Git exercices

1 - Apprendre les bases des commandes GIT (pas si simple que ça)

2 - Place à la suite, créez un nouveau repo que vous allez appeler `git-exercices` sur votre github

```
git clone `https://github.com/vous/git-exercices`
```

- Vous allez créer 3 dossiers, `html`, `css` et `js`
- Vous allez ensuite créer un fichier `index.html` dans le dossier `html`
- Vous allez ensuite créer un fichier `index.js` dans le dossier `js`
- Vous allez ensuite créer un fichier `style.css` dans le dossier `css`

Dans le fichier `index.html`

```
</<!DOCTYPE html>

<html class="no-js">
    <head>
```

```
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>Git exercices</title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" href="../css/style.css">
    </head>
    <body>
        <script src="../js/index.js" async defer></script>
    </body>
</html>
```

Dans le fichier `index.js`

```
console.log("js is ready");
```

```
body {
  background: blue;
}
```

- Faites un `git status`

```
On branch master

No commits yet

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

```
        .DS_Store
        css/
        html/
        js/

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

- Vous allez ensuite `commit` ces 3 dossiers : À la racine de votre projet `git add *`
- Si vous effectuez à nouveau un git status

```
On branch master

No commits yet

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

        new file:   css/style.css
        new file:   html/index.html
        new file:   js/index.js

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

        .DS_Store
```

- Vous pouvez maintenant correctement envoyer votre travail `git commit -m "Initial commit"`

```
[master (root-commit) 00460bd] Initial commit
 3 files changed, 19 insertions(+)
 create mode 100644 css/style.css
 create mode 100644 html/index.html
 create mode 100644 js/index.js
```

- Si vous faites à nouveau un `git status`
- Vous pouvez faire un `git push` simple
- Nous allons maintenant créer une nouvelle branche que nous appellerons `dev`
- `git branch dev`
- Lister toutes les branches `git branch`

```
  dev
* master
```

LA PETITE ÉTOILE INDIQUE NOTRE HEAD, LÀ OÙ NOUS NOUS SITUONS.

ATTENTION, NOUS VENONS DE CRÉER UNE NOUVELLE BRANCHE EN LOCAL (SUR NOTRE ORDINATEUR) ET NON EN REMOTE (SUR LE REPO GIT)

- Changeons de branche pour effectuer un changement sur notre nouvelle branche `dev` : `git checkout dev`

```
Switched to branch 'dev'
```

- un `git status` vous dira où est-ce que vous êtes

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

        .DS_Store

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

- Nous allons maintenant modifier nos fichiers

```
/* ./css/style.css */
body {
  background: red;
}
```

- Refaisons un `git status` pour voir ce qui a changé.

```
On branch dev
Changes not staged for commit:
```

```
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   css/style.css

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

        .DS_Store

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

Le fichier css/style.css vient de changer. Nous allons commit ce changement

- `git add css/style.css`
- `git commit -m "[CSS] Changed the body background color"`

Une bonne pratique est de **toujours** renseigner ce que vous faites dans le message du commit, et plus particulièrement, quel domaine vous avez changé. Le changement que nous avons effectué concerne le CSS, d'où le [CSS] avant l'explication de mon message

-Si vous faites un simple `git push`, git vous enverra un message d'erreur.

```
fatal: The current branch dev has no upstream branch.
To push the current branch and set the remote as upstream, use
```

```
    git push --set-upstream origin dev
```

Je le répète, nous venons de créer une nouvelle branche en LOCAL et non sur notre REPO. pour ce faire nous pouvons suivre les indications de git.

-`git push origin dev`

Si vous retournez sur votre repo Github. Vous pouvez voir cette nouvelle branche.

- Retournons sur notre branche master `git checkout master` Si vous regardez à nouveau votre fichier `css/style.css` le background est de nouveau en bleu.

```
/* css/style.css */
body {
  background: blue;
}
```

Nous allons maintenant effectuer notre premier merge. On va fusionner la branche dev dans la branche master :)

- `git checkout master`
- `git pull`
- `git merge dev`

```
Updating 00460bd..6f281e0
Fast-forward
 css/style.css | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

3 - Passons à un merge conflict

- Créons un fichier readme.md `touch readme.md`

```
//Instructions goes here...
```

-Si vous faites à nouveau un `git status`

```
On branch master
Your branch is up to date with 'origin/master'.

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

        .DS_Store
        readme.md

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

Jusque là tout va bien :)

- Retournons sur notre branche dev `git checkout dev`
- Vous pouvez créer une nouvelle branche et naviguez dessus directement comme ceci : `git checkout -b branchName`
- Ajoutons & commitons notre fichier readme.md
- `git add readme.md`
- `git commit -m "Added a readme.md file"`
- Regardez maintenant les différences entre la branche master et dev en tapant la commande suivante `git diff master dev`

```
diff --git a/readme.md b/readme.md
new file mode 100644
index 0000000..463f8d5
--- /dev/null
+++ b/readme.md
@@ -0,0 +1 @@
+ //Instructions goes here...
```

Step : Créons deux nouvelles branches

Créez deux branches `branch-a` & `branch-b` en tapant :

- `git branch branch-a`
- `git branch branch-b`

Vous pouvez comparer ces deux branches en tapant : `git diff branch-a branch-b`

Note : Cela ne retourne rien car ces deux branches sont identiques.

Pour confirmer quelles sont identiques, vous pouvez également afficher les logs de chacune des nouvelles branches.

- `git log branch-a`
- `git log branch-b`

Les messages du commit et les hash du commit devraient être identiques.

Step : Faisons un changement sur la branche

`branch-a`

-`git checkout branch-a`

- `git branch` pour vous assurez d'être sur la bonne branche

  Ouvrez `readme.md` et ajouterz la ligne suivante `Branch-a instructions` :

  ```
  //Instructions goes here ...
  Branch-a instructions...
  ```

  Ajoutez ce changement pour git dans la branche actuelle (branch-a)

- `git add readme.md`
- `git commit -m "[Readme] Added branch-a instructions"`

Step

Changez de branche et aller sur votre seconde nouvelle branche `branch-b`

- `git checkout branch-b`

Ouvrez `readme.md` a ajoutez la ligne suivante `Branch-b instructions` :

```
//Instructions goes here ...
Branch-b instructions...
```

Ajoutez ce changement pour git dans la branche actuelle (branch-b)

- `git add readme.md`
- `git commit -m "[Readme] Added branch-b instructions"`

Step : Comparons les différences

Comparons `branch-a` et `branch-b` en tapant :

- `git diff branch-a branch-b`

Step : Comparons l'historique des deux branches

- `git log branch-a`
- `git log branch-b`

Les deux branches ont en commun le commit concernant l'ajout du readme.md file `Added a readme.md file`

Step : Nous allons créer une fausse branche `master`

Nous devrions faire ça sur la branche master, mais pour l'exercice nous allons simuler la branche master par une autre branche.

Changer de branche :

- `git checkout dev`

Créons une nouvelle branche à partir de notre branche dev.

-`git checkout -b branch-master`

Step : Mergeons le travail dans notre fausse branche master

`git merge branch-a`

Maintenant la branche master contiendra le travail de la branche `branch-a`

Step : Confirmons que la fausse branche master contient le travail de la branche `branch-a`

- `git log branch-a`

- `git log branch-master`

  Vous pouvez aussi vérifier quelles sont identiques de cette manière :

- `git diff branch-a branch-master`

  Si ça ne retourne rien, c'est que les deux branches sont identiques.

Step: Merge le conflit :)

`git merge branch-b`

Git vous prévient :

```
Auto-merging planets.md
CONFLICT (content): Merge conflict in readme.md
Automatic merge failed; fix conflicts and then commit the result.
```

Pour avoir plus d'informations :

- `git status`

```
On branch branch-master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   readme.md

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

Step : Utilisons un éditeur de texte pour merger le conflit.

```
//Instructions goes here ...
<<<<<<< HEAD
Branch-a instructions ...
=======
Branch-b instructions ...
>>>>>>> branch-b
```

Pour corriger le conflit : Utilisez votre IDE préféré et enlevez les chevrons de git qui indique le conflit.

```
//Instructions goes here...
Branch-a instructions...
Branch-b instructions...
```

Sauvegardez et ajoutez les changements à GIT

- `git add readme.md`

Vérifiez que tout est bon :

- `git status`

```
On branch branch-master
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:

    modified:   readme.md
```

Envoyez les modifications :

`git commit -m "making peace between branch-a and branch-b`

Step : Regardons l'historique

- `git log`

  Vous pouvez également regardé l'historique de toutes les branches entre-elle

- `git diff dev master`
- `git diff branch-master dev`
- `git diff master branch-master`

Poussons toutes nos branche sur notre github

- `git push -all`

Step : Mergeons notre travail dans la vraie branche master

- `git checkout master`

- `git merge branch-master`

Step : Regardeons l'historique de notre master qui devrait avoir tout :)

- `git log master`

  Vous pouvez également revenir en arrière pour examiner en détail le code d'un commit précedent en référençant sont HASH

- `git checkout ca41886d`

  Pour enfin revenir

- `git checkout master`

Vous pouvez supprimer des branches en local et en

remote

Pour supprimer en local :

- `git branch -d dev`
- `git branch -d branch-a`
- `git branch -d branch-b`
- `git branch -d branch-master`

Puis pour supprimer en remote :

- `git push --all`

Interface graphique

- Essayer maintenant avec une interface graphique pour comprendre davantage git.