Ouvrir un terminal (terminal Git Bash à privilégier)

Créer, en ligne de commande, un répertoire tp-git

cd desktop

mkdir tp-git

Se déplacer dans le répertoire

cd tp-git

Vérifier qu'on est dans le bon répertoire en affichant le chemin du répertoire courant

pwd

Initialiser un dépôt Git

git init

Lister tous les fichiers du répertoire (y compris les fichiers cachés) pour s'assurer que le répertoire .git à été créé

ls -a

Ouvrir ce répertoire sous VS Code

code .

Exécuter git status et copier/coller la sortie

```
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

Créer le fichier fichier1.md avec un contenu quelconque et l'enregistrer

Créer le fichier fichier2.md avec un contenu quelconque et l'enregistrer

touch fichier1.md, echo "Contenu quelconque1"

touch fichier2.md, echo "Contenu quelconque2"

Exécuter git status et copier/coller la sortie

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        fichier1.md
        fichier2.md

nothing added to commit but untracked files present (use "git add" to track)
```

Ajouter fichier1.md à l'index de Git (zone de Staging)

```
git add fichier1.md
```

Exécuter git status et copier/coller la sortie

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   fichier1.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        fichier2.md
```

Créer un commit avec pour message : "Ajout de fichier1.md"

```
git commit -m "Ajout de fichier1.md"
```

Exécuter git status et copier/coller la sortie

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        fichier2.md

nothing added to commit but untracked files present (use "git add" to track)
```

Modifier fichier1.md et enregistrer

```
echo "Contenu quelconque11" > fichier1.md
```

Exécuter git status et copier/coller la sortie

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   fichier1.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        fichier2.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Ajouter fichier1.md et fichier2.md à la zone de Staging

git add fichier1.md fichier2.md

Créer un commit "Ajout de fichier2.md et modification de fichier1.md"

git commit -m "Ajout de fichier2.md et modification de fichier1.md"

Copier/coller l'ID des deux premiers commits :

ID commit 1 : `ac1eac239da066f8801723fa34b2afbafc0fe911`

ID commit 2 : `abf660144a590f66cd0ed335e809f331234f652f`

Que signifie qu'un fichier est "tracked" ou "untracked" ?

Un fichier tracked sera dans la zone de Staping, git analysera tout changement sur ce fichier à l'inverse git ne s'occupera pas de celui-ci

Pourquoi doit-on passer les fichiers par la zone de Staging (l'index) avant de les committer ?

Pour importer les nouveaux changements et pour que git puisse comparer correctement

Créer une branche fonctionnalite1

git branch fonctionnalite1

Lister les branches

git branch

Se déplacer sur la branche fonctionnalite1

git checkout fonctionnalite1

Lister les branches

git branch

Que représente l'étoile à côté des noms des branches ?

Indique la branche sur laquelle on se situe

Créer un nouveau fichier fichier3.md

touch fichier3.md

Modifier le fichier fichier2.md

echo "Contenu quelconque22" > fichier2.md

Comment utiliser VS Code pour qu'il nous montre les différences entre l'ancienne version de fichier2.md et la version courante que l'on vient d'éditer ?

Dans l'icône "Source Control" (Ctrl+Shift+G), puis cliquer sur le fichier souhaité

Committer ces deux modifications : "Fonctionnalité 1 - première phase"

git commit -m "Fonctionnalité 1 - première phase"

git add .

git commit -m "Fonctionnalité 1 - première phase"

Créer un nouveau fichier fichier4.md

touch fichier4.md

Modifier de nouveau le fichier fichier2.md

echo "Contenu quelconque222" > fichier2.md

Committer ces deux modifications : "Fonctionnalité 1 - terminée"

git add .

git commit -m "Fonctionnalité 1 – terminée"

Afficher la liste des fichiers du répertoire

ls

Se déplacer sur la branche master

git checkout master

Afficher la liste des fichiers du répertoire

ls

Pourquoi les deux sorties sont-elles différentes ? Les fichiers ont-ils disparus ?

Car lorsqu'on a créé le fichier3 et 4 nous étions sur la branche fonctionnalite1, donc master ne prend pas les fonctionnalités de celle-ci

Créer une nouvelle branche fonctionnalite2

git branch fonctionnalite2

Cette branche ne va pas avoir toutes les données incluses dans fonctionnalite1. Pourquoi ?

Car nous l'avons créée à partir de master qui ne contient pas elle-même ces fonctionnalités

Qu'aurait-il fallu faire si on avait souhaité démarrer la branche fonctionnalite2 en intégrant les modifications récentes de fonctionnalite1 ?

Créer la branche fonctionnalite2 en étant sur la branche et sur le dernier commit de fonctionnalite1

Se déplacer sur la nouvelle branche fonctionnalite2

git checkout fonctionnalite2

Créer un nouveau fichier fichier5.md

touch fichier5.md

Faire un commit intégrant cette ajout : "Ajout fichier5.md"

git add fichier5.md

git commit -m "Ajout fichier5.md"

Entrer la commande git log --oneline --decorate --graph --all pour visualiser, sur le terminal, le graphe des commits sur toutes les branches

Noter la « déviation » entre les deux branches, à partir de la branche master

La branche master se divise en 2 branches à partir du 1er commit "Ajout fichier2.md et modif fichier1.md"

Installer l'extension VS Code Git Graph et visualiser le graphe actuel des commits à l'aide de cette extension

Sur cette représentation, que représente les points ?

Un point = un commit

Comment voit-on sur quelle branche on est actuellement ?

Grâce à la petite puce juste avant le nom de la branche

On considère que la branche originale (master ou main) est la branche d'intégration, c'est-à-dire celle qui va contenir l'historique de toutes les modifications développées au fur et à mesure dans les branches annexes

On va maintenant fusionner la branche fonctionnalite1, qui est terminée, avec la branche d'intégration

Se déplacer sur la branche master

Noter le changement dans l'onglet Git Graph

La puce est devant master

Fusionner avec la branche fonctionnalite1

git merge fonctionnalite1

Noter le changement dans l'onglet Git Graph. Que signifie la mention Fast-forward indiquée par la sortie de la commande ?

Les noms des 2 branches sont côte à côte

On veut maintenant fusionner fonctionnalite2 dans la branche d'intégration (master)

Effectuer cette fusion

git merge fonctionnalite2

Noter le changement dans l'onglet Git Graph. Que signifie la mention Merge made by the ... strategy indiquée par la sortie de la commande ?

Quelle est la différence fondamentale avec la fusion précédente ?

Créer une nouvelle branche fonctionnalite3, se déplacer dessus, et modifier le fichier fichier1.md en y ajoutant une ligne de texte. Committer : "Modification fichier1 pour fonctionnalité 3"

Comment utiliser Git Graph pour qu'il nous montre les différences entre l'ancienne version de fichier1.md et la version courante que l'on vient de committer ?

Repartir sur master, et modifier fichier1.md en y ajoutant aussi une ligne (différente de celle qu'on a ajoutée sur l'autre branche)

Ajouter à l'index et faire un commit

Tenter de fusionner la branche fonctionnalite3 avec master

Que se passe-t-il et pourquoi ?

Lancer un git status

Que doit-on faire si on veut annuler la fusion en cours ?

On veut résoudre le conflit. Plusieurs possibilités :

Conserver uniquement les modifications faites dans fonctionnalite3

Conserver uniquement les modifications faites dans master

Conserver les deux modifications

Supprimer les deux modifications ou remanier sensiblement le fichier pour les intégrer correctement

Git nous laisse totalement la main et ne va pas essayer d'imposer l'un de ces choix pour nous, ni nous assister dans l'application automatique de l'un d'entre eux : il faut examiner le(s) fichier(s) en conflit et éditer nous-mêmes

Ouvrir le fichier en question sous VS Code

La chaîne <<<<<<<<<< marque le début du conflit

La chaîne >>>>>>>>>> marque la fin du conflit

La chaîne ========== sépare les deux versions

Éditer le fichier pour faire en sorte d'intégrer les deux modifications ; à la fin de l'édition :

Sauvegarder

Il ne doit plus y avoir de marques quelconques en dehors des ajouts fonctionnels originaux, c'est-à-dire pas de <<<<<<<<<<, ni de mentions de nom de branche, etc. : vous rendez le fichier tel qu'il doit apparaître dans le commit de fusion, avec les conflits résolus manuellement

Ajouter les modification à l'index et committer

NB : parfois, plusieurs fichiers sont en conflit ; le processus est identique, il faut juste résoudre les conflits sur tous les fichiers

NB : les conflits de fusion sont fréquents lorsqu'on travaille en collaboration (plusieurs personnes vont travailler sur le même fichier pour remplir deux fonctionnalités différentes)

Les branches créées n'ont plus de raison d'exister

Elles avaient pour but de créer une déviation afin de travailler sur des fonctionnalités individuelles

On va vouloir nettoyer le dépôt en les supprimant

Cela ne va bien sûr pas supprimer tous les commits qui y sont associés

Attention cependant d'éviter en général de supprimer une branche qui n'a pas encore été intégrée à la branche d'intégration, sauf on souhaite vraiment abandonner le développement de cette branche

Ne pas réutiliser une branche qui a déjà été intégrée pour démarrer une nouvelle piste : toujours utiliser une nouvelle branche

Nouvelle tâche ? => nouvelle branche à partir d'un commit de la branche d'intégration (en général le plus récent)

Tâche terminée ? => fusion dans la branche d'intégration et suppression de la branche

Supprimer les trois branches fonctionnalitex (attention : on ne peut pas supprimer une branche sur laquelle on est)