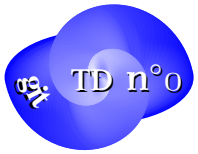

Versionnage avec `git`

Premiers pas

Dans ce document, vous allez vous familiariser avec `git`.

A la différence de `cvs` et `svn` qui sont des systèmes de versionnage centralisés — c'est à dire où il y a partage d'un même dépôt — `git` est un système de versionnage décentralisé, c'est à dire qui permet de collaborer tout en autorisant chacun à avoir son propre dépôt. Il existe d'autres outils de versionnage décentralisé (ou distribué) tels que `bzr`, `mercurial`, `darcs`, etc...

La commande principale est `git`. Les fonctionnalités offertes par cet utilitaire sont accessibles par des sous-commandes : `git` *`sous-commande arguments...`*

Le site web de `git` est ici :`http://git-scm.com/` et un petit tutoriel sympa se trouve là : `http://rogerdudler.github.io/git-guide/index.fr.html`

Exercice 1. Un peu de configuration Pour pouvoir accéder à des dépôts git en dehors de l'université, vous allez devoir configurer les proxys. Ajoutez les variables d'environnement `http_proxy` et `https_proxy` au fichier .bashrc situé à la racine de votre HOME selon les indications qui vous seront données en TD.

Exercice 2. Compte Gitlab

- Créer un compte sur Gitlab : `https://gitlab.com/`
- Choisir un nom de login pas trop long que vous n'oublierez pas.
- Créez un nouveau dépôt nommé "td0" via l'interface Web de gitlab
- choisissez d'initialiser votre projet via l'interface Web, sans spécifier l'option gitignore.

Exercice 3. Copie locale

- Crééz un répertoire gittest dans votre home
- placez vous à l'intérieur et tapez la commande suivante : `git clone URL_de_votre_repo` dans le style :

```
git clone https://gitlab.com/monlogin/mondepot.git
```

- Déplacez-vous dans le répertoire ainsi créé qui porte le nom de votre dépôt

Exercice 4. Prise de contact

Le première chose à faire avec n'importe quel utilitaire en ligne de commande est de consulter l'aide offerte par celui-ci :

```
git --help
```

Ce qui donne entre autres :

```
 1
 2    add         Add file contents to the index
 3    branch      List, create, or delete branches
 4    checkout    Checkout a branch or paths to the working tree
 5    clone       Clone a repository into a new directory
 6    commit      Record changes to the repository
 7    diff        Show changes between commits,
 8                commit and working tree, etc
 9    init        Create an empty Git repository or reinitialize
10    log         Show commit logs
11    merge       Join two or more development histories
         together
12    mv          Move or rename a file, a directory or a
         symlink
13    pull        Fetch from and merge with another repository
14                or a local branch
15    push        Update remote refs along with associated
         objects
16    rm          Remove files from the working tree and
17                from the index
18    show        Show various types of objects
19    status      Show the working tree status
```

Puis on peut obtenir de l'aide sur une sous-commande particulière, par exemple :

```
git help clone
```

```
git help init
```

Exercice 5. Configurer son identité.

La seconde chose à faire avec `git`, c'est de vous munir d'une identité

```
    git config --global user.name "Your Name"
    git config --global user.email you@example.com
```

On va vérifier ça :

```
    git config --list
```

Votre identité est maintenant stockée sous la forme :

```
1 [user]
2         name = Your Name
3         email = you@example.com
```

dans un fichier .gitconfig placé à la racine de votre HOME.

Exercice 6. Ajout de fichiers au projet Affichons maintenant le contenu de la copie locale de votre dépôt.

- pour afficher l'arborescence, invoquez : `tree -a git` a placé de nombreux fichiers ici déjà, mais ils servent simplement à son bon fonctionnement !
- Crééz à présent un fichier "essai.html", insérez y le doctype, les balises head title body footer
- Sauvegardez
- Inspectez les modifications :

  ```
      git status
  ```

- `git` vous informe alors de la présence d'un fichier non versionné
- nous allons donc l'ajouter :

  ```
  git add essai.html
  ```

- Puis à nouveau :

  ```
      git status
  ```

  `git` nous informa alors qu'un nouveau fichier a été indexé mais non commité, c'est à dire enregistrer dans le dépôt
- Pour effectuer cet enregistrement, il faut exécuter un `git commit`. Renseignons nous d'abord sur la commande :

  ```
  bzr help commit
  ```

  Puis mettons la en pratique :

  ```
  git commit -m "ajout fichier initial essai.html"
  ```

- Maintenant `git status` n'affiche plus rien, ce qui signifie qu'il n'y a plus de modification non-commitée
- `git log` liste l'historique des commits qui pour l'instant ne contient qu'une seule révision :

  ```
  $ git status
  $ git log
  ```

Exercice 7. Publication sur le dépôt distant On va publier le nouveau fichier sur la branche *master* en faisant un push :

```
git push origin master
```

Attention : le login et le mot de passe github sont demandés

Ajoutez un header à votre fichier et faites un update :

```
git commit -am "legeres modifications"
```

Puis poussez le nouveau fichier vers GitHub :

```
git push
```

Exercice 8. Ajout d'une feuille de style

Ajoutez maintenant une feuille de style `topstyle.css`mettant un belle couleur de fond à votre page Web. Commitez :

```
git commit -am "ajout feuille de style"
```

l'option "-a" `git commit -a` permet de ne pas executer : `git add topstyle.css`

Exercice 9. Fichiers à ignorer On peut spécifier des fichiers à ignorer dans le répertoire de travail en plaçant dans celui-ci un fichier `.gitignore` qui contient des choses comme :

```
1 # Fichiers .class
2  *.class
3 # Package Files
4 *.jar
5 *.war
6 *.ear
7 # Divers
8 toto.txt
9 *~
```

Versionnez ensuite le fichier .gitignore et ajoutez-le à votre dépôt !

Exercice 10. Fichier de configuration.

Observez le fichier de configuration du dépôt git dans le fichier config du sous-dossier .git Un exemple plus complet est visible à l'adresse : `https://gist.github.com/pksunkara/988716` Beaucoup de choses sont configurables...

Exercice 11. Revenir en arrière Effectuez plusieurs commits/push et consultez ensuite la liste des commits à l'aide d'une commande `git` puis sur la console Web de votre projet Github. Si vous souhaitez annuler le dernier commit publié vous devez copiez le numéro du commit concerné et effectuer la commande suivante :

```
git revert numero_commit ( exemple : 1f45093 )
```

Votre copie locale sera réinitialisée telle qu'elle se trouvait avant le dernier commit.

Exercice 12. Création d'une branche pour un correctif

- vous êtes dans le dossier de votre depot
- Faites une branche dédiée à un correctif, par exemple : `ajoutheader` et placez-vous dans cette nouvelle branche à l'aide des commandes :

  ```
  git branch ajoutheader
  git checkout ajoutheader
  ```

  ou directement par la commande : `git checkout -b ajoutheader`
- ajoutez un header au fichier essai.html
- commitez : `git commit -am "ajout header"`
- revenez dans la branche master :

  ```
  git checkout master
  ```

- intégrer le correctif au master :

  ```
  git merge ajoutheader
  ```

  La fusion est automatique dans ce cas car il n'y a aucun conflit.
- La fusion est terminée et on peut supprimer la branche ajoutheader :

  ```
  git branch -d ajoutheader
  ```

Exercice 13. En cas de conflit

- crééz deux autres branches correctives `modifheader1` et `modifheader2`
- mettez deux textes différents dans le header dans chacun des 2 correctifs
- retourner dans le master et intégrez-y le premier correctif :

  ```
  git checkout master
  git merge modifheader1
  ```

— puis le second :

```
git merge modifheader2
```

— Mais là il va y avoir conflit et la fusion automatique ne peut pas se faire. On appelle donc :

```
git mergetool
```

— Choisissez l'outil d'édition de différences (kdiff3 sous Linux ou opendiff sous Mac) et résolvez le conflit vous même.

— Une fois le conflit résolu, faites simplement :

```
git commit
```

pour intégrer les modifications.

— poussez les mises à jour vers github

— Etudiez la documentation de Git sur la fusion qui propose un exemple similaire, schémas à l'appui :
`http://git-scm.com/book/fr/Les-branches-avec-Git-Brancher-et-fusionner%C2%A0%3A-les-bases`

Exercice 14. Travail en collaboration

— Autorisez votre voisin sur votre repo et réciproquement via l'interface de GitHub (Via les menus Settings => Collaborators)

— Faites un clone du repo de votre voisin à l'aide de la commande :

```
git clone https://github.com/github_mon_voisin/
   depot_voisin.git
```

— Faites une branche corrective sur ce dépôt et laisser votre voisin faire le merge quand vous l'aurez terminée

— Gérez les éventuels conflits et faites régulièrement des `git status`

— N'oubliez pas non plus les `git pull` pour récupérer les éventuelles modifications.

Compléments de configuration pour git

Comme vous avez pu vous en apercevoir, le proxy est authentifié, ce qui nécessite de saisir votre identifiant et votre mot de passe pour chaque connexion, ce qui devient vite pénible. Pour éviter ce désagrément, vous pouvez intégrer votre identifiant et mot de passe directement dans la définition des settings de `http.proxy` et `https.proxy`. Normalement, votre identifiant est votre adresse email. Supposons un étudiant Bilbo Baggins ayant pour mot de passe `gandalf`. Il lui faut alors modifier sa configuration globale de git de la manière suivante :

```
proxyid='bilbo.baggins%40etu.univ-orleans.fr'
proxymdp=gandalf
git config --global http.proxy http://$proxyid:$proxymdp@wwwcacheetu.univ-orleans.fr:3128/
git config --global https.proxy https://$proxyid:$proxymdp@wwwcacheetu.univ-orleans.fr:3128/
```

les caractères `%40` sont en fait une représentation codée (au format URL) du caractère `@`.