Initiation à Git, GitHub et GitLab

KAHOUADJI Mouad

Mouad.kahouadji.int@groupe-gema.com

Versionner en Local

- La première chose à faire après l'installation de Git est de renseigner votre nom et votre adresse de courriel. C'est une information importante car toutes les validations dans Git utilisent cette information et elle est indélébile dans toutes les validations que vous pourrez réaliser :
- ***git config --global user.name "Votre nom"***
- ***git config --global user.email nom@example.com***

Vous pouvez voir tous vos paramétrages et d'où ils viennent en utilisant :

- ***git config --list --show-origin***

Versionner en Local

- **Votre éditeur de texte :**

À présent que votre identité est renseignée, vous pouvez configurer l'éditeur de texte qui sera utilisé quand Git vous demande de saisir un message.

Par défaut, Git utilise l'éditeur configuré au niveau système, qui est généralement Vi ou Vim.

Si vous souhaitez utiliser un éditeur de texte différent, comme Notepad++, vous pouvez entrer ce qui suit :

git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"

Versionner en Local

- La commande **git init** : initialise le dépôt (de mettre sur le bon dossier)

La commande **git init** crée un nouveau dépôt Git. Elle permet de convertir un projet existant, sans version en un dépôt Git ou d'initialiser un nouveau dépôt vide.

La plupart des autres commandes Git ne sont pas disponibles hors d'un dépôt initialisé, il s'agit donc généralement de la première commande que vous exécuterez dans un nouveau projet.

L'exécution de **git init** créer un sous-répertoire .git dans le répertoire de travail actif, qui contient toutes les métadonnées Git nécessaires pour le nouveau dépôt. Ces métadonnées incluent des sous-répertoires pour des objets, des réfs et des fichiers de modèle. Un fichier HEAD est également créé et pointe vers le commit actuellement extrait.

Versionner en Local

La commande ***git add*** ajoute un changement dans le répertoire de travail à la zone de staging. Elle informe Git que vous voulez inclure les mises à jour dans un fichier particulier du commit suivant.

Cependant, ***git add*** n'impacte pas le dépôt de manière significative.

Les changements ne sont pas réellement enregistrés jusqu'à ce que vous exécutiez ***git commit***.

Versionner en Local

La commande ***git commit*** capture un instantané des changements actuellement stagés du projet.

Les instantanés commités peuvent être considérés comme des versions « sûres » d'un projet.

Avant d'exécuter ***git commit***, la commande ***git add*** est utilisée pour promouvoir ou « stager » les changements apportés au projet qui seront stockés dans un commit.

Ces deux commandes ***git commit*** et ***git add*** font partie des plus fréquemment utilisées

Versionner en Local

git commit –amend : Cette option ajoute un autre niveau de fonctionnalité à la commande de commit. Lorsque vous transmettez cette option, le dernier commit sera modifié. Au lieu de créer un commit, les changements stagés sont ajoutés au commit précédent. Cette commande ouvre l'éditeur de texte configuré pour le système et invite à modifier le message de commit précédemment indiqué.

Versionner en Local

La commande ***git status :*** Affiche les chemins qui ont des différences entre le fichier d'index et le commit HEAD actuel, les chemins qui ont des différences entre l'arbre de travail et le fichier d'index, et les chemins dans l'arbre de travail qui ne sont pas suivis par Git (et ne sont pas ignorés par gitignore).

Les premiers sont ce que vous commettriez en courant git commit; les deuxième et troisième sont ce que vous pouvez valider en exécutant git add avant d'exécuter git commit.

Versionner en Local

La commande ***git diff*** ***:*** Affiche les modifications entre l'arbre de travail et l'index ou un arbre, les modifications entre l'index et un arbre, les modifications entre deux arbres, les modifications résultant d'une fusion, les modifications entre deux objets blobs ou les modifications entre deux fichiers sur disque.

Versionner en Local

La commande ***git log :*** Affiche les journaux de validation.

La commande prend les options applicables à la commande git rev-list pour contrôler ce qui est montré et comment, et les options applicables aux commandes git diff' pour contrôler la façon dont les modifications que chaque validation introduit sont affichées.

- Git log n-2 : affiche les 2 derniers commit

Versionner en Local

La commande ***git show*** Affiche un ou plusieurs objets (blobs, arbres, étiquettes et commits).

Pour les commits, il affiche le message de log et la différence textuelle. Il présente également le commit de fusion dans un format spécial comme produit par ’git diff-tree --cc'.

Pour les étiquettes, il affiche le message de l’étiquette et les objets référencés.

Pour les arbres, il affiche les noms (équivalent à’git ls-tree' avec --name only).

Pour les blobs simples, il affiche le contenu simple.

La commande prend les options applicables à la commande’git diff-tree' pour contrôler comment les changements introduits par le commit sont affichés.

Exemple ***git show*** « sha-1 »

Versionner en Local

La commande ***git checkout*** Met à jour les fichiers dans l'arbre de travail pour correspondre à la version dans l'index ou dans l'arbre spécifié.

Si aucun spécificateur de chemin n'est fourni, git checkout met aussi à jour HEAD pour positionner la branche spécifiée comme la branche actuelle.

Exemple :

- ***git checkout « sha-1 »*** remettre la version du commit avec « sha-1 »
- ***git checkout main*** remettre la version la plus récente

Cloner un dépôt Git

La deuxième façon de démarrer un dépôt Git est de cloner localement un dépôt Git déjà existant. Pour cela, on va utiliser la commande Git clone.

En pratique, dans la grande majorité des cas, nous clonerons des dépôts Git distants, c'est-à-dire des dépôts hébergés sur serveur distants pour pouvoir contribuer à ces projets.

Cependant, nous pouvons également cloner des dépôts locaux. Nous parlerons des dépôts distants et apprendrons à les cloner lorsqu'on abordera GitHub. Pour le moment, contentons nous d'essayer de cloner notre dépôt "projet-git" tout juste créé.

Cloner un dépôt Git

Pour cela, on va se placer sur le bureau. Comme je suis pour le moment situé dans mon répertoire “projet-git”, j’utilise la commande Bash cd .. pour atteindre le répertoire parent (c’est-à-dire mon bureau).

J’utilise ensuite la commande git clone en lui passant d’abord le chemin complet du projet à cloner (qui correspond à son nom dans notre cas puisque le répertoire du projet est également sur le bureau) puis le chemin complet du clone qui doit être créé. On va choisir de créer le clone sur le bureau par simplicité et on va donc simplement passer un nom à nouveau. Appelons le clone “DeuxiemProjet” par exemple comme ceci :

Cloner un dépôt Git

```
gcher@MSI MINGW64 ~/Desktop
$ git clone PremierProjet/ DeuxiemProjet
Cloning into 'DeuxiemProjet'...
done.

gcher@MSI MINGW64 ~/Desktop
$
```

- On peut cd dans le répertoire du projet et effectuer un ultime git status pour s'assurer de l'état des fichiers du répertoire :

Supprimer un fichier d'un projet et / ou l'exclure du suivi de version Git

Concernant la suppression de fichiers, il existe plusieurs situations possibles en fonction de ce que vous souhaitez réellement faire : voulez vous simplement exclure un fichier du suivi de version mais le conserver dans votre projet ou également le supprimer du projet ?

Pour supprimer un fichier et l'exclure du suivi de version, nous allons utiliser la commande git rm (et non pas simplement une commande Bash rm).

Pour simplement exclure un fichier du suivi Git mais le conserver dans le projet, on va utiliser la même commande git rm mais avec cette fois-ci une option --cached.

Retirer un fichier de la zone d'index de Git

Le contenu de la zone d'index est ce qui sera proposé lors du prochain commit. Imaginons qu'on ait ajouté un fichier à la zone d'index par erreur. Pour retirer un fichier de l'index, on peut utiliser git reset HEAD nom-du-fichier.

A la différence de git rm, le fichier continuera d'être suivi par Git. Seulement, le fichier dans sa version actuelle va être retiré de l'index et ne fera donc pas partie du prochain commit.

Renommer un fichier dans Git

On peut également renommer un fichier de notre projet depuis Git en utilisant cette fois-ci une commande git mv ancien-nom-fichier nouveau-nom-fichier.

On peut par exemple renommer notre fichier “README.txt” en “LISEZMOI.txt” de la manière suivante :

```
git mv README.txt  LISEZMOI.txt
```

Consulter l'historique des modifications Git

La manière la plus simple de consulter l'historique des modifications Git est d'utiliser la commande git log. Cette commande affiche la liste des commits réalisés du plus récent au plus ancien. Par défaut, chaque commit est affiché avec sa somme de contrôle SHA-1, le nom et l'e-mail de l'auteur, la date et le message du commit.

La commande git log supporte également de nombreuses options. Certaines vont pouvoir être très utiles comme par exemple les options -p, --pretty, --since ou --author.

Consulter l'historique des modifications Git

Utiliser git log -p permet d'afficher explicitement les différences introduites entre chaque validation.

L'option --pretty permet, avec sa valeur oneline, d'afficher chaque commit sur une seule ligne ce qui peut faciliter la lecture dans le cas où de nombreux commits ont été réalisés, et elle s'écrit : git log --pretty=oneline.

L'option --since permet de n'afficher que les modifications depuis une certaine date (on peut lui fournir différents formats de date comme 2.weeks ou 2019-10-10 par exemple).

L'option --author permet de n'afficher que les commits d'un auteur en particulier.

Ecraser et remplacer un commit

Parfois, on voudra annuler une validation (un commit), notamment lorsque la validation a été faite en oubliant des fichiers ou sur les mauvaises versions de fichiers.

La façon la plus simple d'écraser un commit est d'utiliser la commande git commit avec l'option --amend. Cela va pousser un nouveau commit qui va remplacer le précédent en l'écrasant.

Annuler des modifications apportées à un fichier

L’un des principaux intérêts d’utiliser un logiciel de gestion de vision est de pouvoir “roll back”, c’est-à-dire de pouvoir revenir à un état antérieur enregistré d’un projet.

Après un commit, on va continuer à travailler sur nos fichiers et à les modifier. Parfois, certaines modifications ne vont pas apporter les comportements espérés et on voudra revenir à l’état du fichier du dernier instantané Git (c’est-à-dire au dernier état enregistré). On va pouvoir faire cela avec la commande générale git checkout -- nom-du-fichier ou la nouvelle commande spécialisée git restore.

Qu’est ce qu’une branche ?

Qu’est ce qu’une branche ?

Créer une branche, c’est en quelque sorte comme créer une “copie” de votre projet pour développer et tester de nouvelles fonctionnalités sans impacter le projet de base.

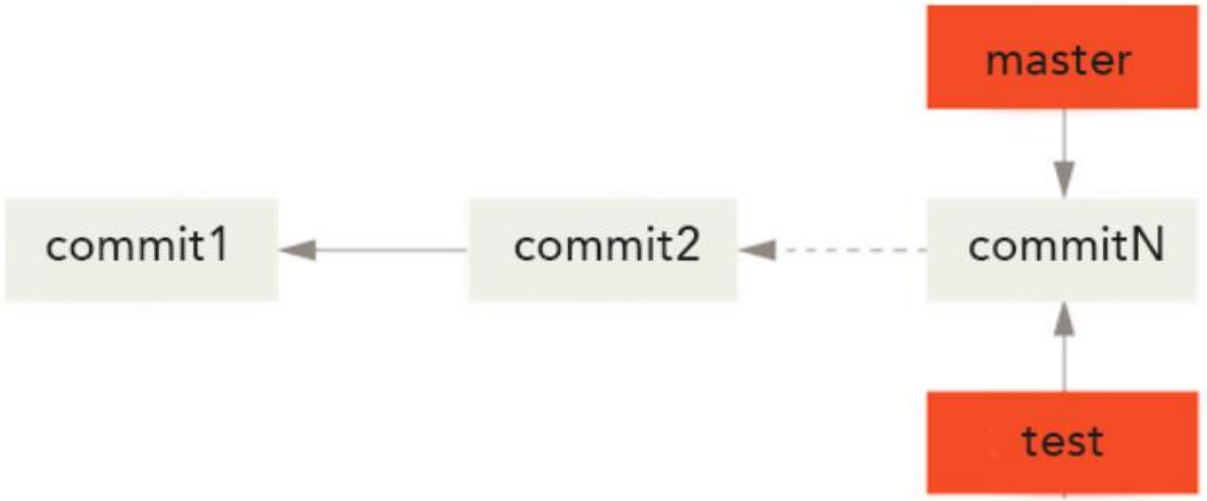
Git a une approche totalement différente des branches qui rend la création de nouvelles branches et la fusion de branche très facile à réaliser. Une branche, dans Git, est simplement un pointeur vers un commit (une branche n’est qu’un simple fichier contenant les 40 caractères de l’empreinte SHA-1 du commit sur lequel elle pointe).

Créer une nouvelle branche

Pour créer une nouvelle branche, on utilise la commande

git branch nom-de-la-branche.

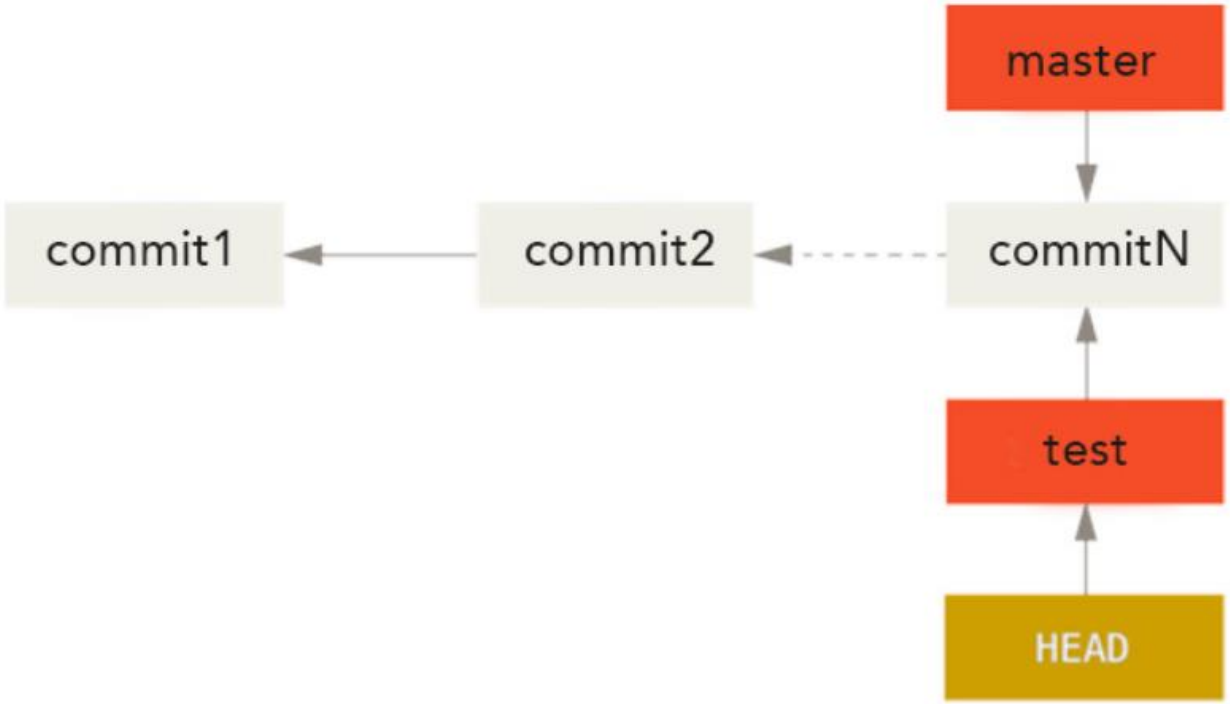
Cette syntaxe va créer un nouveau pointeur vers le dernier commit effectué (le commit courant). A ce stade, vous allez donc avoir deux branches (deux pointeurs) vers le dernier commit : la branche master et la branche tout juste créée.



Créer une nouvelle branche

Notez que la commande git branch permet de créer une nouvelle branche mais ne déplace pas HEAD.

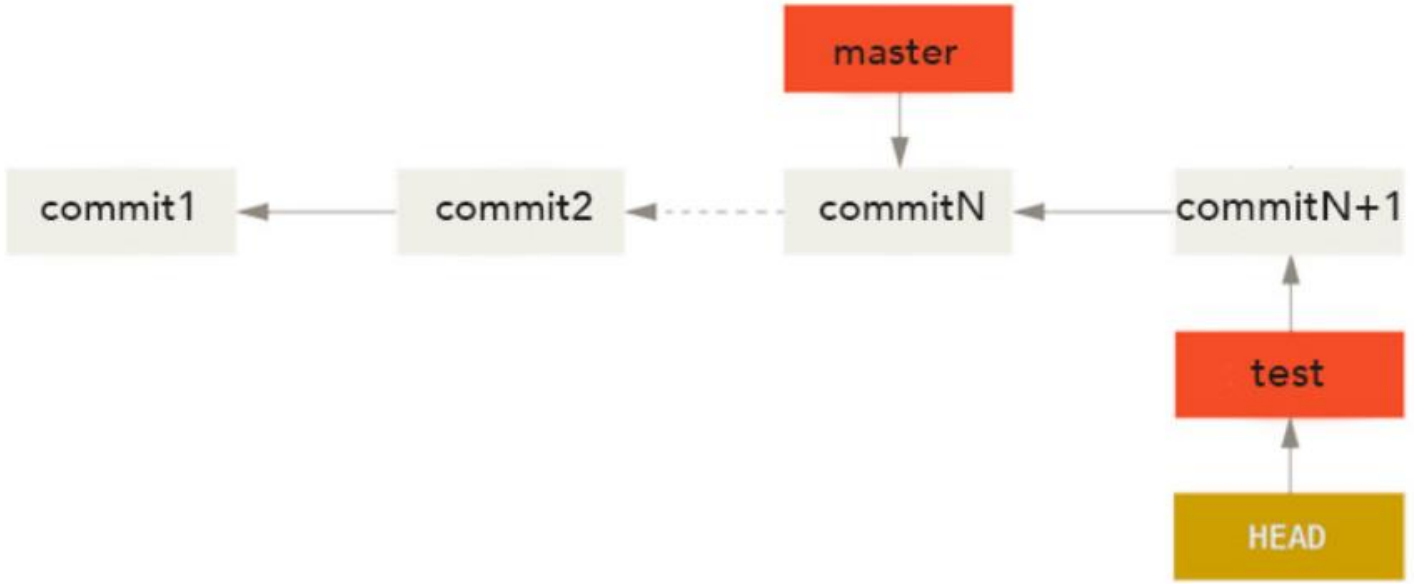
Nous allons donc devoir déplacer explicitement HEAD pour indiquer à Git qu'on souhaite basculer sur une autre branche. On utilise pour cela la commande git checkout suivi du nom de la branche sur laquelle on souhaite basculer.



Créer une nouvelle branche

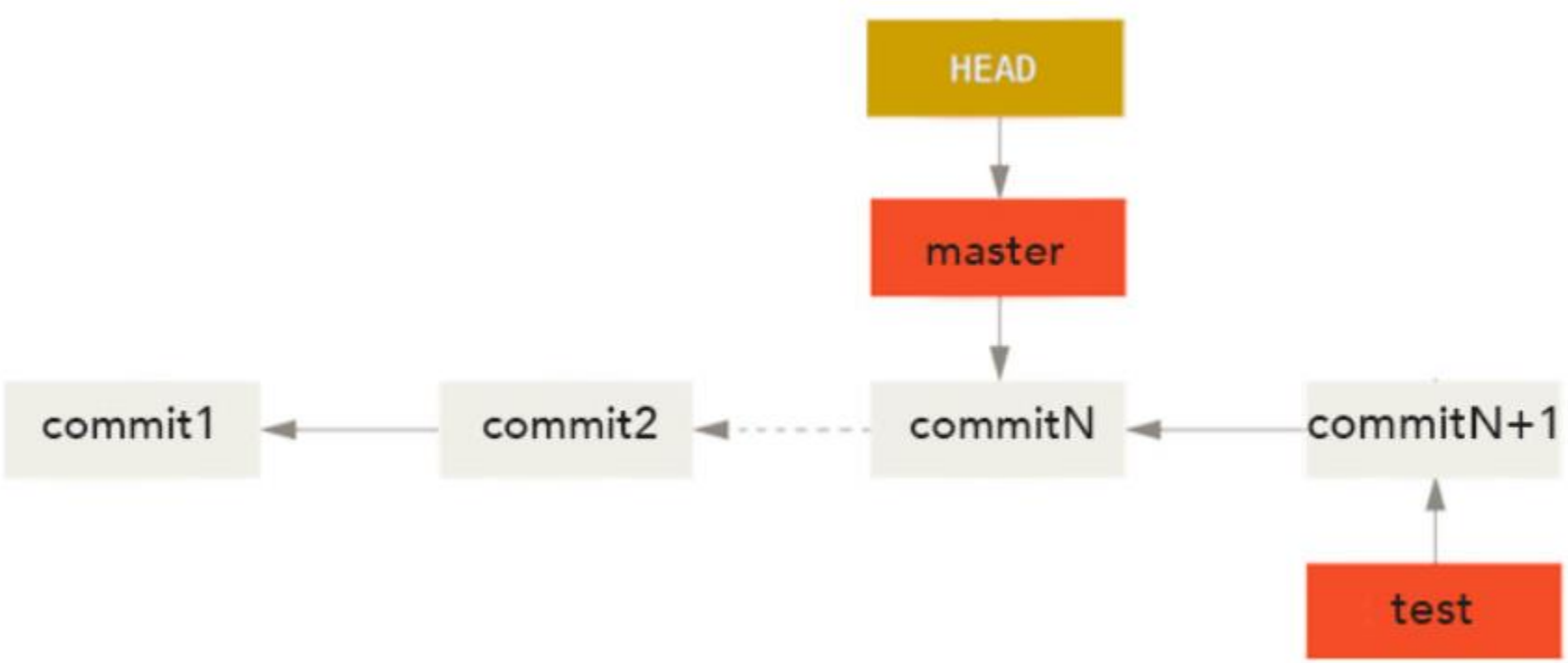
Note : On peut également utiliser git checkout -b pour créer une branche et basculer immédiatement dessus. Cela est l'équivalent d'utiliser git branch puis git checkout.

HEAD pointe maintenant vers notre branche test. Si on effectue un nouveau commit, la branche test va avancer automatiquement tandis que master va toujours pointer sur le commit précédent. C'est en effet la branche sur laquelle on se situe lors d'un commit qui va pointer sur ce commit.



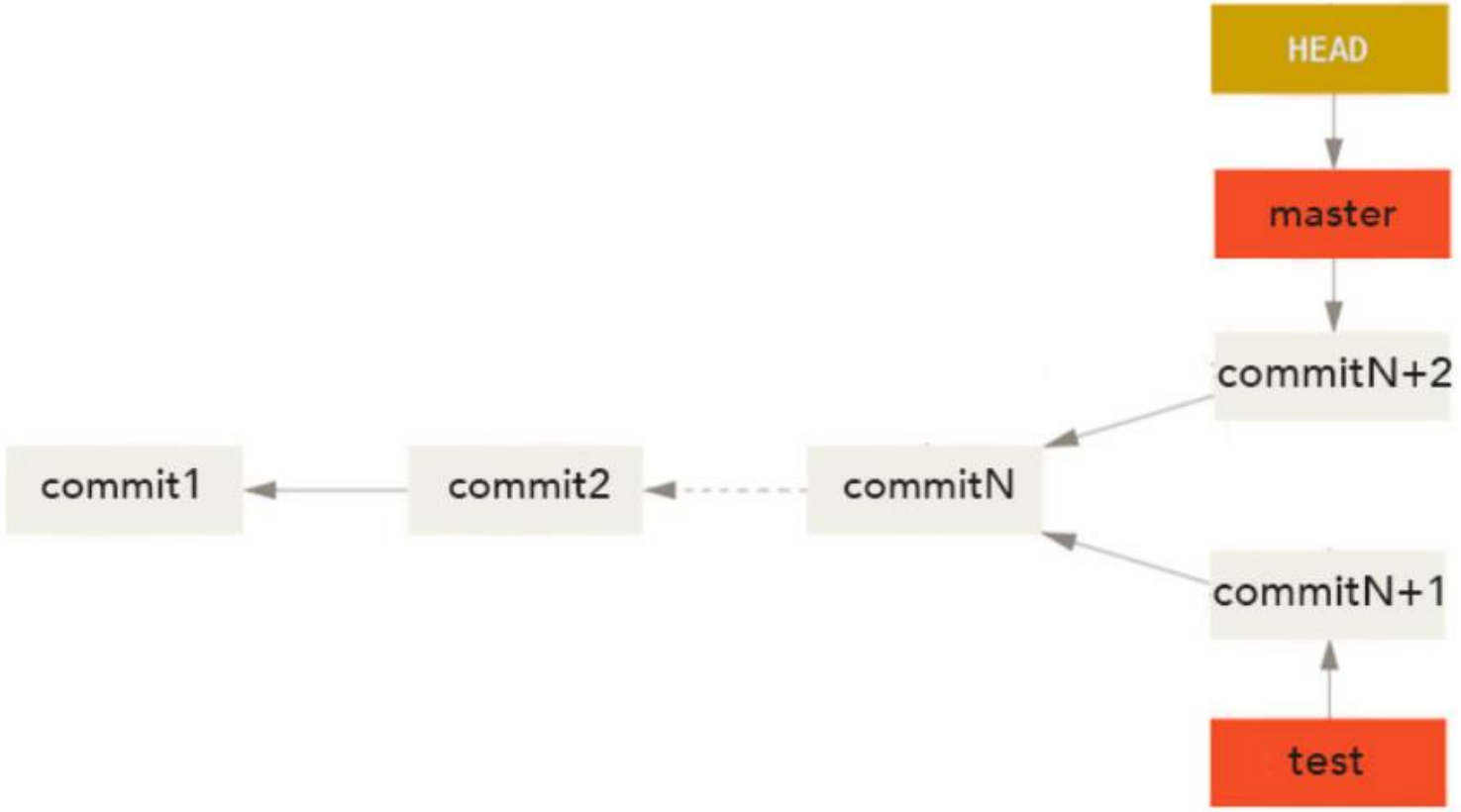
Basculer entre les branches

On peut revenir sur notre branche master en tapant à nouveau une commande git checkout master. Cela replace le pointeur HEAD sur la branche master et restaure le répertoire de travail dans l'état de l'instantané pointé par le commit sur lequel pointe master.



Basculer entre les branches

Si on modifie alors le répertoire de travail et qu'on effectue un nouveau commit, les deux branches master et test vont diverger. On va donc avoir deux branches pointant vers des instantanés qui ont enregistré le projet dans deux états différents.



Fusionner des branches

Dans la précédente, on avait fini avec deux branches master et ‘mabranch’ divergentes. On parle de divergence car les deux branches possèdent un ancêtre commit en commun mais pointent chacune vers de nouveaux commits qui peuvent correspondre à des modifications différentes d’un même fichier du projet.

Pour fusionner nos deux branches, on va se placer sur master avec une commande git checkout puis taper une commande git merge avec le nom de la branche qu’on souhaite fusionner avec master.

Fusionner des branches

Ici, comme la situation est plus complexe, il me semble intéressant d'expliquer comment Git fait pour fusionner les branches. Dans ce cas, Git réalise une fusion en utilisant 3 sources : le dernier commit commun aux deux branches et le dernier commit de chaque branche.

Cette fois-ci, plutôt que de simplement faire un fast forward, Git crée automatiquement un nouvel instantané dont le contenu est le résultat de la fusion ainsi qu'un commit qui pointe sur cet instantané. Ce commit s'appelle un commit de fusion et est spécial puisqu'il possède plusieurs parents.

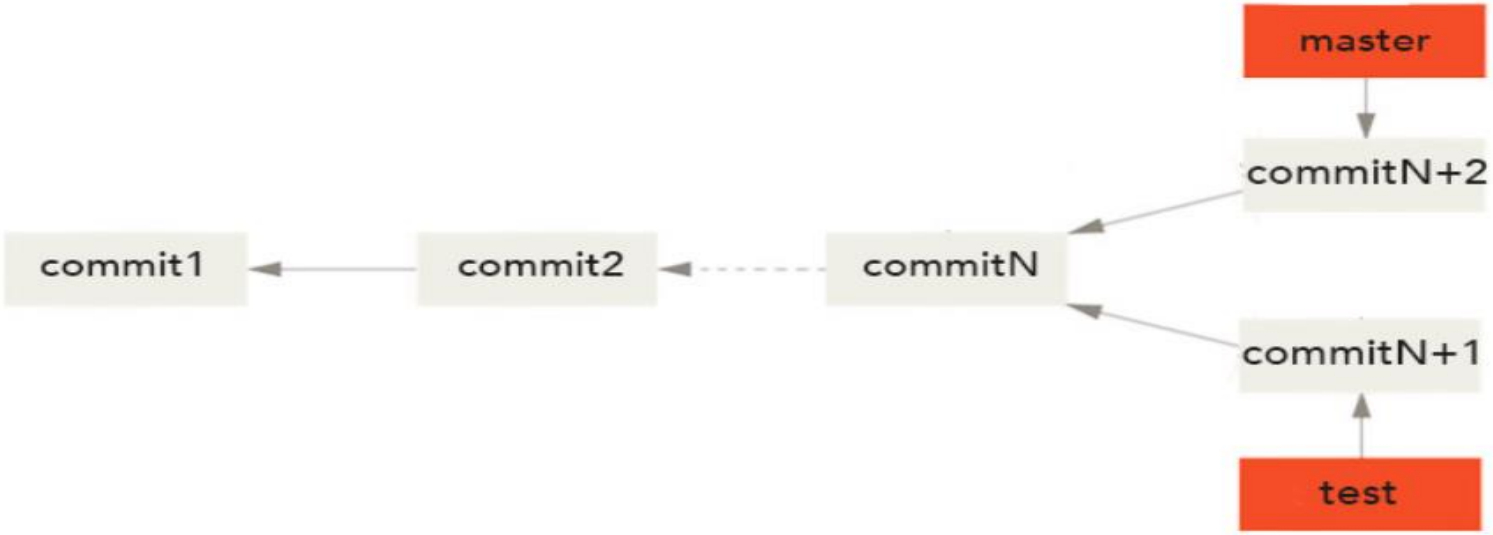
Fusionner des branches

Notez que dans le cas d’une fusion à trois sources, il se peut qu’il y ait des conflits. Cela va être notamment le cas si une même partie d’un fichier a été modifiée de différentes manières dans les différentes branches. Dans ce cas, lors de la fusion, Git nous alertera du conflit et nous demandera de le résoudre avant de terminer la fusion des branches.

Rebaser

Git nous fournit deux moyens de rapatrier le travail effectué sur une branche vers une autre : on peut fusionner ou rebaser. Nous allons maintenant nous intéresser au rebasage, comprendre les différences entre la fusion et le rebasage et voir dans quelle situation utiliser une opération plutôt qu'une autre.

Reprenons notre exemple précédent avec nos deux branches divergentes



Rebaser

Plutôt que d'effectuer une fusion à trois sources, on va pouvoir rebaser les modifications validées dans commitN+1 dans notre branche master. On utilise la commande git rebase pour récupérer les modifications validées sur une branche et les rejouer sur une autre.

Dans ce cas, Git part à nouveau du dernier commit commun aux deux branches (l'ancêtre commun le plus récent) puis récupère les modifications effectuées sur la branche qu'on souhaite rapatrier et les applique sur la branche vers laquelle on souhaite rebaser notre travail dans l'ordre dans lequel elles ont été introduites.

Le résultat final est le même qu'avec une fusion mais l'historique est plus clair puisque toutes les modifications apparaissent en série même si elles ont eu lieu en parallèle. Rebaser rejoue les modifications d'une ligne de commits sur une autre dans l'ordre d'apparition, alors que la fusion joint et fusionne les deux têtes.

Rebaser

Plutôt que d'effectuer une fusion à trois sources, on va pouvoir rebaser les modifications validées dans commitN+1 dans notre branche master. On utilise la commande git rebase pour récupérer les modifications validées sur une branche et les rejouer sur une autre.

Dans ce cas, Git part à nouveau du dernier commit commun aux deux branches (l'ancêtre commun le plus récent) puis récupère les modifications effectuées sur la branche qu'on souhaite rapatrier et les applique sur la branche vers laquelle on souhaite rebaser notre travail dans l'ordre dans lequel elles ont été introduites.

Le résultat final est le même qu'avec une fusion mais l'historique est plus clair puisque toutes les modifications apparaissent en série même si elles ont eu lieu en parallèle. Rebaser rejoue les modifications d'une ligne de commits sur une autre dans l'ordre d'apparition, alors que la fusion joint et fusionne les deux têtes.

Rebaser

Gardez cependant à l'esprit que rebaser équivaut à supprimer des commits existants pour en créer de nouveaux (qui sont similaires de par leur contenu mais qui sont bien des entités différentes). Pour cette raison, vous ne devez jamais rebaser des commits qui ont déjà été poussés sur un dépôt public.

En effet, imaginons la situation suivante :

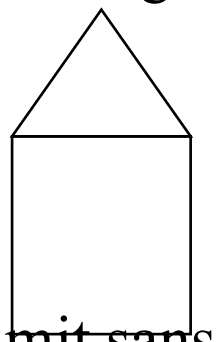
1. Vous poussez des commits sur un dépôt public;
2. Des personnes récupèrent ces commits et se basent dessus pour travailler ;
3. Vous utilisez un git rebase pour "réécrire" ces commits et les poussez à nouveau.

Dans ce cas, des problèmes vont se poser puisque les gens qui ont travaillé à partir des commits de départ ne vont pas les retrouver dans le projet si il veulent récupérer les mises à jour et lorsqu'ils vont pousser leur modification sur le dépôt public les commits effacés vont être réintroduits ce qui va créer un historique très confus et potentiellement des conflits.

Travaux Pratique

Repronons le dossier que vous avez crée précédemment :

1. Avec la commande touch, créer 3 fichier (avec le nom que vous voulez)
    - Fichier 1 un programme python (fonction pour le calcule de la surface d'un rectangle).
    - Fichier 2 un programme python (fonction pour théorème de Pythagore).
    - Fichier 1 un programme python (fonction pour le calcule de la surface de :)



2. Modifier le Fichier 2 pour qu'il accepte que les float et int et créez un commit sans prend en considération Fichie 1 et 3.

Travaux Pratique

3. Cloner le dossier dans un autre dossier-git (vous pouvez choisir le nom que vous voulez).
4. Créez une branch (nommez la test) et switchez sur cette branch
    - Avec la méthode git bracnh
    - Avec une méthode direct (création et positionnement en même temps
5. Changer le nom de fichier 1 et committez.
6. Affichez l'historique (chaque commit sur une seul ligne)
7. Sur la même branch modifier le fichier 2 (changer le nom de variable)
8. Fusionnez le deux brache avec merg puis avec rebase
9. Conclusion

Contribuer à un projet avec GitHub ou le copier

Sur GitHub, nous allons pouvoir contribuer aux projets publics d'autres personnes ou créer nos propres dépôts publics afin que d'autres personnes contribuent à nos propres projets. Commençons déjà par nous familiariser avec l'aspect contributeur de GitHub.

GitHub est une gigantesque plateforme collaborative qui héberge des dépôts Git. Pour rechercher des dépôts auxquels contribuer ou pour rechercher des fonctionnalités intéressantes qu'on aimerait récupérer, on peut aller dans l'onglet "explore" ou chercher un dépôt en particulier grâce à la barre de recherche en haut du site.

Les étapes pour contribuer à un projet

Les étapes pour contribuer à un projet (le cycle de travail) vont toujours être les mêmes :

1. On copie un projet sur notre espace GitHub
2. On créer une branche thématique à partir de master ;
3. On effectue nos modifications / améliorations au projet ;
4. On pousse ensuite la branche locale vers le projet qu'on a copié sur GitHub et on ouvre une requête de tirage depuis GitHub ;
5. Le propriétaire du projet choisit alors de refuser nos modifications ou de les fusionner dans le projet d'origine ;
6. On met à jour notre version du projet en récupérant les dernière modifications à partir du projet d'origine.

Copier un dépôt : clone vs fork

Pour copier un dépôt (repository) GitHub sur nos machines, il suffit d'utiliser l'option "clone URL" de GitHub pour récupérer le lien du repo puis d'utiliser la commande git clone [URL] dans notre console.

On peut également utiliser l'option "fork" de GitHub. Un fork est une copie d'un dépôt distant qui nous permet d'effectuer des modifications sans affecter le projet original.

Copier un dépôt : clone vs fork

La différence entre un fork et un clone est que lorsqu'on fork une connexion existe entre notre fork (notre copie) et le projet original. Cela permet notamment de pouvoir très simplement contribuer au projet original en utilisant des pull requests, c'est-à-dire en poussant nos modifications vers le dépôt distant afin qu'elles puissent être examinées par l'auteur du projet original.

Lorsqu'on clone un projet, on ne va pas pouvoir ensuite récupérer les changements à partir du projet d'origine ni contribuer à ce projet à moins que le propriétaire du projet d'origine ne nous accorde des droits spéciaux (privilèges).

Créer une branche thématique à partir de la branche principale

Pour contribuer à un projet, on va donc très souvent le copier en le forkant. Cela crée une copie du projet dans notre espace GitHub. On va ensuite créer une branche thématique et effectuer nos modifications.

Pour cela, une fois sur la page du projet forké dans notre espace personnel Git, on va cliquer sur le bouton de liste “branch” et ajouter un nom pour créer une nouvelle branche.

Téléchargement d’un dépôt

- En utilisant la commande : git clone avec URL de repository.
- URL on le récupère de la page de Github.
- Une fois le répertoire sera dupliquer sur notre machine, on peut effectuer des modification avec les commande vue précédemment

Mettre le dépôt à jours

- En utilisant la commande : git pull
- Les modification faites par les contributeurs sera afficher une fois on exécute cette commande

Envoyer le modifications vers le Depot distant

- En utilisant la commande : git push origin master
- Une fois les modifications sont faites, on envoie vers le dépôt distant le travaille effectué

Empêcher l'indexation de certains fichiers dans Git

Lorsqu'on dispose d'un projet et qu'on souhaite utiliser Git pour effectuer un suivi de version, il est courant qu'on souhaite exclure certains fichiers du suivi de version comme certains fichiers générés automatiquement, des fichiers de configuration, des fichiers sensibles, etc.

On peut informer Git des fichiers qu'on ne souhaite pas indexer en créant un fichier .gitignore et en ajoutant les différents fichiers qu'on souhaite ignorer. Notez qu'on peut également mentionner des schémas de noms pour exclure tous les fichiers correspondant à ce schéma et qu'on peut même exclure le contenu entier d'un répertoire en écrivant le chemin du répertoire suivi d'un slash.