Git

Git est un gestionnaire de version décentralisé

Concepts importants de Git

- $blob$: Contenus de fichiers sauvegardés par Git ;
- $tree$: Arborescence de fichiers ;
- $commit$: Description d'un changement ;
- $sha1$: Identificateur unique pour tout objet stocké par Git ;
- $branch$: Séquence de commit caractérisée par un nom.

Connexion

- **Configuration :** $git\ config\ --global\ user.name\ "Laurent\ CAI"$
  $git\ config\ --global.email\ nom@email.com$
  $git\ config\ --global\ code.editor\ emacs$
- **Clé SSH :**
  - Communication cryptée ;
  - Authentification ;
  - Cryptographie à clé publique

  $ssh\ login@serveur$ Ouvre un terminal sur un serveur à travers une communication sécurisée.
  ⇒ Peut être utile pour déployer une application !

  $ssh-keygen$ Permet de créer une clé publique et une clé privée. La clé publique doit être transmise à toute instance qui souhaite vous authentifier. La clé privée ne doit bien sûr jamais être divulguée !

Commande Git

- $git\ init$: Créer un dépôt git local ;
- $git\ clone$: Cloner un dépôt git, suivi de :
  - En HTTPS : $git\ clone\ https://github.com/libgit2/ligbit2$;
  - En suivant le protocole : $git\ clone\ git@github.com:ocaml-sf/learn-ocaml.git$.
- $git\ add$: Mettre des documents dans le prochain commit ;
- $git\ status$: Visualiser les modifications non enregistrées ;
- $git\ diff$: Visualiser les modifications non encore enregistrées.
- $git\ commit$: Enregistrer les changements effectués :
  - $git\ commit\ -m\ 'Message'$ Permet de fournir le message en ligne de commande.
  - $git\ commit\ --ammend$ Permet de modifier le message du dernier commit.
- $git\ log$: Visualiser la liste des commit de la branche ;
- $git\ checkout$: Remplacer l'arborescence courante par une autre arborescence ;
- $git\ reset$: Pour ne plus considérer un fichier dans le prochain commit ;
- $git\ pull$: Se synchroniser avec une branche d'un dépôt ;
- $git\ push$: Pousser des changements de la branche locale courante dans une branche ;
- $git\ revert$: Créer un commit pour annuler un commit.
- $git\ rebase$: Réécrire le sommet de l'historique d'une branche.

GitLab

GitLab est un système de gestion de projet (avec plein de fonctions pour la gestion de projet).

« Demande de fusion » ou « Merge requests » (ou « Pull requests ») :

Demande effectuée pour permettre aux autres membres de lire le code puis pour le soumettre dans le dépôt plus tard.

Fork de projet GitLab

GitLab favorise le développement collaboratif en autorisant n'importe qui à créer une copie d'un projet. Cette opération s'appelle un fork dans le jargon de GitLab (« divergence » dans l'interface en français).
Un fork permet à toute personne extérieure aux membres d'un projet GitLab de proposer des contributions. Pour cela, il lui suffit, après avoir créé de fork et y avoir ajouté sa contribution, de créer une MR pour la proposer aux membres du projet d'origine (« upstream project »).

Gestion de tickets avec GitLab

GitLab permet de créer des tickets. Un ticket documente un problème à résoudre sur le projet : cela peut être une erreur à corriger dans le code, une fonctionnalité à rajouter, ...
Les tickets peuvent être référencés par les MR, dans les discussions ou dans les outils de gestion de projet. Cela améliore la traçabilité du projet.
Les tickets peuvent être ouverts (à résoudre) ou fermés (résolus).
Les tickets peuvent aussi être catégorisés par des étiquettes.

Méthode Kanban :

- Méthode d'organisation du travail basée sur un certain nombre de principe ;
- S'applique essentiellement au développement logiciel, elle fait partie des méthodes agiles.
- On y répartit des fichiers Kanban, des post-its représentant des **unités de travail** ;
- Divisé en au moins 3 colonnes : « en attente », « en cours » et « réalisé ».
  Mais on peut détailler plus !
- L'ensemble des colonnes représentent ce qu'on appelle le **workflow**. Celui-ci dépend de la méthode d'organisation utilisée par l'équipe.
  Workflow possible pour Scrum :
  - « $Stories$ » : Tickets brutes « user stories » ;
  - « $ToDo$ » : Items suffisamment détaillé/raffiné, sélectionnés pour le sprint en cours et assignés ;
  - « $InProgress$ » : Travail en cours de réalisation ;
  - « $Testing$ » : Travail réalisé mais pas complètement testé ;
  - « $Done$ » : Travail fini, testé (et intégré à la version de développement).

Méthodologie « git-flow »

Nous aurons une discipline pour organiser les branches des dépôts :

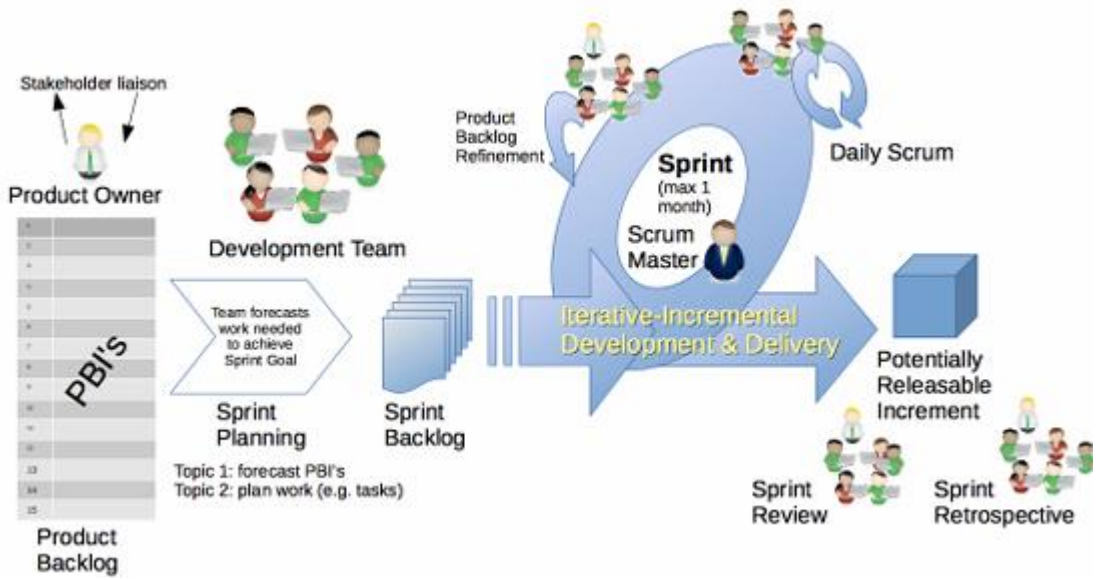
- La branche « master » ou « main » pour des commits de « releases » ;
- La branche « develop » pour des commits d'évolutions bien identifiables ;
- Chaque développeur travaillant sur une évolution est dans une branche locale dédiée.

Les méthodes Agiles

- Méthodes Agile = processus de développement de logiciel ;
- Retour sur les principes des méthodes de cette famille ;
- Scrum est l'une de ces méthodes ;
  Remarque : Ces méthodes ne se consacrent pas forcément aux mêmes aspects du cycle de développement et peuvent parfois se compléter.

Scrum :

- Processus incrémental et itératif ;
- Organisés autour de sprint ;
- Focalisé sur un ensemble fixé d'objectifs mais intègre le raffinement de ces objectifs (pour un sprint ultérieur) ;
- Mené par une équipe pluridisciplinaire qui va chercher le travail où il est ;
- Engendre un logiciel fonctionnel (intégré, testé, documenté et déployable).



Les rôles dans Scrum :

- Product owner : schématiquement, c'est le client. Contrairement aux approches traditionnelles, le client co-développe le logiciel. Le client sait ce qui est important, ce qui a de la valeur ;
- Equipe : Elle est formée de ses membres ($7 +/-2$), capables de prendre en charge tous les aspects du projet ;
- Scrum master : Il aide l'équipe à bien fonctionner en lui rappelant les principes du Scrum. Le Scrum Master partage son expertise et son expérience en développement logiciel et à un point de vue extérieur sur le projet.

« Product backlog »

- Liste ordonnée d'éléments (évolutions, développements interne, corrections de bugs) ;
- Il définit ce qu'est le projet et ce qui ne l'est pas ;
- Il caractérise ce qui est important et ce qu'il est moins.

Tests

Motivations :

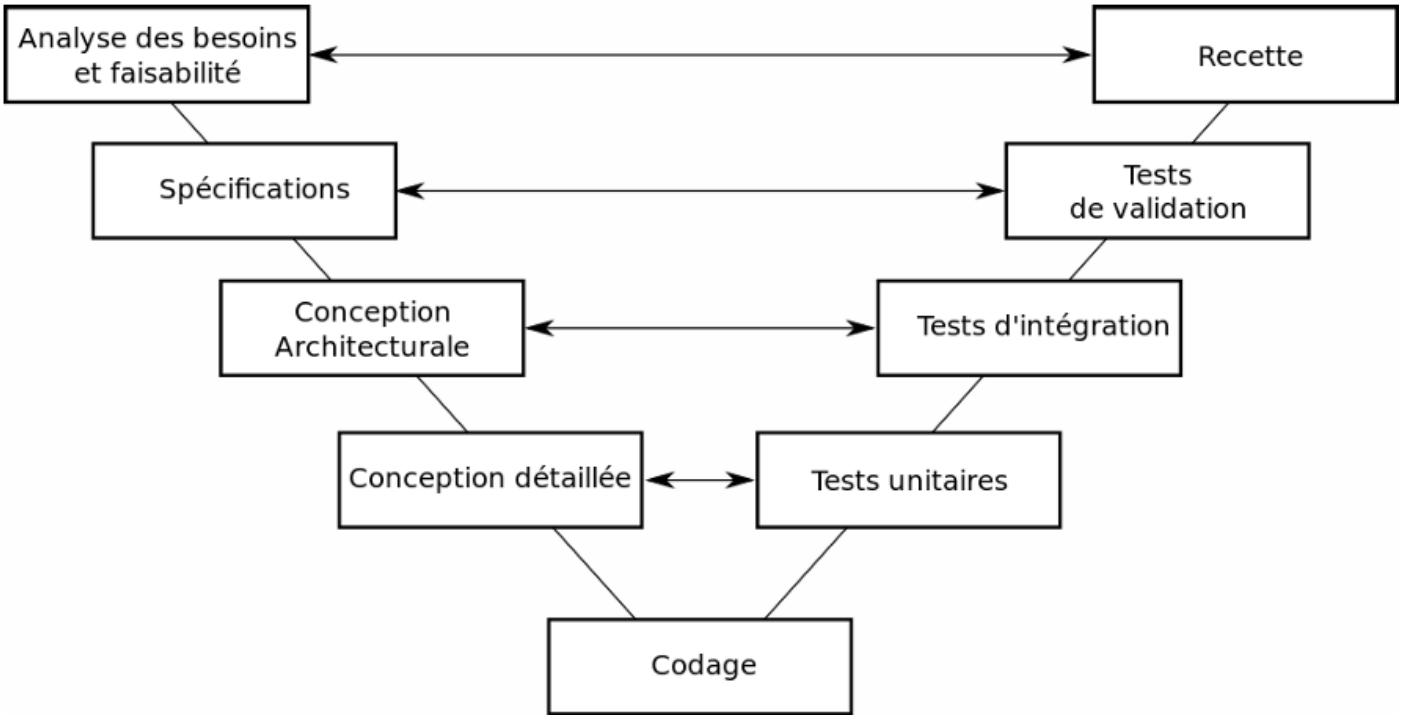
1) Trouver des bugs :
    - Plus les bugs sont trouvés tôt, moins ils coûtent cher ;
    - Par les développeurs.
2) Validation : Convaincre un tiers de la qualité du code
    - Client ;
    - Hiérarchie ;
    - Organisme de certifications.

➔ Problématique de l'évaluation d'un jeu de tests.

Tests

- **Statiques/dynamiques**
    - Statique : Sans exécuter le programme sur des données réelles : Analyse symbolique ou relecture de code ;
    - Dynamique : Exécuter le programme sur des exemples.
- **Dynamiques**
    - Code écrit à la main (tests unitaires) ;
    - Code généré automatiquement ;
    - Action manuelle ;
    - Action automatisée.
- **Ce qui est testé :**
    - Test de conformité ;
    - Test de robustesse ;
    - Test de sécurité ;
    - Test de performance.

Cycle en V



Test : Classement par niveau de détail

- Tests unitaires : Test des unités de programme de façon isolée : uniquement correction fonctionnelle ;
- Tests d'intégration : Test de la composition des modules via leur interface : principalement correction fonctionnelle ;
- Tests de validation : Test du produit fini par rapport au cahier des charges : conformité, robustesse, sécurité, performance ;
- Recette : Test du produit fini par le client.

Tests : Boîte noire/Boîte blanche

- Boîte noire :
  - Test de l'extérieur vis-à-vis des spécifications (formelles ou informelles) sans connaissance de l'implémentation ;
  - Technique applicable à tous les niveaux du cycle en V ;
  - Ecriture des tests possibles avant le code.
- Boîte blanche :
  - Tests en ayant connaissance du code ;
  - Si on change le code il faut mettre à jour les tests.

Oracle

Le test est réalisé vis-à-vis d'un comportement attendu. Celui-ci peut être défini par :

- Des spécifications ;
- Une norme ;
- Des contrats (préconditions, postconditions et invariants) ;
- Des versions précédentes.

Mocks : Faux objets qui imitent le comportement attendu. Utilisé quand le code n'est pas encore fonctionnel, événement non déterministe/rare (timeout, ...)

Approche pair-Wise

- La majorité des erreurs sont détectées par des combinaisons de 2 valeurs de variable ;
- Un seul test couvre plusieurs test couvre plusieurs paires est bien plus petit que le nombre de combinaisons.

Génération : Le problème de la génération d'un ensemble de données de tests de cardinal minimal est NP complet.

Fuzzing

- Génération d'entrées aléatoires pour essayer de faire planter le programme ;
- Génération à partir de rien ou bien par génération par mutation d'entrées valides ;
- Smart Fuzzing : En connaissance de la structure de l'entrée ;
- Dumb Fuzzing : Sans connaître la structure de l'entrée ;
- En boîte grise : Plus efficace si on instrumente le code binaire ;
- Nombreux succès en pratique, par exemple afl : Dumb Fuzzer en boite grise à base de mutation.

CI/CD

Introduction aux systèmes de build

Qu'est-ce qu'un système de build ?

- La compilation du code source ;
- La gestion des dépendances ;
- L'exécution des tests ;
- La génération de la documentation du code ;
- La génération de livrables (JAR, exécutables, etc.).

Pourquoi les utiliser ?

- Indépendance vis-à-vis de l'IDE utilisé ;
- Simplification de la gestion des dépendances ;
- Standardisation et automatisation des processus.

Systèmes de build :

- Maven :
  - Principe de convention plutôt que configuration (ex les sources da,s $src/main/java$, les tests dans $src/test/java$.
  - Gestion de dépendances via un dépôt central (Maven Central) ;
  - Cycle de vie standardisé (clean, compile, test, package, install, deploy).
- Gradle :
  - Plus flexible que Maven ;
  - Utilise un modèle de dépendance similaire à Maven ;

Intégration continue

La CI/ Continuous Integration/Integration continue est la pratique consistant à faire en sorte que les changements apportés au dépôt de développement soient immédiatement intégrables/utilisables en production.
En pratique, il s'agit de fusionner plusieurs fois par jour vers la branche principale tout en s'assurant de maintenir les critères de qualité.
Etape de CI :

- La **compilation** (un minimum) ;
- La vérification de la qualité du code (« **linter** ») ;
- L'exécution de la suite de **tests** ;
- **Déploiement** : Création de packages, publication de l'artefact, lancement de nouvelle version.

Déploiement continu

On veut aussi aller plus loin : Si la qualité est assurée, alors on déploie aussitôt l'artefact produit par le projet :

- Installation sur le serveur de production ;
- Ou bien publication d'un artefact/paquet sur un dépôt centralisé comme Maven Central, JCenter, NVM, FlatHub, voire Docker Hub… (cela dépend de la nature de l'artefact).

Cette partie du CI s'appelle le déploiement continu/continuous deployment.

Linter : Outil permettant de contrôler la qualité d'un code source de façon statique.

Bonnes pratiques

Conventions de nommage

- De classes, interfaces, énumérations et annotations → $UpperCamelCase$;
- De variables (locales et attributs), méthodes → $lowerCamelCase$;
- De constantes ($static\ final$ ou valeur d'$enum$) → $SCREAMING\_SNAKE\_Case$;
- De packages → Tout en minuscules sans séparateur de mots.
- **Se restreindre aux caractères suivants :**
  - $a - z, A - Z$: Les lettres minuscules et capitales (non accentuées) ;
  - $0 - 9$: Les chiffres ;
  - _ : Le caractère soulignement (seulement pour $snake\_case$).
- **Grammaire :**
  - Noms des types au singulier
  - Nom des classes-outil au pluriel
  - Nom des variables : singulier, sauf pour les collections
  - Boolean : past participle (e.g. finished)