

CÔTE D'AZUR


GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Génie Logiciel et UML

Introduction à GIT

Département Génie Biologique
GB5-BIMB – année 2023–2024



Jean-Paul Comet / Guillaume Grataloup

Université Côte d'Azur

19 septembre 2023

1/52

CÔTE D'AZUR

Organisation de l'enseignement

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

- 15h de Cours + 5h de TDs + 10h de suivi de projet
- un projet en commun avec l'UE *BD avancées et Interfaces*
- Intervenants :

Gilles Bernot	Gilles.Bernot@univ-cotedazur.fr
Romain Michelucci	rmichelucci@i3s.unice.fr
Guillaume Grataloup	grataloup@i3s.unice.fr
- Evaluation : sur projet réalisation + rapports + soutenances + fonctionnement du groupe
- Outils mis en œuvre :
 - XAMPP (mise en place d'un serveur de BD avec interface web)
 - GIT (logiciel de gestion de versions)
 - HTML/CSS (pages web)
 - PHP (des pages web qui interrogent une BD via phpMyAdmin)
 - UML (pour la conception)
- support de cours : <https://www.i3s.unice.fr/~comet/SUPPORTS/> □

2/52

CÔTE D'AZUR

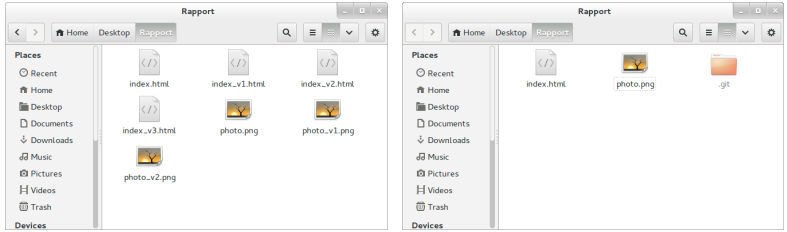
Problématiques des projets à plusieurs

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Si vous avez déjà travaillé en commun sur un projet avec support numérique (éventuellement projet personnel), vous avez certainement déjà rencontré un de ces problèmes :

- Qui a modifié le fichier X, il marchait bien avant et maintenant il provoque des bugs !;
- Robert, tu peux m'aider en travaillant sur le fichier X pendant que je travaille sur le fichier Y ? Attention à ne pas toucher au fichier Y car si on travaille dessus en même temps je risque d'écraser tes modifications !
- Qui a ajouté cette ligne de code dans ce fichier ? Elle ne sert à rien !
- À quoi servent ces nouveaux fichiers et qui les a ajoutés au code du projet ?
- Quelles modifications avons-nous faites pour résoudre le bug de la page qui se ferme toute seule ? >>



Après avoir échangé des versions d'un fichier nommées X_{1.1}.doc, X_{1.2}.doc, X_{1.3}.doc ..., on a reçu d'un contributeur un fichier nommé X_{1.0}.doc ...

4/52

CÔTE D'AZUR

Qu'est-ce que Git ? – gestionnaire de version

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

- La gestion des versions est un travail fastidieux et méthodique.
- Les humains ne sont pas doués pour les travaux fastidieux et méthodiques.
- Laissons cela à l'ordinateur, et concentrons-nous sur la partie du travail où nous sommes meilleurs que l'ordinateur.

Système de contrôle de version (version control system)

- Un ensemble d'outils logiciels pour :
 - Mémoriser.
 - Faciliter le travail collaboratif.
 - Retrouver différentes version d'un projet.
- Initialement développé par Linus Torvalds pour faciliter le développement du noyau Linux.
 - Logiciel libre / open source.
 - Disponible sur toutes les plates-formes.
- travail collaboratif :
 - 1 mémorisation de chaque modification de chaque fichier avec une justification. *Qui a écrit chaque ligne de code de chaque fichier et dans quel but ?*
 - 2 si 2 personnes travaillent simultanément sur un même fichier, fusion des modifications quasi automatique, pour éviter que le travail de l'un ne soit écrasé.

5/52

Logiciels centralisés et distribués

GL & UML

J.-P. Comet / G. Grataloup

Introduction

Installation

Fonctionnement

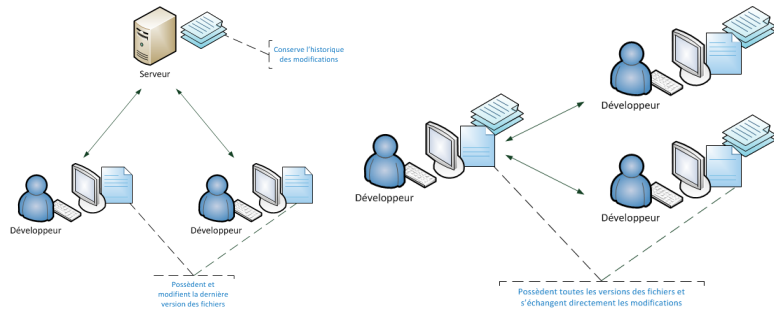
Historique

Branches L.

Dépôts

Branches P

Autres



- Logiciel de gestion de versions centralisé.
- Logiciel de gestion de versions distribué. Il n'y a pas de serveur. Les développeurs conservent l'historique des modifications et se transmettent les nouveautés.
- Logiciel de gestion de versions distribué avec serveur. Le serveur sert de point de rencontre entre les développeurs et possède lui aussi l'historique des versions.

GL & UML

J.-P. Comet / G. Grataloup

Introduction

Installation

Fonctionnement

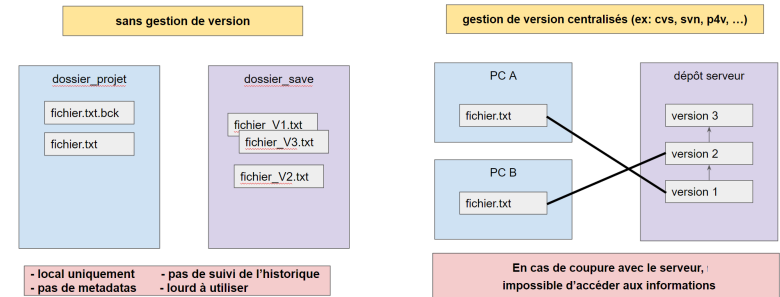
Historique

Branches L.

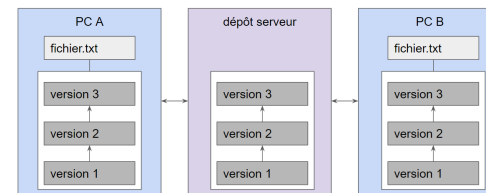
Dépôts

Branches P

Autres



gestion de version distribués (ex: git, Mercurial, Bazaar, ...)



Copie en local de l'intégralité du dépôt

Les différents logiciels de gestion de versions

GL & UML

J.-P. Comet / G. Grataloup

Introduction

Installation

Fonctionnement

Historique

Branches L.

Dépôts

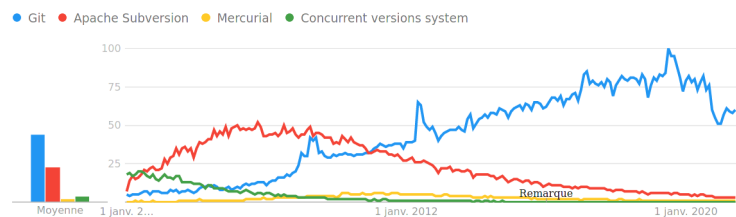
Branches P

Autres



- | | | |
|----------------------|----------------------|-------------------------|
| ● libre | ● licence GNU / GPL | ● libre |
| ● décentralisé | ● décentralisé | ● centralisé |
| ● branches (+) | ● branches (+) | ● branches (-) |
| ● cryptage SHA1 | ● cryptage SHA1 | ● pas de cryptage |
| ● métadonnées (.git) | | ● stockages de fichiers |
| ● 14M. users | ● m principe que Git | |
| ● Gitlab | | |
| ● grosse communauté | | |

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



Installer git sous Linux

GL & UML

J.-P. Comet / G. Grataloup

Introduction

Installation

Fonctionnement

Historique

Branches L.

Dépôts

Branches P

Autres

Avec un gestionnaire de paquets, c'est très simple :

```
>>> sudo apt-get install git-core gitk
```

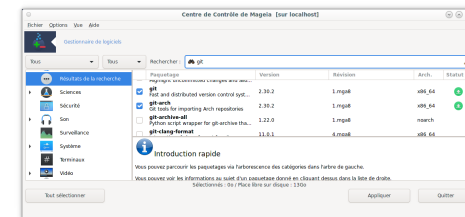
- git-core : c'est git, tout simplement. C'est le seul paquet vraiment indispensable ;
- gitk : une interface graphique qui aide à mieux visualiser les logs. Facultatif.

Il est recommandé de se créer une paire de clés pour se connecter au serveur de rencontre lorsque l'accès à git se fait via SSH. Un tutoriel du SdZ explique comment créer des clés :

<https://openclassrooms.com/fr/courses/>

43538-reprenez-le-contrôle-a-laide-de-linux/

41773-la-connexion-securisée-a-distance-avec-ssh#ss_part_5

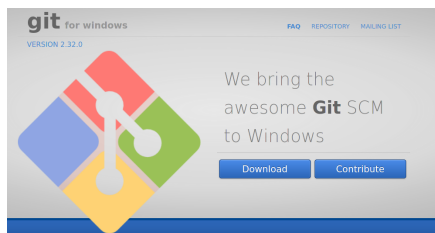


Pour utiliser git sous Windows, il faut installer msysgit :
<https://gitforwindows.org/>

Cela installe msys (un système d'émulation des commandes Unix sous Windows) et git simultanément.

Ce n'est pas une « vraie » version pour Windows mais plutôt une forme d'émulation.

- lancer une console pour pouvoir utiliser git : programme git Bash.
- commandes de base d'Unix disponibles : cd, pwd, mkdir, etc.



1 Avec Homebrew :

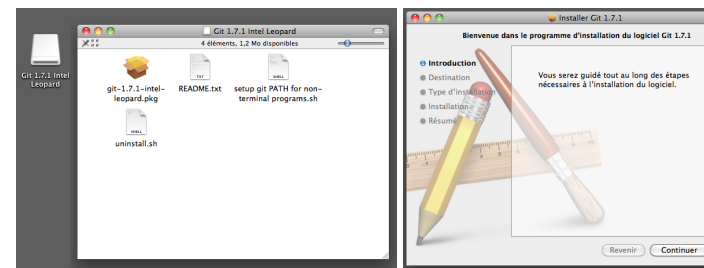
- Installer homebrew : `>>> /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
- puis `>>> brew install git`

2 Avec Xcode : git est fourni avec Xcode, une suite de logiciels pour développeurs.

3 Avec un paquet fourni par Tim Harper :

<https://sourceforge.net/projects/git-osx-installer/>

- télécharger une archive .dmg.
- installer git en double-cliquant sur le .dmg.
- ouvrir un terminal pour avoir accès aux commandes git.



1 Dans la console, commencez par envoyer ces trois lignes :

```
git config --global color.diff auto
git config --global color.status auto
git config --global color.branch auto
```

Elles activeront la couleur dans git. Il ne faut le faire qu'une fois, et ça aide à la lisibilité des messages dans la console.

2 configurer votre nom (ou pseudo) :

```
git config --global user.name "votre_pseudo"
```

3 Puis votre e-mail :

```
git config --global user.email "moi@email.com"
```

4 configurer l'éditeur que git vous présentera pour rédiger les messages de commit (ici, je propose emacs, mais on peut aussi choisir d'autres alternatives comme vim, sublimetext) :

```
git config --global core.editor emacs
```

1 Éditer votre fichier de configuration .gitconfig (répertoire personnel) pour y ajouter une section alias à la fin :

```
emacs /.gitconfig
```

Mon fichier /.gitconfig :

```
[color]
diff = auto
status = auto
branch = auto

[user]
name = votre_pseudo
email = moi@email.com

[alias]
ci = commit
co = checkout
st = status
br = branch

[core]
editor = emacs
```

2 On peut ajouter à la configuration l'alias git hist qui sera très pratique :

```
> git config --global alias.hist "log --pretty=format:'%h - %ad: %s [%an]' --date=short"
```

CÔTE D'AZUR

Un dépôt

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Un dépôt git est un répertoire (ou dossier) dans lequel toute modification peut être suivie dans le temps.

- n'importe quel répertoire peut être un dépôt git.
Tous les sous-répertoires feront partie du dépôt.
- toute commande git cherche à savoir s'il se trouve dans un dépôt ou non.
il teste la présence d'un sous-répertoire caché `.git/` en remontant l'arborescence.
- Création d'un dépôt :


```
>>> cd /home/comet/GIT/
>>> mkdir monPremierDepot
>>> cd monPremierDepot
```

 Si votre projet existe déjà, inutile d'en créer un nouveau. Enfin, initialisez le dépôt git avec la commande :


```
>>> git init
```

 Un dossier caché `.git` vient tout simplement d'être créé.

16/52

CÔTE D'AZUR

Représentation interne du dépôt

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

tout fichier peut être dans l'un des 3 états possibles suivants :

- modifié** : Un fichier dans cet état a été localement modifié, mais git n'a pas validé les différences dans sa base de données locale. Les fichiers dans cet état sont dans le répertoire de travail (ou working directory).
- indexé** : Un fichier dans cet état fait partie des modifications qui sont prêtes à être validées pour le prochain instantané. Les fichiers dans cet état sont dans l'index (ou dans le staging area).
- validé** : Les données dans cet état sont en sécurité dans la base de donnée locale de git (aussi appelée git directory). Les modifications ainsi sauvegardées font partie d'un instantané du projet (appelé aussi commit).

On peut faire un bilan de l'état de chaque fichier dans un dépôt avec la commande status :

```
>>> git status
```

```
>>> git add <nom_fichier>
```

```
>>> git commit
```

17/52

CÔTE D'AZUR

Les instantanés du projet

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

git enregistre dans une BD locale une succession d'instantanés (ou commits). Un instantané a plusieurs caractéristiques fondamentales :

- unicité : Un instantané est unique et identifié par un numéro de série (par exemple `f31a288e90dae7712f49b061a56486f6d489a657`)
- auteur : Un instantané est lié à un auteur (nom + adresse mail)
- date : Un instantané est horodaté
- commentaire : Un instantané est obligatoirement accompagné d'un commentaire de son auteur.

- `git log` : Lister les instantanés.
- `git commit` : Créer un instantané.
- Commenter un instantané. La commande `commit` ouvre une fenêtre d'édition dans laquelle l'auteur va écrire le commentaire attaché à son `commit`. Voici les règles courantes implicites d'un bon message de `commit` :
 - Un titre (maximum 80 caractères)
 - Un saut de ligne
 - Des détails sur les modifications de ce `commit`.
 L'auteur, la date et l'identifiant du `commit` sont automatiquement ajoutés par git.

18/52

CÔTE D'AZUR

Les instantanés du projet

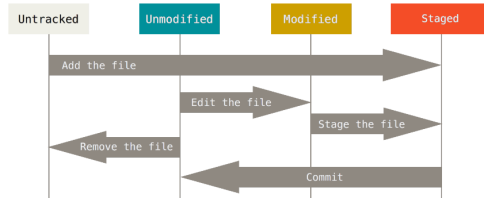
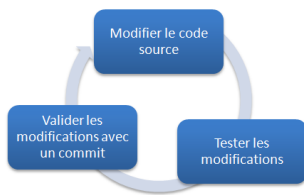
GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Remarques.

- possible de faire un message de `commit` sur plusieurs lignes.
 - première ligne : description synthétique
 - Si pas de message de `commit` celui-ci sera annulé.
- Si impossible de résumer vos changements en 1 ligne :
vous avez passé trop de temps à faire des changements sans « commiter ».
Exemple de messages de `commit` d'une ligne corrects :
 - Améliore la visibilité des post-it sur le forum.
 - Simplifie l'interface de changement d'avatar.
 - Résout #324 : bug qui empêchait de valider un tutoriel à plusieurs.
 Utiliser le présent – Plutôt en anglais.
- Un `commit` avec git est local : personne n'est au courant.
Avantage : si un `commit` est erroné, possibilité de l'annuler

18/52



```
>>> git diff
```

```
diff --git a/src/Symfony/Components/Yaml/Yaml.php b/src/Symfony/Components/Yaml/
index fa0b806..77f9902 100644
--- a/src/Symfony/Components/Yaml/Yaml.php
+++ b/src/Symfony/Components/Yaml/Yaml.php
@@ -19,7 +19,7 @@ namespace Symfony\Components\Yaml;
 /*
  class Yaml
  {
-   static protected $spec = 1.2 ;
+   static protected $spec = 1.3 ;
  /**
   * Sets the YAML specification version to use.
@@ -33,6 +33,8 @@ class Yaml
     if (!in_array($version, array( 1.1 , 1.2 ))) {
         throw new \InvalidArgumentException(sprintf( Version %s of the YAML
     )
+
+   $mtsource = $version;
+   self::$spec = $version;
  }
```

Pour ignorer certains fichiers (*. *~ par exemple) :

- fichier `.gitignore` :
chaque ligne : expression régulière qui spécifie des fichiers à ignorer.
- si on veut que git ignore :
 - tous les fichiers finissant par « ~ »,
 - tous les fichiers d'un sous-répertoire `log`
on va créer le fichier `.gitignore` afin qu'il contienne :


```
*~
log/*
```

Attention, si les fichiers sont déjà dans le suivis, ils continueront de l'être.

- `>>> git log` : Consulter l'historique des commits

```
commit 227653fd243498495e4414218e0d4282eef3876e
Author: Fabien Potencier <fabien.potencier@gmail.com>
Date:
```

```
Thu Jun 3 08:47:46 2021 +0200
```

```
[TwigBundle] added the javascript token parsers in the helper exten
```

```
commit 6261cc26693fa1697bcbbd671f18f4902bef07bc
```

```
Author: Jeremy Mikola <jmikola@gmail.com>
```

```
Date:
```

```
Wed Jun 2 17:32:08 2021 -0400
```

```
Fixed bad examples in doctrine:generate:entities help output.
```

- `>>> git log -p` : détail des lignes qui ont été ajoutées et retirées dans chaque commit,
- `>>> git log --stat` : résumé plus court des commits

Si vous avez fait une faute d'orthographe dans votre dernier message de commit ou que vous voulez tout simplement le modifier, vous pouvez le faire facilement grâce à la commande suivante :

```
>>> git commit --amend
```

L'éditeur de texte s'ouvrira à nouveau pour changer le message.

Cette commande est généralement utilisée juste après avoir effectué un commit lorsqu'on se rend compte d'une erreur dans le message. Il est en effet impossible de modifier le message d'un commit lorsque celui-ci a été transmis à d'autres personnes.

CÔTE D'AZUR Corriger un commit

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Supposons que notre index n'était pas complet au moment du dernier commit (oubli d'ajouter un fichier).

- première étape : compléter notre index (`git add`)
- amender le précédent commit :

```
>>> git commit --amend
```

Le message du dernier commit apparaît, ce qui nous permet de le changer, et on voit dans la liste des changements les nouvelles modifications apportées par l'index. Ainsi, le workflow complet est le suivant :

```
>>> git commit -m 'validation initiale'
>>> git add fichier.oublie
>>> git commit --amend
```

23/52

CÔTE D'AZUR Annuler le dernier commit (soft)

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Si vous voulez annuler votre dernier commit :

```
>>> git reset HEAD^
```

Cela annule le dernier commit et revient à l'avant-dernier. Pour indiquer à quel commit on souhaite revenir, il existe plusieurs notations :

- HEAD : dernier commit ;
- HEAD^ : avant-dernier commit ;
- HEAD^^ : avant-avant-dernier commit ;
- HEAD~2 : avant-avant-dernier commit (notation équivalente) ;
- d6d98923868578a7f38dea79833b56d0326fcb1 : indique un numéro de commit précis ;
- d6d9892 : indique un numéro de commit précis (notation équivalente à la précédente, bien souvent écrire les premiers chiffres est suffisant tant qu'aucun autre commit ne commence par les mêmes chiffres).

Seul le commit est retiré de git ; vos fichiers, eux, restent modifiés. Vous pouvez alors à nouveau changer vos fichiers si besoin et refaire un commit.

24/52

CÔTE D'AZUR Annuler les changements du dernier commit (hard)

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Si vous voulez annuler votre dernier commit et les changements effectués dans les fichiers, il faut faire un reset hard.

- annulera sans confirmation tout votre travail!!!

```
>>> git reset --hard HEAD^ # Annule les commits / perd tous les changements
```

Normalement, git devrait vous dire quel est maintenant le dernier commit qu'il connaît (le nouveau HEAD) :

```
>>> git reset --hard HEAD^
```

HEAD is now at 6261cc2 Fixed bad examples in doctrine:generate:entities help output.

25/52

CÔTE D'AZUR Désindexer un fichier indexé

GL & UML
J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Dans le cas où on a indexé plusieurs modifications (avec la commande `git add`), imaginons qu'on souhaite retirer un des fichiers indexés pour ne pas qu'il soit intégré au prochain instantané.

```
>>> git status
```

Fichiers non suivis:
(utilisez "git add <fichiers>" pour inclure dans ce qui sera validé)

```
plonegroup.rb
schema.rb
```

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez "git add" pour les suivre)

```
>>> git add .
>>> git status
```

Modifications qui seront validées :
(utilisez "git reset HEAD <fichier>..." pour désindexer)

```
nouveau fichier : plonegroup.rb
nouveau fichier : schema.rb
>>> git reset HEAD schema.rb
>>> git status
```

Modifications qui seront validées :
(utilisez "git reset HEAD <fichier>..." pour désindexer)

```
nouveau fichier : plonegroup.rb
```

Fichiers non suivis:
(utilisez "git add <fichiers>" pour inclure dans ce qui sera validé)

```
schema.rb
```

Avec la commande `git reset HEAD <unfichier>` on peut donc extraire un fichier de l'index.

26/52

Annuler les modifications locales

GL & UML

J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

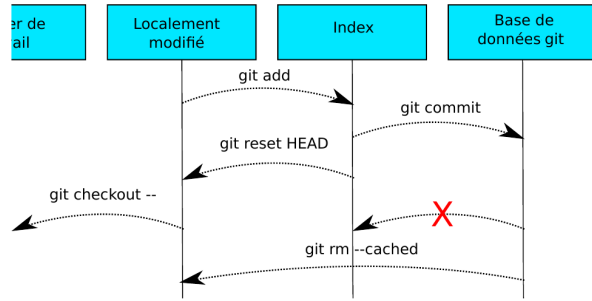
Pour annuler toutes les modifications faites à un fichier qui n'ont pas encore été ajoutées à l'index, la commande est :

```
>>> git checkout -- monfichier
```

ou

```
>>> git restore monfichier
```

Il est important de noter que cette commande est destructive ! En l'utilisant vous retrouvez votre fichier dans l'état de son dernier commit, les modifications non indexées ont été supprimées pour de bon. Cette commande n'a aucun effet si le fichier est déjà ajouté à l'index.



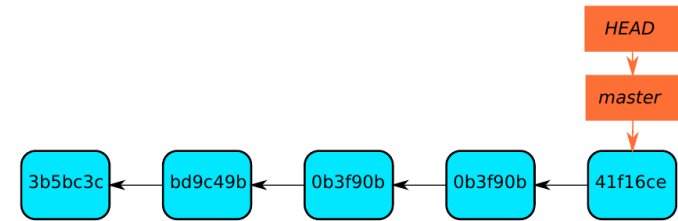
Représentation de l'historique

GL & UML

J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Les instantanés successifs peuvent être représentés comme formant une chaîne dans le temps.



- HEAD est un pointeur utilisé par git pour référencer le commit courant.
- Le terme master représente un autre pointeur sur le commit courant de la branche «master».

Se balader dans l'historique

GL & UML

J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Remonter dans l'historique ≡ déplacer le pointeur HEAD
On déplace le pointeur HEAD grâce à la commande `git checkout`, qui prend en paramètre le numéro de série de l'instantané ciblé :

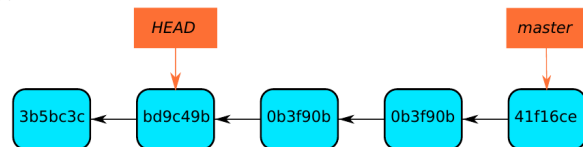
```
>>> git hist
* 7632086 2021-08-20 | Commit supplémentaire (HEAD, master)
* 2f051e6 2021-08-18 | Suppression du fichier [Jonathan Schaeffer]
* bf67c41 2021-08-18 | Premier commit [Jonathan Schaeffer]
```

```
>>> git checkout bf67c41
Note: checking out '2f051e6'.
```

```
>>> git hist
* bf67c41 2021-08-18 | Premier commit (HEAD) [Jonathan Schaeffer]
```

On est remontés dans le temps.

Notez que le message affiché par git lors du checkout explique la situation «detached HEAD».



Mais où sont passés tous les commits postérieurs ? Remarquez que, dans l'affichage des logs, on ne voit plus apparaître le pointeur master. Pour retrouver notre projet complet, il faut retourner à master :

```
>>> git checkout master
```

Annuler le dernier commit

GL & UML

J.-P. Comet / G. Grataloup

Introduction
Installation
Fonctionnement
Historique
Branches L.
Dépôts
Branches P
Autres

Ne pas réécrire l'histoire...

Si on souhaite annuler le dernier commit, par exemple, la meilleure pratique est de créer un nouveau commit qui va inverser toutes les modifications du dernier instantané.

```
>>> git revert HEAD
création d'un nouveau commit qui annule les modifications enregistrées dans le dernier commit.
>>> git revert HEAD
[master f061f93] Revert "Commit supplémentaire"
2 files changed, 107 deletions(-)
delete mode 100644 plonegroup.rb
delete mode 100644 schema.rb
>>> git hist
* f061f93 2021-08-20 | Revert "Commit supplémentaire" (HEAD, master)
* 7632086 2021-08-20 | Commit supplémentaire [Jonathan Schaeffer]
* 2f051e6 2021-08-18 | Suppression du fichier [Jonathan Schaeffer]
* bf67c41 2021-08-18 | Premier commit [Jonathan Schaeffer]
```

Et nous nous retrouvons dans le même état que le commit 2f051e6.

Lorsqu'on crée un nouveau commit :

- git crée l'instantané avec son numéro de référence,
- déplace le pointeur HEAD sur ce dernier commit.
- déplace aussi le pointeur de la branche courante (master par défaut) sur ce dernier commit.

Pour étiqueter des versions importantes (stables, diffusables...) :
git permet de mettre un pointeur personnalisé, appelé tag, sur n'importe quel instantané :

```
>>> git tag version-1.0
```

```
>>> git hist
```

```
* f061f93 2021-08-20 | Revert ... (HEAD, tag: version-1.0, master)
```

On peut ensuite retourner sur un tag passé avec la commande checkout.