Spécialités GIS / IMA 3ème année

Introduction outil `GIT`

© W. Rudametkin – Polytech Lille
Adaptation : M.E. Kessaci, O. Caron, J. Dequidt

`https://rudametw.github.io/teaching/`

1 Objectifs

— Apprendre à se servir d'un Logiciel de Gestion de Versions [1].
— Maîtrisez le versionnement d'un projet logiciel.
— Partager un projet et travailler en équipe.

IMPORTANT !
Lisez attentivement la sortie de ***chaque*** commande `git`. Il est fortement conseillé de travailler exclusivement sur terminal pour ce TP.

2 Contexte et préparation

2.1 Configuration de votre environnement

`.bashrc` et `git-prompt.sh` Nous vous fournissons une configuration avancée de `bash` (l'interpréteur en ligne de commande utilisé par défaut) avec une aide GIT intégré. Utiliser un prompt spécial GIT est très pratique pour visualiser rapidement où vous en êtes (connaître la branche sur laquelle vous travaillez par exemple). La commande suivante exécute un script shell qui modifie votre prompt, nous vous proposons une qui est simple, git-prompt.sh [2]. Ce prompt ne sera actif que dans les répertoires de vos projets GIT :

```
source ~wrudamet/public/bashrc-students
```

Pour ne pas devoir lancer cette commande à chaque connexion, vous pouvez l'insérer au début de votre fichier `~/.bashrc`.

Astuce si vous êtes sur votre ordinateur personnel !
Optionnellement, à l'aide de `scp`, `rsync` ou un outil graphique qui supporte le protocole `sftp`, vous pouvez copier le fichier `~wrudamet/public/bashrc-students` pour adapter votre `fichier ~/.bashrc` local et télécharger `git-prompt.sh`, en suivant la documentation, pour intégrer le support de git manuellement.

Terminaux Nous vous proposons de travailler sur plusieurs terminaux ouverts simultanément, une disposition possible est indiquée dans la figure 1 mais vous pouvez les ranger à votre convenance tout au long du TP.

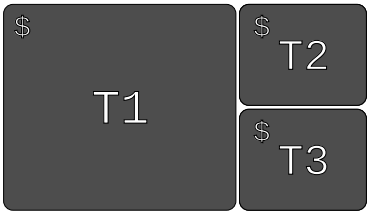

Figure 1 : Disposition proposée des terminaux

Cette disposition vous permettra d'exécuter vos commandes `git` dans le terminal `T1`, tout en exécutant des commandes de *suivi* dans les terminaux `T2` et `T3`. Dans un premier temps, positionnez `T1`, `T2` et `T3` dans le

1. `https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions`
2. `https://github.com/git/git/blob/master/contrib/completion/git-prompt.sh`

répertoire où vous allez travailler via la commande `cd`.

La commande `watch`[3] permet de répéter une commande toute les X secondes et la commande `tree` permet de visualiser toute l'arborescence de dossiers fichiers. Dans le terminal `T2` nous allons surveiller le répertoire `tpgit`, alors exécutez :

```
watch -d -n3 tree -a tpgit
```

Et dans le terminal `T3` exécutez :

```
watch cat ~/.gitconfig
```

Il est normal que ces commandes affichent pour le moment des erreurs car nous n'avons pas encore créé le répertoire `tpgit` ni le fichier `.gitconfig`. N'hésitez pas à réduire la taille de police des terminaux pour mieux afficher les informations plus tard, la sortie des commandes va grandir rapidement.

2.2 Configuration de votre compte `git`

Dans un projet géré par plusieurs, il est important de savoir qui a fait quoi. `git` a donc besoin d'une configuration pour identifier vos futurs *commits*, et vous pouvez également spécifier certains comportements. Les commandes suivantes vont modifier votre fichier `.gitconfig`, tapez les une à une en regardant les changements dans le terminal `T3` :

```
git config --global user.name "votre nom"
git config --global user.email nom.prenom@polytech-lille.net
git config --global push.default simple
git config --global color.decorate full
git config --global merge.conflictstyle diff3
git config --global core.editor 'nano'
```

Si vous préférez un autre éditeur à la place de `nano`, vous pouvez l'utiliser (*e.g.*, `kate -b`, `vim`, `emacs`, `gedit`).

Rappel !
Les configurations de `git` et de `bash` doivent se faire une fois pour chaque compte. Votre binôme devra réaliser la même configuration sur son compte, et si vous comptez travailler sur un autre PC, il faudra récupérer votre fichier `.gitconfig` ou relancer les commandes de configuration.

3 Initialisation d'un dépôt `git`

Nous allons travailler sur un projet qu'on appellera `tpgit`. En vous aidant des supports du cours (*e.g.*, slide 12), vous devez :

- créer un répertoire `tpgit`
- initialiser dans ce répertoire un dépôt git vide
- regarder les fichiers créés pour ce dépôt, ils seront affichés dans le terminal `T2` dans le dossier `.git`
- créer un fichier `fruits.txt`
- réaliser un commit en ajoutant 2 fruits à `fruits.txt`. Pour rappel, l'enchaînement des commandes est : `status, add, status, commit, status, log`. **Lisez la sortie de chaque commande** pour comprendre leur fonctionnement.
- réaliser 3 commits de plus en ajoutant à chaque fois des fruits.

Astuce !
Utilisez les commandes `git status`, `git diff` *`nom_fichier`* et `git log` pour comprendre l'état de votre dépôt. La commande `git log --graph --oneline --decorate --all` vous donne une historique détaillé et globale avec toutes les branches, étiquettes et commits.

Dans le terminal `T3`, arrêter le processus (`ctrl+c`) et exécuter la commande suivante dans le répertoire `tpgit` pour visualiser l'historique du dépôt : `watch git log --graph --oneline --decorate --all` . Si vous voulez avoir une sortie colorée, vous pouvez ajouter l'option `--color` aux commandes `watch` et `git` comme suit : `watch --color git log --graph --oneline --decorate --all --color`

3. `https://en.wikipedia.org/wiki/Watch_(Unix)`

4 Branches `git`

Branches `legumes`
- créer une branche `legumes`, se placer dans cette branche (vérifier avec le prompt ou avec la commande `git branch` que vous êtes bien sur la branche `legumes`)
- ajouter un fichier `legumes.txt` sur cette branche
- réaliser 3 commits différents en ajoutant des légumes à `legumes.txt`
- vérifier l'historique de vos commits à l'aide de `git log`.

Branches `sauces`
- créer une branche `sauces`
- ajouter un fichier `sauces.txt` sur cette branche
- réaliser 2 commits différents
- vérifier l'historique de vos commits et l'existence de **3 branches** (master, legumes, sauces) à l'aide de `git branch` et `git log`.

5 Merges `git`

Vous avez décidé que vos commits dans les branches `legumes` et `sauces` sont pertinents, maintenant vous devez les intégrer dans `master`.
- aller sur la branche `legumes`
- vérifier l'état de votre espace de travail (*e.g.,* l'inexistence du fichier `sauces.txt`)
- merger `master` dans `legumes`
- si tout c'est bien passé, merger `legumes` dans `master`
- vérifier l'historique de vos commits et l'apparition de `legumes.txt` sur la branche `master`

Vous pouvez maintenant merger la branche `sauces` avec `master` en suivant le même processus. Vérifiez que tous les branches sont à jour.

6 Dépôt distant

Nous allons activer vos comptes sur le serveur GitLab de l'Université de Lille (`https://gitlab.univ-lille.fr`) et créer un dépôt vide où vous allez pousser votre dépôt existant `tpgit`. (Vous êtes également libre d'utiliser Github, GitLab.com, ou le serveur GitLab de Polytech `https://archives.plil.fr/`.)
GitLab vous permet de gérer votre projet et fourni plusieurs fonctionnalités intéressantes (*e.g.,* issues, graphes, édition de fichiers, recherche de contributeurs, dashboard).

Attention !
Il y a deux protocoles réseau pour échanger avec GitLab, HTTPS et SSH. Nous vous conseillons de travailler avec HTTPS dans un premier temps. Dans le futur vous pouvez choisir SSH, qui est plus puissant et flexible, mais il faudra configurer vos clés privés et publiques en suivant l'aide `https://gitlab.com/help/ssh/README#generating-a-new-ssh-key-pair`

Créer un dépôt sur GitLab et poussez `tpgit`

1. Se connecter sur `https://gitlab.univ-lille.fr`
2. Cliquer sur *New Projet* en haut pour créer un nouveau projet
3. Donner un nom à votre projet et cliquer sur *Create project* (NE PAS COCHER *Initialize Repository !*)
4. Vous êtes emmenés dans votre projet, qui est vide, avec les instructions des différentes formes d'initialisation. **Lisez les propositions.** Vous devez **impérativement vérifier et changer l'url de `SSH` à `HTTPS`** si vous n'avez pas configuré des clés SSH pour vos connexions.
5. Retrouvez les instructions *Push an existing Git repository*. Vous allez copier et exécuter la commande : `git remote add origin https://gitlab.com/<user>/<project.git>` avec le bon url suivi de `git remote -v` pour vérifier le dépôt, et `git push -u origin master` pour indiquer que vous voulez envoyer la branche actuelle vers la branche `master` du dépôt distant surnommé `origin`.
6. Rafraîchissez la page de votre projet GitLab, votre projet `tpgit` doit s'y retrouver.
7. Naviguez votre projet sur Gitlab pour voir l'historique de commits, vos fichiers, vos graphes, etc.

Gestion des membres de votre projet. Ajouter votre binôme aux membres de votre projet avec, au minimum, le niveau de droit *Developer* (il devra créer son compte et se connecter une première fois avant d'apparaître dans la liste des membres)

1. Aller dans les *Settings* de votre projet (menu de gauche),
2. Cliquer sur *Members* dans le menu *Settings* (menu de gauche),
3. Cliquer sur *New project member* en haut à droite,
4. Chercher le login de la personne que vous voulez ajouter (votre binôme),
5. Sélectionner le niveau de droit de cette personne (*Developer*),
6. Cliquer sur *Add users*.
7. Cliquer sur *Protected branches* dans le menu *Settings* (menu de gauche), *Repository*, puis cliquer sur *Protected branches* et cliquer sur *Unprotect* pour la branche `master`

7 Travail collaboratif

Vous allez travailler en binôme sur le dépôt de binôme 1. Binôme 2 doit donc cloner le projet de binôme 1 dans un dossier nommé tg-git-binôme. Pour cela :

— Binôme 2 doit se connecter sur le projet de binôme 1 dans GitLab
— Cliquer sur HTTPS
— Cloner le projet à l'aide de la commande `git clone <url> tp-git-binome`
— Regardez l'historique du projet.

8 Travail simultané

Vous allez travailler en même temps sur ce projet commun, chacun sur son propre dépôt local (binôme 1 sur son dossier local originel, et binôme 2 sur son nouveau dossier local tp-git-binome), et vous allez synchroniser les changements après.

8.1 Merge sans conflit

Travail de Binôme1

— créer une branche `epices`
— ajouter un fichier `epices.txt` sur cette branche
— réaliser 2 commits différents
— merger vos commits dans `master`
— récupérer les changements sur le serveur (`git pull`)
— pousser vos changements vers le serveur GitLab à l'aide de la commande `git push`

Travail de Binôme2

— créer une branche `herbes`
— ajouter un fichier `herbes.txt` sur cette branche
— réaliser 2 commits différents
— merger votre branche dans `master`
— récupérer les changements sur le serveur (`git pull`)
— pousser vos changements vers le serveur GitLab à l'aide de la commande `git push`

Tous les deux

— vérifier l'existence des fichiers `epices.txt` et `herbes.txt` dans vos deux dépôts
— vérifier l'historique des deux dépôts... est-ce que tous les ID de commits sont les mêmes ? Vous avez trouvé le dernier commit de merge ? Vous voyez les commits de votre binôme ?

Un de vous deux a été le premier à pousser, et l'autre a dû fusionner les changements avant de pouvoir pousser son travail. Parce que vous avez travaillé sur des fichiers différents, le `merge` a été résolu automatiquement. Nous allons maintenant vous obliger à générer des conflits et à les résoudre.

Astuce !
Vous en avez marre de taper votre login et mot de passe à chaque `pull` et `push` ? Git peut se souvenir de votre identité pendant un temps déterminé, par exemple, pour une heure utilisez la commande :
`git config --global credential.helper 'cache --timeout=3600'`

8.2 Merge avec conflit

Générer un conflit

- éditez tous les deux, chacun sur son dépôt, le fichier `fruits.txt`, de façon à **effacer** certains fruits et à **ajouter** des nouveaux (faites des changements différents)
- commitez vos changements mais ne poussez pas encore
- binôme1 pousse ses changements en premier
- binôme2 pull les changements de binôme1 et... CONFLIT !
- `git status`, lisez la sortie
- à l'aide du cours (slides 26-29), essayez de résoudre ce conflit
- une fois résolu, fusionnez vos changements et poussez vers le serveur
- binome1, récupérez les changements et vérifiez l'état de `fruits.txt`

Générer un 2ème conflit, inverser les rôles

- comme précédemment, générer un conflit dans le fichier `legumes.txt` mais cette fois inversez les rôles : binome2 pousse en premier, binome1 résout le conflit.
- au moment du conflit, c'est-à-dire, tout de suite après un `git pull` ou `git merge`, vous pouvez annuler le merge avec la commande `git merge --abort`. Essayez la commande et vérifier que votre dépôt revient dans l'état avant le `pull` (plus de conflits, vérifier avec `status`). Refaite `git pull` et vous allez retrouver le conflit, résolvez-le cette fois.

Astuce !
Pour minimiser les conflits, il faut s'assurer que tout le monde maintienne son dépôt à jour. Il faut régulièrement merger vers la branche `master`, faire `git pull` sur vos branches, et corriger les éventuels petits conflits quand ils sont simples. Avant de pousser vos changements avec `git push`, il faut toujours faire un `git pull`.

9 Documenter votre projet à l'aide de Markdown

Markdown est un langage de balisage léger, plus rapide à écrire que HTML et très flexible. Il est interprété automatiquement par GitLab (et plein d'autres produits) et permet de documenter votre projet ou même d'écrire vos rapports.

Créez un fichier `README.md` à la racine de votre dépôt en utilisant la syntaxe markdown pour spécifier des titres, listes, des extraits de code. Vous devez au minimum :

- donnez une description de votre projet
- listez les auteurs dans une section *Auteurs*
- listez les objectifs dans une section *Objectifs*

Vous pouvez vous inspirer de cet exemple sur Github :
`https://gist.github.com/PurpleBooth/109311bb0361f32d87a2`
(cliquer sur RAW pour avoir les sources). Vérifiez que votre README s'affiche correctement sur GitLab.

10 Eclipse et GIT (Optionnel)

Sous Eclipse, il existe une extension GIT qui présente quelques avantages. La première est de développer et gérer les versions sans quitter Eclipse. La seconde permet de s'abstraire de la syntaxe des commandes en ligne. Des outils graphiques viennent également simplifier la gestion des conflits.
Voici un rapide tutoriel sur quelques commandes de base de cette extension.

1. Sous Eclipse (`/usr/local/eclipseNeon/eclipse &`, créer un projet PHP de nom `"demoPHP"`.
2. **Réalisez l'équivalent de `"git init"`** :
   Sélectionnez le projet et, avec le clic droit de la souris, sélectionnez "Team/Share Project...", sélectionnez "Git" puis cliquez sur `"Next>"`, cliquez sur "Create..." (en haut à droite), sélectionnez un répertoire de nom "git_demoPHP", cliquez sur `"Finish"`, cochez le projet `"demoPHP"` puis cliquez sur `"Finish"`.
3. Créez un fichier `index.html` dans votre projet.
4. **Réalisez l'équivalent de `"git add ... ; git commit"`** :
   Sélectionnez le projet et, avec le clic droit de la souris, sélectionnez "Team/Commit...". Une vue "Git Staging" est alors disponible. Déplacez le fichier `index.html` de la sous-fenêtre "Unstaged Changes" vers la sous-fenêtre "Staged Changes" (équivalent de `git add`), donnez un titre à votre futur commit dans la sous-fenêtre "Commit Message" puis cliquez sur "Commit" (équivalent de `git commit`).
5. Connectez-vous sur `https://gitlab.univ-lille.fr` et créez un projet `"demoPHP"`, mémorisez l'adresse du projet : `https://gitlab.univ-lille.fr/<votre_login>/demoPHP.git`
6. **Réalisez l'équivalent de `"git add ... ; git commit ; git push"`** :
   c'est la même procédure que le point 4 mais il faut cliquer sur "Commit and Push" à la fin. Pour ce premier "push", une fenêtre de dialogue vous demande l'URI, saisissez `https://gitlab.univ-lille.fr/<votre_login>/demoPHP.git`, cliquez sur `"Next>"`, saisissez vos login/password puis cliquez sur `"OK"` et `"Finish"`.
7. **Réalisez l'équivalent de `"git pull"`** :
   Il suffit de sélectionnez "Team/Pull" et suivre les instructions.

A partir du menu "Team", vous disposez de l'équivalent de toutes les commandes `git`.
Pour créez un projet Eclipse à partir d'un projet déjà stocké dans le repository local ou distant, il faut sélectionnez dans le menu Eclipse "File/Import.../Git/Projects from Git" et suivre les inscriptions.

11 Continuer à apprendre

Si vous voulez maîtriser `git`, n'hésitez pas à faire le tutoriel donné par le laboratoire CRIStAL de l'université de Lille. Il y a beaucoup de fonctionnalités avancées qui peuvent s'avérer très utiles. `http://www.cristal.univ-lille.fr/TPGIT/` .

Vous pouvez également consulter les liens suivants :

Liens, aides et outils

- Où stocker vos projets
  - `https://gitlab.univ-lille.fr/`
  - `https://archives.plil.fr/`
  - `https://about.gitlab.com/`
  - `https://github.com/`
  - `https://bitbucket.org/`
  - Votre serveur perso (*e.g.,* installer GitLab chez vous)
- Tutoriels
  - `http://www.cristal.univ-lille.fr/TPGIT/`
  - `https://crypto.stanford.edu/~blynn/gitmagic/intl/fr/book.pdf`
  - `https://learngitbranching.js.org/`
  - `https://try.github.io/`
  - `https://git-scm.com/book/fr/v2`
- Vidéos
  - `https://www.youtube.com/watch?v=OqmSzXDrJBk`
  - `https://www.youtube.com/watch?v=uR6G2v_WsRA`
  - `https://www.youtube.com/watch?v=3a2x1iJFJWc`
  - `https://www.youtube.com/watch?v=1ffBJ4sVUb4`
  - `https://www.youtube.com/watch?v=duqBHik7nRo`