Conduite de projet : Introduction

Aldric Degorre
(IRIF, U-Paris) – adegorre@irif.fr
(d'après transparents originaux de Yann Régis-Gianas)

18 septembre 2023

Un cours pré-professionalisant

Objectifs

- Ce cours vise à vous préparer au métier de **développeur**.
- Savoir : qu'est-ce que le cycle de vie d'un logiciel ?
- Savoir-faire : comment développer un logiciel ?
  - méthodes d'organisation
  - outils logiciels
- Savoir-être : comment se comporter comme un développeur ?

Dans votre cursus

- Vous allez appliquer ces compétences dans le cours "projet" du S4.
- Ce cours sera complété par "Génie logiciel avancé" en Master 1.
- Des compétences utiles pour tous les cours et pour votre futur métier.

Plan

Qu'est-ce qu'un logiciel ?

Qu'est-ce qu'un logiciel ?

Définition

Un logiciel est un ensemble de composants servant à faire fonctionner un système informatisé. Ces composants incluent (liste non exhaustive) : le code source, les outils de compilation, la documentation du système, la documentation utilisateur, les conditions d'utilisation, la licence, les méta-informations de diffusion et de déploiement, les fichiers de configurations, ...

Qu'est-ce qu'un logiciel ?

Définition

Un logiciel est un ensemble de composants servant à faire fonctionner un système informatisé. Ces composants incluent (liste non exhaustive) : le code source, les outils de compilation, la documentation du système, la documentation utilisateur, les conditions d'utilisation, la licence, les méta-informations de diffusion et de déploiement, les fichiers de configurations, ...

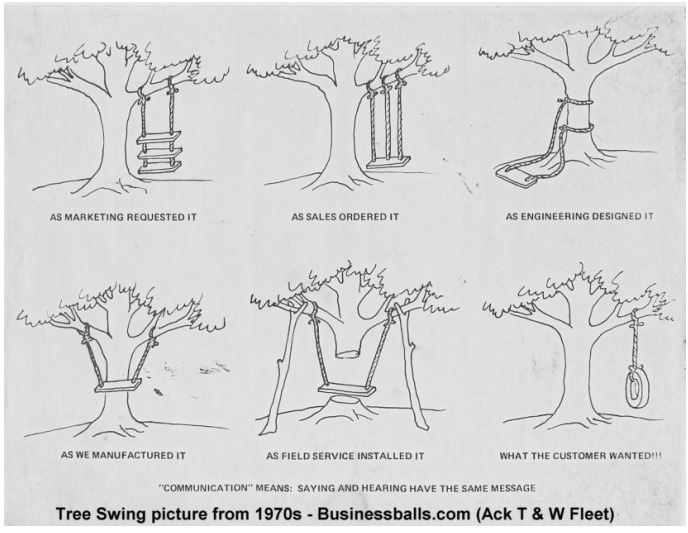
Point essentiel : Un logiciel a généralement des **utilisateurs**.

Qu'est-ce que le développement logiciel ?

Définition

Le développement logiciel est une activité dont l'objectif est de construire **un logiciel de qualité qui répond aux besoins de ses utilisateurs**.
(Building the right system right.)

Une célèbre illustration



https://en.wikipedia.org/wiki/Tree_swing_cartoon

Retour en 1968 : La crise du logiciel

Quand le coût du logiciel dépassa le coût du matériel.

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly : as long as there were no machines, programming was no problem at all ; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.
— Edsger Dijkstra, The Humble Programmer (EWD340), CACM

L'ingénierie à la rescousse

Source : Wikipedia

La notion de projet d'après l'AFNOR[1]

Processus unique, qui consiste en un ensemble d'activités coordonnées et maîtrisées comportant des dates de début et de fin, entrepris dans le but d'atteindre un objectif conforme à des exigences spécifiques, incluant des contraintes de délais, de coûts et de ressources.

1. https://groupe.afnor.org/pdf/fondamentaux-gestion-projet.pdf

La terminologie de la gestion de projet

Client

Le **client** ou **maître d'ouvrage** est l'organisation qui commande les travaux.

Fournisseur

Le **fournisseur** ou **maître d'œuvre** est l'organisation qui exécute les travaux.

Phase d'un projet

Un projet est une succession de **phases**. Chaque phase est constituée d'un objectif, d'une spécification, d'une documentation contractuelle, de livrables et d'une revue.

La terminologie de la gestion de projet

Client

Le **client** ou **maître d'ouvrage** est l'organisation qui commande les travaux.

Fournisseur

Le **fournisseur** ou **maître d'œuvre** est l'organisation qui exécute les travaux.

Phase d'un projet

Un projet est une succession de **phases**. Chaque phase est constituée d'un objectif, d'une spécification, d'une documentation contractuelle, de livrables et d'une revue.

Hypothèses : L'organisation réduit les risques et permet d'optimiser les ressources humaines et matérielles.

Quelles sont les spécificités du développement logiciel ?

- Le logiciel est un objet abstrait, fragile et dont la complexité n'est pas limité par des contraintes matérielles (fortes).
- Le logiciel est un objet en constante évolution. Tant qu'il a des utilisateurs, il est amené à évoluer ou à être adapté à l'environnement qui l'entoure.
- Le logiciel est utilisé par des utilisateurs non informaticiens. L'intuition de ce qui est faisable, ou pas, avec un logiciel n'est pas la chose la mieux partagée au monde.
- Le logiciel est d'une nature très variée. Construire une application web moderne est très différent du développement d'un logiciel embarqué dans un avion. Il existe une méthodologie adaptée à chaque type de logiciel.

Les grandes phases du développement d'un logiciel

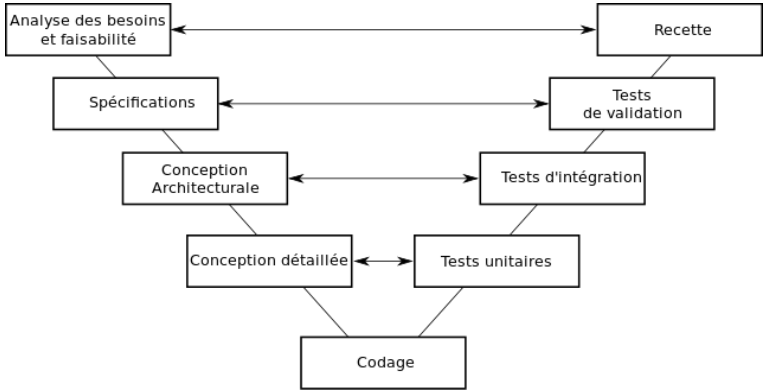
1. **Explicitation des besoins** : quel problème cherche-t-on à résoudre ?
2. **Conception** : comment pense-t-on résoudre ce problème ?
3. **Implémentation** : comment le résout-on effectivement ?
4. **Validation** : était-ce vraiment le bon problème ? et le cas échéant, est-ce une bonne solution ?

Qu'est-ce qu'un bon logiciel ?

Un bon logiciel est :

- **maintenable** :
  Il doit pouvoir évoluer.
- **sûr et sécurisé** :
  Il doit fonctionner comme prévu et ne doit causer aucun dommage.
- **efficace** :
  Il doit utiliser les ressources (temps, électricité, ...) avec parcimonie.
- **acceptable** :
  Il doit être adapté aux utilisateurs pour lesquels il a été conçu.

Méthodes traditionnelles : Cycle en V



La transition entre étapes est conditionné par la production d'un document.

https://fr.wikipedia.org/wiki/Cycle_en_V

Inadéquation des méthodes traditionnelles

The Mythical Man-Month – Fred Brooks

"For the human makers of things, the incompletenesses and inconsistencies of our ideas become clear only during implementation."

"The bearing of a child takes nine months, no matter how many women are assigned."

"Adding manpower to a late software project, makes it later."

Méthodes de la famille Agile (1/2)

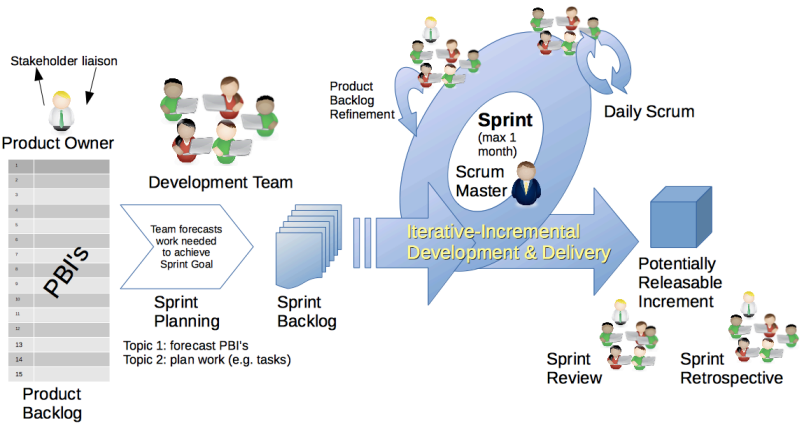
Tirés de https://fr.wikipedia.org/wiki/Manifeste_agile

- La plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- Accueillez positivement les changements de besoins, même tard dans le projet.
- Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.
- Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.
- Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont elles ont besoin et faites-leur confiance pour atteindre les objectifs fixés.

Méthodes de la famille Agile (2/2)

- Privilégiez la co-location de toutes les personnes travaillant ensemble et le dialogue en face à face comme méthode de communication.
- Un logiciel opérationnel est la principale mesure de progression d'un projet.
- Les processus agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.
- Une attention continue à l'excellence technique et à un bon design.
- La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle.
- Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées. À intervalles réguliers, l'équipe réfléchit aux moyens possibles pour devenir plus efficace. Puis elle s'adapte et modifie son mode de fonctionnement en conséquence.

Scrum


Stakeholder liaison
Product Owner
Development Team
Product Backlog Refinement
Sprint (max 1 month)
Scrum Master
Daily Scrum
PBI's
Team forecasts work needed to achieve Sprint Goal
Sprint Planning
Sprint Backlog
Iterative-Incremental Development & Delivery
Potentially Releasable Increment
Topic 1: forecast PBI's
Topic 2: plan work (e.g. tasks)
Product Backlog
Sprint Review
Sprint Retrospective

Grands principes du génie logiciel

Pour concevoir un bon logiciel, il faut être attentif à :

- **Abstraction** : trouver les concepts pertinents [2].
- **Encapsulation** : cacher les détails d'implémentation [3].
- **Modularité** : séparer les responsabilités des composants.
- **Hiérarchisation** : ordonner les abstractions.

Ces principes seront introduits dans ce cours sur des exemples.
Leur étude se fera en Master 1. [4]

2. Pour augmenter l'efficacité et diminuer la complexité du programme.
3. Pour limiter l'impact des changements d'un module dans les autres modules.
4. Mais ils seront aussi évoqués dans les divers cours de programmation.

Quelques citations à discuter

"First do it, then do it right, then do it better." - Addy Osmani

Quelques citations à discuter

"There's nothing more permanent than a temporary hack." - Kyle Simpson

Quelques citations à discuter

"Weeks of coding can save you hours of planning." - Unknown

Quelques citations à discuter

"When debugging, novices insert corrective code ; experts remove defective code." - Richard Pattis

Quelques citations à discuter

“Programming is the art of doing one thing at a time” - Michael Feathers

Quelques citations à discuter

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." – Martin Fowler

Quelques citations à discuter

"In programming, the hard part isn't solving problems, but deciding what problems to solve." - Paul Graham

Quelques citations à discuter

"Programming : when the ideas turn into the real things." - Maciej Kaczmarek

Quelques citations à discuter

"Plan to throw one away. You will, anyway." - Fred Brooks

Quelques citations à discuter

"Commenting your code is like cleaning your bathroom - you never want to do it, but it really does create a more pleasant experience for you and your guests." - Ryan Campbell

Plan

Mises en situation professionnelle

- Vous allez être plongés dans un projet qui suit un processus (inspiré de) Scrum.
- Vous êtes membres d'une équipe de développement de 6 personnes.
- Les rôles de "product owner" et de "scrum master" seront joués par l'un d'entre vous (à tour de rôle), et parfois par l'enseignant.
- Une semaine sur deux, la séance de TD sera le lieu de réunion pour l'équipe.
- Bien sûr, nous ne sommes pas en conditions réelles mais...
- Vous devez dédier 2 heures/semaine, en plus du TP, à la réalisation de vos tâches.
- Que peut-on faire avec $6 * 2 * 2 = 24h.\,etu$?

Ce que l'on attend de vous

Originalité de cours :
Vous serez évalués en partie sur votre comportement professionnel.

- Communiquer
- Collaborer
- S'impliquer
- S'organiser

Ce que l'on attend de vous

Originalité de cours :
Vous serez évalués en partie sur votre comportement professionnel.

- Communiquer
  - Exprimer ses difficultés pour obtenir de l'aide
  - Proposer des solutions en prenant en compte l'interlocuteur
  - Extraire des connaissances d'un expert
  - Expliciter les besoins d'un client
- Collaborer
- S'impliquer
- S'organiser

Ce que l'on attend de vous

Originalité de cours :
Vous serez évalués en partie sur votre comportement professionnel.

- Communiquer
- Collaborer
  - Agir conformément à son rôle
  - Mettre de côté son ego
  - Reconnaître ses erreurs
  - Accepter les compromis
  - Accepter les choix de l'équipe même s'ils sont en contradiction avec ses convictions
  - Savoir faire des critiques constructives
- S'impliquer
- S'organiser

Ce que l'on attend de vous

Originalité de cours :
Vous serez évalués en partie sur votre comportement professionnel.

- Communiquer
- Collaborer
- S'impliquer
  - Etre force de proposition.
  - Défendre ses opinions.
  - Etre sensible au bon fonctionnement du projet (et donc de l'équipe).
- S'organiser

Ce que l'on attend de vous

Originalité de cours :
Vous serez évalués en partie sur votre comportement professionnel.

- Communiquer
- Collaborer
- S'impliquer
- S'organiser
  - Evaluer sa productivité
  - Expliciter ses tâches à réaliser
  - Travailler en temps limité
  - Donner une priorité plus forte à ce qui est important et risqué.

Ce que l'on attend de vous

Mais concrètement, vous serez évalués sur :

- l'utilisation effective des outils présentés (d'après la trace laissée sur gitlab) [5] ;
- l'avancement final du projet (d'après le rendu final et la vidéo), cette note est collective [6] ;
- votre comportement professionnel (d'après les observations faites en séance) ;
- les connaissances acquises (d'après vos notes aux deux partiels).

5. Chaque semaine, on vous indiquera de quelles notions il sera indispensable de démontrer la mise en pratique.
6. L'idée n'est en aucun cas d'évaluer votre niveau individuel en programmation.

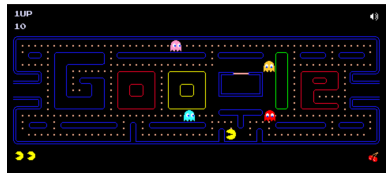
- L'évaluation sera continue et transparente.
- Ne pas venir en travaux dirigés, c'est s'exposer à ne pas valider le cours.
- Pas de seconde session.

Le sujet du projet

Pac-Man, un classique des années 80, à revisiter...

Contexte

- Créé par Tōru Iwatani pour l'entreprise Namco, commercialisé au Japon en 1980.
- 1 Pac-Man (joueur) et 4 fantômes (ordinateur) se déplacent dans un labyrinthe.
- Pac-Man doit éviter les fantômes pour ne pas être mangé... sauf quand il mange une super pac-gomme, ce qui inverse les rôles.



Version de Pac-Man développée par Google pour fêter les 30 ans du jeu.

Plus de détails : `https://fr.wikipedia.org/wiki/Pac-Man`.

Le sujet du projet

Problème

Votre studio de développement de jeu vidéo pense que le concept a encore du potentiel et veut commercialiser une nouvelle version de Pac-Man aux goûts du jour.

Malheureusement, la première équipe de développement vient d'être débauchée par un concurrent. Vous faites partie de la nouvelle équipe, à vous de jouer !

Plan

Le problème de la gestion de versions

Comment faire pour modifier les fichiers d'un projet ?

Le problème

- Les fichiers sont partagés entre les membres de l'équipe.
- Les développeurs font des modifications concurrentes sur ces fichiers.
- On ne veut perdre le travail de personne.
- Peut-être que certaines modifications doivent être annulées.
- La gestion de projet demande une tracabilité des modifications.

Le problème de la gestion de versions

Comment faire pour modifier les fichiers d'un projet ?

Le problème

- Les fichiers sont partagés entre les membres de l'équipe.
- Les développeurs font des modifications concurrentes sur ces fichiers.
- On ne veut perdre le travail de personne.
- Peut-être que certaines modifications doivent être annulées.
- La gestion de projet demande une tracabilité des modifications.

Les mauvaises réponses

- S'envoyer `projetv1.zip`, ..., `projetv43242.zip` par email.
- Utiliser un espace de stockage partagé (`DropBox`, `NextCloud`, ...).

Principe de fonctionnement d'un gestionnaire de versions

Définition

Un **système de contrôle de versions** (Version Control System, VCS) est un outil qui maintient l'ensemble de toutes les modifications effectuées sur un ensemble de fichiers donnés, appelés **dépôt** (repository en anglais).

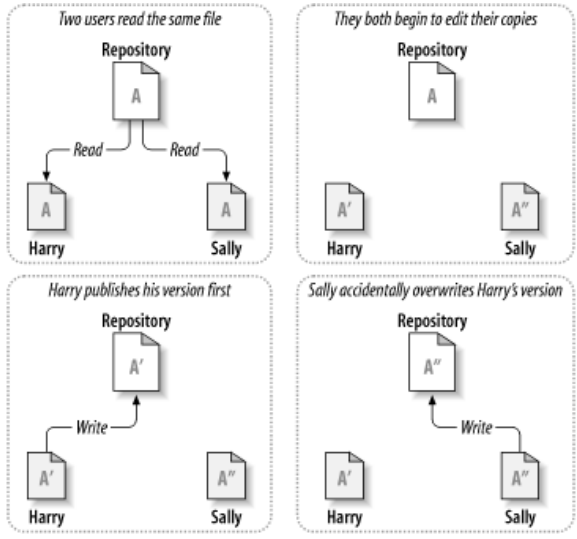
Garanties

- Sauf demande explicite de l'utilisateur, un VCS ne perd aucune modification.
- Un VCS détecte les modifications concurrentes et permet la résolution de conflits.

Cas d'usage

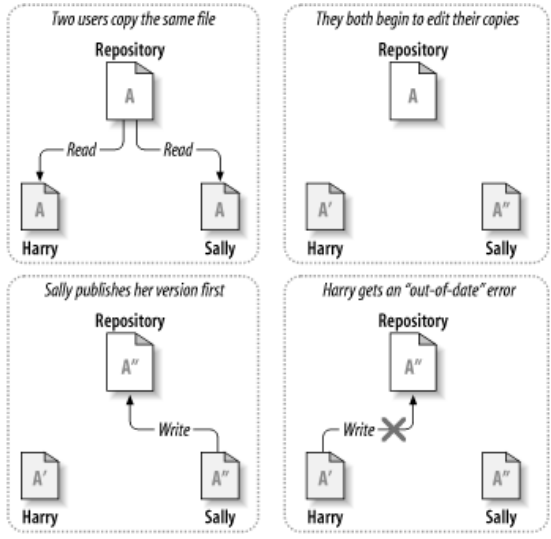
- Historique de ses fichiers (undo possible).
- Système de sauvegarde.
- Développement collaboratif.

Différents modèles de collaboration



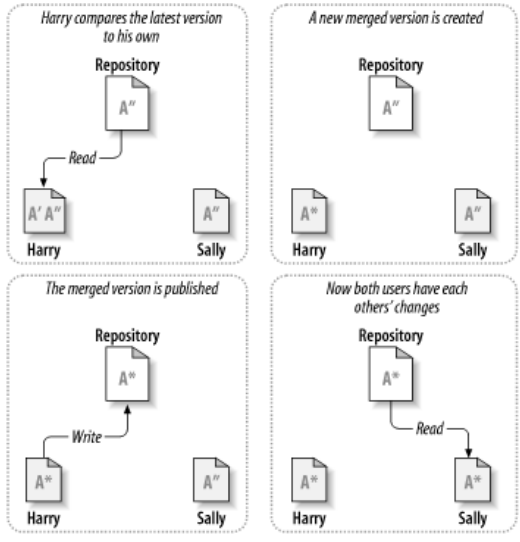
Source : SVN book.

Différents modèles de collaboration



Source : SVN book.

Différents modèles de collaboration



Source : SVN book.

Spécificités de Git

- Git est un gestionnaire de version **décentralisé**.
- Cela signifie que chaque utilisateur a son propre dépôt.
- Les utilisateurs de Git s'échangent des changements.
- C'est le système de contrôle de version le plus populaire.

Nous allons maintenant passer à une démonstration des fonctionnalités principales de Git. Cette démonstration ne sera pas suffisante pour que vous maîtrisiez cet outil important.

Vous devez donc :

- Rejouer cette démonstration sur votre machine.
- Lire de la documentation, comme par exemple l'excellent GitBook :

`https://git-scm.com/book/fr/v2`

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 commits caractérisée par un nom.

Installation de Git

```
$ sudo apt install git
```

(ou tout équivalent sur votre distribution GNU/Linux préférée.)

Pour plus d'information :

https://git-scm.com/

Configuration de Git

```
1  $ git config --global user.name "John Doe"
2  $ git config --global user.email johndoe@example.com
3  $ git config --global core.editor emacs
```

Il existe aussi

- des outils/plugins pour intégrer les commandes git à vos environnements de développement : magit pour Emacs,git-vim pour Vim, egit pour Eclipse, ...
- des environnements de développement dans lesquels git est intégré par défaut : IntelliJ IDEA, Visual Studio Code, ...

Avertissement : quand git est intégré dans une GUI, ce n'est pas toujours évident de savoir quel contrôle lance quelle commande git...

Création de premier dépôt Git

Pour créer un dépôt git local à votre machine :

```
1  $ git init
```

Clonage d'un dépôt Git

Pour créer un dépôt git local à votre machine à partir d'un dépot existant :

En HTTPS :

```
$ git clone https://github.com/libgit2/libgit2
```

En suivant le protocole git/ssh :

```
$ git clone git@github.com:ocaml-sf/learn-ocaml.git
```

Parenthèse : Qu'est-ce que SSH ?

- Communication cryptée
- Authentification
- Cryptographie à clé publique

```
$ ssh login@serveur
```

ouvre un terminal sur un serveur à travers une communication sécurisée.

$\Rightarrow$ 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 !

Vous verrez au premier TP comment installer votre clé SSH.

`git add`

Pour demander à git de prendre en charge un fichier dans le prochain commit.

```
$ git add filename
```

Avertissement : tout fichier n'a pas vocation a être pris en charge par git. Évitez "~~`git add .`~~".

- **Doit** être suivi par git : tout fichier sur lequel vous avez directement travaillé (code source, texte de documentation, ...)
- **Ne doit pas** être suivi :
  - tout fichier automatiquement généré (par le compilateur, par l'IDE, ...)
  - ce qui appartient seulement à la configuration locale
  - ce qui est confidentiel (mots de passe, ...)

Pensez à renseigner le fichier `.gitignore`.

git status

Pour connaître l'état des fichiers du dépôt.

```
$ git status
```

git diff

Pour visualiser les modifications non encore enregistrées.

```
$ git diff
```

git commit

Pour enregistrer les changements effectués sous la forme d'un commit.

```
$ git commit
```

ouvrira un éditeur pour y écrire un **message descriptif du changement**.

Nous verrons que ce message doit être rédigé avec soin.

```
$ git commit -m 'Message'
```

permet de fournir le message en ligne de commande.

```
$ git commit --amend
```

permet de modifier le message du dernier commit.

git log

Pour visualiser la liste des commits de la branche courante :

```
$ git log
```

git checkout

Pour remplacer l'arborescence courante par une autre arborescence sauvegardée :

```
$ git checkout sha1
```

```
$ git checkout -- filename
```

restaure un fichier dans son état précédent.

git reset

Pour ne plus considérer un fichier dans le prochain commit :

```
$ git reset HEAD fichier
```

git pull

Pour se synchroniser avec une branche d'un dépôt :

```
1 $ git pull repository refspec
```

récupère les changements du dépôt **repository** dans la branche **refspec** et les fusionnent (merge) avec les changements de la branche locale courante.

Si le dépôt a été cloné :

```
1 $ git pull
```

récupère les changements de la branche distante correspondant à la branche locale courante [7] du dépôt d'origine.

7. La branche par défaut est souvent appelée "**master**" ou "**main**".

git pull

Pour se synchroniser avec une branche d'un dépôt :

```
1 $ git pull repository refspec
```

récupère les changements du dépôt **repository** dans la branche **refspec** et les fusionnent (merge) avec les changements de la branche locale courante.

Si le dépôt a été cloné :

```
1 $ git pull
```

récupère les changements de la branche distante correspondant à la branche locale courante[7] du dépôt d'origine.

Et si la fusion est impossible parce qu'il y a un **conflit** ?

7. La branche par défaut est souvent appelée "**master**" ou "**main**".

Résolution des conflits (1/2)

Si deux utilisateurs ont modifié en même temps une même ligne d'un fichier donné, la fusion automatique des changements est impossible.

Git détecte cette situation et produit un message de la forme :

```
Auto-merging filename
CONFLICT (content): Merge conflict in filename
Automatic merge failed; fix conflicts and then commit the result.
```

Comme l'indique ce message, Git vous demande de corriger vous même les conflits. Pour cela, il faut éditer le fichier qui contient un conflit (ici, **`filename`**) et le résoudre (voir transparent suivant).

Résolution des conflits (2/2)

Les conflits sont repérés dans le fichier par des lignes de la forme suivante :

```
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```

On y distingue deux zones qui correspondent aux deux modifications en conflit. Il vous suffit d'éditer le fichier en supprimer toutes ces lignes et en les remplacant par le contenu de votre choix.

Une fois ces corrections faites, les commandes

```
$ git commit -m 'Fix conficts and merge.'
```

finalisera la fusion des changements sous la forme d'un nouveau `commit`.

`git push`

Pour pousser des changements de la branche locale courante dans une branche **`refspec`** d'un dépôt **`repository`** :

```
$ git push repository refspec
```

Pour pousser les changements de la branche locale courante dans la branche amont ("upstream") :

```
$ git push
```

Remarques :

- Si la branche amont a été mise à jour entre temps, il faut d'abord faire un **`git pull`**.
- Dans un dépôt cloné, toute branche a une branche amont qui lui correspond dans le dépôt d'origine (appelé **`origin`**).

git checkout -b branch

Pour créer une nouvelle propre branche dans votre dépôt local, faire :

```
$ git checkout -b branch-name
```

Pour exporter cette nouvelle branche sur un autre dépôt :

```
$ git push -u repository branch-name
```

Par exemple, si votre dépôt est issu d'un clone, faites :

```
$ git push -u origin branch-name
```

pour pousser la branche **branch-name** sur le dépôt d'origine.

Autres commandes importantes

Les commandes suivantes pourront aussi vous être utiles :

- `git rebase`
- `git pull --rebase`
- `git revert`
- `git blame`
- `git bisect`

`git rebase` et `git pull -rebase` seront vues en cours. [8]

Pour le reste, regardez la documentation !

Renseignez-vous aussi sur rôle du fichier `.gitignore`.

8. Peut-être les autres si le temps le permet.

Plan

Ce qu'il faut retenir avant tout

- Le logiciel, ce n'est pas que le code.
- Développer, c'est programmer en équipe.
- Collaborer, cela demande de communiquer.
- Le développement logiciel a des spécificités.
- Les méthodes agiles sont des processus adaptés à celles-ci.
- Un développeur doit maîtriser Git et GitLab (prochain cours).

Liens utiles

- Page moodle
  `https://moodle.u-paris.fr/course/view.php?id=1630` [9]
- Page du cours sur gitlab :
  `https://gaufre.informatique.univ-paris-diderot.fr/cproj/cours/`
- Page du projet Pac-Man à cloner (attention, pas encore en ligne !) :
  `https://gaufre.informatique.univ-paris-diderot.fr/cproj/pacman/`

9. On utilisera moodle surtout pour les annonces. Mais vous pourrez retrouver les autres liens sur cette page.

À faire avant le prochain TP

- Lire `https://git-scm.com/book/fr/v2`
- Lire `https://scrumprimer.org/primers/fr_scrumprimer20.pdf`

(mais pas de panique, nous en reparlerons !)

Et surtout...

- S'assurer que vous pouvez vous connecter sur `https://gaufre.informatique.univ-paris-diderot.fr/` (avec votre compte UFR d'info).
- Cloner `https://gaufre.informatique.univ-paris-diderot.fr/cproj/pacman/` sur sa machine.
- Le compiler et l'exécuter.
- Former des trinômes !. Vous pouvez vous aider du forum dédié sur Moodle. Le chargé de TP réunira les trinômes par 2 pour former les équipes.