I - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

https://git-scm.com/	site officiel de la technologie git
https://openclassrooms.com/courses/gerez-vos-codes-source-avec-git	tutoriel sur GIT

2. TP

2.1. Installation de git en mode texte

Installer git en fonction du type d'ordinateur

Exemple pour linux debian ou ubuntu :

```
$ sudo apt update
$ sudo apt upgrade -y
$ sudo apt install git
$ git --version
```

Exemple pour windows :

https://git-scm.com/download *site officiel pour télécharger l'installeur de git*

https://github.com/git-for-windows/git/releases/download/v2.41.0.windows.1/Git-2.41.0-64-bit.exe

Lancer l'installeur avec choix par défauts

Vérification:

git --version
git version 2.41.0.windows.1

2.2. Configuration locale fondamentale de git

git config --global user.name "*PrenomNom*"
git config --global user.email "*prenom.nom@ici_ou_la.fr*"
git config –list

2.3. Eventuelle installation de compléments graphiques :

Liste des clients "git" graphiques: https://git-scm.com/downloads/guis

TortoiseGit *(gratuit , sous windows)*

https://tortoisegit.org/download/

https://download.tortoisegit.org/tgit/2.14.0.0/TortoiseGit-2.14.0.1-64bit.msi

GitKraken

https://www.gitkraken.com/download/windows64 *(produit payant , version gratuite limitée à 7jours à priori)*

+ ***plugins "git" des principaux IDE*** *(eclipse, intelliJ , Visual studio code , ...)*

2.4. Git élémentaire en mode local

Se créer un répertoire de tp (ex: ***c:\tp\tp-git*** *)*

Créer les sous répertoires c:\tp\tp-git***local-git-repositories-devA**

et c:\tp\tp-git***local-git-repositories-devA\p1**

Au sein du répertoire p1 , créer (par exemple via notepad++) les fichiers simples suivants :

p1\f1.txt

```
ligne1
ligne2
```

p1\f2.txt

```
abc
def
```

p1\inutile.bad

```
fichier pas toujours utile et potentiellement de grande taille
```

Création d'un référentiel local git au sein du projet p1 :

se placer dans ***local-git-repositories-devA\p1***

```
git init
```

→ Visualiser via l'explorateur de fichiers , le **nouveau sous répertoire caché** "**.git**" .

Générer (dans p1) le souvent indispensable fichier **.gitignore**

```
echo *.bad > .gitignore
```

Visualiser le contenu de ***.gitignore*** via notepad++ et ajouter y les lignes suivantes :

.gitignore

```
*.bad
*.jar
/node_modules
/bin
/target
```

Lancer les commandes suivantes et visualiser bien les effets de chacune d'elles

```
git status
git add *.txt
git add .gitignore
git status
git commit -m "initial"
git status
git log
```

En exercice :

- **ajouter** le fichier ***f3.txt*** contenant "*f3 c'est 3 fois rien*"
- **modifier** le contenu de ***f1.txt*** avec quelques majuscules
- lancer git status , git add … , git commit -a -m "…" de façon à enregistrer tous les changements
- réafficher l'historique via **git log --stat**
- **ajouter** le fichier ***f4.txt*** contenant "*f4 est un nouveau fichier*"
- **modifier** le contenu de ***f1.txt*** avec une ligne en plus
- lancer git status , git add … , git commit -a -m "…" de façon à enregistrer tous les changements
- **réafficher l'historique via git log**
  et copier dans le presse papier l'**id de l'avant dernier commit** associé à "*f3 et modif f1*"
  (exemple : ***775c58b7a5f39c2d2d61d1e0bcc7bcbe3f412905***)

Retour en arrière (vers ancienne version sauvegardée) :

```
git checkout 775c58b7a5f39c2d2d61d1e0bcc7bcbe3f412905
```

→ visualiser le nouvel état de p1 (f4.txt a disparu et f1.txt ne comporte plus le 3ème ligne ajoutée)

Revenir en tête de branche principale :

```
git checkout main
git log
```

→ visualiser le nouvel état de p1 (f4.txt est revenu et f1.txt comporte la 3ème ligne ajoutée)

Associer le tag *v1* à l'avant dernier commit :

```
git tag v1 775c58b7a5f39c2d2d61d1e0bcc7bcbe3f412905
git tag -l
git log
```

git checkout tags/v1

→ *visualiser l'absence de f4.txt*

git checkout main

→ *visualiser le retour de f4.txt*

2.5. Gestion élémentaire des branches de git

git branch

→*par défaut une seule branche master ou* **main**

création d'une nouvelle branche b1 :

```
git branch b1
```

Se positionner sur branche b1

git checkout b1

git branch

main

** b1 (avec * devant branche courante/sélectionnée).*

En exercice :

- ajout de f5.txt et nouveau commit sur branche b1
- ajout de f6.txt et encore nouveau commit sur branche b1
- git log

- se positionner sur branche main (via git checkout)
  → visualiser absence de f5.txt et f6.txt
- se positionner sur branche b1 (via git checkout)
  → visualiser retour de f5.txt et f6.txt

Fusion élémentaire de branches (fast forward) :

Modifications effectuées que sur b1 , aucune modif effectuées sur main → fast forward

git checkout main
git merge b1
git log
git branch -d b1 ← *suppression ancienne branche b1*
git log

2.6. Merge de fusion sans conflit

En exercice :

- partir du sommet de la branche main
- créer un nouvelle branche temporaire b2
- se placer sur la branche b2
- ajouter le fichier fa.txt
- ajouter *** en fin de la première ligne de f1.txt
- commiter tous ces changements
- se placer sur la branche main
- ajouter fb.txt
- ajouter dernièreNouvelleLigne### en fin de f1.txt (sans changer les premières lignes)
- commiter tous ces changements
- rester sur la branche main
- déclencher une fusion avec la branche b2
- visualiser tous les effets (git log et nouveau commit de fusion, état des fichiers fa.txt , fb.txt , f1.txt)
- supprimer l'ancienne branche temporaire b2

2.7. Merge git avec résolution de conflit

- partir du sommet de la branche main
- créer un nouvelle branche temporaire b3

- se placer sur la branche b3
- modifier la première ligne de f2.txt (abc transformé en ABC)
- commiter tous ces changements
- se placer sur la branche main
- modifier différement la première ligne de f2.txt (abc transformé en a_b_c)
- commiter tous ces changements
- rester sur la branche main
- déclencher une fusion avec la branche b3
- visualiser tous le conflit (message de la commande, état du fichier f2.txt)

  Auto-merging f2.txt

  CONFLICT (content): Merge conflict in f2.txt

  Automatic merge failed; fix conflicts and then commit the result.
- **git status**
- On pourrait éventuellement annuler la fusion via *git merge --abort* (si trop de conflits à résoudre). On va préférer résoudre le conflit pour finaliser la fusion.
- visualiser l'état temporaire de ***f2.txt***

```
<<<<<<< HEAD
a_b_c
=======
ABC
>>>>>>> b3
def
```

- ne garder dans ***f2.txt*** qu'une des versions proposées a_b_c ou ABC ou un mixte des 2.

```
A_B_C
def
```

- indiquer la fin de la résolution du conflit sur f2.txt via la commande
  git add f2.txt
- finaler la fusion via la commande

  git commit
- git log
- supprimer l'ancienne branche temporaire b3

NB : un futur TP permettra d'expérimenter un merge avec un IDE tel que eclipse/java .

2.8. Git élémentaire en mode remote (clone, push, pull)

Se placer dans répertoire de tp (ex: ***c:\tp\tp-git*** *)*

Créer les sous répertoires c:\tp\tp-git***local-git-repositories-devB**

et c:\tp\tp-git***git-shared-repositories**

NB :

- ***git-shared-repositories*** *regoupera des* ***référentiels git partagés*** *(éventuellement accessibles à distance selon accès réseaux ...)*
- ***local-git-repositories-devA*** *regoupera des référentiels locaux pour le développeur devA*
- ***local-git-repositories-devB*** *regoupera des référentiels locaux pour le développeur devB*

Se placer dans le répertoire c:\tp\tp-git***git-shared-repositories**

et créer le sous répertoire c:\tp\tp-git***git-shared-repositories\p1.git**

Se placer dans le répertoire c:\tp\tp-git***git-shared-repositories\p1.git**

et ***initialiser le référentiel distant/partagé*** *en lançant la commande*

```
git init --bare
```

Visualiser le contenu de p1.git

Référencement de ***p1.git*** *en tant que* ***version distante/partagée*** *de* ***local-git-repositories-devA/****p1*

Se placer dans **local-git-repositories-devA/**p1 et lancer les commandes suivantes :

```
git remote -v
(git remote remove originTpP1 pour supprimer ancienne config en cas d'erreur d'URL)
git remote add originTpP1  file:///C:\tp\tp-git\git-shared-repositories\p1.git
git remote -v
```

NB: ***originTpP1*** ***est à voir comme un*** ***alias*** ***pour l'url*** ***file:///C:\tp\tp-git\git-shared-repositories\p1.git***

Première publication partagée depuis ***local-git-repositories-devA/****p1:*

```
git push --set-upstream  originTpP1 main
```

Clonage du référentiel partagé depuis ***local-git-repositories-devB*** *(développeur B) :*

Se placer dans c:\tp\tp-git***local-git-repositories-devB**

```
git clone file:///C:\tp\tp-git\git-shared-repositories\p1.git
```

→ *Visualiser le sous répertoire p1 construit (par clonage)*

cd p1 *(depuis ...-devB)*

git remote -v

origin file:///C:\tp\tp-git\git-shared-repositories\p1.git (fetch)

origin file:///C:\tp\tp-git\git-shared-repositories\p1.git (push)

git checkout main

→ *visualiser les copies locales des fichiers f1.txt , ...*

Modif et publication depuis un développeur de l'équipe dev-B ou dev-A

- *modifier un des fichiers (ex : nouvelle ligne dans f1.txt)*
- *intégration d'éventuelles modifications distantes via git pull*
- *commit local*
- *publication via* ***git push***

Rapatriation des modifications enregistrées/partagées depuis autre développeur dev-B ou dev-A

- ***git pull***
- *visualiser les changements*

Variantes d'URL pour ce TP *:*

file:///C:\tp\tp-git\git-shared-repositories\p1.git *(sur un seul ordinateur)*

file:///Y:\git-shared-repositories\p1.git *où Y: est un lecteur réseau associé à un répertoire partagé par un autre ordinateur via le réseau*

https://github.com/xyz/p1.git *où p1.git est un référentiel git partagé géré par l'utilisateur xyz ayant un compte sur github .*

2.9. Merge git avec résolution de conflit et branches distantes

Expérimenter des commits/push/pull avec conflits à résoudre entre deux utilisateurs travaillant sur un même fichier d'un même projet (ex : p1).

On pourra effectuer se Tp en simulant deux utilisateurs sur le même ordinateur (configuration du TP précédent) ou bien en configurant plusieurs comptes ***gihub*** *ou* ***gitlab*** *ou* ***gitbucket*** *.*

Bonnes pratiques (en général) :

- commit d'abord sur branche locale

- pull avant merge ou push

2.10. Git avec résolution de conflit depuis un IDE

1. Créer un référentiel git comportant un petit projet java/maven très simple
2. Charger ce projet depuis un IDE (ex : eclipse , intelliJ , visual studio code, …)
3. Expérimenter les opérations git basiques (commit, push, pull)
4. Expérimenter les résolutions de conflits depuis l'IDE

2.11. Git en mode http (via github ou autre)

1. Se créer un compte gratuit sur github ou gitlab (une adresse email valide sera vérifiée) et mémoriser les infos d'authentification (username/password ou token ou ...)
2. Créer un nouveau référentiel distant (vide au départ)
3. Publier un projet local vers le référentiel distant (git remote … , git push …)
4. Cloner le référentiel distant ailleurs (git clone , ….)
5. effectuer une série de changements (commit , push , pull, …) sans conflit puis avec conflit soit en ligne de commande , soit via tortoiseGit , soit via un IDE (ex : eclipse).
6. On pourra éventuellement autoriser d'autres développeurs amis à effectuer des publications sur notre référentiel partagé github ou gitlab .

2.12. Expérimentation de certains aspects avancés de GIT

- Rebase , cherry-pick , ...