Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Introduction au système de contrôle de version Git

Formation continue — Université Lille
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille.fr

Licence trimestre 3 — 2020-21

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Un système de contrôle de version est un ensemble d'outils logiciels permettant de :

- mémoriser et retrouver différentes version d'un projet ;
- faciliter le travail collaboratif multi-plateforme.

Initialement développé par Linus Torvald pour l'évolution du noyau Linux, l'outil dominant est actuellement (2020) Git. C'est :

- un outil décentralisé ;
- sous licence GNU GPL a.k.a. logiciel libre open source ;
- disposnible sous toutes les plateformes.

git (plural gits) : (Britain, slang, derogatory) A silly, incompetent, stupid, annoying or childish person (usually a man).

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Git est incontournable pour le développement logiciel mais aussi pour l'écriture (articles, pages web statiques, etc).
Pour les fichiers "binaires" utilisez git-lfs (large file storage).

Définition

Une version est le contenu d'un projet à une étape de son cycle de vie.
Un dépôt est l'historique du projet contenant toutes ses versions.
Une branche est la variante d'un projet.

Git permet :

- d'enregistrer les versions du projet dans un dépôt et de retrouver une version spécifique ;
- de travailler sur différentes branches et de les fusionner ;
- de partager les versions et principalement de fusionner des versions faites par d'autres.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Les commandes de git sont des filtres shell du type :

git “command”.

Par exemple l'aide en ligne s'invoque comme suit

```
% git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

Autre exemple pour voir et manipuler les variables de git :

```
% git config -l
user.name=Sedoglavic Alexandre
user.email=alexandre.sedoglavic@univ-lille1.fr
```

Pour (ré)initialiser un dépôt local à partir d'un répétoire du système de fichier, on utilise :

```
% git init
Initialized empty Git repository in /tmp/essai/.git/
```

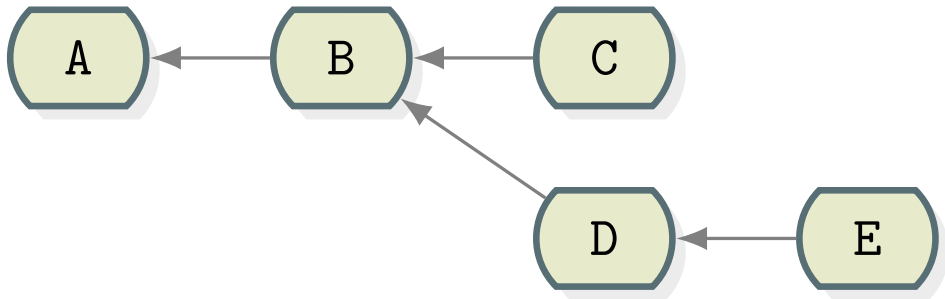
Les commandes sont bavardes et en lisant la sortie standard, on sait généralement ce qu'il convient de faire.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Un *commit* est un nœud — codant une version — d'un graphe acyclique orienté — codant le dépôt.

- il représente un *ensemble* de modifications aux constituants — fichiers et répertoires — du dépôt. Un commit peut donc synthétiser un grand nombre de modification sur beaucoup de fichiers ou une seul modification sur un seul fichier.
- il est identifié de manière *unique* par un *hash code* (actuellement basé sur la fonction de hachage SHA-1).

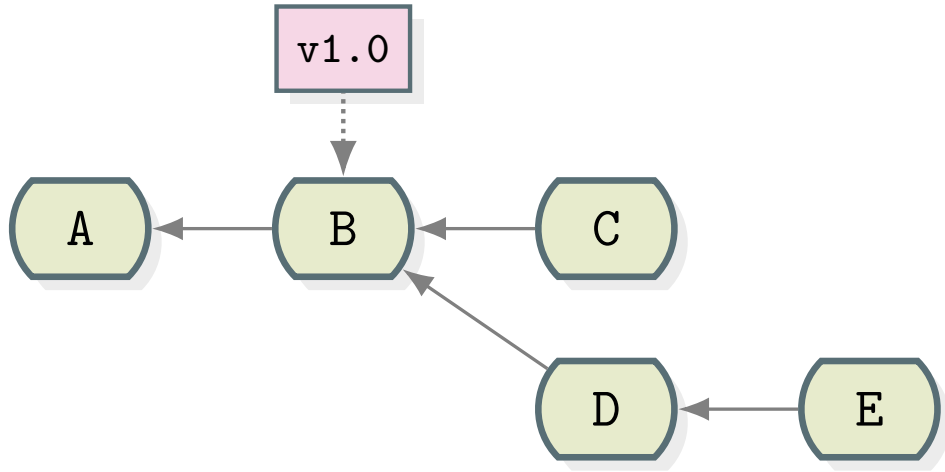
Un arc va d'un commit B à un commit A si B est obtenu à partir d'une modification sur A.



Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Nommer un commit par un nombre hexadécimal est malcommode.

Un *tag* est une étiquette associé à un commit et permet de pointer sur des versions que l'utilisateur veut distinguer.

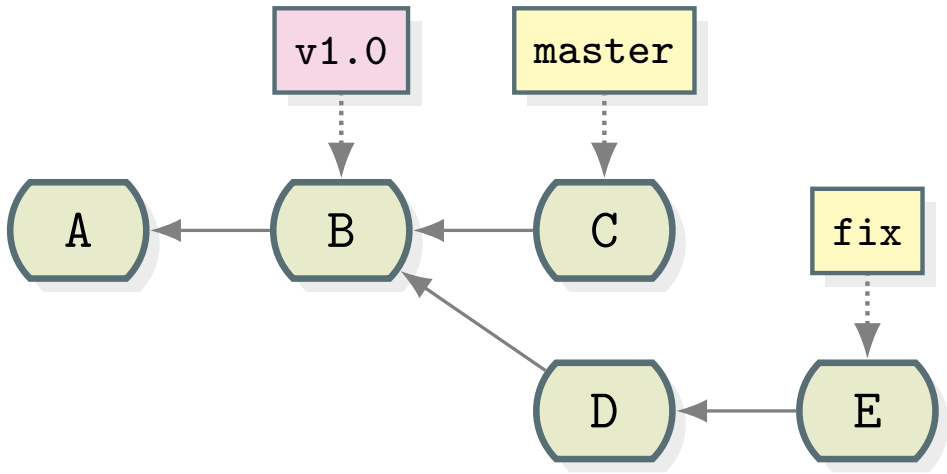


Ainsi, le tag est un alias pour le hash hexadécimal d'un commit.
On peut utiliser l'un ou l'autre dans les commandes git.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Une *branche* est un pointeur sur un commit.

Git définit une branche courante qui — contrairement au tag — est mise à jour à chaque fois qu'un nouveau commit est inclus dans le dépôt et donc pointe sur ce commit.

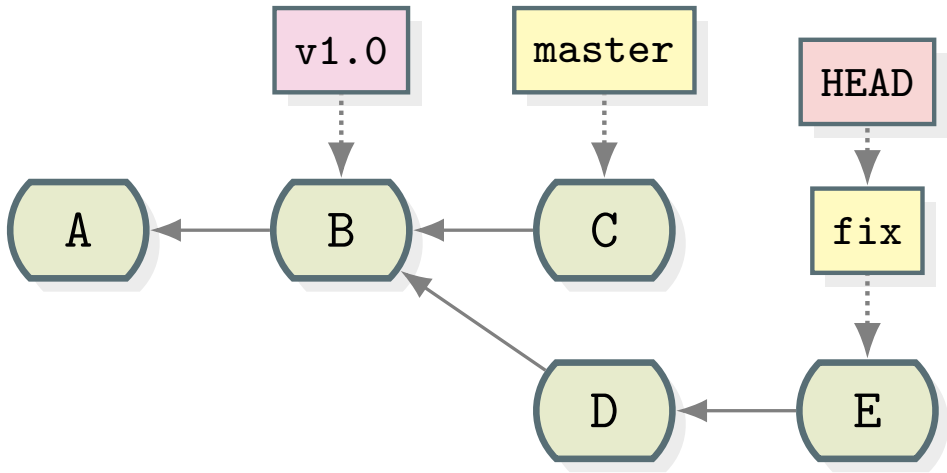


La branche par défaut est nommée master.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Le pointeur HEAD indique l'état courant du répertoire de travail.

Il est mis à jour à chaque nouveau commit inclus dans le dépôt.



Nous verrons que l'on peut déplacer cette tête ce qui différencie cette notion de la branche courante.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

git log

La commande *git log* permet d'observer la branche portant la tête.

```
% git log
commit 94388216778ae08a79d234ac5ee180e41a4bb166 (HEAD -> fix)
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sat Aug 8 12:14:34 2020 +0200

    <feat> step E

commit f5a20cce1e8b781438d9f873a9cbbbc1e2ebf481
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sat Aug 8 12:14:07 2020 +0200

    <feat> step D

commit 65bfe36af77337a175600a5bf30b266a60235eaa (tag: v1.0)
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sat Aug 8 12:08:11 2020 +0200

    <feat> step B

commit 73512bbdb3072281f2c7e160a6d5d610a379c52d
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sat Aug 8 12:07:30 2020 +0200

    <feat> step A
```

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

On peut aussi observer une autre branche

```
% git log master
commit fad4af34ff3bea172aac669f5ab17fa11ec3b72c (master)
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sat Aug 8 12:09:01 2020 +0200

    <feat> step C

commit 65bfe36af77337a175600a5bf30b266a60235eaa (tag: v1.0)
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sat Aug 8 12:08:11 2020 +0200

    <feat> step B

commit 73512bbdb3072281f2c7e160a6d5d610a379c52d
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sat Aug 8 12:07:30 2020 +0200

    <feat> step A
```

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Tracer et visualiser les modifications

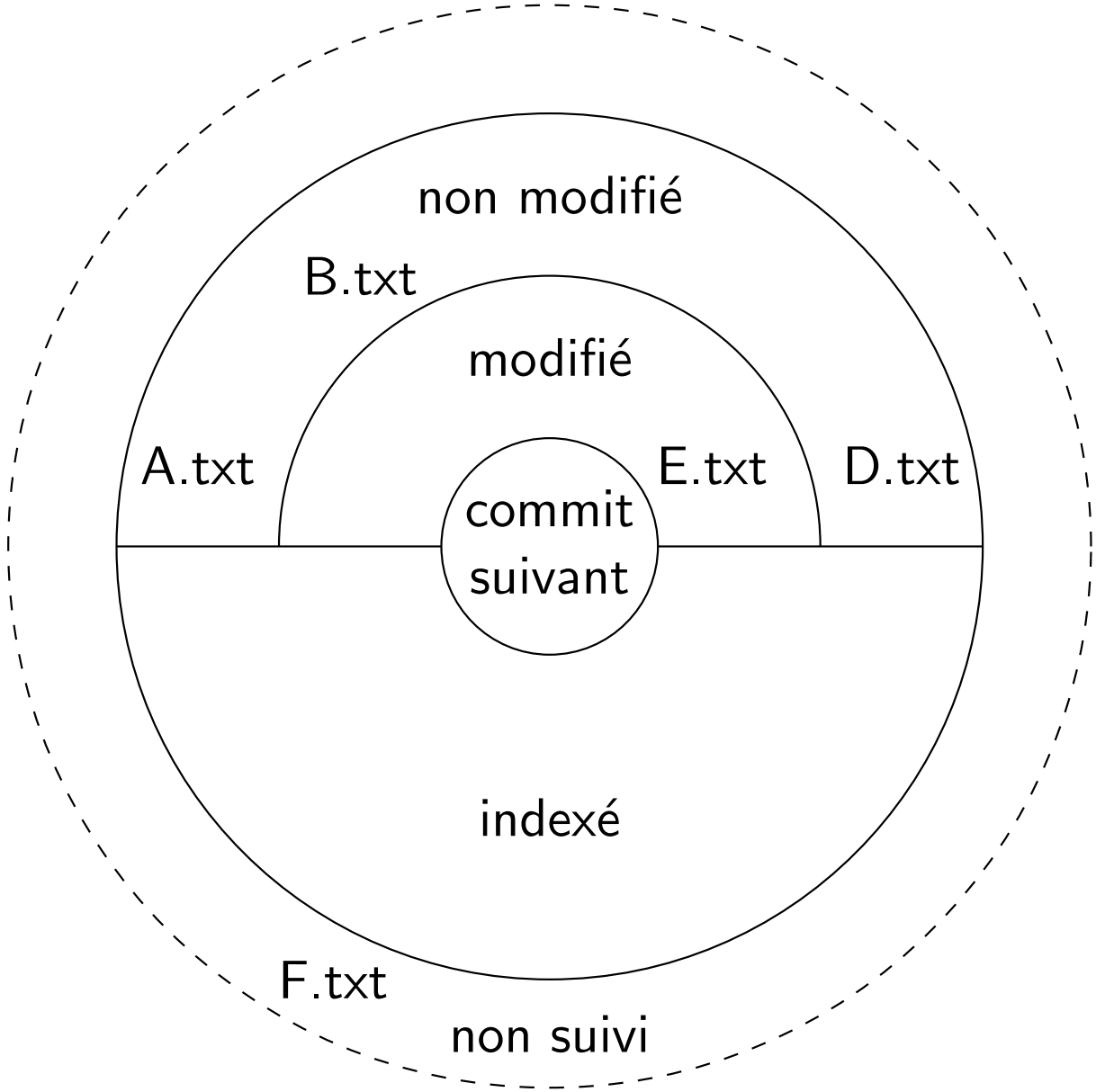
La commande git status permet de vérifier l'état des fichiers.

```
% ls -rt
A.txt B.txt D.txt F.txt E.txt

% git status
On branch fix
Changes not staged for commit: (use "git add <file>..." to update what will
be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified:   E.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

F.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

git diff permet de voir précisément ce qui a été modifié :

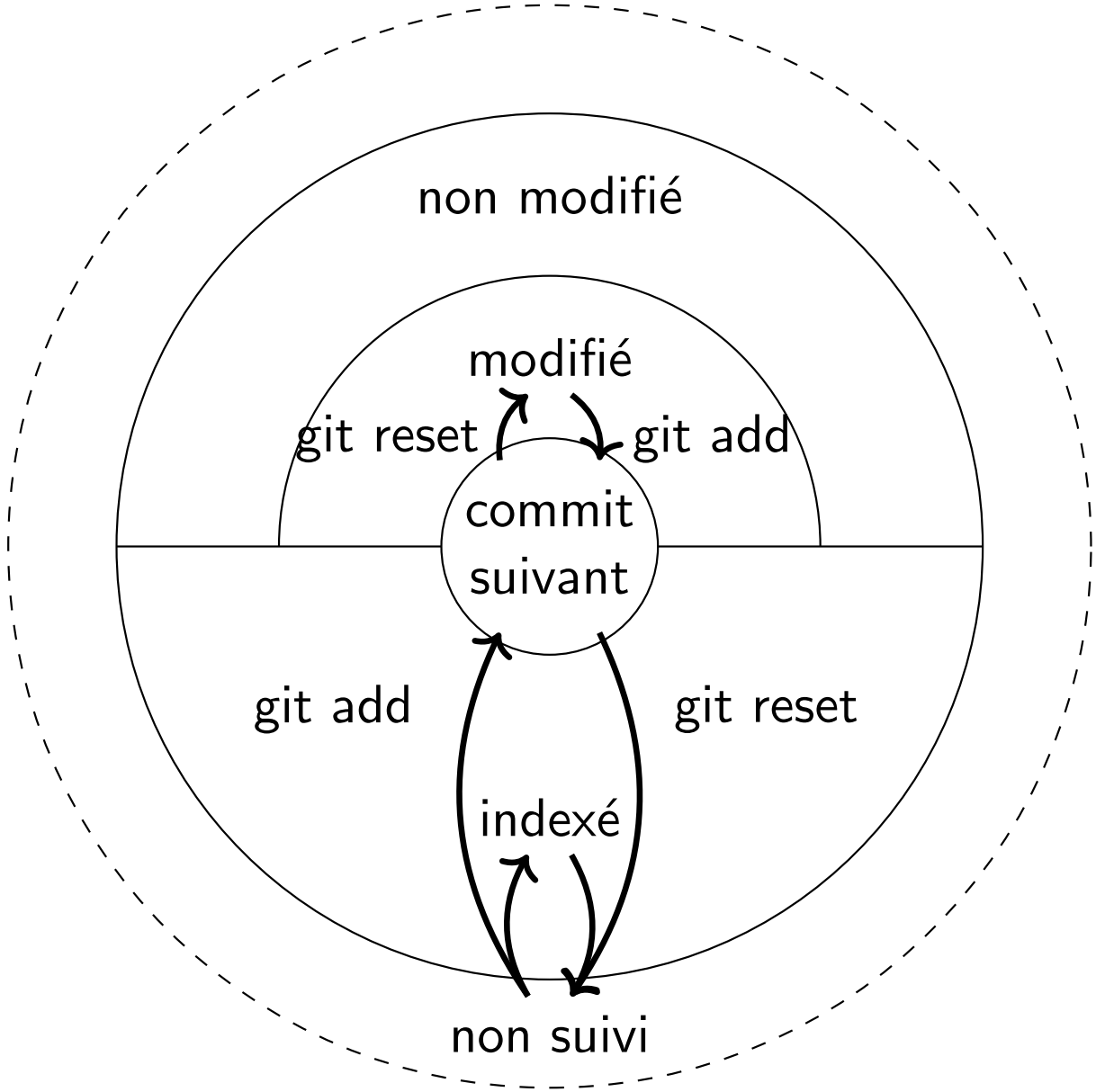
```
% git diff
diff --git a/E.txt b/E.txt
index 87cd58d..6d26147 100644
--- a/E.txt
+++ b/E.txt
@@ -1 +1,3 @@
-Cette ligne est l'\'etat d'origine.
+On peut ajouter une ligne avant.
+Cette ligne est l'\'etat d'origine et on la modifie.
```

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?



Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

git add file — git reset file



Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Constituer une version : git commit

Une fois la version prête (après git add, etc.), la commande git commit permet de créer le nœud

```
% git status
On branch fix
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

modified:   E.txt
new file:   F.txt
% git add E.txt F.txt
% git commit -m "<feat> step F"
[fix f81db80] <feat> step F
 2 files changed, 1 insertion(+)
 create mode 100644 F.txt

% git status
On branch fix
nothing to commit, working tree clean

% git log
commit f81db80282ea991c2a621951bf0943c8ef2dcbce (HEAD -> fix)
Author: Sedoglavic Alexandre <alexandre.sedoglavic@univ-lille1.fr>
Date:   Sun Aug 9 16:00:50 2020 +0200

    <feat> step F
```

L'option -m permet de se passer d'un éditeur en utilisant une chaîne de caractère sur la ligne de commande.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Aménagement de la création d'une version

Une fois indexé (git add) ou versionné (git commit), git diff ne montre naturellement plus de différence.

Le fichier .gitignore énumére les patrons de noms de fichiers à ignorer automatiquement, par exemple :

```
% cat .gitignore
*.[oa]
*~
```

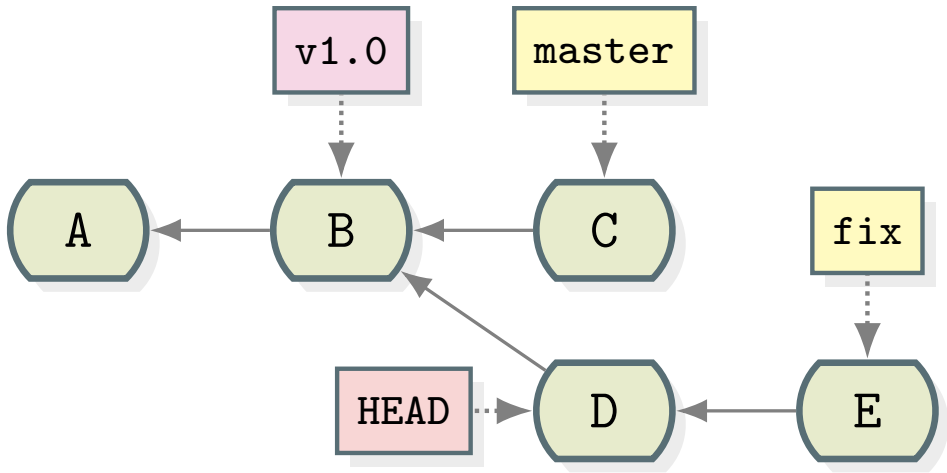
En cas d'erreur, on peut corriger localement la dernière version en date.
Il suffit d'agir comme pour préparer un nouveau commit mais d'utiliser la commande :

```
git commit --amend
```

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Se déplacer dans le dépôt et extraire

```
% git checkout HEAD~2
Note: checking out 'HEAD~2'

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at f5a20cc <feat> step D
```



Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Un déplacement est une extraction totale

Il est possible d'extraire d'une version un fichier précis sans faire l'extraction complète de la version (i.e. sans déplacer la tête sur la version en question du projet).

```
% git status
HEAD detached at f5a20cc
nothing to commit, working tree clean
% git checkout master~1 C.txt
% git status
HEAD detached at f5a20cc
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

new file:   C.txt

%
```

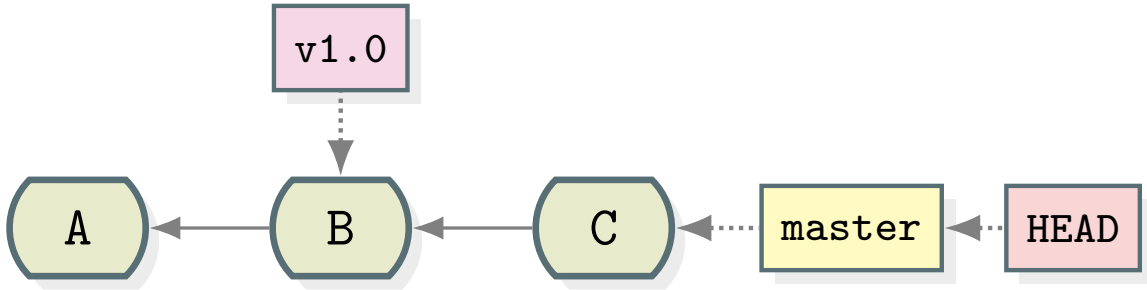
On peut aussi se déplacer sur une autre branche

```
% git checkout master
Previous HEAD position was f5a20cc <feat> step D
Switched to branch 'master'
% git status
On branch master
nothing to commit, working tree clean
%
```

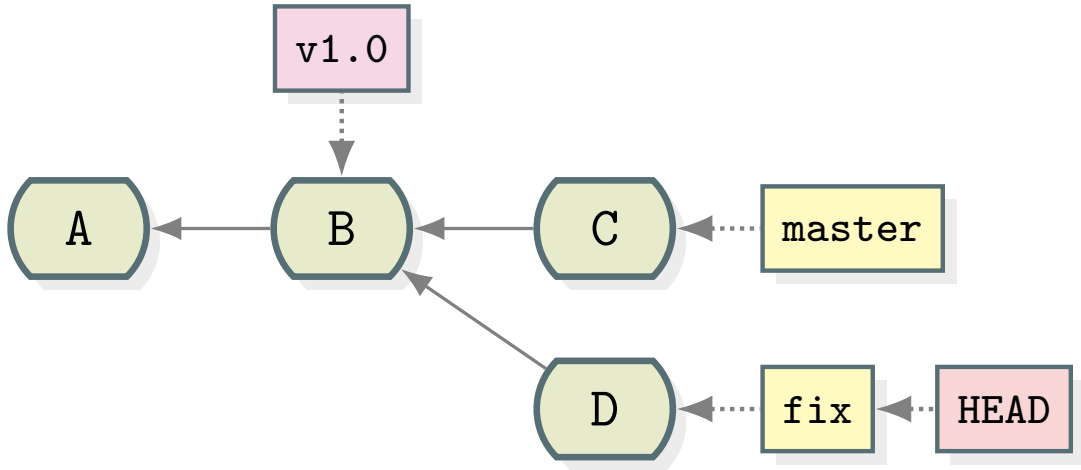
ou en créer une autre.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Création d'une branche

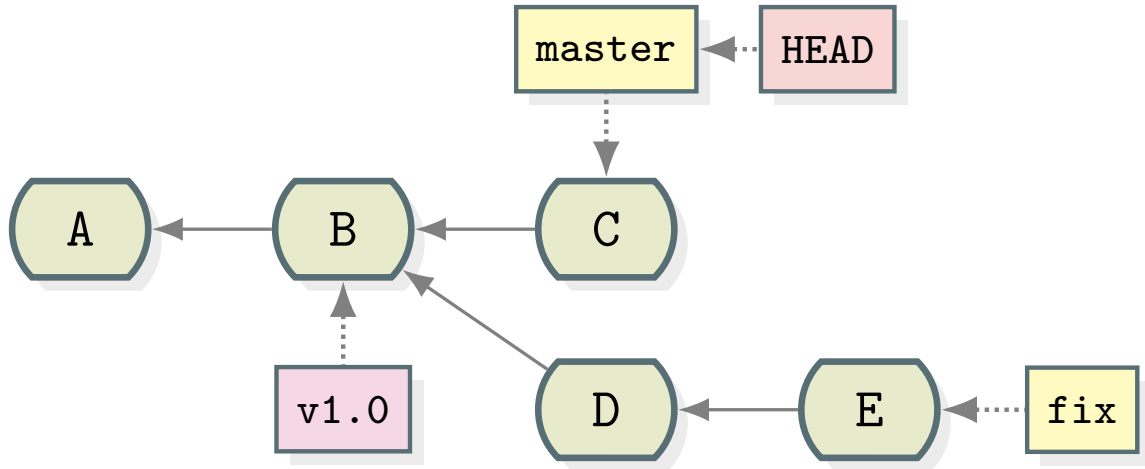



```
% git checkout HEAD~1
% git checkout -b fix # git branch fix ; git checkout fix
% git add D.txt
% git commit -m "<feat> step D"
```

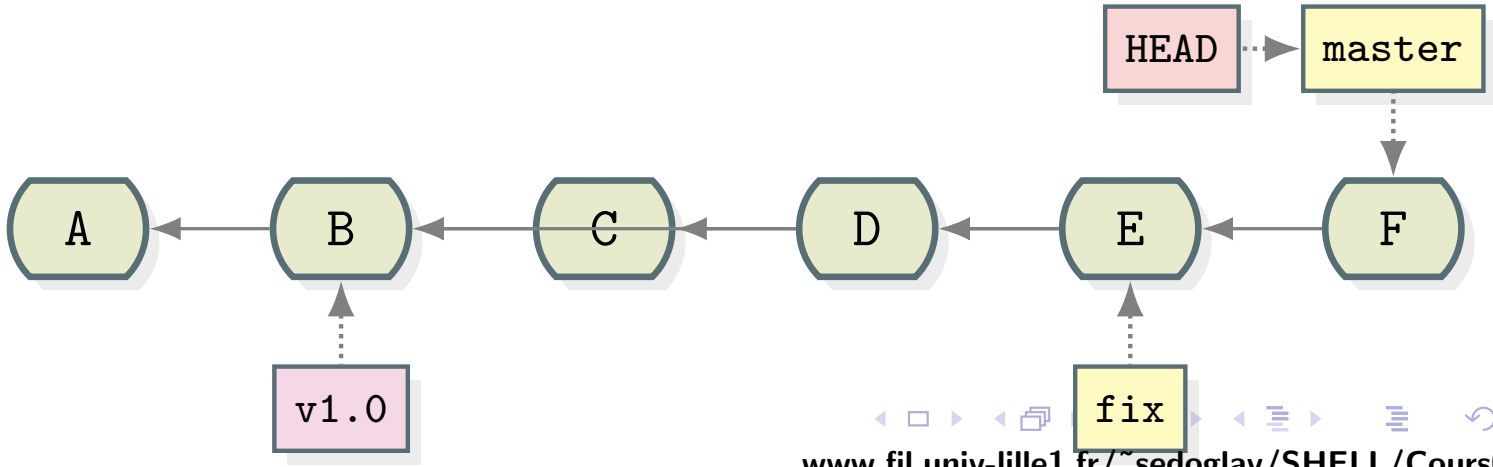


Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

```
% git checkout master
Switched to branch 'master'
```




```
% git merge fix
Merge made by the 'recursive' strategy.
 D.txt | 0
 E.txt | 1 +
 3 files changed, 1 insertion(+)
 create mode 100644 D.txt
 create mode 100644 E.txt
```



Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Des conflits peuvent survenir

Dans l'exemple précédent, les modifications sur les branches master et fix sont totalement indépendantes et donc leur fusion peut se faire sans intervention humaine.
Supposons qu'avant la fusion, les branches master et fix avaient chacunes une version du fichier E.txt incompatibles.
Dans ce cas,

```
% git merge fix
Auto-merging E.txt
CONFLICT (add/add): Merge conflict in E.txt
Automatic merge failed; fix conflicts and then commit the result.
% cat E.txt
<<<<<<< HEAD
La version sur la branche master
=======
La version de la branche fix
>>>>>>> fix
% sed -i '1,3d' E.txt ; sed -i '2d' E.txt
% git add E.txt
% git commit -m "<fix> resolve conflict on E.txt"
[master 3bb4255] <fix> resolve conflict on E.txt
%
```

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Idéalement

La branche master devrait toujours être propre et fonctionnelle (l'ensemble des tests passent).

Pour faire une modification :

- on construit une nouvelle branche à partir de master ;
- on développe et on teste sur cette branche jusqu'à l'obtention d'un état stable, fonctionnelle et testé ;
- on fait passer une revue de cette branche par un tier qui merge avec la branche master.

De très nombreuses méthodologies de développement existent et une première étape dans la collaboration consiste à se mettre d'accord sur celle utilisée.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Dépôts local et distant

Pour lister l'ensemble des dépôts distants en relation avec le dépôts local :

```
% git remote --v
origin git@gitlab-ens.fil.univ-lille1.fr:sedoglav/SHELL.git (fetch)
origin git@gitlab-ens.fil.univ-lille1.fr:sedoglav/SHELL.git (push)
```

Pour "pousser" son travail du dépôt local vers le dépôt distant :

git push origin master

Pour "tirer" son travail du dépôt distant vers le dépôt local :

git pull origin master

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

L'environnement Gitlab

Gitlab est une application web basée sur git qui permet de gérer :

- le cycle de vie de projects git ;
- les participants aux projets (droits, groupes, etc.) ;
- la communication entre ces participants (tickets, etc.) ;
- construction, test et déploiement automatiques.

C'est un produit d'une société privée dont la "community edition" est libre.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Créer et partager un dépôt distant

GitLab permet de créer un nouveau dépot en cliquant sur New project.
Un identifiant est nécessaire et vous l'indiquez par le biais de Project name.
Pour vos travaux dans l'UE, vous devrez cochez qu'il s'agit d'un projet Private.
Pour finir, validez en cliquant sur Create project.

Vous pouvez donner des droits à votre enseignant et à votre binôme sur ce dépôt pour qu'ils puissent accéder et/ou contribuer au code (il faut que votre binôme se soit connecté au moins une fois à GitLab).
Pour ce faire, cliquez sur le menu Members (en haut à droite) renseignez l'identifiant de votre binôme (son *login*), changez les droits d'accès pour Master, puis cliquez sur l'icône Add to project.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Vos enseignants partageront probablement des dépôt avec vous.
Pour ce faire, il vous communiqueront l'URL d'un dépôt distant préxistant.
Dans votre espace gitlab, après avoir saisi cette URL vous devrez la *forker* en cliquant sur l'icône fork.
L'interface gitlab vous proposera alors un domaine de nom (namespace) dans lequel effectuer le fork (cela conditionne vos droits sur ce nouveau dépôt).
Ceci fait, vous venez d'obtenir un nouveau dépôt distant que vous pourrez ensuite cloner, etc.

Généralement, il conviendra de supprimer la dépendance entre le dépôt que vous venez de forker et le dépôt initial.
Pour ce faire, vous devez naviguer dans la configuration avancée (settings – advanced – remove) lors du fork.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Première synchronisation du dépôt

Pour construire votre dépôt local à partir du dépôt distant, vous devez cloner ce dernier en utilisant son URL :

```
% git clone <protocole><url>
```

Après avoir choisit un dépôt dans Gitlab, une icône en bas à droite permet de coller l'URL correspondante dans le presse-papier.

On peut utiliser soit le protocole SSH (nécessitant d'avoir votre clef privée en local et votre clef publique sur Gitlab) ou HTTP (nécessitant votre couple login/password de Gitlab). Cette opération n'est à faire qu'une seule fois :

- elle crée le dépôt local ;
- en positionnant correctement les informations de git remote ;
- ensuite on utilise git pull, git push.

Introduction
Git ?
Pourquoi ?
Rudiment Git
Concepts de base
Observer
Modifier
Entériner
Déplacement
Développement topiaire
Dépôts local et distant
Écosystème Git
Gitlab ?

Synthèse minimale

Après avoir créé un dépôt distant puis l'avoir cloné en un dépôt local une seule fois sur votre espace NFS, votre travail avec git peut se résumer à :

- faire un git pull en début de séance ;
- itérer autant que nécessaire le cycle :
  - git add
  - git commit
- en fin de séance, il convient de pousser votre travail sur le dépôt distant par un git push.

Cet usage minimal est important car la plupart de vos TP seront évalués à partir d'un dépôt git.