TP `git`

Installation de `git >= 2.23`

Nous utiliserons les commandes `restore` et `switch` introduites dans la version `2.23` de `git`, ce sont des alternatives aux commandes historiques `checkout` et `reset` qui sont surchargées et causent trop de confusions.

- Si vous utilisez Debian 11 `bullseye` ou si vous utilisez `homebrew` sur macOS, installez simplement le paquet `git`.
- Si vous utilisez Debian 10 `buster`, la version de `git` distribuée est `2.20`. Vous pouvez installer une version plus récente de `git` depuis les backports :
  i. Ajoutez la ligne suivante dans un fichier nommé `/etc/apt/sources.list.d/backport.list` :
  ```
  deb https://deb.debian.org/debian buster-backports main
  ```
  ii. Téléchargez les informations des nouveaux paquets, installez `git` en spécifiant que vous voulez qu'il soit téléchargé depuis le dépôt `buster-backports` :
  ```
  # apt update
  # apt install -t buster-backports git
  ```
  iii. Vérifiez que la commande `git --version` retourne une version supérieure à `2.23`.

Installation de `tig`

De nombreuses interfaces graphiques permettent de visualiser l'historique et l'état du dépôt.

Le paquet `tig` fournit une interface curses permettant une telle visualisation dans votre terminal.

1. Installez le paquet `tig`

Configuration

Les commits contiennent le nom et l'adresse mail de la personne qui l'a crée (cette personne est appelée *committer*).

1. Pour que les commits vous identifient correctement comme étant l'auteur et/ou le committer, tapez les commandes suivantes :

   ```
   $ git config --global user.name "<Prénom> <Nom>"
   $ git config --global user.email "<adresse_mail>"
   ```

2. Observez le résultat dans le fichier `~/.gitconfig`.

Solo linéaire : création du dépôt et premier commit

1. Sur votre machine locale, dans votre répertoire de travail d'administration système, créez un répertoire nommé `git-test/` et initialisez-y un dépôt git.
2. Dans ce répertoire, créez un fichier `plop` dont le contenu est la chaîne `hop`.
3. Ajoutez le fichier `plop` à la staging area (index).
4. Créez un commit dont le commentaire est "mon premier commit".

Solo linéaire : deuxième commit et observation

1. Modifiez le fichier `plop` en remplaçant la chaîne de caractères `hop` par `hap`.
2. Observez ce que retourne chaque commande du slide "Informations".
3. Ajoutez le fichier `plop` à la staging area (index).
4. Observez ce que retourne chaque commande du slide "Informations".
5. Créez un fichier `toto` dont le contenu contient 4 lignes de votre choix.
6. Ajoutez le fichier `toto` à la staging area (index).
7. Observez ce que retourne chaque commande du slide "Informations".
8. Modifiez la 2e ligne du fichier `toto`.
9. Observez ce que retourne chaque commande du slide "Informations".
10. Ajoutez le fichier `toto` à la staging area (index).
11. Observez ce que retourne chaque commande du slide "Informations".
12. Commitez vos changements.
13. Observez ce que retourne chaque commande du slide "Informations".

Solo linéaire : troisième commit et observation

1. Modifiez la 2e ligne du fichier `toto`.
2. Ajoutez le fichier `toto` à la staging area (index).
3. Observez ce que retourne chaque commande du slide "Informations".
4. Commitez vos changements.
5. Observez ce que retourne chaque commande du slide "Informations".
6. Baladez-vous dans vos 3 commits en utilisant `tig`.

Solo linéaire : oups

1. Éditez le fichier `toto` et supprimez sa première ligne.
2. C'est une boulette, récupérez le contenu de la version précédente du fichier `toto`.
3. Éditez le fichier `toto` et modifiez sa 4e ligne.
4. Ajoutez le fichier `toto` à la staging area.
5. C'est encore une boulette, nettoyez la staging area ainsi que votre répertoire de travail de sorte à ce que le fichier `toto` soit identique à celui du 3e commit.

Solo programmation

À faire si vous êtes en avance, sinon passez au prochain paragraphe et revenez-y plus tard.

1. Lisez la page du slide intitulée "Ne versionner que le source, ignorer les sous-produits".
2. Créez un répertoire nommé `programmation/` dans votre répertoire de travail.
3. Créez-y un fichier de type "Hello world" nommé `hello.c`, et commitez-le.
4. Compilez-le et exécutez-le.
5. Que donne la commande

   ```
   $ git status
   ```

6. Créez un fichier `.gitgnore` à la racine de votre répertoire de travail (worktree) de sorte à éviter de commiter par inadvertance le fichier compilé.
7. Que donne la commande

   ```
   $ git status
   ```

8. Si les fichier compilés n'apparaissent plus, commitez le fichier `.gitgnore`.

9. Modifiez le fichier `hello.c` de sorte à ce qu'il fasse plutôt "Hello git".
10. Compilez-le et exécutez-le.
11. Commitez-le.

Solo branches

1. Créez une branche nommée `essai`
2. Allez dessus (*i.e.* faites de `essai` votre branche courante).
3. Créez un fichier `test.txt`, ajoutez-le à la staging area et commitez.
4. Créez un tag nommé `milieu-test`
5. Modifiez le fichier `test.txt`, ajoutez-le à la staging area et commitez.
6. Regardez le contenu du fichier `.git/HEAD` et des différents fichiers et sous-répertoires du répertoire `.git/refs/`
7. Essayez de comprendre comment `git` gère ses références (tags, branches, branche courante).
8. Retournez sur la branche `master`.
9. modifiez le fichier `plop`, ajoutez-le à la staging area et commitez.
10. Mergez la branche `essai` dans la branche courante `master`.
11. Observez le résultat avec `tig`

Solo branches bonus

À faire si vous êtes en avance, sinon passez au prochain paragraphe et revenez-y plus tard.

1. Imaginez un DAG un peu complexe avec des tags à certains endroits, et amusez-vous à réaliser ce DAG comme le DAG des commits de votre dépôt avec les commandes `add,commit,branch,switch,tag,merge`.

   Si vous ne voulez pas perdre de temps à créer du contenu à commiter mais vous concentrer sur la gestion du DAG, vous pouvez au choix :

   - dessiner directement votre DAG dans le simulateur learngitbranching
   - utiliser l'alias suivant qui crée un fichier avec un nom aléatoire et un contenu aléatoire puis l'ajoute dans la staging area :

     ```
     $ alias Stage_un_truc='RND=${RANDOM};echo ${RND}>fichier_${RND};git add fichier_${RND}'
     ```

     Ainsi, il suffit de taper la commande `Stage_un_truc` pour avoir quelque chose à commiter. De même, utilisez l'option `-m` de `git commit` pour mettre un message rapidement.
2. Mettez-vous sur une branche qui a de nombreux ancêtres, visez un commit particulier et essayez d'en faire le commit courant sans utiliser son hash, mais avec l'adressage relatif (cf slides).

Collectif : constituer un groupe si vous n'avez pas encore de groupe projet

1. Lorsque vous avez fini avec les exercices précédents, connectez-vous sur le salon mattermost `sysadmin_git` et écrivez "j'ai fini la partie solo".
2. Dès qu'un nombre suffisant d'étudiant·es a fini la partie solo, il·les créent un groupe avec un salon jitsi pour travailler ensemble sur un dépôt distant commun.
3. Voici quelques infos pour pouvoir cloner le dépôt distant commun:
   - la machine distante est un conteneur accessible via le nom de domaine `yologit.netlib.re`
   - les fingerprints des clefs publiques du serveur SSH de ce conteneur sont :

```
ECDSA   : SHA256:9j1WU56gjFkSACA3zILrVv2EdTZ2Aphs2sxuas/P004
RSA     : SHA256:UBSnfyQlOscdN6T3ftStt+ne6OL0fiqVA9h6+wVisVM
ED25519 : SHA256:PYktqMJQNdLdc8IGCnlO4On6sCfQYR6X4hR1ANKN/9I
```

- l'user qui héberge le dépôt distant sur ce conteneur s'appelle `gituser`,
- votre clef publique SSH a été installée dans le fichier `~/.ssh/authorized_keys` de l'user `gituser` de sorte que vous n'avez pas besoin de mot de passe pour vous connecter.
- le nom du dépôt distant attribué à votre groupe vous sera donné par un enseignant.

4. Essayez de vous connecter par SSH sur la machine `yologit.netlib.re` en tant qu'user `gituser`. Vérifiez et validez le fingerprint. Vous serez ensuite interdit·e de vous connecter: c'est normal, le shell de l'user `gituser` a été configuré pour n'accepter que des commandes git. En effet, le fichier `/etc/passwd` de la machine `yologit` contient la ligne :

   `gituser:x:1000:1000:,,,:/home/gituser:/usr/bin/git-shell`

5. Retournez dans votre répertoire de travail d'administration système (quittez `git-test/`) et clonez le dépôt :

   `$ git clone gituser@yologit.netlib.re:<nom_du_depot_distant>`

   (si vous devez utiliser tor, ajoutez un bloc dans votre `./ssh/config` avec une option `ProxyCommand` comme pour votre conteneur)

Collectif : constituer un groupe si vous avez un groupe de projet

1. Lorsque vous avez fini avec les exercices précédents, reseignez-vous sur l'avancement des membres de votre groupe. Si vous avez de l'avance, attendez votre groupe en faisant les paragraphes "Solo programmation" et "Solo branches bonus".
2. Une fois que tout le monde est prêt, manifestez-vous sur le salon mattermost `sysadmin_git` et indiquez que votre groupe est prêt pour travailler ensemble sur un dépôt distant commun.
3. Voici quelques infos pour pouvoir cloner le dépôt distant commun:
   - la machine distante est un conteneur accessible via le nom de domaine `yologit.netlib.re`
   - les fingerprints des clefs publiques du serveur SSH de ce conteneur sont :

     ```
     ECDSA   : SHA256:9j1WU56gjFkSACA3zILrVv2EdTZ2Aphs2sxuas/P004
     RSA     : SHA256:UBSnfyQlOscdN6T3ftStt+ne6OL0fiqVA9h6+wVisVM
     ED25519 : SHA256:PYktqMJQNdLdc8IGCnlO4On6sCfQYR6X4hR1ANKN/9I
     ```

   - l'user qui héberge le dépôt distant sur ce conteneur s'appelle `gituser`,
   - votre clef publique SSH a été installée dans le fichier `~/.ssh/authorized_keys` de l'user `gituser` de sorte que vous n'avez pas besoin de mot de passe pour vous connecter.
   - le nom du dépôt distant attribué à votre groupe vous sera donné par un enseignant.
4. Essayez de vous connecter par SSH sur la machine `yologit.netlib.re` en tant qu'user `gituser`. Vérifiez et validez le fingerprint. Vous serez ensuite interdit·e de vous connecter: c'est normal, le shell de l'user `gituser` a été configuré pour n'accepter que des commandes git. En effet, le fichier `/etc/passwd` de la machine `yologit` contient la ligne :

   `gituser:x:1000:1000:,,,:/home/gituser:/usr/bin/git-shell`

5. Retournez dans votre répertoire de travail d'administration système (quittez `git-test/`) et clonez le dépôt :

   `$ git clone gituser@yologit.netlib.re:<nom_du_depot_distant>`

   (si vous devez utiliser tor, ajoutez un bloc dans votre `./ssh/config` avec une option `ProxyCommand` comme pour votre conteneur)

Collectif : travailler en parallèle sur des fichiers disjoints

1. Dans le répertoire de travail de votre nouveau dépôt local se trouve un répertoire nommé `perso/`. Créez-y un fichier dont le nom est votre `prenom.nom` et ajoutez-y des informations (non-compromettantes) sur vous.
2. Commitez ces changements et poussez-les sur le dépôt distant.
3. Recommencez l'opération en rajoutant des détails dans le fichier `perso/<prenom.nom>` et repoussez votre nouveau commit.
4. Recommencez jusqu'à devoir merger les changements effectuées par un·e autre étudiant·e de votre groupe.
5. Continuez jusqu'à ce que tout le monde ait pu faire des `pull` des `push` et des `merge`.
6. Observez l'historique et le DAG des commits.

Collectif : travailler en parallèle sur des parties distinctes d'un même fichier, sur une même branche

Le but de ce paragraphe est de jouer au cadavre exquis inventé par les surréalistes, mais avec `git`.

1. Dans le répertoire de travail de votre dépôt local partagé se trouve un répertoire nommé `cadavre_exquis/`. Dans ce répertoire se trouve un fichier nommé `template.txt`.
2. Faites en sorte qu'un·e étudiant·e copie le template en un fichier `test1.txt`, le pousse et que tout le monde le récupère et se trouve sur la même branche.
3. Répartissez-vous chacun·e une des 5 lignes à remplir (1. substantif, 2. adjectif, 3. verbe, ...).
4. Une fois que vous êtes d'accord, créez chacun.e une branche ayant pour nom votre prénom.
5. Modifiez le fichier `test1.txt` en ajoutant au niveau du tiret qui correspond à votre ligne, un ensemble de mots admissible.
6. Poussez votre branche.
7. Faites merger les différentes branches dans la branche `master` par un·e étudiant·e du groupe.
8. Récupérez la branche master du dépôt collectif.
9. Lisez le résultat du cadavre exquis.

Collectif : travailler en parallèle sur des parties communes d'un même fichier en étant sur une même branche

1. Dans le répertoire de travail de votre nouveau dépôt local, créez un fichier nommé `baston`, ajoutez-y du contenu, commitez-le et poussez vos changements sur le dépôt distant.
2. Recommencez l'opération en rajoutant des détails dans le fichier `baston` et repoussez votre nouveau commit.
3. Recommencez jusqu'à devoir merger les changements effectués par un·e autre étudiant·e de votre groupe.
4. Continuez jusqu'à ce que tout le monde ait pu faire des `pull` des `push` et des `merge` en rapport avec le fichier `baston`.
5. C'est facile ou on s'organise ensemble ?

Collectif : choisissez votre workflow

Si vous êtes arrivé·e à ce paragraphe dans les 3h, félicitations !

1. Discutez avec votre équipe, et répartissez-vous des tâches avec des fichiers à modifier, des noms de branches, et éventuellement des rôles pour savoir qui merge quoi (ou pas).
2. Essayez de les réaliser et amusez-vous.
3. Gardez en tête que git est un outil flexible qui n'impose pas d'organisation collective particulière, que le meilleur workflow est celui que vous déciderez ensemble.