Colloque « Sciences ouvertes »

Le développement collaboratif en recherche avec GitLab

Novembre 2024

Sommaire

Sommaire

Sommaire

Problématiques générales de recherche

- Ouverture de la science
  - mouvement international : recherche et données produites accessibles à tous
  - appuyée par des politiques nationales, européennes
- Au CNRS, s'inscrit dans une démarche plus large (voir `https://www.cnrs.fr/sites/default/files/ressource-file/Pratiquer-une-recherche-integre-et-responsable-2017.pdf`)

- Un lien intéressant pour resituer : `https://barometredelascienceouverte.esr.gouv.fr/`

Concerne qui ? quoi ?

- Concerne déjà de nombreuses disciplines : astronomie, biologie, éducation, informatique, mathématique, médecine, sciences sociales...
- Concerne toutes les données manipulées
  - ***open data*** : corpus, expérimentations, indicateurs, données brutes ou synthétisées diverses (pas qu'en recherche d'ailleurs, voir `https://data.gouv.fr/`)
  - ***open source*** : codes informatiques (idem, voir `https://code.gouv.fr/`)
- Données et code (quand approprié) vont de pair
- En recherche, on parle d'***open science***

Un exemple apparenté récent

(*fondateur de Vitemadose et de CovidTracker*)

Avantages d'une science ouverte

Une science ouverte présente de nombreux atouts

- meilleures collaborations, meilleures participations
- transparence, traçabilité
- reproductibilité
- diffusion + universelle, potentiellement tous niveaux de sociétés
- meilleure réactivité
- meilleur facteur d'impact au niveau scientifique (contraintes de soumission parfois, adoption favorisée ensuite)

Quid du stockage et des manipulations ?

- Ok, je suis convaincu mais... je fais quoi ? comment ?
- Nécessité de plusieurs briques et besoins de *standards*
  - sur le stockage de données
  - sur la manipulation des données (qui ? comment ? quels droits ?)
  - sur le code informatique
  - et encore : sur la sauvegarde, sur le *versionning*...
- Les données de la science ouverte s'appuient sur des solutions, outils et protocoles éprouvés, universels, interopérables

Panorama rapide des solutions de stockage

Principales solutions collaboratives

	Stockage données	**Usage type**	**Gestion des droits**	**Sauvegarde et reprise**	**Versionning**
Serveur FTP	fichiers	*data*	RW	à gérer côté serveur	non
Bases de données	données structurées	*data*	RWA très fin	à gérer côté serveur	oui, mais pas nativement
Drive / Cloud	fichiers	*data*	RW	peuvent être natives et multiples	propre à chaque solution
Gest. versions	fichiers et branches	code	RWA + workflows	natives et multiples	natif

(*R : lecture, W : écriture, A : administration,*

branche : ensemble de fichiers dont l'état — la version — est cohérent globalement)

Sommaire

Problématiques générales liées au code informatique

- Comment gérer l' historique des fichiers d'un projet ?
  - archivage
  - comparaison de la version courante par rapport à une ancienne
  - récupération d'une ancienne version
- Comment gérer les différentes versions d'un projet ?
  - version 1, version 2... : une version est un ensemble de fichiers dans un état donné
  - développements parallèles : version stable, alternatives

Problématiques spécifiques au travail en groupe

- Comment partager du code ?
- Comment travailler à plusieurs sur du code ?
- Comment travailler au même moment sur du code ?
- Comment réconcilier les changements de contributeurs ?
- Comment ne pas perdre de travail ?

Une solution : les systèmes de gestion de version (VCS)

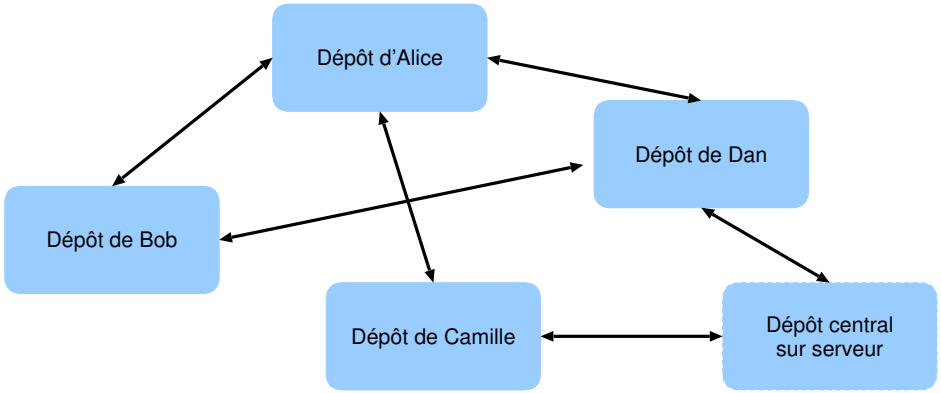
Ensemble de méthodes et d'outils qui maintiennent les différentes versions d'un projet à travers tous les fichiers qui le composent

- permet le développement collaboratif et simultané
- permet de garder tout l'historique de tous les fichiers
- permet le développement parallèle : version principale, versions alternatives
- permet de savoir pourquoi, quand et par qui une modification a été introduite

Git

- Créé en 2005 par Linus Torvalds pour la gestion des sources de Linux, dont il est également le créateur
- Usage intensif depuis pour le développement logiciel
- Popularisé par des sites et outils comme Github, Bitbucket, Gitlab...
- Très performant sur la gestion des codes informatiques, mais également utilisé pour des données *binaires* (images, PDF, documents « office », *etc*.)
- Juste sur Github (source : *Github blog*, 2024)
  - $\approx$ 100 millions d'utilisateurs
  - $\approx$ 500 millions de dépôts

Principe général de fonctionnement



- Chaque utilisateur possède un *dépôt* local complet, contenant tout l'historique du projet
- Les opérations sont réalisées localement (*i.e. offline*)
- Côté synchronisation
  - des serveurs *peuvent* assurer les échanges de données
  - des données *peuvent* aussi être échangées avec d'autres utilisateurs sans serveur centralisé

Sommaire

Sommaire

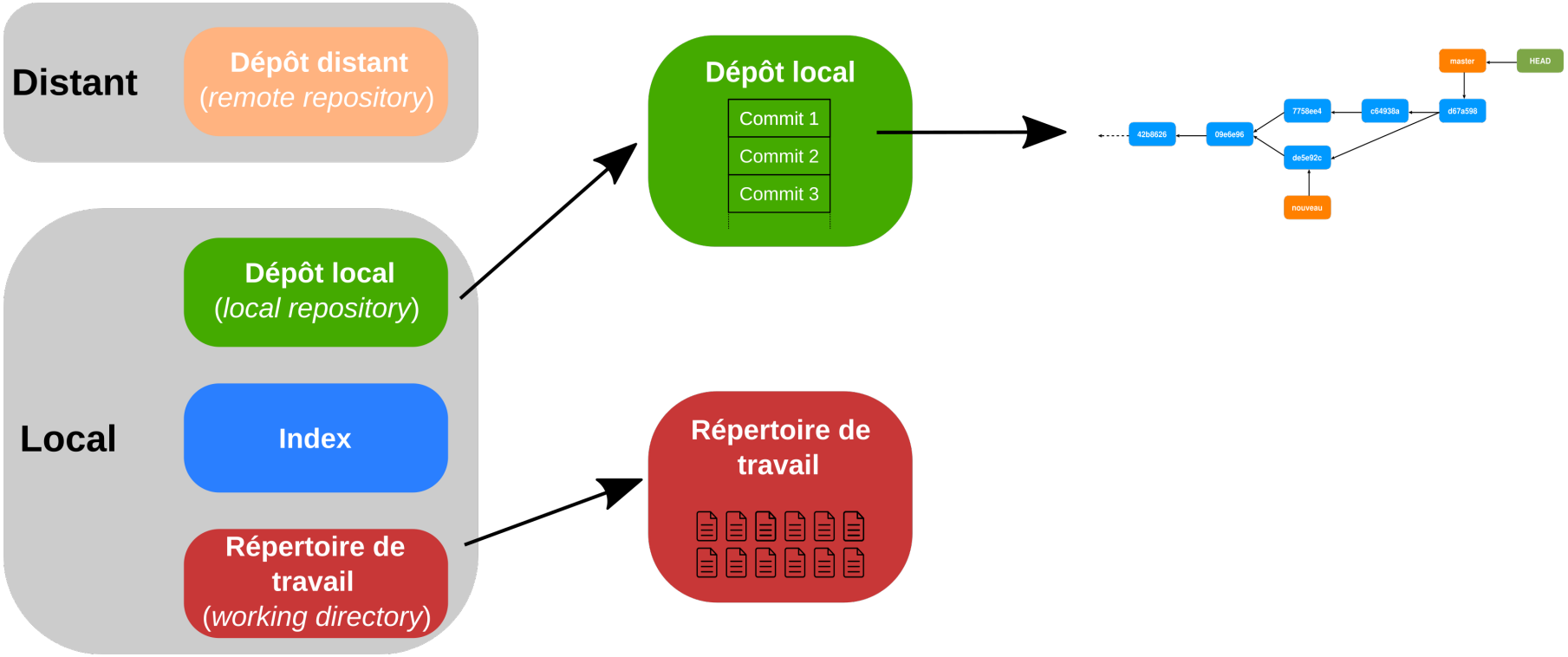
Possibilités et principes

- Travaille principalement sur les fichiers texte (`.txt`, `.c`, `.py`, `.html`, `.xml`...)
- Les fichiers binaires (`.jpg`, `.doc`, `.pdf`...) peuvent également être intégrés mais ne peuvent prétendre qu'au versionage, pas à l'édition collaborative
- Que faut-il stocker dans un dépôt ?
  - **toutes** les ressources nécessaires à la construction d'un projet...
  - ...**à l'exception** de celles qui sont générées automatiquement (`.o` en C, `.class` en Java...) et des données **sensibles**

Principe d'utilisation d'un dépôt (*repository*)

- Création d'un dépôt vide ou clonage d'un existant
- Alimentation du dépôt par l'intermédiaire de *commits*
- Un *commit* contient
  - un **ensemble de modifications de données**, suite aux manipulations des fichiers du projet (création, édition, suppression, renommage...)
  - un ***log*** associé : commentaire sur la nature des modifications
  - des **méta-informations** : identifiant de *commit*, auteur, date

Différents niveaux de stockage



Différents niveaux de stockage

- *Répertoire de travail*
  $\rightarrow$ contient la copie locale des sources du projet
- *Index*
  $\rightarrow$ espace temporaire utilisé pour préparer la transition de données entre le répertoire de travail et le dépôt local
- *Dépôt local*
  - contient la totalité de toutes les versions de tous les fichiers du projet, par l'intermédiaire des *commits*
  - chaque contributeur possède son propre dépôt local
- *Dépôt(s) distant(s)*
  $\rightarrow$ dépôt centralisé ou local d'un autre utilisateur, utilisé pour la collaboration

Sommaire

Les commandes de `git`

- `git` est un *toolkit*
- `git`, c'est plus de 150 commandes différentes
  - $\approx$ une cinquantaine de haut-niveau (*porcelaine*)
  - $\approx$ une centaine de bas-niveau (*plomberie*)
- En pratique, une trentaine de commandes suffisent à un usage « classique » (**une quinzaine au quotidien**)
- Elles sont utilisables en ligne de commande, mais également interfacées à la quasi-totalité des IDE
- Cela autorise de nombreux *workflows* d'utilisation, même si seulement 2 ou 3 sont très populaires (et suffisants pour la plupart des usages), comme les *pull requests*

Principales commandes, par thème

Création

```
git init
git clone
```

Ajout

```
git add
git commit
```

Interrogation

```
git log
git show
git status
git diff
```

Opérations

```
git restore
git mv
git rm
git tag
```

Synchronisation

```
git push
git pull
```

Sommaire

Démonstration

Sommaire

Introduction

- De nombreuses solutions d'hébergement de dépôts existent (les *forges*)
- Elles procurent des accès à des dépôts distants centralisés, toujours disponibles, même si on peut travailler en P2P avec `git`...
- Les plus populaires sont Github, Bitbucket, Framagit...
- Avec Gitlab, il est possible d'auto-héberger sa propre *forge*
- L'Université de Lorraine a déployé une instance Gitlab

Présentation (rapide) de Gitlab

- Gitlab : logiciel libre de *forge* basé sur `git`
- Propose des outils : dépôts `git`, mais aussi wiki, gestion des bugs, documentation, CI/CD...
- Utilisé par de nombreux laboratoires et universités, comme par exemple
  - Université de Lorraine
  - CNRS
  - INRIA
- Nombre d'utilisateurs estimé à 30 millions, déployé par environ 100.000 organisations
  (source : `https://about.gitlab.com/company/`)

Présentation du Gitlab UL

- Doc : `https://numerique.univ-lorraine.fr/catalogue-des-services/gitlab-forge-git`
- Lien : `https://gitlab.univ-lorraine.fr/`
- Lancé il y a quelques années, en phase de « rodage », pleinement fonctionnel (montée en puissance planifiée)
- Accessible à tout membre (personnel, étudiant) de l'université (interfacé à l'annuaire LDAP de l'UL)
- Possibilité d'inviter des extérieurs (création d'*invités numériques*, accessible à tout personnel UL)

Caractéristiques du Gitlab UL

- Pour les membres (personnels, étudiants)
  - limite actuelle à 5 dépôts
  - quota de 10 Go par dépôt
- Possibilité de créer un *groupe de projets*, **solution conseillée** (composante / projet / laboratoire)
  $\rightarrow$ plus de limite sur le nombre de projets
- Authentification
  - connexion SSH non supportée pour le moment (en attente de la nouvelle infrastructure prévue)...
  - ...mais support des *tokens*, solution à la fois plus fine et plus souple (propres à un site, facilement révocables)
  - voir *Préférences / Access tokens*

Sommaire

Conclusion

- `git` est un outil universel de gestion de code, un standard de fait
- Le Gitlab de l'UL est la forge naturelle pour les projets UL
- Nombreuses ressources pour se former
  - site officiel : `https://git-scm.com/`
  - par ailleurs : vidéos, documentations, sites de type *serious games*
- Et ensuite ? Quelques scénarios typiques
  - **création** de projet / dépôt : seul, en groupe
  - **participation** à un projet existant : contributions, *pull requests*
  - **bifurcation** d'un projet existant : clonage d'un dépôt existant pour lui donner une nouvelle orientation (*fork*)
- **Questions ?**