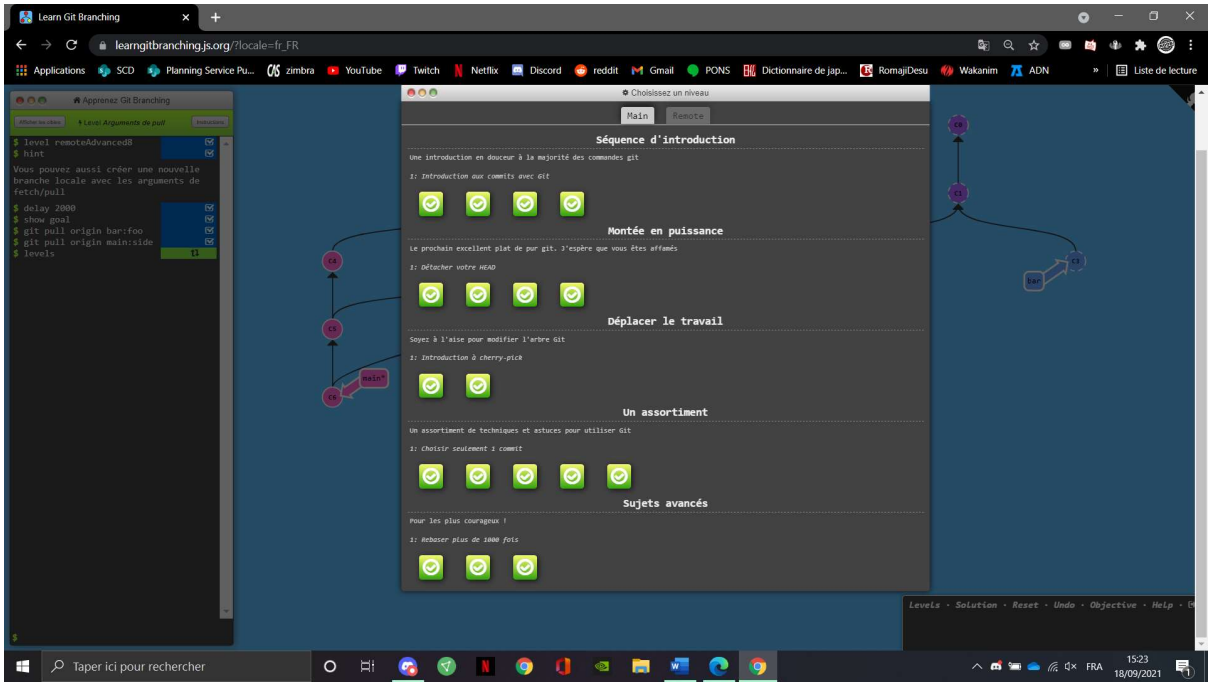

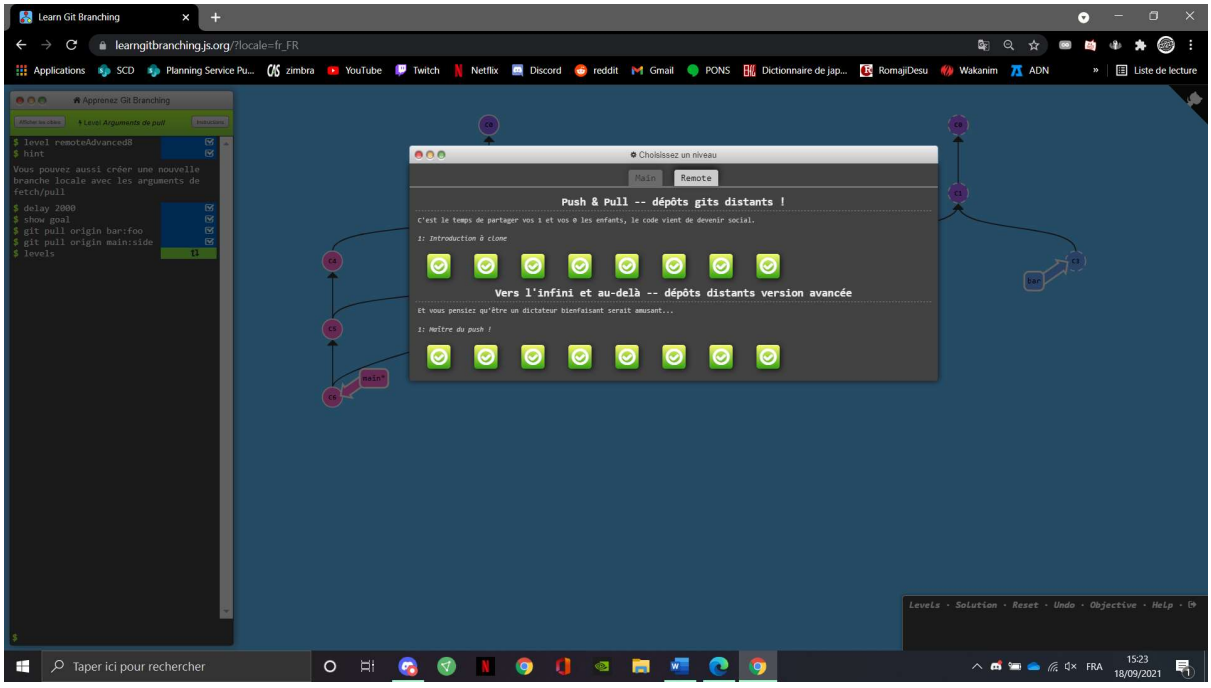

<u>Commandes git :</u>

- Git config : permet à l'aide de différents arguments de modifier la configuration de git (en modifiant les fichiers de configuration) : on utilise souvent l'argument `--global` pour modifier le fichier `~/.gitconfig` spécifique à l'utilisateur plutôt que `--system` qui affecte le fichier `/etc/gitconfig` qui contient des valeurs pour tous les utilisateurs du système et tous leurs dépôts. Ensuite vient la modification à faire : `core.editor` pour changer l'éditeur de texte par défaut (`git config --global core.editor <nom de l'éditeur>`), `user.name` pour changer notre nom d'utilisateur git (`git config --global user.name <nouveau nom>`),… On peut décrire cette

commande comme la commande de personnalisation de git, allant même jusqu'à permettre de créer des templates par défaut pour les commits.

- Git init : Permet de créer un nouveau dossier git. Cela peut permettre de créer un nouveau projet ou être réalisé dans un dossier contenant déjà un projet commencé en local pour créer un dépôt git de ce projet. Tous les fichiers nécessaires à git sont ainsi créés et une branche initiale sans commits est créée.

- Git status : cette commande donne des informations sans modifier de commits ou faire de changement sur le dépôt local et peut donc être utilisée sans risque. Son objectif principal est de donner des informations. Ces informations dépendent de la situation. La commande donne d'abord le commit ou la branche sur lequel pointe actuellement la HEAD du dépôt. Si des modifications ont été faites et n'ont pas encore été commits dans le dépôt actif, les fichiers modifiés sont donnés, en faisant une distinction entre les modifications ajoutée via git add mais pas encore commit et les modifications qui n'ont pas encore été ajoutées. La branche en cours et son état par rapport à son remote track, c'est-à-dire si elle est en avance ou en retard en termes de commits, sont également décrits. Enfin, dans le cas d'un conflit de merge, les fichiers de conflits seront indiqués.

- Git add : permet d'ajouter des fichiers ou des modifications de fichiers présents dans le dossier courant au tracking de git (dans la « zone de staging »). Cette commande informe git que l'on veut inclure des mises à jour dans le prochain commit et doit donc être utilisée en amont d'un git commit. Un nom de fichier peut être précisé (git add [nom de fichier]) pour n'ajouter que les modifications de certains fichiers au prochain commit. On peut même aller jusqu'à n'ajouter que certains types de fichiers (git add *.cpp pour n'ajouter que les fichiers source C++ et leurs modifications) ou directement tous les fichiers non ignorés d'un dossier (git add .). Une suppression de fichier peut aussi être ajoutée pour supprimer le fichier concerné du remote track.

- Git push : permet de publier les changements locaux, notamment tous les nouveaux commits, vers le dépôt distant. Il s'agit de la commande qui permet d'envoyer son travail vers le dépôt central, de « publier » son travail. Grâce aux différents arguments qui peuvent être rajoutés, il est possible de définir exactement quelle(s) branche(s) et quel(s) commit(s) doivent être envoyés dans quelle(s) branche(s) du dépôt distant.

- Git merge : permet de créer un commit spécial avec « 2 parents », c'est-à-dire d'inclure des modifications venant de deux branches différentes dans un seul commit. Cette commande est donc principalement utilisée pour fusionner deux branches ensembles (même si cette application fréquente n'est pas sa seule utilité comme pour la plupart des commandes git).

- Git diff : permet de comparer des ensembles de données pour voir les changements entre eux. On peut ainsi voir les modifications apportées entre deux commits, ou entre deux branches, ou entre une branche et un commit, … Les différentes options de la commande et les paths données définissent la comparaison à effectuer.

- Git blame : permet d'afficher les informations relatives à l'évolution de fichiers au fil des commits. Le contenu d'un fichier est examiné ligne par ligne afin de savoir à quelle date chaque ligne a été modifiée pour la dernière fois. L'id du dernier commit, son auteur, son horodatage, le numéro de ligne et le contenu de la ligne s'affichent ainsi pour chaque ligne du fichier avec l'utilisation de git blame <nom de fichier>. De nombreuses options permettent de modifier le format de sortie de la commande ou d'ignorer des changements minimes tels que l'ajout d'espaces pour le formatage du code.