CSC4102 : Gestion de versions

Denis Conan, avec Olivier Berger

Janvier 2025

1 Motivations et objectifs

1.1 Contexte : Qui demande des versions ?

- Linus's law : « *given enough eyeballs, all bugs are shallow* » [Raymond, 1999]
  - « Avec suffisamment d'yeux, tous les bugs sautent aux yeux »
- Cette affirmation verbalise un principe fort des promoteurs du code ouvert
  - Code ouvert $\Longrightarrow$ code source accessible
  - Code accessible $\Longrightarrow$ relecture par les pairs, promotion des bonnes pratiques, vérification de la non-existence de codes cachés, etc.
- Même si le code n'est pas ouvert, l'accessibilité en interne est souhaitable
- Code accessible = dans une forge logicielle et sous gestion de versions
  - Dans CSC4102, c'est GitLabEnse avec Git

1.1.1 « *Experts keep sketches* » [van der Hoek, 2018]

Image extraite de [van der Hoek, 2018]

1.2 Cette séance et après

- Contenu — Quels sont les objectifs de la séance ?
  - Mise en place de votre projet GitLabEnse
  - Pratique de la gestion de versions avec Git
    - Concepts de base
    - Processus de travail
- Après la séance — À quoi ça sert ? où ? quand ?
  - Tout artefact (modèle ou code) dans le dépôt Git de votre projet GitLabEnse
  - Les évaluations sont effectuées en étudiant les dépôts Git
    - Pour le suivi (évaluation formative) et la notation (évaluation certificative)
- Avertissement
  - La gestion de version est importante pour la suite de votre formation ainsi que pour votre insertion en entreprise ou dans des projets informatiques
  - Que les deux membres du binôme contribuent, et ce tout au long du module
  - La pratique régulière est importante pour acquérir une aisance minimale

2 Concepts de la gestion de versions avec Git

2.1 Entrée sous suivi de version (*tracked entry*) et instantané du dépôt local (*snapshot*)

- Entrée sous suivi de version = fichier/répertoire « connu » de GIT
  - Après clonage (`git clone`), toute l'arborescence récupérée est sous suivi de version
- Instantané (*snapshot*) = image de l'arborescence correspondant à l'état après une validation (*commit*)
  - Le résultat de l'opération `git commit` est un nouvel instantané
- Ajout d'une entrée pour le suivi de version
  - Après création d'une entrée (p.ex., `touch new.txt`), besoin de mettre la nouvelle entrée sous suivi (avec `git add`)
    - C'est le premier rôle de la commande `git add`

2.2 Zone de transit (*staging area*) et validation (*commit*)

- Zone de transit (*staging area*)[1] = entrées dites « indexées » pour ajout au prochain instantané du dépôt local
- La zone de transit contient :
  - Les entrées non encore suivies ajoutées pour suivi de version
    - P.ex., `new.txt` n'existe pas,
      puis « `touch new.txt; git add new.txt` »
  - Les entrées modifiées dont les modifications seront dans le prochain instantané
    - P.ex., `exists.txt` déjà dans le dépôt local,
      puis « `echo ajout >> exists.txt; git add exists.txt` »
      - C'est le second rôle de la commande `git add`
- Effet d'une validation (`git commit`) = création d'un nouvel instantané par application des modifications de la zone de transit

1. par abus de langage de « rendre une valeur économique solidaire ou dépendante de la valeur d'un élément de référence »

2.3 États possibles d'une entrée

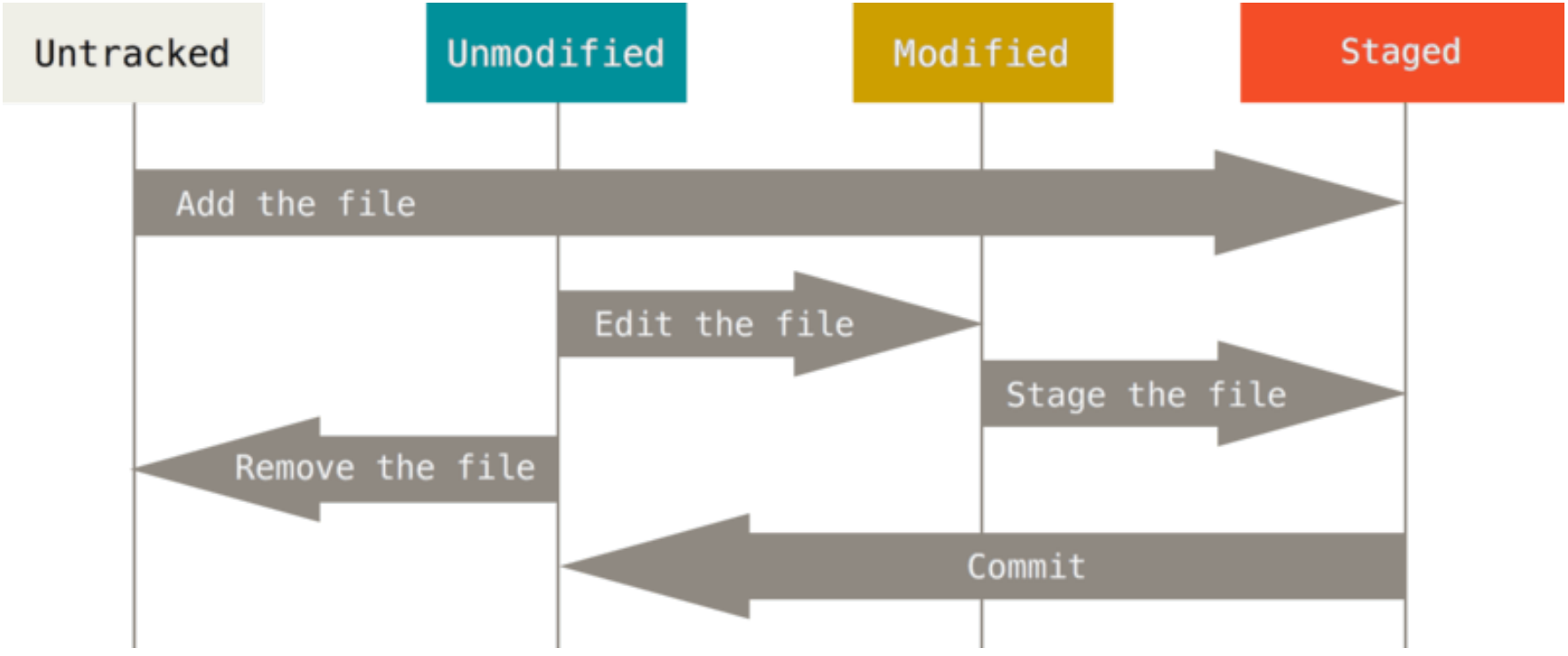


Image extraite de [Chacon and Straub, 2014]

- Cf. commande `git status` pour connaître l'état des entrées

2.4 Entrées ignorées

- Certains fichiers (temporaires) n'ont pas vocation à être sous suivi de version
  - Langage C : les fichiers `*.o`, etc. et les exécutables
  - Langage JAVA : les fichiers `*.class`
  - Eclipse : les répertoires `bin`
  - Maven : les répertoires `target`
  - Etc.
- Utiliser le fichier `.gitignore` pour indiquer les entrées à exclure

```
**/*.[oa]
**/*.class
**/bin/
**/target/
```

- Ne pas oublier « `git add .gitignore` » puis « `git commit` »
- Un fichier `.gitignore` est fourni dans le dépôt de départ

2.5 Contenu de l'arborescence locale

- Répertoires d'un projet GIT =

  (Instantané du dépôt local

  + Entrées non suivies (incluant les entrées « ignorées »)

  + Entrées sous suivi, modifiées, mais non indexées

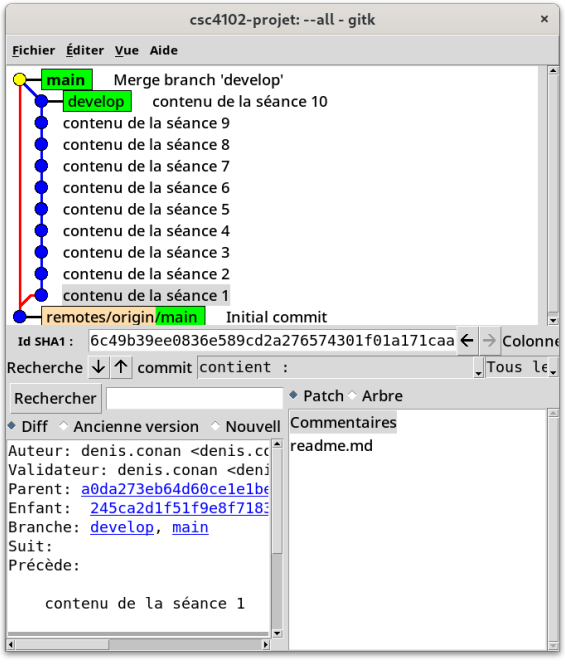
  + Zone de transit (aussi appelée les entrées indexées))

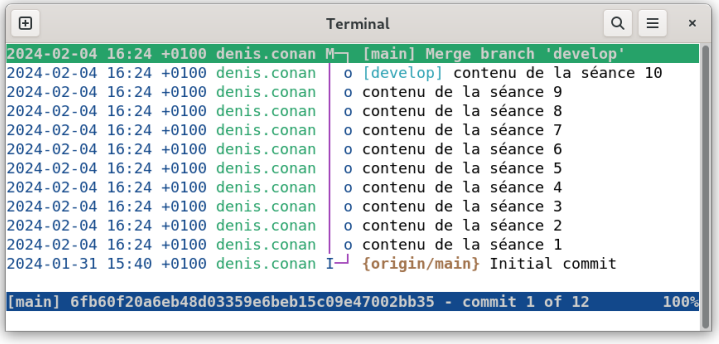
2.6 Dépôt et branches

- Dépôt d'un projet GIT =
  - Conceptuellement : un ensemble d'instantanés organisés dans un graphe
    - Chaque instantané possède un identifiant (`SHA1`)
    - Chaque instantané connaît ses ancêtres directs (p.ex. 2 lors d'une fusion)
  - Pratiquement : méta-données (`.git/`) + entrées sous suivi de version
- Branche = « division », « ramification », « lignée » d'instantanés
  - Par convention, la première branche ou branche principale est nommée `main`
- Un utilisateur passe de branches en branches lors de son travail
  - La branche entre deux livraisons est souvent appelée `develop`
  - La correction d'un *bug* peut être faite dans une branche séparée
  - Le développement d'une nouvelle fonctionnalité peut être fait dans une branche

2.7 Fusion de branches

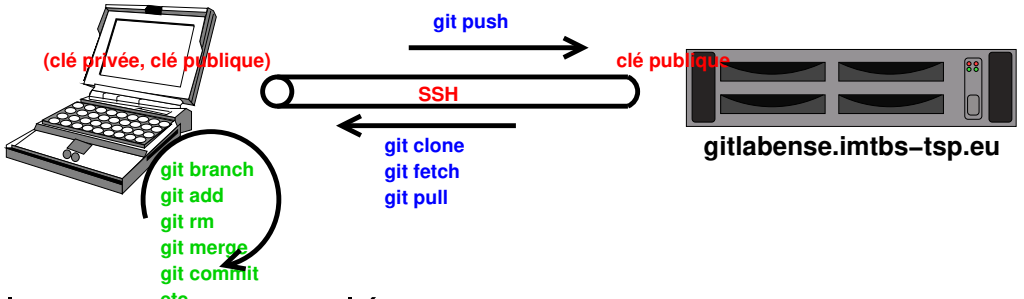
- Avec « `git merge --no-ff` »[2] : un instantané dédié à la fusion (`gitk`/`tig`)





2. *no fast forward*

2.8 Dépôts distants



- Par convention, le dépôt source du clonage est appelé `origin`
- Possibilité de créer d'autres liens vers des dépôts distants (*remotes*)
  - Liste et manipulation (`git remote` ——help)

```
$ git remote -v
origin git@gitlabense.imtbs-tsp.eu:enseignants-csc4102/csc4102-projet.git (fetch)
origin git@gitlabense.imtbs-tsp.eu:enseignants-csc4102/csc4102-projet.git (push)
```

- Chaque lien est bidirectionnel :
  - Tirer du contenu depuis un dépôt distant :
    - Récupérer les modifications sans modifier son arborescence locale
      p.ex. (`git fetch origin`)
    - Appliquer les modifications sur l'arborescence de la branche courante
      p.ex. (`git pull origin` *`branche`*)
      - Avant `git pull` vérifier avec `git branch` que l'on est sur *`branche`*
      - Il existe peut-être des conflits sur certains fichiers
        p.ex. `git mergetool --tool=meld`
  - Pousser du contenu sur un dépôt distant (`git push origin` *`branche`*)

3 Mise en pratique en TP (2h) (+ HP [1h])

- À partir d'un tutoriel, en TP
  - Création et configuration d'un projet dans la forge logicielle GitLabEnse de TSP
  - Manipulations de base d'un dépôt Git et transfert entre dépôts
  - Manipulations des branches et des étiquettes
  - Projet privé GitLabEnse configuré + branche `develop` créée
- Contenu du HP avant la séance 2 (1h)
  - Préparation de la séance 2
    - Lecture du cahier des charges de l'étude de cas du projet du module

Références I

Chacon, S. and Straub, B. (2014).

Pro Git, 2nd Edition.

APress.

https://git-scm.com/book/en/v2.

Raymond, E. (1999).

The Cathedral & the Bazaar — Musings on Linux and Open Source by an Accidental Revolutionary.

O'Reilly.

van der Hoek, A. (2018).

What Makes Expert Software Designers Successful ?

ACM Learning Webinar.

Pourquoi Git ?

Linus Torvalds, IEEE Computer Society Computer Pioneer Award, 2014

- Libre
- Initialement développé pour le noyau Linux par Linus Torvalds
- Utilisé par de nombreux « gros » projets
  - `https://git.wiki.kernel.org/index.php/GitProjects`
  - « The Git Distributed Version Control System—An Interactive Development History »
- Grande flexibilité dans l'organisation du travail
  - Complètement réparti = Liens point à point quelconques entre deux dépôts
  - Branche = concept de première classe
  - La plupart des opérations sont « locales », c.-à-d. sans besoin de connexion à un dépôt distant

Avertissement : `main` *Vs.* `master`

- Depuis mars 2021, la branche principale d'un projet Git dans la forge logicielle GitLabEnse est nommée `main`
  - Cf. *The new Git default branch name*
  - Vous n'êtes pas autorisés à utiliser un nom autre que `main` pour la branche principale de votre projet
    - C'est une hypothèse pour nos scripts d'analyse de vos dépôts Git
- Par conséquent, faites attention aux « copier/coller » intempestif en provenance du Web
  - Nous proposons un apprentissage et des procédures
  - Prendre des commandes d'une page Web sans trop les comprendre vous expose à des confusions entre `main` et `master` qui sont difficiles à résoudre