# TP : Système de gestion de version *Git*

**Consignes**

- Ouvrir le cours sur Git et l'avoir en permanence à l'écran.
- Les manips ci-dessous se feront dans un environnement Linux/Debian (machine virtuelle), où l'étudiant est administrateur. Le login de l'utilisateur est **`user1`**/**`user1`**.
- Pour simuler le fonctionnement d'une équipe et pour s'éviter les problématiques d'accès distant, vous aller créer deux utilisateurs qui vont chacun faire des éditions de leur coté. Ceci sera simulé sur la machine par deux consoles ouvertes côte à côte.
- Bien lire le sujet et respectez scrupuleusement les manips proposées (noms de dossiers notamment).
- En cas d'erreur, prendre le temps de lire le message et essayer de le comprendre. Relire le sujet pour vérifier que la solution n'est pas expliquée. **NE PAS** passer à la question suivante avant d'avoir résolu le problème, au besoin en appelant l'enseignant.
- **cocher au fur et à mesure les questions, pour toujours savoir où vous en êtes !**

**Introduction** Ce TP est en deux parties. Dans la 1re, vous créerez un dépot et ferez des manipulations basiques dessus. Dans la 2e, vous analyserez le dépot d'un projet réel.

# 1 Création et manipulation d'un dépôt

## 1.1 Configuration de la machine et création des utilisateurs

1.1 - Ouvrir un shell, vérifier que git est installé, et noter la version :

**`$ git --version`** :

A defaut, passer "root", mettre à jour les dépôts avec **`apt-get update`**

Puis installer Git avec : **`apt-get install git`**

1.2 - Pour rendre la console plus lisible, on va activer la coloration automatique : Il faut éditer le "fichier modèle" de la configuration du shell, qui sera dupliqué lors de la création d'utilisateurs.
- Passer root, et editer le fichier : **`nano /etc/skel/.bashrc`**
- Recherchez la ligne **`#force_color_prompt=yes`** et dé-commentez là, puis sauvegarder (CTRL-X).

1.3 - Passer root et créer les 2 utilisateurs Alice et Bob avec

```
adduser bob
adduser alice
```

Vous donnerez comme mot de passe le login de l'utilisateur ("**`alice`**" et "**`bob`**").

1.4 - Associer les utilisateurs à un groupe d'utilisateurs **`devs`** :

```
addgroup devs
usermod -g devs bob
usermod -g devs alice
usermod -g devs user1
```

1.5 - Sortir du mode root avec **`exit`**, vérifier qu'on est bien dans le "home" de **`user1`** (avec **`pwd`**). Créer ensuite un dépot git nommé "tpgit", qui servira de dépôt de référence, par rapport auquel les autres utilisateurs vont se synchroniser. Ajouter à la commande l'option "bare" pour indiquer qu'on ne souhaite pas créer de "working copy" (copie de travail), uniquement le "repo" :

**`git init --bare tpgit`**

Vérifier ce qui a été crée, et mesurer sa taille avec **`du -h`** :

1.6 - Afin d'éviter des problèmes de permissions, il faut modifier les permissions de ce dossier pour donner le droit d'écriture au groupe. En se placant dans le dossier parent : **`chmod -R g+ws tpgit`**

Il faut aussi modifier le groupe propriétaire des fichiers. Passer root et faire : **`chgrp -R devs tpgit`**
A quoi sert l'option **`-R`** ?

1.7 - Il faut aussi spécifier ques les utilisateurs pourront utiliser les applications graphiques (accès au server X)

depuis une console ouverte à partir d'une session d'un autre utilisateur.
En "root", tapez : **`xhost +`** Ceci désactive le contrôle d'accès au serveur graphique pour tous les utilisateurs.

1.8 - Indiquer à Git d'utiliser les permissions de groupe au lieu des permissions de fichier. En se plaçant dans le dossier **`/home/user1/tpgit`** et en tant que "root", faire :
**`git config core.sharedRepository true`**

1.9 - Toujours en *root*, installer Firefox avec la commande **`apt-get install firefox-esr`**.

1.10 - Ouvrir deux consoles, adapter leurs fenêtres de façon à les avoir cote à cote. Dans l'une, prendre le rôle de Bob et dans l'autre, Alice, et se placer dans leur "home" respectif (la commande **`pwd`** doit donner **`/home/bob`** et **`/home/alice`**).

## 1.2 Partie 1 : travail de Bob

1.11 - Dans le dossier **`home`** de Bob, cloner le dépot avec **`git clone /home/user1/tpgit`**

1.12 - Descendez dans **`tpgit`** et visualisez le contenu du dossier avec **`ls -l`**
Que constatez-vous ?

Recommencez en utilisant la commande **`ls -al`** : Que pouvez-vous dire ?

1.13 - Créer dans ce dossier un fichier vide nommé **`index.html`** avec **`touch index.html`**

1.14 - Editer ce fichier avec l'éditeur GUI (**`pluma index.html`**) et y placer le squelette d'une page html5 valide, que vous prendrez par exemple sur `https://www.w3schools.com/html/html5_intro.asp`

1.15 - Visualisez l'état du dépot avec **`git status`**

Le fichier est-il versionné ?

1.16 - Déclarer le fichier comme "suivi" avec : **`git add index.html`** puis refaites **`git status`**

Le fichier est-il versionné ?

1.17 - Essayer la commande **`git commit`**
Peut-être cela échoue parce que vous ne vous êtes pas identifié, Git interdit les commit "anonymes". Il faut alors configurer Git. Dans la console de Bob, faire :

```
git config --global user.email "bob@bob.com"
git config --global user.name bob
```

(A répéter de façon similaire pour Alice dans l'autre console)

1.18 - Lire **les** fichiers de config avec la commande **`cat`** et vérifier que vous retrouvez bien les informations :
(leur emplacement est donné dans le cours).
Dans quel fichier sont stockés les informations de l'utilisateur ?

1.19 - Vérifier que ça correspond avec ce qui est donné par **`git config --list`**

1.20 - Relancer la commande **`git commit`**
Git ouvre alors l'éditeur par défaut (Nano) pour y saisir le "message de commit", commentaire obligatoire qui accompagne chaque commit, qui est stocké dans la base et qui décrit les modifications apportées.
Fermer Nano sans rien enregistrer (CTRL-X) et spécifier ce message avec la commande :
**`git commit -m "premier commit"`**
(c'est ce que vous ferez par la suite) :

1.21 - Propager le dépot au dépot source avec **`git push`**. Lire le message d'erreur et appliquer la commande proposée (... **`push.default simple`**), puis refaire le "push".

## 1.3 Alice : clonage du dépot et modifications

1.22 - Dans la console de Alice, cloner le dépot à partir de **`/home/user1/tpgit`**

1.23 - Se placer dans le dossier de travail avec **`cd tpgit`**

1.24 - Editer le fichier index.html et modifier le titre dans le "head" en "Mon super site Web"

1.25 - Ajouter un fichier README avec :
**`echo "mon super site web" > README`**

1.26 - Vérifiez l'état du dépot avec **`git status`**. Quelle est la différence entre le statut de README et index.html :

1.27 - Pour visualiser les différences entre la version du dépot et la version du "*working directory*", lancer la commande **`git diff`**.
Comment sont identifiées les lignes :
- ajoutées :
- supprimées :
Pourquoi ne voit-on pas les modifications du fichier README ? :

1.28 - Ajouter les fichiers à l'index, committer les changements, et vérifier que c'est bon avec **`git status`** (aucun fichier ne doit être affiche comme "non suivi").

1.29 - Propager les changements au dépot d'origine.

## 1.4 Bob : continue son travail

1.30 - Dans la console de Bob, faire un **`git status`** : les changements faits par Alice apparaissent-il ?

1.31 - Importer les changements fait par Alice.

1.32 - Faire un git status et un git log. L'importation a-t-elle produit un nouveau commit ?

1.33 - Comment puis-je visualiser les changements qui ont été fait par cette importation ?

1.34 - Commiter les changements

### 1.4.1 Création d'une 1re branche

1.35 - On souhaite développer la page en y ajoutant un tableau de 3 lignes x 4 colonnes, mais sans modifier la branche "master". Créer une branche **`test1`** et basculer le dossier de travail dessus.
Vérifier avec "status"

1.36 - Editer la page et ajouter la structure du tableau, puis committer les changements.

1.37 - Remplir le tableau avec les valeurs suivantes

| | 2 | 3 | 4 |
|---|---|---|---|
| 2 | 4 | 9 | 15 |
| 3 | 8 | 27 | 64 |

1.38 - Vérifier que l'affichage dans un navigateur est correct, puis committer ces changements.

1.39 - Integrer ces modifications dans la branche principale ("master") (cf. cours)

### 1.4.2 Création d'une 2e branche

1.40 - Créer une deuxième branche **`test2`**, et basculer la copie de travail dessus.

1.41 - Ajouter en dessous un deuxième tableau contenant les produits :

| | 2 | 3 | 4 |
|---|---|---|---|
| 2 | 4 | 6 | 8 |
| 3 | 6 | 9 | 12 |

Commiter les changements.

1.42 - Il y a une une cellule avec une erreur dans le 1er tableau. La voyez-vous ? (indice : opérateur ^). Rebasculer sur la branche "master" et corriger, puis commiter le changement.

1.43 - Rebasculer sur "test2", et importer les changements qui ont été faits dans "master".

1.44 - On peut examiner l'historique du projet avec la commande `git log`, qui montre les différent commit. (Naviguer avec les touches de direction, sortir avec "q".)
Donner les différence entre l'affichage de cette commande et les suivantes :

| | |
|---|---|
| `git log --oneline` | |
| `git log -p` | |

1.45 - En utilisant l'utilitaire wc avec l'option -l, donner la commande permettant de compter le nombre de commits :

### 1.4.3 Gestion d'un conflit

1.46 - Toujours sur "test2", éditer le fichier et remplacer dans le "head" le titre par : `<title>TP Git<title>`, puis committer ce changement.

1.47 - Basculer sur "master", et mettre dans le "title" `<title>TP sur Git<title>`, puis committer ce changement.

1.48 - Essayer ensuite de "merger" les changement fait par la branche "test2". Git vous signale un conflit. Resolvez le en l'éditant, committer, puis propager les changements.

1.49 - Créer un tag nommé `v0.1` pour la version courante de la branche "master".

## 1.5 Alice récupère les changements

1.50 - Dans la console d'Alice, importer les changements fait.
A-t-on récupéré les branches qui ont été crées, ou bien juste la branche "master" ?

1.51 - Essayer de récuperer la version taggée "v0.1". La commande `git checkout v0.1` fonctionne-t-elle ? Quelle commande faut-il faire pour que ça marche ?

## Commandes

Quelles sont les commandes qui modifient le dépot local ? le dépot distant ? (cocher)

| | dépot local | depot distant |
|---|---|---|
| **`log`** | | |
| **`diff`** | | |
| **`commit`** | | |
| **`push`** | | |
| **`pull`** | | |
| **`status`** | | |
| **`init`** | | |
| **`add`** | | |
| **`remote`** | | |
| **`blame`** | | |

# 2 Analyse d'un dépot

Rendez vous sur le projet `https://github.com/onlywei/explain-git-with-d3`

2.1 - Combien y a-t-il de contributeurs ?

2.2 - Clonez le projet dans le dossier **`/tmp`**, en mesurant le temps :

**`time git clone http:...`**

Quel place occupe-t-il sur le disque ?

2.3 - Lancer la commande **`git blame`** sur le fichier **`index.html`**
En vous aidant de **`git help blame`**, expliquer en une phrase à quoi sert cette commande :

2.4 - Qui a commité la ligne 329 :

2.5 - Notez le hash de ce commit :

2.6 - Avec la commande **`git show`** sur ce commit, notez le message de l'auteur du commit :

2.7 - Date de ce commit :

2.8 - En vous aidant de la commande **`git show`**, donner le hash du commit correspondant au tag de la version 1.0.0

2.9 - Avec **`git show`**, qui est l'auteur de ce commit ?

Date ?

2.10 - Avec la même commande que la question 2.8 -, comptez les commit en faisant un "pipe" avec l'utilitaire **`wc`**
Donner la commande exacte :

et le résultat :