Introduction générale aux forges logicielles. Gitlab : tour d'horizon et prise en main

Pierre-Antoine Bouttier - Franck Pérignon

Journée Gitlab, 29 juin 2023

De quoi va t'on parler ce matin ?

Forges et gestionnaires de développements collaboratifs

Des plateformes dont le but est de faciliter le travail collaboratif, la gestion de projets de développements (au sens large).

Quelles utilisations ?

Collaborations autour du développements de codes (logiciels, applications)

- Usage connu/standard, à l'origine du développement des forges.
- Aujourd'hui : un outil indispensable et largement utilisé par la communauté des développeurs de logiciels

Quelles utilisations ?

Mais une forge vous sera utile dans de nombreuses autres circonstances

- Rédaction collaborative (ou pas) d'articles scientifiques, de manuscrit de thèse, ...

Quelles utilisations ?

Mais une forge vous sera utile dans de nombreuses autres circonstances

- Rédaction collaborative (ou pas) d'articles scientifiques, de manuscrit de thèse, ...
- Enseignement (cours, matériel pour les travaux pratiques, ...)

Quelles utilisations ?

Mais une forge vous sera utile dans de nombreuses autres circonstances

- Rédaction collaborative (ou pas) d'articles scientifiques, de manuscrit de thèse, ...
- Enseignement (cours, matériel pour les travaux pratiques, ...)
- Génération, publication et maintenance de sites web

Quelles utilisations ?

Mais une forge vous sera utile dans de nombreuses autres circonstances

- Rédaction collaborative (ou pas) d'articles scientifiques, de manuscrit de thèse, ...
- Enseignement (cours, matériel pour les travaux pratiques, ...)
- Génération, publication et maintenance de sites web
- Espace de partage de fichiers, de données
  ⚠ une forge n'est pas
  - un entrepôt de données type https://recherche.data.gouv.fr/fr
  - un NextCloud ou DropBox
- etc.

Quelle(s) forge(s) utiliser ?

La référence  GitHub $\approx$ 73 millions d'utilisateurs en 2021, 100 millions de projets, rachetée par Microsoft en 2019.

Alternatives équivalentes :

Bitbucket
https://bitbucket.org

GitLab
https://about.gitlab.com

Quelle(s) forge(s) utiliser ?

Forges externes à l'ESR, commerciales ou communautaires

- très fonctionnelles,
- intègrent de nombreux outils,
- peu de contraintes,
- très utilisées, stables, très régulièrement mises à jour et améliorées.

Quelle(s) forge(s) utiliser ?

Forges externes à l'ESR, commerciales ou communautaires

😀

- très fonctionnelles,
- intègrent de nombreux outils,
- peu de contraintes,
- très utilisées, stables, très régulièrement mises à jour et améliorées.

😦

- commerciales, non académiques
- hébergement non contrôlé (respect RGPD, réglementation européenne ?)
- parfois payantes 💰
- durée de disponibilité indéterminée (Google code ...) ?

Quelle(s) forge(s) utiliser ?

Une bonne solution : les plateformes "ESR" auto-hébergées
(39 forges disponibles[1] dans les structures ou labos)

En pratique

- Le choix dépendra de votre projet, des gens impliqués, des habitudes de votre communauté ...
- Possible et relativement simple d'utiliser plusieurs plateformes
  - outils et utilisations semblables,
  - transferts de projets simples.

[1] Forges de l'Enseignement supérieur et de la Recherche - Définition, usages, limitations rencontrées et analyse des besoins, D. Le Berre, J.Y. Jeannas, R. Di Cosmo, F. Pellegrini, 2023, hal-04098702v2

Aujourd'hui : focus sur GitLab

- Un logiciel développé par la compagnie Gitlab Inc.
- Une distribution "open source" gratuite (Gitlab CE) et une version propriétaire payante (Gitlab EE) proposant des fonctionnalités supplémentaires.
- De très nombreux déploiements (des site webs !) : des forges auto-hébergées
  - *https://about.gitlab.com/* : la forge de Gitlab inc
  - 37 instances Gitlab répertoriées dans l'ESR

Nos objectifs

À la fin de cette journée, nous aimerions que chacun

- ait compris ce qu'est (ou n'est pas) Gitlab,
- soit convaincu de son utilité pour son activité quotidienne,
- soit autonome pour un usage standard/basique,
- soit au fait de quelques bonnes pratiques,
- ait résisté jusqu'au bout de cette présentation 😀

Avertissements

- Pour un public novice !
- Pas (uniquement) pour les développeurs/contributeurs de logiciels.
- Notre pratique, notre expérience
- 1h30, c'est court ...
- Session interactive : n'hésitez pas à nous interrompre !

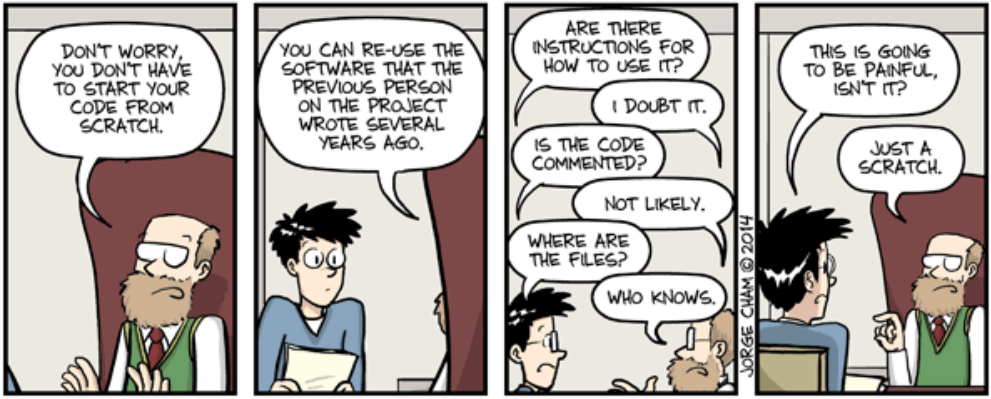
Tous les participants présents ce matin sont inscrits dans ce groupe :
https://gricad-gitlab.univ-grenoble-alpes.fr/gtdonnees-gitlab2023/sandbox

Organisation

- Qu'est-ce qu'une forge, comment ça marche, à quoi ça sert, ... ?
- Prise en main d'une plate-forme Gitlab
- Gestion de version à travers Gitlab
- Gitlab, quelques autres fonctionnalités
- Intégration continue (cet après-midi)

Les présentations sont disponibles ici :
https://gtdonnees-gitlab2023.gricad-pages.univ-grenoble-alpes.fr/intro-gitlab

Ça, c'était avant ...



Aujourd'hui, les Forges ! Systèmes de gestion de projets de développements ***collaboratifs***

Principe :

- Rassembler des utilisateurs de profils différents (chercheurs, ingénieurs, développeurs, coordonnateurs, ...) autour de projets.
- Mettre à disposition un ensemble d'outils adaptés pour développer, gérer, suivre, diffuser, valoriser, les projets.

Aujourd'hui, les Forges ! Systèmes de gestion de projets de développements ***collaboratifs***

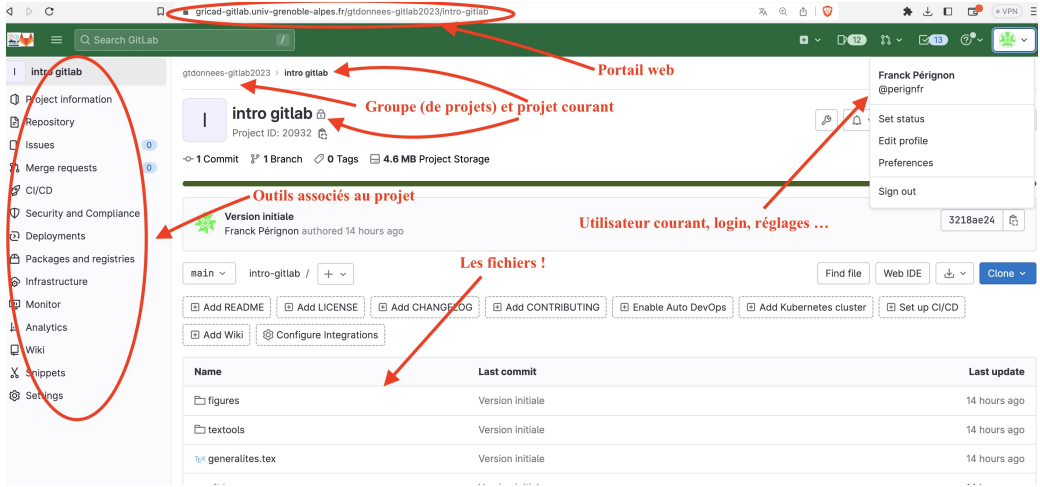
Principe :

- Rassembler des utilisateurs de profils différents (chercheurs, ingénieurs, développeurs, coordonnateurs, ...) autour de projets.
- Mettre à disposition un ensemble d'outils adaptés pour développer, gérer, suivre, diffuser, valoriser, les projets.

En pratique :

- un site web,
- des utilisateurs (login) associés à des projets,
- des outils configurables via l'interface web,
- différents niveaux de droits sur les outils et les projets (visibilité, écriture ...).

Exemple : gitlab ESR Grenoble



Revue (non exhaustive) des outils et fonctionnalités disponibles

- Structuration : groupes, projets et utilisateurs authentifiés
  - collaborations possibles avec n'importe qui
  - gestion fine des droits d'accès et d'utilisation de tous les outils
  - gestion de la visibilité des groupes, des projets, des fichiers.

Revue (non exhaustive) des outils et fonctionnalités disponibles

- Structuration : groupes, projets et utilisateurs authentifiés
  - collaborations possibles avec n'importe qui
  - gestion fine des droits d'accès et d'utilisation de tous les outils
  - gestion de la visibilité des groupes, des projets, des fichiers.
- Planification, ordonnancement, gestionnaire de tâche et de problèmes (issues)
  - Constituer des **communautés** (contributeurs, utilisateurs, étudiants ...)
  - **Suivre**, organiser le travail de l'équipe
  - **Collecter**, consigner et suivre les problèmes, les défauts du produit
  - **Gérer** les demandes et les réponses d'utilisateurs, tickets
  - **Partager** de l'information, espace d'échange, wiki

Revue (non exhaustive) des outils et fonctionnalités disponibles

- Gestion de versions

  Pratique consistant à conserver/maintenir toutes les versions d'un ensemble de fichiers.

  - Centraliser l'activité
  - Archiver et suivre les modifications, conserver l'historique, les évolutions
  - Permettre/faciliter/gérer les contributions (merge-requests ...)

Revue (non exhaustive) des outils et fonctionnalités disponibles

- Gestion de versions

  Pratique consistant à conserver/maintenir toutes les versions d'un ensemble de fichiers.

  - Centraliser l'activité
  - Archiver et suivre les modifications, conserver l'historique, les évolutions
  - Permettre/faciliter/gérer les contributions (merge-requests ...)
- Partage, gestion , de documents et fichiers
  - **Sauvegarde** (et donc restauration possible), archivage
  - **Edition en ligne**

Revue (non exhaustive) des outils et fonctionnalités disponibles

- CI/CD Intégration et déploiement continu
  Pratique consistant à vérifier systématiquement et automatiquement l'impact de toute modification des sources

Revue (non exhaustive) des outils et fonctionnalités disponibles

- CI/CD Intégration et déploiement continu

  Pratique consistant à vérifier systématiquement et automatiquement l'impact de toute modification des sources

  - Permet de créer du logiciel à partir de son code source (compilation, tests automatiques, assurance qualité, diffusion des livrables)
  - Production de versions diffusables, utilisables
  - Reproductibilité
  - Génération, stockage et fourniture d'images (code prêt à l'emploi, notebooks, ...), "container registries"

Revue (non exhaustive) des outils et fonctionnalités disponibles

- Gitlab Pages : publication, hébergement et maintenance de sites web
  - création et déploiement de sites web
  - production de documentation,
  - lié à la CI

Revue (non exhaustive) des outils et fonctionnalités disponibles

- Gitlab Pages : publication, hébergement et maintenance de sites web
  - création et déploiement de sites web
  - production de documentation,
  - lié à la CI
- Interaction avec Software Heritage. À suivre : présentation de B. Chauvet

Pour plus de détails voir *https://about.gitlab.com/features/*
Et les présentations de l'après-midi !

Pourquoi utiliser une forge?

- un accès à tous les outils via un seul portail web ;
- des outils bien intégrés, configurables pour chaque projet, une utilisation relativement intuitive ;
- ne nécessite aucune installation préalable ;
- un accès multi-sites, multi-utilisateurs ;
- un outil adapté à
  - des degrés de participation variés : développement, encadrement, tests, diffusion ...
  - des contextes d'utilisation très différents : recherche, collaboration industrielle, enseignement, ...

Pourquoi utiliser une forge?

Collaboration "interne" au projet, travail collectif aisé

- Espace de travail (git ...) et d'échanges.
- Statistiques, activité, état d'avancement du projet.
- Collaboration inter-organismes
- Pérennisation du projet (sauvegardes, doc ...)

Pourquoi utiliser une forge?

Collaboration "interne" au projet, travail collectif aisé

- Espace de travail (git ...) et d'échanges.
- Statistiques, activité, état d'avancement du projet.
- Collaboration inter-organismes
- Pérennisation du projet (sauvegardes, doc ...)

Visibilité, diffusion du projet.

- Interface de contacts et d'échanges, communauté (support en ligne, documentation, suivi ...).
- Mise à dispositon de documents, de code, d'outils ...
- Vue sur l'activité, la maturité du projet.
- Pour une structure : exposer l'ensemble de la production du site

Adopter de bonnes pratiques et aller vers la science ouverte

- Favoriser le travail collaboratif, faciliter la vie de tout le monde, collaborateurs ET utilisateurs (présents et futurs)
- Fournir et valoriser des produits de qualité, partageables et réutialisables
- Être dans une démarche de "recherche reproductible"

  👍 Réseau Grenoblois autour de la Recherche Reproductible : *https://reproducibility.gricad-pages.univ-grenoble-alpes.fr/web/*

Les forges sont au coeur de ces processus

Un peu de lecture

Plan d'action du CNRS pour la valorisation des logiciels de recherche - 4 plaquettes :

- Je code : les bonnes pratiques de développement logiciel, https://hal.archives-ouvertes.fr/hal-02083801v1
- Je code : quels sont mes droits ? Quelles sont mes obligations ?, https://hal.archives-ouvertes.fr/hal-02399517
- Je code : les bonnes pratiques en matière de diffusion, https://hal.archives-ouvertes.fr/hal-02400300
- Je code : les bonnes pratiques en écoconception de service numérique à destination des développeurs de logiciels, https://hal.archives-ouvertes.fr/hal-03009741/document

Pour les démos d'aujourd'hui, nous utiliserons gricad-gitlab

- Plateforme créée en 2017
- commune à tous les établissements du périmètre grenoblois liés à l'enseignement supérieur et à la recherche,
- accessible via les identifiants "université Grenoble" ou en compte externe
- aujourd'hui : près de 9000 utilisateurs, plus de 14000 projets, plus de 2500 groupes.

https://gricad-gitlab.univ-grenoble-alpes.fr.

Premiers pas avec gricad-gitlab

Les bases pour prendre en main une plate-forme de type gitlab

- création/gestion d'un compte utilisateur,
- création/gestion de projets et groupes,
- visibilité et droits.

Pré-requis ?

- Un navigateur web, une connexion
- Et c'est tout !

Accès et connexion à gricad-gitlab

Qui : n'importe qui !
Comment ?

- ESR Grenoble : compte *agalan* ⇒ accès complet à tous les outils.
  Création/Connexion : onglet **LDAP UGA**
- comptes 'externes', plus limités ⇒ création de groupes ou projets personnels non autorisée.
  Création d'un compte : lien **Register**.
  Connexion : onglet **Standard**.

Identification : un username et un email unique (impossible d'avoir 2 comptes avec le même email).

Accès et connexion à gricad-gitlab

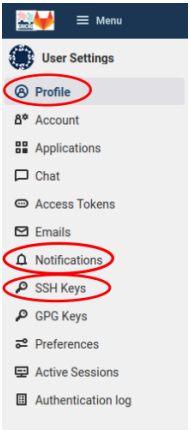
Page d'accès : *https://gricad-gitlab.univ-grenoble-alpes.fr*

👉 faites le test, connectez vous !

Tableau de bord

- Configuration du compte.
- Accès aux projets et groupes.
- todo-list, notifications ...

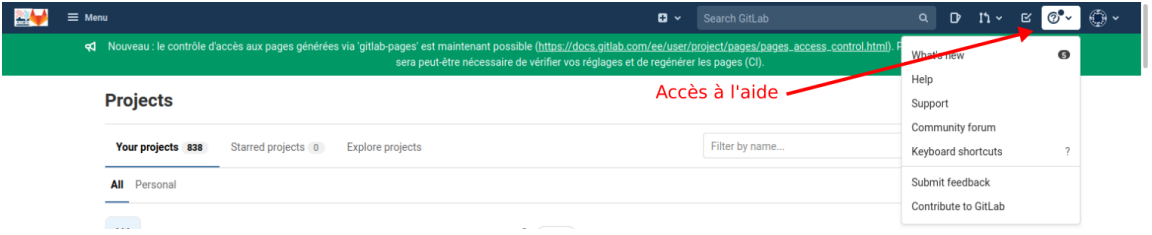
Gestion du compte, le menu Settings



Premières étapes *indispensables* :

- **Profile** : vérifiez les différents champs.
- **SSH keys** : déposez une clé ssh (*non traité aujourd'hui*).
- **Notifications** : contrôle du niveau de notifications
  Visualisez les différentes possibilités et fixez un mode par défaut.
  (👍 "disabled" pour éviter l'envoi intempestif d'emails)

Où trouver de l'aide ?



Groupes et de projets

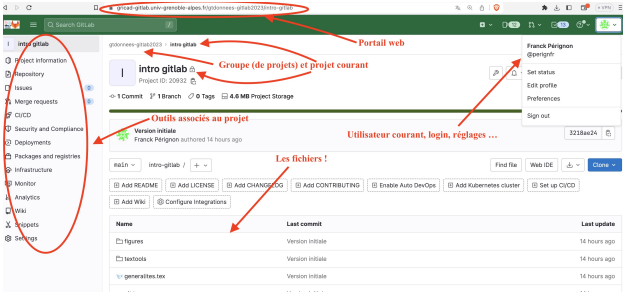
👉 un sous-groupe **sandbox** dans le groupe **gtdonnees-gitlab2023**
Expérimentez, créez des groupes et des projets dans sandbox, faites n'importe quoi !

https://gricad-gitlab.univ-grenoble-alpes.fr/gtdonnees-gitlab2023/sandbox

Projets

un “espace” pour

- héberger, sauvegarder, partager des fichiers,
- gérer ensemble de participants, avec des droits réglables individuellement,
- gérer /configurer des outils



Création d'un projet

Menu *Projects* du tableau de bord : liste de vos projets et onglet *New project*

Création d'un projet

Les informations nécessaires à la création :

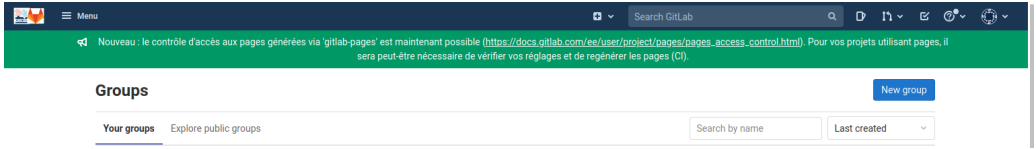
- un emplacement (**namespace** $\approx$ dossier), un nom, une description
- un niveau de **visibilité**,
  - privé : visible uniquement par les membres du projet (qui devront donc être ajoutés par l'admin)
  - interne : visible par tout utilisateur connecté (possibilité de demander à rejoindre le projet)
  - public : visible par n'importe qui

Groupes

- Ensemble de projets ou de sous-groupes,
- associés à un ensemble d'utilisateurs
- des droits/autorisations par défaut

Création d'un groupe

Menu *Groups* du tableau de bord : accès à la liste de vos groupes et onglet *New group*



Informations nécessaires à la création :

- un nom et une description
- un niveau de **visibilité**, comme pour les projets.

⚠ les projets héritent de la visibilité du groupe : par exemple, un groupe privé ne peut contenir que des projets privés.

Groupes et projets

- Un projet appartient nécessairement à un **namespace** ($\approx$ dossier).
  - Votre '**username**', projets personnels (manuscrit de thèse, forks ...), (**non-disponible pour les comptes externes**).

    https://gricad-gitlab.univ-grenoble-alpes.fr/ votre_login/nom_du_projet
  - Un **groupe** ou un **sous-groupe**.

    https://gricad-gitlab.univ-grenoble-alpes.fr/ nom_du_groupe/nom_du_projet

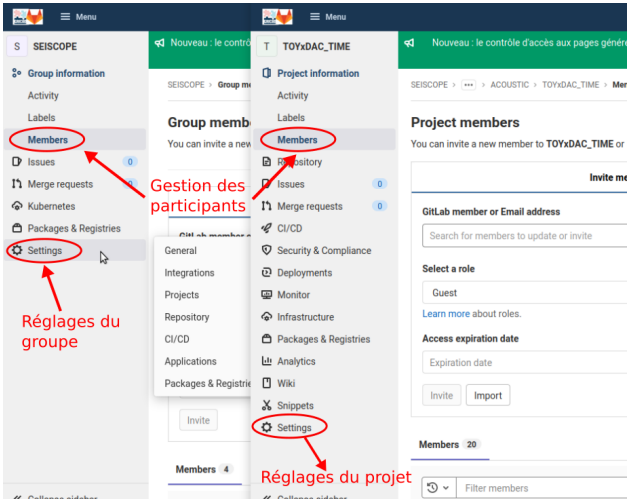
Rôles et permissions

Les rôles : guest, reporter, developper, maintainer, owner

Détails :
https://gricad-gitlab.univ-grenoble-alpes.fr/help/user/permissions

- Créateur d'un groupe ou projet : automatiquement "owner"
- ⚠ un utilisateur hérite dans un projet de ses droits sur le groupe
- Un utilisateur peut faire partie d'un projet sans appartenir au groupe.

Configuration des groupes et projets



- Accès limité (rôle owner/maintainer)
- Deux niveaux de réglage différents : groupe et projet
- Renommage, déplacement, suppression ...
- Configuration des différents outils
- Ajout/suppression de participants et gestion de leurs rôles

Groupes et projets, compléments

- Groupe gitlab = organization GitHub
- Rejoindre un projet ou un groupe existant : lien “request access” (si activé par admin projet)
- import de projet possible (exemple : github vers gitlab).

Groupes et projets, bonnes pratiques

- **Organisez les projets dans des groupes**/sous-groupes thématiques (labo, thèmes de recherche ...)
- Réfléchissez au **nommage des projets**

  $\Rightarrow$ impact sur l'affichage/la visibilité du projet (pages web, références etc).

Groupes et projets, bonnes pratiques

- ⚠ **Prenez le temps de configurer** correctement la liste des membres, leurs droits, les rôles, dates d'expiration de participation etc.
- ⚠ aux membres d'un groupe : n'ajoutez que les personnes susceptibles de participer à tous les projets du groupe.

Pourquoi utiliser un gestionnaire de version?

Un projet, des participants qui collaborent ...

Besoins/problèmes potentiels :

- travail simultané sur les mêmes fichiers,
- besoin d'intégrer les modifications d'autres personnes
- besoin de distribuer aux utilisateurs une version à jour des fichiers
- besoin de conserver l'historique (pour d'éventuels retours en arrière)

Que faire?

- **Beaucoup de mauvaises solutions :** interdire aux développeurs de travailler sur les mêmes fichiers,
  fusionner/synchroniser les fichiers à la main,
  trouver quelqu'un qui accepte de coordonner tout ça ...

Que faire?

- **Beaucoup de mauvaises solutions :** interdire aux développeurs de travailler sur les mêmes fichiers,
  fusionner/synchroniser les fichiers à la main,
  trouver quelqu'un qui accepte de coordonner tout ça ...
- **La bonne solution :** utiliser un gestionnaire de version.

  Un outil qui est au coeur des forges, et qui en est le principal intérêt !

Projet Gitlab $\approx$

des répertoires et des fichiers (**repository**)
+
un gestionnaire de version *intégré*.

Gestion de version

Pratique qui consiste à conserver/maintenir toutes les versions d'un ensemble de fichiers.

Deux fonctions principales :

- permet le travail simultané de plusieurs personnes sur un ensemble de fichiers :
  - synchronisation des modifications, fusion automatique des fichiers,
  - détection et résolution (plus ou moins automatique ...) des conflits,
- une gestion de l'historique du projet :
  - accès à n'importe quelle version archivée,
  - information sur qui a fait une modification, quand, où, pourquoi ...
  - notifications automatiques aux participants

Bref, un outil indispensable.

Quelques notions de base

- **Repository/dépôt** : les données et méta-données concernant l'ensemble des répertoires et fichiers "versionnés".
  $\Rightarrow$ **une référence.**
- Des **utilisateurs** i.e. des personnes référencées susceptibles de lire/modifier le dépôt.
- **Commit**/Révision : enregistrement dans le repository d'un ensemble de modifications,
  par extension, désigne un état d'un repository, une version,
  un instantanée de l'ensemble des fichiers.

Quelques exemples de logiciels de gestion de versions :

- cvs (1990) : *http://cvs.nongnu.org*
- subversion (svn, 2000) : *http://subversion.apache.org/*
- git (2005): *http://git-scm.com/*
- mercurial (2005): *http://mercurial.selenic.com/*
- bazaar (2005): *http://bazaar.canonical.com/en/*

Tous sont libres et en général disponibles facilement.

Quel gestionnaire utiliser ?

Git !

- le plus utilisé (https://rhodecode.com/insights/version-control-systems-2016)
- difficile d'ignorer git,
- outil intégré à Gitlab.

La suite ...

- Git : principes de fonctionnement
- Démo : utilisation de Git via Gitlab
- Git en ligne de commande : euh non, pas le temps voir https://gtdonnees-gitlab2023.gricad-pages.univ-grenoble-alpes.fr/intro_gitlab

Git, principes de fonctionnement

Empilement (dans le temps) de versions dans une base de données.

Chaque version d'un repository est un **instantané (snapshot)** de l'état de l'ensemble des fichiers, sauvé dans une base de donnée.

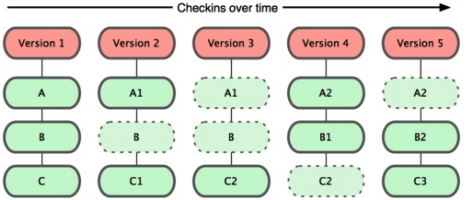


Figure: source, git-scm.com

Nouvelle version :

- copie de l'état pour les fichiers nouveaux ou changés,
- référence ($\approx$ lien) vers les fichiers inchangés.

Git, les différents états d'un fichier

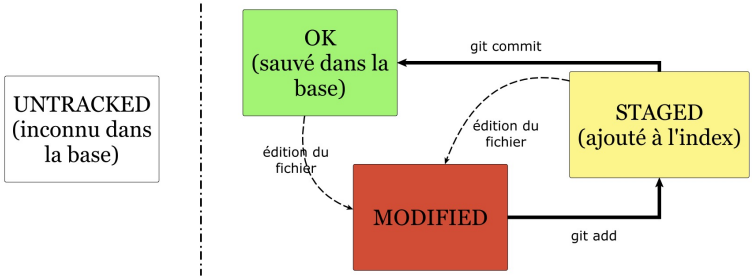
Le cycle simplifié de vie des fichiers d'un projet :

1. extraction de la base de donnée d'un "instantané" du projet
   $\Rightarrow$ une version de l'ensemble des fichiers : le répertoire de travail.
2. modification(s)/éditions des fichiers du répertoire de travail,
3. indexage des modifications $\approx$ liste de ce qui a vocation à être versionné, candidats au commit
4. validation (commit) : création d'un nouvel instantané, sauvegardé dans la base de données

Git, les différents états d'un fichier

Repository/working directory : extraction d'une version de l'ensemble des fichiers

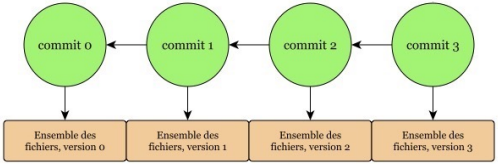
Du point de vue de git, les fichiers de ce répertoire passent par différents états :



Zone d'index (staging) : stockage des informations concernant ce qui fera partie de la prochaine version.

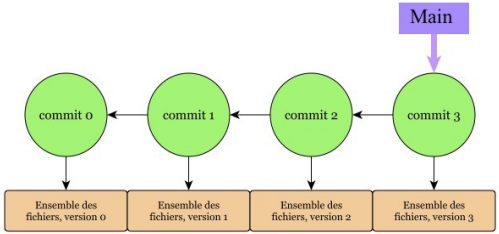
Git, les branches

commit : un lien (un pointeur) vers un instantanée (snapshot) de l'état de l'ensemble des fichiers. Git empile les commits au fil des validations pour construire l'historique du projet.



Git, les branches

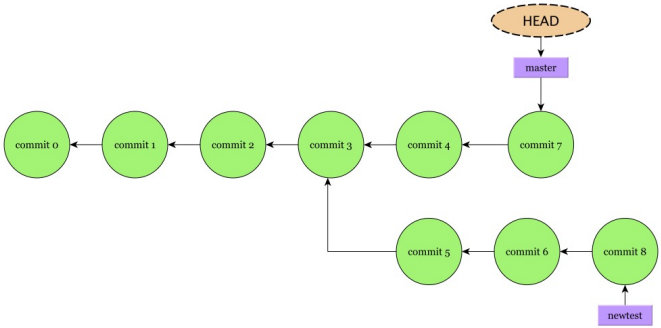
commit : un lien (un pointeur) vers un instantanée (snapshot) de l'état de l'ensemble des fichiers. Git empile les commits au fil des validations pour construire l'historique du projet.



Branche : un pointeur vers un commit particulier + historique
Une branche par défaut : main.
Extraction possible de n'importe quel commit de la branche ⇒ accès à la version des fichiers de ce commit

Git, les branches

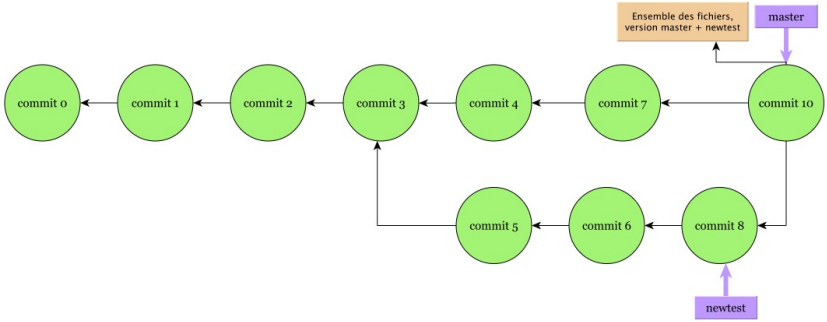
Nouvelle branche: divergence par rapport à la ligne principale (nouvelle fonctionnalité, correction de bug ...)



Identification de la branche en cours : le pointeur HEAD.

Git, les branches

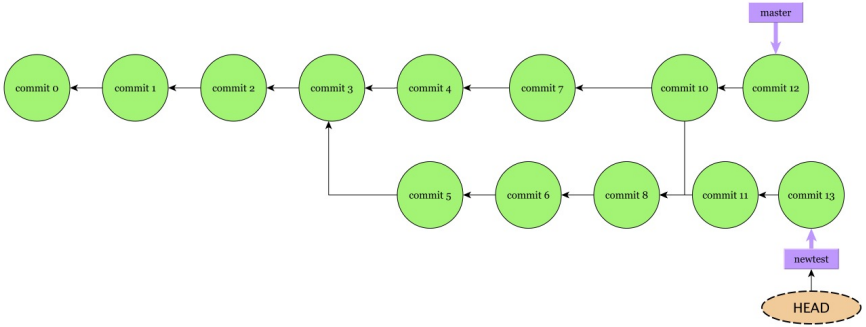
Nouvelle branche: divergence par rapport à la ligne principale (nouvelle fonctionnalité, correction de bug ...)



Fusion ...

Git, les branches

Nouvelle branche: divergence par rapport à la ligne principale (nouvelle fonctionnalité, correction de bug ...)



Evolution en parallèle ...

Git : principes de fonctionnement

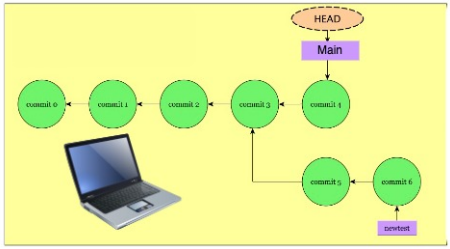
Git: un gestionnaire de version **décentralisé**

- pas de notion de dépôt/serveur central de référence,
- coexistence possible de plusieurs dépôts (locaux, distants ...), désynchronisés, indépendants,

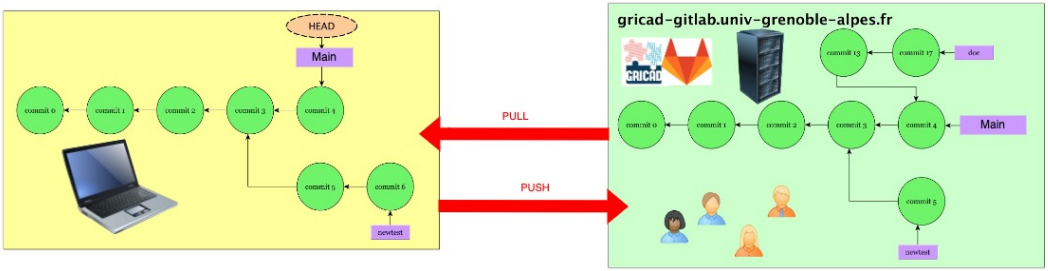
Deux "niveaux" de travail :

- **local** à chaque "repository" avec les fonctionnalités classiques d'un gestionnaire de version,
- **distant** : avec la possibilité de synchroniser votre repository avec d'autres dépôts, dits remote/distants.

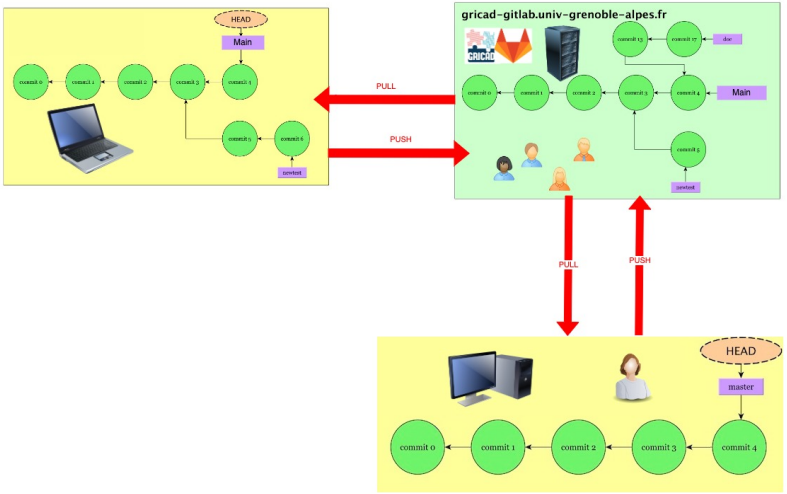
Un repository “local”



Un repository “local” lié à un “remote” sur une forge



Un repository “local” lié à un “remote” sur une forge, lié à d’autres repositories



Ce qu'il faut retenir ...

Git : un outil très puissant vous garantissant

- la sauvegarde de tout l'historique de vos fichiers
- avec toutes les méta-données associées (utilisateur, date, ligne ...)
- la possibilité de naviguer, comparer, fusionner, revenir en arrière ...

Rien ne peut-être perdu !

Ce qu'il faut retenir ...

Git : un outil très puissant vous garantissant

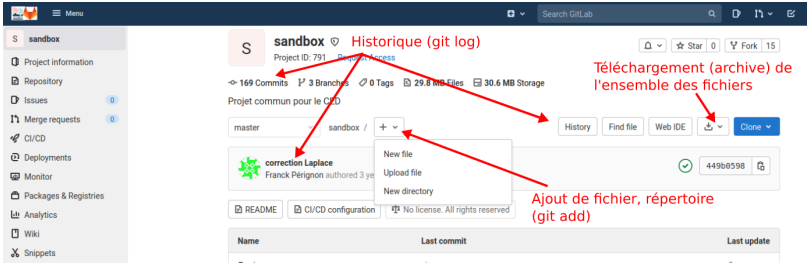
- la sauvegarde de tout l'historique de vos fichiers
- avec toutes les méta-données associées (utilisateur, date, ligne ...)
- la possibilité de naviguer, comparer, fusionner, revenir en arrière ...

Rien ne peut-être perdu !
Mais 😦 une courbe d'apprentissage un peu raide.
Rassurez-vous 😀 **Gitlab vous donne un accès simple à git**

A suivre : utilisation de Git via Gitlab

Démo : opérations git via l'interface gitlab

Toutes les opérations git peuvent être effectuées via l'interface.

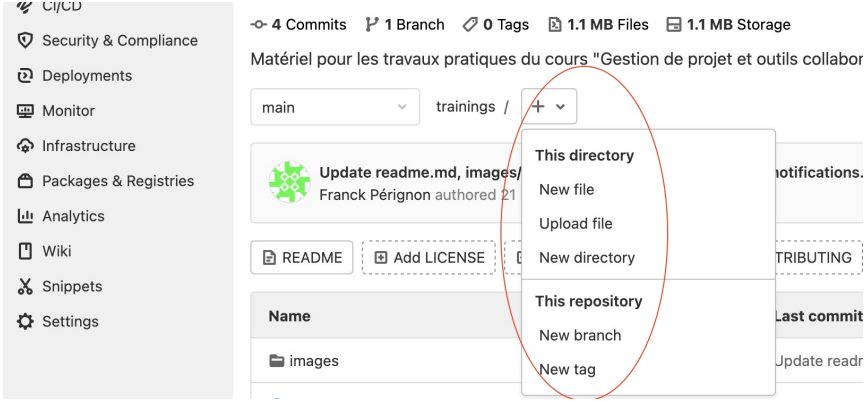


A suivre en mode "démo":

1. Navigateur de fichiers
2. Le menu repository : on passe sur chaque item ...
3. History, blame
4. Download
5. Choix de la branche

Edition et opérations git en ligne

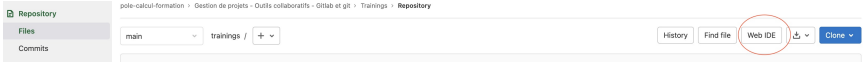
- le web editor.



Création, chargement de fichiers, création de branches etc

Edition et opérations git en ligne

- Web IDE



Gestion en ligne de tous les fichiers du repository

Edition en ligne et markdown

Le format **Markdown** :

- Une syntaxe enrichie...
- ...permettant de formatter simplement du texte (gras, italique, titres et sous-titres, bloc de code, images)...
- ...dans des fichiers de texte brut (et donc facilement versionnable, au contraire de *.doc* par exemple)

À partir d'un fichier au format markdown (*.txt, .md, .markdown*), il est très facile de produire une page web, un pdf, etc.

Edition en ligne et markdown

Format markdown = simplicité et efficacité

- Éditer un fichier texte peut se faire partout et rapidement
- Courbe d'apprentissage très rapide...
- ... Pour un usage excessivement courant : produire des documents formattés, pour une multitude de formats de publication (site web, rapport, présentations, etc.)

Exemple de syntaxe markdown

```
# Ceci est un titre

Voici *l'italique*, **le gras**, ~~le texte barré~~.

On peut insérer [des liens vers des pages web](https://mon-lien.com) et

## Ceci est un sous-titre.

On peut faire des listes à puces :
- 1
- Deux
- Three
```

Edition en ligne et markdown

gitlab flavored markdown : version étendue du Markdown

quelques règles syntaxiques supplémentaires pour les commentaires, issues, fichier d'aide etc.
Un peu de documentation :

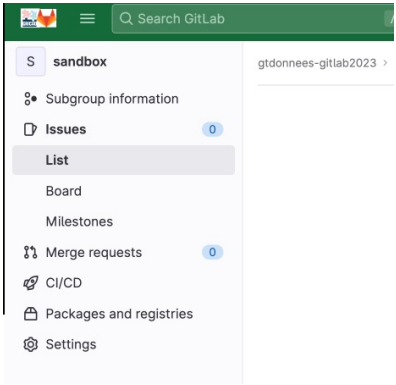
- https://docs.gitlab.com/ee/user/markdown.html
- https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Edition en ligne et markdown

Conseils/bonnes pratiques :

- Apprenez les bases du Markdown
- Au moins un README.md par projet
  (description sommaire du projet, du contenu, aide, liens utiles...)
  👍 meilleure lisibilité de votre projet !
- GitLab, vous pouvez éditer un fichier texte avec la syntaxe markdown, il sera affiché formatté (et donc plus lisible et attrayant)

Gestionnaire de problème : le menu **Issues**



- suivre l'évolution des problèmes (du signalement à la résolution),
- discuter de nouvelles idées,
- suivre l'ajout de nouvelle fonctionnalité, leur évolution
- List et Board pour visualiser, classer, étiqueter ...
- Milestones : définition d'étapes de développement, planification du projet

Gestionnaire de problème : le menu **Issues**

Quelques conseils et fonctionnalités utiles

- Créez des labels explicites pour classer vos issues (documentation, bugs, ...).
- Utilisez le markdown pour rédiger vos issues, vos commits
  - **@username** ⇒ email + ajout dans todo-list 

,
  - #*id* : réference à une issue
  - **Fix** #*id* : clore automatiquement une issue via un message de commit

  Voir (*https://docs.gitlab.com/ee/user/project/issues/managing_issues.html*).

Notifications et envoi d'emails pour les commits

Pour suivre l'activité du projet !

- Configuration des notifications (globales, par projet etc)

- Envoi d'emails à chaque push : *Settings* → *Integration* → *emails on push*

Git - Récupérer le dépôt du serveur gitlab sur votre machine

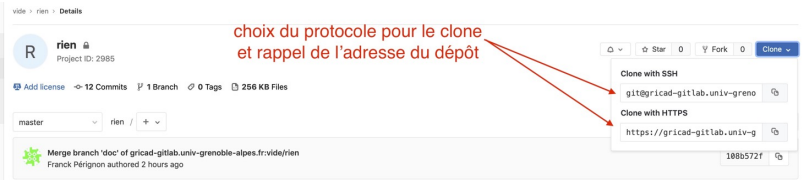
Deux protocoles disponibles

- “https” : authentification via login/mot de passe gricad-gitlab

```
git clone https://gricad-gitlab.univ-grenoble-alpes.fr/vide/rien
```

- “ssh” : authentification via une clé ssh

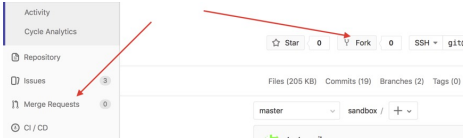
```
git clone git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git
```



Les merge-requests

Un mécanisme pour gérer proprement les contributions

- réserver la branche main pour la version principale et stable du projet,
- isoler les essais/implémentations de nouvelles fonctionnalités, corrections d'un bug etc.



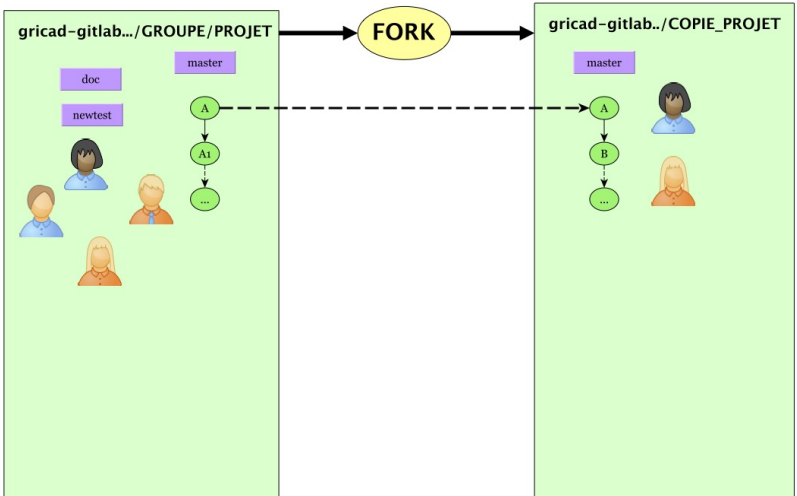
Merge-request : soumission d'une demande de fusion d'une branche dans une autre

- information puis révision par les membres du projets
- fusion dans la branche principale

Fork : duplication d'un projet (et donc copie du repository git)

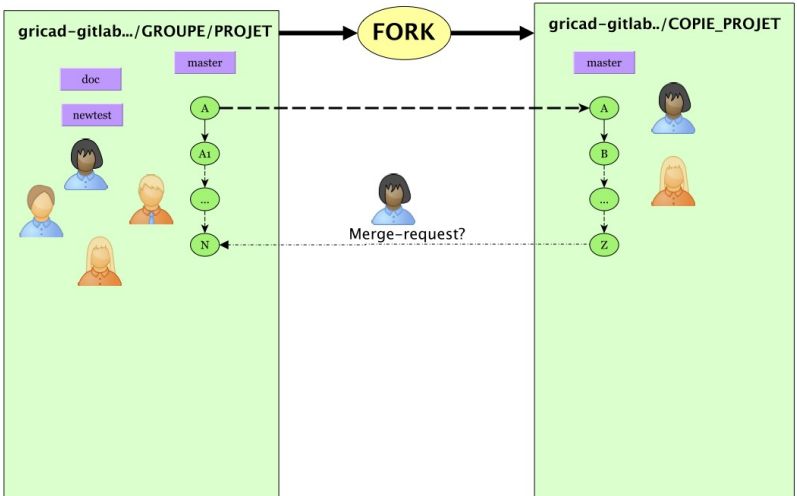
Les merge-requests, principe

Copie du projet gitlab vers un autre namespace, **Fork** ⇒ création d'un nouveau projet.



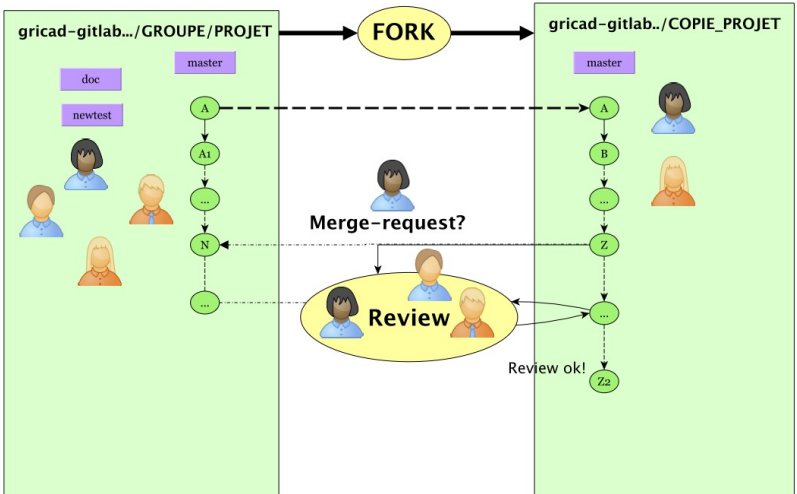
Les merge-requests, principe

Soumission d'une merge-request



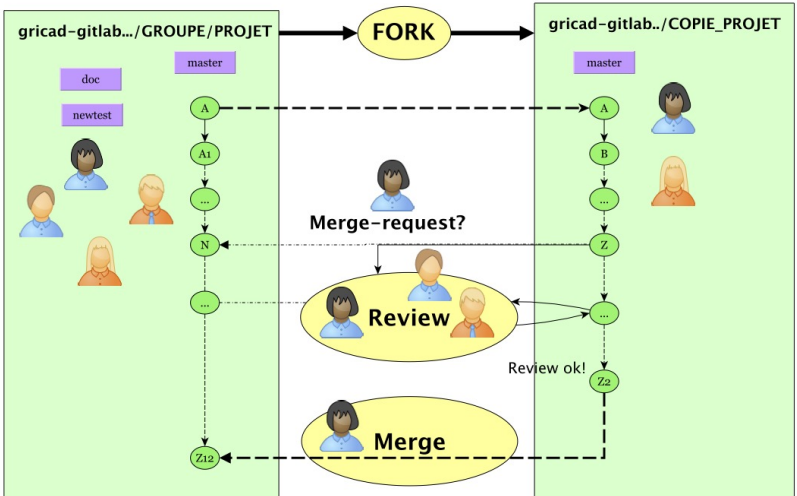
Les merge-requests, principe

Review, jusqu'à obtention d'une solution (un commit) acceptable et fusionnable



Les merge-requests, principe

Merge de la branche du fork vers le projet principal, cloture éventuelle de la MR.



API gitlab

Un outil d'interaction avec la plateforme permettant d'automatiser certaines opérations.
Une solution possible : le package python-gitlab

- des scripts (Python ou autre) pour automatiser des actions sur les utilisateurs, projets, groupes ...
- pré-requis : posséder un token personnel.
  Voir Personnal token dans la doc gitlab
- démo : voir le projet "Compléments"

Workflow git ?

Une méthode, une organisation de la gestion du repository

Pourquoi choisir un workflow ?

- Différents intervenants dans plusieurs contextes, avec leurs habitudes propres,
- d'où une gestion compliquée voir chaotique et donc probablement inefficace.
- Risques de perte de temps !

Quel "workflow" choisir ?

Beaucoup de possibilités et pas de réponse unique ...

- Workflow centralisé : tout le monde travaille sur la même branche.
  - nécessite une petite équipe et beaucoup de discussions
  - difficile de gérer les releases, les versions instables ...
- Git workflow
- GitHub flow
- OneFlow
- ...

Un exemple plus détaillé, Gitflow

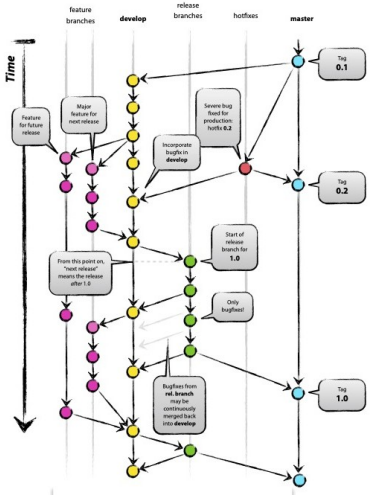
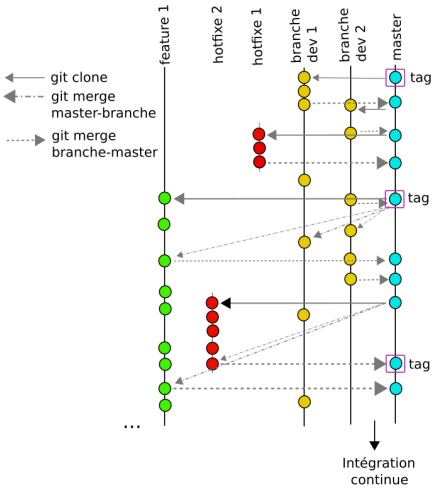
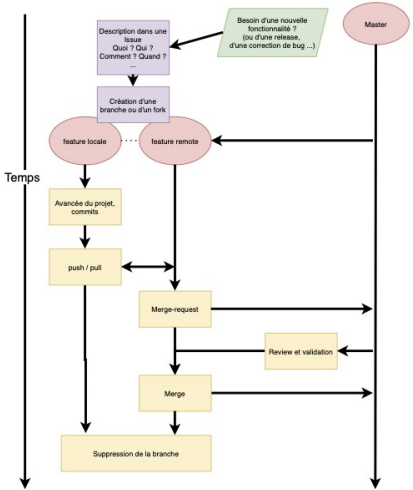


Figure: Gitflow

Gitlab, git, workflow, ... un exemple d'utilisation



Gitlab, git, workflow, ... Quelques recommandations



1. Utilisez les issues pour déclarer chaque problème, nouveau développement, etc
2. Créez une nouvelle branche (ou un fork) pour chaque nouvelle "feature", release, résolution de bug ...
3. Utilisez les merge-requests et profitez du processus de review.
4. Synchronisez votre repository régulièrement (pull/push). Plus une branche vit longtemps, plus le merge sera pénible ...