UNIVERSITÉ de la NOUVELLE-CALÉDONIE

Cours de Développement Web
Gestion de versions / Git

Frédéric Flouvat

Université de la Nouvelle-Calédonie

frederic.flouvat@univ-nc.nc

Quelques références bibliographiques

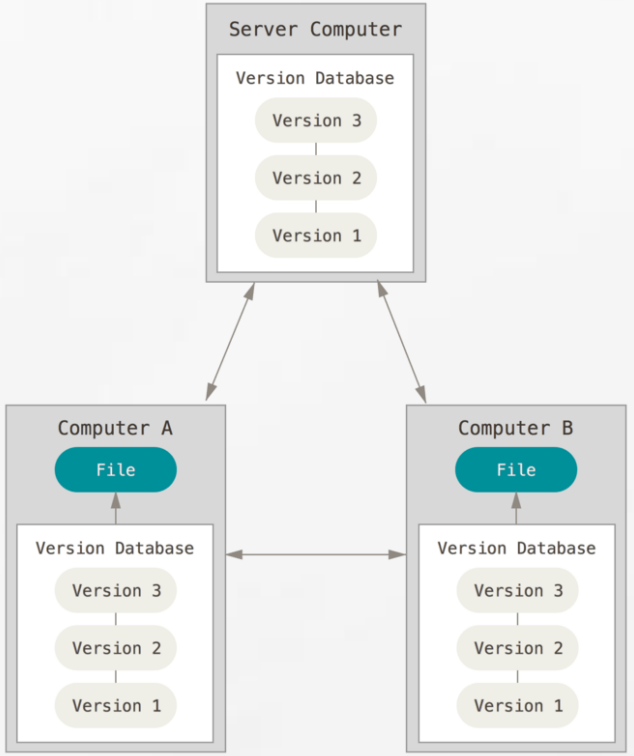
- Documentation officielle https://git-scm.com/book/fr/v2/Démarrage-rapide-Rudiments-de-Git
- "Pour arrêter de galérer avec git", T. Jouannic https://www.miximum.fr/blog/enfin-comprendre-git/
- "Git – un petit guide", R. Dudler http://rogerdudler.github.io/git-guide/index.fr.html
- "Gérer votre code avec Git et GitHub", M.G. Gauthier, OpenClassrooms https://openclassrooms.com/courses/2342361-gerez-votre-code-avec-git-et-github
- "Become a git guru", Atlassian, https://www.atlassian.com/git/tutorials
- "La puissance des workflow git", https://medium.com/@OVHUXLabs/la-puissance-des-workflows-git-12e195cafe44

La gestion de versions (*Version Control System*)

- "Système enregistrant l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce qu'on puisse rappeler une version antérieure d'un fichier à tout moment" (Documentation Git)

- Une gestion de version distribuée
  - Un dépôt central (*server*)
  - Des clients avec des copies complètes du dépôt central

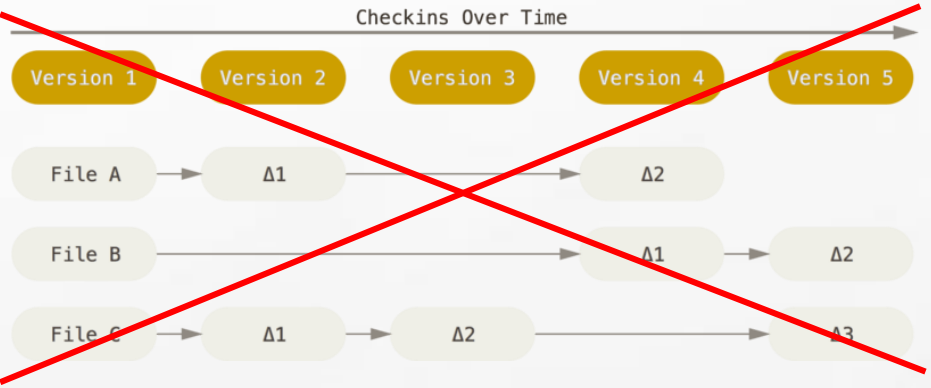
  ➢ Une architecture robuste en cas de panne du serveur

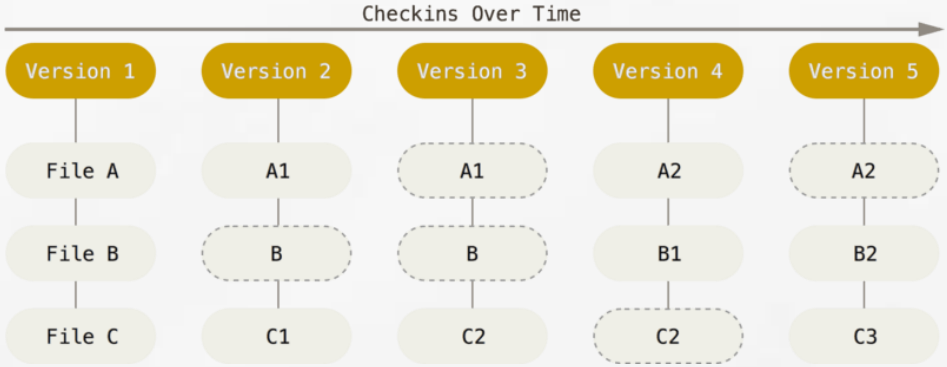


https://git-scm.com/book/fr/v2/Démarrage-rapide-A-propos-de-la-gestion-de-version

Introduction à Git

- Un outil libre (avec une interface en ligne de commandes) pour faire de la gestion de versions
  - Rapide, "simple", distribué, robuste pour des grands projets, …
- Stocke des instantanés (*snapshots*) et non des différences
  - Similaire à un instantané d'un mini système de fichiers

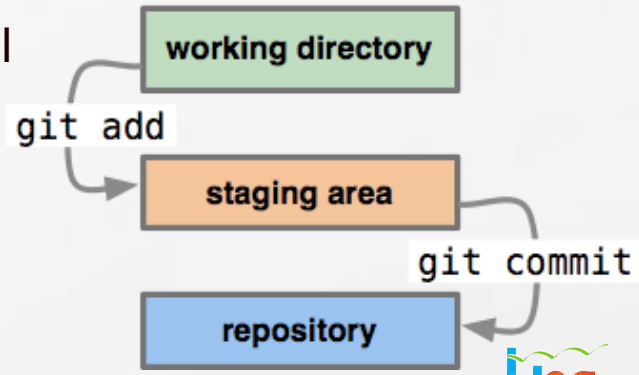




- Gère l'intégrité
  - Impossible de perdre des modifications ou des données
- Généralement, uniquement des ajouts de données (modifications, fichiers, etc)
  - La plupart des actions sont réversibles

Architecture de Git en local

- Trois états pour les fichiers et trois zones de stockage pour les données
  - Répertoire .git (*repository*): stocke les métadonnées et la base de données compressée des objets du projet
    - les fichiers validés / commande **commit**
  - Répertoire de travail (*working directory*): extraction sur le disque d'une version du projet sur laquelle on travaille
    - les fichiers modifiés
  - La zone d'index (*staging area*): simple fichier listant les modifications à intégrer dans le prochain instantané du projet
    - les fichiers indexés / commande **add**
- Utilisation standard de Git
  - Travailler/modifier dans le répertoire de travail
  - Ajouter le nom des fichiers modifiés dans l'index pour qu'ils soient intégrés lors de la prochaine validation
  - Valider/enregistrer un instantané dans la base de données du répertoire Git

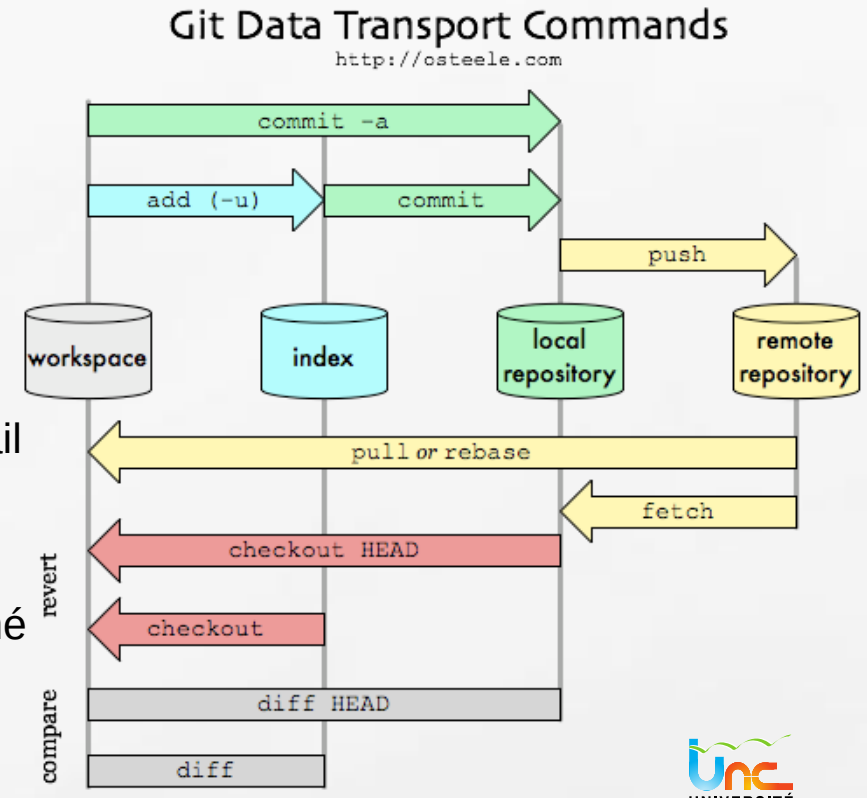


https://www.keblog.me/2015/01/bare-git-repository/

Architecture distribuée de Git

- Travailler dans un répertoire Git (*repository*) local
- Synchroniser le répertoire local avec un répertoire distant (*remote repository*)
  - p.ex. un dépôt distant sur GitHub

- Exemple de commandes
  - **add:** ajouter des modifications/fichiers à l'index (pour activer le suivi des versions)
  - **commit:** enregistrer l'instantané dans le répertoire Git local
  - **push:** envoyer un instantané validé (une branche) vers le serveur
  - **pull/rebase:** intégrer les modifications du dépôt distant dans son répertoire de travail
  - **fetch**: importer les modifications validées dans le répertoire Git local
  - **checkout**: changer de version d'instantané (ou restaurer une autre version)
  - **diff**: afficher les modifications avec le répertoire de travail



Démarrer un dépôt Git

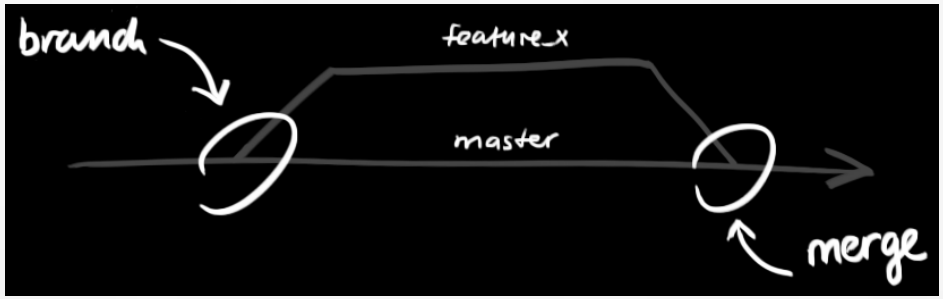
- Option 1: initialiser un dépôt Git dans un répertoire existant
  - **init** dans le répertoire du projet
    - squelette de dépôt, rien n'est encore versionné
  - **add** pour placer un ou des fichiers sous suivi de version
  - **commit** pour enregistrer une première version du projet dans le répertoire Git

- Option 2: cloner (**clone**) un dépôt existant (p.ex. GitHub)
  - Copie toutes les données (même l'historique) du dépôt existant dans un nouveau répertoire Git (p.ex. en local) et extraie une copie de la dernière version dans le répertoire de travail
  - Nom par défaut du serveur distant: **origin**

Les branches avec Git

- Créer une branche = "diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne" (Documentation Git)
  - Branche par défaut: **master**
    - **origin/master** pour la branche par défaut du dépôt distant
  - Branche courante pointée par **HEAD**

- Créer de nouvelles branches (**branch** *feature_X*) pour développer de nouvelles fonctionnalités ou faire des tests, puis fusionner (**merge**) avec la branche principale (**master**)
  - Attention: la création d'une nouvelle branche n'a pas basculé la copie de travail sur celle-ci

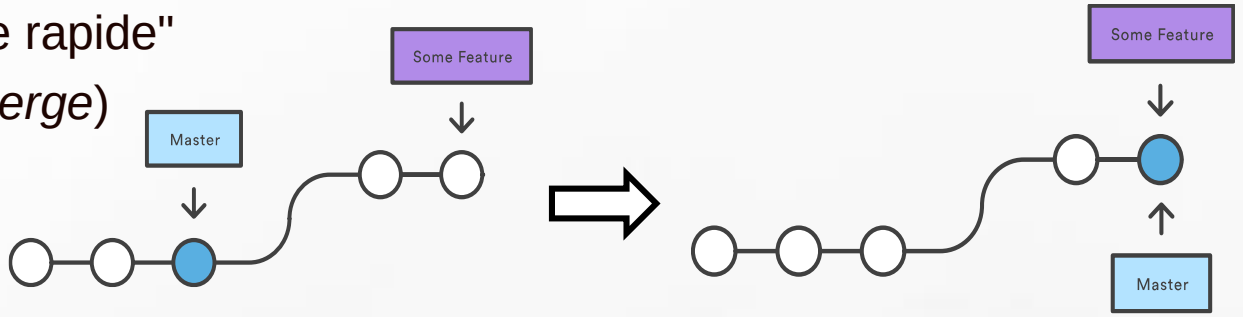


  - ➢ Changer de branche (**checkout** *feature_X*)
    - change le pointeur HEAD et l'instantané utilisé dans le répertoire de travail
  - Attention: la création d'une branche se fait "localement"
  - ➢ Faire un **push** pour la rendre accessible sur un dépôt distant

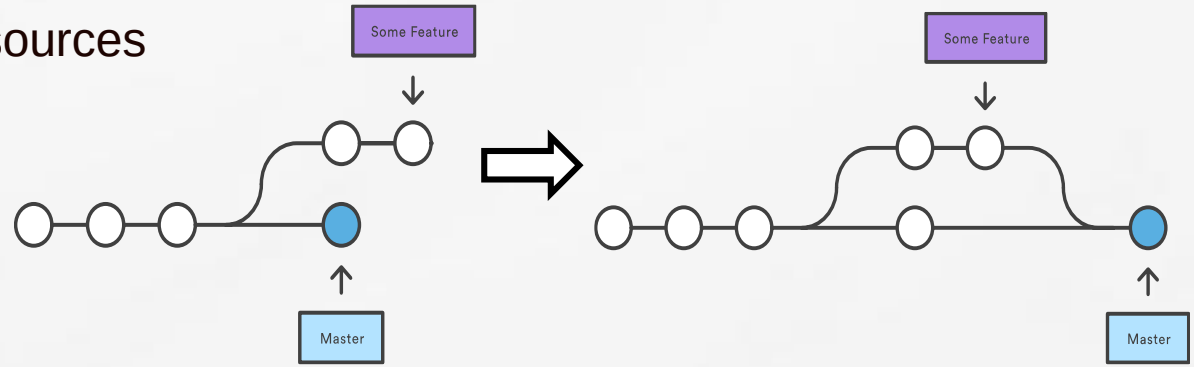
http://rogerdudler.github.io/git-guide/index.fr.html

Branches Git et fusion

- Deux types d'approches pour fusionner (**merge**) des branches
  - Fusion "avance rapide"
    (*fast-forward merge*)



    - avance simplement si pas d'autres branches (**merge)** ou déplace la base de la branche vers un autre commit (**rebase)**
    - ➢ adaptée pour des petites fonctionnalités, des résolutions de bug
  - Fusion à trois sources
    (*3-way merge*)



    - créé un nouvel instantané qui résulte de la fusion des 3 sources et le valide (*merge commit*)
    - ➢ adaptée pour l'intégration de fonctionnalités importantes
      - garde l'historique des branches

https://www.atlassian.com/git/tutorials/git-merge

Branches Git et fusion: conflits de fusion

- Erreur à la fusion si deux branches ont modifié la même partie d'un fichier
  - Processus de fusion mis en pause
  - Fichiers en conflits notés **unmerged**
  - Ajout dans le fichier de commentaires indiquant les conflits

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> prob53:index.html
```

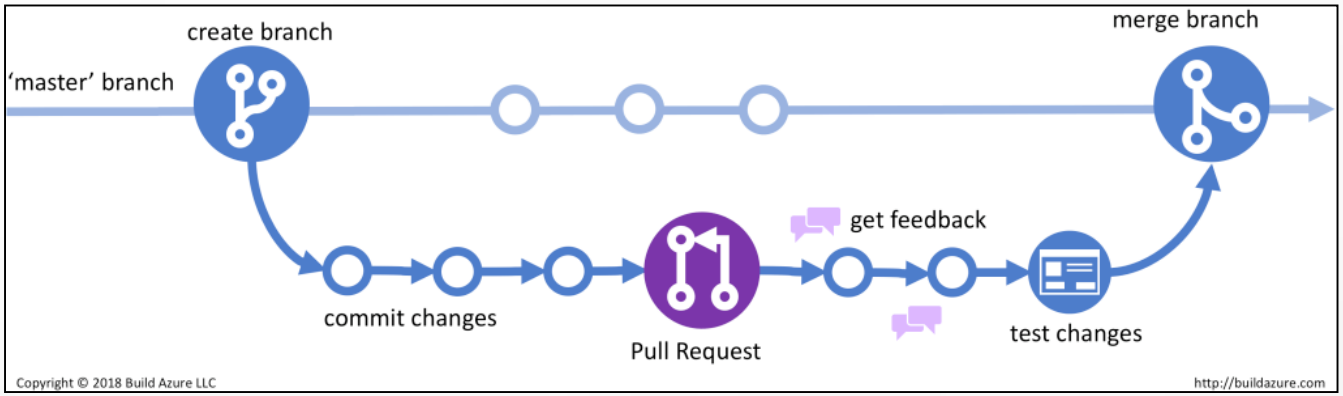
- Résolution:
  1. Corriger manuellement les erreurs dans le fichier et signaler les modifications en ajoutant celui-ci à l'index (**add**)

```
<div id="footer">
 please contact us at support@github.com
</div>
```

  1. Valider la fusion (**commit**)

Les *workflow* Git

- "L'outil n'est pas la méthode" (OVH UX Labs) ➔ différents *workflow* possibles
- Le *Github flow :* le plus simple

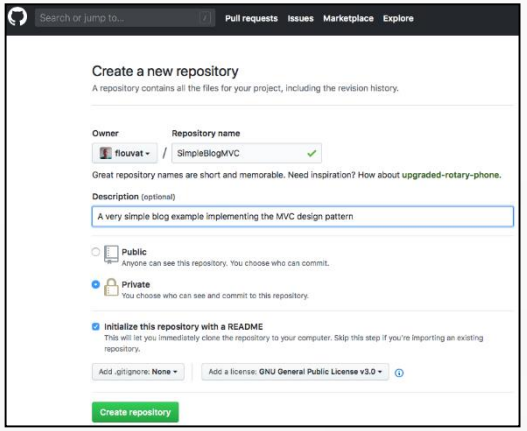


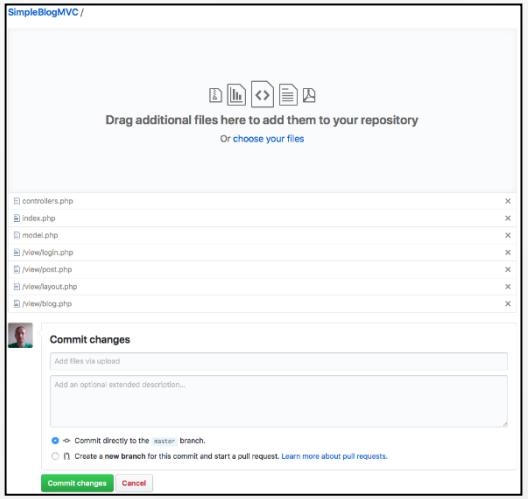
  - Tout ce qui est sur master est stable et déployable
  - Pour travailler sur quelque-chose, créer une branche avec un nom significatif
    - p.ex. *feature/add-menu…*
  - Valider la branche localement après chaque changement (**commit** avec un message associé) et régulièrement pousser (**push**) sur une branche du même nom sur le serveur
  - Une fois le développement terminé, envoyer une notification (**pull request)** pour recueillir des retours et des tests
    - demande à d'autres développeurs d'intégrer/tester dans leur *repository* la branche créée
  - Une fois les tests réalisés par les collaborateurs, fusionner (**merge**) sur master

https://buildazure.com/2018/02/21/introduction-to-git-version-control-workflow/

Exemple: SimpleBlogMVC, GitHub et PhpStorm

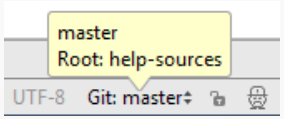
- Inscription à GitHub: https://education.github.com/pack
- Créer un nouveau *repository* **privé** dans GitHub
- Uploader les codes sources déjà implémentés dans GitHub
- Enregistrer le compte GitHub dans PhpStorm https://www.jetbrains.com/help/phpstorm/manage-projects-hosted-on-github.html
  - *Settings* > *Version Control* > *GitHub* > *add account* et entrer vos identifiants GitHub
- Cloner le dépôt GitHub dans un nouveau projet PhpStorm (en local)
  - (fermer les projets ouverts)
  - dans la fenêtre d'accueil de PhpStorm, cliquer sur *Get from Version Control*
  - cliquer sur l'onglet GitHub et sélectionner votre projet
  - Cliquer sur *Clone*





Exemple: SimpleBlogMVC, GitHub et PhpStorm

- Le GitHub flow dans PhpStorm https://www.jetbrains.com/help/phpstorm/using-git-integration.html
  - Créer d'une nouvelle fonctionnalité/branche (**branch + checkout**)
    - VCS > Git > Branches > New branch



  - Ajouter un fichier sous suivi de version (**add)**
    - VCS > Git > Add
  - Valider localement les changements et envoyer les fichiers sur le dépôt distant (**commit + push**)
    - VCS > Git > Commit File
    - Mettre un message indiquant la nature des modifications
    - Commit and Push

  - Envoyer une notification aux autres membres du projet pour qu'ils puissent intégrer et tester la nouvelle fonctionnalité (**pull request**)
    - VCS > Git > Create Pull Request
    - Mettre « master » en *Base Branch*

  - Tester, modifier et échanger sur le nouveau code
  - Publier sa fonctionnalité dans la branche master (**checkout + pull**)
    - VCS > Branches > Local Branches > master > checkout
    - Sélectionner VCS > Git > Pull > Branches to merge > origin/master
    - Sélectionner VCS > Git > Pull > Branches to merge > origin/nouvelle branche
    - VCS > Git > Push

- Mettre à jours son répertoire local (**fetch + merge**)
  - VCS > Update