Travail à rendre TP AP4A

Réalisé par **: NANMEGNI NGASSAM Gilles Pavel**

Rendu N°1

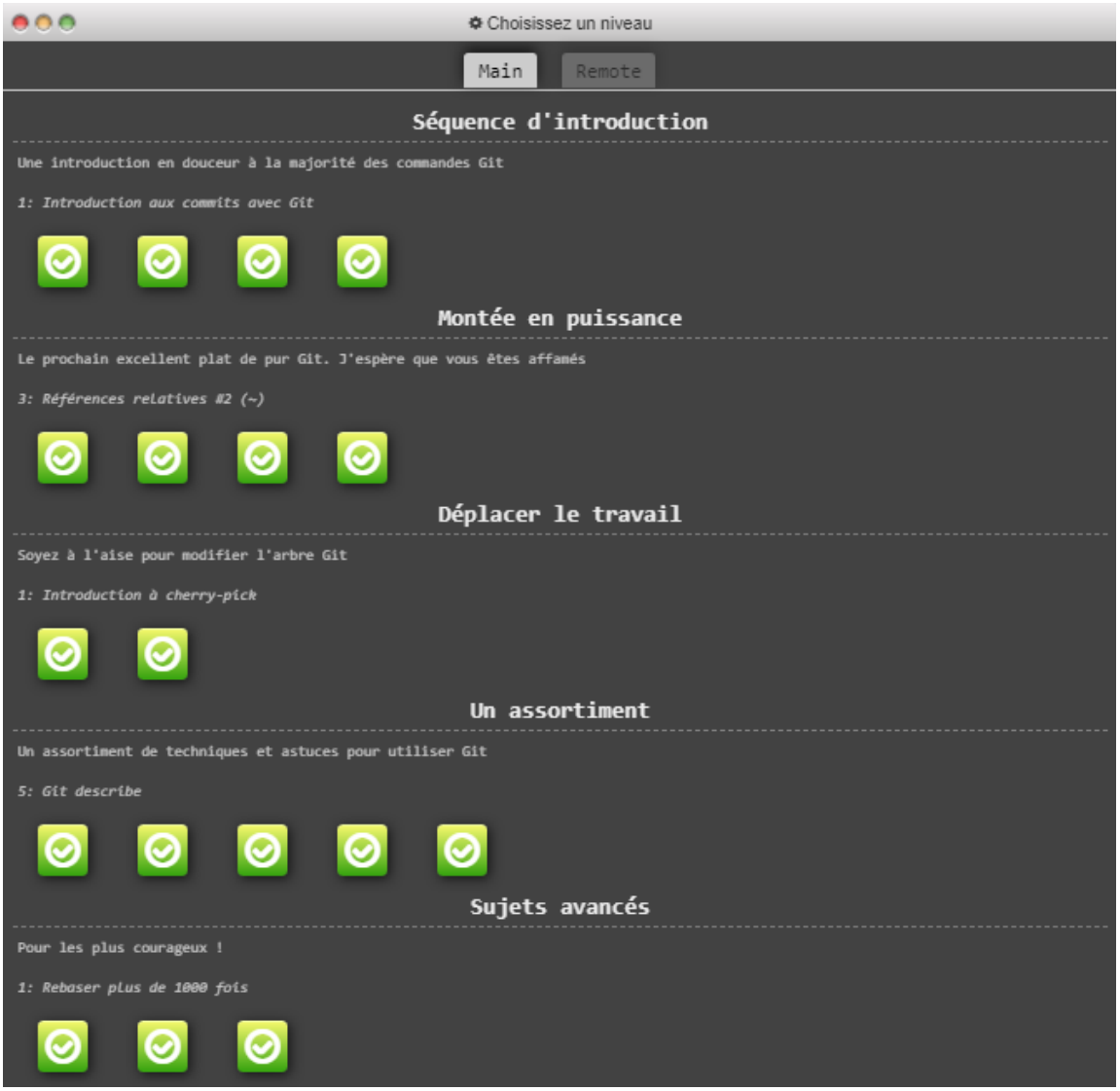
Au cours de notre première séance de TP, en ce qui concerne l'UV AP4A (Programmation Orientée Objet) nous avons été initiés à :

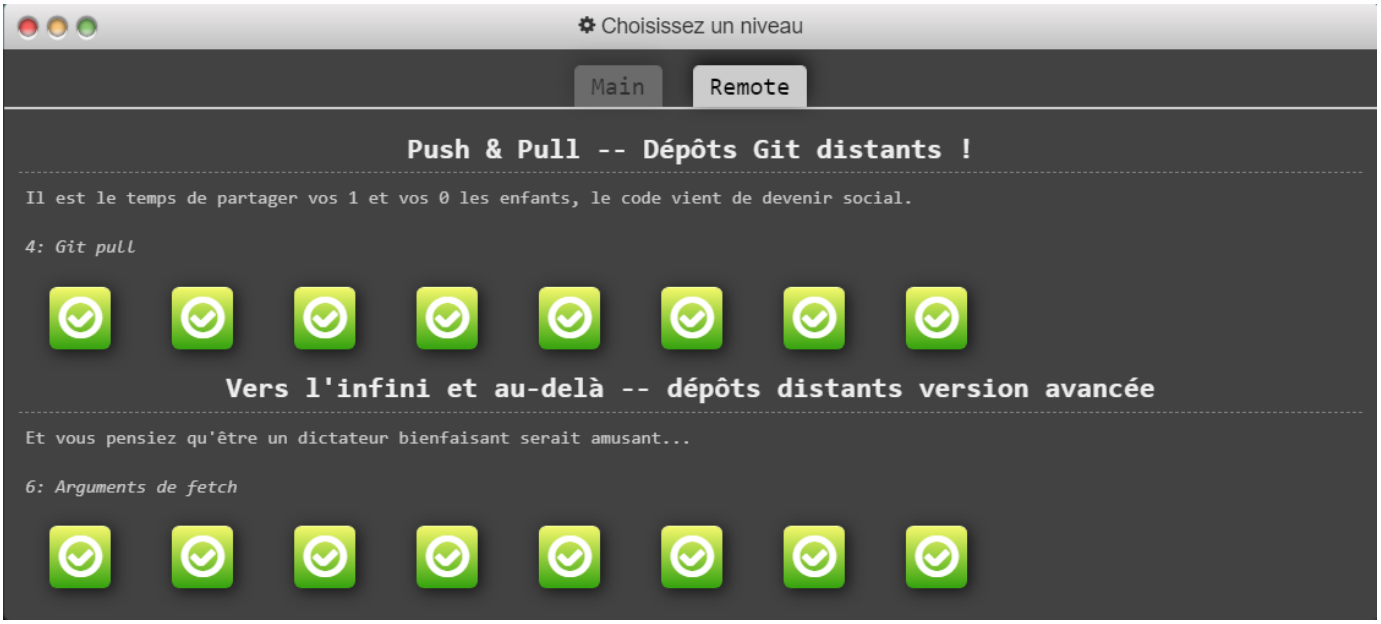
- La mise en place de l'environnement de développement en C++, notamment sous Windows : Avec l'installation de l'IDE (Vs Code ou Eclipse), le téléchargement des thèmes de développement C++, la mise en place du compilateur g++ via MinGW ;
- La création d'un document C++, à sa compilation (build) et à son exécution dans la console ;
- L'outil de versionnage, de sauvegarde et de travail collaboratif Git ;

A l'issu de cette séance de travail, il nous a été remis deux tâches :

- Finir et appréhender les notions expliquées dans le tutoriel Git dont l'adresse suit (https://learngitbranching.js.org/?locale=fr_FR) ;

  Ceci a été fait et les captures suivantes peuvent attester de cela.





- Fournir un récapitulatif sur les commandes les plus utilisés ;
  Ce qui est fait dans la suite du document

Récapitulatif de quelques commandes Git

Git est un système de contrôle de version Open Source. Il peut être utilisé pour l'hébergement de votre code, le contrôle de version sur un projet et également le travail collaboratif (En équipe). Afin de pouvoir l'utiliser, vous devez au préalable l'avoir installé sur votre machine, ceci vous mettra directement en place sur votre machine un serveur local. Par la suite vous devez vous créer un compte sur GitHub (Serveur distant). Enfin, vous devez relier votre machine (serveur local) avec votre compte GitHub (Serveur Distant). En ce qui concerne la suite, les commandes (Instructions) suivantes vous seront nécessaires. Pour le faire, ouvrez votre terminal (PowerShell, Git Bash…) :

- **Git --version : Vérification de la version de Git installé sur votre machine**

De manière pratique, on utilise cette commande à la fin de l'installation de Git pour vérifier si tout a fonctionné correctement. Également en cas d'incompatibilité avec d'autres programmes, on peut vérifier la version de Git que l'on a afin d'envisager une possible mise à jour.

- **Git init : Création d'un dépôt Local ou Initialisation d'un dépôt existant**

Cette commande permet d'indiquer à Git que le dossier à l'intérieur duquel nous avons tapé la commande *git init* est un dépôt local. Et que ce dossier pourra contenir des fichiers qui seront plus tard placés sur le serveur local. Également, git créera dans ce dossier un dossier caché nommé .git qui contiendra le fichier de configuration propre à ce dépôt (config.txt) ainsi que d'autres fichiers et dossiers.

- **Git config : Spécification des paramètres de configuration de git**

Cette commande nous permet de définir nos préférences en ce qui concerne l'utilisation de git. Utilisée avec l'option --*global*, il permet de définir des paramètres qui seront généraux à l'utilisation de git. On a par exemple :

  - Le nom d'utilisateur (votre pseudo GitHub) : *git config global user.name ''pseudo GitHub''*
  - Votre adresse E-mail (Celle renseignée sur GitHub) : *git config global user.email adresse_email*
  - …

- **Git clone : Copie des fichiers d'un dépôt distant sur notre dépôt local**

Cette commande permet de copier les fichiers et les branches contenus dans un dépôt distant sur notre dépôt local. La syntaxe est la suivante : *git clone <URL du dépôt à recopier>.*

- **Git add : Ajout de fichiers sur le dépôt local git**

En exécutant cette commande, on choisit de mettre le ou les fichiers sélectionnés dans une boite qui sera plus tard étiqueté et placé sur le serveur local et bien après envoyé ou pas sur le serveur distant. La syntaxe *git add <nom_du_fichier.extansion>* mettra le fichier nommé nom_du_fichier.extansion dans la boîte . On peut également écrire *git add ** pour spécifier qu'on veut mettre dans la boite tous les fichiers contenus dans notre dossier.

- **Git status : Statut du dépôt local git**

La commande *git status* permet de vérifier le contenu de la boîte et ainsi de connaitre si un fichier a déjà été inclus ou non. Son exécution retourne la branche sur laquelle on se trouve, le nom des fichiers qui ont déjà été placés dans la boîte et pour finir le nom de ceux qui sont hors de la boite.

Pour les fichiers qui sont déjà dans la boîte, il précise via le mot clé *modified* si ce dernier a été modifié après l'avoir placé dans la boîte.

- **Git commit : Etiquetage de la boîte et dépôt sur le serveur local**

Cette commande nous permet de placer la boîte contenant nos fichiers sur le serveur local. Mais afin de pouvoir plus tard retrouver cette boîte facilement, on va, avant de la placer sur le serveur local, l'étiqueter en ajoutant sur la commande *git commit*, l'option *-m*. On aura donc : *git commit -m <message de l'étiquette collé sur la boîte>*.

- **Git push : Envoi du contenu du serveur local vers le serveur distant**

Une fois que les différentes boîtes ont été placés sur notre serveur local, on utilise la commande *git push* pour l'envoyer vers le dépôt (serveur) distant, afin que tous les membres de l'équipe avec laquelle nous travaillons puisse le voir et y apporter des modifications.

- **Git branch : Listing des différentes branches et ajout de nouvelles branches**

Un dépôt GitHub est constitué de plusieurs branches (ce sont des dérivations du projet principal crées afin de développer de nouvelles fonctionnalités tout en limitant les bugs sur le projet principal). La branche principale s'appelle main ou master. Pour ajouter une nouvelle branche, la commande est la suivante : *git branch <nom_nouvelle_branche>*. Pour lister les branches existantes et éventuellement connaitre la branche sur laquelle on se trouve, la syntaxe est : *git branch*. La branche en cours d'utilisation sera marquée d'un astérisque (*).

- **Git checkout : Positionnement sur une branche**

Cette commande permet de se déplacer entre les différentes branches de notre dépôt. La syntaxe est la suivante : *git checkout <nom de la branche sur laquelle on veut se rendre>*.

- **Git merge : Fusion de branches**

La commande git merge permet de mélanger le contenu de deux branches (copier le contenu d'une branche dans l'autre). La syntaxe est la suivante : *git merge < nom de la branche dont on veut copier le contenu>*

- **Git diff : Fonction de différenciation sur des sources de données git**

La syntaxe de cette commande est la suivante : *git diff <identifiant_commit1> <identifiant_commit2>*. Cette commande permet de déterminer quelles sont les différences entres les fichiers ou lignes de code du commit 1 et du commit 2.

- **Git blame : Fonction de détection de l'auteur d'une écriture/modification**

La commande Git blame permet de retourner une sortie des différentes lignes d'un fichier tout en indiquant qui a modifié/écris cette ligne de code et quand est ce qu'il l'a fait. La syntaxe est *git blame <nom_fichier>*.

Remarque : Il existe plein d'autres options et fonctionnalités sous git on peut les retrouver directement sur la documentation (https://git-scm.com/book/fr/v2/Personnalisation-de-Git-Configuration-de-Git)