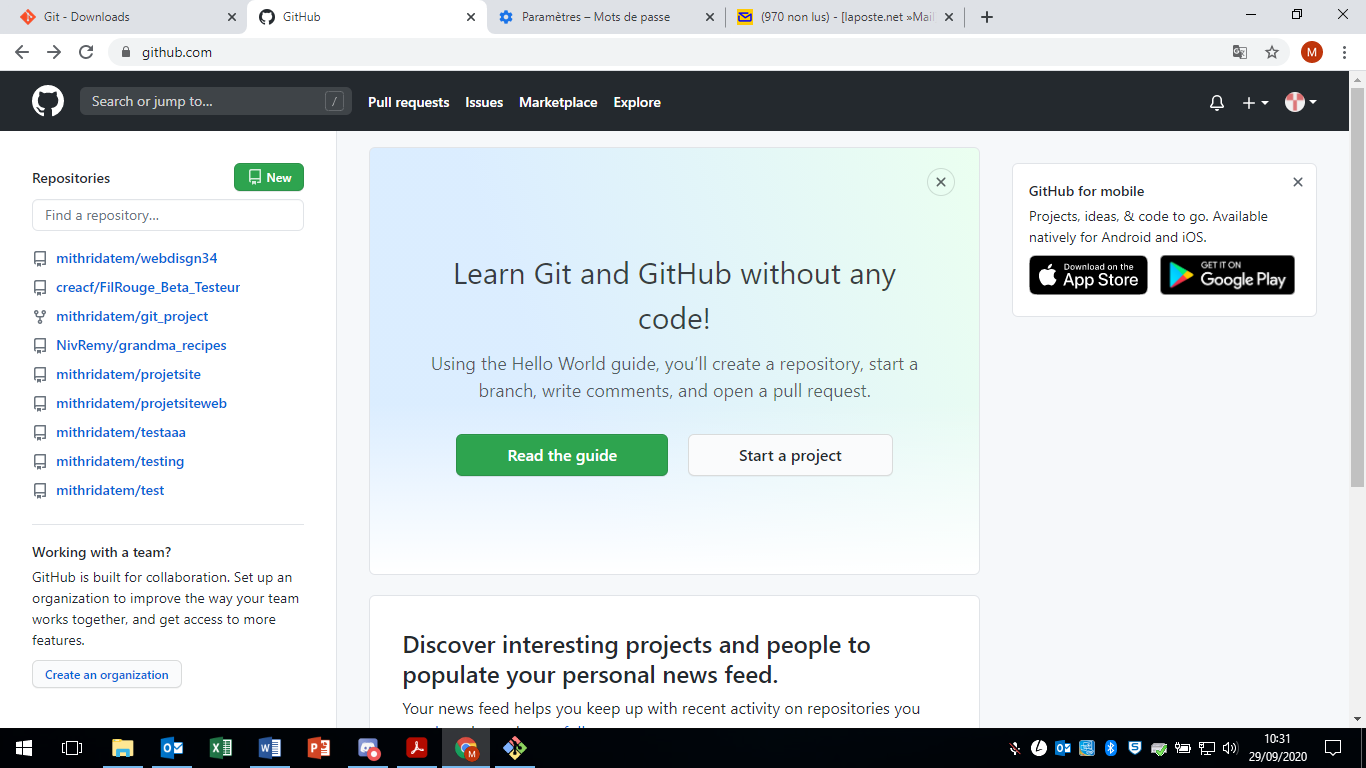


Renseigner un Username *(nom utilisateur*)

Renseigner un Email (*adresse mail*)

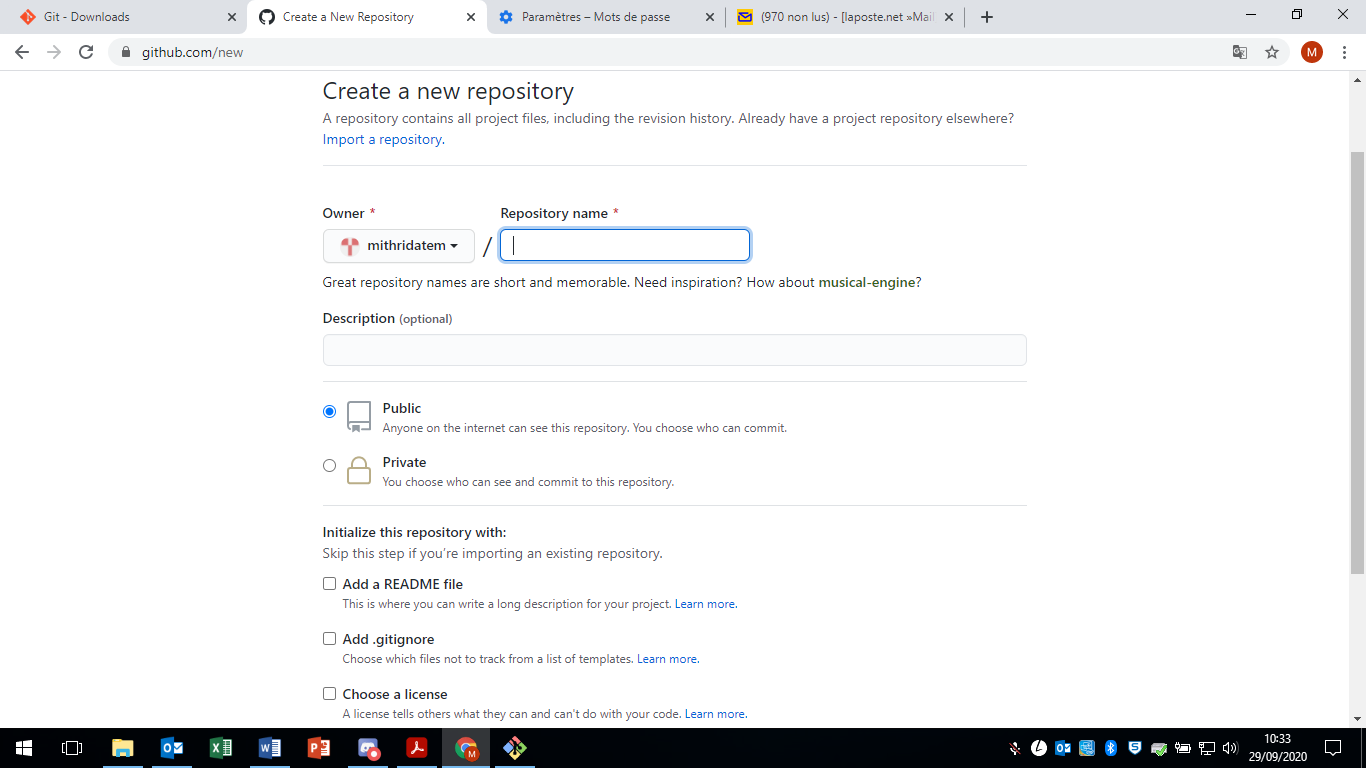
Renseigner un Password (*mot de passe*)

## Création d’un repository sur github :



1 Cliquer sur new.

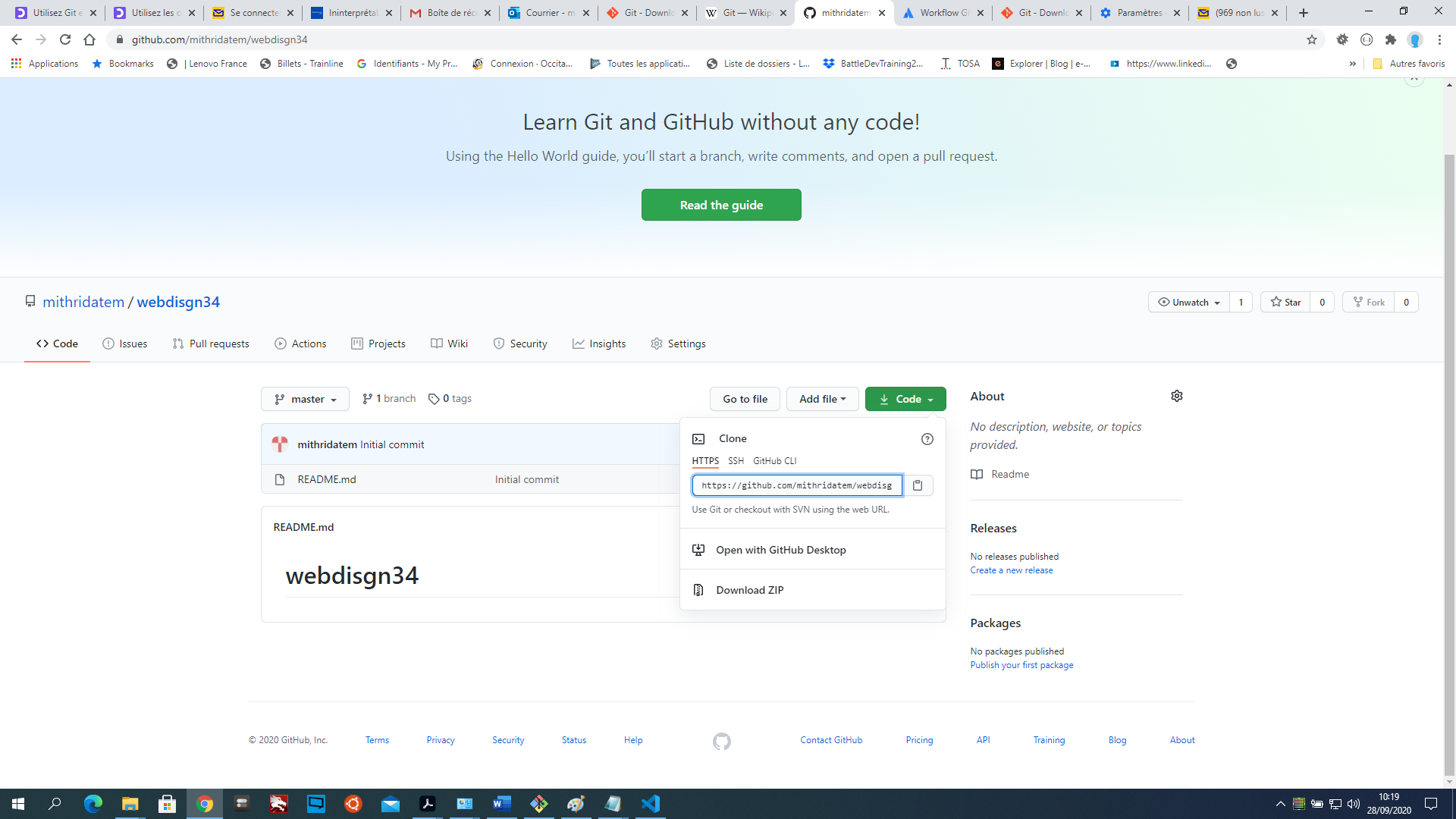
2 Choisir un nom pour votre repository (nom\_repository, cocher public et cocher add readme file):



3 Cliquer sur create repository bouton en bas de la page.

## Cloner un dépôt github (dépôt distant) sur votre repository (local)

1 Cliquer sur code (interface de github) :

2 Récupérer l’url de votre dépôt (https) :

3 Taper la ligne de commande suivante dans la console git :

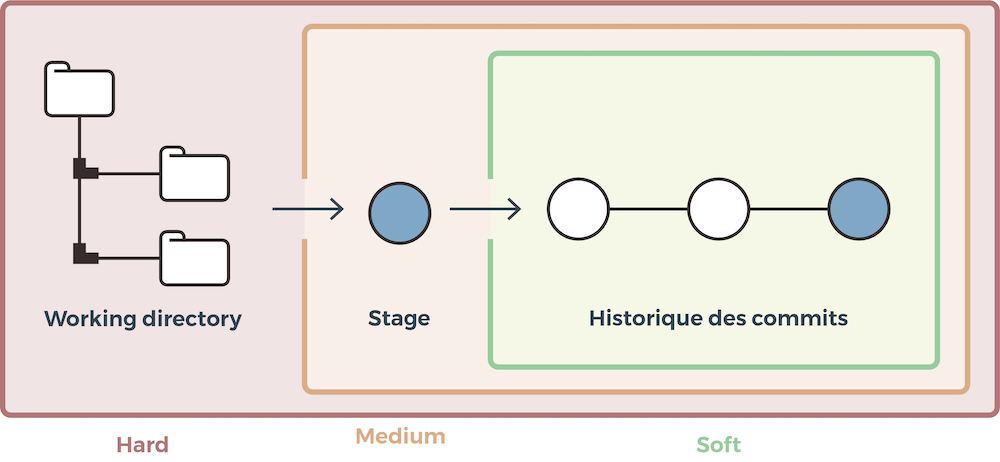
$ git remote add clone ***https://github.com/votre\_url.git***

4 méthode alternative :

$ git remote add origin ***https://github.com/votre\_url.git***

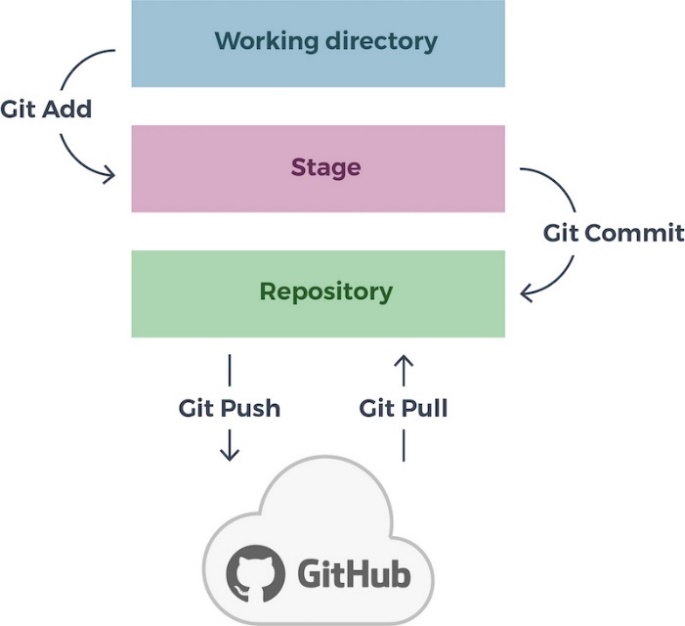
# Le système de sauvegarde (commit) :

## Présentation du système de sauvegarde de git :



Git gère les versions de nos travaux locaux à travers 3 zones locales majeures :

* **Le répertoire de travail** (working directory/WD) ;
* **L’index, ou *stage***(File d’attente) ;
* **Le dépôt local**(Git directory/repository **commit**).



## Gestion des fichiers et commits (sauvegarde) :

1 Créer un fichier dans votre répertoire :

Ex : test.txt

2 Ajouter le fichier à git (Stage) :

Utiliser la commande ci-dessous :

$ git add **test.txt**

3 Sauvegarde des modifications (commit) :

Vous avez réalisé des évolutions sur la branche ***nom de la branche*** et il va falloir maintenant demander à Git de les enregistrer.

Pour cela on va utiliser la commande commit avec –m (pour ajouter un message).

$ git commit -m ***“message de la modification”***

4 Sauvegarde des modifications (commit simple) :

$ git commit

Cette commande va ouvrir l’éditeur Vim.

On vous demande alors d'indiquer le message du commit puis de valider. Pour valider le message, une fois que vous l'avez écrit, appuyez sur Echap (votre curseur va basculer sur la dernière ligne) et

Saisir

***:x!***

Cette commande va sauvegarder et quitter l'éditeur des messages de commit.

5 Sauvegarder plusieurs création, modifications :

Plutôt que faire plusieurs commits individuel on va utiliser le paramètre –am sur la commande commit voir exemple ci-dessous :

On va ajouter le/les fichiers (stage) avec la commande suivante :

$ git add \* ou git add -A

Sauvegarde (commit) avec la commande suivante :

$ git commit -am ***“message modifications multiples”***

6 sauvegarder des modifications de fichiers (édition de fichiers) :

Pour sauvegarder des modifications dans des fichiers on va utiliser la même méthode que pour ajouter des fichiers :

Éditer un fichier en local :

-Ouvrir un fichier et l’éditer (par ex : ajouter du contenu) puis sauvegarder le.

Ajouter le fichier à git avec la commande suivante :

$ git add **nom\_fichier.extension (ex : test.txt)**

Sauvegarder la modification avec la commande suivante (commit) :

$ git commit -m ***“message de la modification”***

7 Suppression de fichiers :

Pour supprimer des fichiers on va utiliser la commande suivante (stage):

$ git rm nom\_fichier.extension

On répète l’opération si on n’a plusieurs fichiers à supprimer.

Sauvegarde des suppressions (commit) on va utiliser la commande suivante :

$ git commit –m ou –am ***“message de la modification”***

## Les Remises (Stash) :

Les remises permettent de mettre de côté des modifications (ajout, modifications …) pour pouvoir les copier sur la branche actuelle ou une autre branche à n’importe quel moment. Cela fonctionne comme une liste d’attente.

1 Voir l’état des fichiers :

$ git status

2 Création d’une remise :

Pour pouvoir effectuer une remise il faut avoir précédemment avoir ajouter les nouveaux fichiers ou modifications avec la commande ci-dessous :

$ git add \*

Cette commande va permettre d’ajouter tous les fichiers à la fois (*modifications et nouveaux fichiers*)

Pour effectuer la remise nous allons utiliser la commande ci-dessous :

$ git stash

Cela va créer une remise et mettre en attente les modifications (*les fichiers ou modifications disparaitront de votre dossier local*).

3 Application de la remise :

Cette commande va appliquer toutes les modifications précédentes (*Créer une remise*) sur la branche sélectionnée.

(BrancheCommit).

$ git stash apply

4 Voir la liste des remises :

Nous avons la possibilité de faire plusieurs remises afin de voir la liste des remises nous allons utiliser la commande ci-dessous :

$ git stash list

Cela permet de voir la liste des remises avec un identifiant associé à chaque remise.

5 Application d’une remise avec son identifiant :

$ git stash apply stash@{0}

Cette commande applique sur la branche sélectionné la remise « ***stash@{0}*** » *(la première dans ce cas*).

6 Supprimer une remise avec son identifiant :

Récupérer l’identifiant de la remise à supprimer avec la commande suivante :

$ git stash list

7 Supprimer une remise individuelle utiliser la commande suivante :

$ git stash stash@{id de la remise} drop

8 Vider la liste des remises :

$ git stash drop

Cela aura pour effet de vider complétement la liste des remises.

# Interaction avec un repository distant (github) :

## Téléchargement des données distantes vers votre dépôt local (dossier)

1 La première des choses à faire quand on souhaite travailler sauvegarder son travail en ligne est de créer un dossier dans son ordinateur.

2 Sélectionner le dossier : clic droit -> Git Bash here ou cd url du dépôt local (ex : c:/users/test/desktop/depot\_local/.

3 Initialiser le dossier avec la commande ci-dessous :

$ git init

4 Clonage du dépôt distant (github) :

Pour plus d’information consulter la page 8 (Cloner un dépôt github (dépôt distant) sur votre repository (local)

Saisir la commande ci-dessous :

$ git remote add clone ***https://github.com/votre\_url.git***

Cela va connecter votre dépôt local(dossier) à votre dépôt distant(github).

5 Téléchargement des données distantes :

Pour télécharger le contenu d’un repository (github) nous allons utiliser la commande ci-dessous :

$ git pull ***https://github.com/votre\_url.git***

L’adresse du dépôt distant se trouve sur le site de github (bouton code -> https)

NB : *Pour pouvoir télécharger des données un fichier doit se trouver sur le dépôt distant (à la création veuillez penser à cocher la case add à readme file ou créer un fichier sur le repository github distant)*

## Envoi des données locales vers votre dépôt distant (repository github)

Il faut avoir téléchargé les données du dépôt distant en Amon (*git pull url\_dépôt\_github*).

1 Créer ou plusieurs fichiers dans votre dossier local puis ajoutez les a Git avec la commande :

$ git add **nom\_fichier.extension** ou git add \*(si plusieurs)

2 Sauvegarder les modifications avec la commande ci-dessous :

$ git commit –m ***“message modifications multiples***” ou -am

3 Envoie des données sur le repository (github) avec la commande ci-dessous :

$ git push ***https://github.com/votre\_url.git*** nom\_branche

Par défaut on utilisera la branche **master**.

*La première fois que vous envoyez des données à distance la console git va vous demander de valider vos identifiants de connexion de github et d’autoriser la connexion. Une fenêtre popup apparaitra elle va vous guider pour autoriser la connexion (cela se fait une seule fois). Suivant la taille, le nombre de fichiers, votre connexion internet cela sera plus ou moins long.*

NB : (Optionnel) :

Si on a créé une autre branche et que l’on souhaite l’envoyer en ligne (repository github on devra en premier lieu se positionner sur la branche désirée avec la commande :

$ git checkout nom\_branche

On réutilisera la commande git push url\_github nom\_de\_la\_branche. La branche se créera automatiquement sur github.

# Le système de branches :

## Présentation du système de branches :

Le principal atout de Git est son système de branches. Il faut voir le système de branche comme des « dossiers virtuel ».

La branche principale est appelée la branche master. C’est celle-ci, où au final, vous aurez à la fin toutes vos modifications. Le but est de ne surtout pas réaliser les modifications directement sur cette branche, mais de réaliser les modifications sur d’autres branches, et après tests, les intégrer sur la branche master.

**Branche Master**

**Branche Secondaire**

## Gestion des branches :

1 Afficher la liste des branches :

$ git branch

Cela va s’afficher comme ci-dessous : (*git branch*) :

git branch

***\*master***

Cette commande va afficher la liste des branches avec un astérisque à côté de la branche sélectionnée dans ce cas-là la branche **master**.

2 Créer une branche :

$ git branch **nom de la branche**

Ex : git branch secondaire (*création d’une nouvelle branche qui s’appelle* ***secondaire***).

3 Basculer sur la branche secondaire :

$ git checkout **secondaire**

La branche va fonctionner comme un dossier virtuel. Avec la commande **Git checkout**, on va être téléporté dans le dossier virtuel nom de la branche « (secondaire) ».

# Correction de problèmes Git :

1 Modification du message du dernier commit :

Dans le cas où vous souhaitez modifier le message de votre dernier commit nous allons utiliser la commande ci-dessous :

$ git commit --amend -m "***Votre nouveau message de commit***"

2 Ajouter un fichier oublié dans le dernier commit :

Vous avez fait votre commit mais vous réalisiez que vous avez oublié un fichier. Ce n'est pas bien grave ! Nous allons réutiliser la commande git --amend, mais d'une autre manière. La fonction git --amend, si vous avez bien compris, permet de modifier le dernier commit.

Nous allons donc réutiliser cette fonction, mais sans le -m qui permettait de modifier son message.

Nous allons dans un premier temps ajouter notre fichier, et dans un deuxième temps réaliser le git --amend.

$ git add **nom\_fichier.extension** ou git add \*(si plusieurs)

$ git commit --amend --no-edit

Votre fichier a été ajouté à votre commit et grâce à la commande --no-edit que nous avons ajoutée, nous n'avons pas modifié le message du commit.

Pour résumer, git commit --amend vous permet de sélectionner le dernier commit afin d'y ajouter de nouveaux changements en attente. Vous pouvez ajouter ou supprimer des changements afin de les appliquer avec

*Commit --amend*.

Si aucun changement n'est en attente*, --amend* vous permet de git commit --amend -le dernier message de log du commit avec *-m*. exemple ci-dessous :

$ git commit --amend –m "***Votre nouveau message de commit***"

3 Restauration de la branche master (*optionnel voir page 22*):

Si vous avez effectué un commit (update des fichiers) non désiré on peut restaurer nos fichiers à un état antérieur.

Pour se faire on va lancer la commande suivante pour récupérer l’id du commit :

-se positionner sur la branche master

$ git checkout master

-Afficher le log des commits pour récupérer l’identifiant avec la commande ci-dessous :

$ git log

-Exemple d’un commit affiché dans git log :

commit ***ca83a6dff817ec66f443420071545390a954664949*** *(identifiant*)

Author: ***nom\_utilisateur*** <***mail\_utilisateur***>

Date: Mon Mar 19 21:52:11 2019 -0700

-Récupérer l’identifiant ci-dessus :

-Assurez-vous de bien être sur la branche master.

-Taper la commande ci-dessous :

$ git reset -–hard HEAD^

-Basculez sur la branche ou vous voulez copier les modifications :

$ git checkout nom\_de\_la\_branche

-taper la commande ci-dessous : (on n’a besoin que des 8 premiers caractères de l’identifiant)

$ git reset -–hard ***ca83a6df***

-Cela va appliquer un commit (sauvegarde) sur la branche sélectionnée

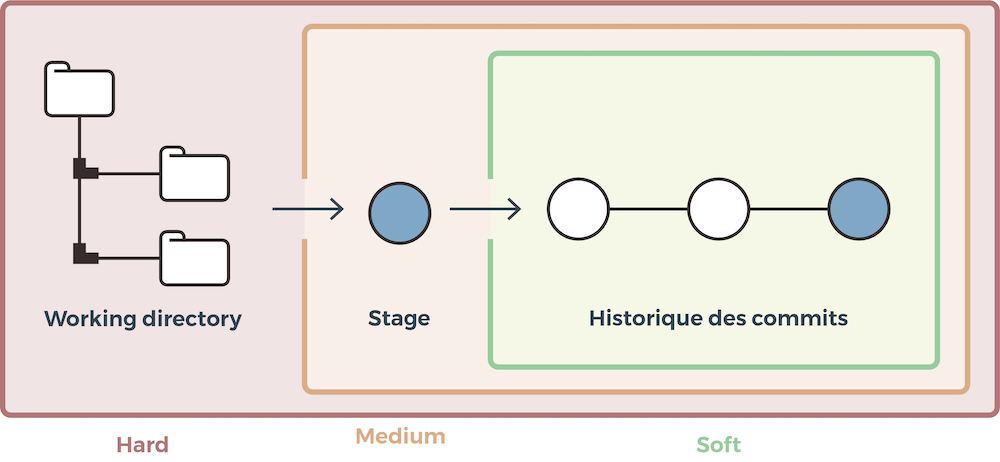
4 Corriger un mauvais commit à distance :

Pour corriger un mauvais commit (le dernier) envoyé avec un push sur votre Github saisir la commande ci-dessous :

$ git revert HEAD^

## Les 3 types de réinitialisation de GIT :

La commande git reset est un outil complexe et polyvalent pour **annuler les changements**. Elle peut être appelée de trois façons différentes, qui correspondent aux arguments de ligne de commande**--soft, --mixed et --hard**.



1 Commande Reset –hard :

Cette commande permet de revenir à n’importe quel commit mais il faut faire très attention car l’on **perd tous les commits suivants !!!**

Pour se faire il nous faut l’identifiant du commit que l’on obtient avec la commande :

$ git log

Puis on saisit la commande suivante :

$ git reset ***identifiant\_commit*** --hard

Cette commande est à utiliser en dernier recours car l’on **perd tous les commits suivants** !!!!

2 Commande Reset –mixed :

Le git reset --mixed va permettre de revenir juste après votre dernier commit ou le commit spécifier, sans supprimer vos modifications en cours. Il va par contre créer un HEAD détaché. Il permet aussi, dans le cas de fichiers indexés mais pas encore commités, de désindexer les fichiers.

Si rien n'est spécifié après git reset, par défaut il exécutera un git reset --mixed HEAD~

3 Commande Reset –soft :

Le git reset --Soft permet juste de se placer sur un commit spécifique afin de voir le code à un instant donné ou créer une branche partant d'un ancien commit. Il ne supprime aucun fichier, aucun commit, et ne crée pas de HEAD détaché.

## Scénarios d’erreurs :

**1° cas J’ai créé une branche que je n’aurais pas dû créer :**

Avant de créer une branche, vous devez créer votre branche principale (master). Pour créer la branche master, vous devez simplement ajouter un fichier et le commiter.

1 Créez un fichier "PremierFichier.txt" dans votre répertoire Test, et ajoutez-le avec la commande :

$ git add PremierFichier.txt

$ git commit ou git commit –m « message »

On vous demande alors d'indiquer le message du commit puis de valider. Pour valider le message, une fois que vous l'avez écrit, appuyez sur Echap (votre curseur va basculer sur la dernière ligne) et tapez :x où :x!

Cette commande va sauvegarder et quitter l'éditeur des messages de commit.

2 Nous allons maintenant créer une branche brancheTest avec la commande ci-dessous :

$ git branch brancheTest

Nous pouvons le vérifier avec la commande ci-dessous :

$ git branch

P317#yIS1

Youppiiii !

En fait, non, nous voulions ajouter nos fichiers avant de la créer et nous sommes maintenant bloqués avec cette branche que nous ne voulions pas tout de suite. Heureusement, il est très simple sous Git de supprimer une branche que nous venons de créer.

3 Pour cela, il suffit d'exécuter la commande :

$ git branch –d brancheTest

Attention, si toutefois vous avez déjà fait des modifications dans la branche que vous souhaitez supprimer, il faudra soit faire un commit de vos modifications, soit mettre vos modifications de côté.

4 Forcer la suppression en faisant :

$ git branch –D brancheTest

**Attention** : Forcer la suppression de cette manière entraînera la suppression de tous les fichiers et modifications que nous n'aurez pas commités sur cette branche.

**2 ° cas J’ai modifié la branche principale :**

Nous allons dans un premier temps voir ensemble le cas où vous avez modifié votre branche master mais que vous n'avez pas encore fait le commit, et nous verrons dans un second temps le cas où vous avez commité.

Vous avez modifié votre branche master avant de créer votre branche et vous n'avez pas fait le commit. Ce cas est un peu plus simple. Nous allons faire ce qu'on appelle une remise. La remise va permettre de mettre vos modifications de côté, le temps de créer votre nouvelle branche et ensuite appliquer cette remise sur la nouvelle branche.

Afin de voir comment cela fonctionne, allez sur votre branche master, modifiez des fichiers. Vous pouvez à tout moment voir à quel état sont vos fichiers en faisant :

$ git status

1 créer une remise :

$ git stash

git stash

Vous pouvez maintenant vous assurer que votre branche master est de nouveau propre, en faisant un nouveau

$ git status

Vous devriez avoir :

P342#yIS1

2 créer notre branche brancheCommit :

$ git branch brancheCommit

3 basculer sur la nouvelle branche :

$ git checkout brancheCommit

Et finalement, nous allons pouvoir appliquer la remise, afin de récupérer nos modifications sur notre nouvelle branche.

4 réaliser la remise :

$ git stash apply

Cette commande va appliquer la dernière remise qui a été faite. Si pour une raison ou une autre, vous avez créé plusieurs remises, et que la dernière n'est pas celle que vous souhaitiez appliquer, pas de panique, il est possible d'appliquer une autre remise. Nous allons d'abord regarder la liste de nos remises.

5 afficher la liste des remises :

$ git stash list

Cette commande va nous retourner un "tableau" des remises avec des identifiants du style :

P358#yIS1

Maintenant, admettons que vous ayez réalisé vos modifications et qu'en plus vous ayez fait le commit. Le cas est plus complexe, puisque vous avez enregistré vos modifications sur la branche master, alors que vous ne deviez pas.

6 modifiez des fichiers, et réalisez-le commit :

Nous allons devoir aller analyser vos derniers commits avec la fonction **git log**, afin de pouvoir récupérer l'identifiant du commit que l'on appelle couramment le *hash*. Par défaut, **git log** va vous lister par ordre chronologique inversé tous vos commits réalisés.

$ git log

commit ca83a6dff817ec66f443420071545390a954664949

Author: Mathieu <mathieu.mith@laposte.com>

Date: Mon Mar 19 21:52:11 2019 -0700

Maintenant que vous disposez de votre identifiant, gardez-le bien de côté. Vérifiez bien que vous êtes sur votre branche master

$ git checkout master

7 réalisez la commande suivante :

$ git reset -–hard HEAD^

Cette ligne de commande va permettre de supprimer de la branche master votre dernier commit.  Le Head^ indique que c'est bien le dernier commit que nous voulons supprimer.

8 nous allons maintenant créer notre nouvelle branche :

$ git branch brancheCommit

9 basculer sur cette branche :

$ git checkout brancheCommit

Maintenant que nous sommes sur la bonne branche, nous allons de nouveau faire un **git reset**, mais celui-ci va permettre d'appliquer ce commit sur notre nouvelle branche ! Il n'est pas nécessaire d'écrire l'identifiant en entier. Seuls les 8 premiers caractères sont nécessaires.

$ git reset --hard identifiant

Notre cas est résolu.

**3 ° cas je souhaite changer le message de mon commit :**

Lorsque l'on travaille sur un projet avec Git, il est très important, lorsque l'on propage les modifications, de bien marquer dans le message descriptif les modifications que l'on a effectuées. Si jamais ne vous faites une erreur dans l'un de vos messages de commit, il est tout à fait possible de changer le message après coup.

**Attention** cette commande va fonctionner sur votre dernier commit.

Imaginons que vous veniez de faire un commit et que vous ayez fait une erreur dans votre message. L'exécution de cette commande, lorsqu'aucun élément n'est encore modifié, vous permet de modifier le message du commit précédent sans modifier son instantané. L'option **-m** permet de transmettre le nouveau message.

$ git commit --amend -m "Votre nouveau message de commit"

git commit --amend -m "Votre nouveau message de commit"

On peut vérifier avec la commande **git log**.

**4 ° cas J’ai oublié un fichier dans mon dernier commit** :

Imaginons maintenant que vous ayez fait votre commit mais que vous réalisiez que vous avez oublié un fichier. Nous allons réutiliser la commande git **--amend**, mais d'une autre manière. La fonction **git --amend**, si vous avez bien compris, permet de modifier le dernier commit.

Nous allons donc réutiliser cette fonction, mais sans le **-m** qui permettait de modifier son message.

1 ajouter votre fichier, et dans un deuxième temps réaliser le **git --amend** :

$ git add *nom du fichier.extension*

$ git –-amend –-no-edit

Votre fichier a été ajouté à votre commit et grâce à la commande --no-edit que nous avons ajoutée, nous n'avons pas modifié le message du commit.

Pour résumer, **git commit --amend** vous permet de sélectionner le dernier commit afin d'y ajouter de nouveaux changements en attente. Vous pouvez ajouter ou supprimer des changements afin de les appliquer avec commit --amend. Si aucun changement n'est en attente, **--amend** vous permet de modifier le dernier message de log du commit avec **-m**.

**Corrigez vos erreurs en local et à distance**:

Vous avez par mégarde push des fichiers erronés. Le problème, c'est que maintenant ce n'est plus que sur votre dépôt local, mais à disposition de tout le monde.

Il est possible d'annuler son commit public avec la commande **Git revert**. L'opération Revert annule un commit en créant un nouveau **commit**. C'est une méthode sûre pour **annuler des changements**, car elle ne risque pas de **réécrire l'historique du commit**.

2 utiliser la commande :

$ git revert HEAD^

Nous avons maintenant revert notre dernier commit public et cela a créé un nouveau commit d'annulation. Cette commande n'a donc **aucun impact sur l'historique** ! Par conséquent, il vaut mieux utiliser git revert pour annuler des changements apportés à une branche publique, et git reset pour faire de même, mais sur une branche privée.

# Fusion (merge):

**Comment fonctionne la fusion sous Git ?**

Il est très courant sous Git de vouloir fusionner le travail fait sur différentes branches. Pour cela, nous avons la fonction **Merge**. Un git merge ne devrait être utilisé que pour la récupération fonctionnelle, intégrale et finale d’une branche dans une autre, afin de préserver un graphe d’historique sémantiquement cohérent et utile, lequel représente une véritable valeur ajoutée. Comme son nom l’indique, merge réalise une **fusion**. git merge va combiner plusieurs séquences de commits en un historique unifié. Le plus souvent, git merge est utilisé pour combiner deux branches. git merge va créer un nouveau commit de merge.

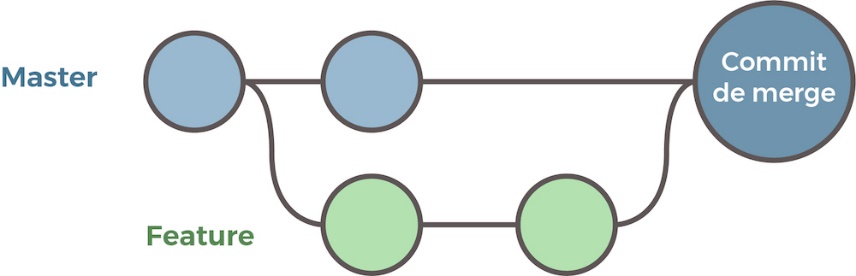
Imaginons que vous ayez votre branche master et une branche "Nouvelle fonctionnalité". Nous souhaitons maintenant faire un merge de cette branche de fonctionnalité dans la branche master. Appeler cette commande permettra de merger la fonctionnalité de branche spécifiée dans la branche courante, disons master.

**Attention** Il faut toujours préparer le terrain avant de réaliser un merge !

Vous devez toujours vous assurer d'être sur **la bonne branche**. Pour cela, vous pouvez réaliser un git status. Si vous n'êtes pas sur la bonne, réalisez un git checkout, pour changer de branche. Maintenant que le terrain est prêt, vous pouvez réaliser votre merge.

$ git merge ***nom\_branche***

Votre branche Nouvelle fonctionnalité va être fusionnée sur la branche master en créant un nouveau commit.



Si les deux branches que vous essayez de fusionner modifient toutes les deux la même partie du même fichier, Git ne peut pas déterminer la version à utiliser. Lorsqu'une telle situation se produit, Git s'arrête avant le commit de merge, afin que vous puissiez résoudre manuellement les conflits.

Merge une branche utiliser les commandes suivantes :

1 Se déplacer sur la branche à merge :

$ git checkout branche\_qui\_va\_recevoir\_le\_merge

2 Merge la branche :

$ git merge Nom\_branche\_à\_merge