


# Le gestionnaire de sources GIT

## 1 Git en local


Git est un *gestionnaire de code source* (ou SCM pour *Source Control Management*), c'est-à-dire un outil qui permet de conserver l'historique des modifications apportées à un projet de programmation. Git prend également en charge l'intégration des modifications apportées par les différents développeurs qui collaborent simultanément sur le projet. À l'origine, Git a été conçu pour le développement du noyau Linux. C'est un outil très performant, et très utilisé dans l'industrie et la recherche. Dans ce TP, vous allez apprendre les bases de l'utilisation de Git qui sont nécessaires pour la réalisation du projet Unix.

1. Git s'utilise principalement en ligne de commande. Commencez par vérifiez que la `git` est disponible sur votre système en tapant simplement `git`  dans un terminal. Si tout se passe bien, vous obtiendrez l'aide de la commande `git`.
2. Avant de commencer à utiliser Git, vous devez lui indiquer votre nom et votre adresse email. Vous pouvez le faire grâce aux deux commandes suivantes:

---

```
$ git config --global user.name "prenom nom"
$ git config --global user.email "prenom.nom@dauphine.psl.eu"
```

---

3. Ok. Maintenant, rendez-vous dans votre répertoire *home* et créez un nouveau répertoire `git-test` avec la commande `mkdir`, puis rendez-vous dans le répertoire `git-test` et transformez le en *dépot Git* en utilisant la commande `git init` .
- Attention:** soyez très attentif avec les commandes Git, certaines erreurs peuvent être compliquées à résoudre. Pour cette question, soyez bien sûr d'être dans le répertoire `git-test` lorsque vous tapez `git init`.
4. Dans votre répertoire `git-test`, créez un fichier `README` avec votre nom, ainsi qu'un programme `bonjour.c` qui affiche "bonjour" lorsqu'on l'exécute. Puis compilez et testez votre programme avec `gcc`.
5. Si tout s'est bien passé, vous devriez avoir trois fichiers dans votre répertoire `git-test`: `README`, `bonjour.c` et `a.out`. Git permet de conserver l'historique des changements dans vos fichiers sources mais pour y parvenir, il doit savoir quels sont ces fichiers source. Un fichier source est un fichier qui ne peut pas être recréé à partir d'un autre fichier source. En d'autres termes, les fichiers source représentent l'ensemble minimal des fichiers nécessaires pour générer votre exécutable. Parmi les 3 fichiers ci-dessous, quels sont les fichiers source? Donnez des exemples de fichiers qui ne doivent pas faire partie des sources.
6. Ajoutez les fichiers sources du projet avec la commande `git add fichier1 ...`. Vous pouvez répéter cette commande autant de fois que nécessaire.
7. Effectuez votre premier *commit* en utilisant la commande `git commit`. Le *commit* va enregistrer les sources de votre projet telles qu'elles sont actuellement en tant que première version de votre projet.

Lorsque vous tapez la commande, Git ouvre un éditeur de fichier et vous demande de fournir une description des changements effectués depuis la dernière version. Comme il s'agit du

- premier commit, entrez le message « Initial import »<sup>1</sup>. Enregistrez et quittez l'éditeur pour compléter votre commit.
8. Apportez une modification quelconque au fichier README, (par exemple, ajoutez votre adresse email à la fin du fichier) et sauvegardez le fichier. Modifiez également le fichier `bonjour.c` pour qu'il affiche « salut » à la place de « bonjour ». Sauvegardez et retournez dans le terminal pour taper `git diff`. Qu'observez vous? Quel est le rôle de la commande `git diff` (aidez vous du man de `git diff` si nécessaire.)
  9. Ajoutez l'ensemble des modifications apportées à votre base de code en tapant de nouveau `git add` suivi du nom des fichiers source modifiés depuis le dernier commit (README et `bonjour.c`). Puis, effectuez votre deuxième commit en tapant `git commit`. Entrez un nouveau message pour décrire les changements apportés, et quittez l'éditeur pour finaliser le commit. **Note:** un bon message de *commit* pour la modification proposée serait « *Add author email and improve main function.* ».
  10. Consultez l'historique des modifications apportées à votre projet depuis la version initiale en utilisant la commande `git log`. Vous pouvez également utiliser l'interface graphique `gitk` pour visualiser l'historique de votre projet. **Note:** si vous tapez `gitk` dans le terminal, vous n'aurez plus la main sur le terminal jusqu'à la fin de l'exécution de la commande `gitk` (i.e. jusqu'à ce que vous fermiez l'interface graphique). Pour lancer `gitk` tout en gardant la main sur le terminal, vous devez taper la commande `gitk &`.
  11. Chaque commit est identifié par un *hash* SHA-1 représenté sous la forme d'une séquence de caractères alpha-numériques. À l'aide de la commande `git log`, trouvez et copiez le *hash* de votre premier commit (celui que vous avez normalement décrit par le message « *Initial import.* »).
  12. En utilisant `git diff hash-commit-1 hash-commit-2`, vous pouvez visualiser l'ensemble des modifications apportées entre deux commits. Testez cette commande pour visualiser la différence entre le premier et le deuxième commit.
  13. Pour faciliter la tâche, le commit le plus récent est toujours référencé par le mot clé HEAD. Il est également possible de faire référence au commit précédent le dernier commit grâce au mot clé HEAD^. Reproduisez le résultat de la question précédente sans utiliser les *hash* des commits.
  14. Certaines modifications sont inutiles et ne doivent pas être enregistrées dans l'historique. Si vous avez apporté une modification que vous ne voulez pas conserver. Vous pouvez utiliser la commande `git restore fichier` pour rétablir le fichier tel qu'il était lors de votre dernier commit. Attention, vos changements seront perdus. Apportez une modification au fichier `bonjour.c`, et testez la commande `git restore`.

Vous connaissez maintenant les commandes de base pour manipuler un dépôt Git. Il existe beaucoup d'autres commandes utiles, vous êtes très fortement encouragés à suivre le tutoriel proposé par les créateurs de Git disponible ici <https://git-scm.com/docs/gittutorial>. (À la maison.)

## 2 Synchronisation de dépôts Git

Dans cet exercice, vous allez utiliser le service web Github qui permet d'héberger des dépôts git, et de les rendre accessibles partout sur internet.

---

1. Traditionnellement, les message de commit sont écrit en anglais, vous pouvez écrire en Français si vous préférez mais restez cohérents au cours du développement de votre projet.

1. Rendez vous sur `github.com` et créez un compte si vous n'en n'avez pas déjà un. Vous pouvez utiliser n'importe quel nom d'utilisateur, cela n'a pas d'importance. **Attention:** Github vous demandera de confirmer votre adresse email, il faut donc entrer une adresse email valide. (Par exemple, celle de Dauphine.)
2. Une fois que vous avez confirmé votre adresse, rendez vous sur la page Github du cours en cliquant sur le lien suivant.

`https://classroom.github.com/a/NCTJdt2G`

Acceptez le devoir (*assignment*) et cliquez sur votre nom dans la liste des étudiants. Une fois que l'opération est terminée, cliquez sur le lien pour vous rendre sur la page de votre projet (*assignment*). Vous venez de créer un nouveau dépôt (clone de `gittest2`) dans votre espace GitHub. Cliquez sur *code*, et copiez l'url de votre dépôt sur Github. L'url doit avoir la forme suivante `https://github.com/DauphinePSL/gittest2-username`.

3. Vous ne pouvez pas modifier directement votre dépôt distant (celui hébergé sur Github). Pour mettre à jour votre dépôt distant, vous devez en obtenir une copie locale, apporter les modifications dans votre copie locale, créer de nouveaux commits, et synchroniser votre copie locale avec les commandes `git pull` et `git push`. Nous allons voir comment faire ça dans les questions suivantes.

Depuis le terminal, rendez vous dans votre répertoire *home*, et clonez le dépôt avec la commande

```
git clone url-du-dépôt-git depot1
```

Grâce à cette commande, vous venez d'obtenir une copie locale de votre dépôt Github. Utilisez `git log` et `gitk` analyser le projet.

4. Modifier le fichier `README.md` pour ajouter votre nom dans la liste des auteurs. Puis faites un `git add` et un `git commit` pour enregistrer les changements dans votre dépôt local.
5. Rendez vous sur la page web de votre dépôt, pour constater que vos changements ne sont pas présents. C'est parce que vous n'avez pas encore synchronisé votre dépôt local avec votre dépôt GitHub. Avec la commande `git log`, comment pouvez vous voir que les deux dépôts ne sont pas synchronisés? Même question avec `gitk`.
6. Pour synchroniser votre dépôt local, (avec la modification des auteurs dans le `README.md`) avec le dépôt distant sur GitHub tapez `git push`. Après l'opération, vérifiez que les deux dépôts sont synchronisés avec la commande `git log`, puis avec `gitk`. **Note:** N'oubliez jamais d'envoyer vos modifications à la fin de votre session de travail avec `git push`! Cela vous évitera des problèmes.
7. Si vous travaillez depuis plusieurs endroits, par exemple depuis la fac et depuis chez vous. Vous allez devoir synchroniser vos trois dépôts: celui de la fac, celui de chez vous et celui de GitHub. Pour comprendre comment faire ça, nous allons créer un deuxième clone de votre dépôt

Dans votre répertoire *home*, créez un deuxième clone du dépôt Github.

```
git clone url-du-dépôt-git depot2
```

Ici, les deux dépôts se trouvent sur le même ordinateur, mais on imaginera que le premier dépôt est sur un ordinateur de la fac, et que le deuxième est sur votre ordinateur personnel, chez vous.

8. Notez qu'à ce stade, les 3 dépôts sont synchronisés, c'est-à-dire qu'ils contiennent le même historique de commits. Rendez-vous dans `depot1`, et faites une modification **au début** du fichier `README.md` puis faite un commit pour enregistrer ce changement (`git add README.md` puis `git commit`). Envoyez vos changements sur le dépôt de Github avec `git push`.
9. Rendez vous dans `depot2` et faites une modification **à la fin** du fichier `README.md`. Tentez d'envoyer vos changements sur le dépôt git avec `git push`. Que se passe t'il? Pourquoi?
10. Toujours depuis le répertoire `depot2`, tapez la commande `git pull`, puis consultez le fichier `README.md`, que constatez vous? Observez et interprétez l'historiques des modifications avec `git log` ou `gitk`. Que s'est-il passé?
11. Que devez vous faire pour que les 3 dépôts soit de nouveau synchronisés?
12. À votre avis, que se serait-il passé si vous aviez apporté les changements sur la même ligne du fichier `README.md`? Vous pouvez en lire plus à cette adresse <https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/>.