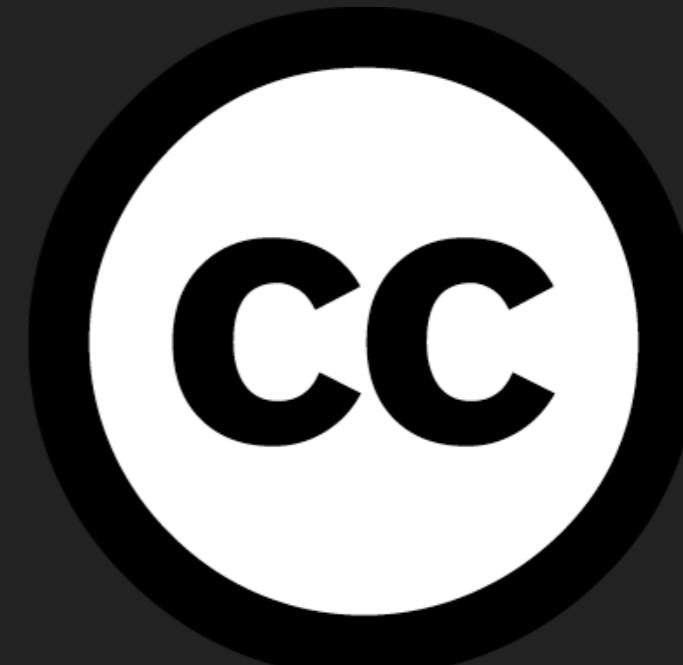


GESTION DE CODE & INTEGRATION CONTINUE

ENVIRONNEMENT DE
DEVELOPPEMENT AVANCE

CREATIVE COMMONS BY

- ▶ La présentation est sous licence
CREATIVE COMMONS BY
- ▶ [https://creativecommons.org/
licenses/by/3.0/fr/](https://creativecommons.org/licenses/by/3.0/fr/)
- ▶ Certaines images, schémas
viennent d'auteurs différents
avec des licences qui me
permettent de les utiliser ici.
Elles restent leur propriété.
Cf la partie crédits à la fin.



NICOLAS ROUVIERE

- ▶ Développement depuis 2003
- ▶ Java / Python
- ▶ Linux / Windows
- ▶ GiT, SVN, Clearcase
- ▶ DevOps, équipe de 100+
- ▶ Airbus Defence & Space



nicolas.rouviere@gmail.com

A black and white aerial photograph of a massive concrete dam. The dam is a thick, curved wall that slopes down to a river or lake. A narrow walkway runs along the top of the dam, with railings on both sides. Two small figures of people are visible on the walkway, emphasizing the enormous scale of the structure. The water in the reservoir behind the dam is calm.

CHUT CA COMMENCE

INTRODUCTION

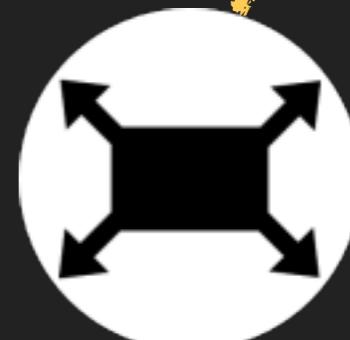
DIFFERENCES ENTRE UN PROJET PERSO ET UN PROJET INDUSTRIEL ?

Johnny Appleseed

REPRODUCTIBILITE



ECHELLE



COLLABORATION



CODE LEGACY

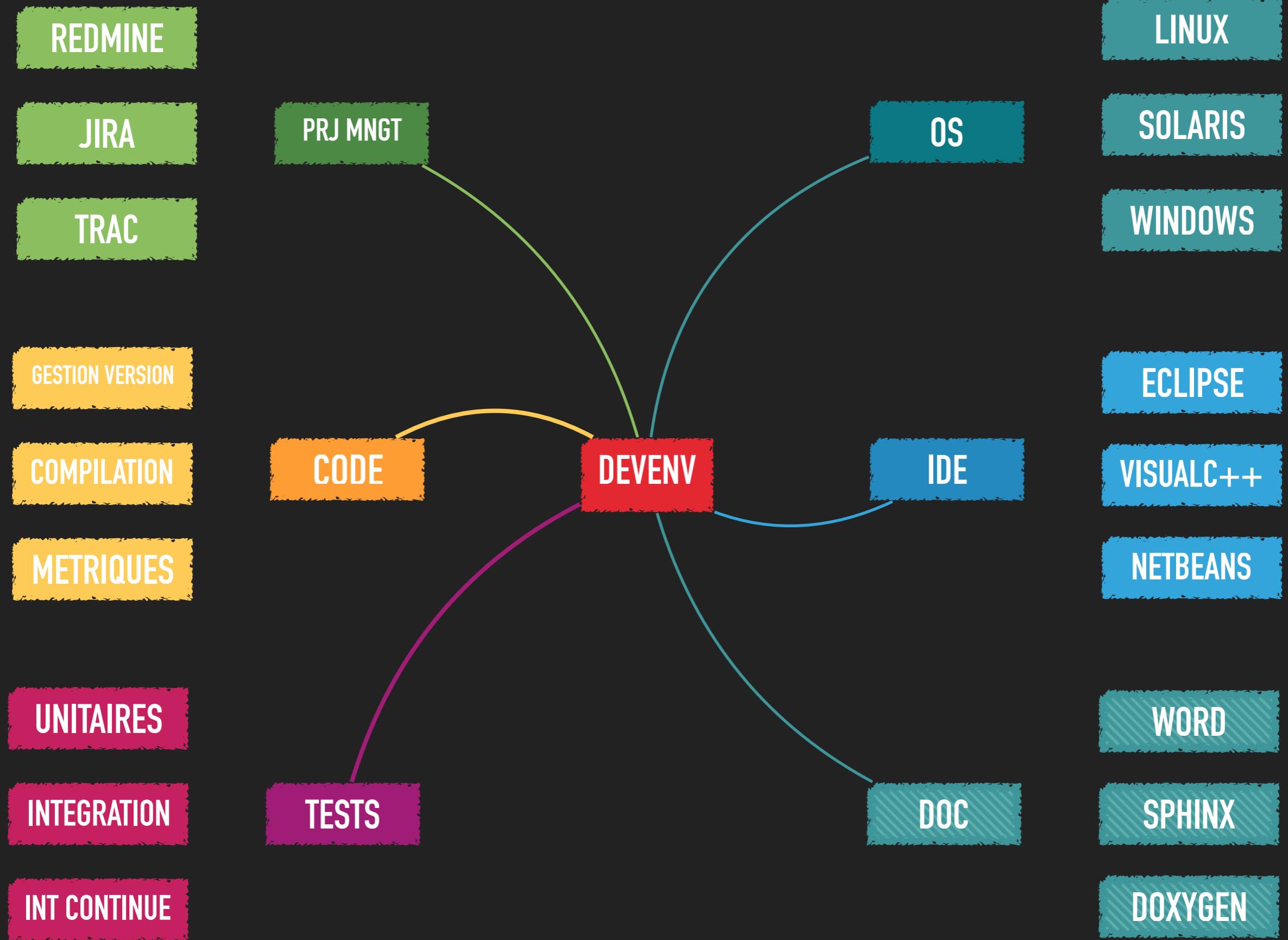


DATES LIVRAISON



PROPRIETE





QU'EST CE QUE LA

GESTION DE VERSION ?



A screenshot of a computer screen showing a code editor in Brackets. The file open is 'content-single-product-rightcol-fullheight.php'. The code is PHP, related to WooCommerce product pages. It includes comments explaining various hooks and actions used in the template. The Brackets interface shows tabs for other files like 'content-single-product-rightcol.php' and 'single-product.php'. The status bar at the bottom indicates 'Ligne 24, colonne 34 - 117 lignes'. The Mac OS X dock is visible at the bottom of the screen.

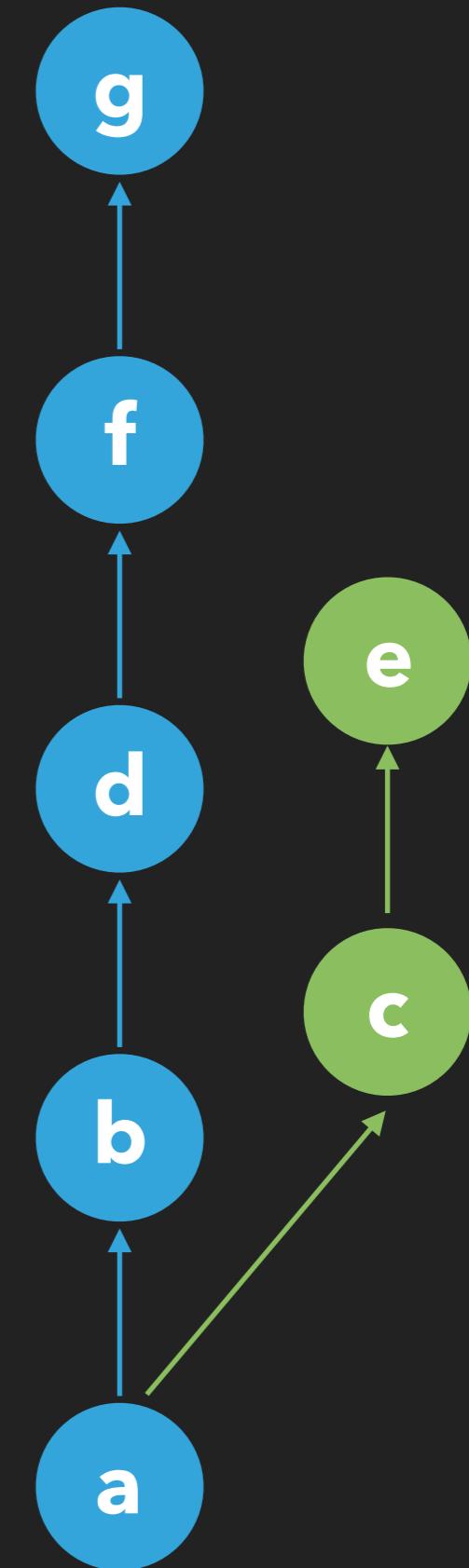
```
11 <?php
12 /**
13 * woocommerce_before_single_product hook
14 *
15 * @hooked woocommerce_show_messages - 10
16 */
17 do_action( 'woocommerce_before_single_product' );
18 ?>
19
20 <div itemscope itemtype="http://schema.org/Product" id="product-<?php the_ID(); ?>" <?php post_class(); ?>>
21
22 <div class="row">
23
24 <div class="large-9 columns">
25 <div class="row">
26 <div class="large-8 columns product-gallery">
27
28 <?php
29 /**
30 * woocommerce_show_product_images hook
31 *
32 * @hooked woocommerce_show_product_sale_flash - 10
33 * @hooked woocommerce_show_product_images - 20
34 */
35 do_action( 'woocommerce_before_single_product_summary' );
36
37 </div><!-- end large-6 - product-gallery -->
38
39 <div class="product-info large-12 small-12 columns left">
40 <div class="droite">
41 <?php
42 /**
43 * woocommerce_single_product_summary hook
44 *
45 * @hooked woocommerce_template_single_title - 5
46 * @hooked woocommerce_template_single_price - 10
47 * @hooked ProductShowReviews() (inc/template-tags.php) - 15
48 * @hooked woocommerce_template_single_excerpt - 20
49 * @hooked woocommerce_template_single_add_to_cart - 30
50 * @hooked woocommerce_template_single_meta - 40
51 * @hooked woocommerce_template_single_sharing - 50
52 */
53 do_action( 'woocommerce_single_product_summary' );
54
55 </div>
56 </div><!-- end product-info large-4 -->
57
58 </div><!-- row -->
59
60
61 <div class="row">
62 <div class="large-12 columns">
63
```

NOM DE ZEUS MARTY

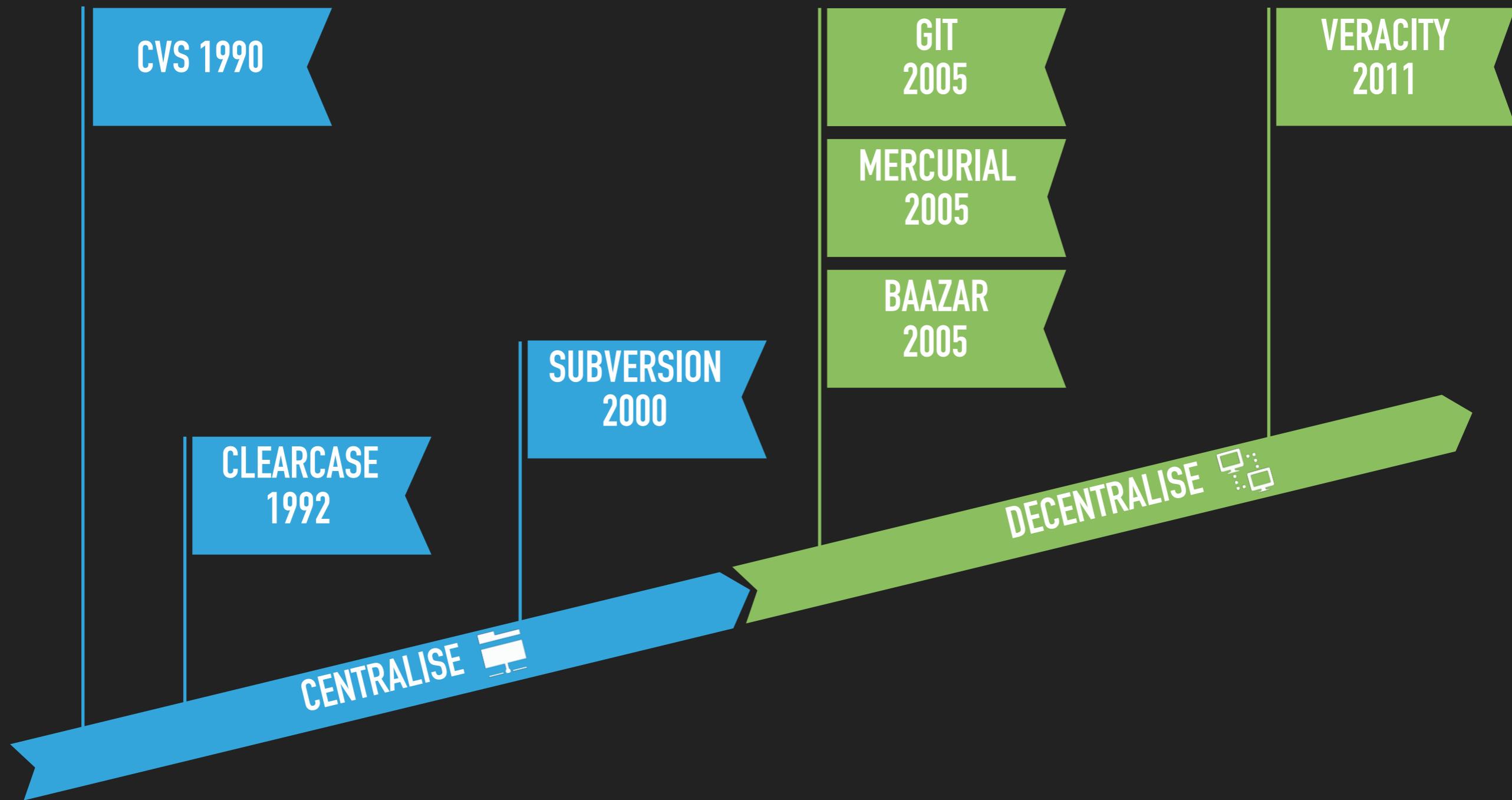
Emmet Brown

LA GESTION DE VERSION PERMET DE

- ▶ Sauver des états du code
- ▶ Collaborer sur du code
- ▶ Travailler de manière concurrente
- ▶ Gérer les conflits
- ▶ Marquer des versions
- ▶ Revenir en arrière



LES OUTILS DE VERSIONING



QU'EST CE QUE L'

INTEGRATION CONTINUE

L'INTEGRATION CONTINUE PERMET DE

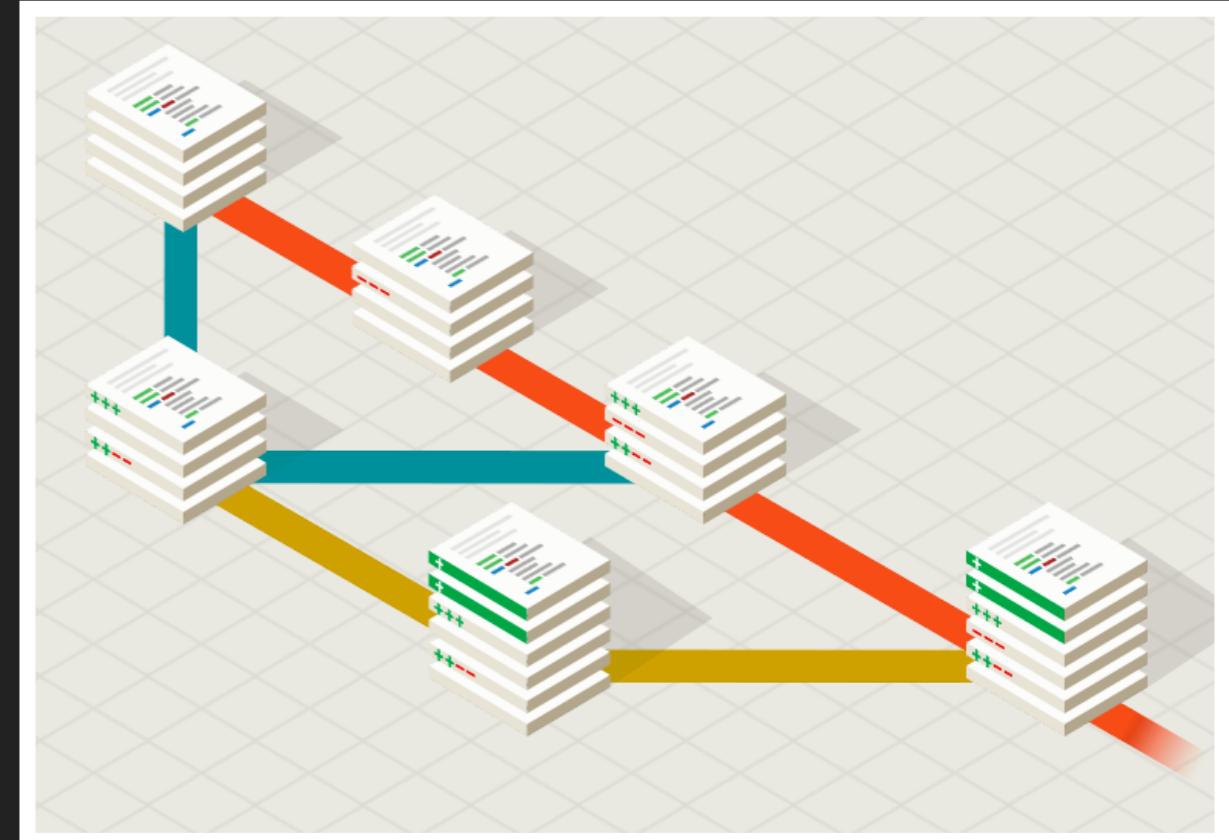
- ▶ Lancer les tests à chaque changement de code
- ▶ Prévenir en cas de régression
- ▶ Vérifier l'intégration des composants
- ▶ Tester sur plusieurs OS
- ▶ Livrer en continu



INTRODUCTION

MENU

- ▶ GESTION DE CODE
with git, gitlab, SourceTree
- ▶ INTEGRATION CONTINUE
with gitlabci

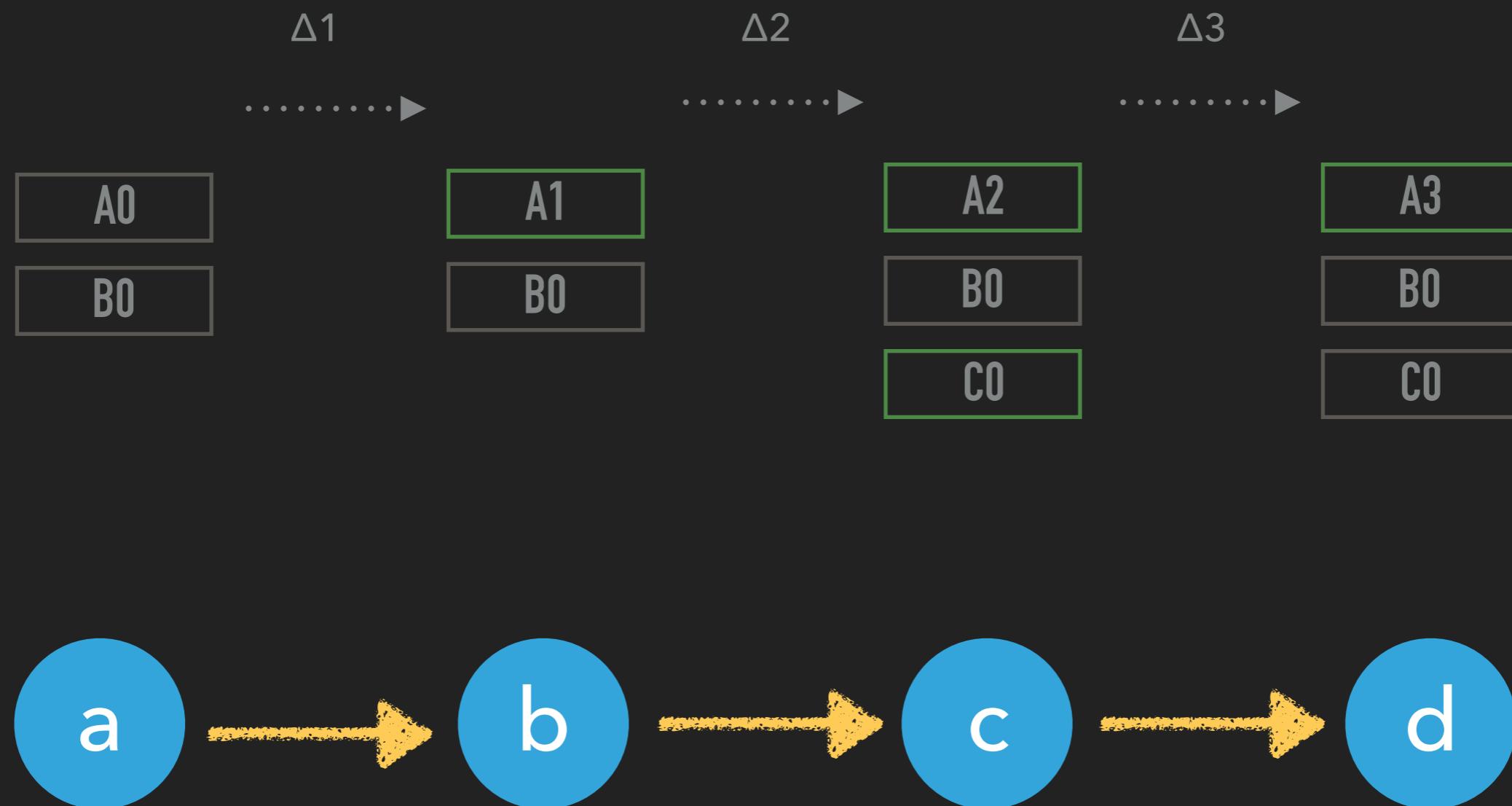


```
005 17:33 best\ddRén 816 WikisSéBéWék
004 21:27\external\sushi@iBis 816 WikiChange
004 :\Progs\external\sushi@iBis 816 . 2.451
004 21:28 <DIR> .
004 17:33 best\ddRén )
005 17:33 best<DIR>n )
005 17:33 best\ddRén ) 816 WikisSéBéWék
005 :\Progs\external\sushi@iBis 816 WikisSéBéWék
004 :\Progs\external\sushi@iBis 816 WikisSéBéWék
004 :\Progs\external\sushi@iBis 816 . 2.451
005 :\Progs\external\sushi@iBis 816 . 2.451
005 17:33 best\ddRén )
005 17:03 best\ddRén )
005 :\Progs\exteDáR\sushi@iBis 816 WikisSéBéWék
005 :\Progs\external\sushi@iBis 816 WikisSéBéWék
004 :\Progs\external\sushi@iBis 1.818 WikiChabg
005 21:28 <DIR> .
005 17:33 best<DIR>n )
005 17:33 best\ddRén ) 816 WikisSéBéWék
```

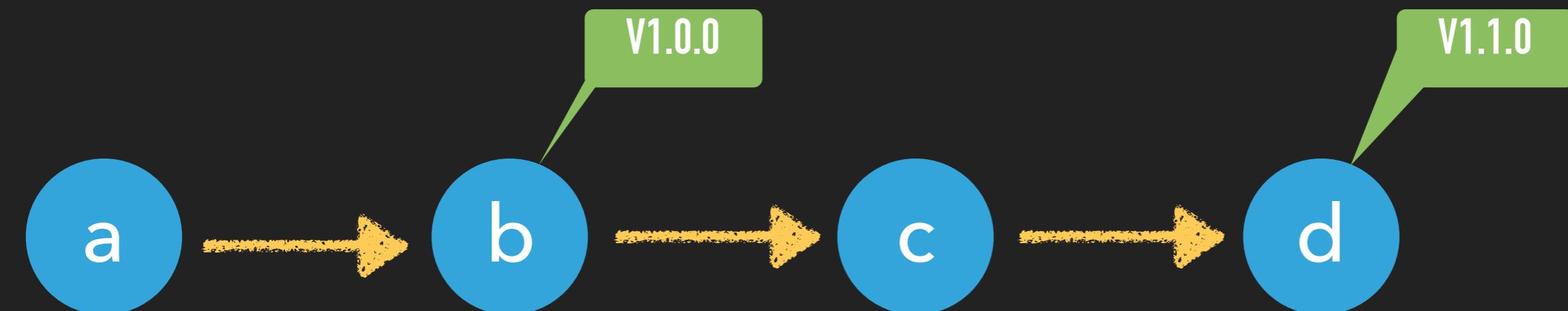
GESTION DE CONF. VERSIONING

GESTION DE CODE

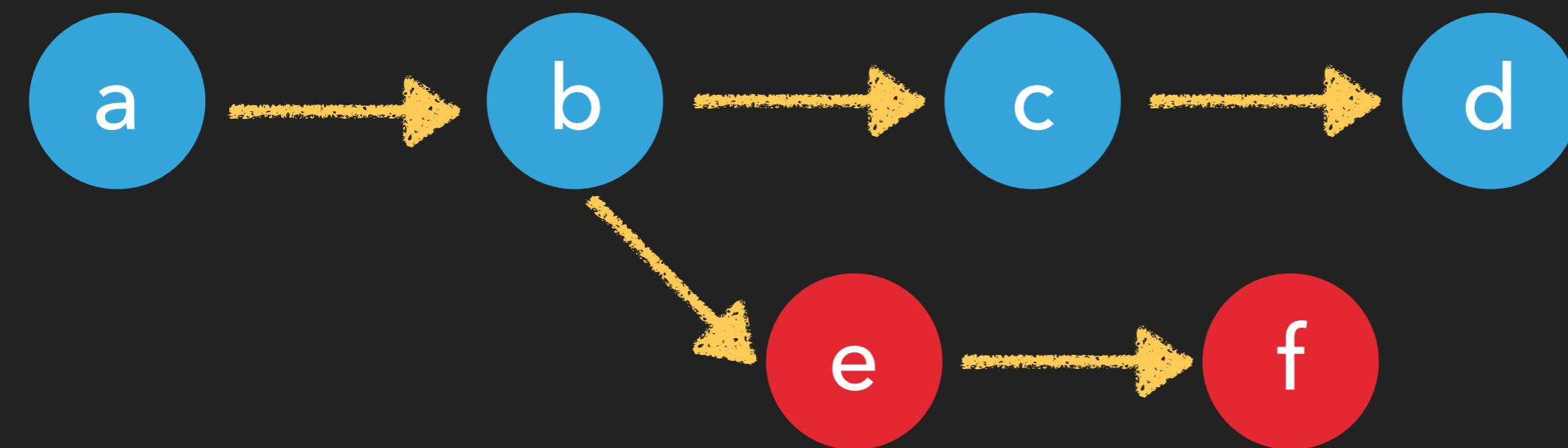
JALON DE CODE = COMMIT



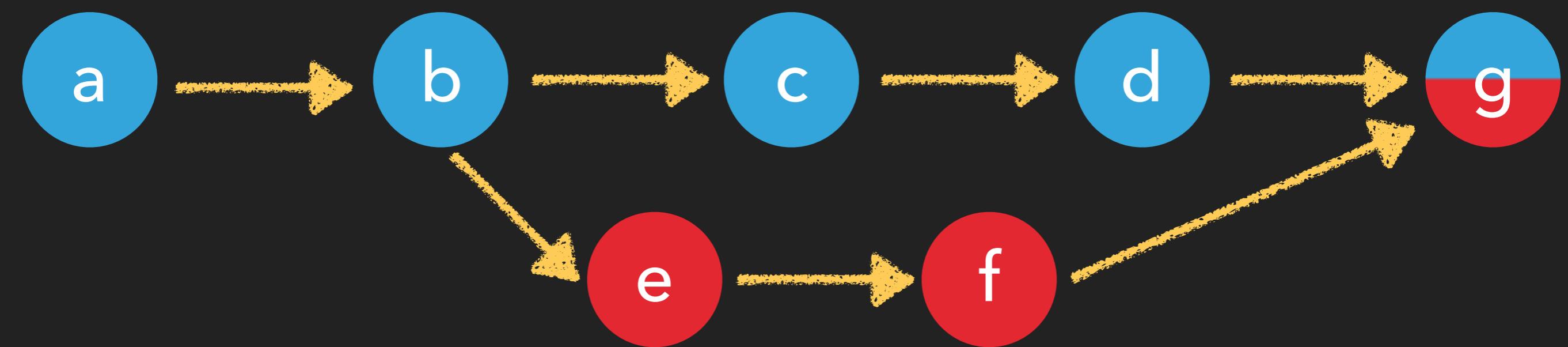
MARQUER UNE VERSION = TAG



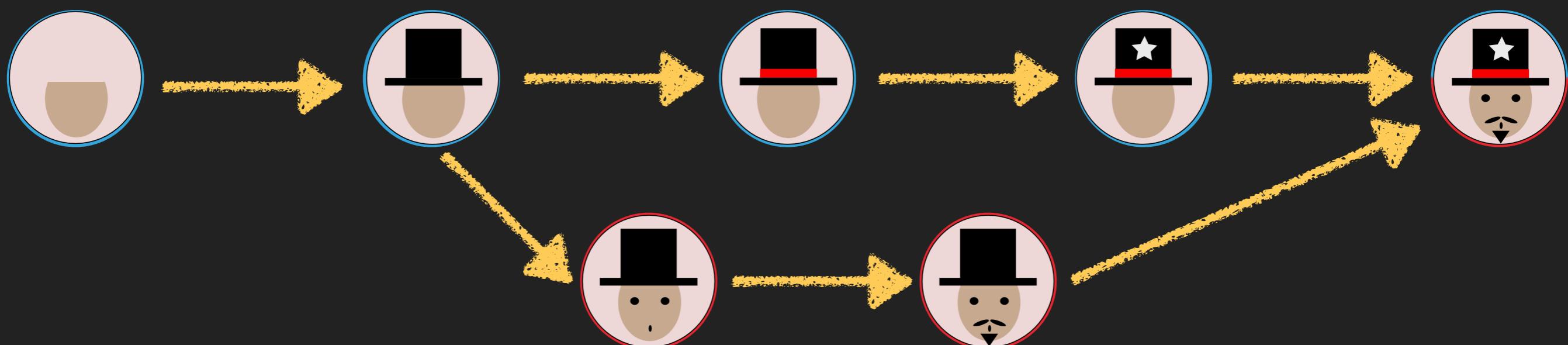
DÉVELOPPEMENT PARALLÈLE = BRANCHE



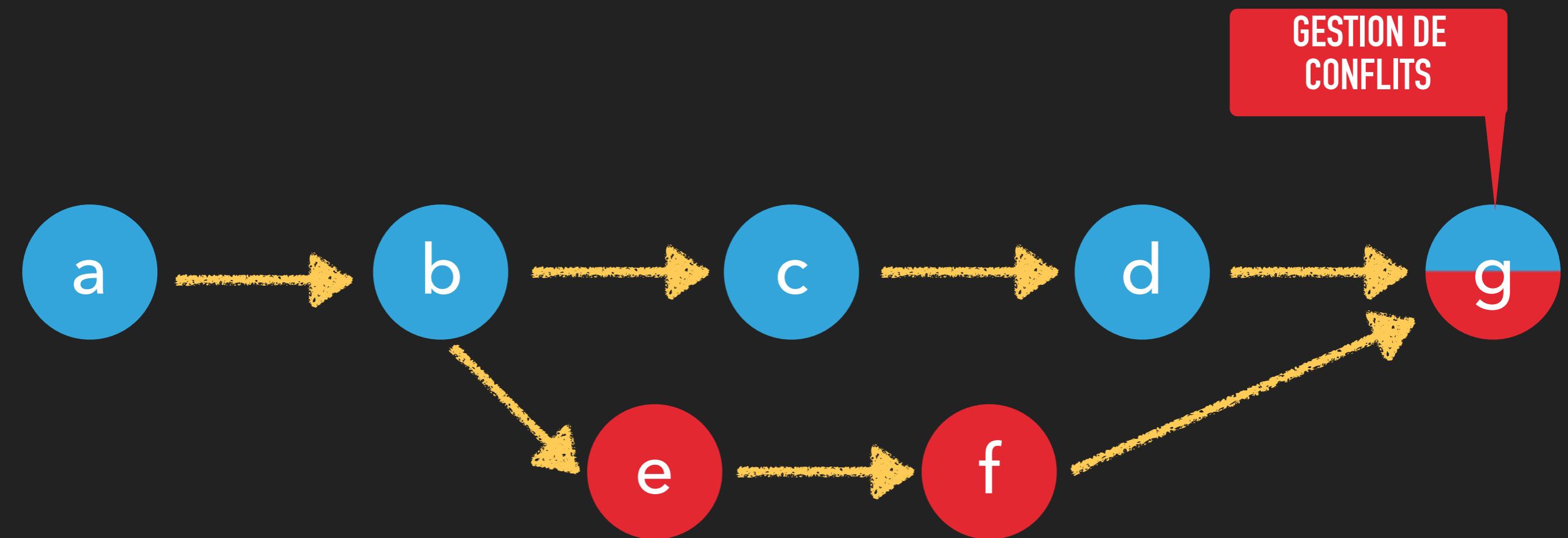
RÉINTÉGRER UNE BRANCHE = MERGE



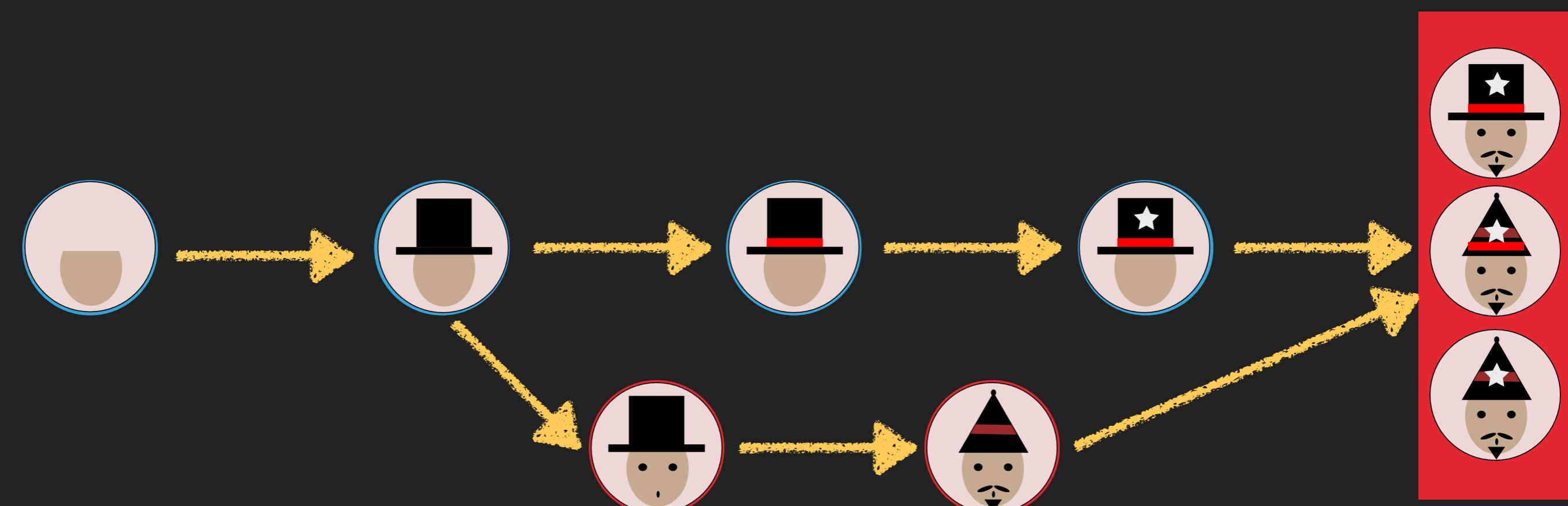
BRANCHER ET MERGER

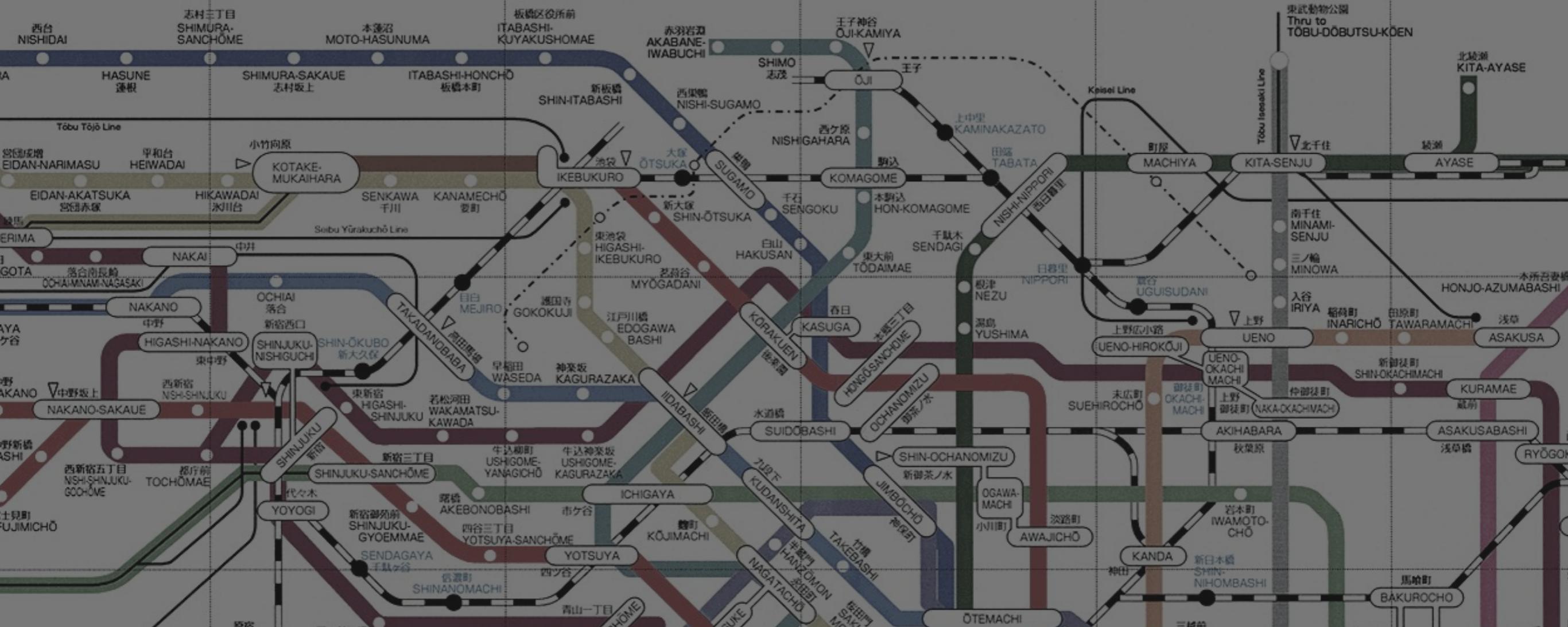


EDITER LE MÊME FICHIER EN // = CONFLIT



BRANCHER ET CONFLIT





LA ROUTE ? LÀ OÙ ON VA, ON N'A PAS BESOIN DE ROUTE !

GIT INTRO

CRÉÉ PAR LINUS

- ▶ Créé en 2005
- ▶ Conçu pour gérer le noyau linux
- ▶ Logiciel libre GNU GPL V2

LINUS TORVALDS

DÉCENTRALISÉ

OPTIMISÉ

PUISSANT

DÉCENTRALISÉ

- ▶ Chaque noeud du réseau peut
 - ▶ Accéder à **tout** l'historique
 - ▶ Travailler sans réseau
 - ▶ Faire office de serveur

LINUS TORVALDS

DÉCENTRALISÉ

OPTIMISÉ

PUISSANT

OPTIMISÉ

- ▶ Rapide
- ▶ Compressé

LINUS TORVALDS

DÉCENTRALISÉ

OPTIMISÉ

PUISSANT

PUISSANT

- ▶ Maîtrise de l'historique
- ▶ Configuration avancée
- ▶ Tout est possible avec Git

LINUS TORVALDS

DÉCENTRALISÉ

OPTIMISÉ

PUISSANT

COMMANDES PHARE

- ▶ merge / rebase
- ▶ clone / push / pull
- ▶ branch / reset
- ▶ bisect

LINUS TORVALDS

DÉCENTRALISÉ

OPTIMISÉ

PUISSANT



QUITTE À VOYAGER DANS LE TEMPS AU VOLANT D'UNE VOITURE, AUTANT EN CHOISIR UNE QUI AIT DE LA GUEULE !

LE MODELE DE GIT

UN GRAPHE ORIENTÉ



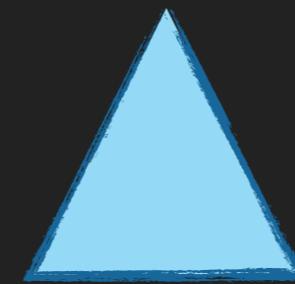
objet commit



relation père fils



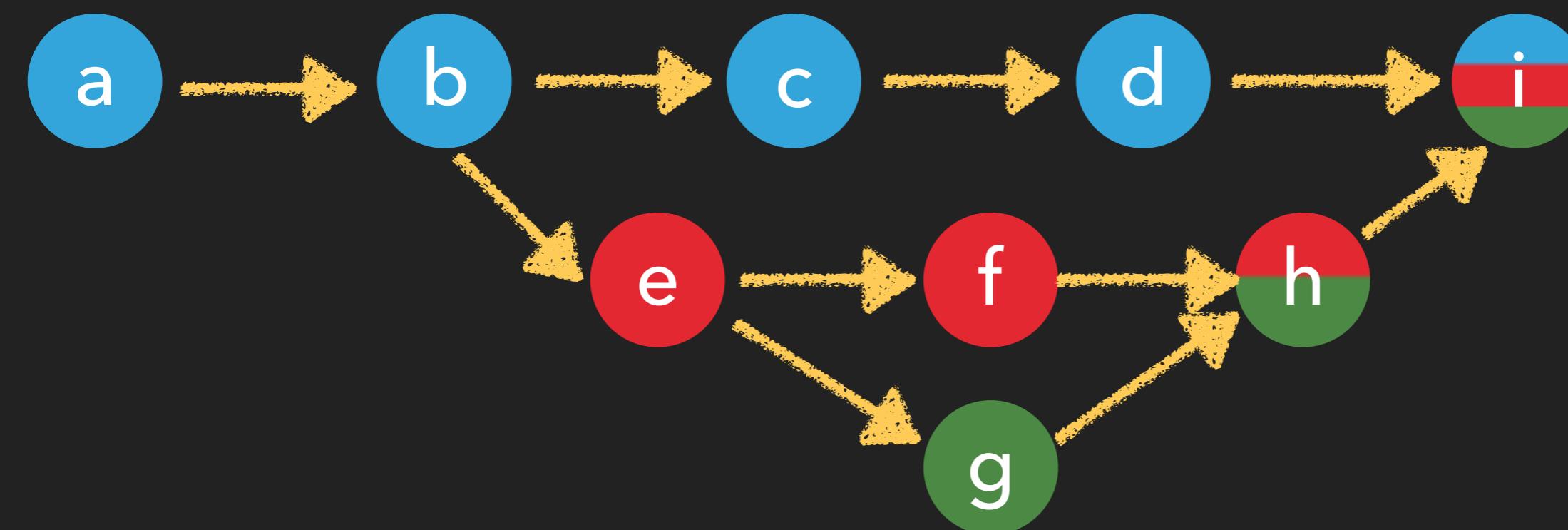
référence



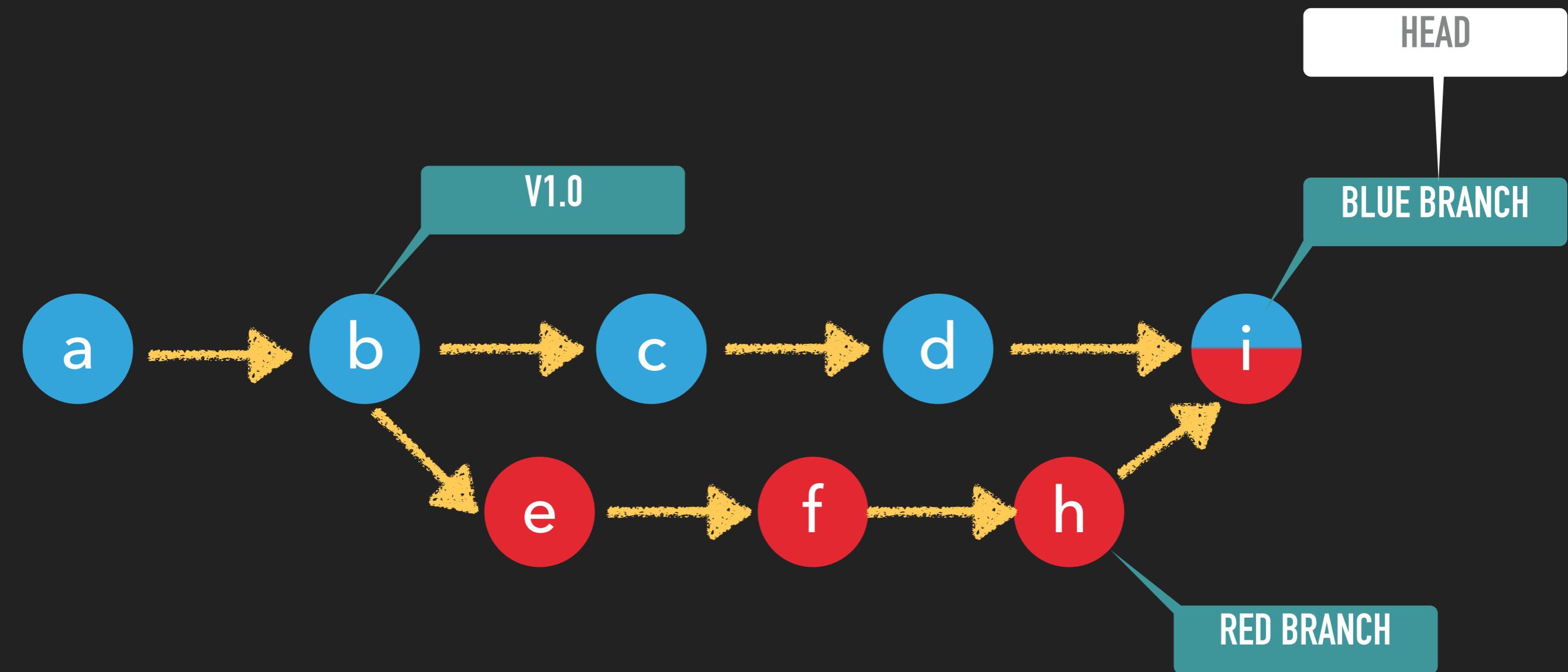
patch

COMMIT

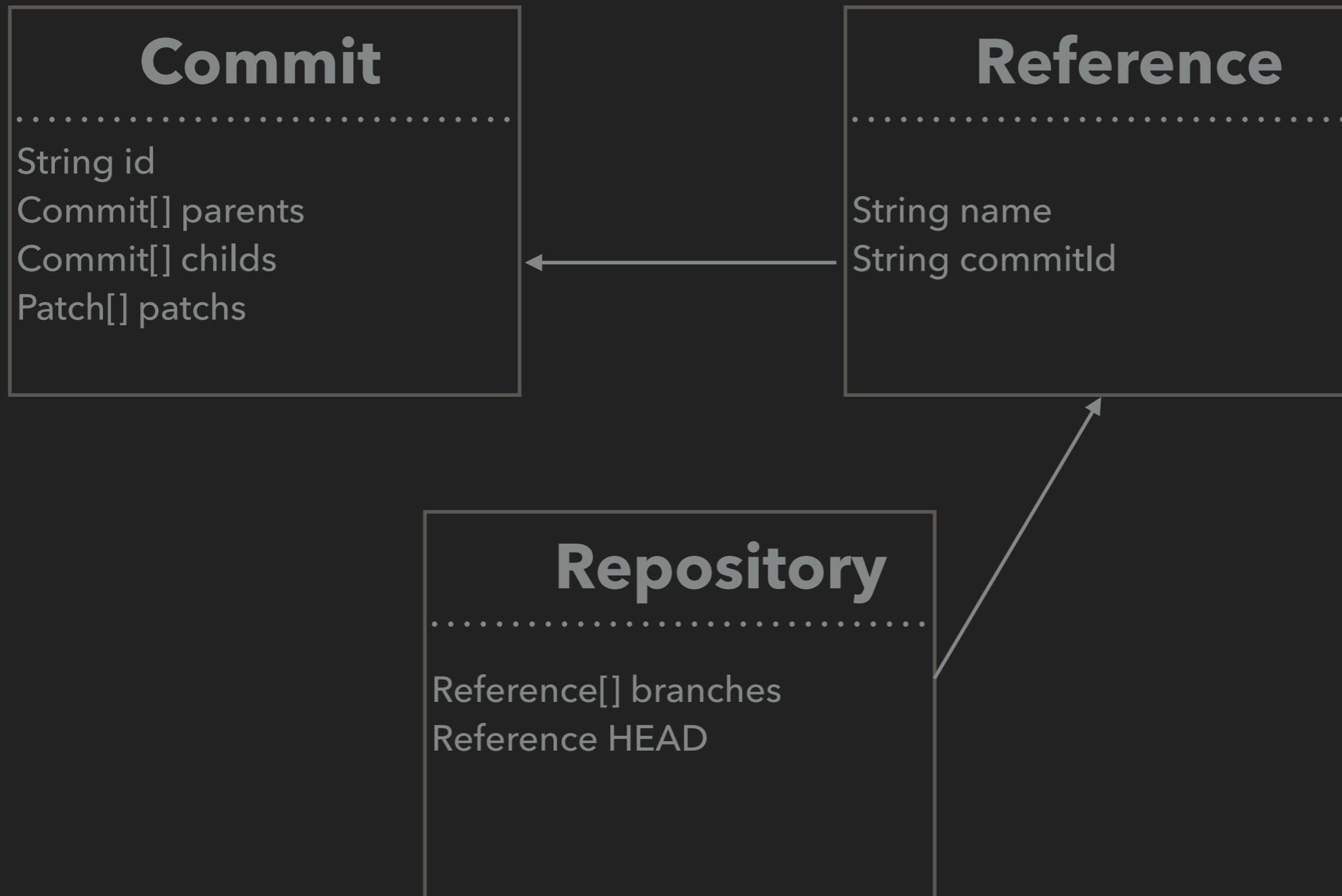
un **commit** a un ou plusieurs enfants qui ont un ou plusieurs enfants qui ont un ou plusieurs parents



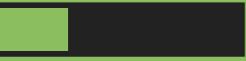
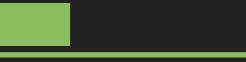
REFERENCE

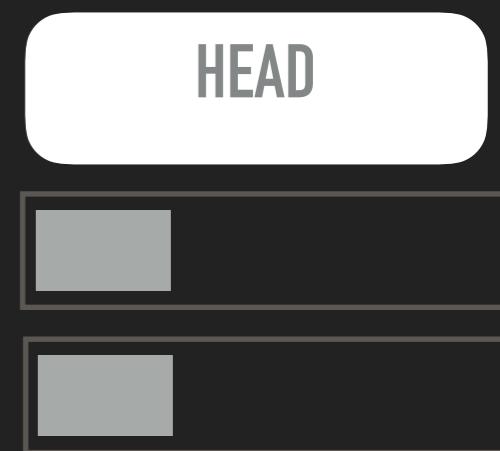
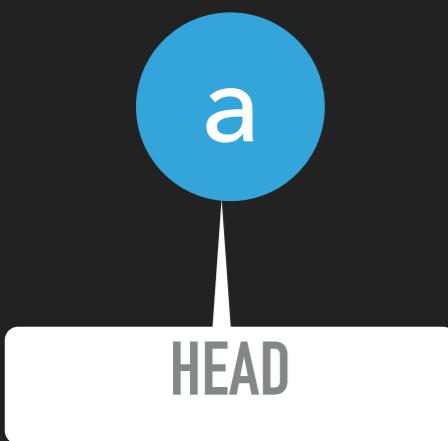


MODELE



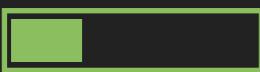
PATCH

A.cpp 
B.cpp 

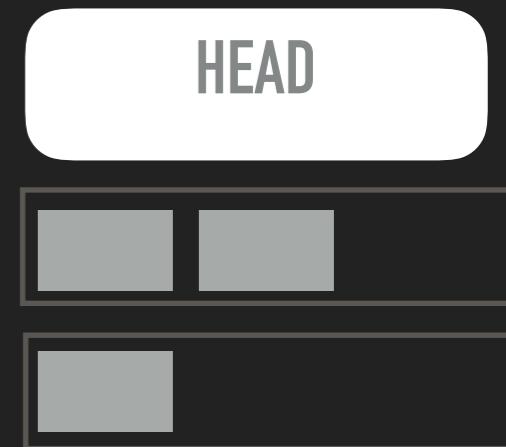
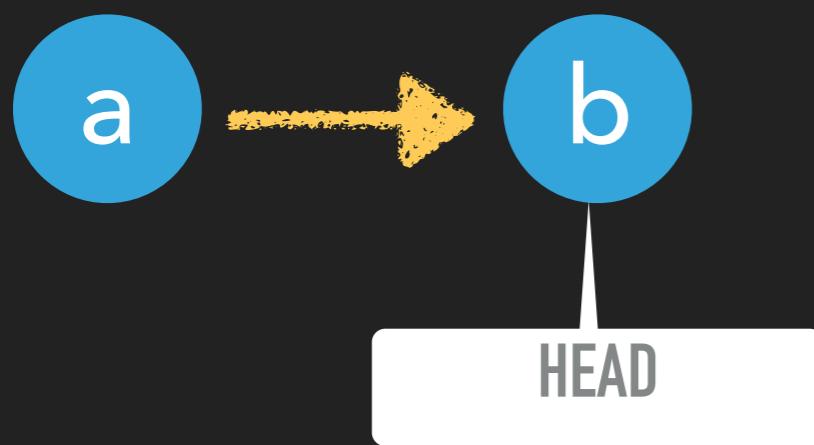
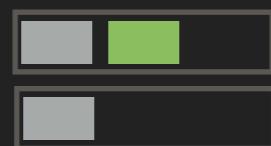
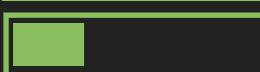


PATCH

A.cpp

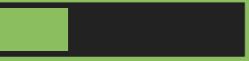


B.cpp

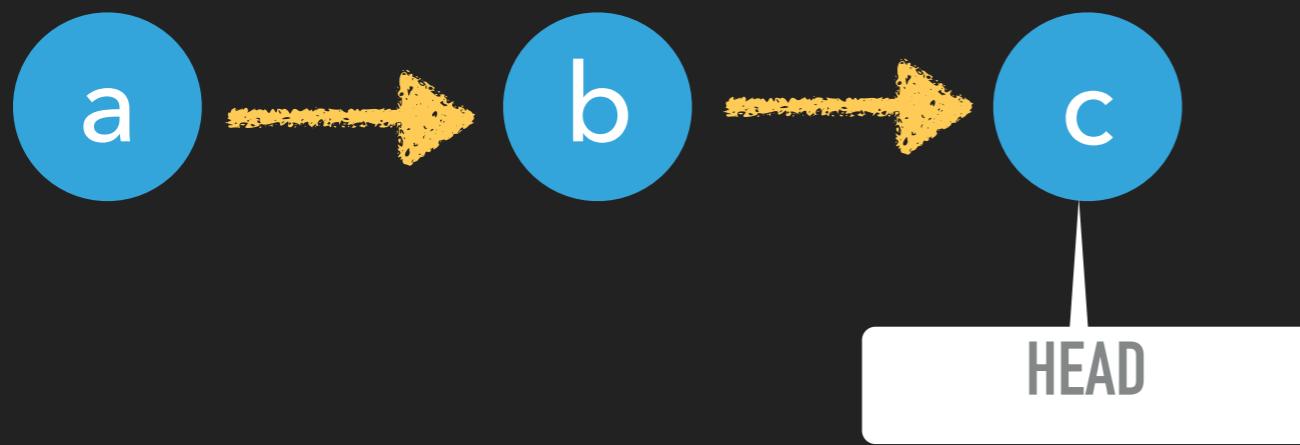
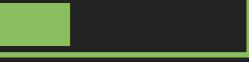


PATCH

A.cpp

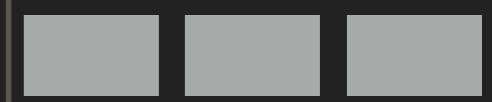


B.cpp

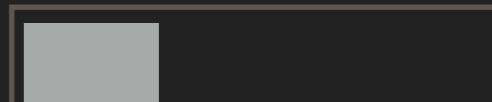


HEAD

A.cpp

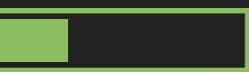


B.cpp

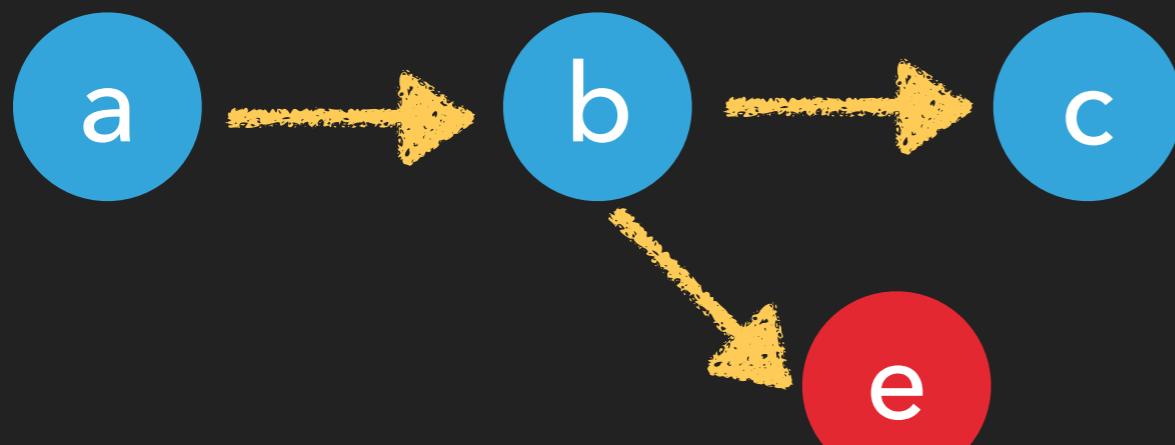
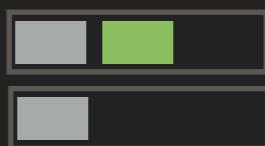
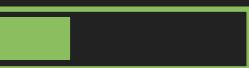


PATCH

A.cpp



B.cpp



A.cpp

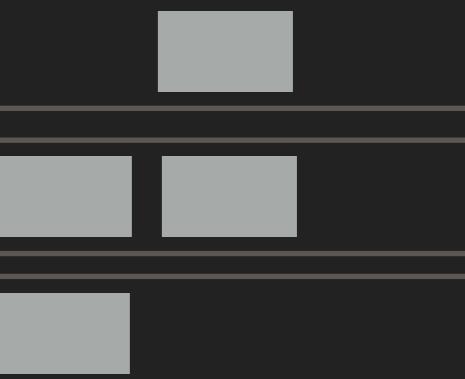


B.cpp

C.cpp

HEAD

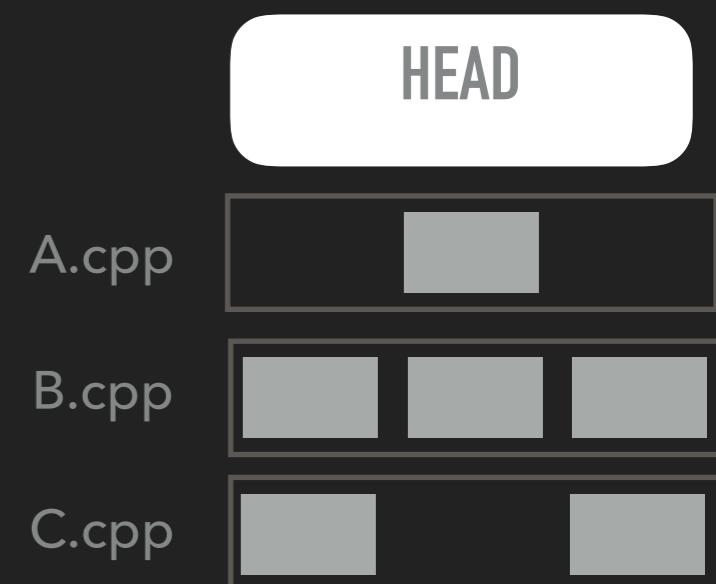
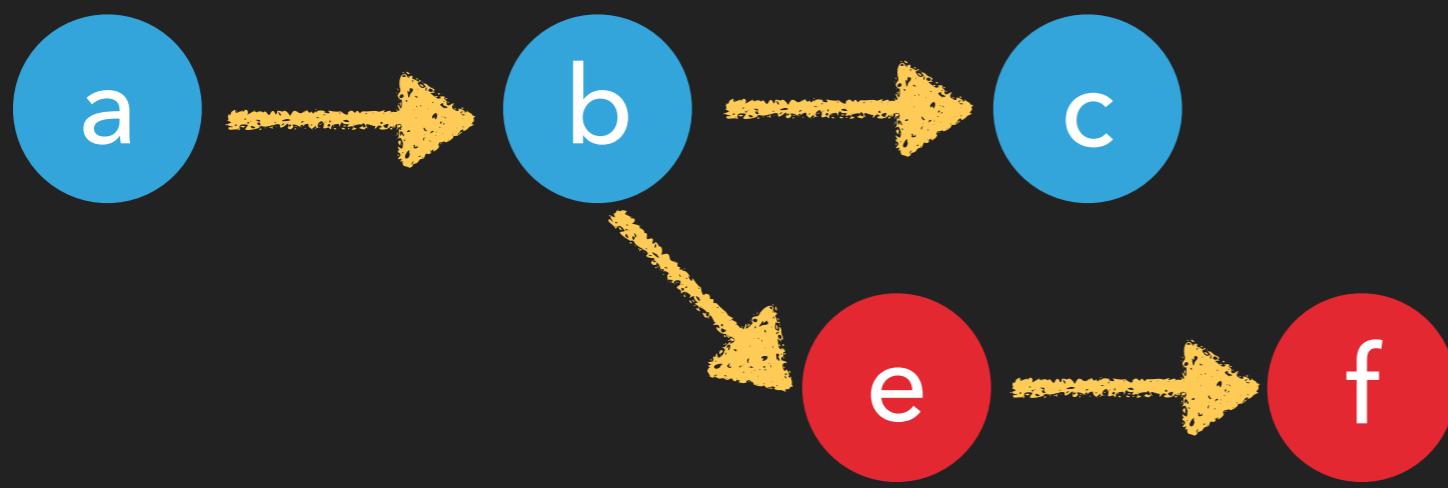
A.cpp



B.cpp

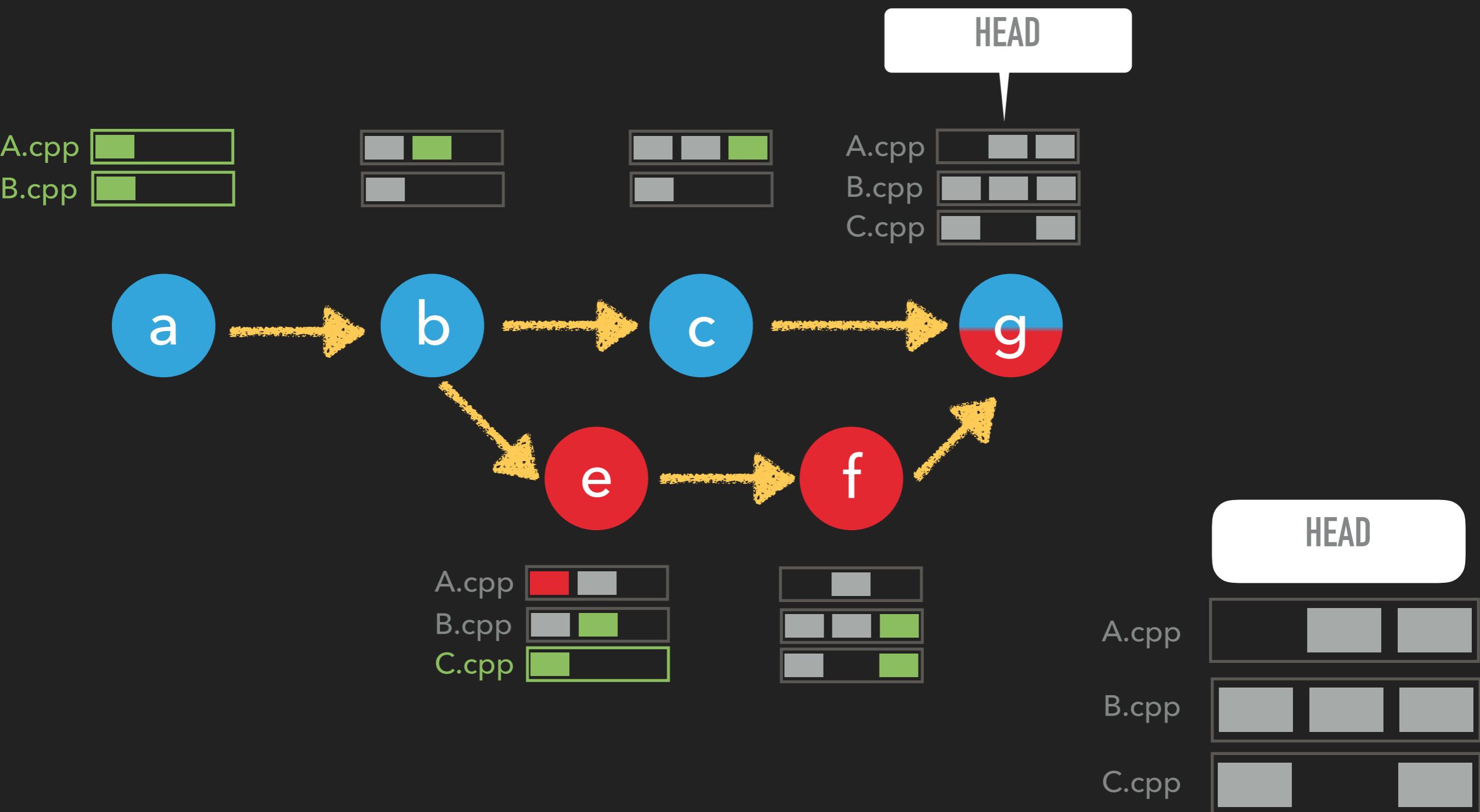
C.cpp

PATCH



HEAD

PATCH





SOURCETREE, GITK, ECLIPSE

LES CLIENTS GIT

LES CLIENTS GIT

GITK

- ▶ Client officiel
- ▶ win/linux/mac
- ▶ moche
- ▶ efficace

The screenshot shows the gitk graphical interface for Git. The top half displays a timeline of commits from June 2005, with each commit represented by a colored dot and a line connecting it to its parent or parents. The bottom half provides a detailed view of a specific commit, identified by its SHA1 ID: `9f793d2c77ec5818679e4747c554d9333cecf476`. The commit details are as follows:

Author: Pete Zaitcev <zaitcev@redhat.com> 2005-06-06 14:54:59
Committer: Greg Kroah-Hartman <gregkh@suse.de> 2005-06-09 02:38:11

[PATCH] USB: fix ub issues

This smoothes two imperfections:
- Increase number of LUNs per device from 4 to 9. The best solution would be to remove this limit altogether, but that has to wait until the time when more than 26 hosts are allowed.
- Replace mdelay with msleep in a probing routine.

Signed-off-by: Pete Zaitcev <zaitcev@yahoo.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

----- drivers/block/ub.c (modified) -----

```
/*
 * This many LUNs per USB device.
 * Every one of them takes a host, see UB_MAX_HOSTS.
 */
#define UB_MAX_LUNS 4
#define UB_MAX_LUNS 9

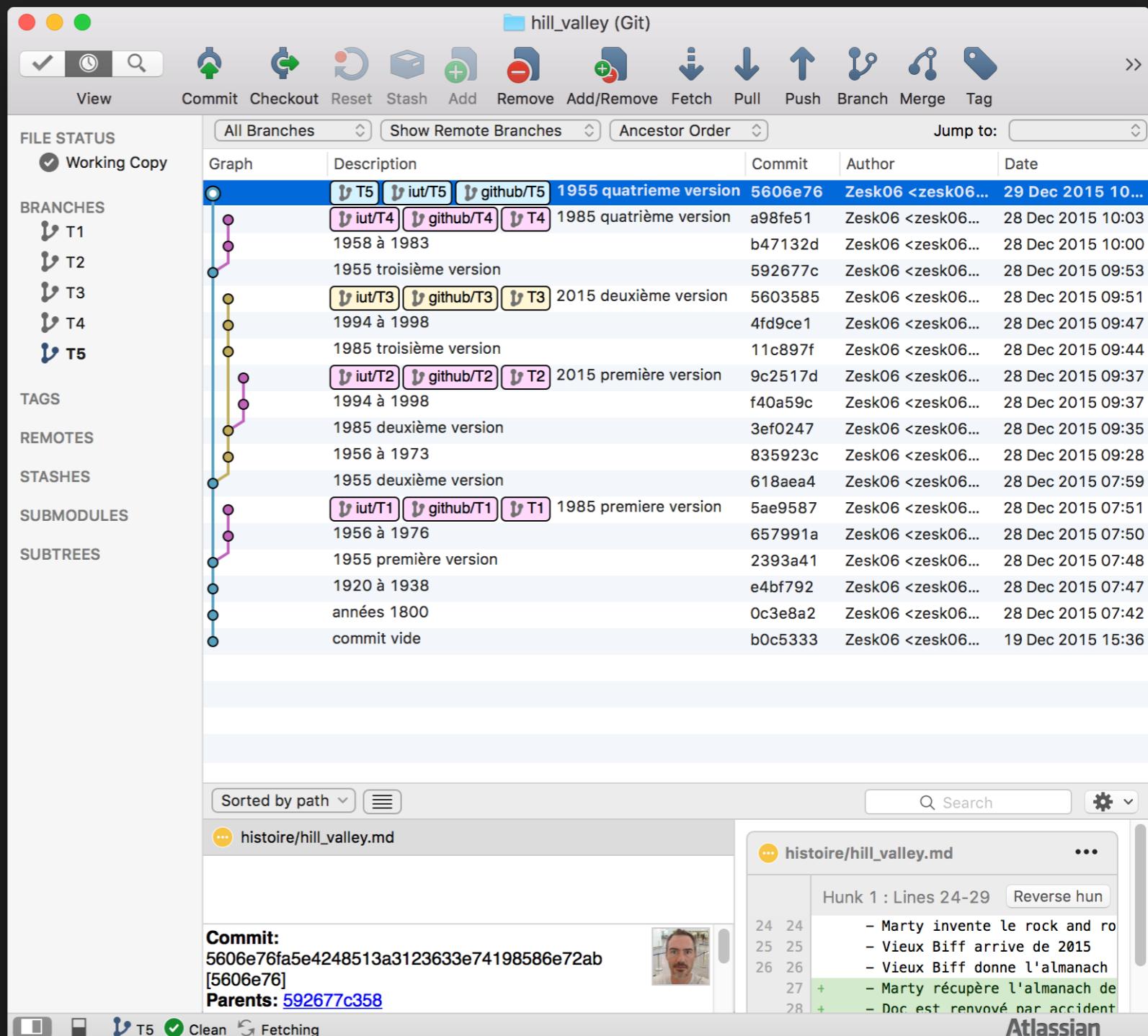
/*
 */
#define UB_MINORS_PER_MAJOR 8
2098 2098 static int ub_probe(struct usb_interface
)
    if (rc != 0) {
        nluns = rc;
        break;
    }
    mdelay(100);
    msleep(100);
}

for (i = 0; i < nluns; i++) {
    ub_probe_lun(sc, i);
}
```

The code editor shows a diff view of the `drivers/block/ub.c` file, highlighting changes in red and additions in green. The commit message also includes a note about increasing the number of LUNs from 4 to 9 and replacing `mdelay` with `msleep`.

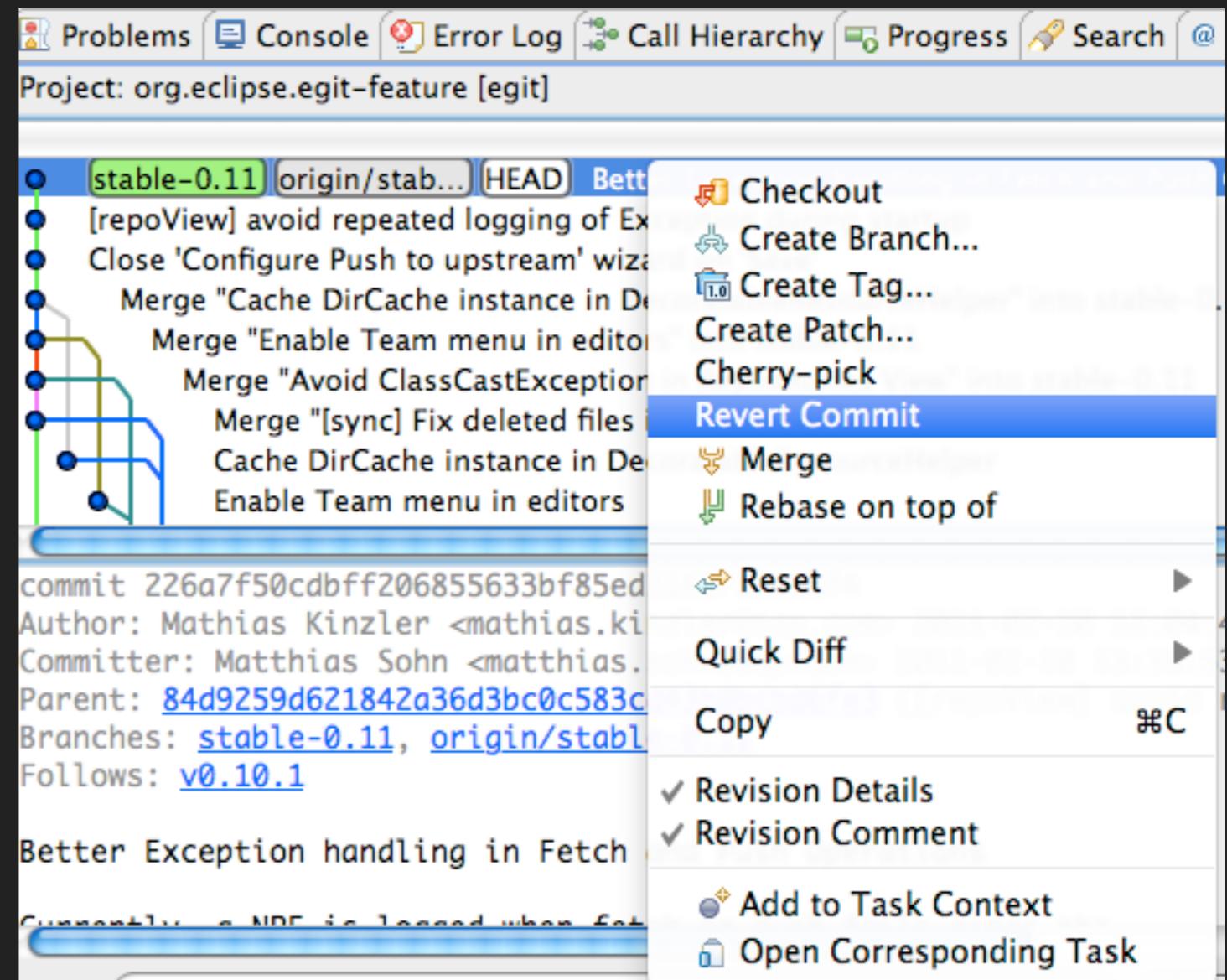
SOURCETREE

- ▶ Atlassian
- ▶ win/mac
- ▶ gitflow
- ▶ mercurial



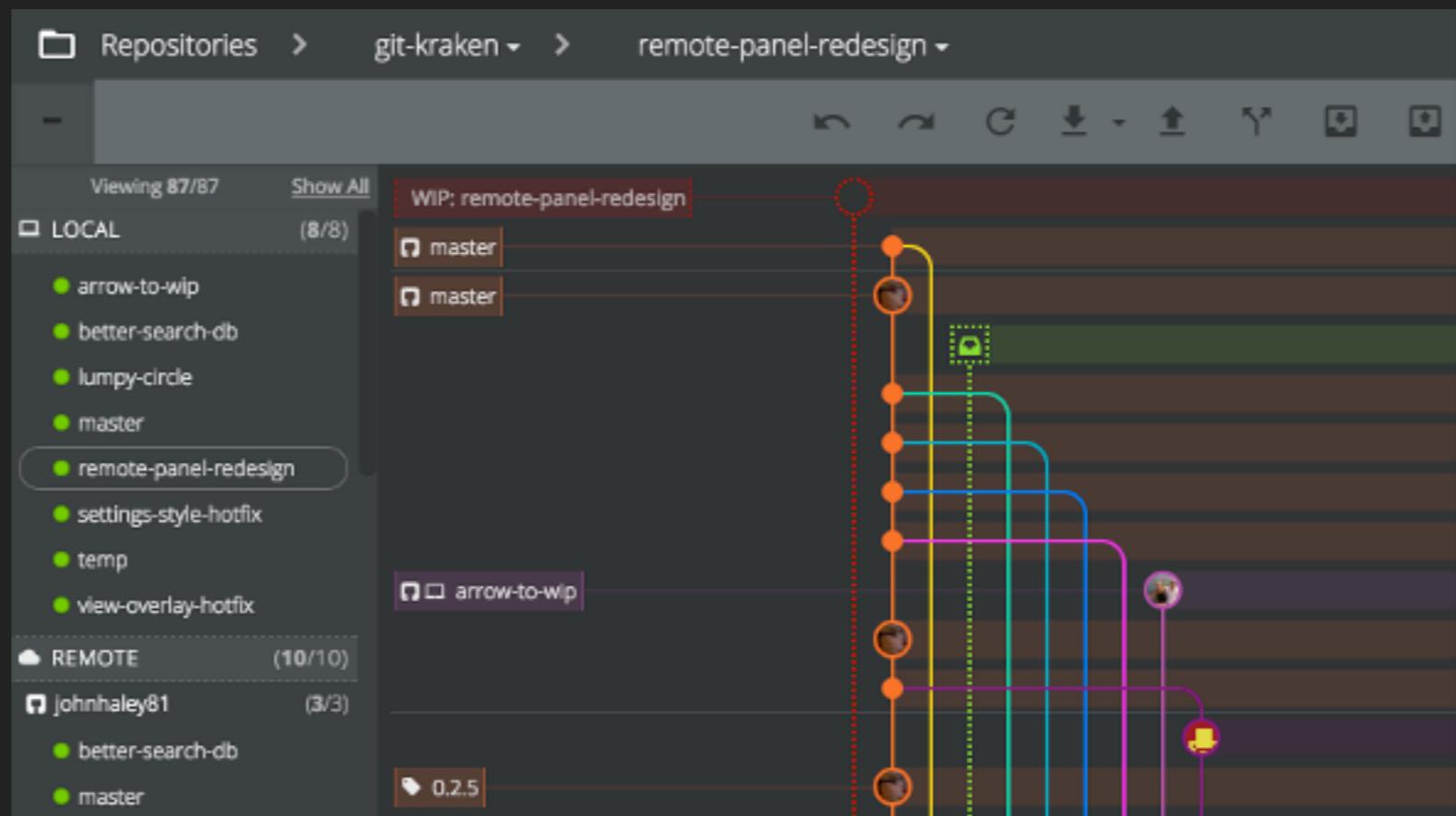
ECLIPSE EGIT

- ▶ eclipse
- ▶ win/linux/mac
- ▶ integrated



GITKRAKEN

- ▶ axosoft
- ▶ win/mac/linux
- ▶ free for NC use
- ▶ 60\$ / year



<https://www.gitkraken.com/>

SUBLIME MERGE

- ▶ win/mac/linux
- ▶ essai illimité
- ▶ ou
- ▶ 99\$ perm
- ▶ 75\$ / year

The screenshot shows the Sublime Merge application interface. At the top, it displays a file path: `~/git/delorean-java`. Below this, the main window has two panes: a tree view on the left and a code editor on the right.

Tree View (Left):

- No changes
- Commit Changes
- 2 improve README.md (Emmet Brown, develop 2, Tue, 17 Oct 2017)
- run coverage before tests (Emmet Brown, Tue, 17 Oct 2017)
- 7 some changes (Emmet Brown, Tue, 17 Oct 2017)
- Pin build image to maven:3.5 (Emmet Brown, Sun, 1 Oct 2017)
- 2 Merge branch 'date' into develop (Emmet Brown, http/develop, Wed, 19 Oct 2016)
- Test the date functions (Emmet Brown, date github/date, Wed, 19 Oct 2016)
- implements setYear/getYear (Emmet Brown, Wed, 19 Oct 2016)
- 2 Merge branch 'plutonium' into develop (Emmet Brown, Wed, 19 Oct 2016)
- 2 test plutonium (Emmet Brown, plutonium github/plutonium, Wed, 19 Oct 2016)
- addPlutonium method (Emmet Brown, Wed, 19 Oct 2016)
- 2 Merge branch 'set_speed' into develop (Emmet Brown, Wed, 19 Oct 2016)
- tested speed methods (Emmet Brown, set_speed github/set_speed, Wed, 19 Oct 2016)
- Ajout de setSpeed / getSpeed (Emmet Brown, Wed, 19 Oct 2016)
- 2 Added Delorean class (Emmet Brown, Wed, 19 Oct 2016)
- 2 CI setup (Emmet Brown, Wed, 19 Oct 2016)
- 6 maven archetype simple (Emmet Brown, master 2, Wed, 19 Oct 2016)
- commit vide (Emmet Brown, Wed, 19 Oct 2016)

Code Editor (Right):

```

public class Delorean {
    private int speed;
    private boolean plutonium = false;

    /**
     * @param speed: The speed in m/s
     */
    public void insertPlutonium() {
        this.plutonium = true;
    }

    public boolean hasPlutonium() {
        return this.plutonium;
    }

    public static void main( String[] args ) {
        System.out.println("Doc construit une delorean");
        Delorean delo = new Delorean();
        //System.out.println("Nous sommes le %s" % delo.get_date());

        System.out.println("Doc refait le plein de plutonium");
        //delo.insert_plutonium();
        delo.insertPlutonium();

        System.out.println("Doc programme un voyage dans le temps pour retourner en 1955");
        //delo.set_target_date("05/11/1955");
    }
}

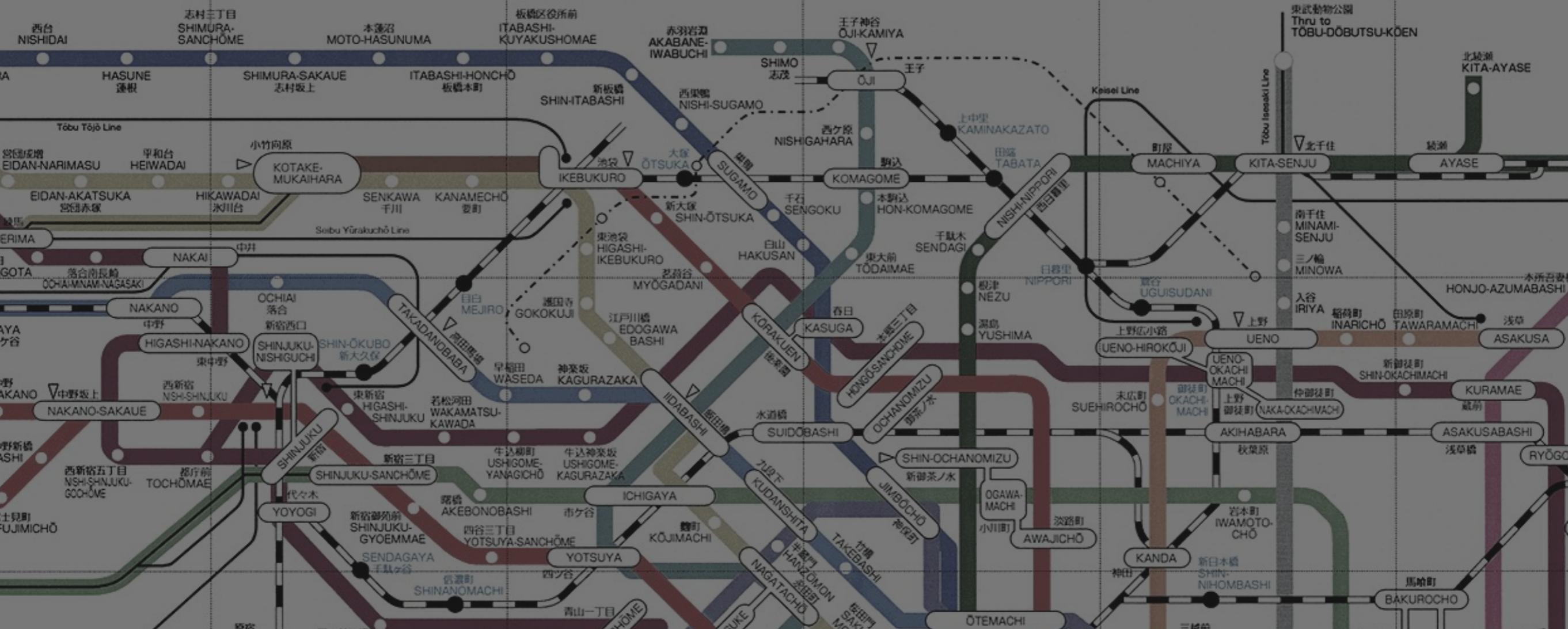
```

The code editor shows Java code for a `Delorean` class with methods for inserting plutonium and setting speed. The code is color-coded for syntax highlighting.

<https://www.sublimemerge.com/>

AUTRES CLIENTS NOTABLES

- ▶ linux:
 - ▶ git-cola
 - ▶ gitg
- ▶ windows:
 - ▶ tortoise-git
 - ▶ git-extensions



LA ROUTE ? LÀ OÙ ON VA, ON N'A PAS BESOIN DE ROUTE !

GIT EN SOLO

INSTALLEZ GIT!

- ▶ Download tools
- ▶ Install tools

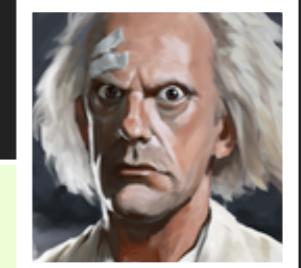
```
doc@labo$ sudo apt-get install git gitk gitg  
doc@labo$ sudo dpkg -i Downloads/*.deb
```



git help [command]

- ▶ Affiche l'aide de command

```
doc@labo$git help help
```



GIT-HELP(1)

Git Manual

GIT-HELP(1)

NAME

git-help – Display help information about Git

SYNOPSIS

`git help [-a|--all] [-g|--guide]`

a

DESCRIPTION

With no options and no COMMAND or GUIDE given, the synopsis of the `git` command and a list of the most commonly used Git commands are printed on the standard output.

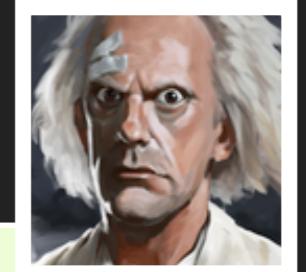
If the option `--all` or `-a` is given, all available commands are printed on the standard output.

If the option `--guide` or `-g` is given, a list of the useful Git guides is also printed on the standard output.

```
git config --global user.name "Emmett Brown"  
git config --global user.email "emmett.brown@hill_valley.com"
```

- ▶ Configurer son nom et son mail

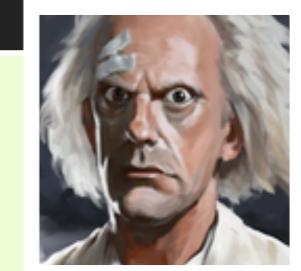
```
doc@labo$git config --global user.name "Emmett Brown"  
doc@labo$git config --global user.email "emmett.brown@hill_valley.com"
```



git init [path]

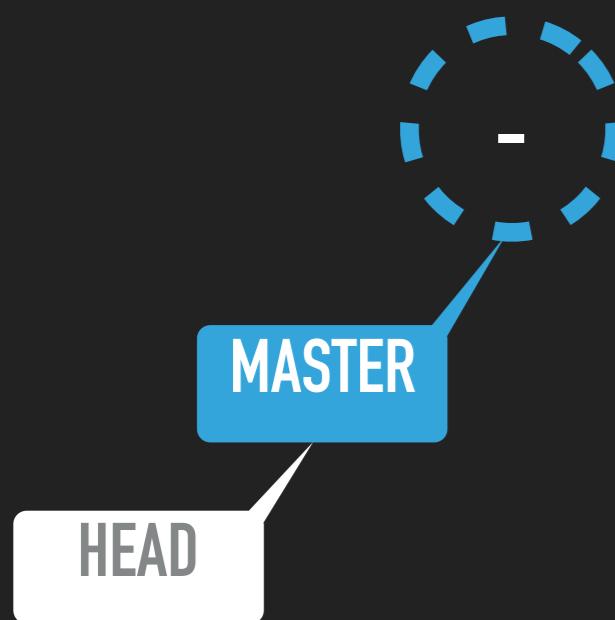
- ▶ Initialise une copie locale *working copy* dans le dossier path

```
doc@labo$ cd  
doc@labo$ mkdir git  
doc@labo$ cd git  
doc@labo$ git init test  
Initialized empty Git repository in /Users/doc/git/test/.git/  
doc@labo$ cd test/
```



git init [path]

- ▶ Initialise une copie locale *working copy* dans le dossier path
- ▶ Crée un dossier .git dans path



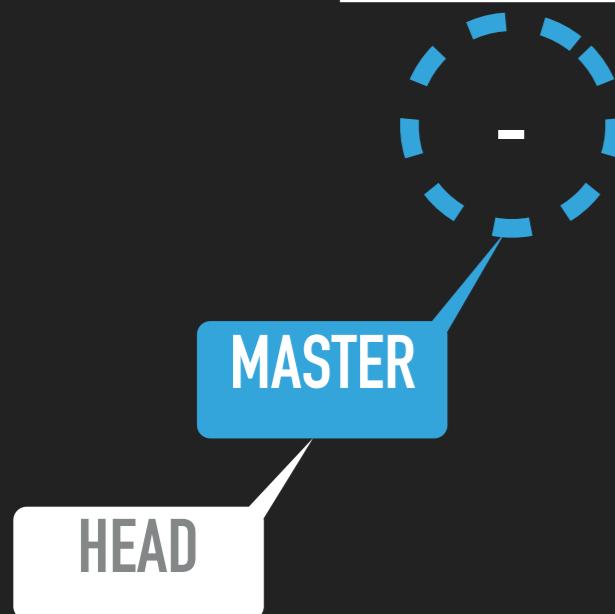
git status

- ▶ Donne le **status** des fichiers de la working copy

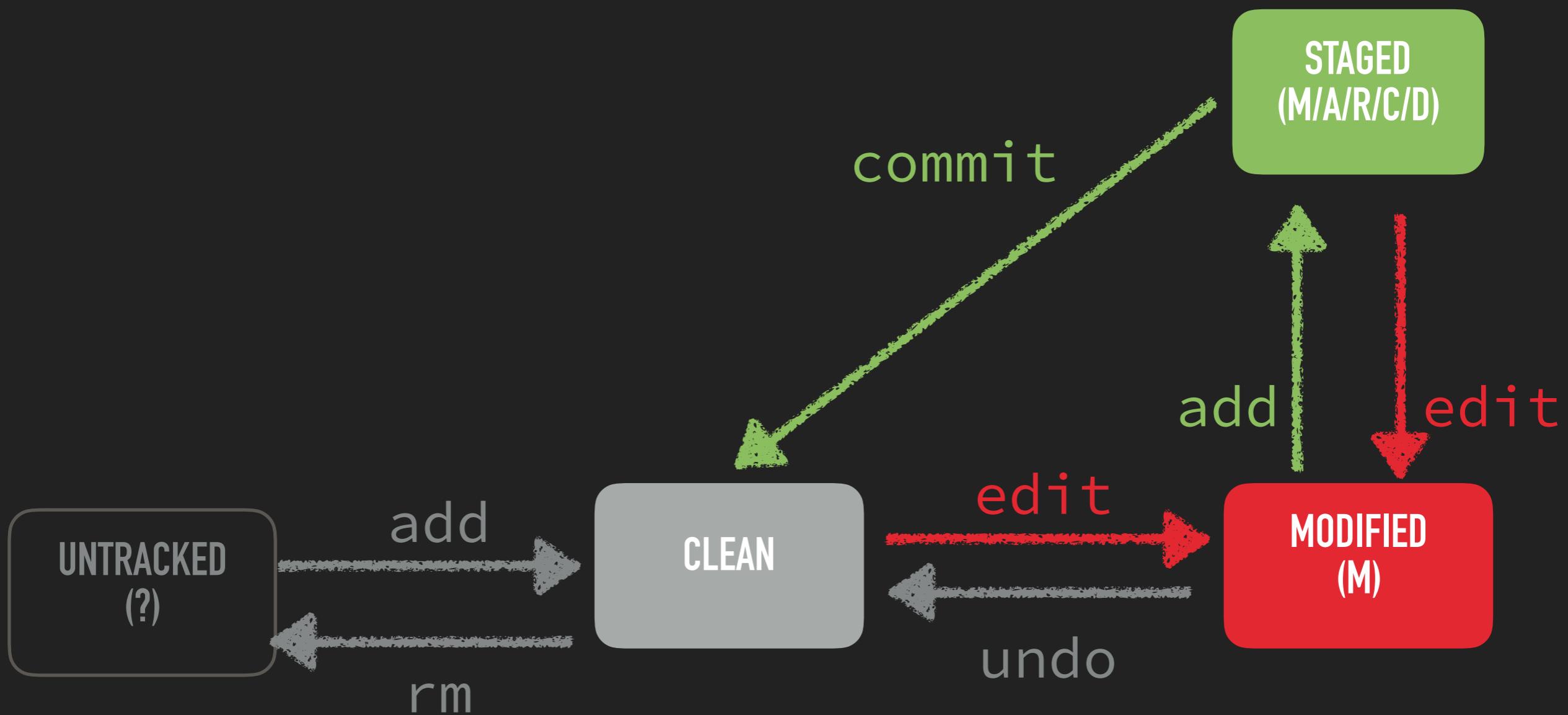
```
doc@labo$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

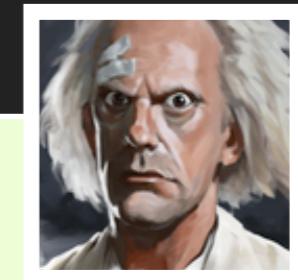


LES ÉTATS D'UN FICHIER



git add file...

- ▶ Ajoute le/les fichiers ou les modifications de fichier à l'index



```
doc@labo$ echo "# README" > README.md  
doc@labo$ git status  
On branch master
```

Initial commit

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

README.md

nothing added to commit but untracked files present (use "git add" to track)

```
doc@labo$ git add README.md  
doc@labo$ git status  
On branch master
```

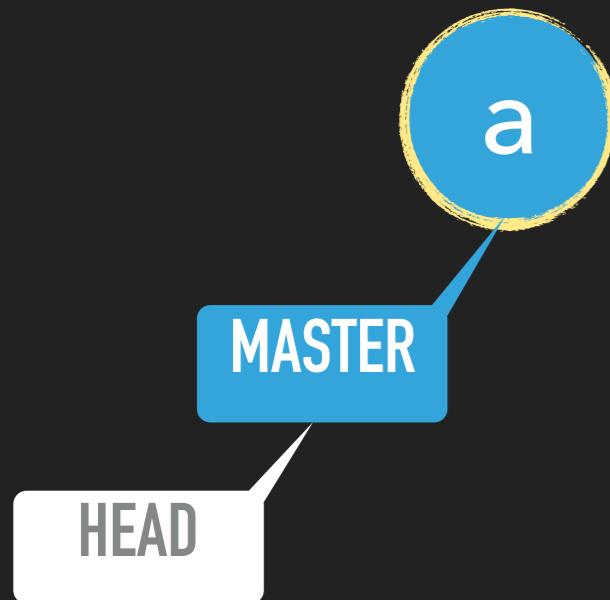
Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

new file: README.md

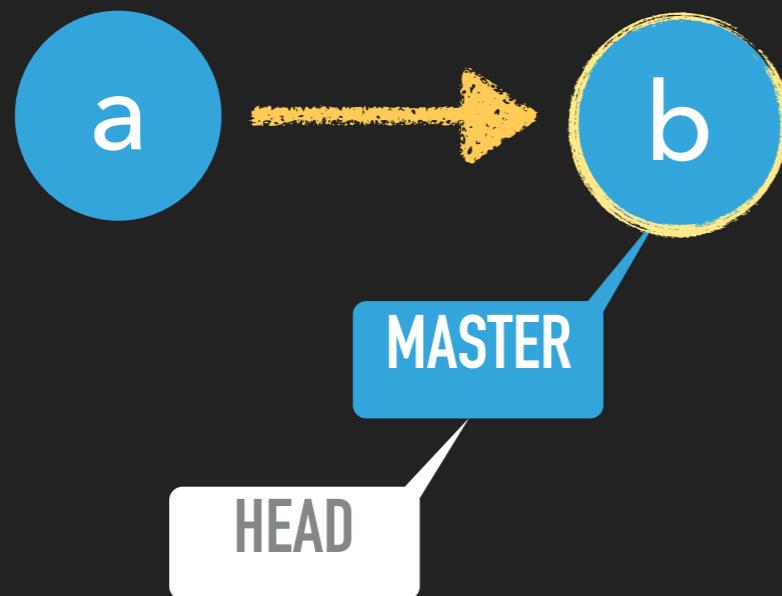
git commit -m “commentaire”

- ▶ Crée un object **commit** avec le message **commentaire**
- ▶ Se base sur les modifications présentes dans l'**index / stage**

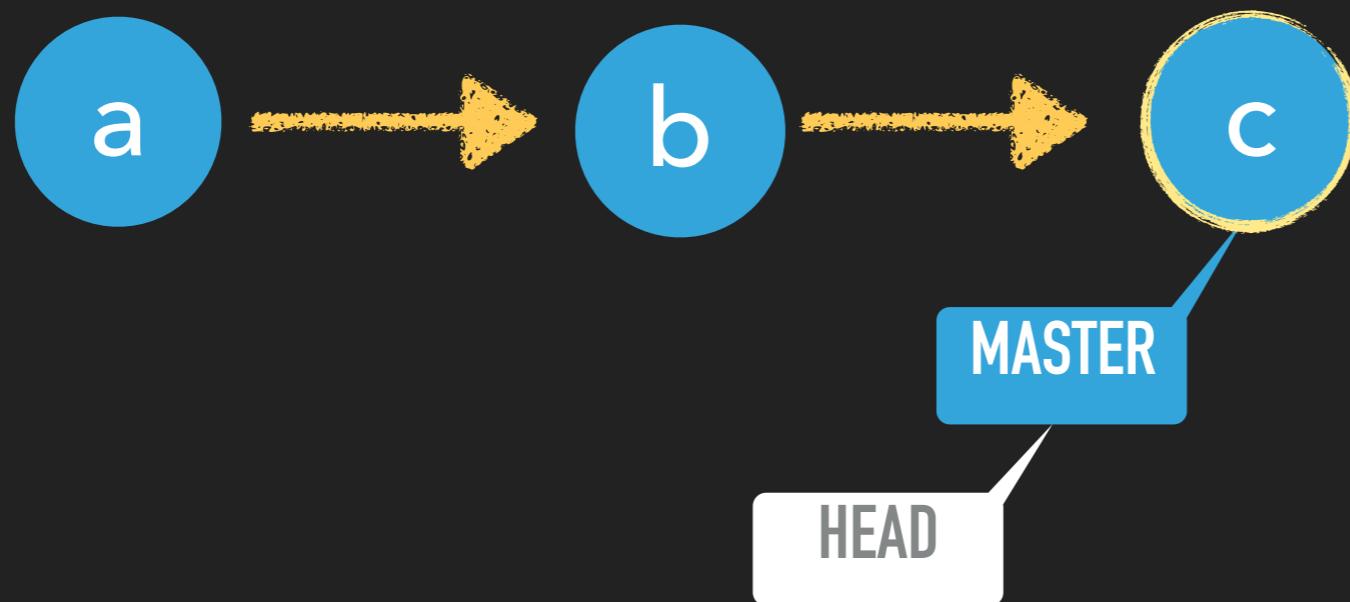


git commit -m “deuxième commit”

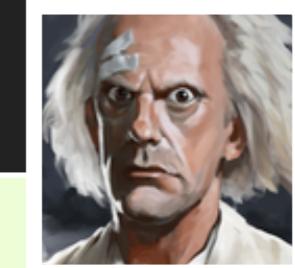
- ▶ Le nouveau commit a pour parent le commit pointé par HEAD
- ▶ La branche pointée par HEAD se déplace sur ce nouveau commit



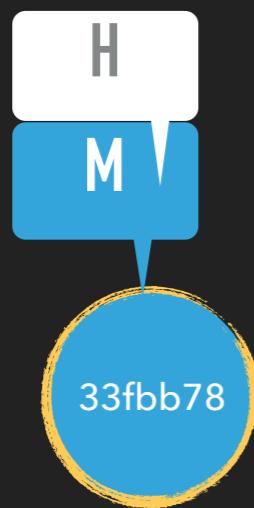
git commit -m “et encore un”



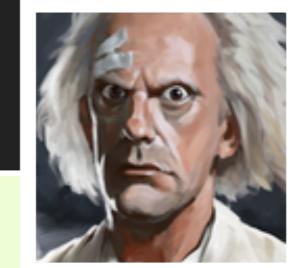
TROIS COMMITS



```
doc@labo$ doc@labo$ git commit -m "commentaire"  
[master (root-commit) 33fbb78] commentaire  
 1 file changed, 1 insertion(+)  
 create mode 100644 README.md
```

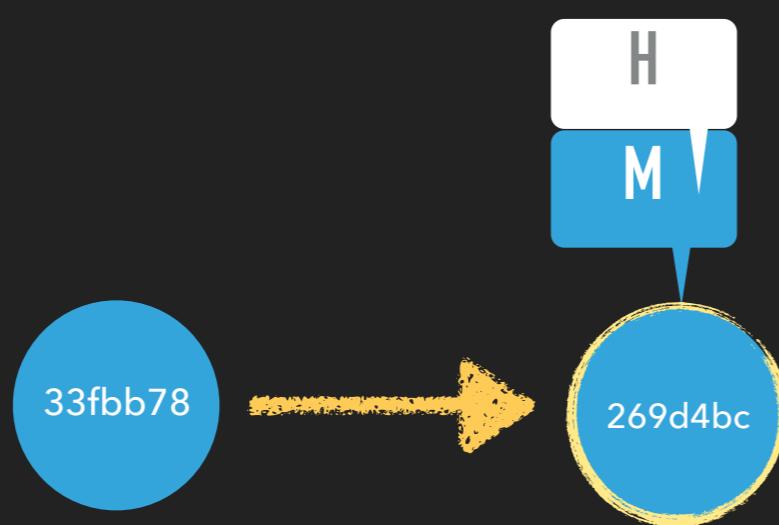


TROIS COMMITS

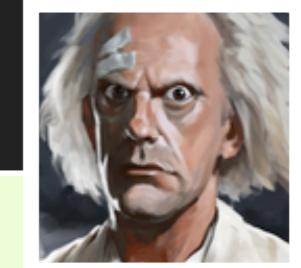


```
doc@labo$ doc@labo$ git commit -m "commentaire"  
[master (root-commit) 33fbb78] commentaire  
 1 file changed, 1 insertion(+)  
 create mode 100644 README.md
```

```
doc@labo$ echo "some text" >> README.md  
doc@labo$ git add README.md  
doc@labo$ git commit -m "deuxieme commit"  
[master 269d4bc] deuxieme commit  
 1 file changed, 1 insertion(+)
```



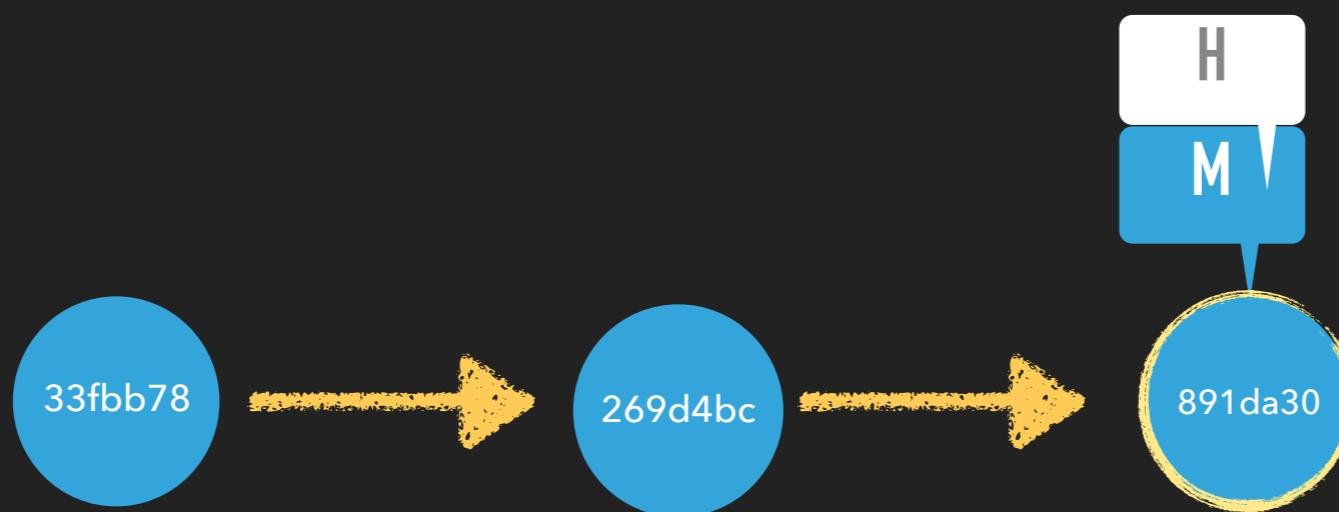
TROIS COMMITS



```
doc@labo$ doc@labo$ git commit -m "commentaire"  
[master (root-commit) 33fbb78] commentaire  
 1 file changed, 1 insertion(+)  
 create mode 100644 README.md
```

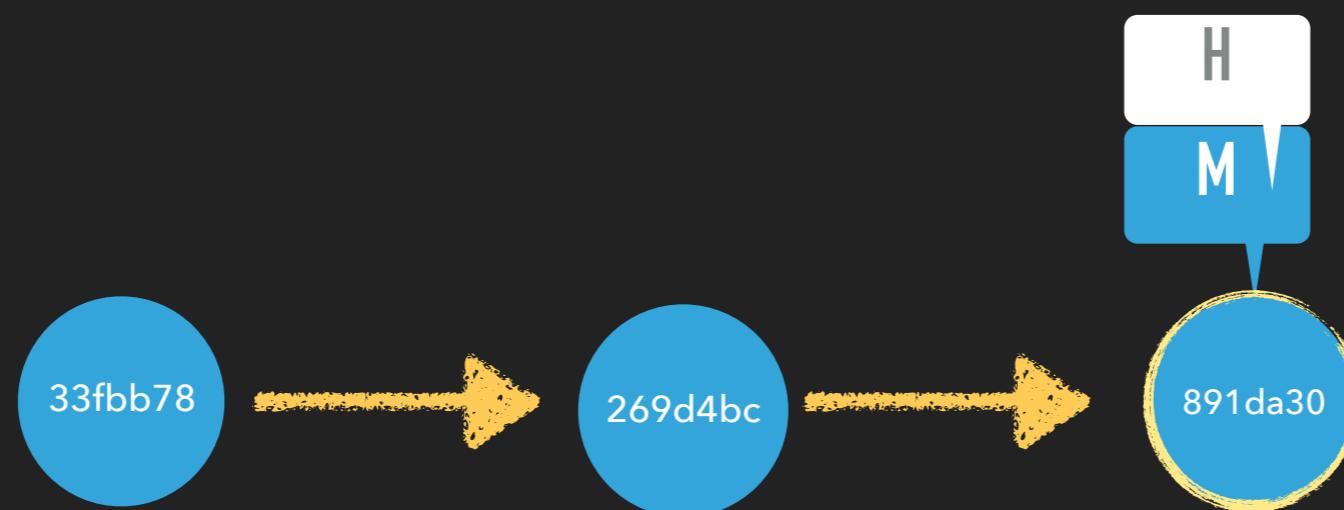
```
doc@labo$ echo "some text" >> README.md  
doc@labo$ git add README.md  
doc@labo$ git commit -m "deuxieme commit"  
[master 269d4bc] deuxieme commit  
 1 file changed, 1 insertion(+)
```

```
doc@labo$ echo "text again" >> README.md  
doc@labo$ git add README.md  
doc@labo$ git commit -m "et encore un"  
[master 891da30] et encore un  
 1 file changed, 1 insertion(+)
```



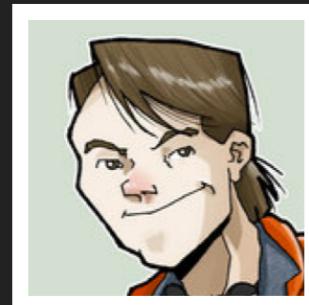
LE SHA1

- ▶ Identifiant de commit
- ▶ Fonction de HASH sur le PATCH, le commiter, la date
- ▶ Infalsifiable, Unique
- ▶ Sert de référence absolue, d'adresse pour trouver un commit



IGNORER DES FICHIERS

- ▶ On ne met **pas** dans la gestion de version:
 - ▶ Les produits de compilation: target, *.class, *.so, *.o
 - ▶ Les dépendances: *.jar, *.zip, *.exe



COMMENT IGNORER CES FICHIERS?

IGNORER DES FICHIERS

- ▶ Mettre les pattern à ignorer dans un fichier .gitignore
- ▶ Commiter ce fichier !

```
doc@labo$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    compiled.pyc
    foo.so
doc@labo$ echo "*.so" > .gitignore
doc@labo$ echo "*.pyc" >> .gitignore
doc@labo$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
doc@labo$ git add .gitignore
doc@labo$ git commit -m "Ajout du fichier gitignore"
[master 43fe137] Ajout du fichier gitignore
  1 file changed, 2 insertions(+)
  create mode 100644 .gitignore
doc@labo$ git status
On branch master
nothing to commit, working directory clean
```



IGNORER DES FICHIERS

- ▶ Une collection de .gitignore disponible
<https://github.com/github/gitignore>

EXERCICE 1.1

- ▶ Initialiser un repository git nommé `cours_git` dans votre dossier personnel
- ▶ Ajouter un fichier *README.md*
Décrire les principales caractéristiques de git
- ▶ <https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>
- ▶ <http://dillinger.io/> pour un éditeur online
- ▶ Ajouter et commiter le *README.md*

README.md

GIT

Le gestionnaire de conf en vogue dans nos banlieues

Caractéristiques

- rapide
- ...

EXERCICE 1.2

- ▶ Créer un dossier *commandes* et à l'intérieur de celui-ci
- ▶ Pour **chaque** commande vue:
 - ▶ Créer un fichier *commandes/nom_commande.md*
 - ▶ Décrire la commande
 - ▶ L'ajouter et le commiter

init.md

```
# git init

git init [PATH]

## Description

Permet d'initialiser un repository git dans le dossier `PATH`


## Paramètres

– PATH: Le dossier de la working copy

## Examples d'utilisation

git init ~/git/cours_git
```

ADD / COMMIT

- ▶ Initialiser une working copy:

```
git init mondossier
```

- ▶ Voir le status:

```
git status
```

- ▶ Ajouter une modification sur fichier.txt:

```
git add fichier.txt
```

- ▶ Commiter le contenu du stage:

```
git commit -m "mon commentaire"
```



« MAIS MERDE, QUAND SONT-ILS ?! »

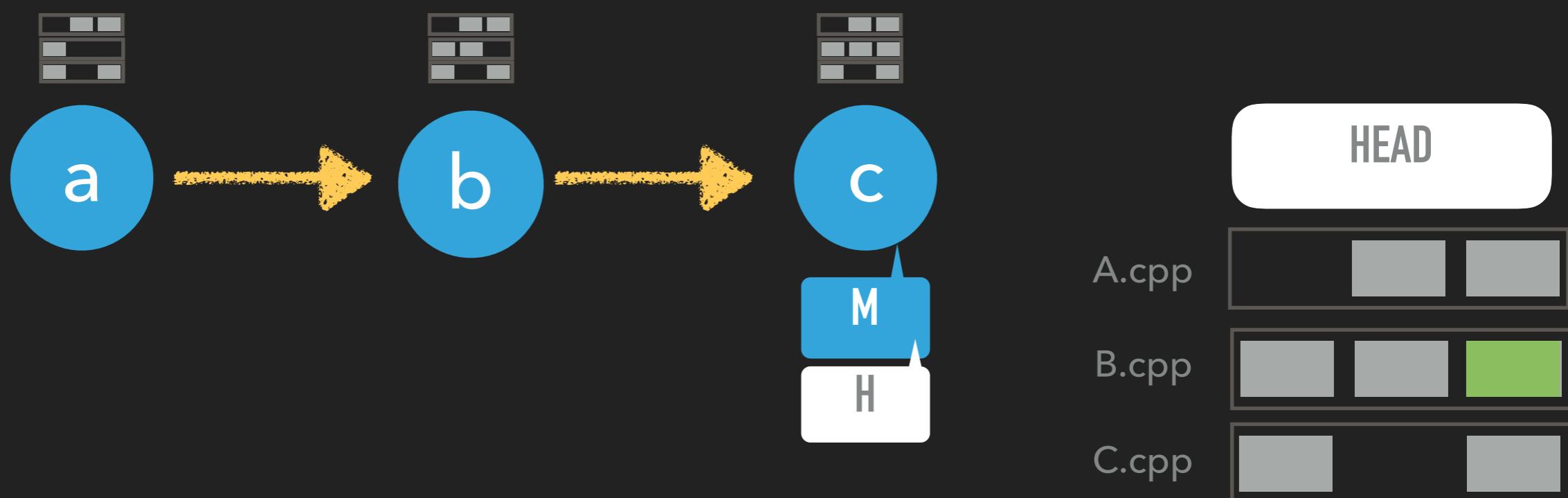
REVENIR EN ARRIERE

git reset [mode] [ref]

- ▶ mode:
 - ▶ --mixed: conserve les modifications (défaut)
 - ▶ --hard: abandonne TOUTES les modifications
- ▶ ref:
 - ▶ Le commit où l'on veut revenir, SHA1 ou référence relative
 - ▶ HEAD par défaut

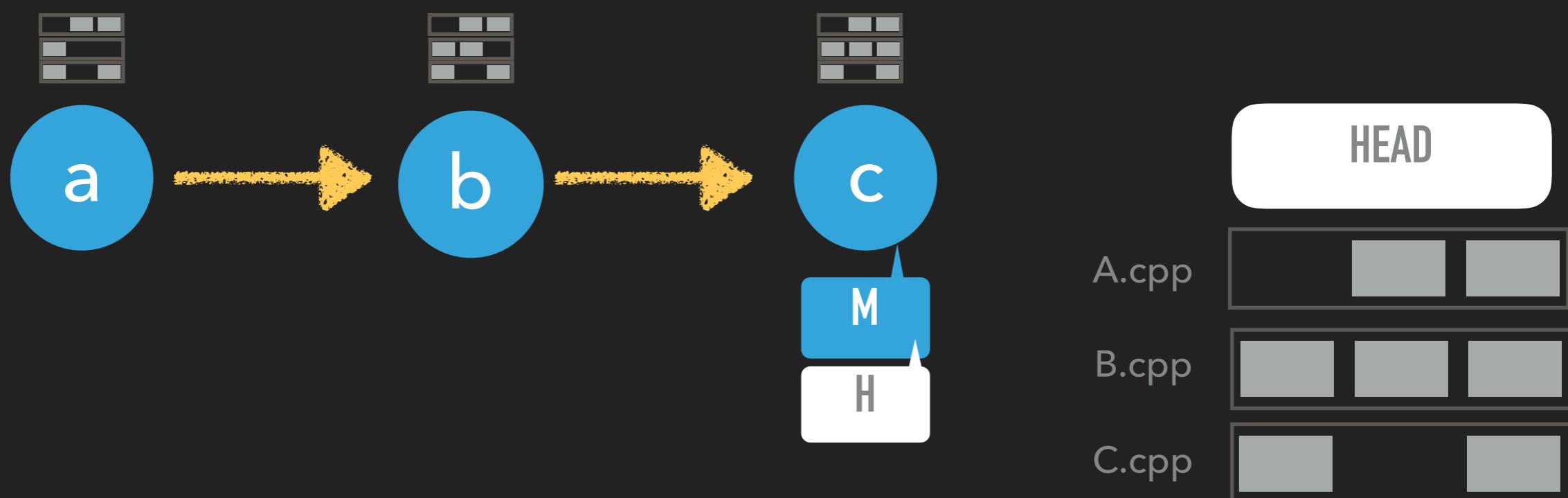
git reset [ref]

- ▶ Reset la branche courante sur la reference
- ▶ git reset b
- ▶ git reset HEAD~



git reset --hard [ref]

- ▶ Reset la branche courante sur la **reference**
 - ▶ git **reset --hard b**
 - ▶ git **reset --hard HEAD~**



LE MODE HARD

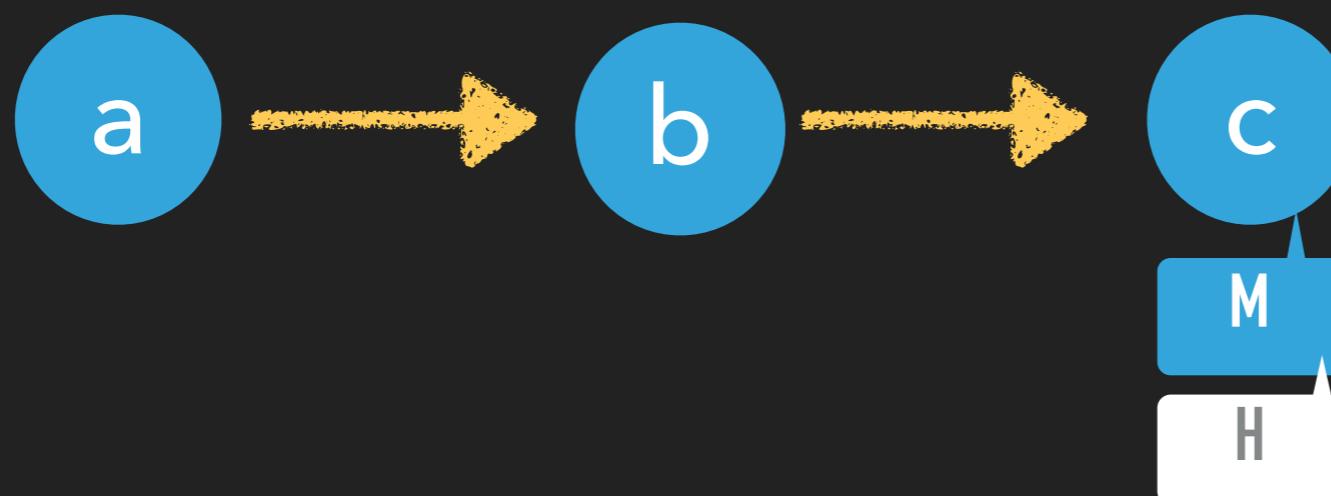
- ▶ abandonne TOUTES les modifications
- ▶ y compris les modifications en cours!
- ▶ pour toujours

ANCESTRY REFERENCES / REFERENCES RELATIVES

- ▶ ~: Le premier parent
- ▶ HEAD~: Le premier parent
- ▶ HEAD~~ ou HEAD~2: Le premier parent du premier parent (le grand père)
- ▶ ^*i*: Le *i*ème parent
- ▶ HEAD^: Le premier parent
- ▶ HEAD^2: Le second parent
- ▶ HEAD^^: Le premier parent du premier parent (le grand père)

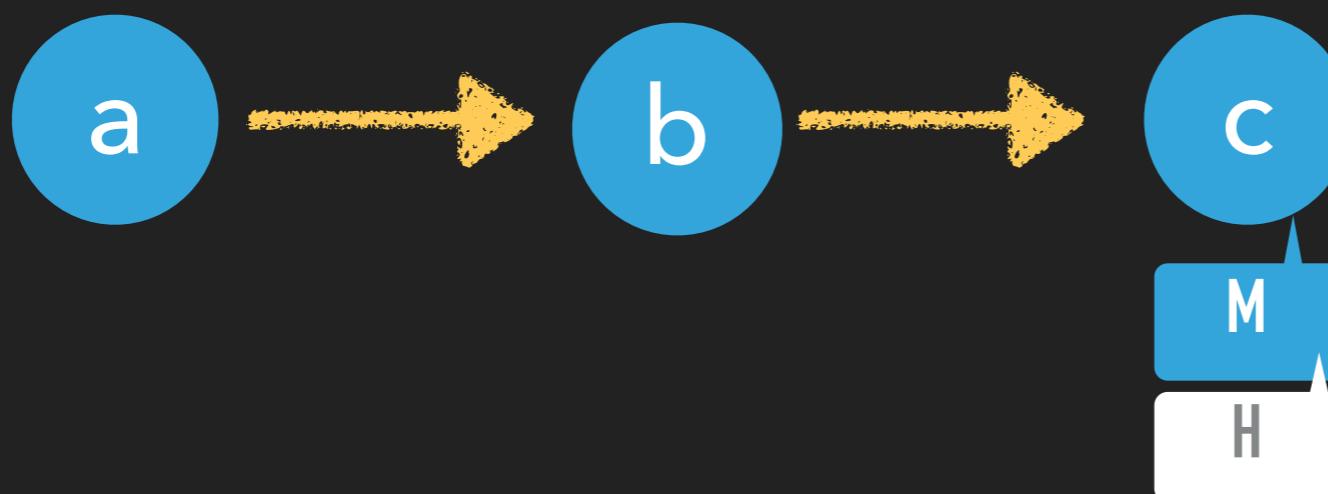
ANCESTRY REFERENCES / REFERENCES RELATIVES

- ▶ `git reset HEAD~~` or `git reset HEAD~2`
- ▶ `git reset HEAD^^`



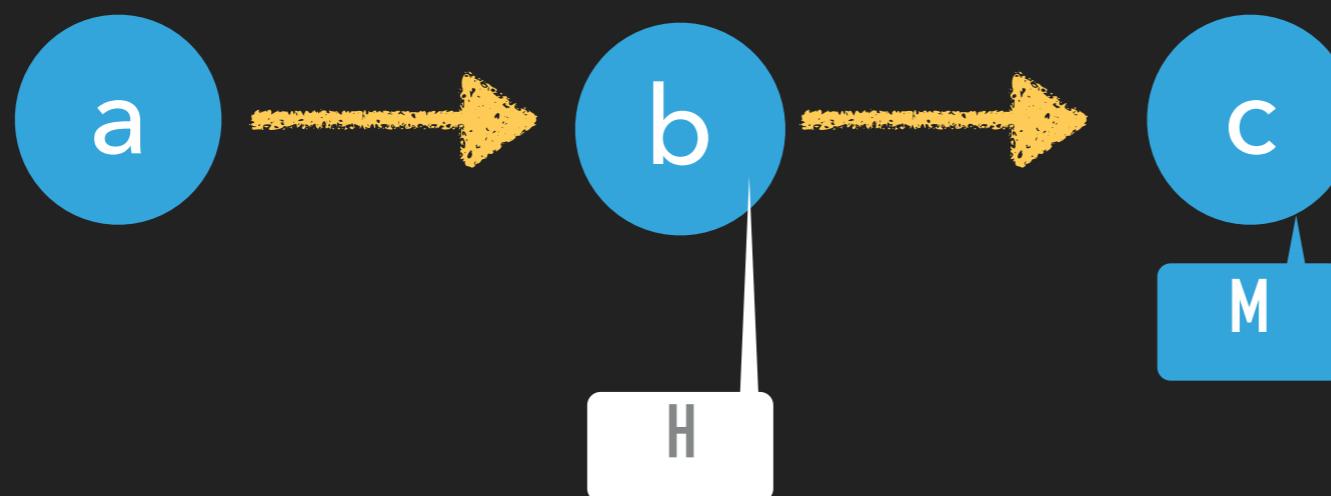
git checkout [ref]

- ▶ Déplace le HEAD sur la référence, branche ou commit
- ▶ Si des fichiers sont dans l'état modified, conserve ces modifications



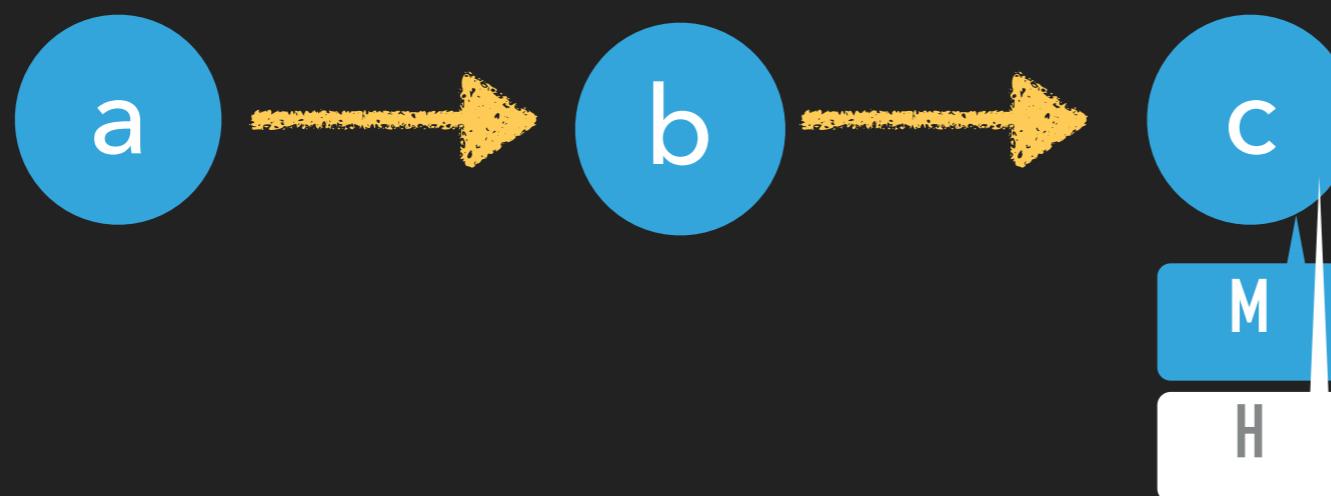
git checkout [ref]

- ▶ git checkout HEAD~



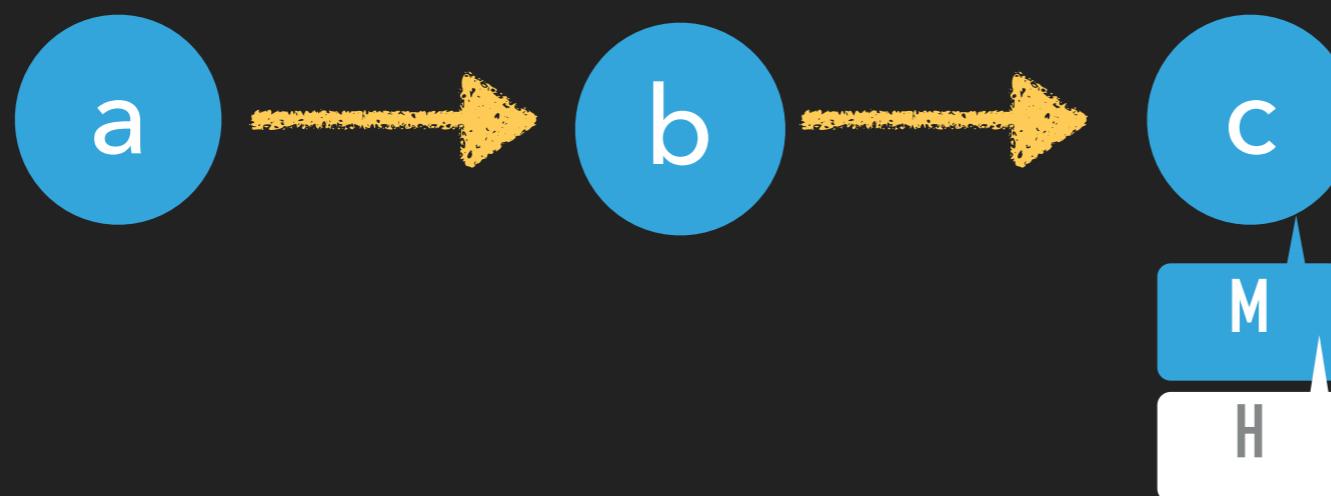
git checkout [ref]

► git checkout c



git checkout [ref]

- ▶ git checkout master



MODE TÊTE DÉTACHÉE

- ▶ Quand le HEAD ne pointe pas sur une branche
- ▶ Attention aux **commit** en tête détachée !

MODE TÊTE DÉTACHÉE

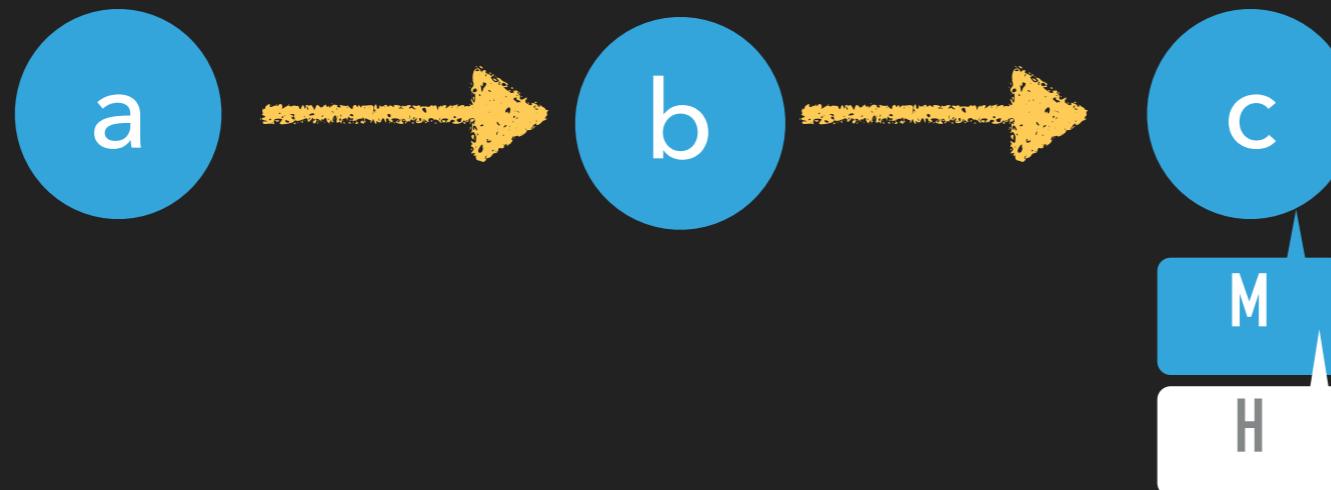


```
doc@labo$ git checkout HEAD~  
Note: checking out 'HEAD~'.
```

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 891da30... et encore un  
doc@labo$ git st  
HEAD detached at 891da30  
nothing to commit, working directory clean
```



MODE TÊTE DÉTACHÉE

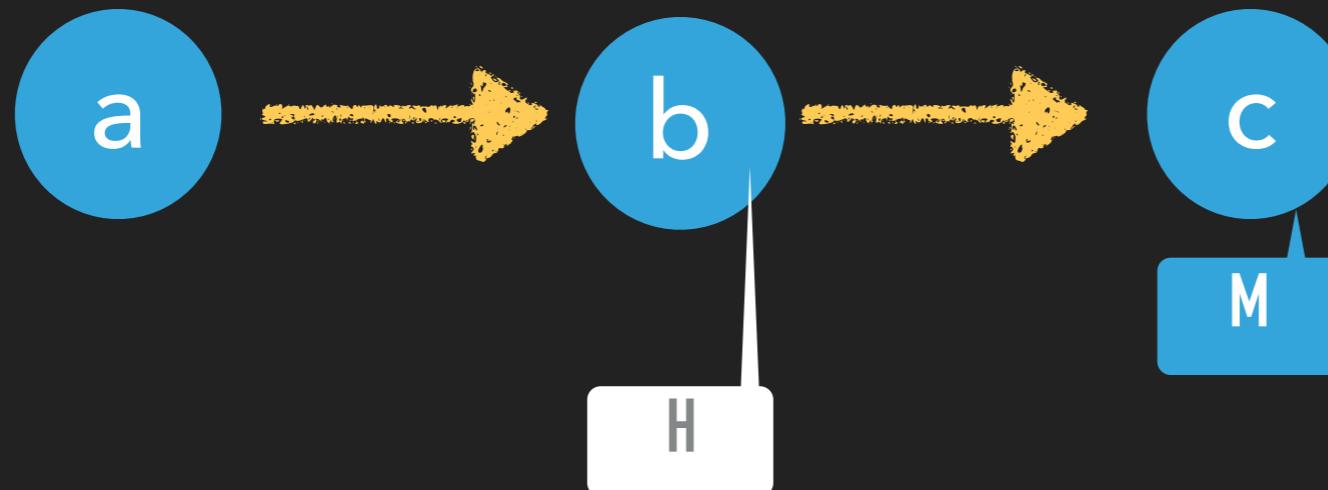


```
doc@labo$ git checkout HEAD~  
Note: checking out 'HEAD~'.
```

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:

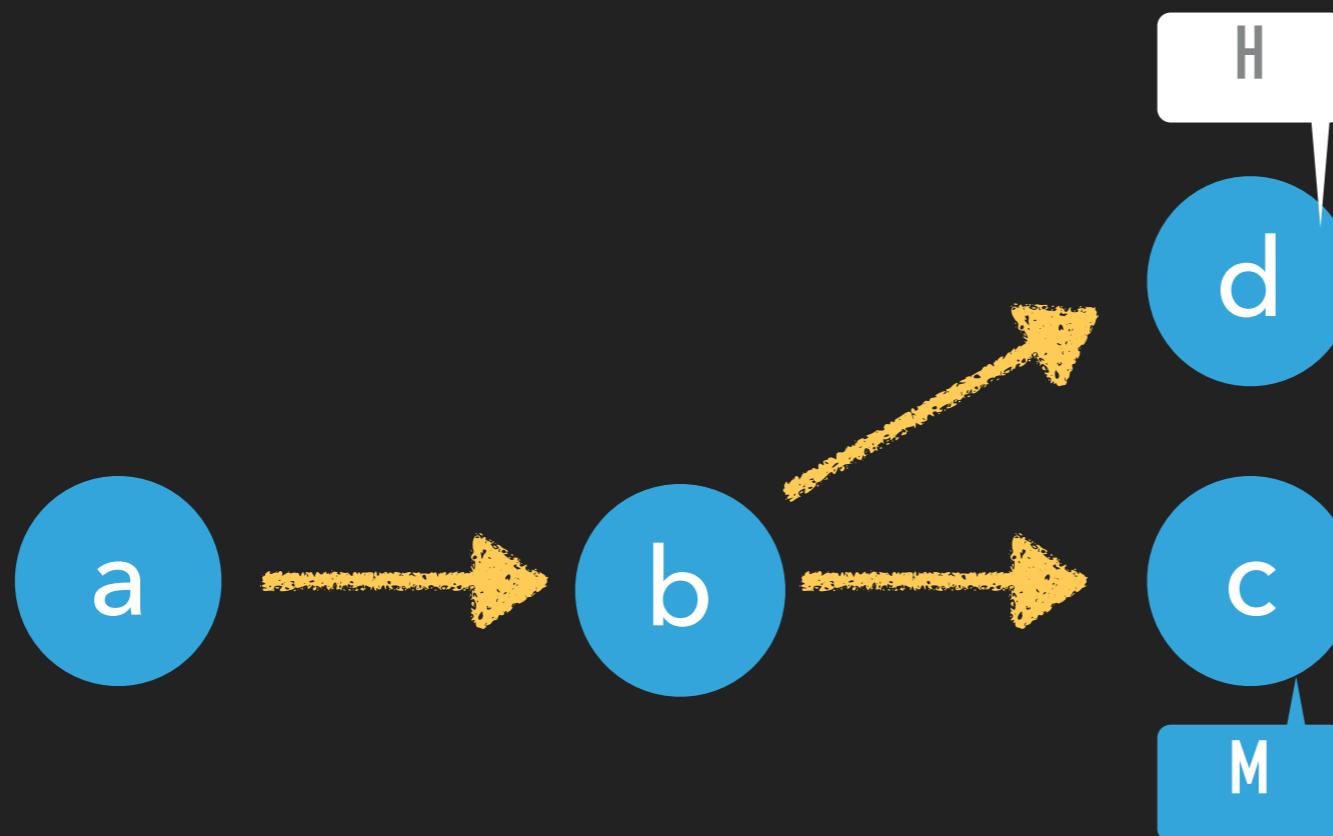
```
git checkout -b <new-branch-name>  
  
HEAD is now at 891da30... et encore un  
doc@labo$ git st  
HEAD detached at 891da30  
nothing to commit, working directory clean
```



MODE TÊTE DÉTACHÉE



```
doc@labo$ echo "Pas la tete" >> README.md
doc@labo$ git add README.md
doc@labo$ git commit -m "commit en tete detachee"
[detached HEAD dee2c9c] commit en tete detachee
 1 file changed, 1 insertion(+)
```



MODE TÊTE DÉTACHÉE

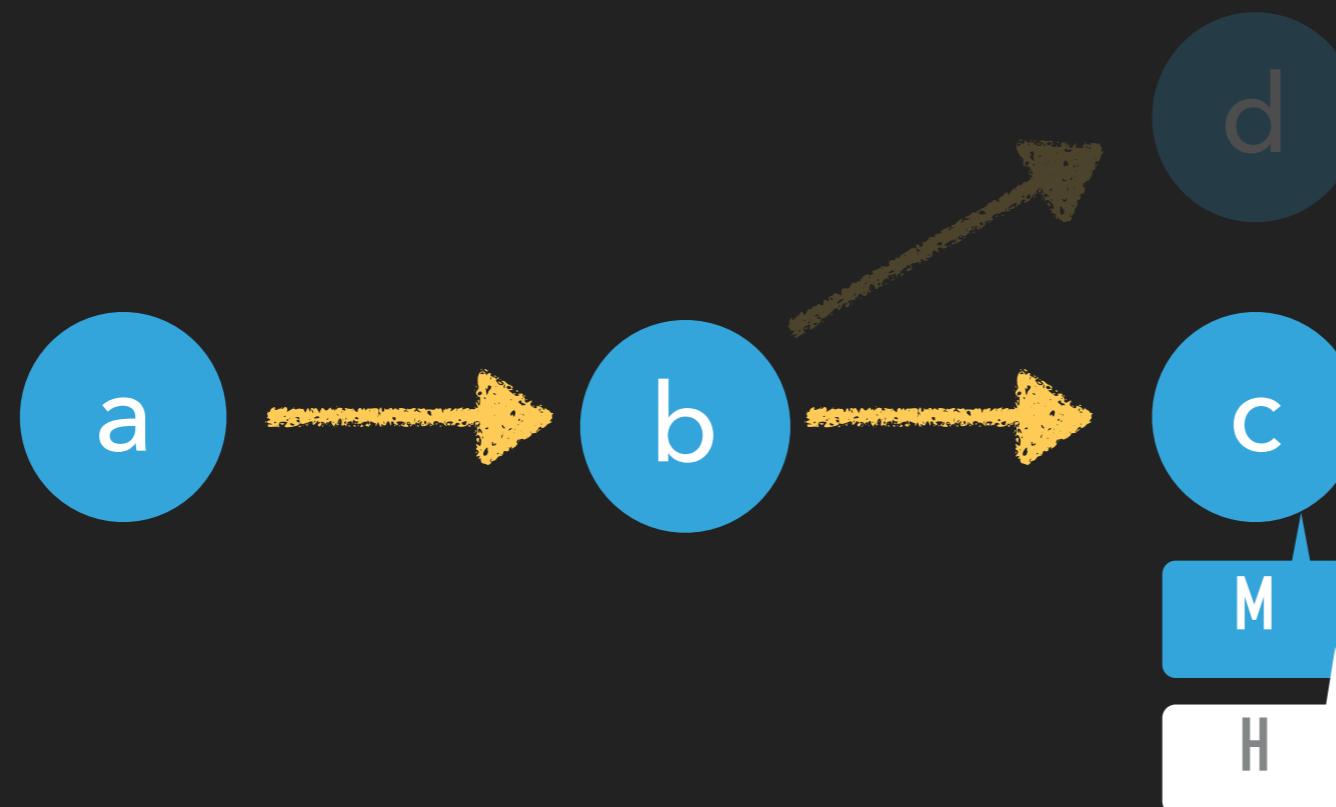
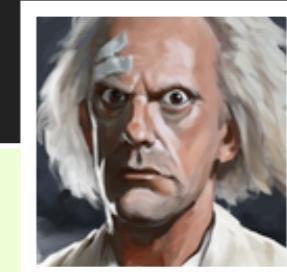
```
doc@labo$ git checkout master  
Warning: you are leaving 1 commit behind, not connected to  
any of your branches:
```

```
dee2c9c commit en tête détachée
```

If you want to keep it by creating a new branch, this may be a good time
to do so with:

```
git branch <new-branch-name> dee2c9c
```

```
Switched to branch 'master'
```

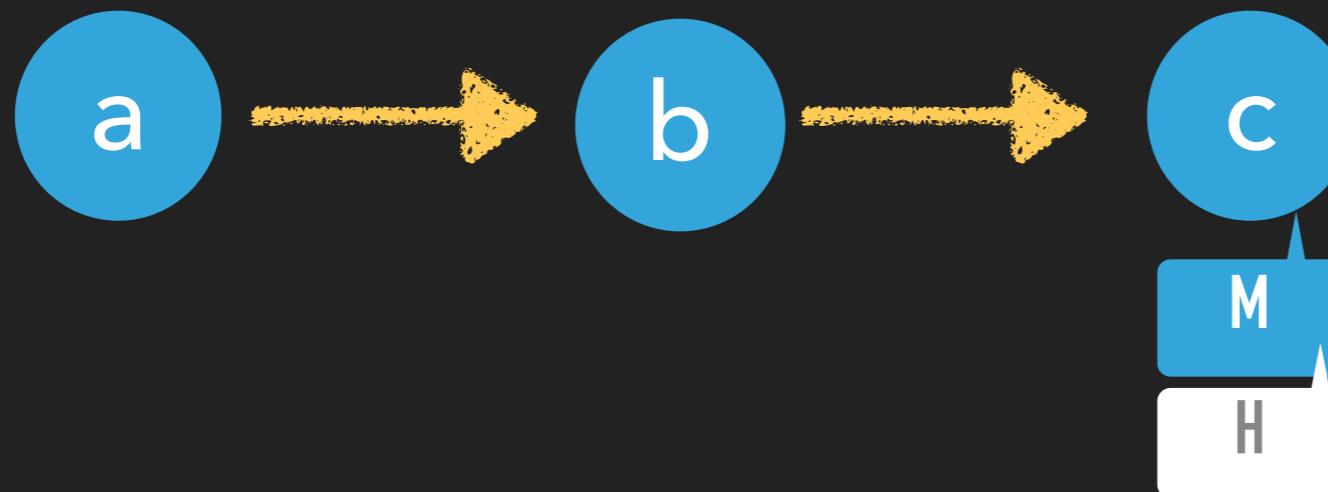


git branch

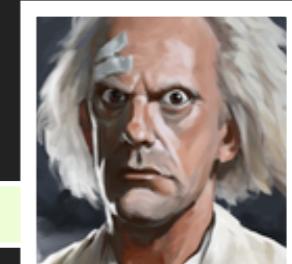
- ▶ Donne la liste des branches
- ▶ Donne la branche courante
- ▶ Permet de créer/renommer/déplacer/effacer des branches

git branch name [ref]

- ▶ Crée une branche qui porte le nom **name**
- ▶ Sur le commit pointé par la référence
- ▶ Si **ref** n'est pas précisé, crée une branche sur le commit pointé par HEAD



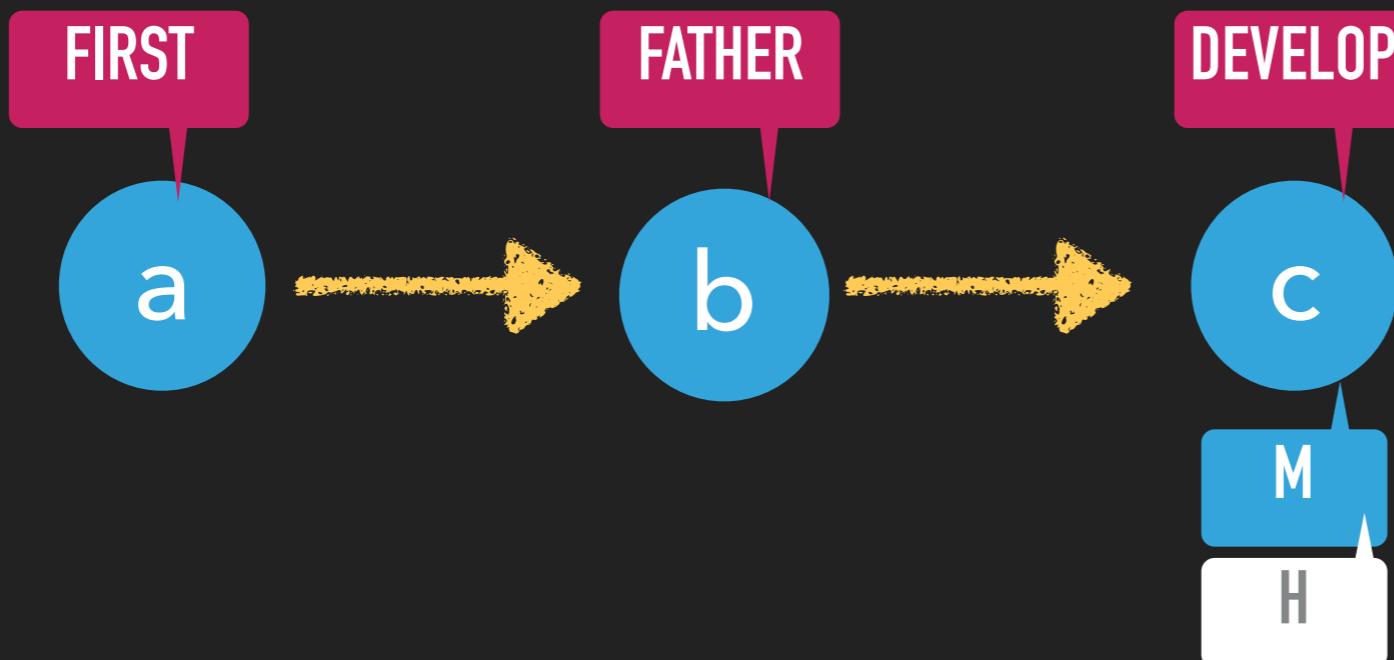
git branch name [ref]



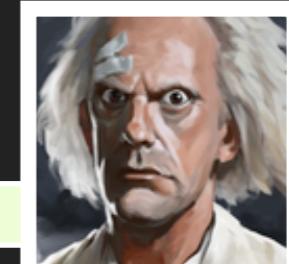
```
doc@labo$ git branch develop
```

```
doc@labo$ git branch father HEAD~
```

```
doc@labo$ git branch first a
```



git branch name [ref]

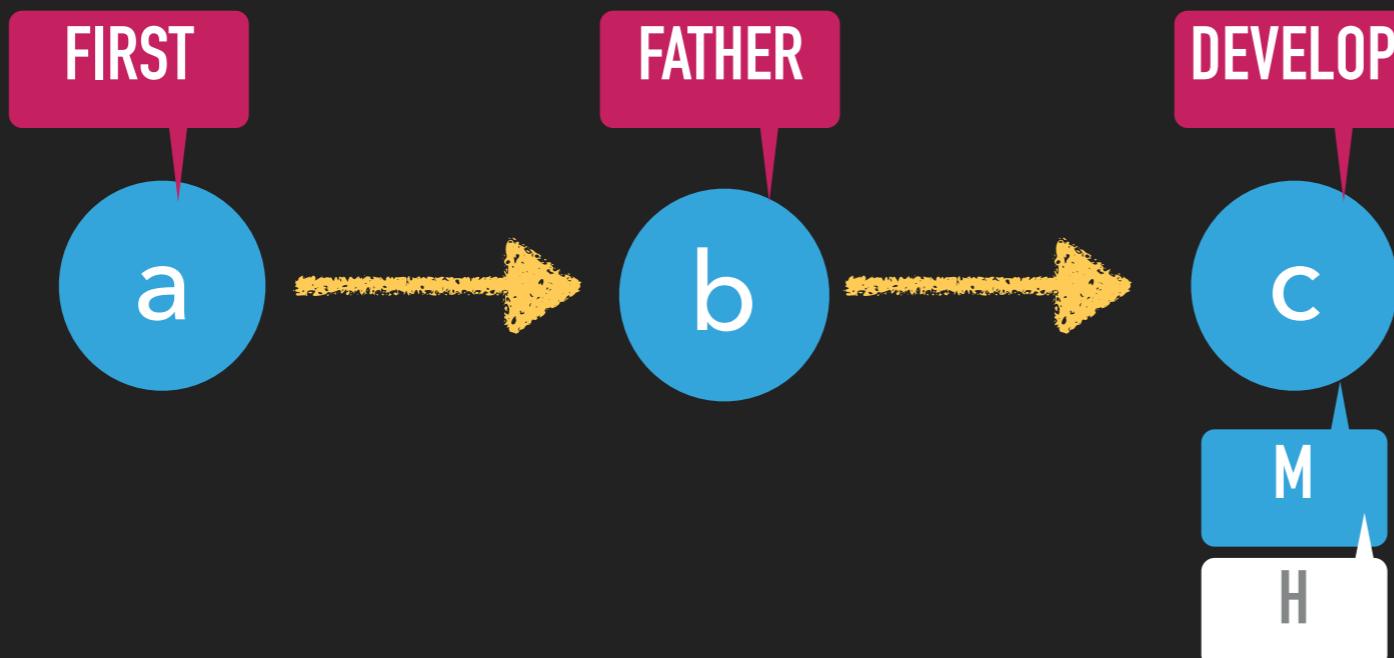


```
doc@labo$ git branch develop
```

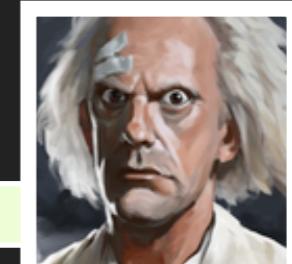
```
doc@labo$ git branch father HEAD~
```

```
doc@labo$ git branch first a
```

```
doc@labo$ git branch -f first HEAD~
```



git branch name [ref]

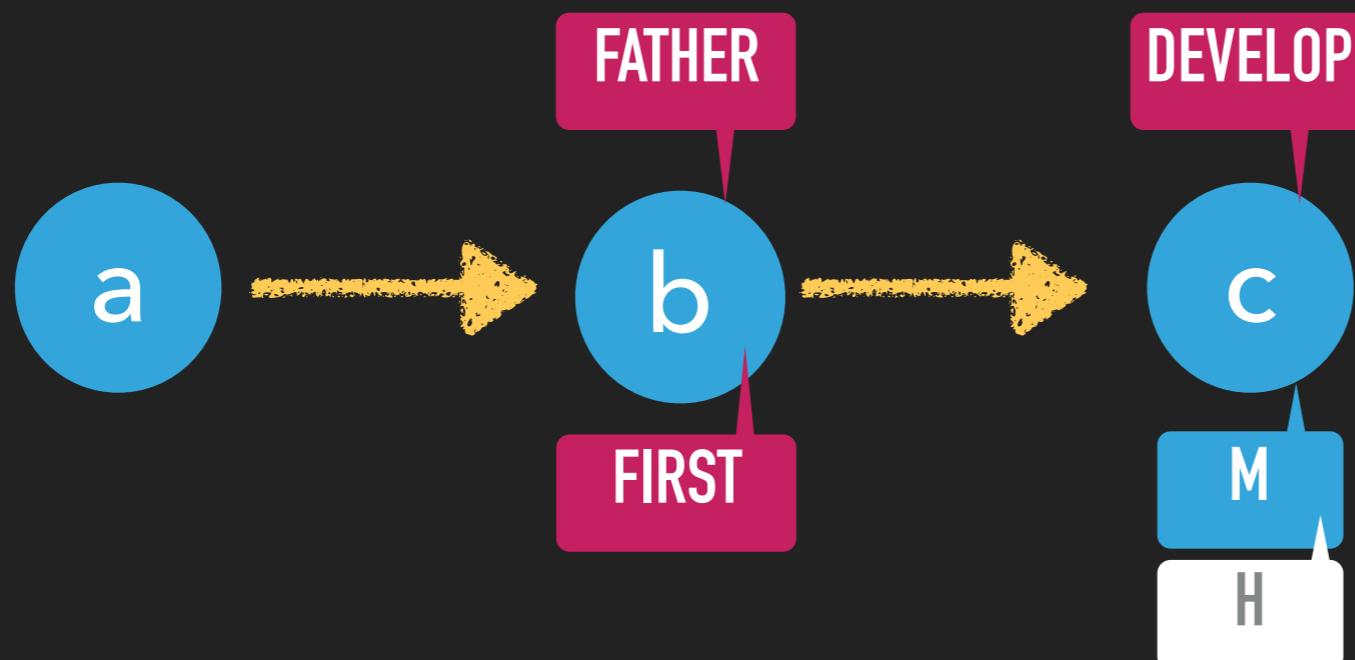


```
doc@labo$ git branch develop
```

```
doc@labo$ git branch father HEAD~
```

```
doc@labo$ git branch first a
```

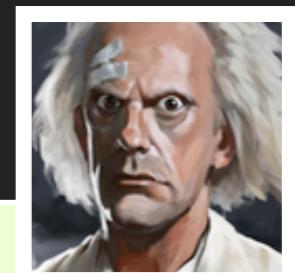
```
doc@labo$ git branch -f first HEAD~
```



EXERCICE 2.1: MAUVAIS COMMIT

- ▶ Reprendre le repository `cours_git`, branche `master`
- ▶ Rajouter un fichier `commandes/ooops.md`.
- ▶ L'ajouter et le **commiter**, il appartient à la branche `master`
- ▶ Faire en sorte que `master` ne contienne plus ce commit

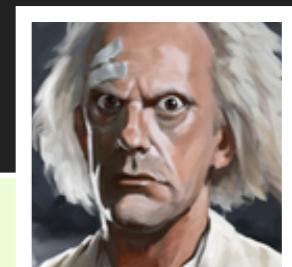
```
doc@labo$ cd ~/git/cours_git
doc@labo$ echo "oops" >> commandes/ooops.md
doc@labo$ git add commandes/ooops.md
doc@labo$ git commit -m "Je voulais pas commiter"
```



EXERCICE 2.2: MAUVAIS COMMENTAIRE

- ▶ Rajouter un fichier *commandes/reset.md* avec un contenu intéressant
- ▶ Le commiter avec un commentaire erroné
- ▶ Annuler ce commit en gardant les modifications et le refaire avec un bon commentaire

```
doc@labo$ echo "# reset" >> commandes/reset.md
doc@labo$ git add commandes/reset.md
doc@labo$ git commit -m "Je ne C pas aicrire"
```





EXERCICE 2.3: BRANCHE

- ▶ Sur **master**:

Créer le fichier

commandes_en/.gitkeep

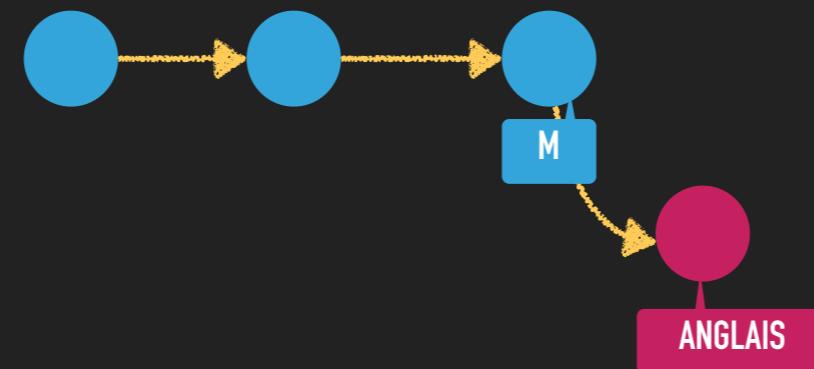
L'ajouter et le commiter

```
doc@labo$ mkdir commandes_en && \  
        touch commandes_en/.gitkeep  
doc@labo$ git add commandes_en && \  
        git commit -m "creation dossier"  
doc@labo$ git branch anglais  
doc@labo$ git checkout anglais  
Switched to a new branch 'anglais'  
doc@labo$ echo "# init" >> commandes_en/init.md  
doc@labo$ git add commandes_en/init.md  
doc@labo$ git commit -m "Added init.md"
```

- ▶ Créer une branche **anglais**

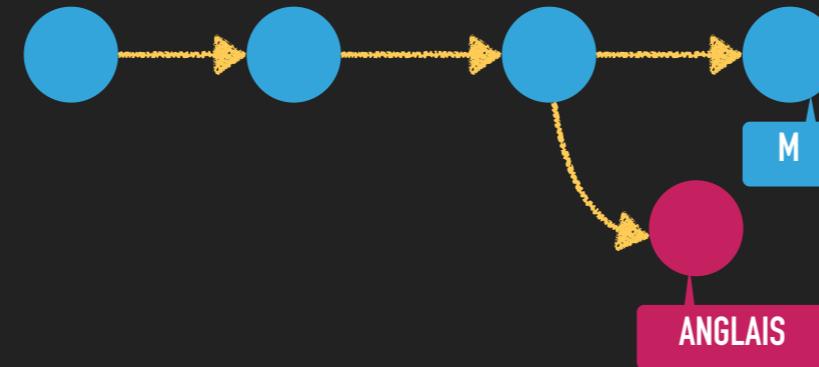
- ▶ Sur **anglais**, créer le fichier *commandes_en/init.md*

L'ajouter et le commiter



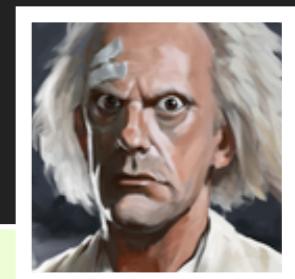
EXERCICE 2.4: BRANCHE 2

- ▶ Se repositionner sur la branche `master`
- ▶ Ajouter la description de la commande `checkout` dans `commandes/checkout.md`
- ▶ L'ajouter et le commiter



EXERCICE 2.5: MAUVAISE BRANCHE

- ▶ Se placer sur **master**
- ▶ Créer un fichier en anglais dans *commandes_en/reset.md* avec du contenu, l'ajouter et le commiter sur **master**
- ▶ Annuler ce commit sur **master tout en conservant les modifications** et le refaire sur la branche **anglais**



```
doc@labo$ echo "# I DO SPEAK ENGLISH" > commandes_en/reset.md
doc@labo$ git add commandes_en/reset.md
doc@labo$ git commit -m "commit sur la mauvaise branche -- mince alors"
```

RESET / BRANCH

- ▶ Créer une branche **feature_name**:

```
git branch feature_name
```

- ▶ Se positionner sur une branche **feature_name**:

```
git checkout feature_name
```

- ▶ Annuler le dernier commit en conservant les modifications:

```
git reset HEAD^
```

- ▶ Annuler le dernier commit en **PERDANT** les modifications:

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



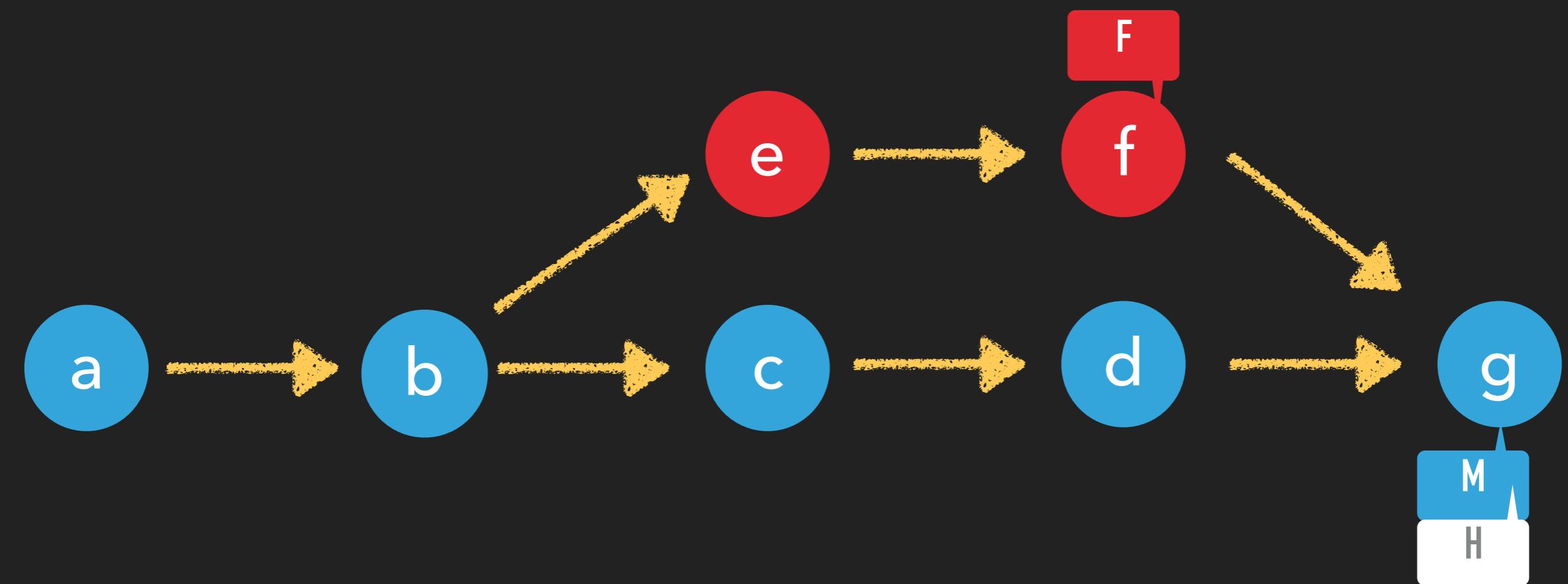
PERSONNE NE ME DIT QUE J'AI LES FOIES.

MERGE

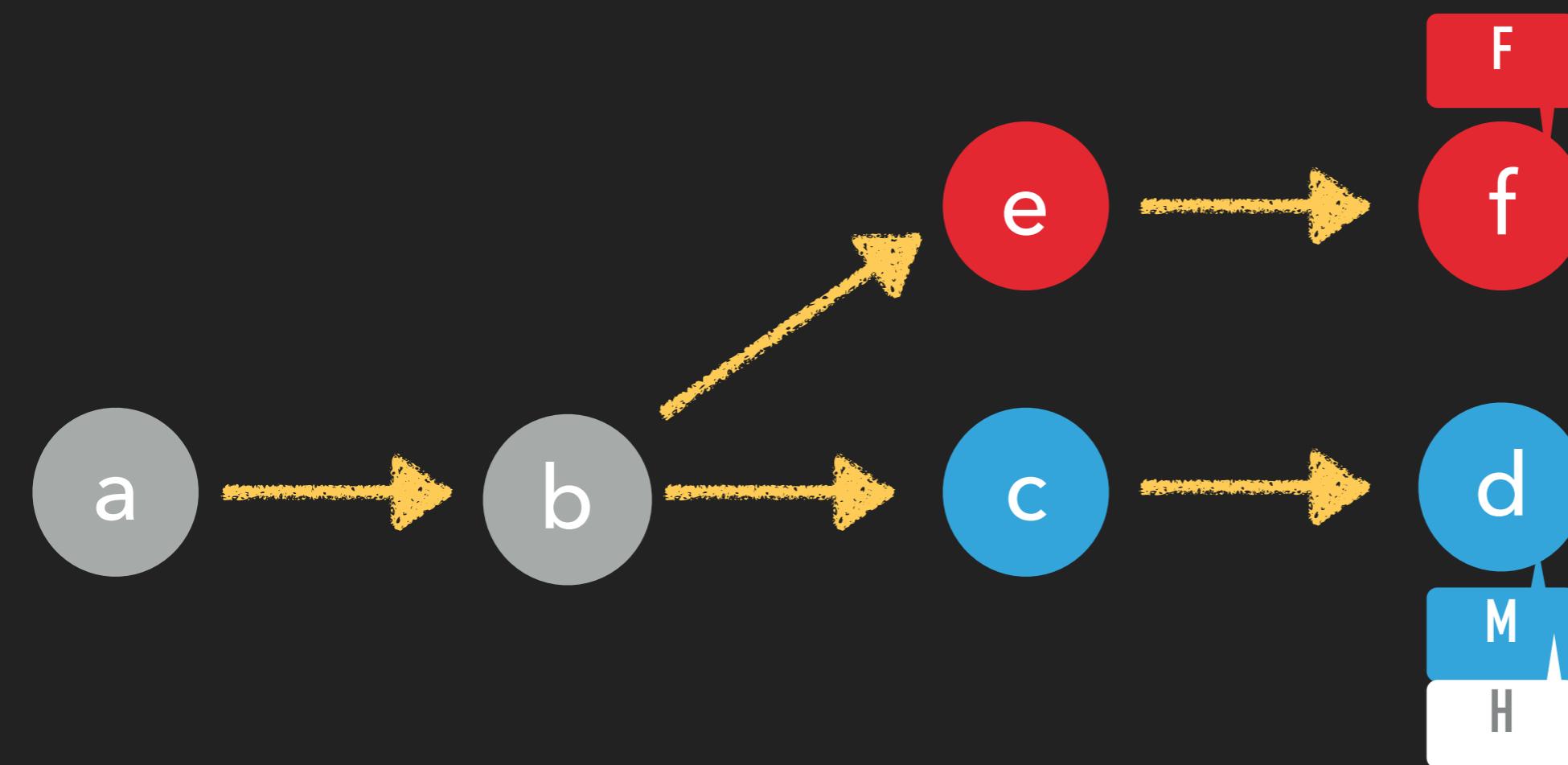
MERGE

- ▶ C'est la réunion de deux historiques
- ▶ Crée un commit qui a deux parents
- ▶ Peut créer des conflits

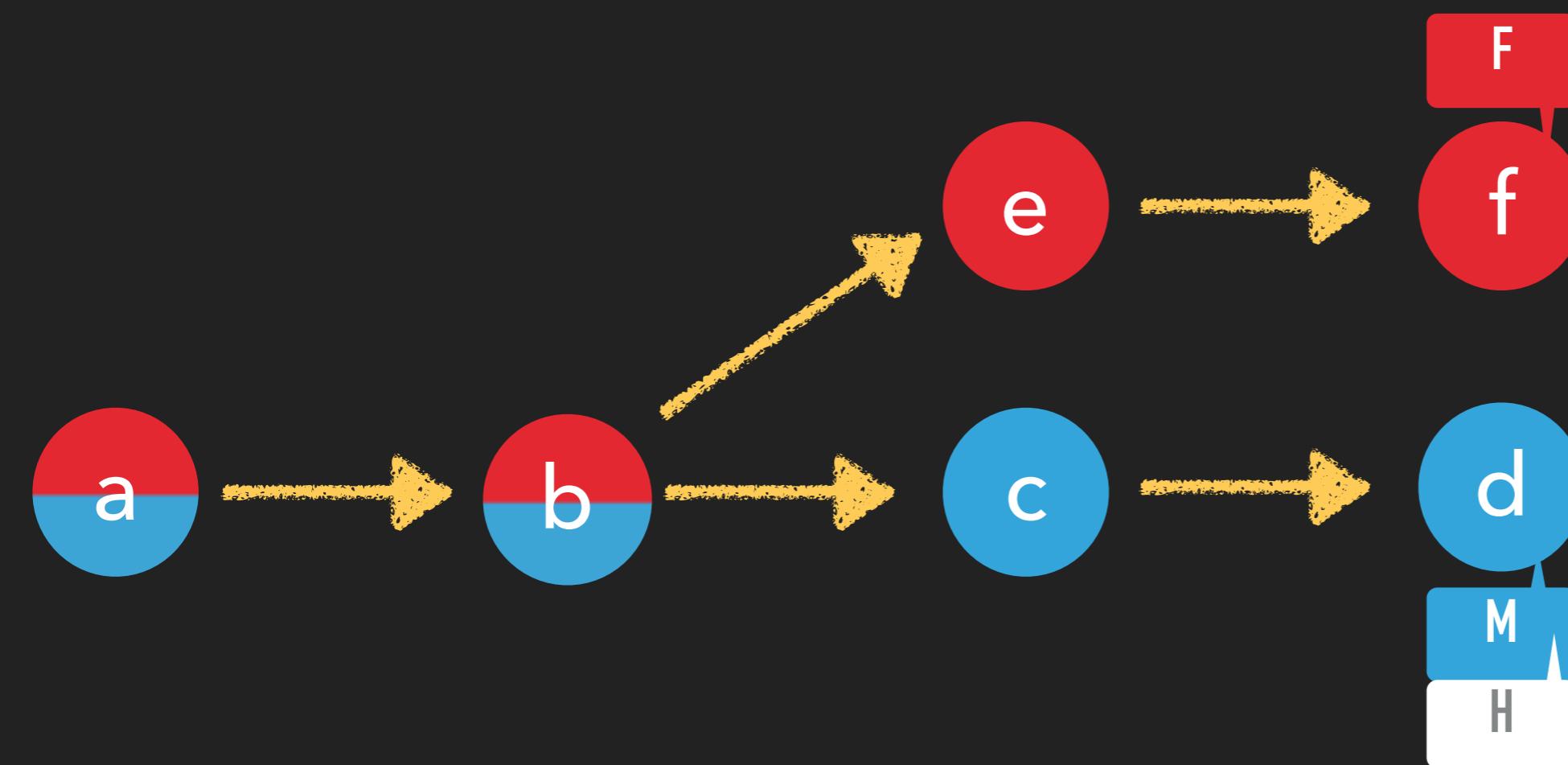
MERGE



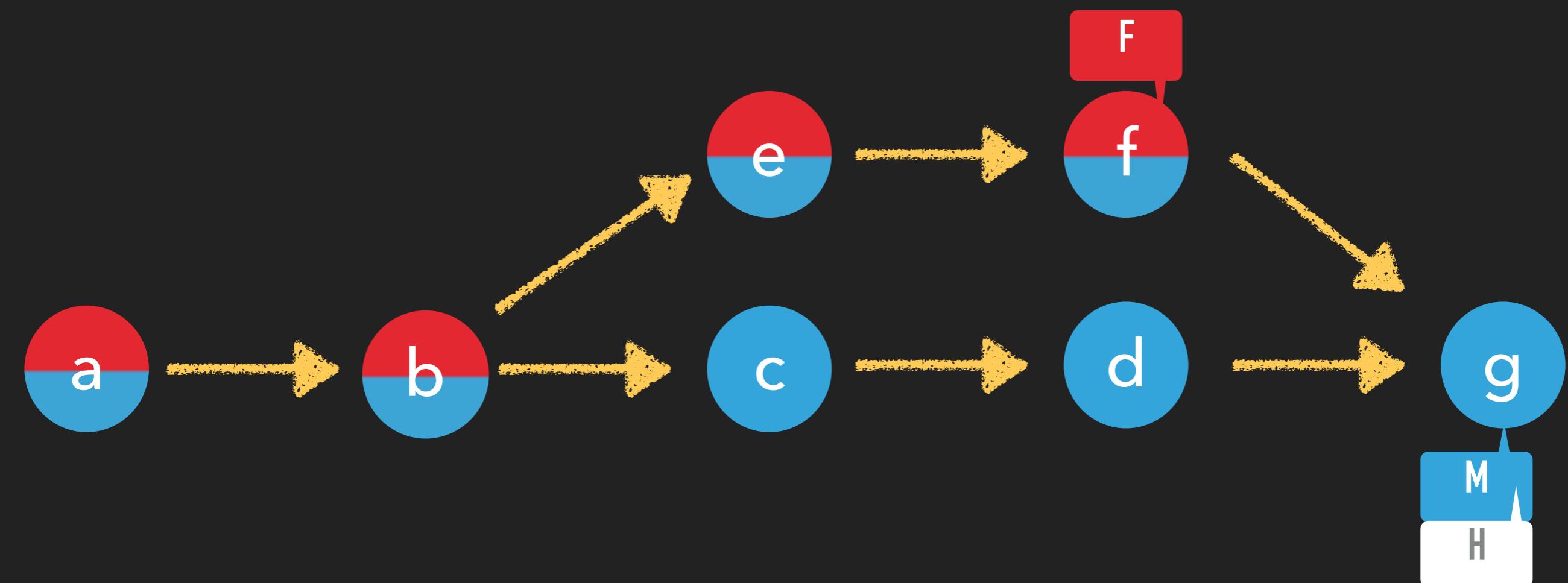
MERGE ET APPARTENANCE



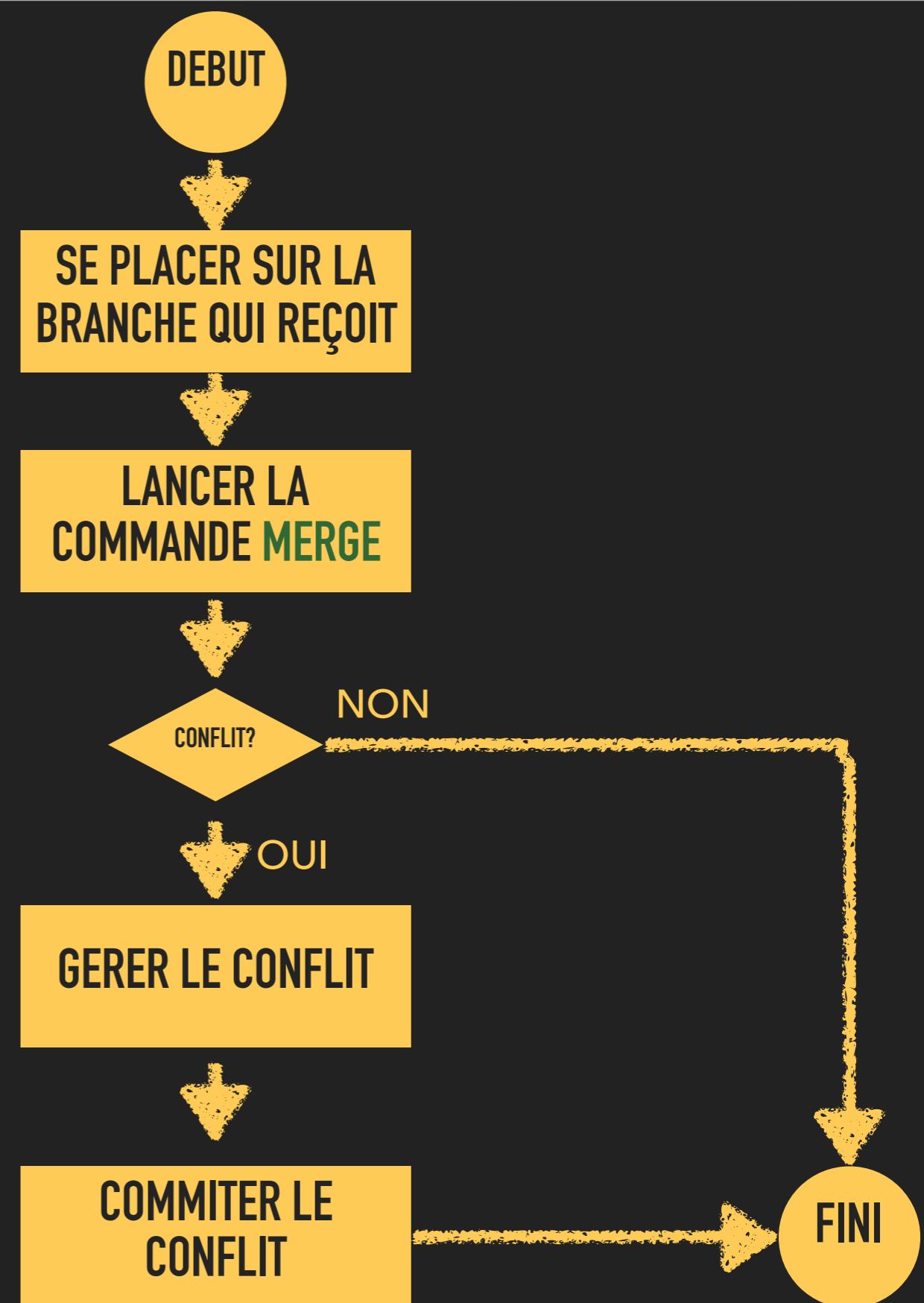
MERGE ET APPARTENANCE



MERGE ET APPARTENANCE

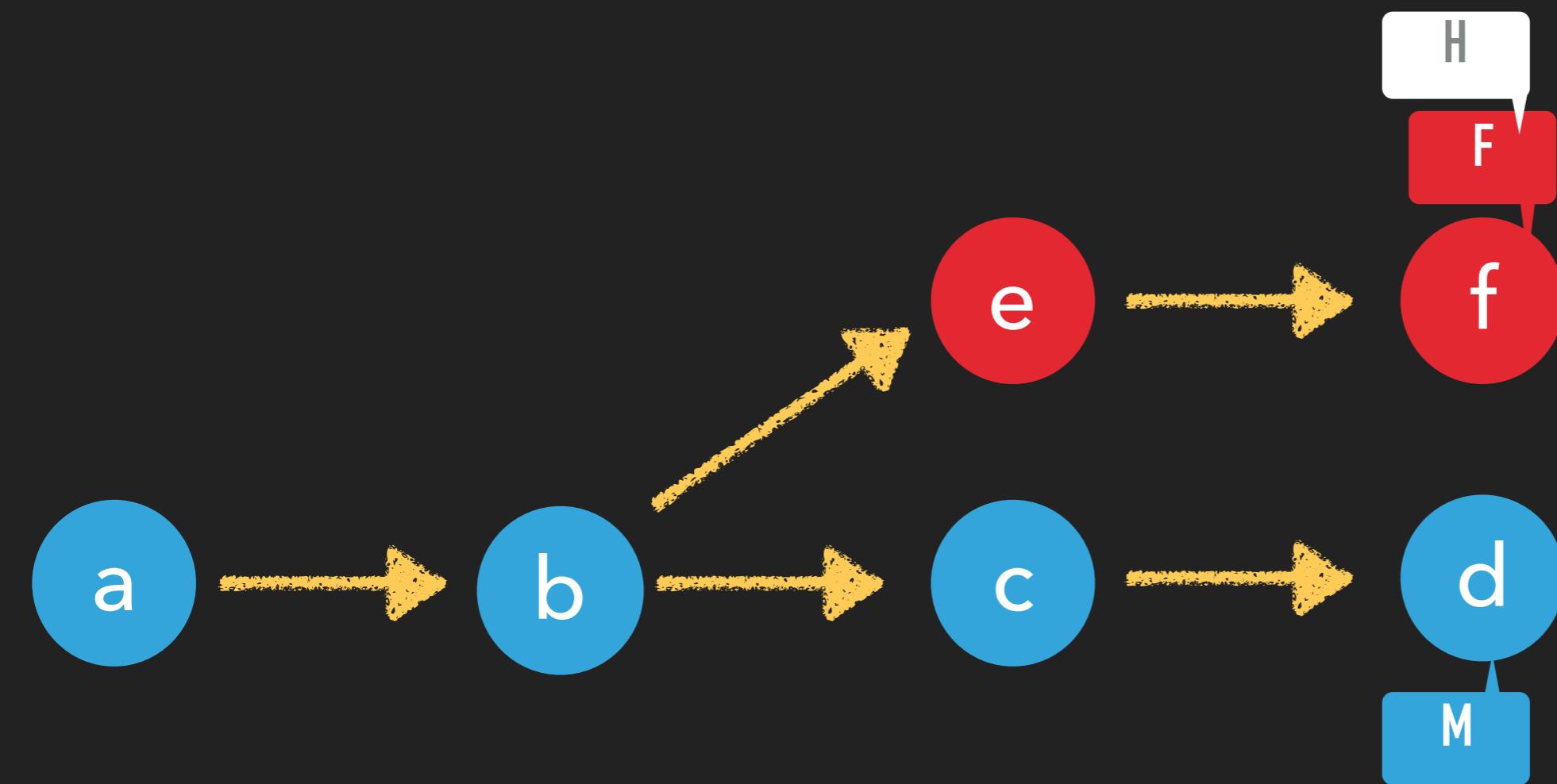
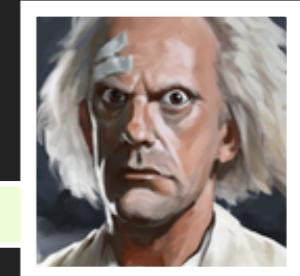


LES ETAPES DU MERGE

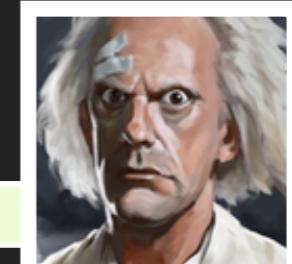


git merge branch -m “message”

```
doc@labo$ git checkout master
```

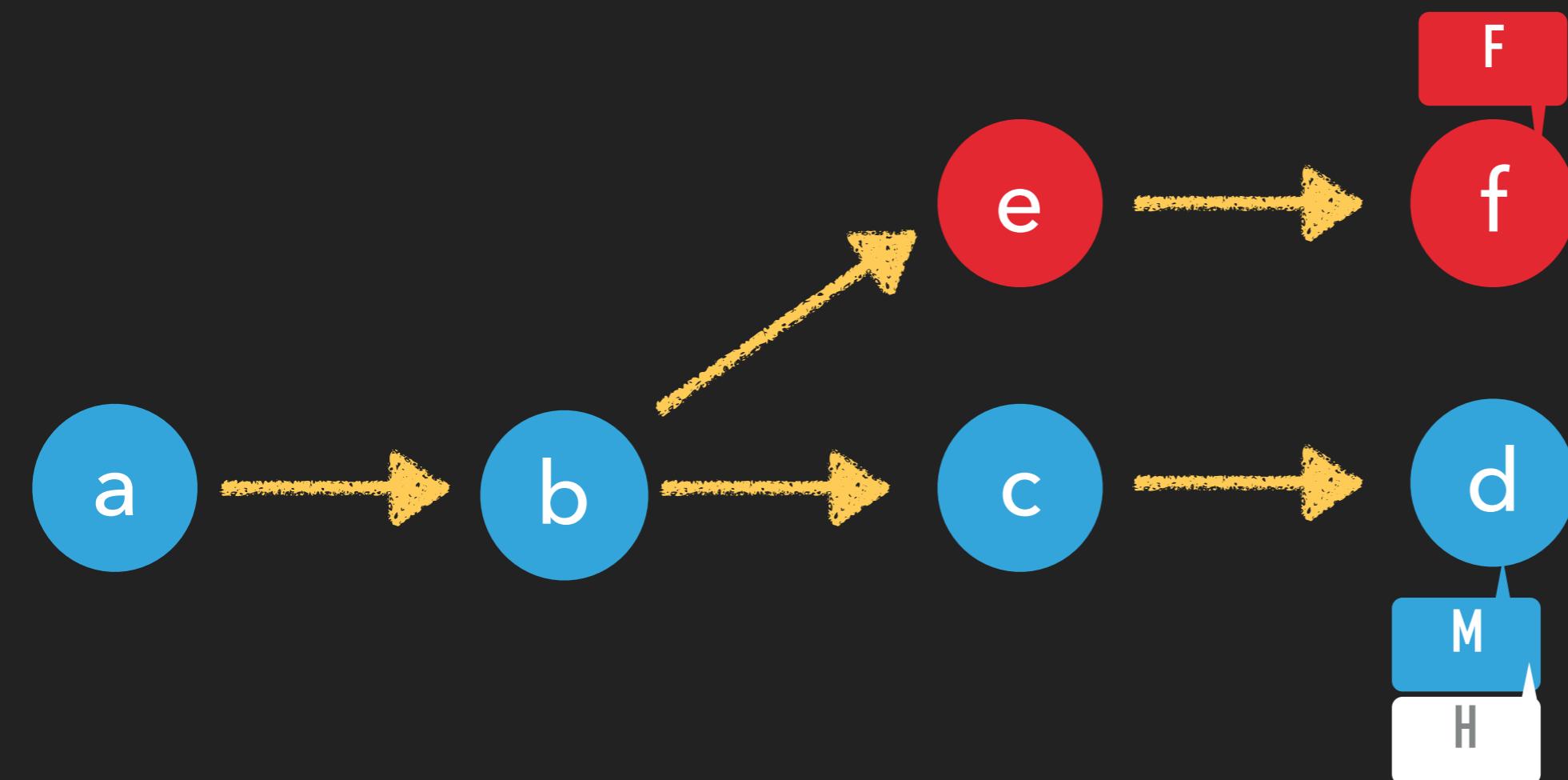


git merge branch -m “message”

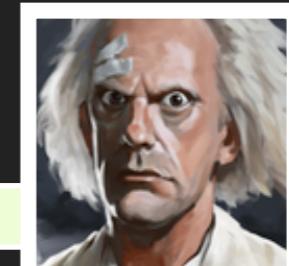


```
doc@labo$ git checkout master
```

```
doc@labo$ git merge feature -m "merge de la feature dans master"
```

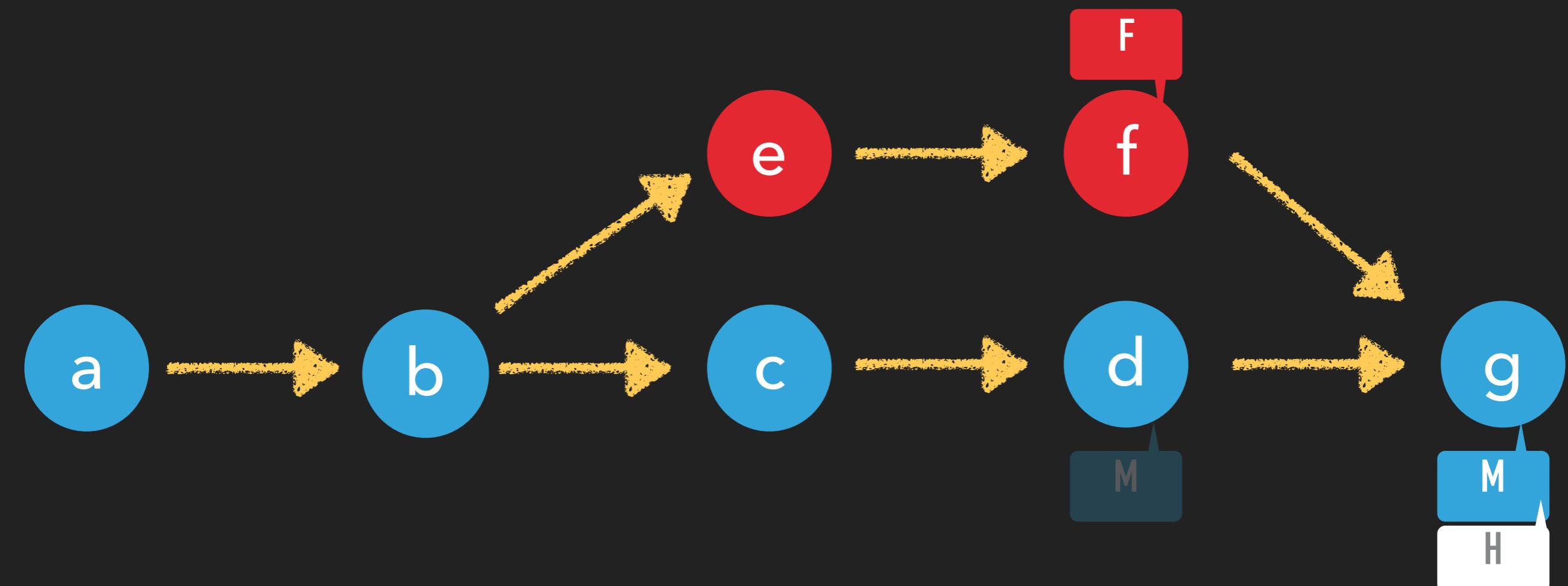


git merge branch -m “message”



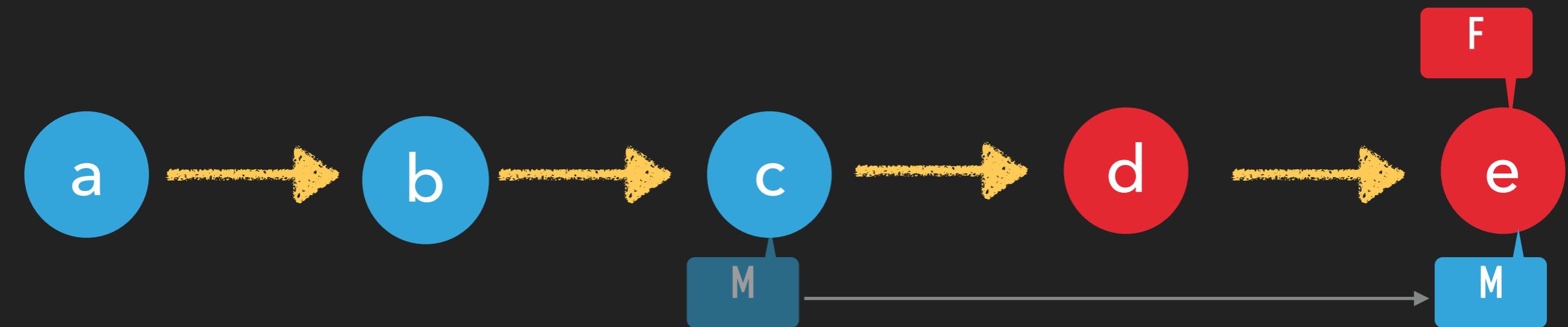
```
doc@labo$ git checkout master
```

```
doc@labo$ git merge feature -m "merge de la feature dans master"
Merge made by the 'recursive' strategy.
 commandes/ideefolle.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 commandes/ideefolle.txt
```



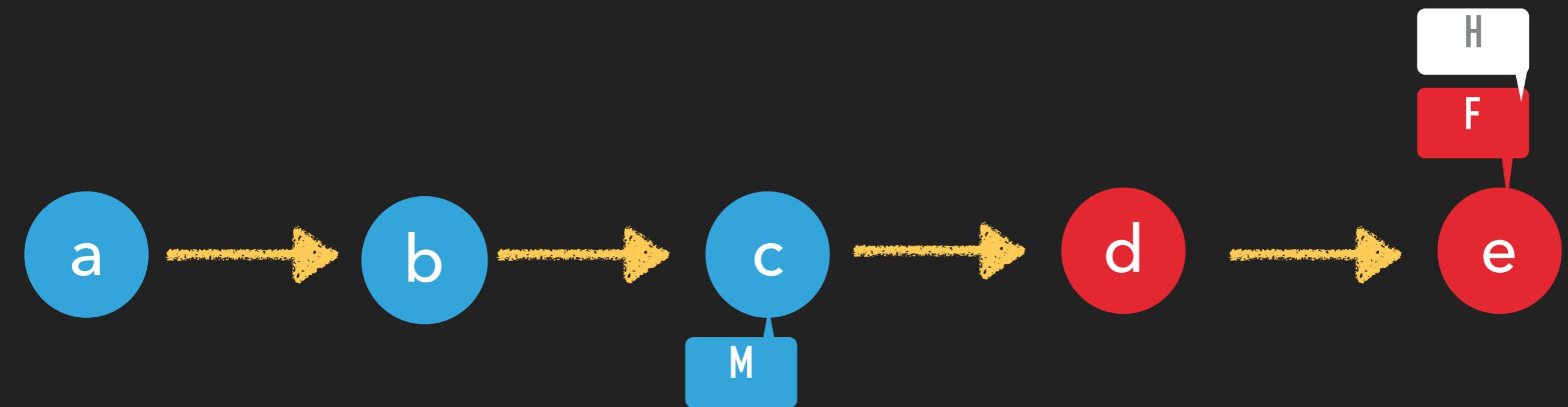
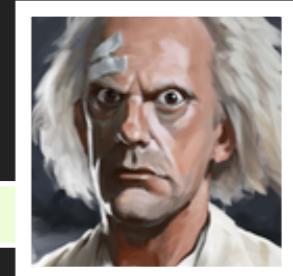
LE MERGE FAST FORWARD

- ▶ Se produit quand la branche à merger est une descendante directe
- ▶ La branche courante avance à la branche à merger
- ▶ On appelle ce mécanisme le *fast-forward*



git merge branch -m “message”

```
doc@labo$ git checkout master
```

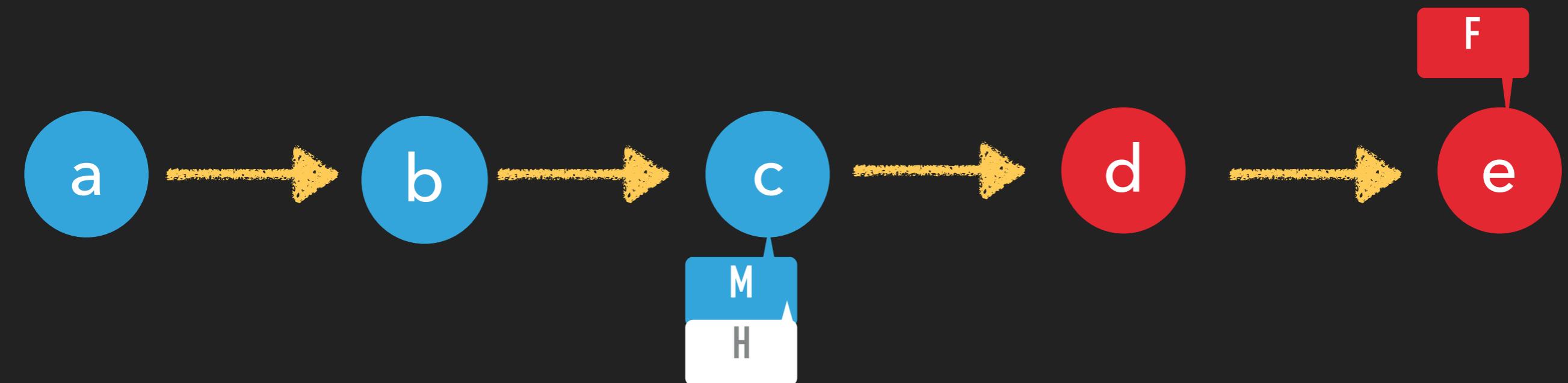


git merge branch -m “message”

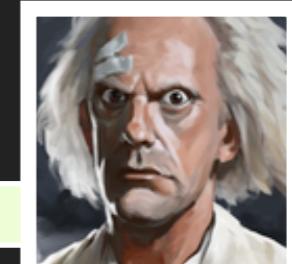


```
doc@labo$ git checkout master
```

```
doc@labo$ git merge feature
```

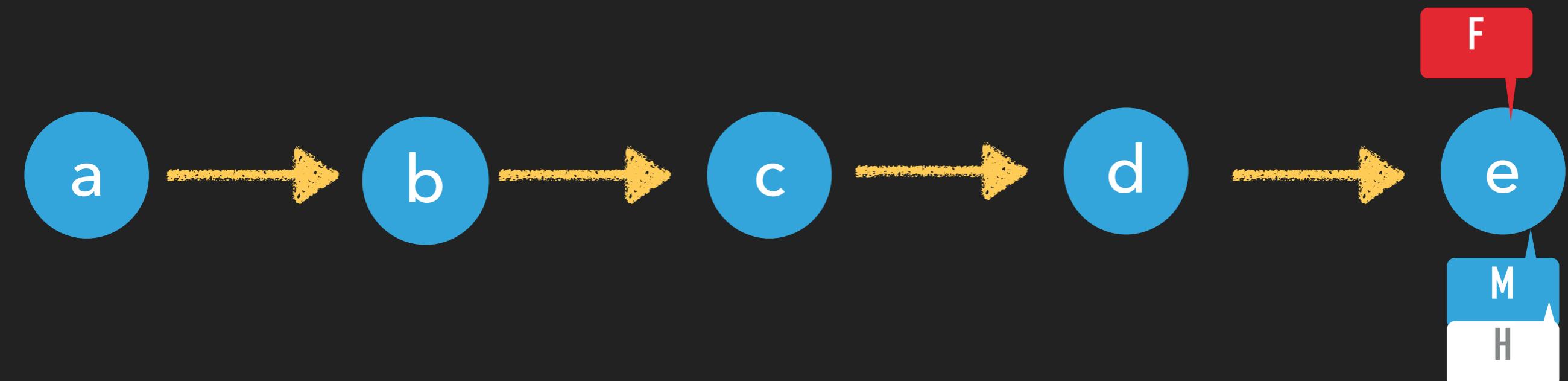


git merge branch -m “message”



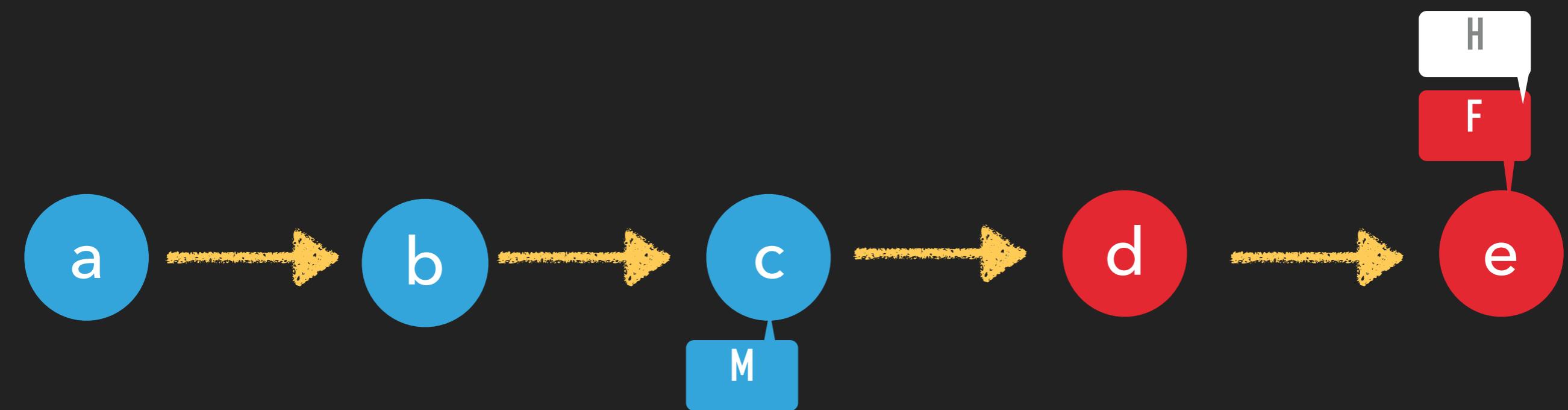
```
doc@labo$ git checkout master
```

```
doc@labo$ git merge -m "haha" feature
Updating 7ca71b9..759a26a
Fast-forward (no commit created; -m option ignored)
 commandes/ideefolle.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 commandes/ideefolle.txt
```



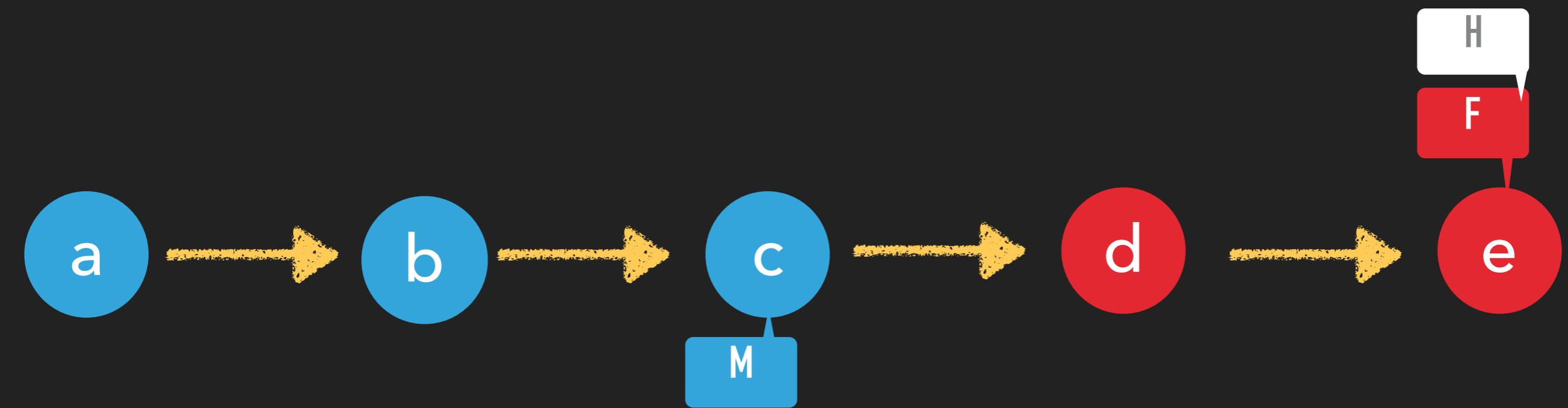
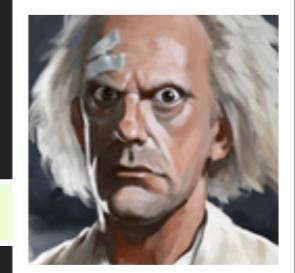
git merge --no-ff branch -m “pas de fast forward”

- ▶ --no-ff: no fast forward
- ▶ Crée un commit de merge, vide de patch en cas de descendance directe



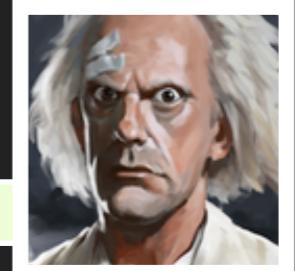
git merge --no-ff branch -m “pas de fast forward”

```
doc@labo$ git checkout master
```

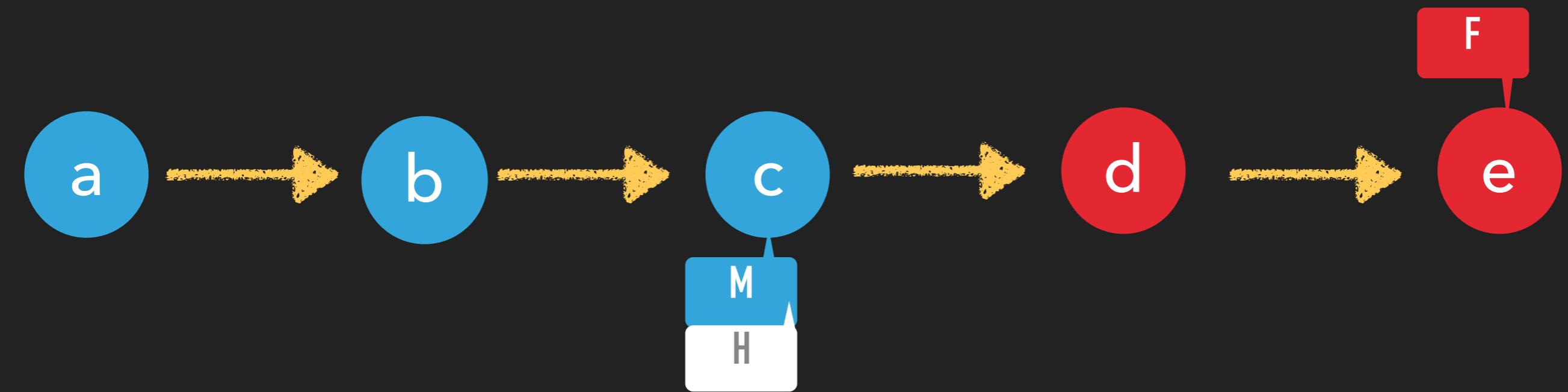


```
git merge --no-ff branch -m "pas de fast forward"
```

```
doc@labo$ git checkout master
```

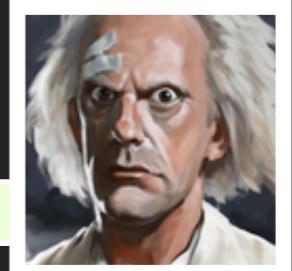


```
doc@labo$ git merge --no-ff feature -m "pas de fast forward"
Merge made by the 'recursive' strategy.
 commandes/ideefolle.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 commandes/ideefolle.txt
```

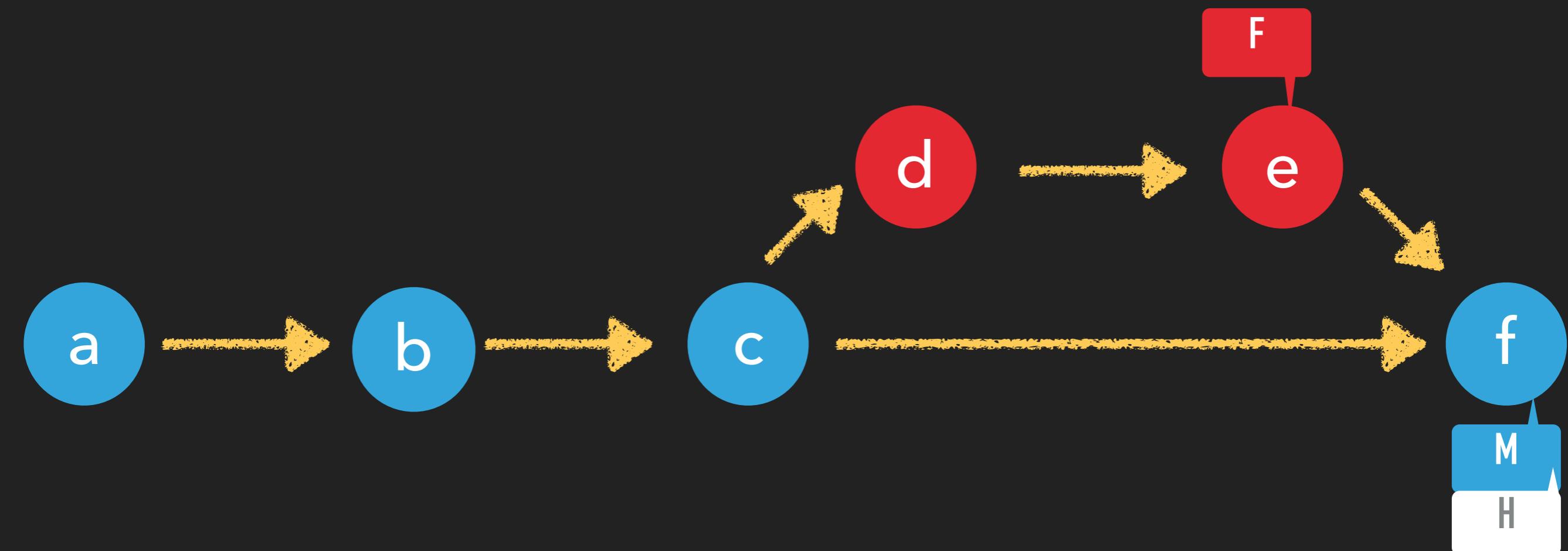


git merge --no-ff branch -m "pas de fast forward"

```
doc@labo$ git checkout master
```



```
doc@labo$ git merge --no-ff feature -m "pas de fast forward"
Merge made by the 'recursive' strategy.
 commandes/ideefolle.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 commandes/ideefolle.txt
```





TU AS LES FOIES MAC FLY

CONFLITS

LES SITUATIONS DE CONFLIT

- ▶ Se produit lors d'un **merge**
- ▶ Si deux branches éditent le même fichier de *manière concurrente*
- ▶ Un conflit est levé, le merge (ou le rebase) sont mis en pause
- ▶ Il faut résoudre le conflit avant de continuer

LES ÉDITIONS CONCURRENTES

- ▶ Si une branche modifie un fichier que l'autre a effacé
- ▶ Si une branche modifie la même ligne que l'autre
- ▶ ...
- ▶ Modifier un fichier renommé ou déplacé ne provoque pas de conflit

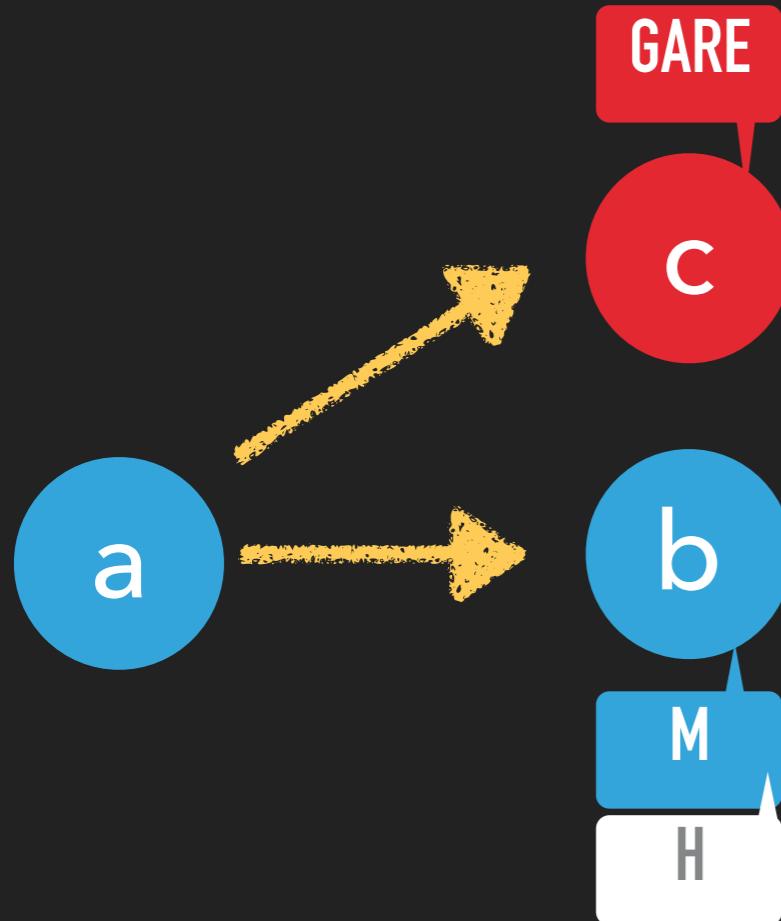
RESOLUTION D'UN CONFLIT



rendezvous.md

```
# rendez vous  
- Lieu: gare  
- Heure: 15:00
```

```
doc@labo$ git checkout master  
doc@labo$ git merge gare -m "gare au merge"
```



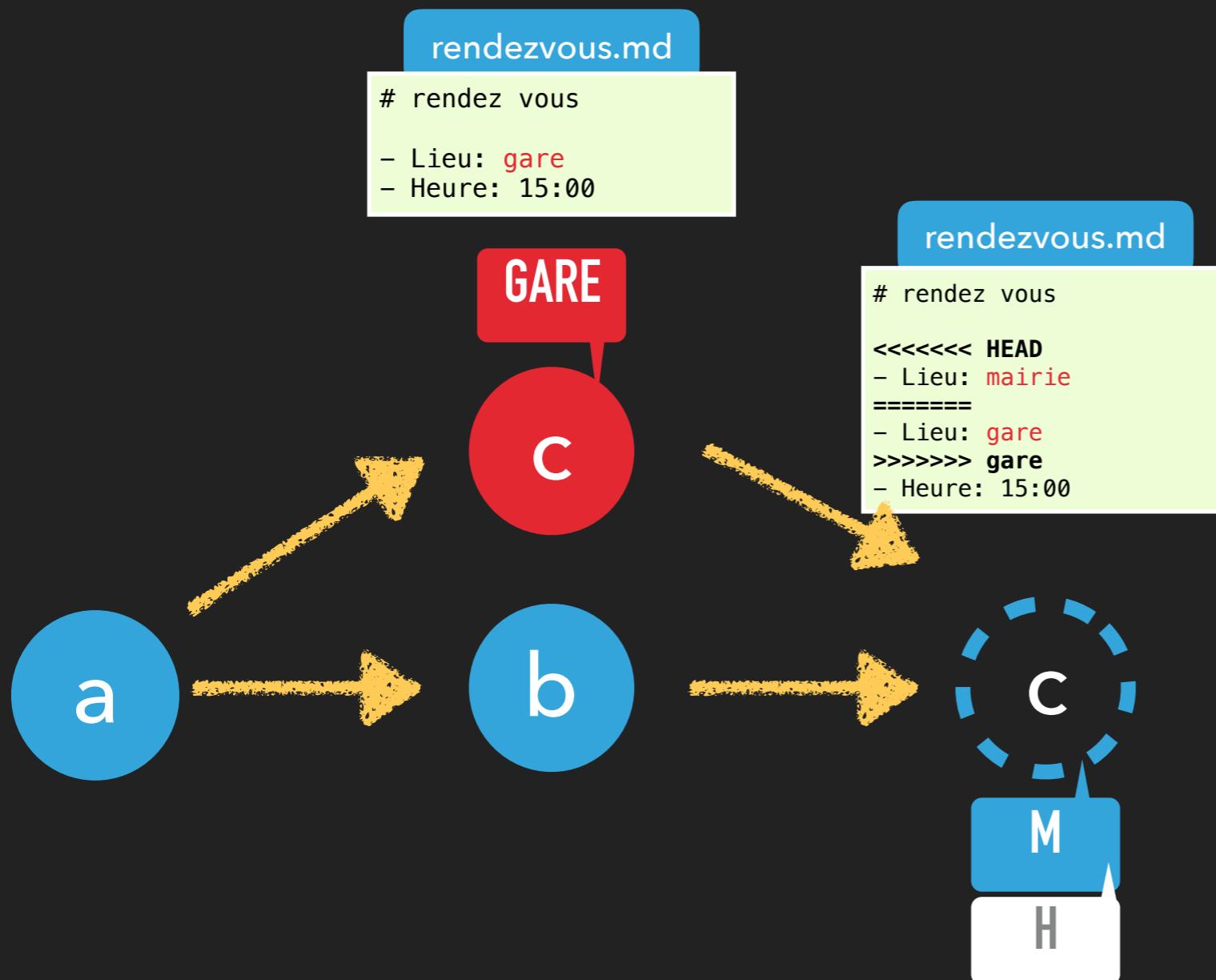
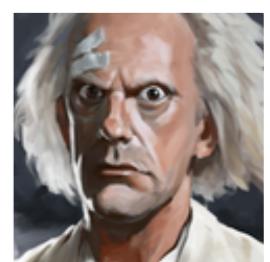
rendezvous.md

```
# rendez vous  
- Lieu: centre ville  
- Heure: 15:00
```

rendezvous.md

```
# rendez vous  
- Lieu: mairie  
- Heure: 15:00
```

RESOLUTION D'UN CONFLIT



rendezvous.md

```
# rendez vous
- Lieu: gare
- Heure: 15:00
```

rendezvous.md

```
# rendez vous
<<<<< HEAD
- Lieu: mairie
=====
- Lieu: gare
>>>>> gare
- Heure: 15:00
```

```
doc@labo$ git checkout master
doc@labo$ git merge gare -m "gare au merge"
Auto-merging rendezvous.md
CONFLICT (content): Merge conflict in rendezvous.md
Automatic merge failed; fix conflicts and then
commit the result.
```

```
doc@labo$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)
```

both modified: rendezvous.md

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

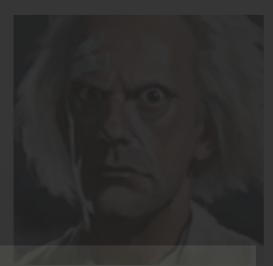
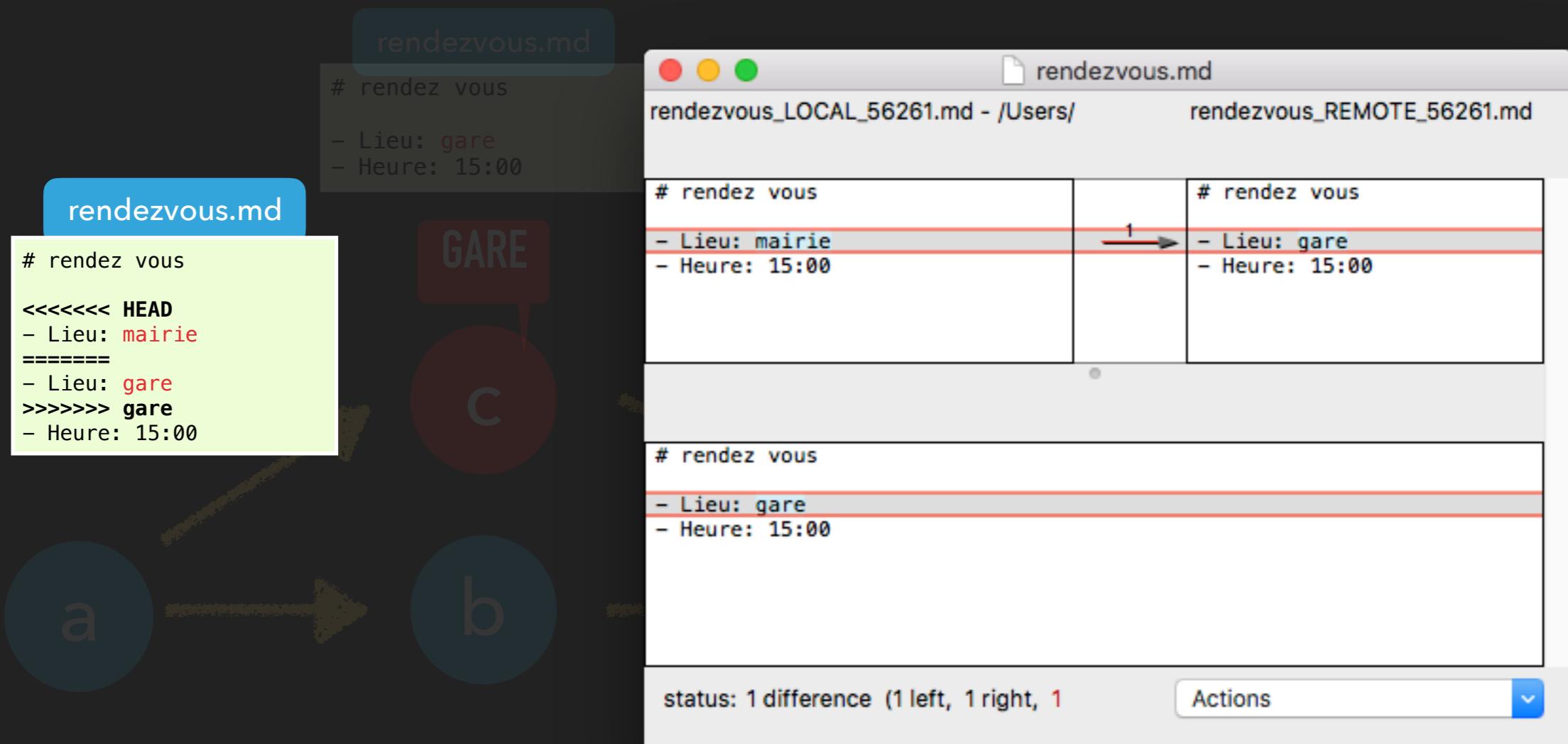
rendezvous.md

```
# rendez vous
- Lieu: centre ville
- Heure: 15:00
```

rendezvous.md

```
# rendez vous
- Lieu: mairie
- Heure: 15:00
```

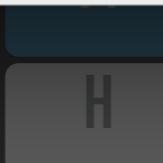
RESOLUTION D'UN CONFLIT



"u merge"
in `rendezvous.md`
its and then

t")
resolution)

t add" and/or



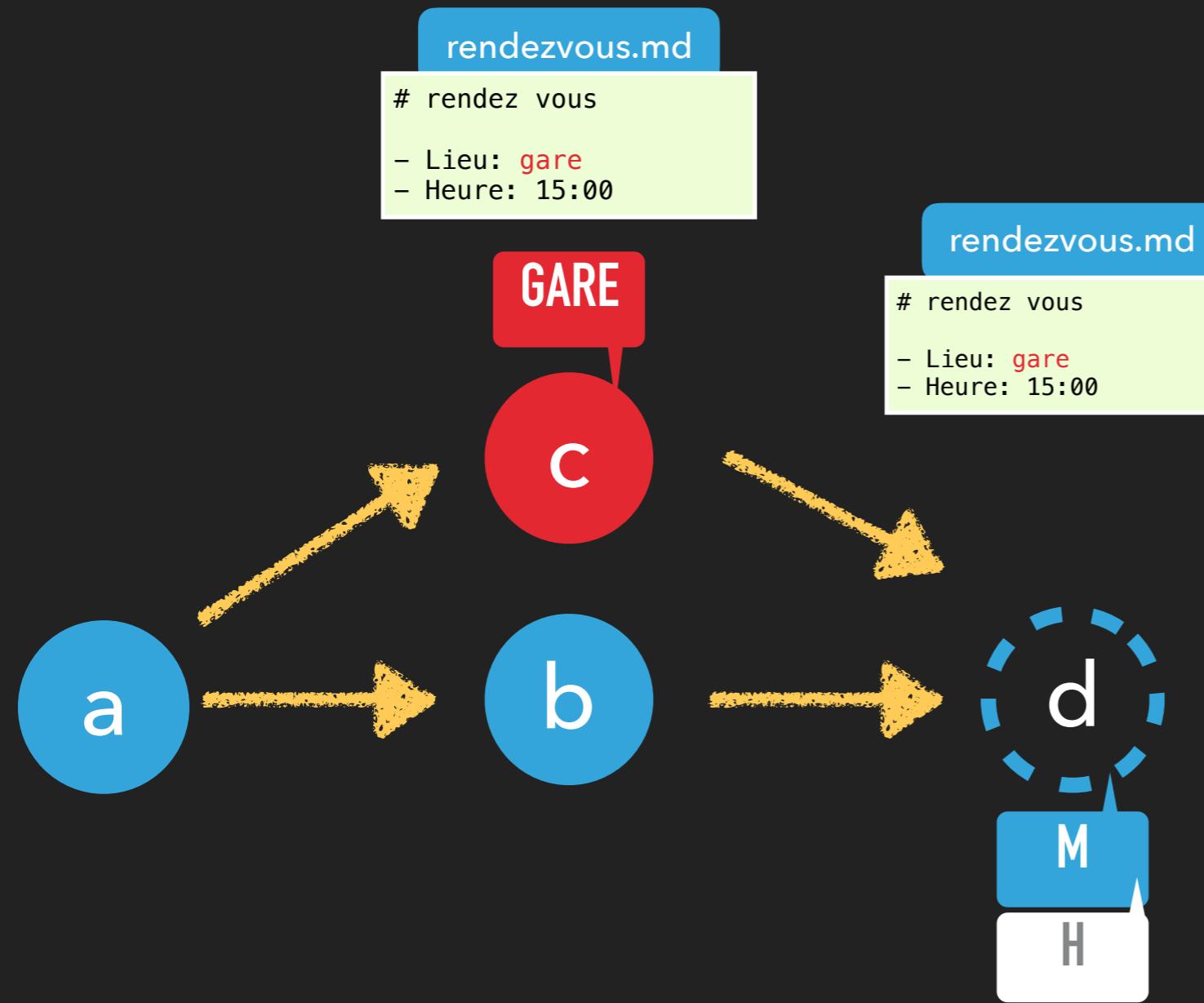
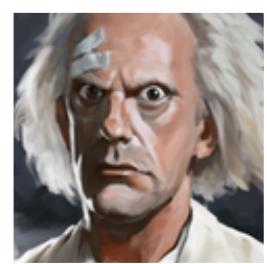
rendezvous.md

```
# rendez vous
- Lieu: centre ville
- Heure: 15:00
```

rendezvous.md

```
# rendez vous
- Lieu: mairie
- Heure: 15:00
```

RESOLUTION D'UN CONFLIT



```

doc@labo$ git checkout master
doc@labo$ git merge gare -m "gare au merge"
Auto-merging rendezvous.md
CONFLICT (content): Merge conflict in rendezvous.md
Automatic merge failed; fix conflicts and then
commit the result.

```

```

doc@labo$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

```

both modified: rendezvous.md

```

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

```

```

doc@labo$ # edit rendezvous.md
doc@labo$ git add rendezvous.md
doc@labo$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

```

Changes to be committed:

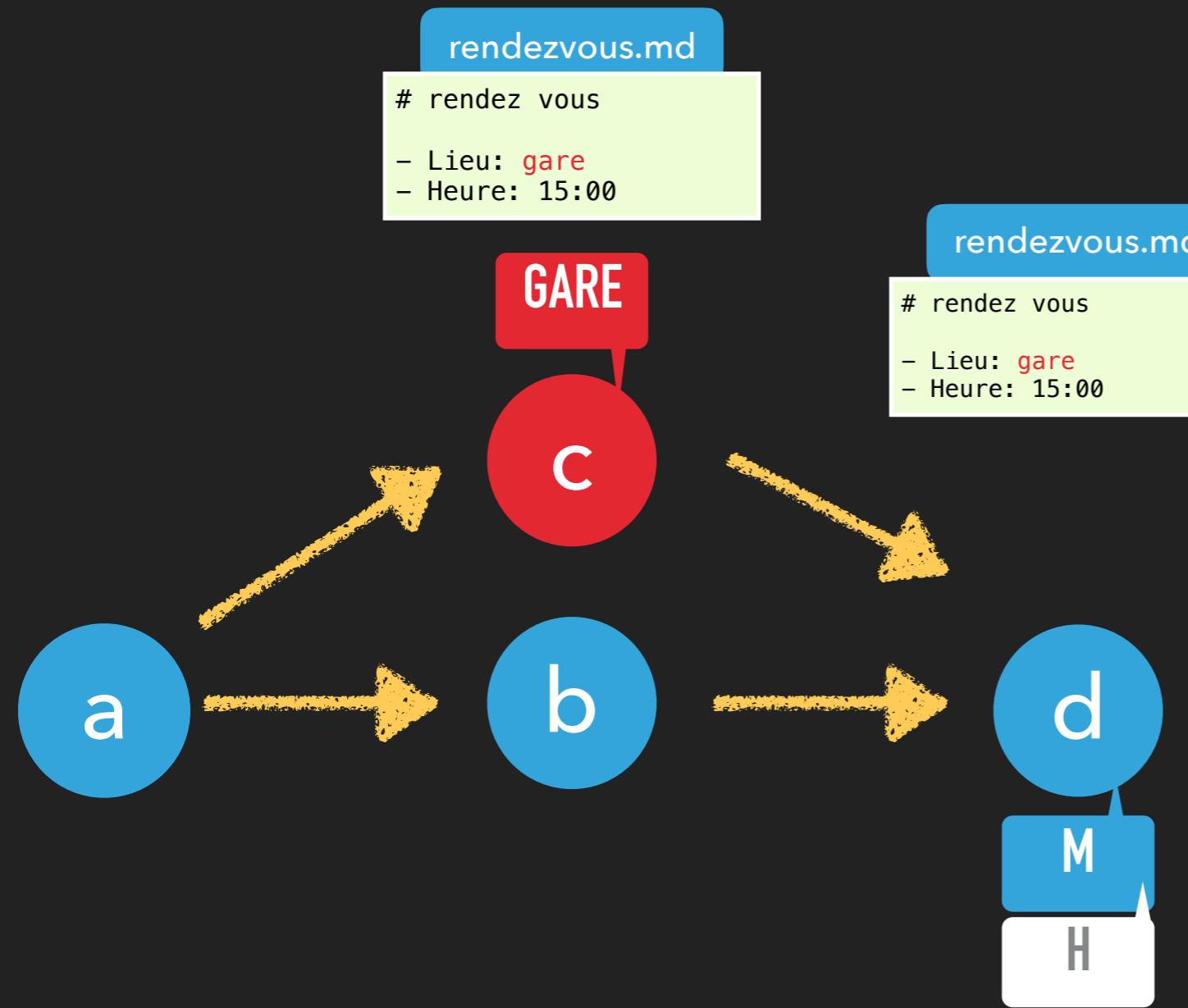
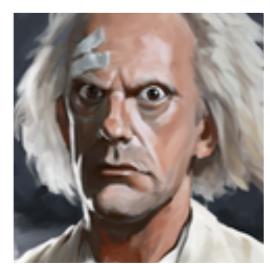
modified: rendezvous.md

```

doc@labo$ git commit -m "gare au merge"
[master 04ac40f] gare au merge

```

RESOLUTION D'UN CONFLIT



```

doc@labo$ git checkout master
doc@labo$ git merge gare -m "gare au merge"
Auto-merging rendezvous.md
CONFLICT (content): Merge conflict in rendezvous.md
Automatic merge failed; fix conflicts and then
commit the result.

```

```

doc@labo$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:  rendezvous.md

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

```

```

doc@labo$ # edit rendezvous.md
doc@labo$ git add rendezvous.md
doc@labo$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

```

Changes to be committed:

```

modified:  rendezvous.md

```

```

doc@labo$ git commit -m "gare au merge"
[master 04ac40f] gare au merge

```

rendezvous.md

```

# rendez vous
- Lieu: centre ville
- Heure: 15:00

```

rendezvous.md

```

# rendez vous
- Lieu: mairie
- Heure: 15:00

```

MERGE

- ▶ Merger / réintégrer une branche `feature_name`:

```
git merge --no-ff feature_name -m  
“commentaire”
```

- ▶ Gérer un conflit sur `fichier.txt`:

```
git add fichier.txt  
git commit -m “commentaire de merge”
```

- ▶ Abandonner un merge en cours:

```
git merge --abort
```

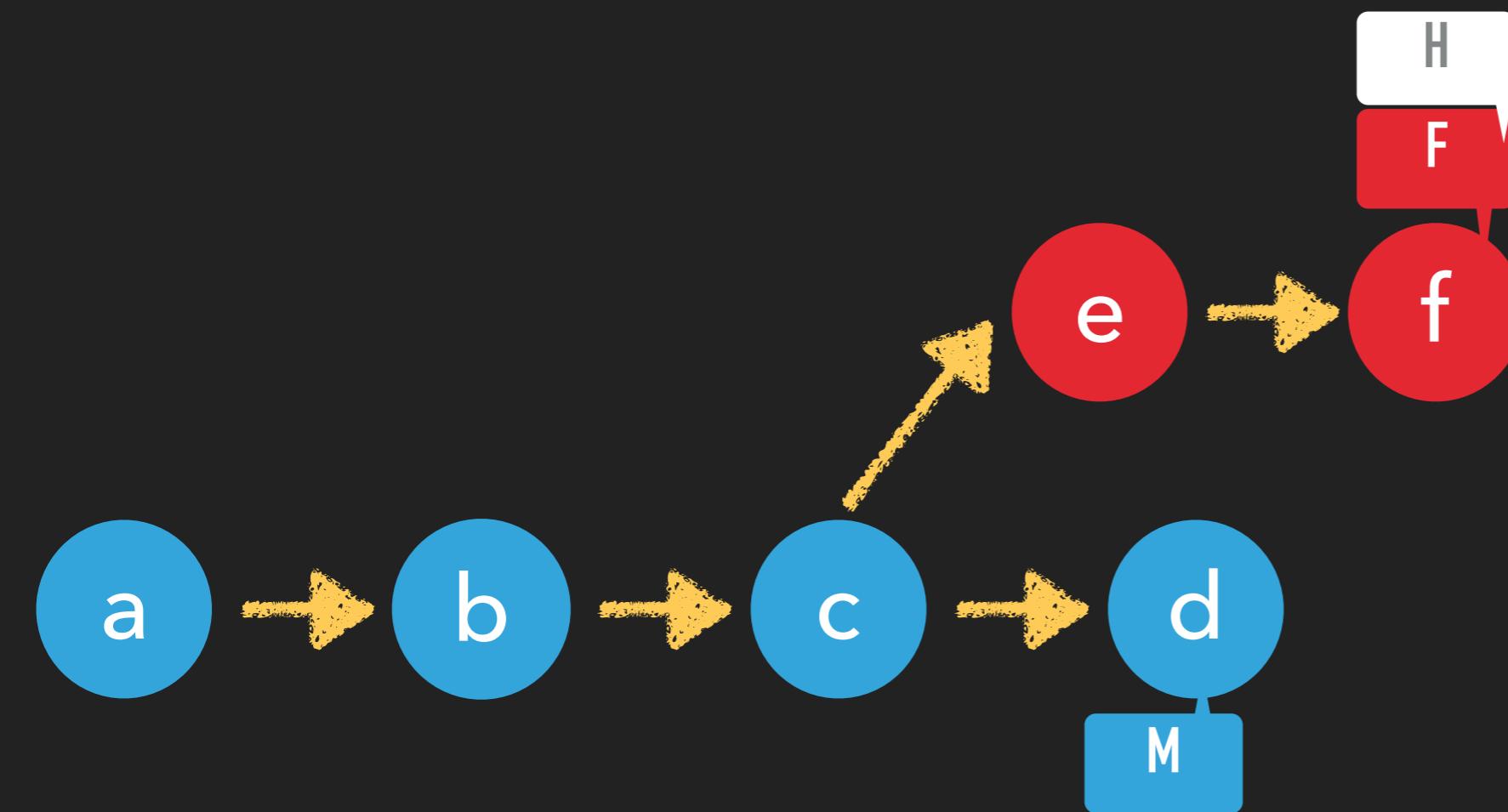


TU TE TIRES AILLEURS, TRIPLE BUSE !

REBASE

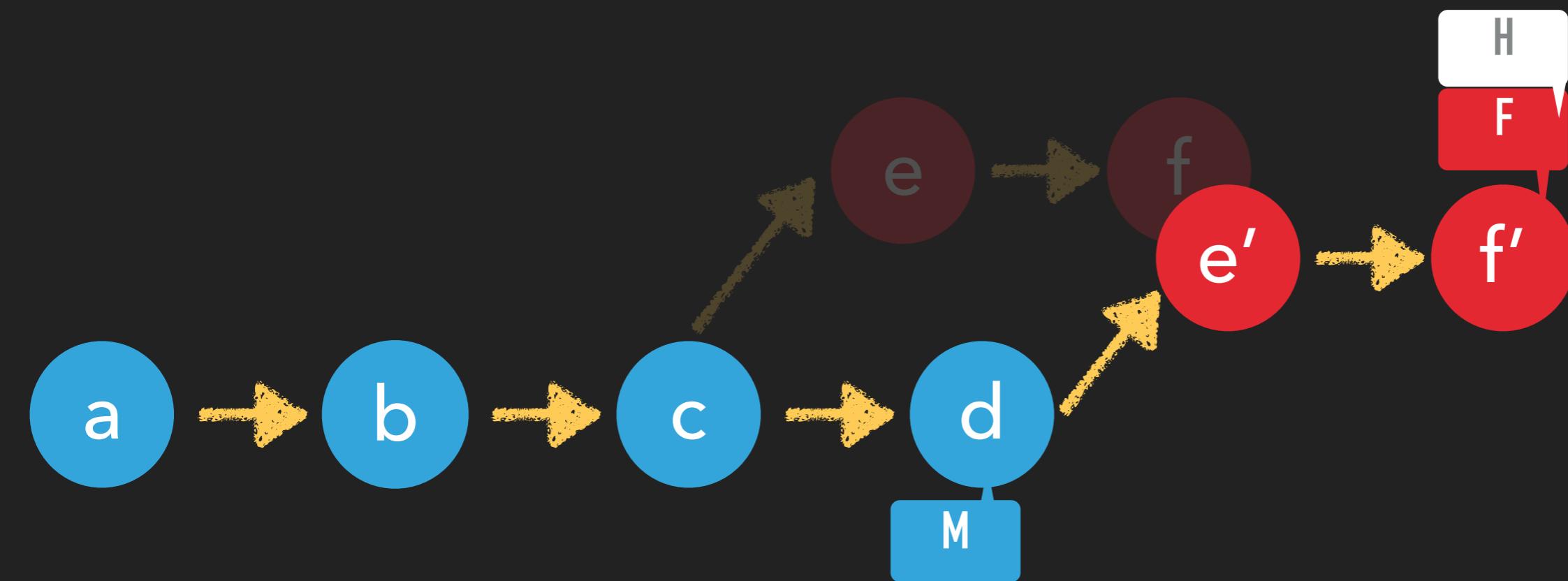
LE REBASE PERMET DE

- ▶ Déplacer la base de la branche, ou rebaser la branche.
- ▶ Utile pour récupérer un fix dans sa branche sans merger

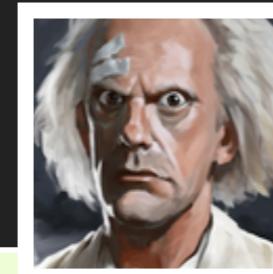


LE REBASE PERMET DE

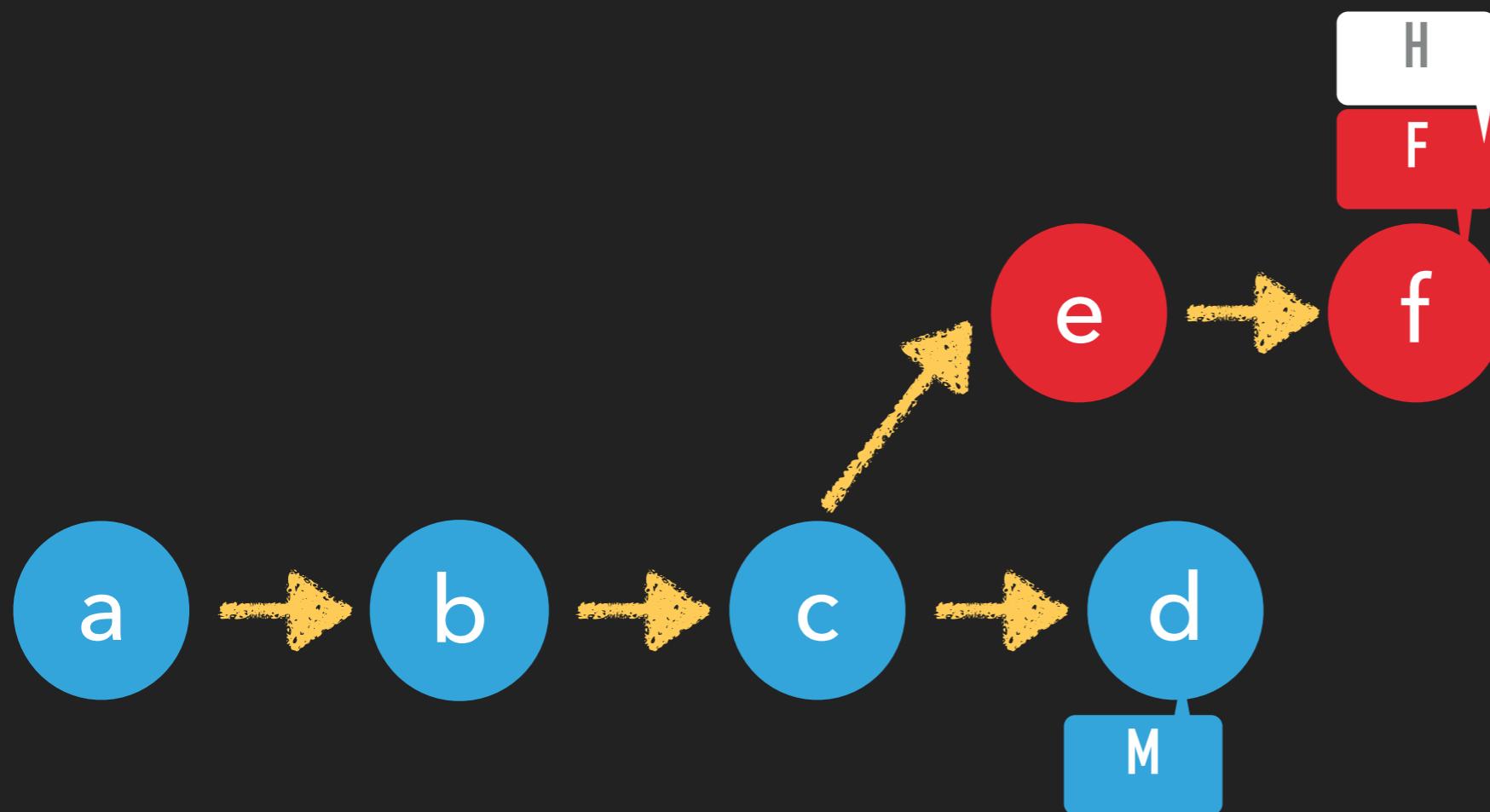
- ▶ Déplacer la base de la branche, ou rebaser la branche.



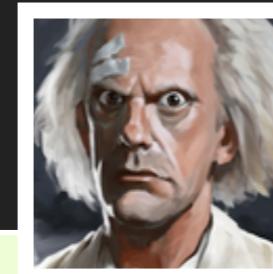
git rebase master



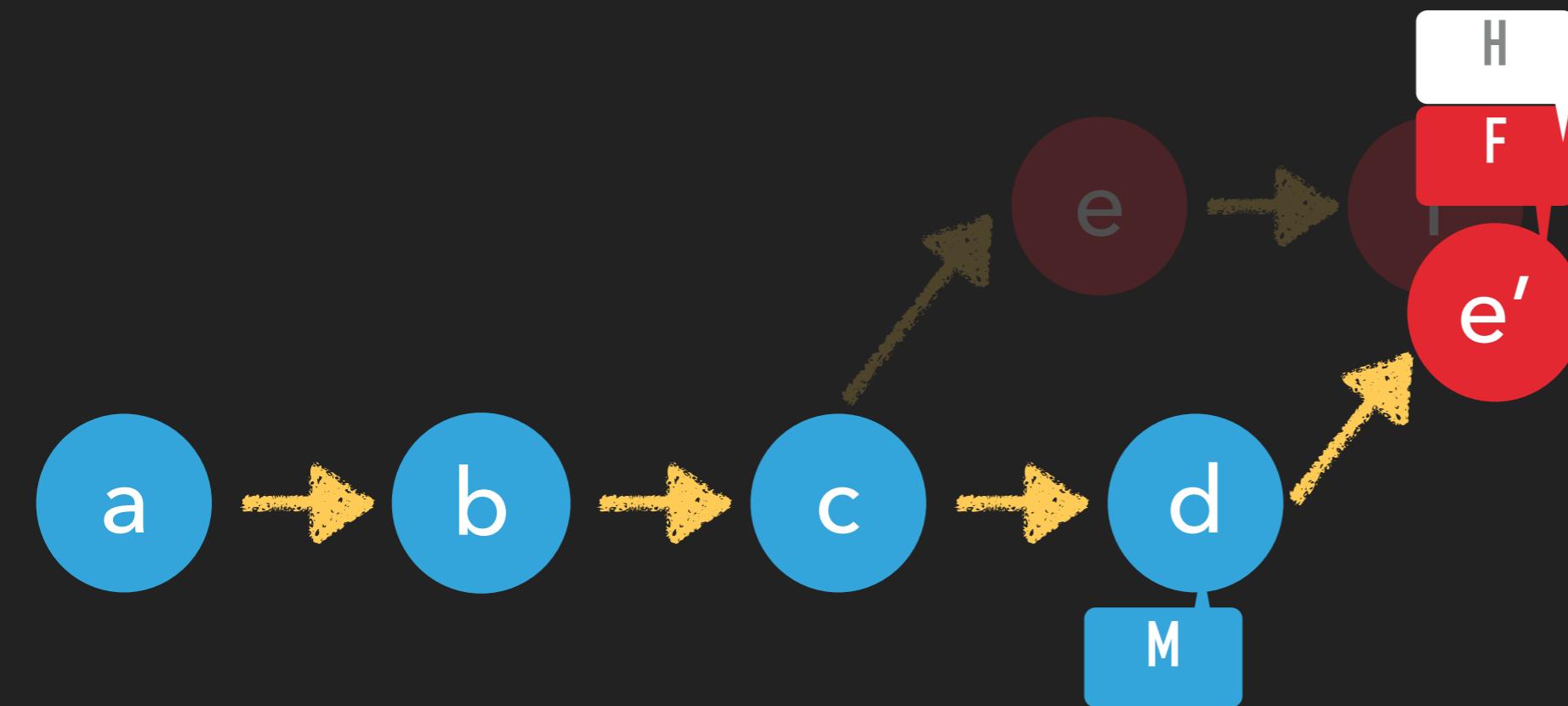
```
doc@labo$ git checkout feature
doc@labo$ git rebase master
First, rewinding head to replay your work on top of it...
```



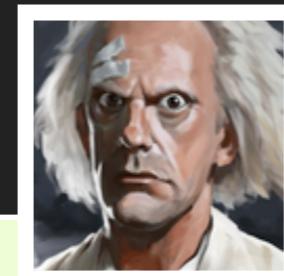
git rebase master



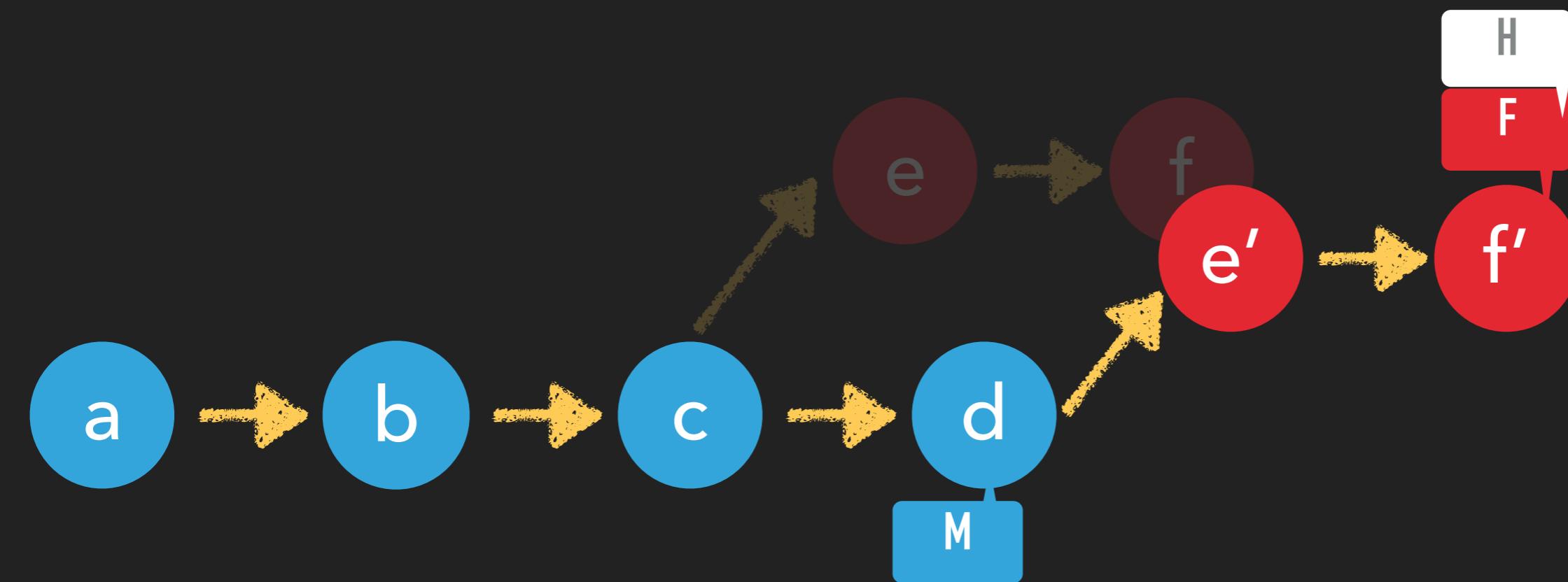
```
doc@labo$ git checkout feature
doc@labo$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: un commit a rebaser (e -> e')
```



git rebase master



```
doc@labo$ git checkout feature
doc@labo$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: un commit a rebaser (e -> e')
Applying: un autre commit a rebaser (f -> f')
```



git rebase -i master

- ▶ rebase interactif
- ▶ permet de configurer le replay
 - ▶ changer l'ordre des commit
 - ▶ édition de commentaire
 - ▶ merge de commit
 - ▶ édition des fichiers
- ▶ Utilisé surtout pour changer les commentaires

```
pick 52c1584 un commit a rebaser
pick cfbcddad un autre commit a rebaser

# Rebase 1cff4c8..cfbcddad onto 1cff4c8 (2 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
```



```
reword 52c1584 un commit a rebaser
pick cfbcddad un autre commit a rebaser

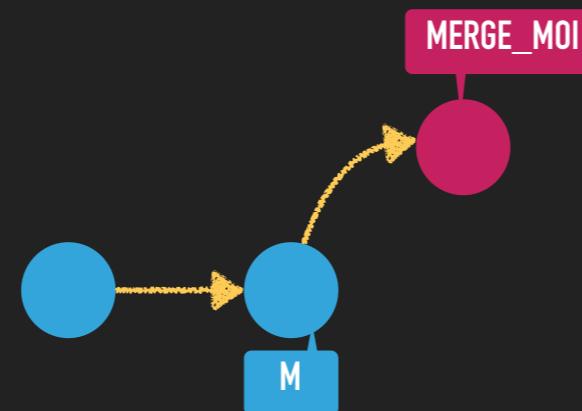
# Rebase 1cff4c8..cfbcddad onto 1cff4c8 (2 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
```

LES CONFLITS DANS UN REBASE

- ▶ Les conflits lors d'un rebase sont à résoudre au fur et à mesure du replay
- ▶ On résout un conflit de rebase avec
 - ▶ `git add fichier_en_conflit...`
 - ▶ `git rebase --continue`
- ▶ Abandon du rebase avec
 - ▶ `git rebase --abort`

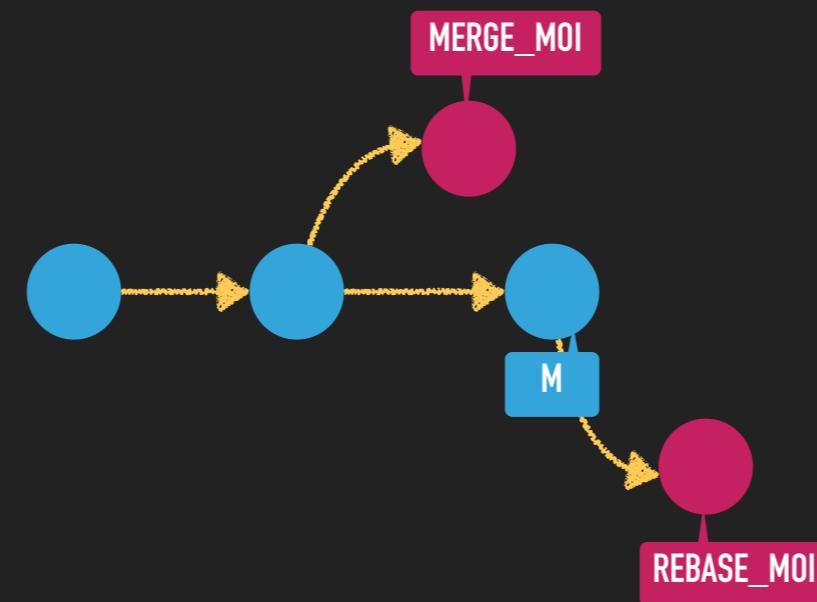
EXERCICE 3.1: NOUVELLE BRANCHE

- ▶ Reprendre le repository `cours_git`, branche `master`
- ▶ Créer une nouvelle branche `merge_moi` et se positionner dessus
- ▶ Rajouter le fichier `merge.md` dans le dossier *commandes*,
add + commit



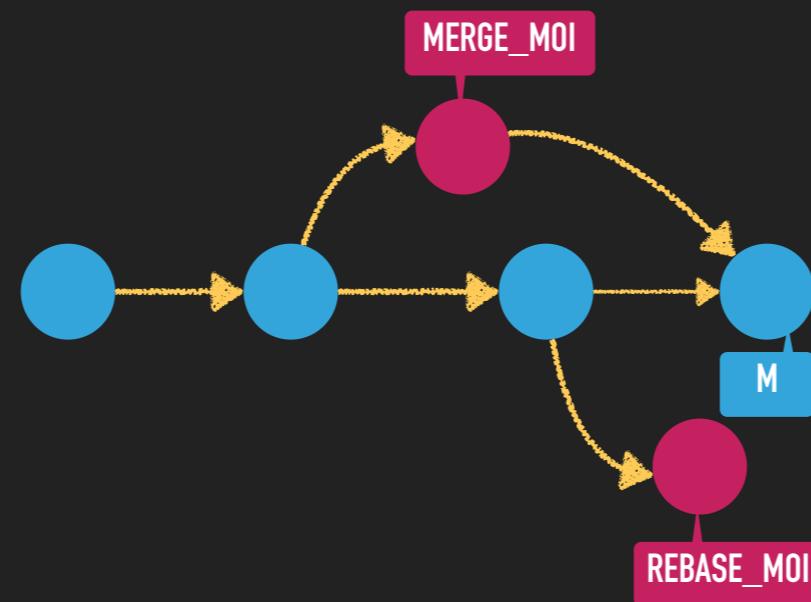
EXERCICE 3.2: NOUVELLE BRANCHE

- ▶ retourner sur master, ajoutez une ligne au README.md, **add + commit**
- ▶ Créer une nouvelle branche **rebase_moi** à partir de **master** et se positionner dessus
- ▶ Rajouter le fichier **rebase.md** dans le dossier **commandes**, **add + commit**



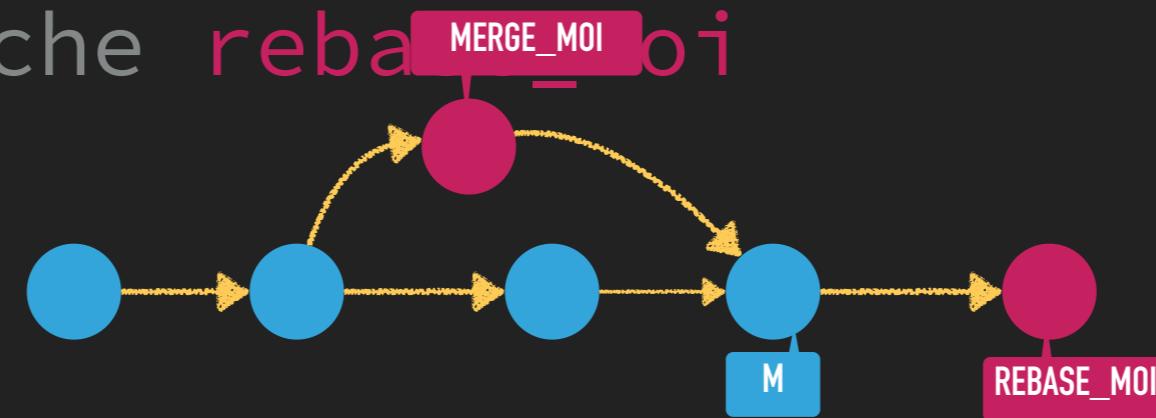
EXERCICE 3.3: MERGE DE LA BRANCHE MERGE

- ▶ Merger la branche `merge_moi` dans `master`
- ▶ `master` contient maintenant le fichier `merge.md`



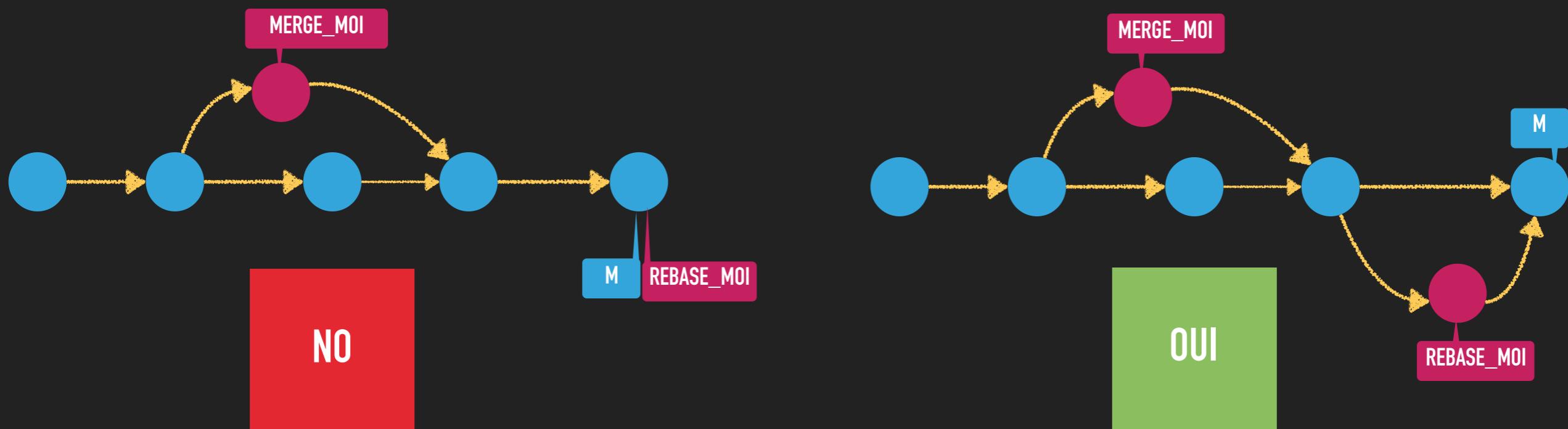
EXERCICE 3.4: REBASE DE LA BRANCHE REBASE

- ▶ Se placer sur `rebase_moi`, le fichier `merge.md` n'existe pas sur la branche `rebase_moi`.
- ▶ Rebaser la branche `rebase_moi` sur la branche `master`
- ▶ Le fichier `merge.md` est maintenant présent dans la branche `rebase_moi`

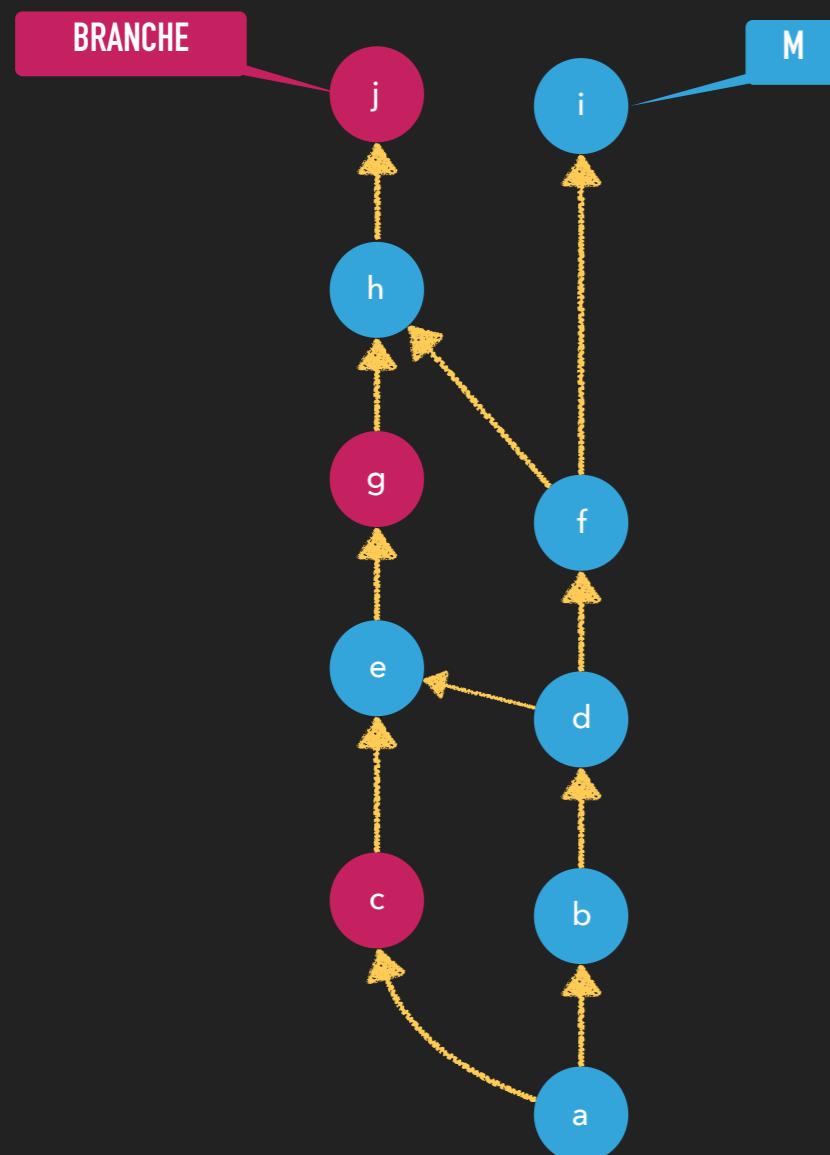


EXERCICE 3.5: MERGE DE LA BRANCHE REBASE

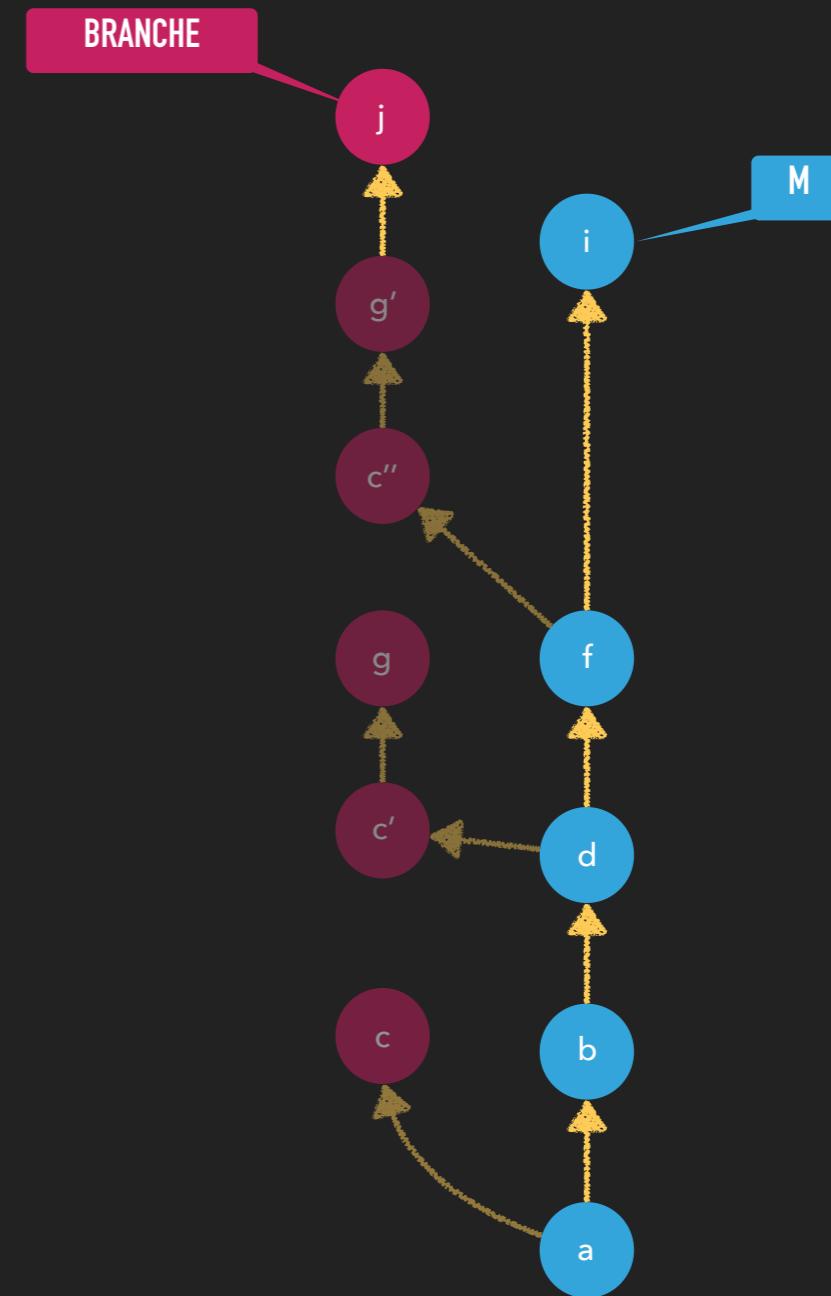
- ▶ Merger la branche `rebase_moi` sur `master`
- ▶ Faire en sorte qu'un commit de merge soit créé malgré la situation de descendance directe
- ▶ en cas d'erreur: `git reset --hard HEAD^`



REBASE >> MERGE



MERGE



REBASE

REBASE

- ▶ Rebaser une branche sur develop:

```
git rebase develop
```

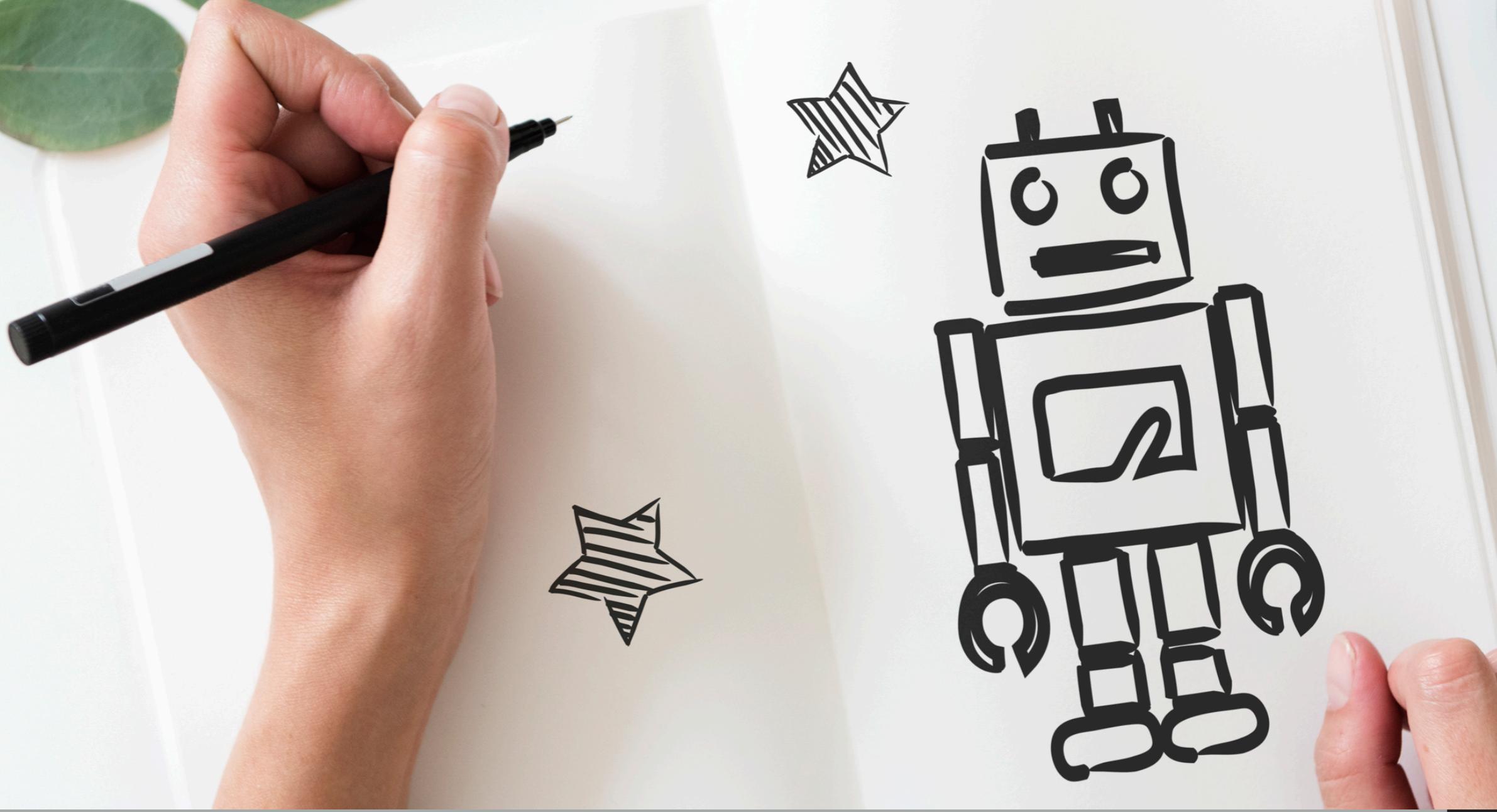
- ▶ Gérer un conflit sur fichier.txt:

```
git add fichier.txt
```

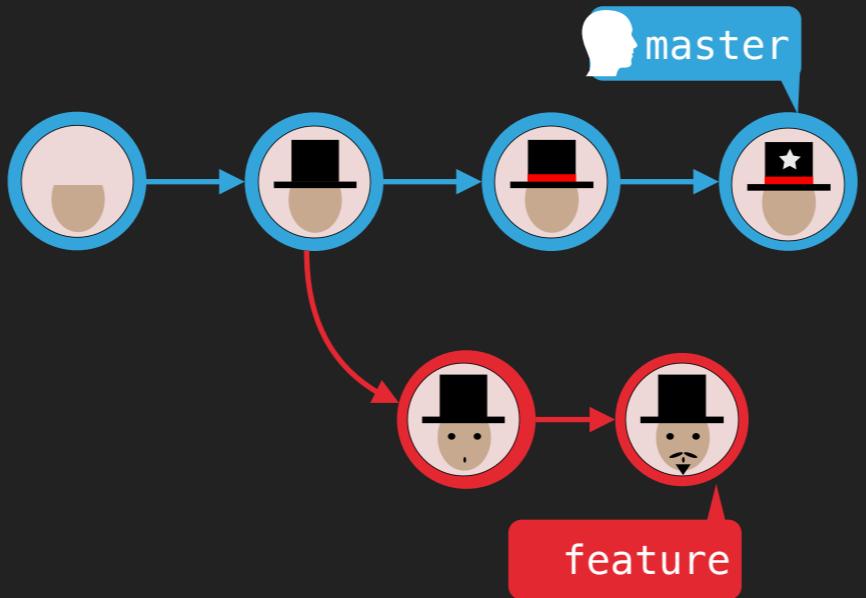
```
git rebase --continue
```

- ▶ Abandonner le rebase

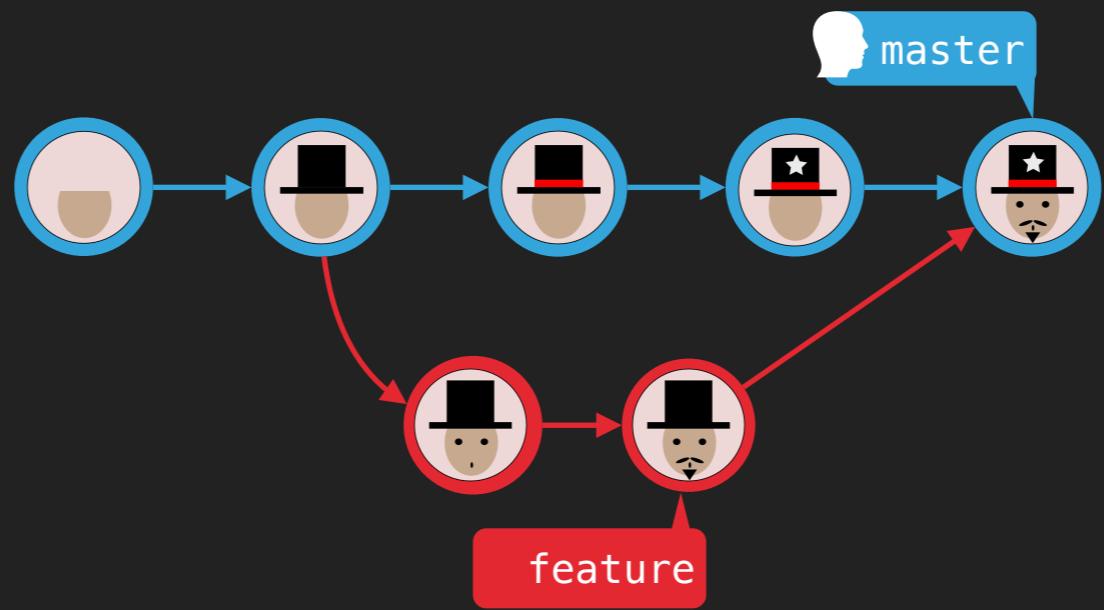
```
git rebase --abort
```

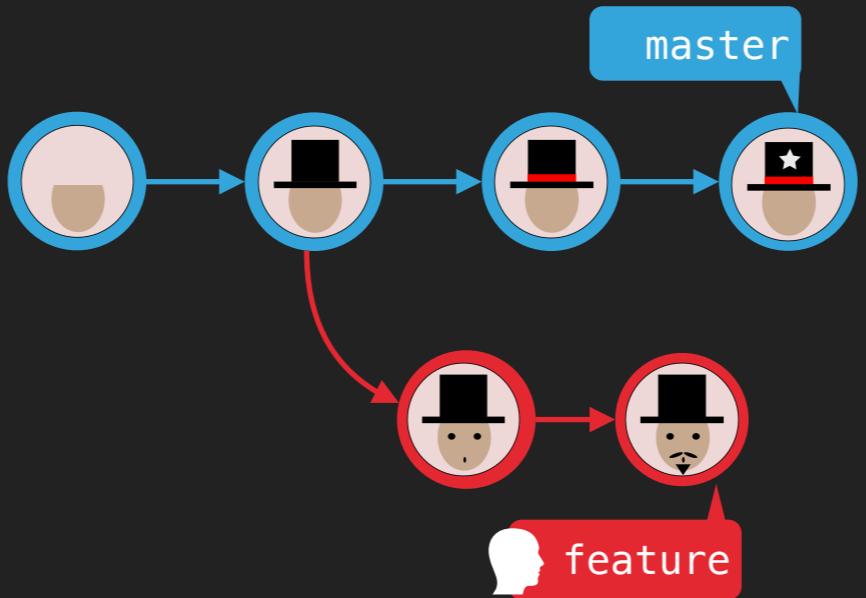


DRAW WITH GIT

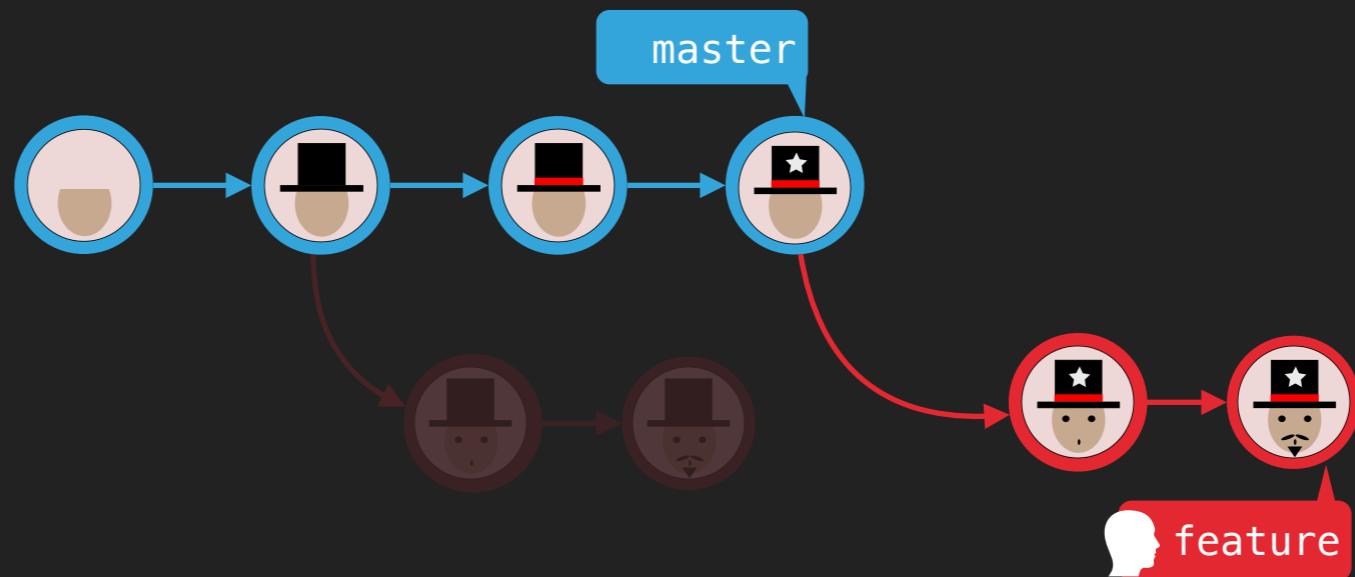


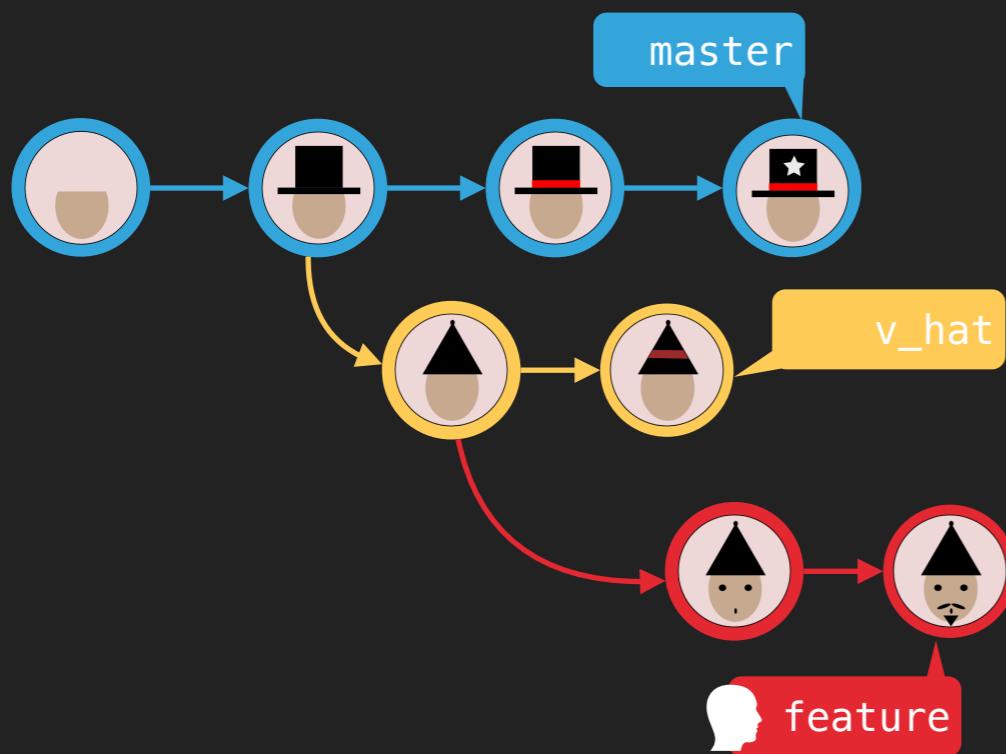
git merge feature



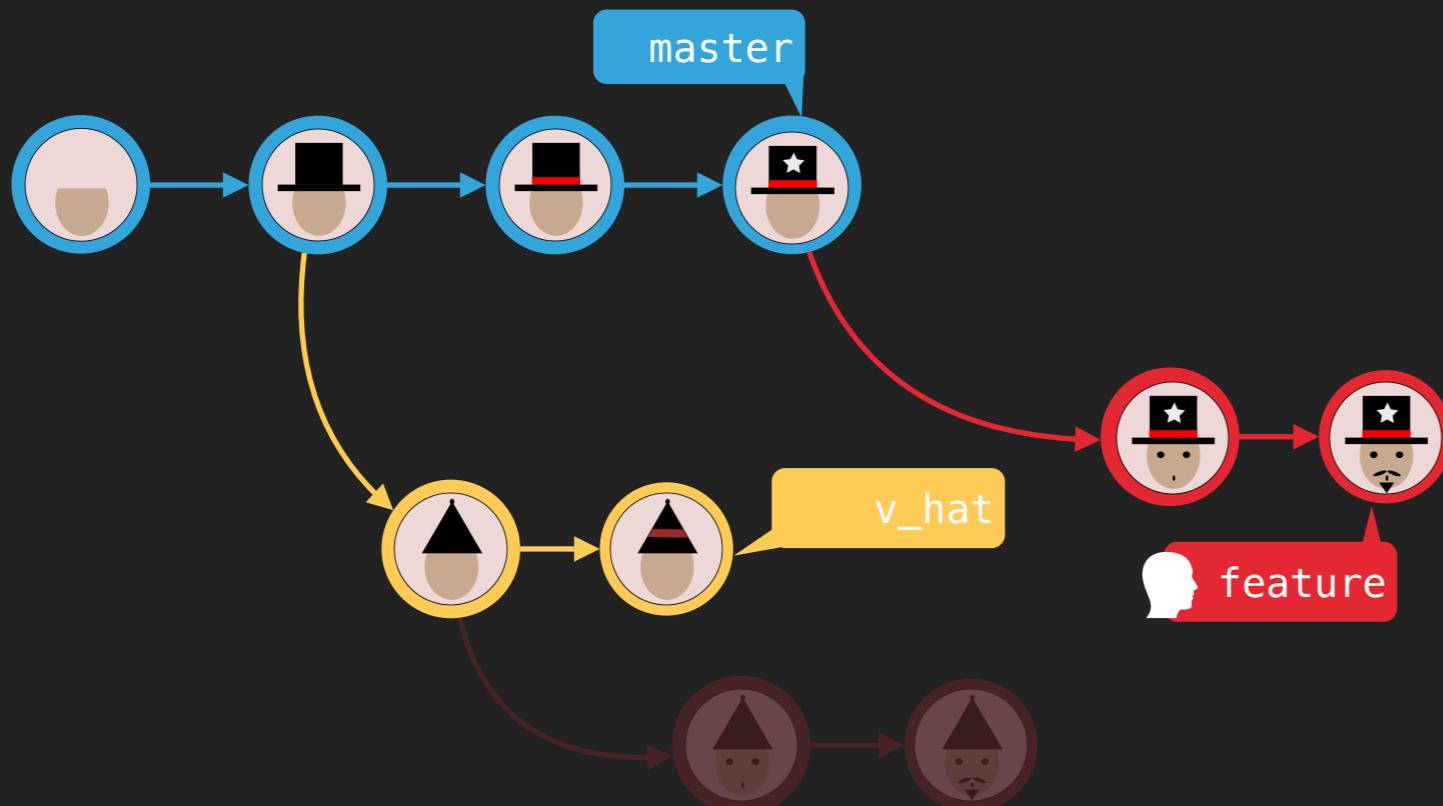


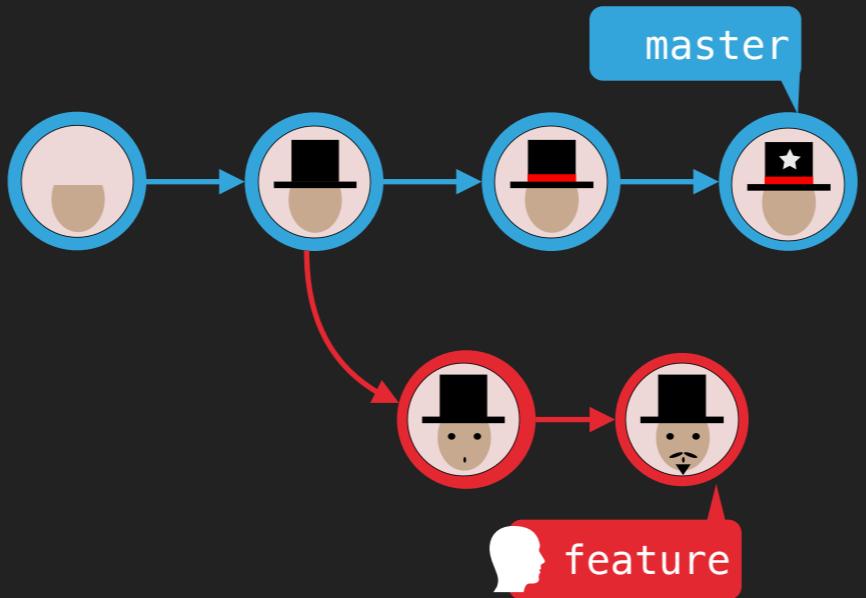
git rebase master





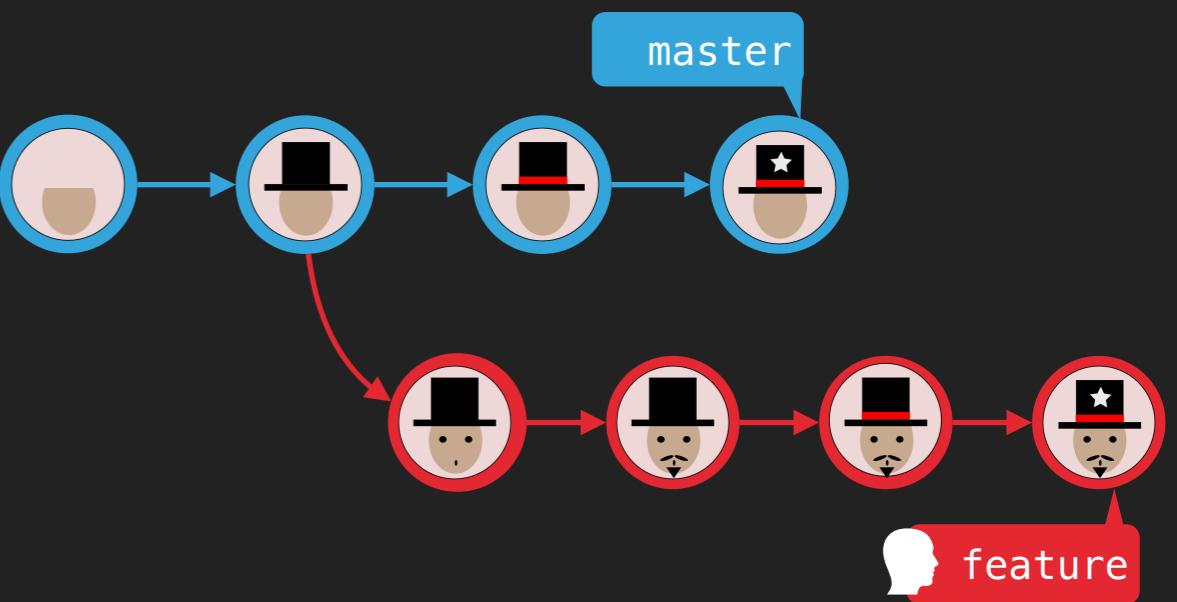
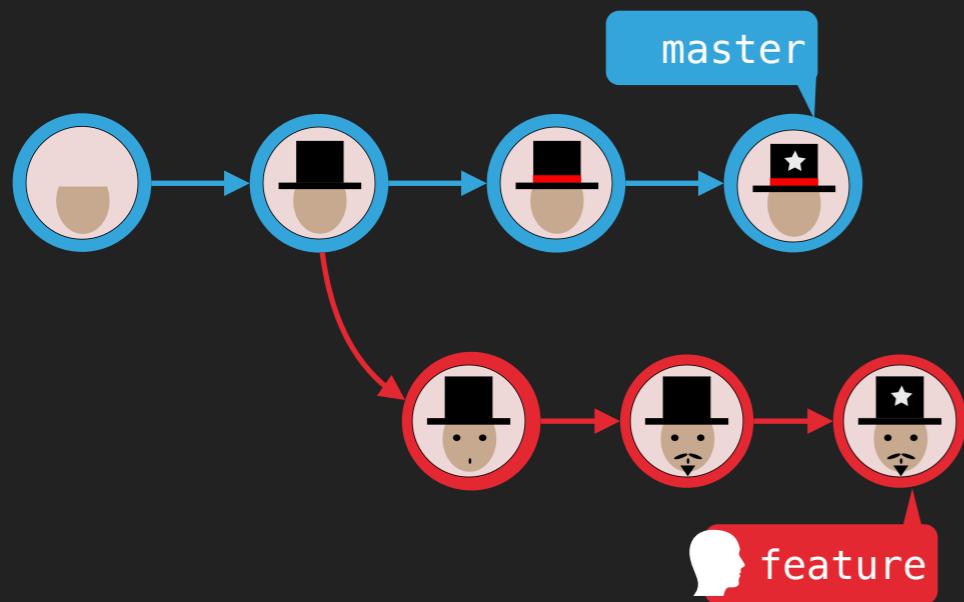
git rebase --onto master v_hat



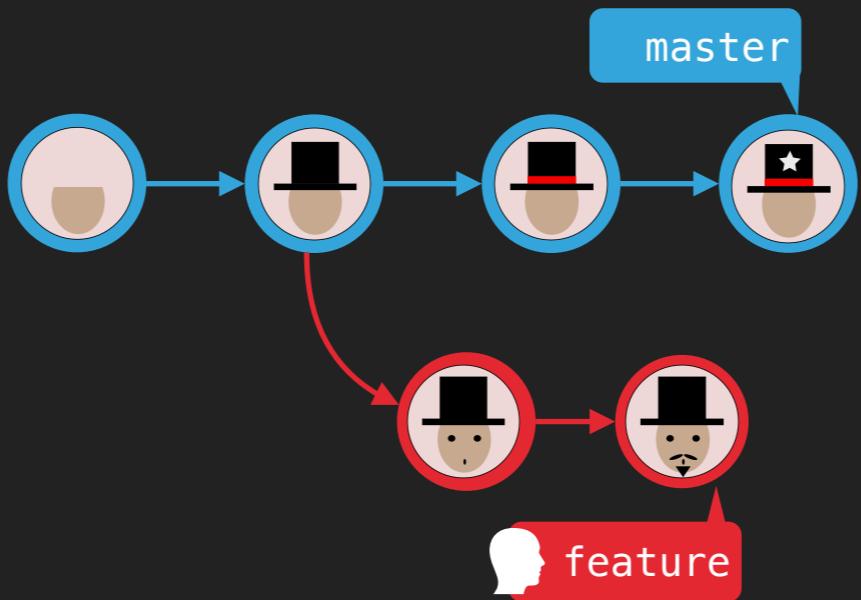


git cherry-pick master

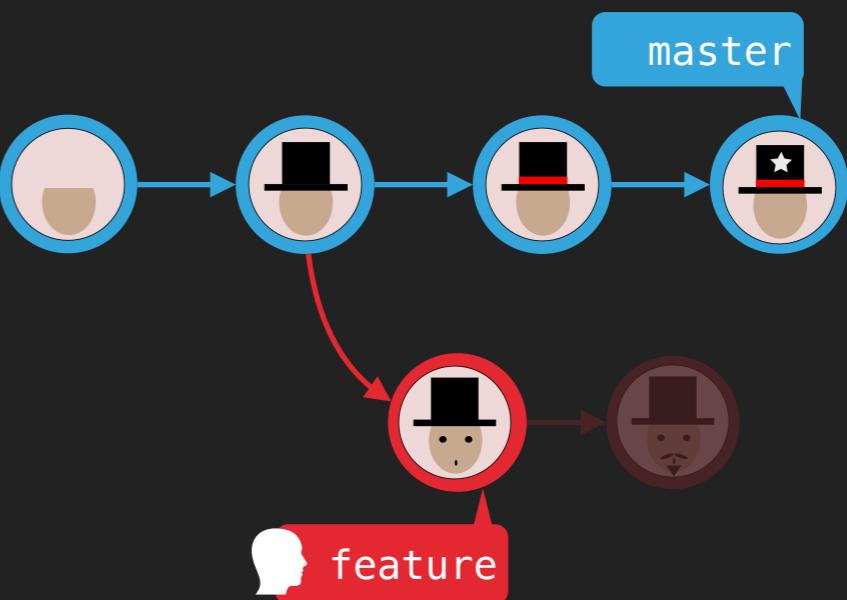
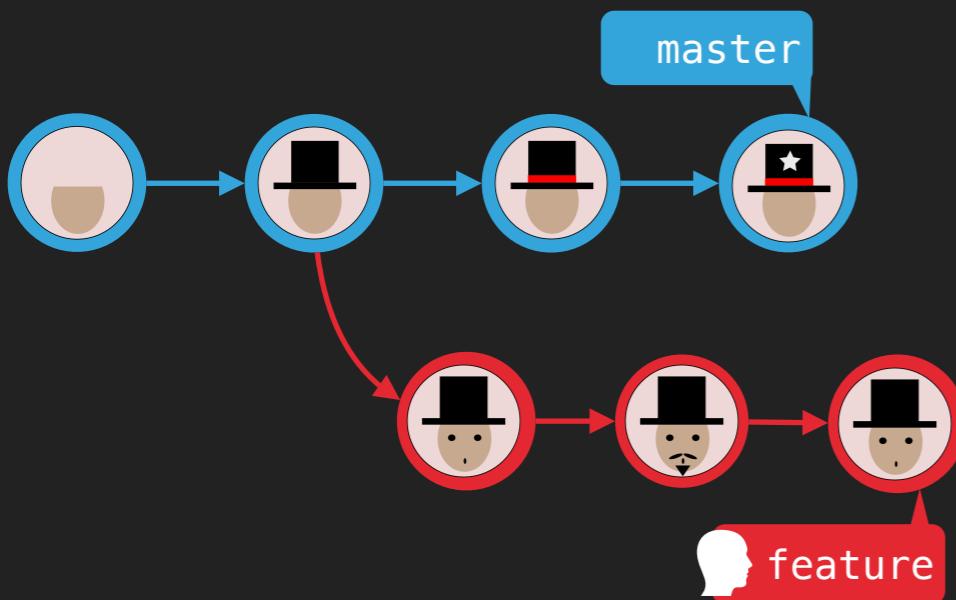
git cherry-pick ..master



git revert feature



```
git reset --hard feature~  
git reset --hard HEAD~  
git reset --hard @~
```





FAUT RÉFLÉCHIR, MCFLY, FAUT RÉFLÉCHIR !

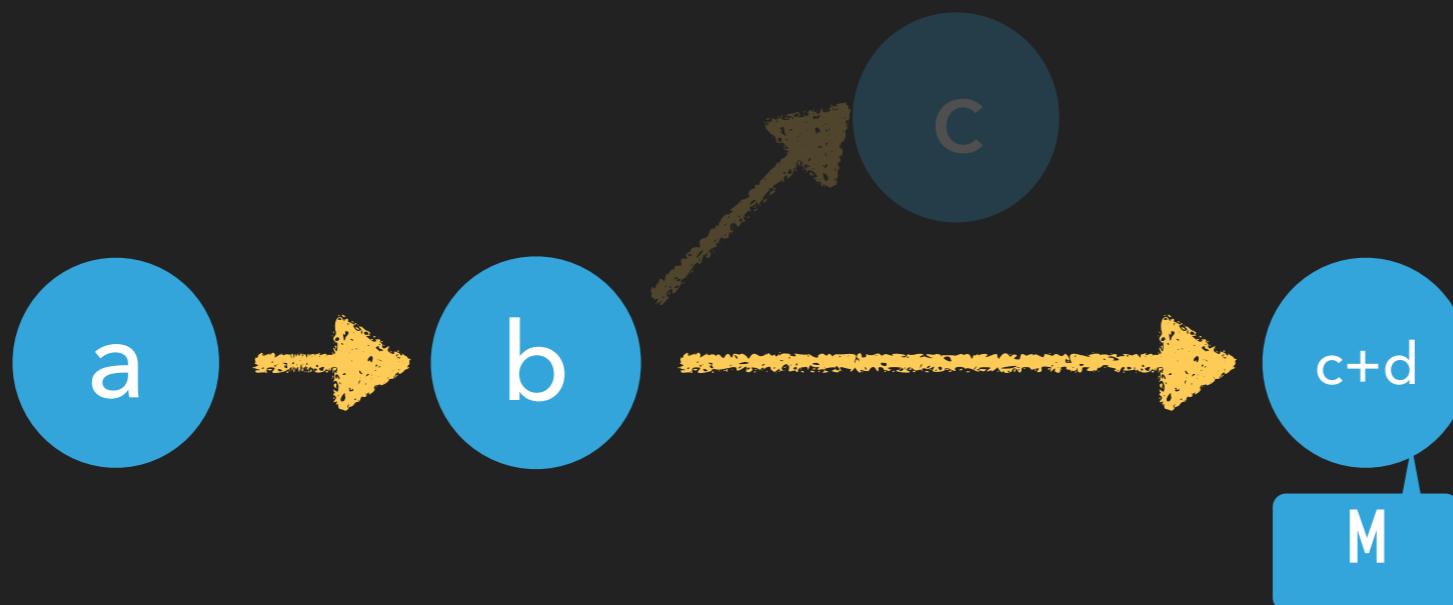
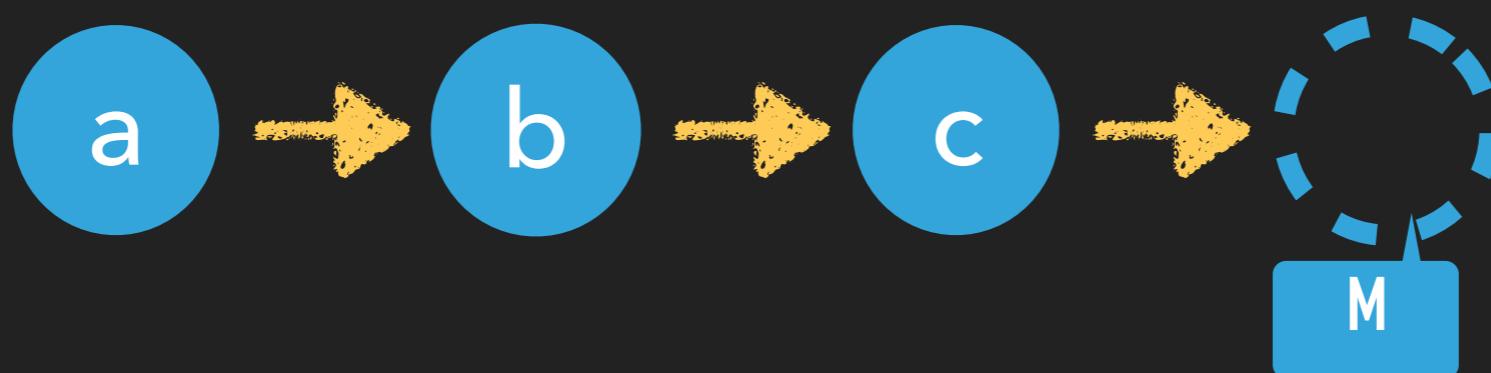
DES COMMANDES EN VRAC

git [add|rm|mv] <file>...

- ▶ add: planifier l'ajout d'un fichier
- ▶ rm: planifier l'effacement d'un fichier
- ▶ mv: renommer/déplacer un fichier

git commit --amend -m "surcharge le précédent"

- ▶ --amend permet de merger le commit créé avec le précédent



git tag TAGNAME -m “ceci est un tag”

- ▶ Crée une référence de type *tag*
- ▶ Contrairement à une branch un *tag* ne bouge pas
- ▶ Se placer sur un *tag* nous met en tête détachée
- ▶ -a pour annoté, ou -m pour message, permet d'ajouter des infos sur le tag

git config --global

- ▶ Sans **--global** la configuration est locale au repository dans lequel la commande est lancée
- ▶ Avec **--global** la configuration est valable pour tous les repository

LE FICHIER DE CONFIGURATION GLOBAL

~/.gitconfig

```
[user]
  name = Emmett Brown
  email =emmett.brown@hill_valley.com
[push]
  default = simple
[alias]
  co = checkout
  ci = commit
  st = status
  br = branch
[core]
  excludesfile = /Users/emmett/.gitignore_global
  autocrlf = input
```



LES OPTIONS PRINCIPALES

- ▶ `git config --global user.name "Marty MacFly"`
- ▶ `git config --global user.email "m.mfly@hvlly.com"`
- ▶ `git config --global core.autocrlf false` 
- ▶ `git config --global core.filemode false`

git config alias.ALIAS ‘commande longue’



```
git config --global alias.co checkout
git config --global alias.ci commit
git config --global alias.st status
git config --global alias.br branch
git config --global alias.hist 'log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short'
git config --global alias.type 'cat-file -t'
git config --global alias.dump 'cat-file -p'

doc@labo$ git st
# On branch develop
nothing to commit (working directory clean)
```

```
# File .bashrc
# bash shell aliases
alias gss='git status'
alias gb='git branch'
alias gp='git push'
```

CHEAT SHEET - [HTTP://OVERAPI.COM/GIT](http://OVERAPI.COM/GIT)

OverAPI.com Python jQuery NodeJS PHP Java Ruby Javascript ActionScript CSS Express More »

Search

Resource

Online
Official Website

Download
GitCheatSheet [.pdf]
Git cheat sheet, extended edition prepared [.svg]
Git Quick Reference [.pdf]
Git Cheat Sheet In Chinese

Related
Bazaar
CVS
SVN

Branching/Tagging

List branches
git branch

Switch to branch
git checkout branch

Create new branch
git branch new

Create branch from existing
git branch new existing

Delete branch
git branch -d branch

Tag current commit
git tag tagname

Create Git

From existing directory

cd project_dir
git init
git add .

From other repository

git clone existing_dir new_dir
git clone git://github.com/user/repo.git
git clone https://github.com/user/repo.git

Remote Update / Publish

List remotes

git remote -v

Show information

git remote show remote

Add remote

git remote add path/url

Fetch changes

git fetch remote

Fetch + merge

git pull remote branch

Publish local to remote

git push remote :branch

Delete remote branch

git push remote :branch

Publish tags

git push origin/upstream --tags

Local Changes

Changed in working directory

git status

Tracked file changes

git diff

Add changed files

git add file1 file2 file3

Remove file

git rm file

git rm dir/ -r

(recursive under directory)

See files ready for commit

git diff --cached

Commit changes

git commit
git commit -m "My message"
git commit -a -m "My Message"
(tracked files only, auto add)

Change last commit

git commit --amend

Revert changes to file

git checkout -- file

Revert changes (new commit)

git revert HEAD

Return to last committed state

git reset --hard HEAD

History

Show all commits

git log

Short Format

git log --pretty=short

Patches

git log -p

Show file commits

git log file

Show directory commits

git log dir/

Stats

git log --stat

Who changed file

git blame file

Merge/Rebase

Merge branch into current
git merge branch

Rebase into branch
git rebase branch
git rebase master branch

Abort rebase
git rebase --abort

Merge tool to solve conflicts
git mergetool

To view the merge conflicts
git diff
complete conflict diff
git diff --base \$file
against base file
git diff --ours \$file
against your changes
git diff --theirs \$file
against other changes

To discard conflicting patch
git reset --hard
git rebase --skip

After resolving conflicts
git add \$conflicting_file
do for all resolved files
git rebase --continue

Useful Commands

Finding Regressions

git bisect start
to start
git bisect good \$id
\$id is the last working version
git bisect bad \$id
\$id is a broken version
git bisect bad/good
to mark it as bad or good
git bisect visualize
to launch gitk and mark it
git bisect reset
once you're done

Check for Errors and Cleanup Repository

git fsck
git gc --prune

Search Working Directory for foo()

git grep "foo()"

Ad

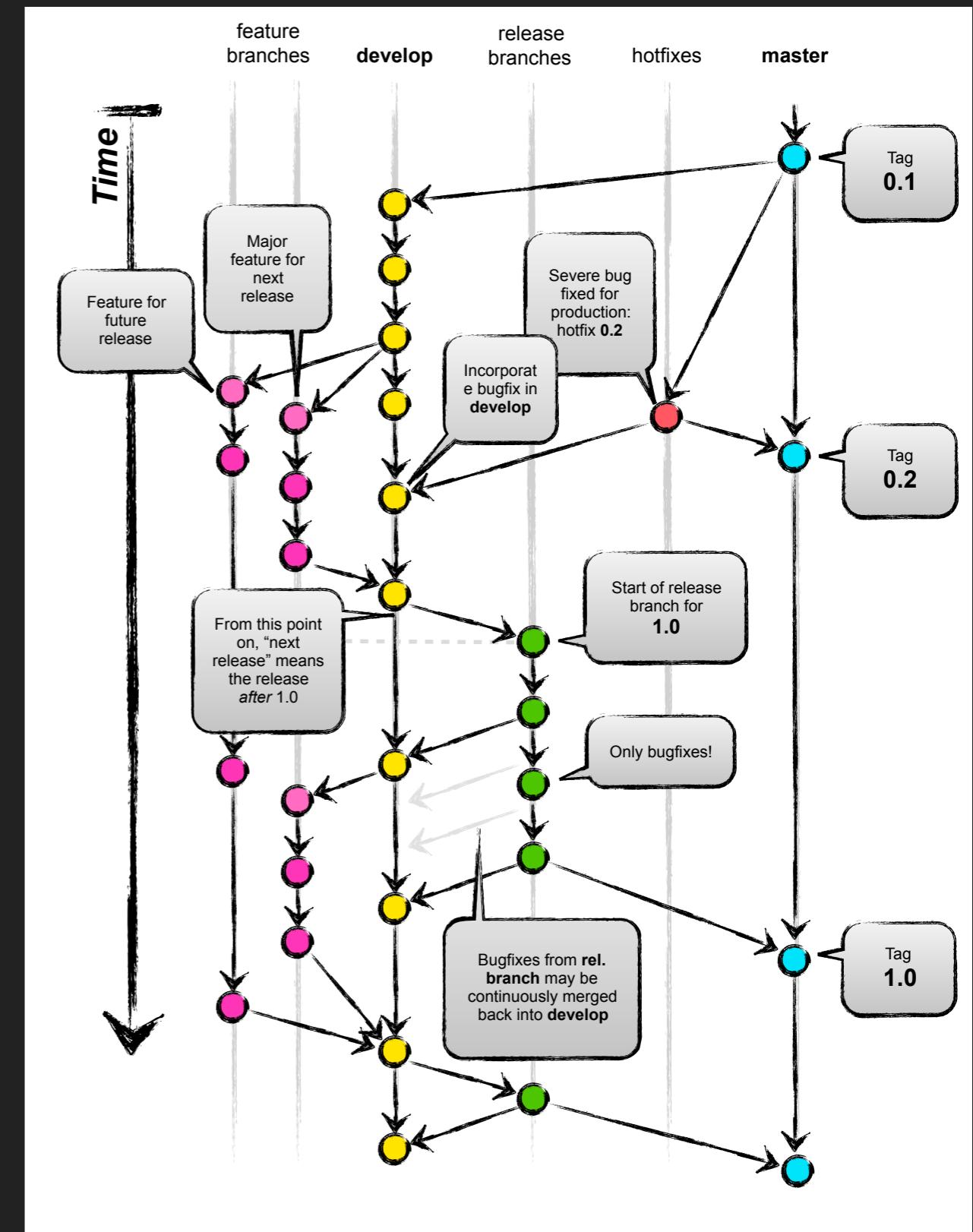
```
005 17:33 best\DDREN 816 Wikis 2.451
004 21:27\external\sushi@iB 816 Wikichang
004:\Progs\external\sushi@iB 816 . 2.451
004 21:28 <DIR> .
004 17:33 best\DDREN .
005 17:33 best<DIR> .
005 17:33 best\DDREN .
005:\Progs\external\sushi@iB 816 Wikis 2.451
004:\Progs\external\sushi@iB 816 Wikis 2.451
005:\Progs\external\sushi@iB 816 . 2.451
005 17:33 best<DIR> .
005 17:03 best\DDREN .
005 17:03 best\DDREN .
005:\Progs\external\sushi@iB 816 Wikis 2.451
005:\Progs\external\sushi@iB 816 Wikibabg
005 21:28 <DIR> 1.818 Wikibabg
2005 17:33 best<DIR> .
2005 17:33 best\DDREN .
```

LES TYPES DE BRANCHES

NVIE.COM

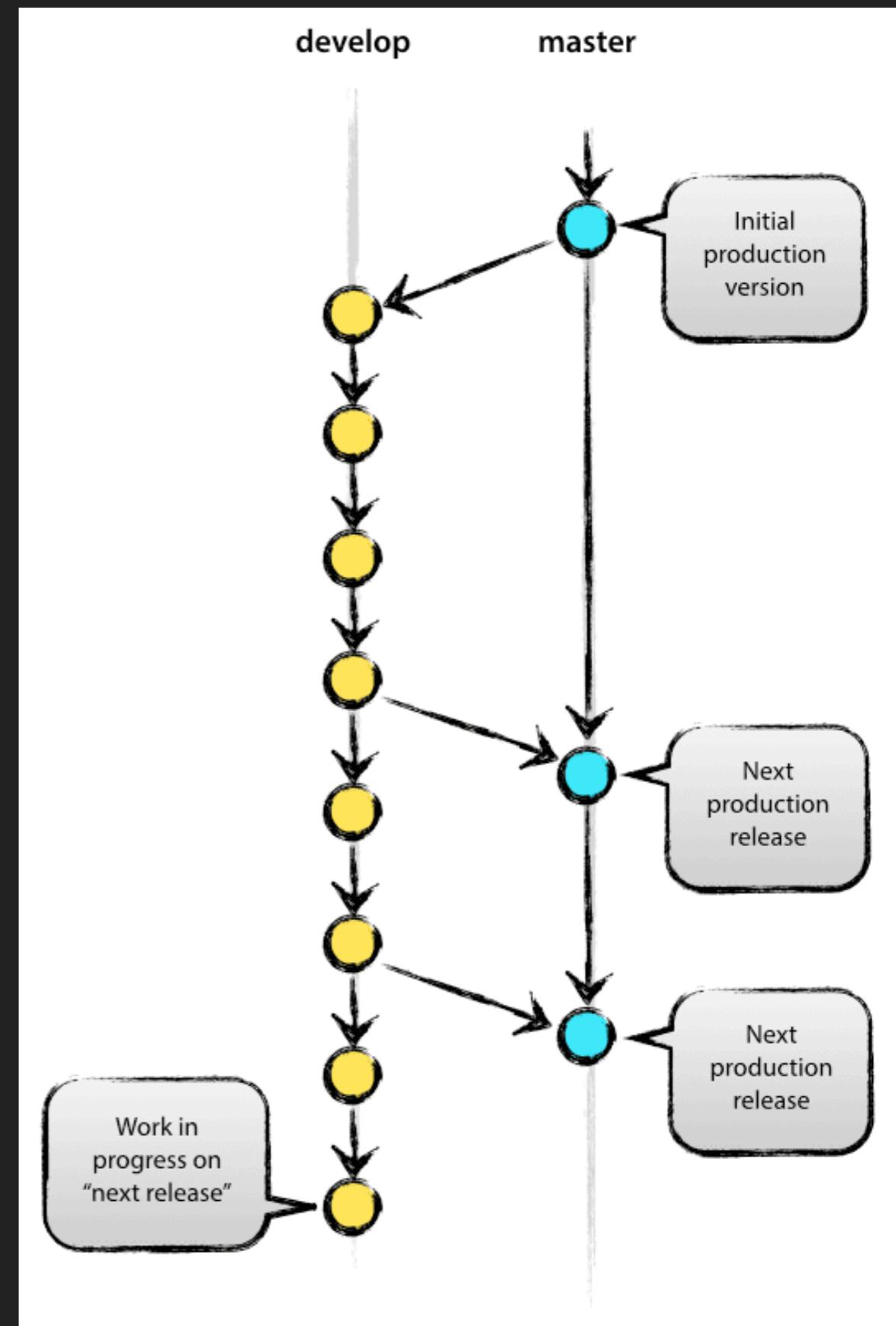
GUIDE DE BRANCHES

- ▶ Règles de création de branches
- ▶ Règles de nommage de branches
- ▶ **master, develop, feature, release, hotfix**
- ▶ Que faire dans chaque branche



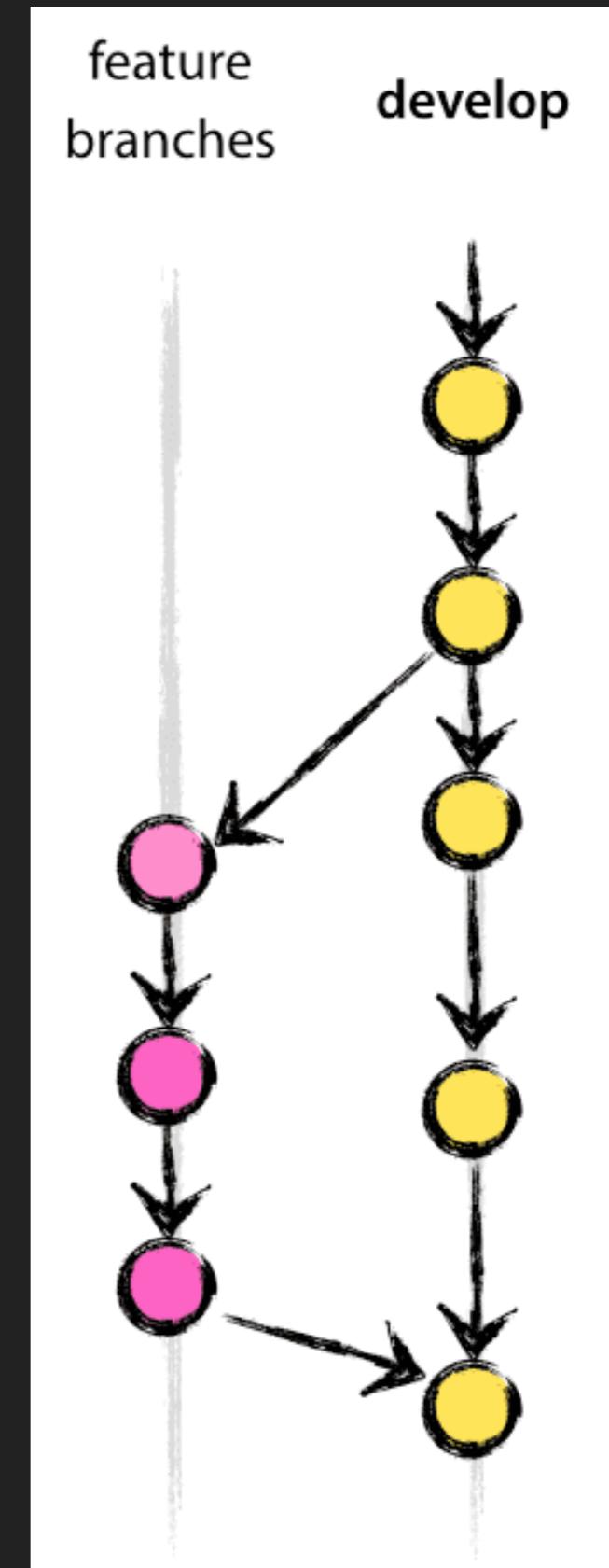
BRANCHES PRINCIPALES

- ▶ **master**
- ▶ stable
- ▶ production
- ▶ **develop**
- ▶ instable
- ▶ développement actif
- ▶ préparation de la prochaine release



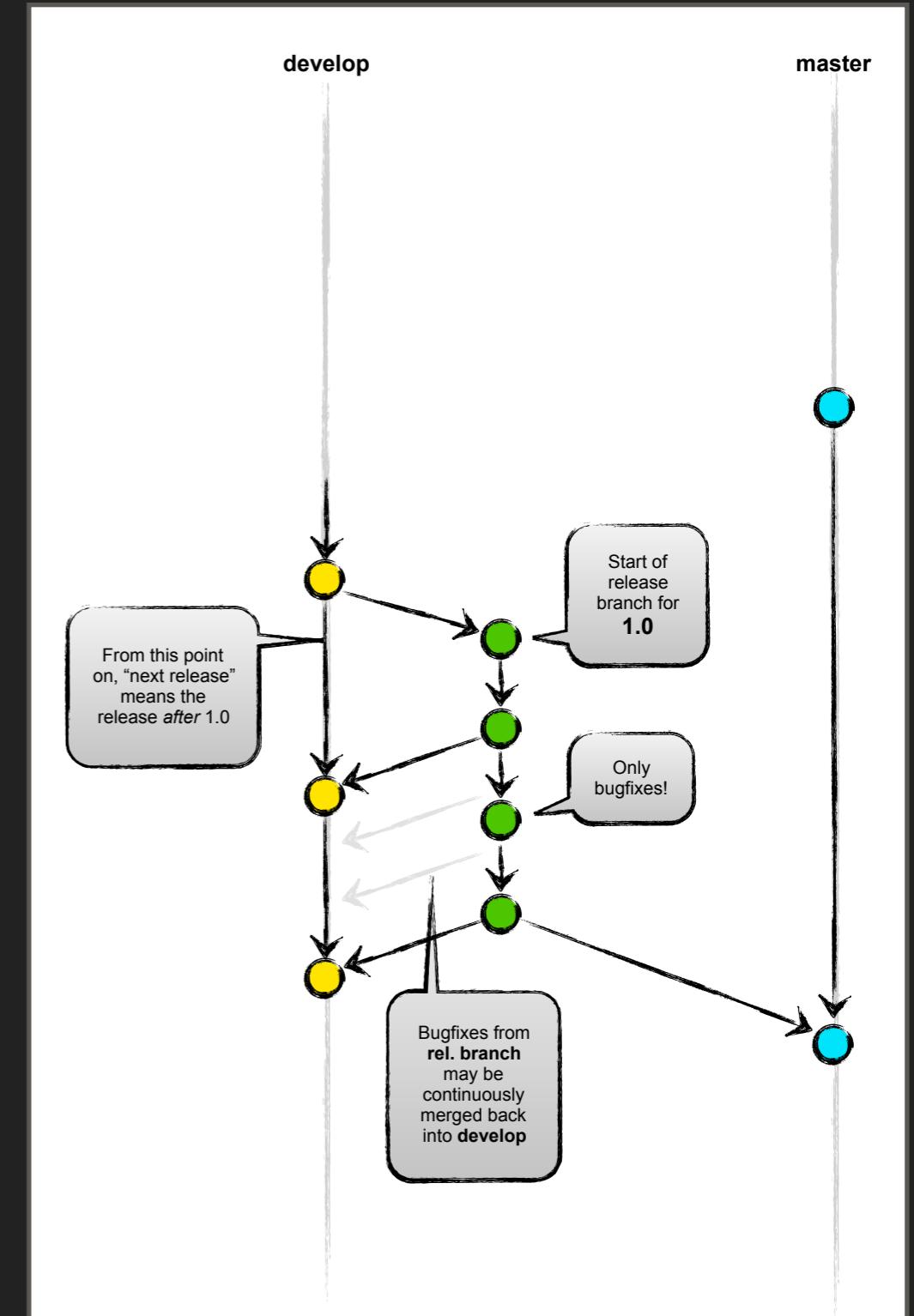
BRANCHES DE FONCTIONNALITÉ

- ▶ **feature/***
- ▶ **UNE fonctionnalité par branche**
- ▶ merge dans develop
- ▶ merge-request/pull-request



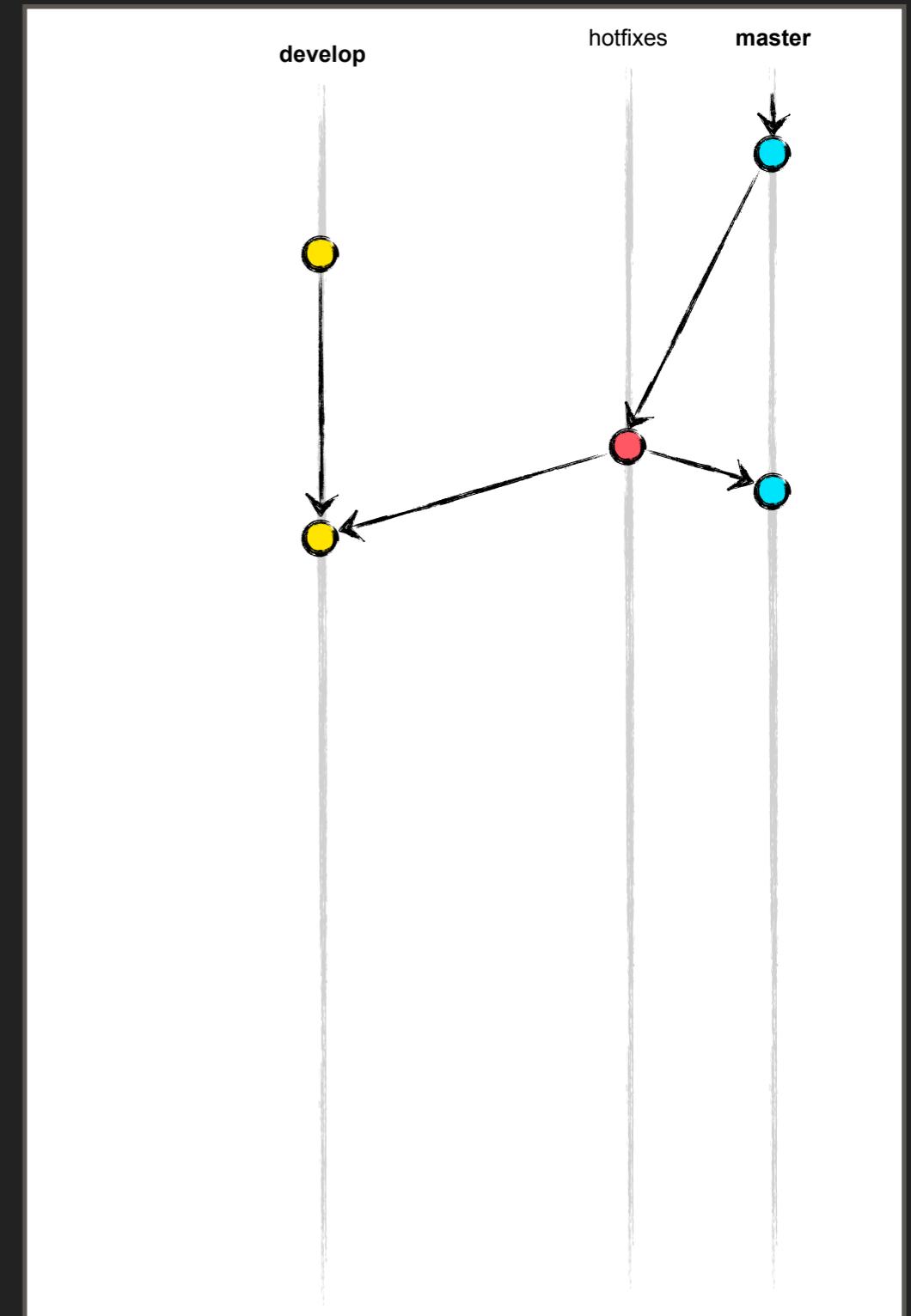
BRANCHE DE LIVRAISON

- ▶ **release/***
- ▶ modifications relatives à une livraison
- ▶ Remplissage de la note de version
- ▶ Correction de bugs mineurs
- ▶ merge dans **master** et **develop**



BRANCHE DE HOTFIX

- ▶ **hotfix/***
- ▶ correction de bug sur version livrée
- ▶ merge dans **master** et **develop**



NVIE.COM

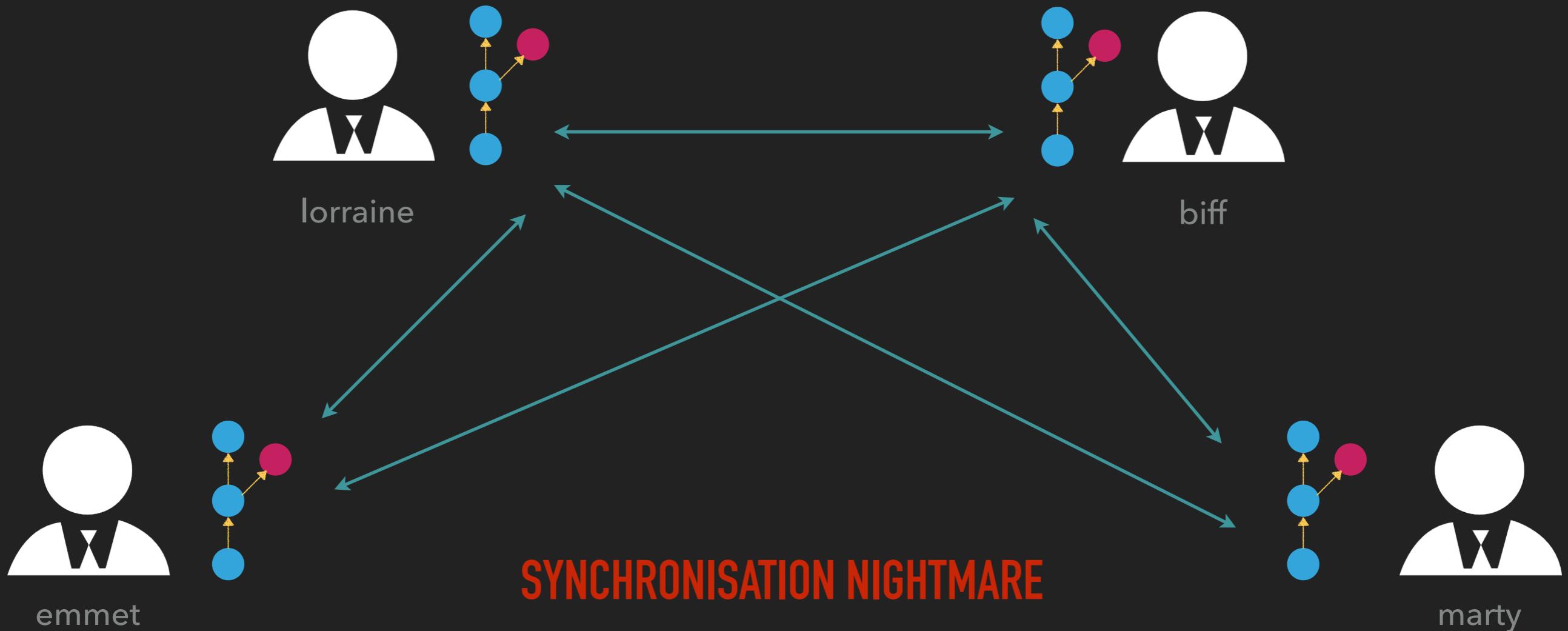
- ▶ Les branches principales:
master et **develop**
- ▶ Les branches de fonctionnalités:
feature/* ou **feature-***
- ▶ Les branches de release
release/* ou **release-***
- ▶ Les branches de hotfix
hotfix/* ou **hotfix-***



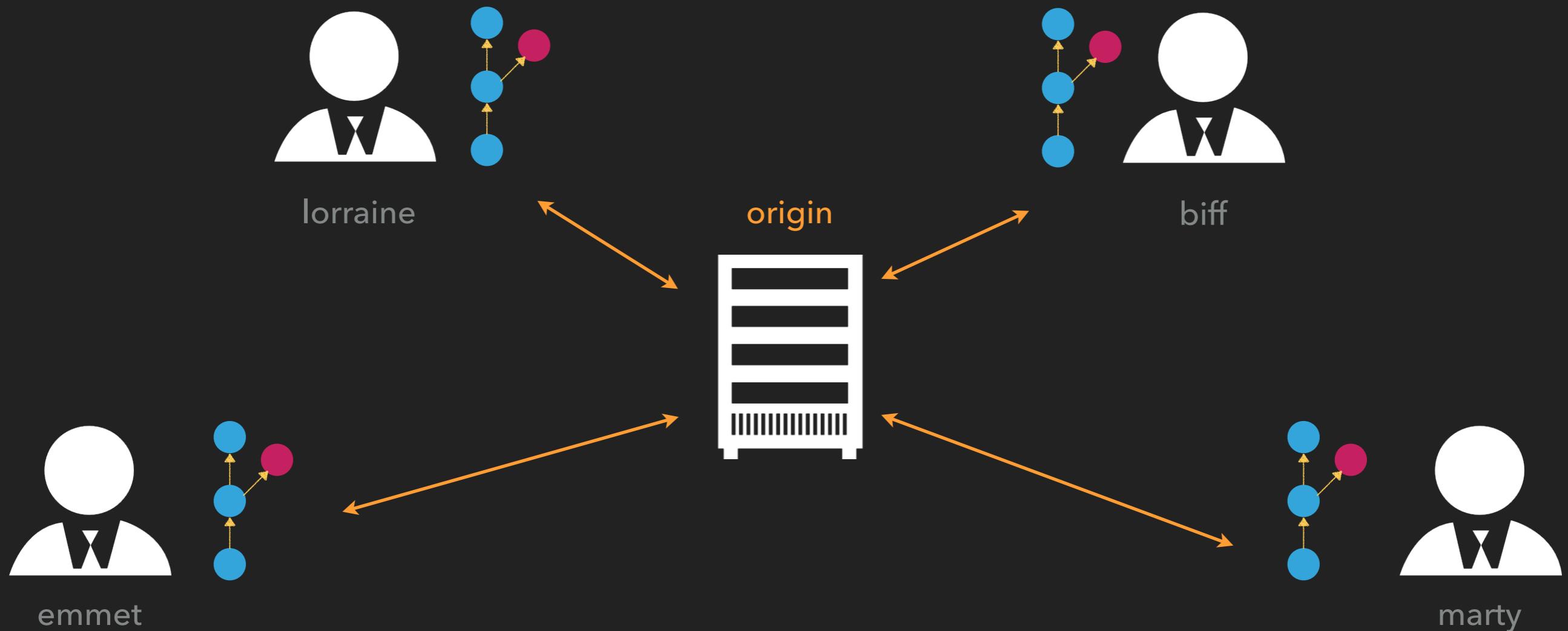
2,21 GIGOWATTS !!! 2,21 GIGOWATTS !!! MON DIEU !!!

GIT MULTIJOUEURS

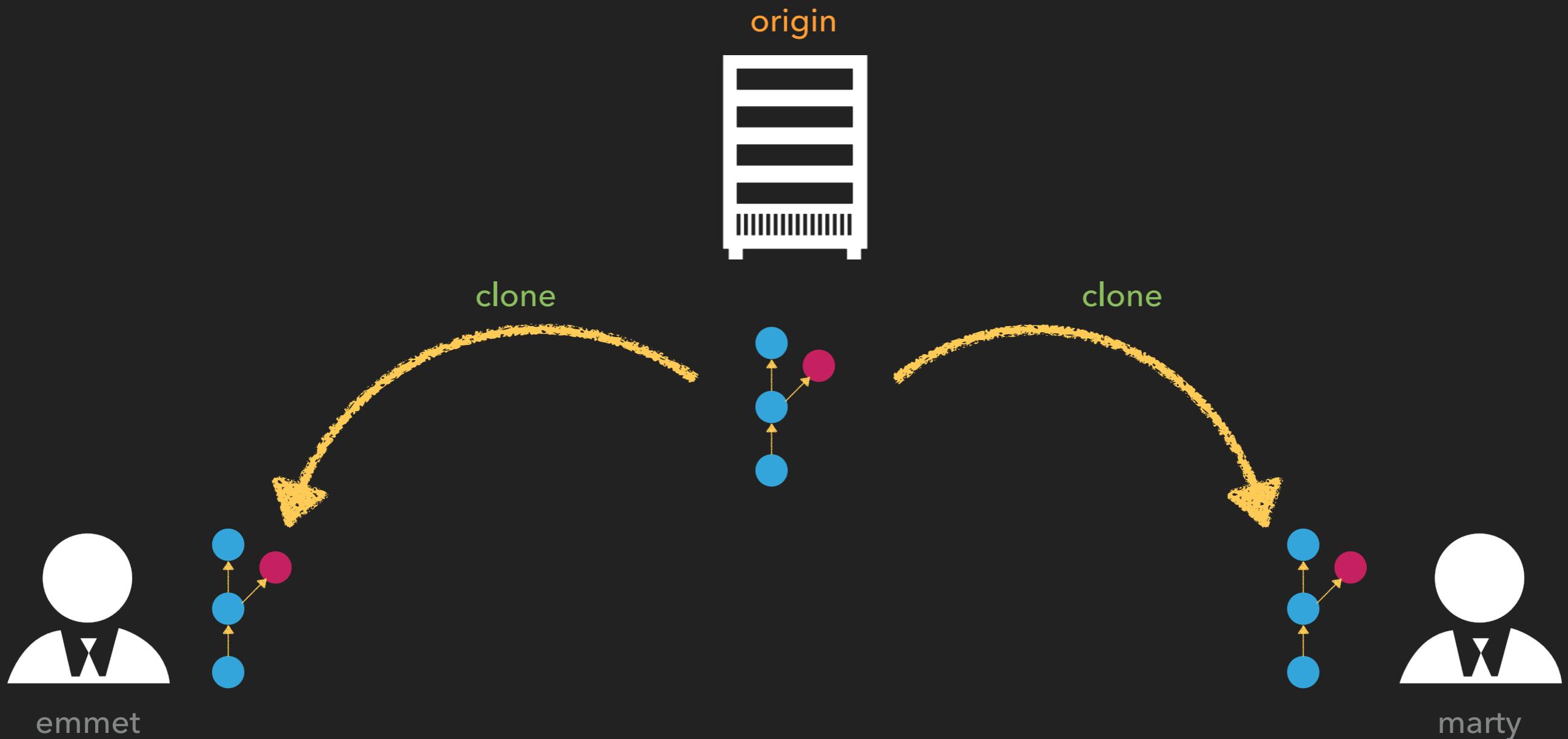
DES REMOTES PAR MILLIERS



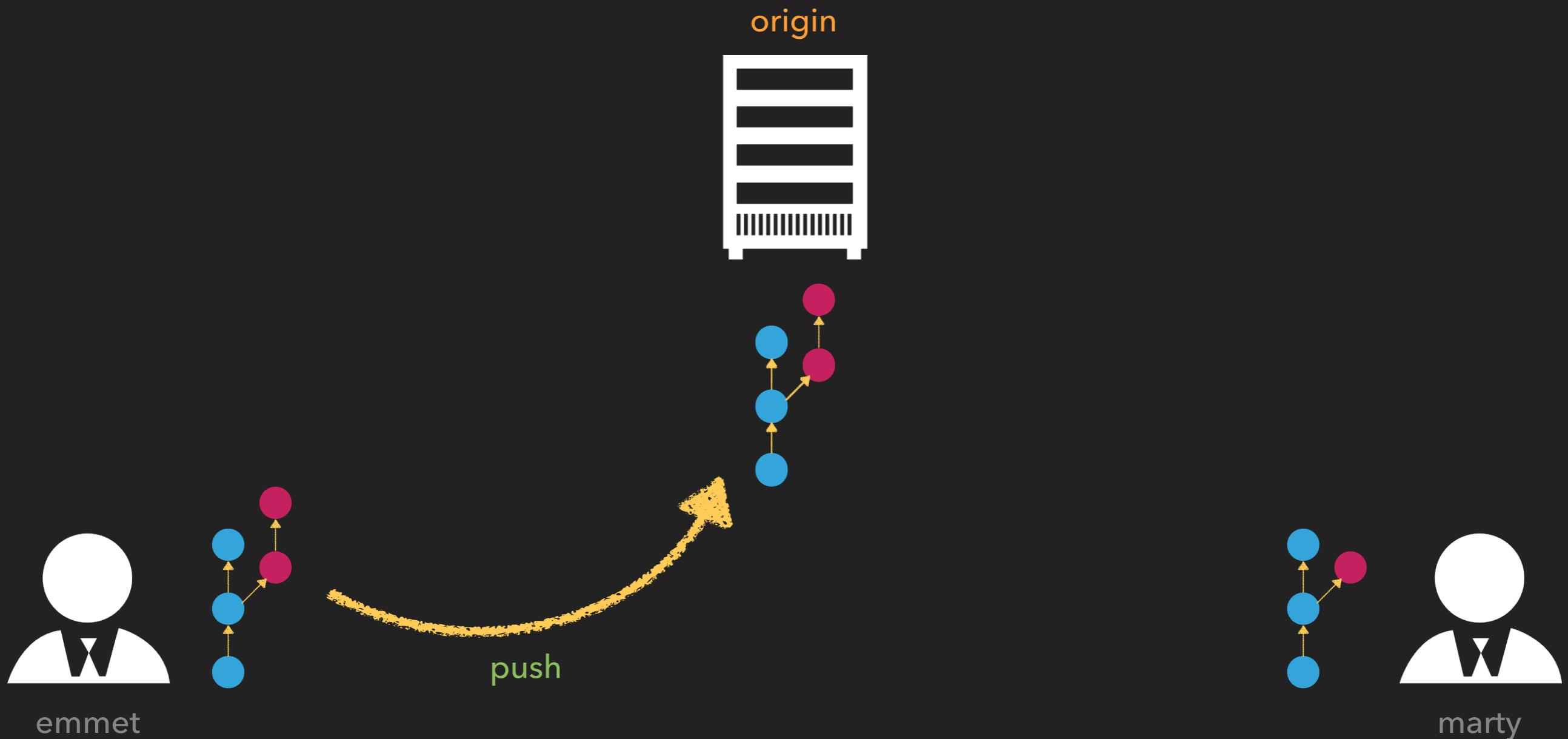
DES REMOTES PAR MILLIERS MAIS UNE SEULE ORIGINE



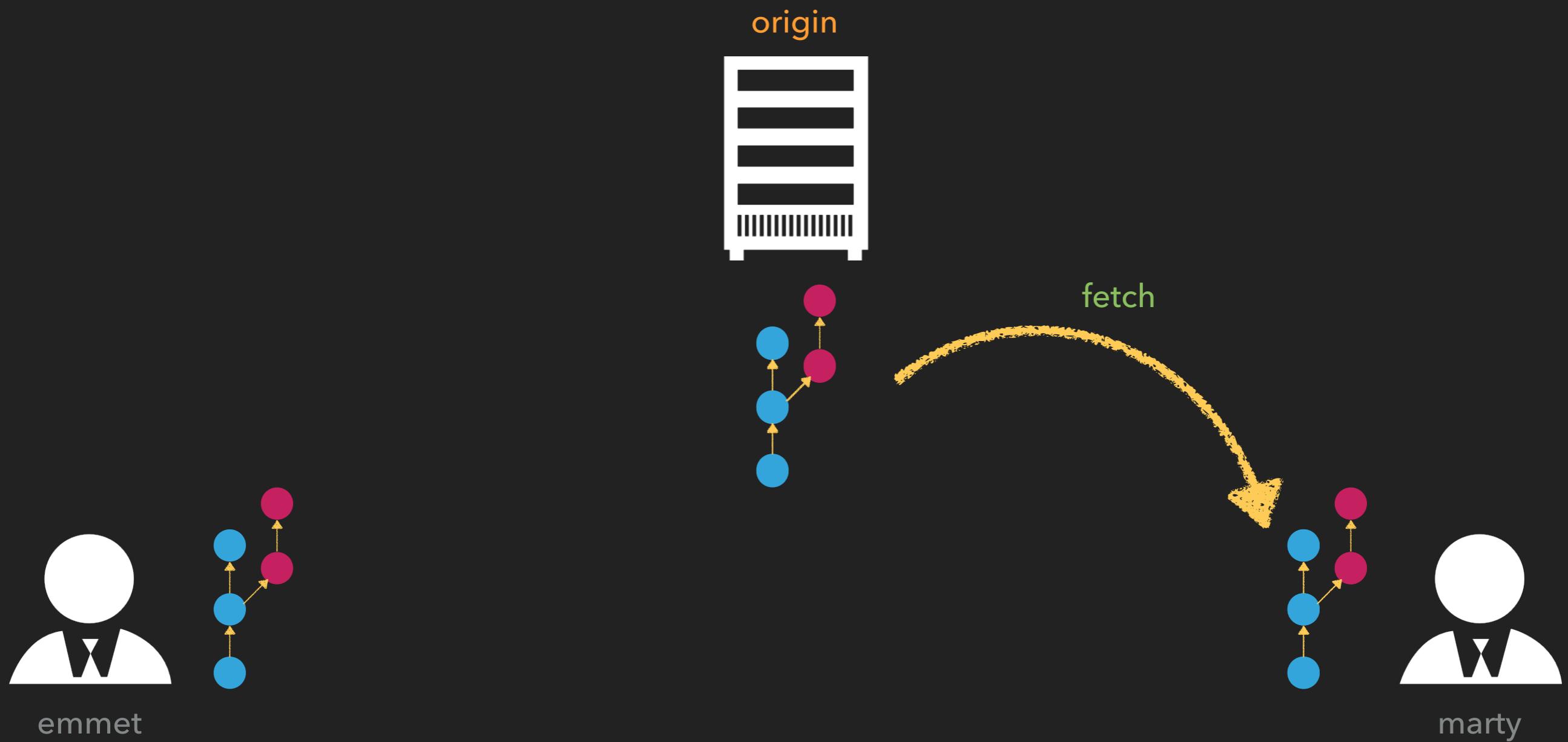
CREER LA COPIE LOCALE



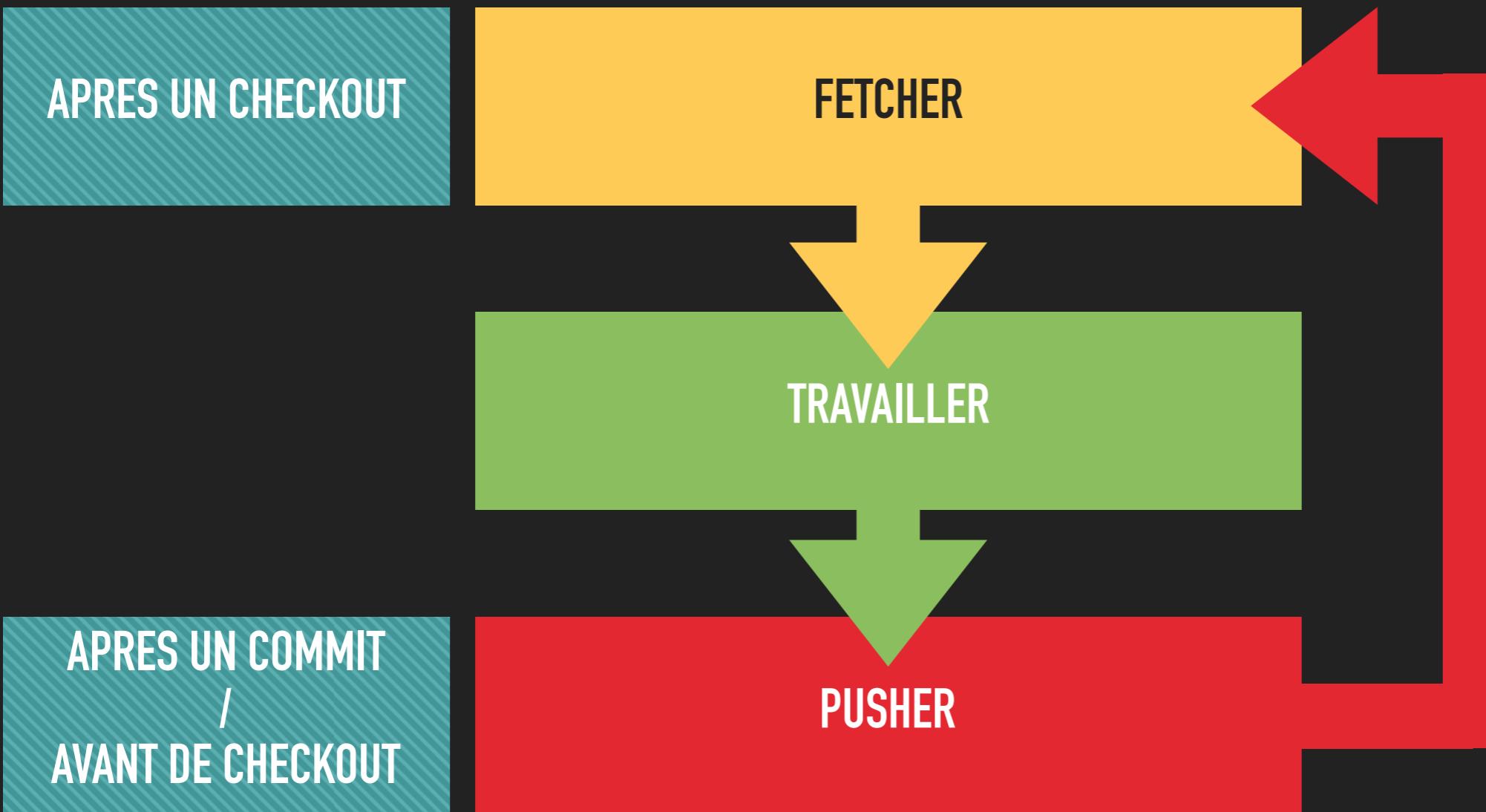
TRAVAILLER ET POUSSER



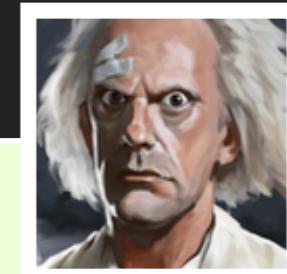
RÉCUPÉRER AVANT DE TRAVAILLER



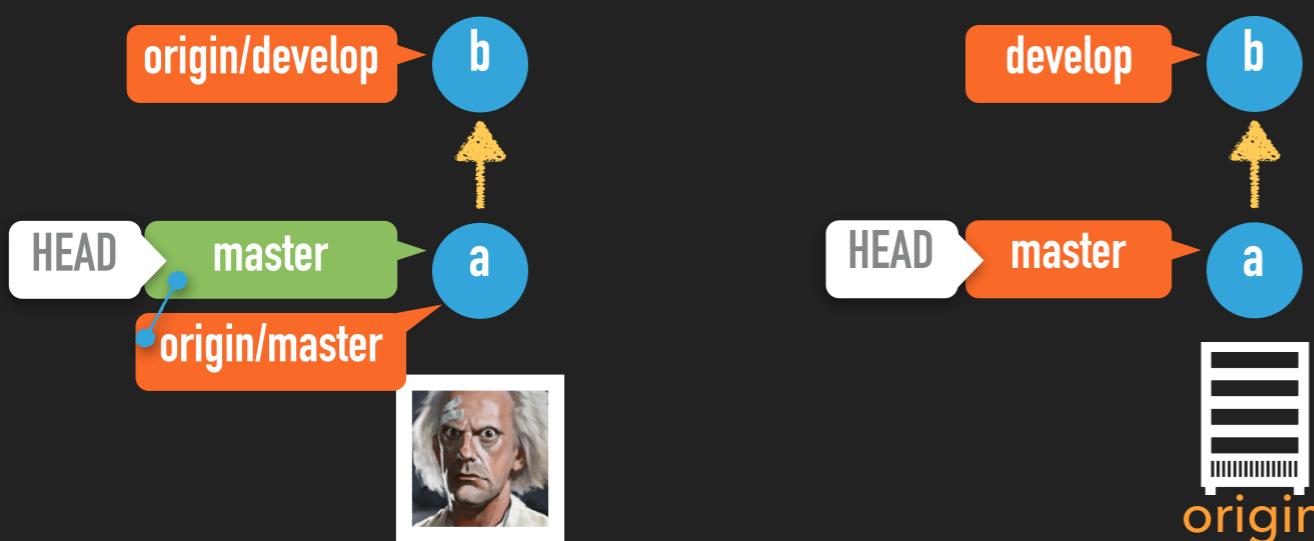
LES 3 PHASES DU TRAVAIL À PLUSIEURS



git clone url [path]

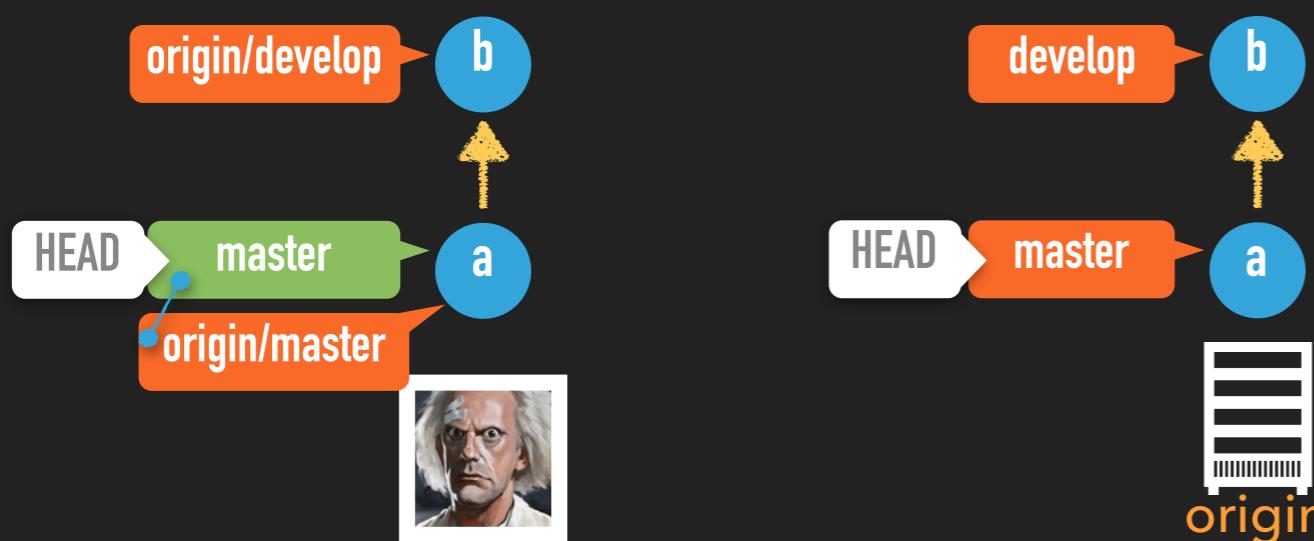


```
doc@labo$ git clone ssh://git@algec.iut-blagnac.fr:2200/git_training/basic.git
Cloning into 'basic'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (6/6), 628 bytes | 0 bytes/s, done.
Checking connectivity... done.
doc@labo$ cd basic/
doc@labo$ ls
README.md
doc@labo$ git branch
* master
doc@labo$ git branch -r
  origin/HEAD -> origin/master
  origin/develop
  origin/master
```

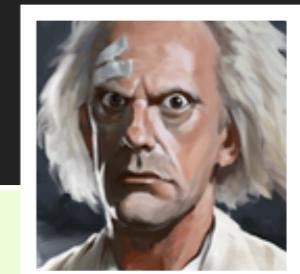


git clone url [path]

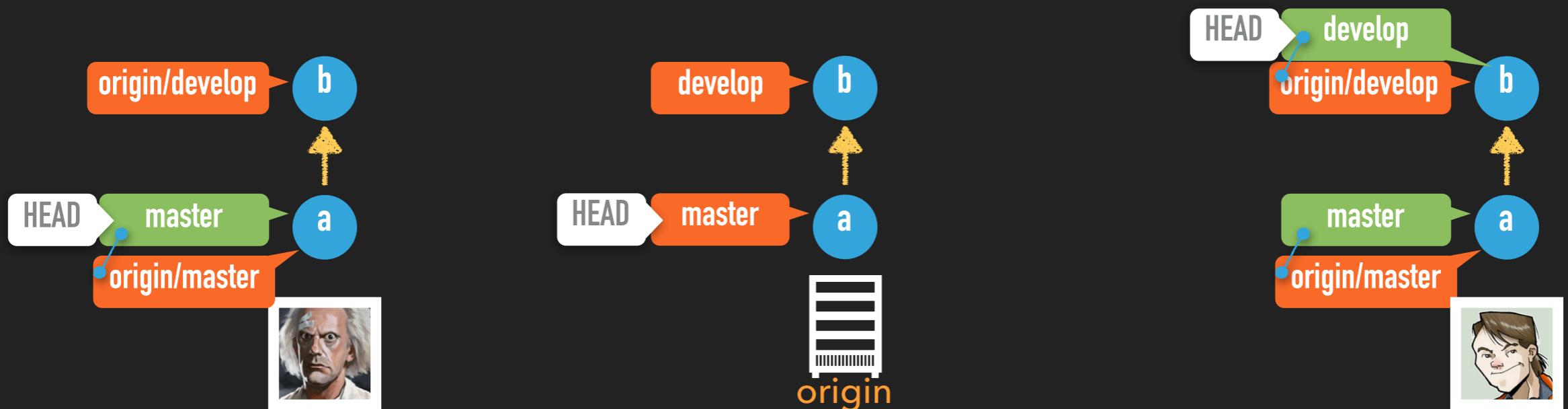
- ▶ Un clone du repository est créé dans path
- ▶ TOUS les commits sont clonés
- ▶ La position de chaque branche du serveur est notée en local origin/branchname
- ▶ Le HEAD se place sur la branche par défaut du serveur



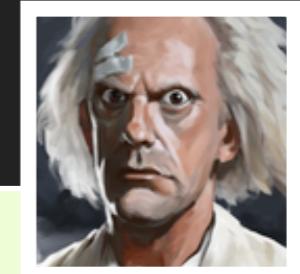
git checkout [ref]



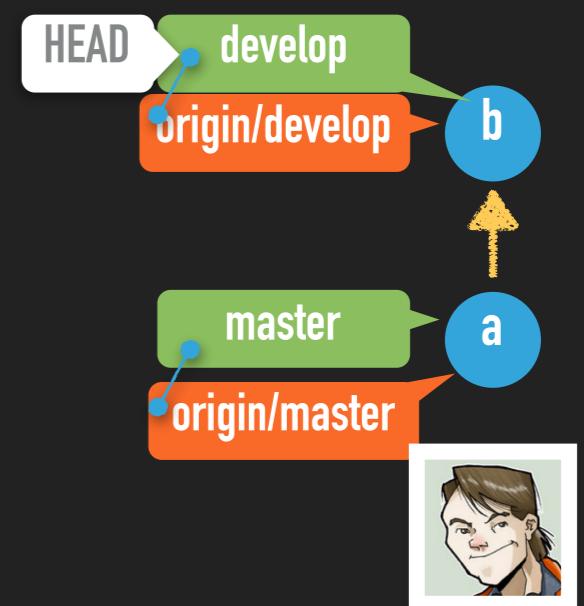
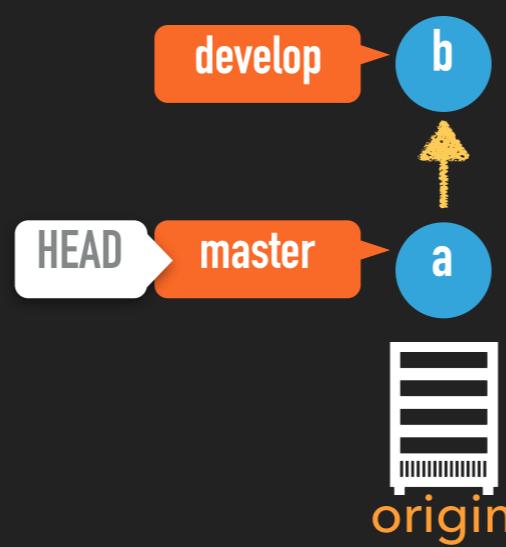
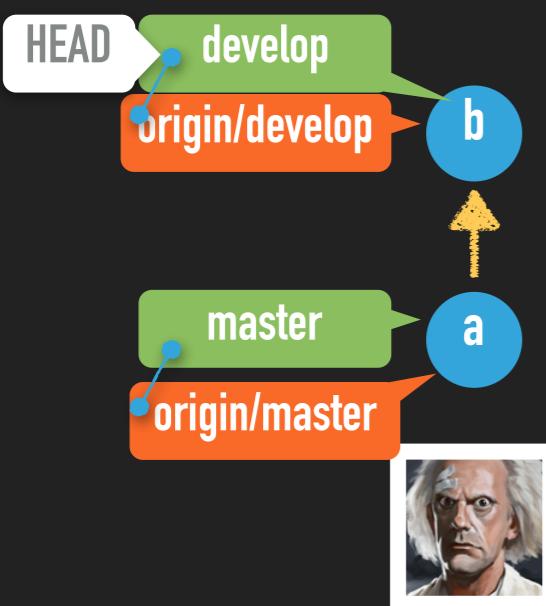
```
doc@labo$ git branch
* master
doc@labo$ git checkout develop
Branch develop set up to track remote branch develop from origin.
Switched to a new branch 'develop'
git branch
* develop
  master
```



git status

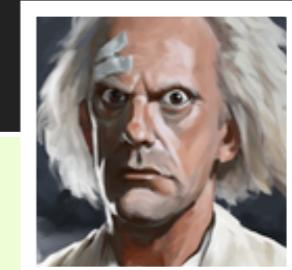


```
doc@labo$ echo "nouvelle ligne" >> README.md
doc@labo$ git add README.md
doc@labo$ git commit -m "Ajout d'un commit sur develop"
doc@labo$ git commit -a -m "Ajout d'un commit sur develop"
[develop 505fb26] Ajout d'un commit sur develop
 1 file changed, 1 insertion(+), 1 deletion(-)
```

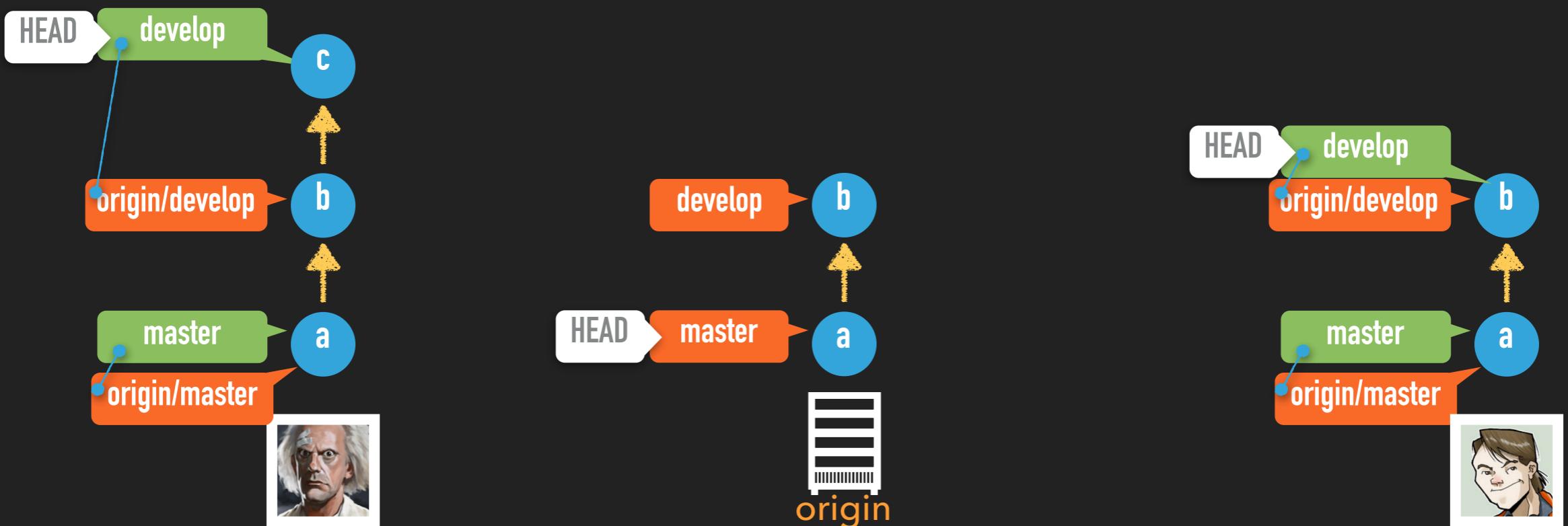


git status

```
doc@labo$ echo "nouvelle ligne" >> README.md
doc@labo$ git add README.md
doc@labo$ git commit -m "Ajout d'un commit sur develop"
[develop 505fb26] Ajout d'un commit sur develop
 1 file changed, 1 insertion(+), 1 deletion(-)
```



```
doc@labo$ git status
On branch develop
Your branch is ahead of 'origin/develop' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```



git push

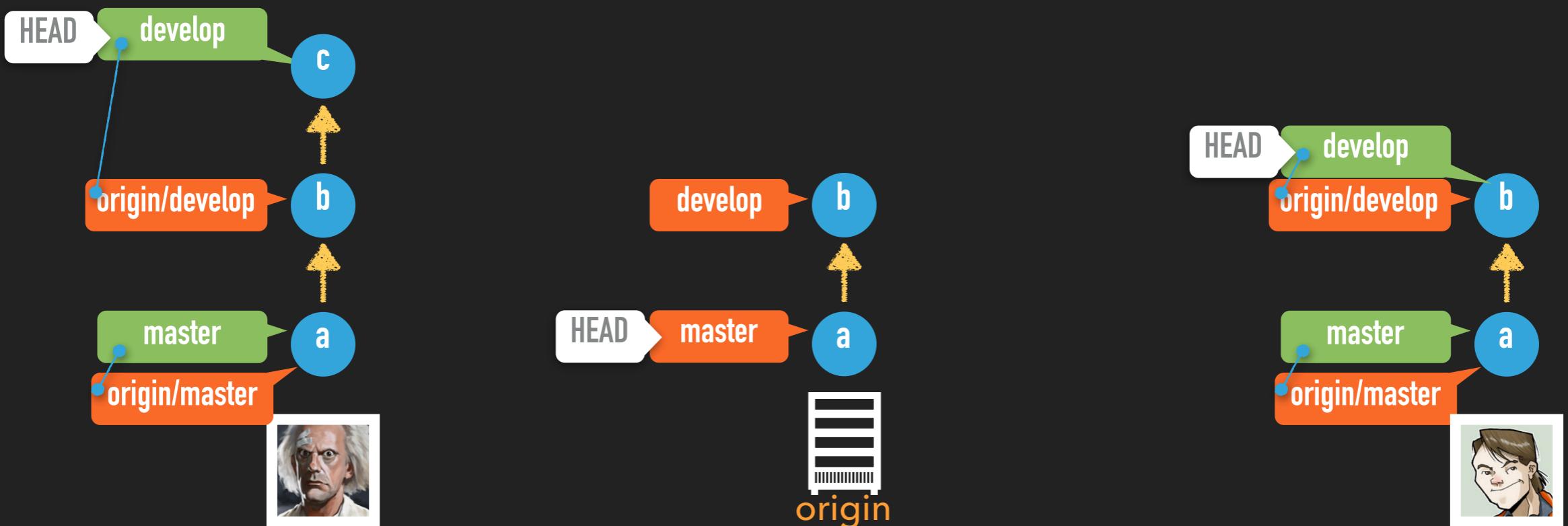
- ▶ Envoie la branche **courante**
- ▶ et TOUS les commits permettant d'y accéder
- ▶ sur la branche **suivie**
- ▶ sur le serveur



git push



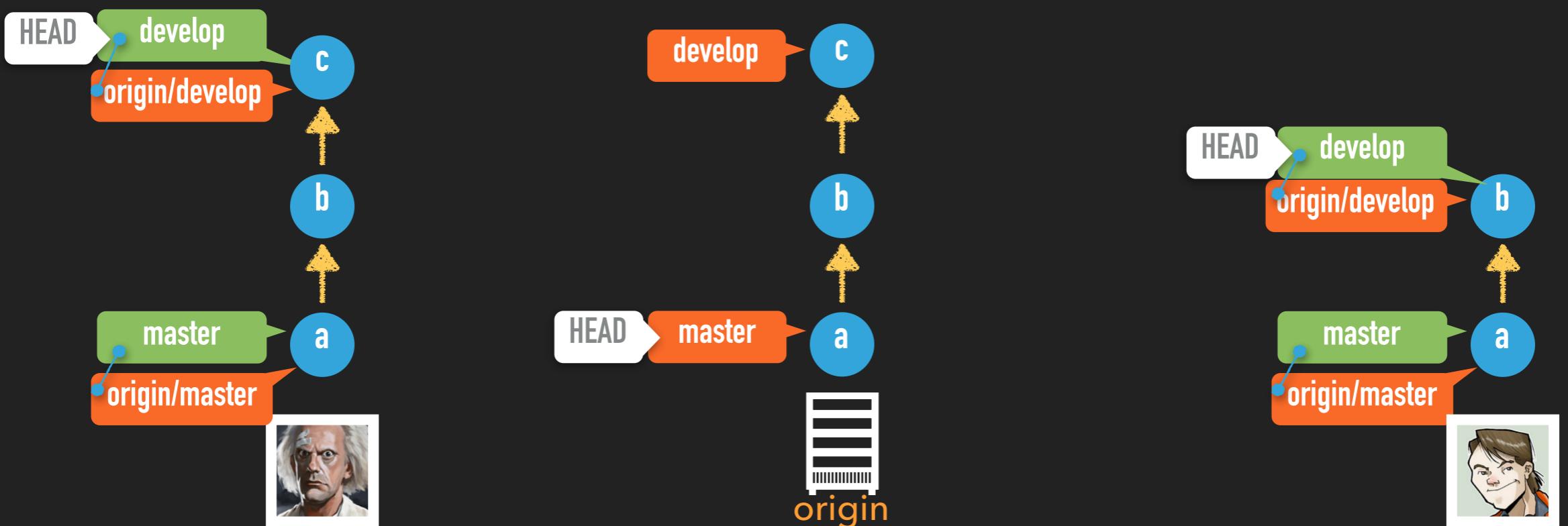
```
doc@labo$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 328 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To ssh://git@algec.iut-blagnac.fr:2200/git_training/basic.git
 b98d62d..505fb26  develop -> develop
```



git push



```
doc@labo$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 328 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To ssh://git@algec.iut-blagnac.fr:2200/git_training/basic.git
 b98d62d..505fb26 develop -> develop
```



```
git push -u                                origin nouvelle_branche  
git push --set-upstream origin nouvelle_branche
```

- ▶ Quand une branche n'a jamais été poussée
- ▶ Quand la branche n'existe pas sur le serveur
- ▶ Il faut préciser
 - ▶ le serveur de destination **origin**
 - ▶ son nom **nouvelle_branche**
 - ▶ et établir le lien de suivi **-u**
- ▶ Et après, on peut faire des **git push**

git push --follow-tags

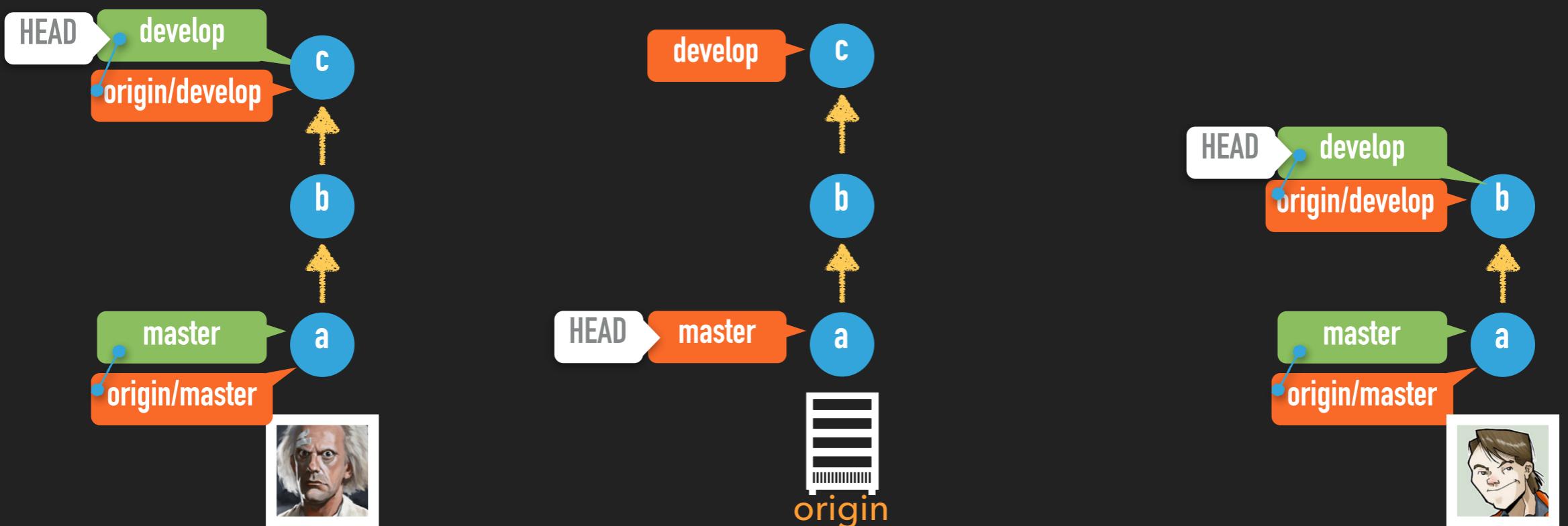
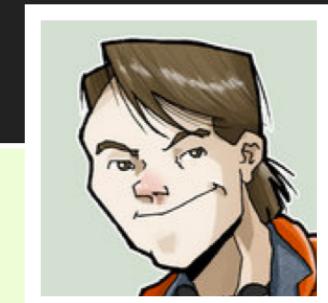
- ▶ Permet d'envoyer les tags présents sur la branche sur le serveur

git fetch

- ▶ Récupère TOUS les commits du serveur
- ▶ et les références, branches et tags
- ▶ met à jour les branches **origin/xxxxx**
- ▶ ne mène JAMAIS à un conflit

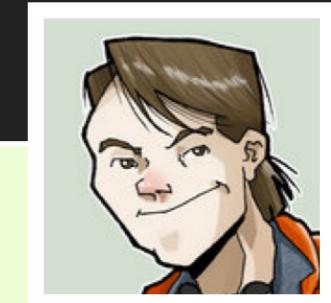
git fetch

```
marty@lycee$ git fetch
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
Unpacking objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
From ssh://algec.iut-blagnac.fr:2200/git_training/basic
 476382d..931e7cd develop    -> origin/develop
```

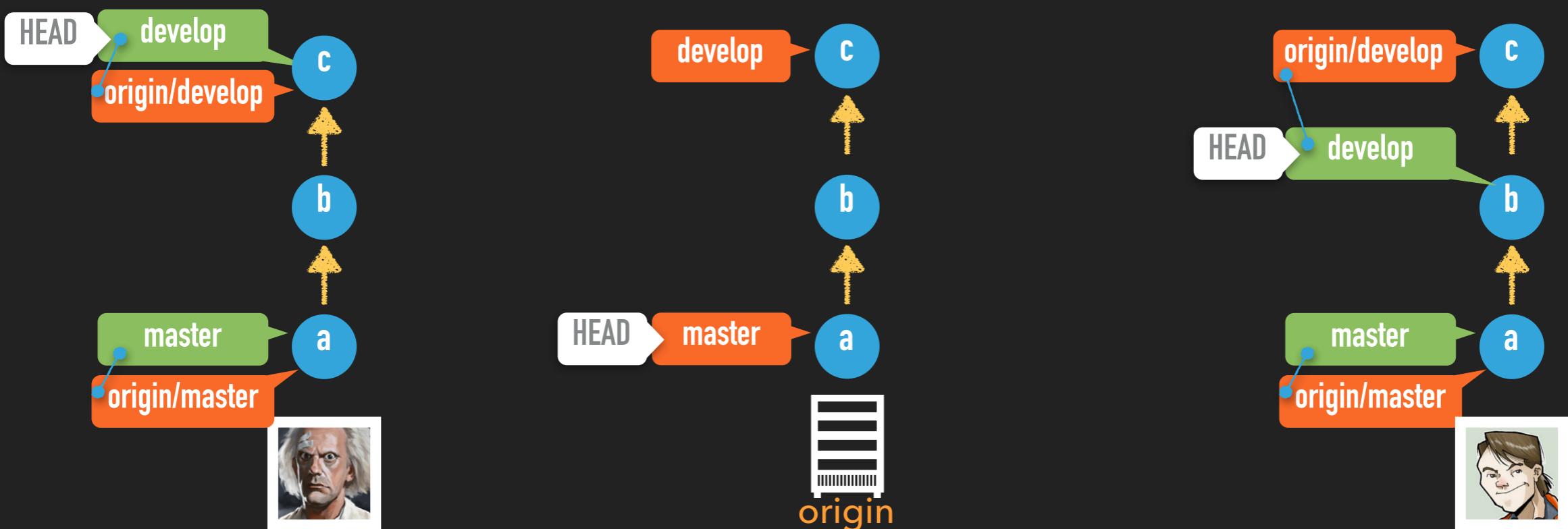


git fetch

```
marty@lycee$ git fetch
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
Unpacking objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
From ssh://algec.iut-blagnac.fr:2200/git_training/basic
  476382d..931e7cd develop    -> origin/develop
```

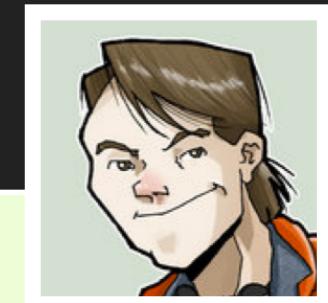


```
marty@lycee$ git status
On branch develop
Your branch is behind 'origin/develop' by 1 commit, and can be fast-forwarded.
nothing to commit, working directory clean
```

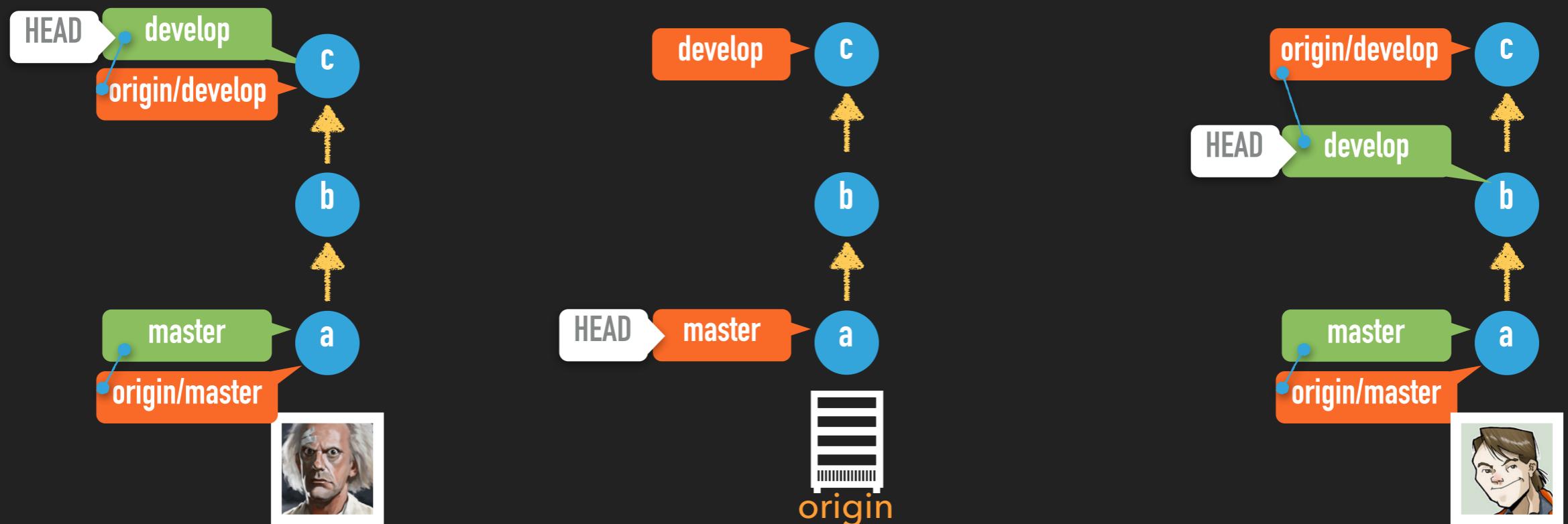


git merge origin/develop

```
marty@lycee$ git status
On branch develop
Your branch is behind 'origin/develop' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
nothing to commit, working directory clean
```

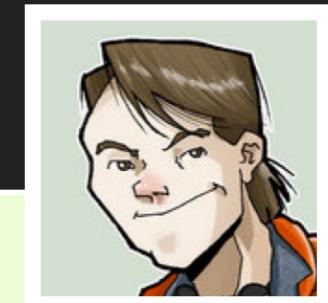


```
marty@lycee$ git merge origin/develop
Updating 476382d..931e7cd
Fast-forward
 CHangelog.md | 1 +
 1 file changed, 1 insertion(+)
```

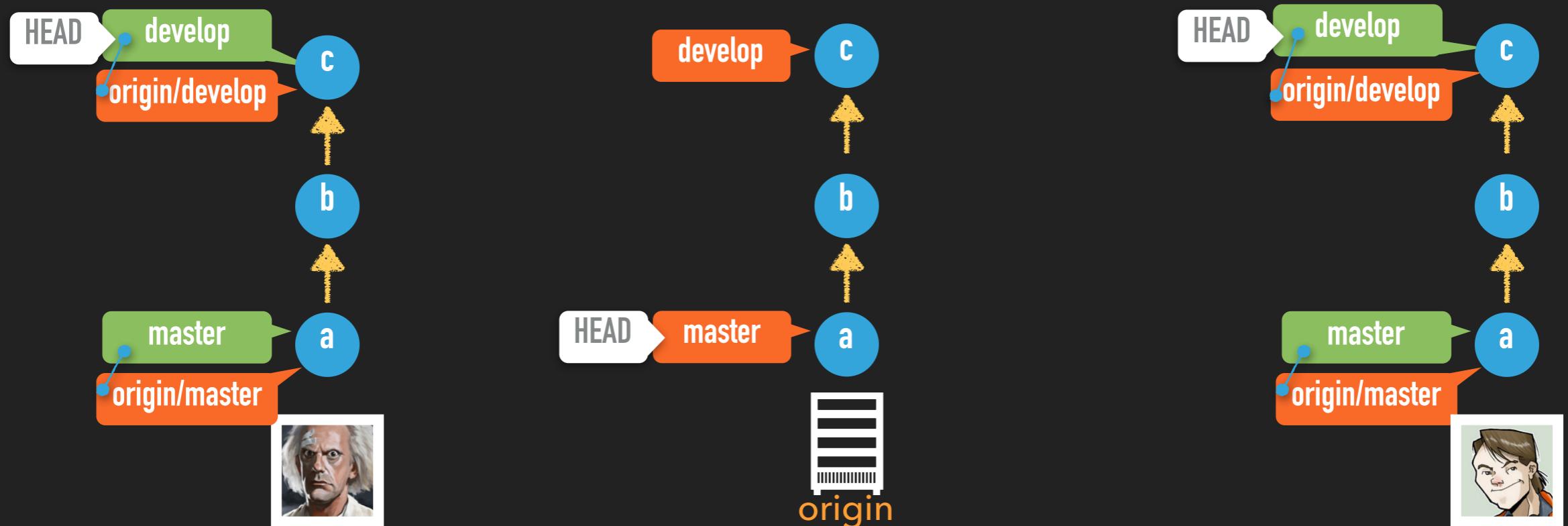


git merge origin/develop

```
marty@lycee$ git status
On branch develop
Your branch is behind 'origin/develop' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
nothing to commit, working directory clean
```



```
marty@lycee$ git merge origin/develop
Updating 476382d..931e7cd
Fast-forward
 CHangelog.md | 1 +
 1 file changed, 1 insertion(+)
```

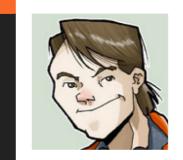
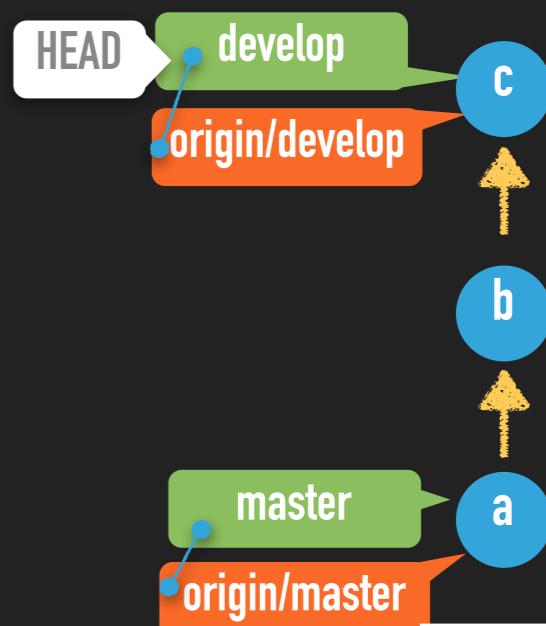
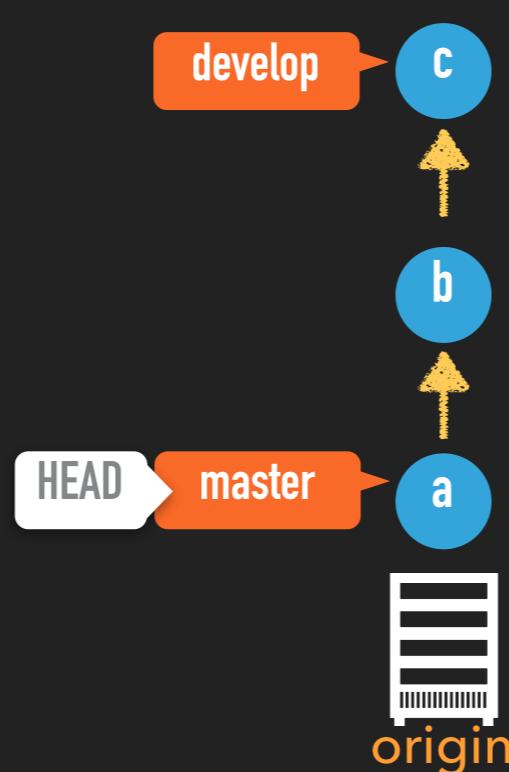


git fetch + git merge = git pull

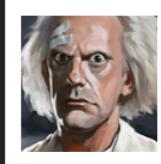
- ▶ Plutôt que d'enchaîner fetch + merge
- ▶ on peut faire directement pull
- ▶ Comme le pull contient une action de merge
- ▶ il peut conduire à un CONFLIT
- ▶ Ne merge que la branche courante

git fetch + git merge = git pull

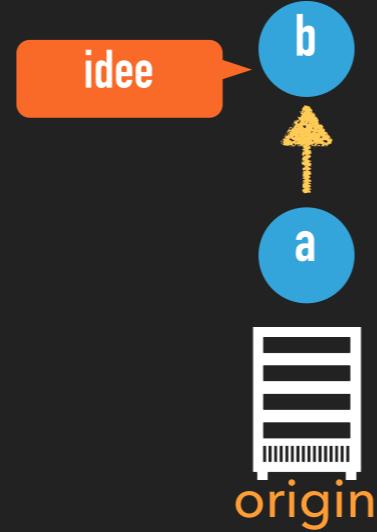
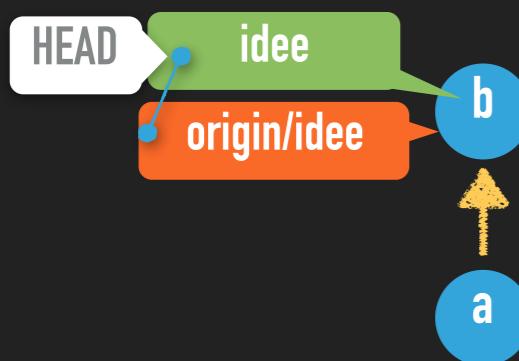
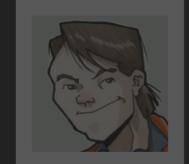
```
marty@lycee$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From ssh://algec.iut-blagnac.fr:2200/git_training/basic
  931e7cd..3009b83  develop    -> origin/develop
Updating 931e7cd..3009b83
Fast-forward
  CHANGELOG.md | 1 +
  1 file changed, 1 insertion(+)
```



BOSSEZ À DEUX SUR UNE BRANCHE



git commit



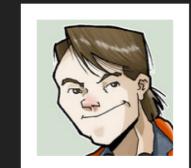
origin



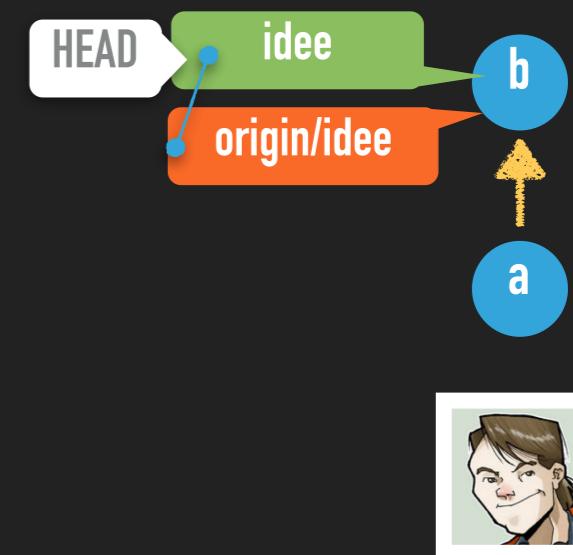
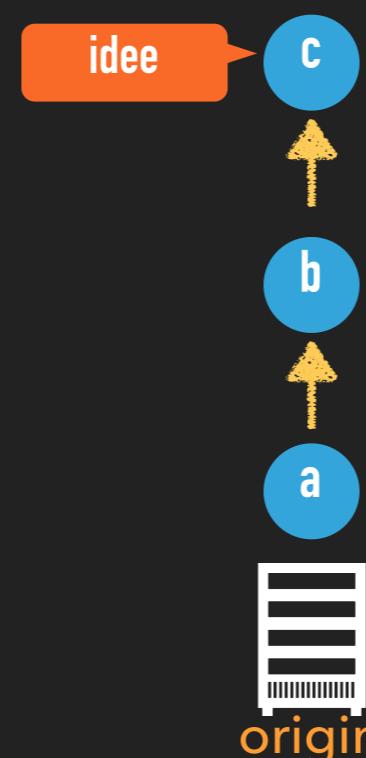
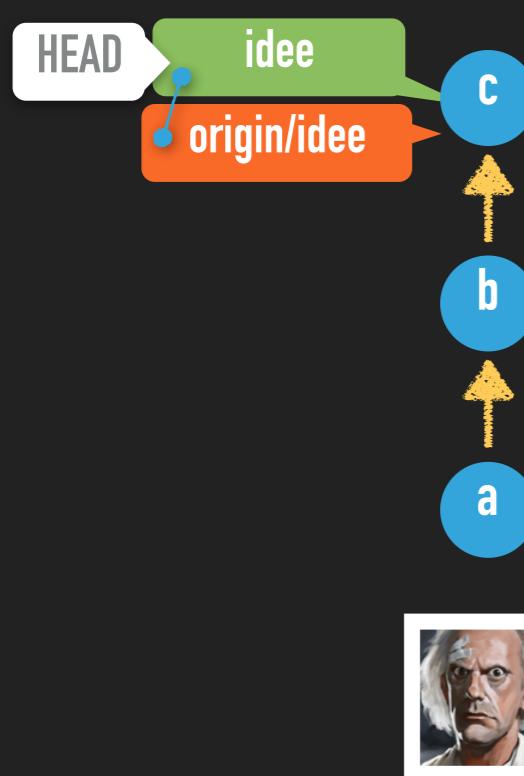
BOSSEZ À DEUX SUR UNE BRANCHE



git commit
git push



git commit



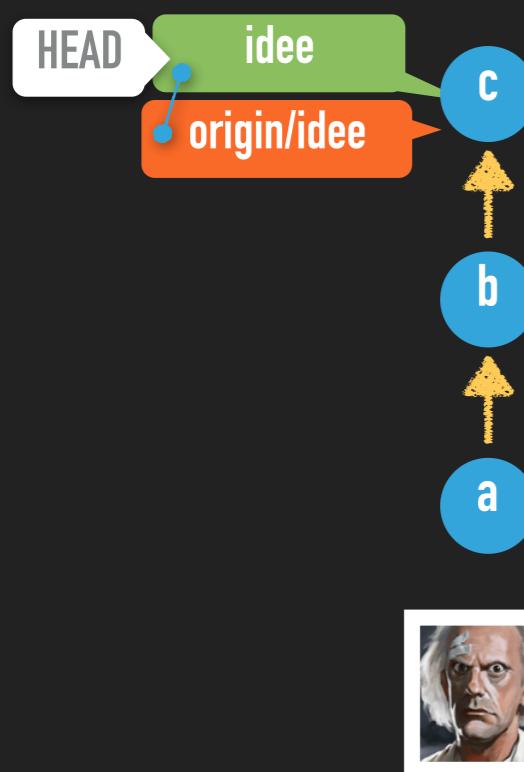
BOSSEZ À DEUX SUR UNE BRANCHE



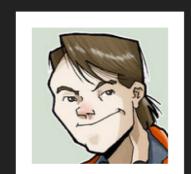
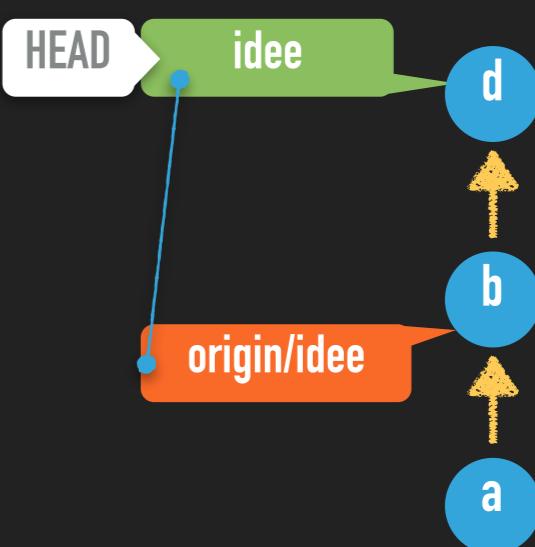
git commit
git push



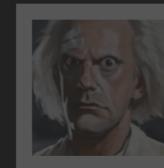
git commit
git push
push rejected



$c \neq b$
CONFLIT



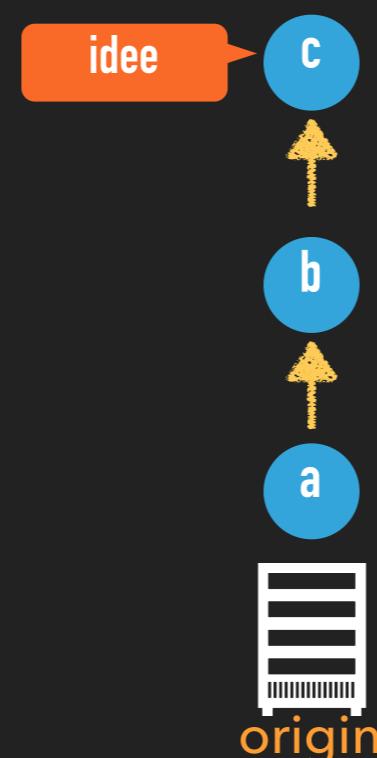
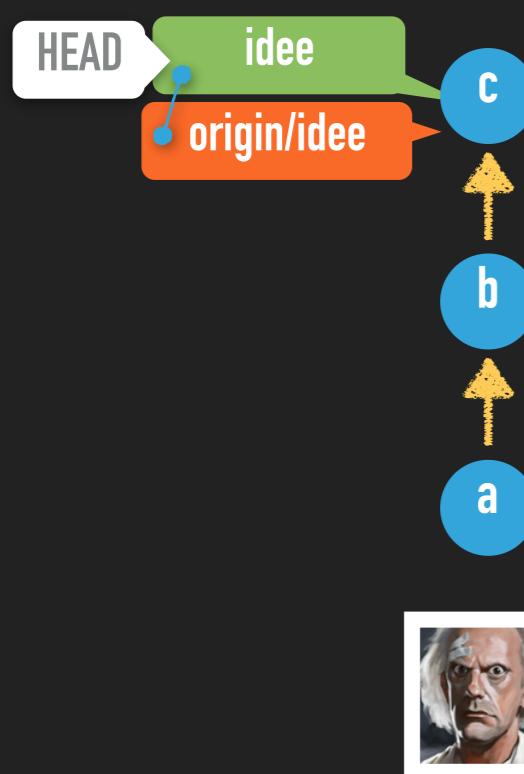
BOSSER À DEUX SUR UNE BRANCHE



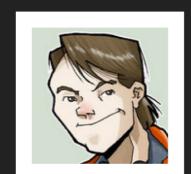
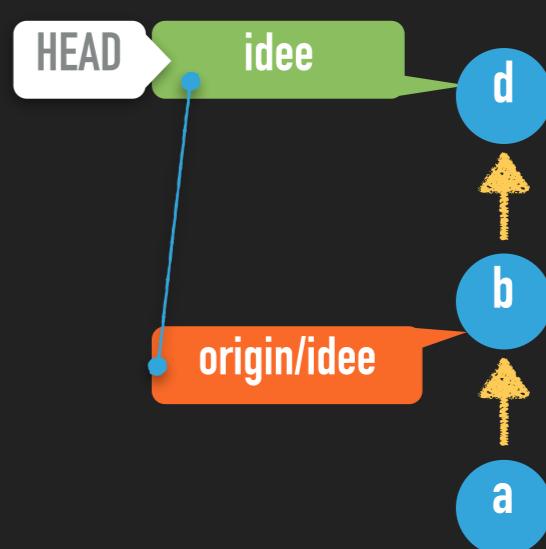
git commit
git push



git commit
git push
push rejected : pull first
git pull



$c \neq b$
CONFLIT



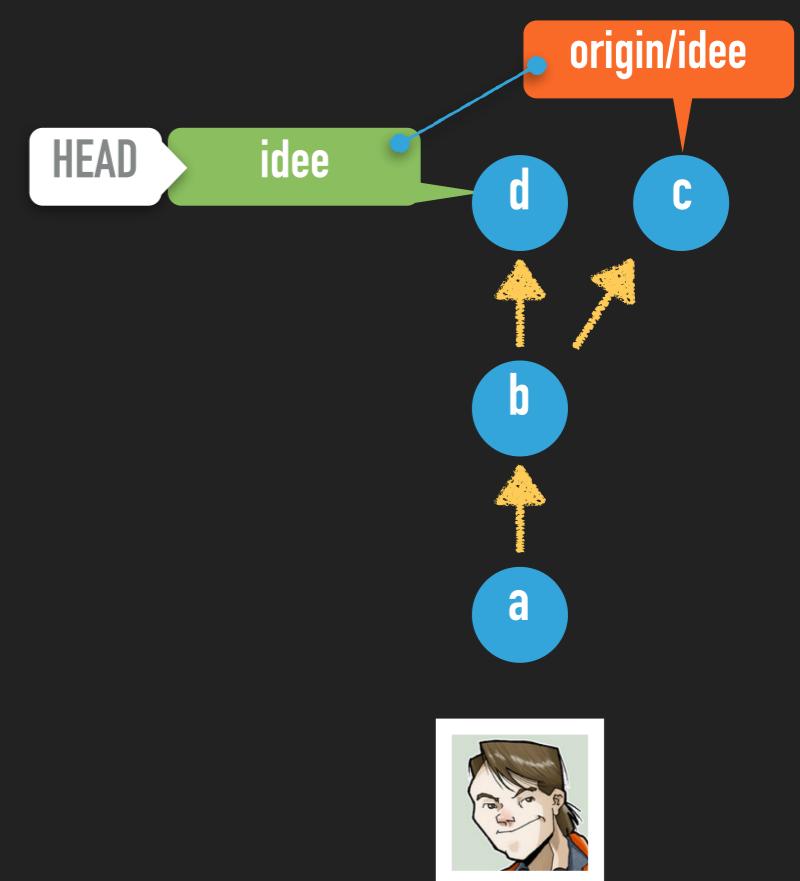
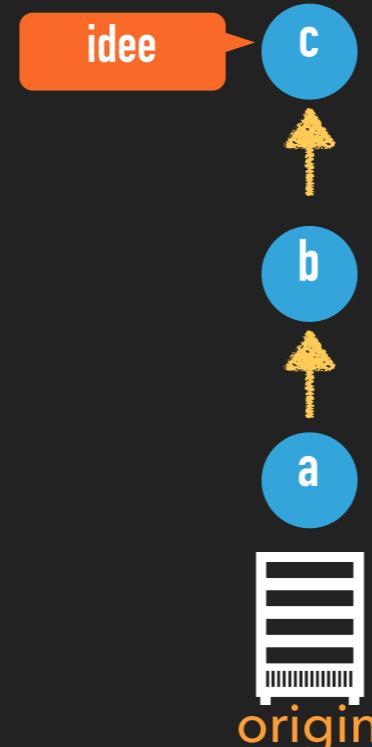
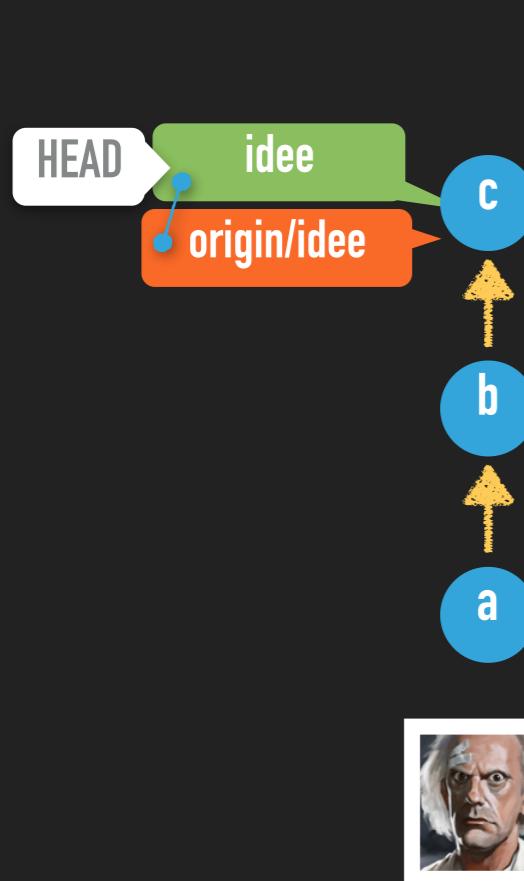
BOSSER À DEUX SUR UNE BRANCHE



git commit
git push



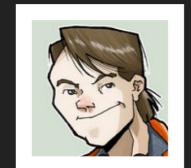
git commit
git push
push rejected : pull first
git pull fetch phase



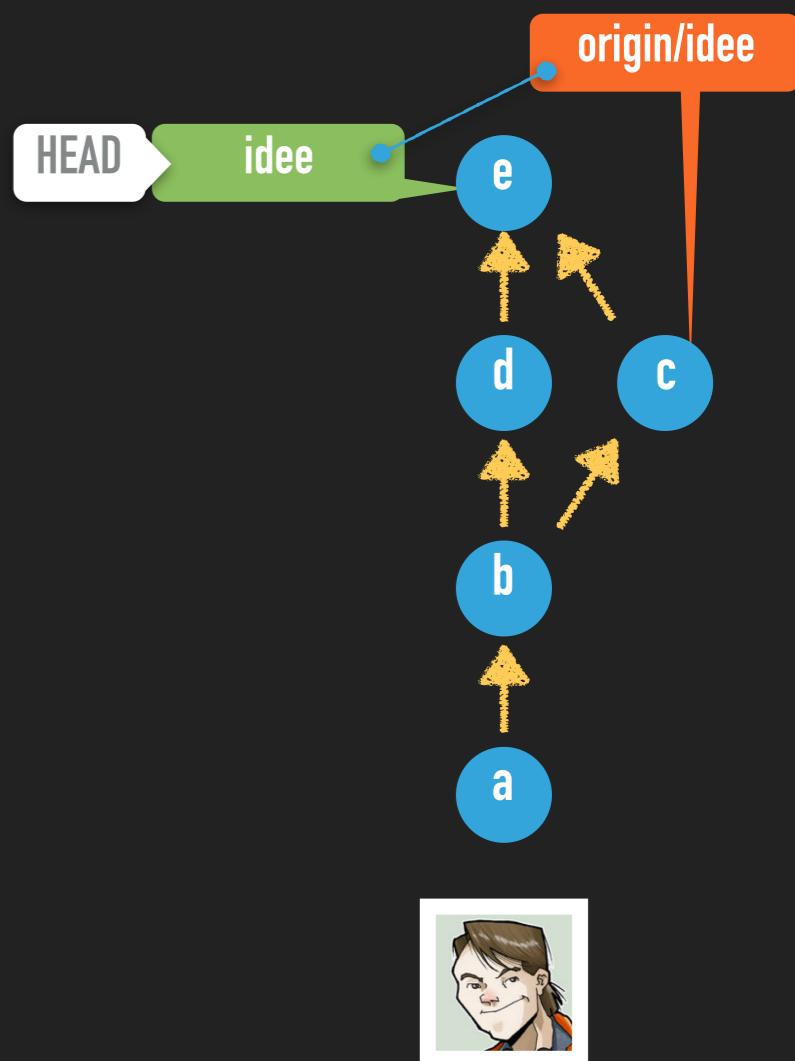
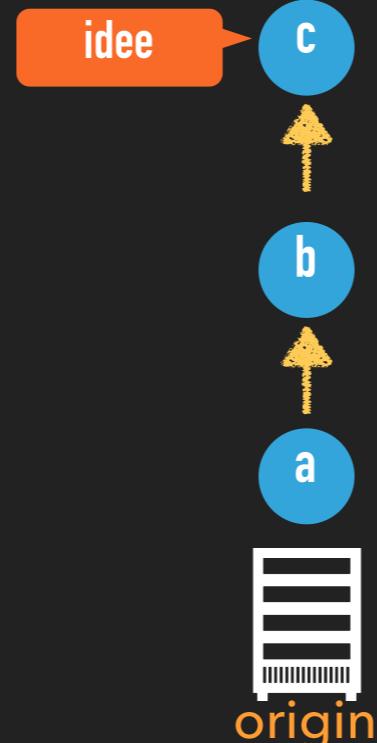
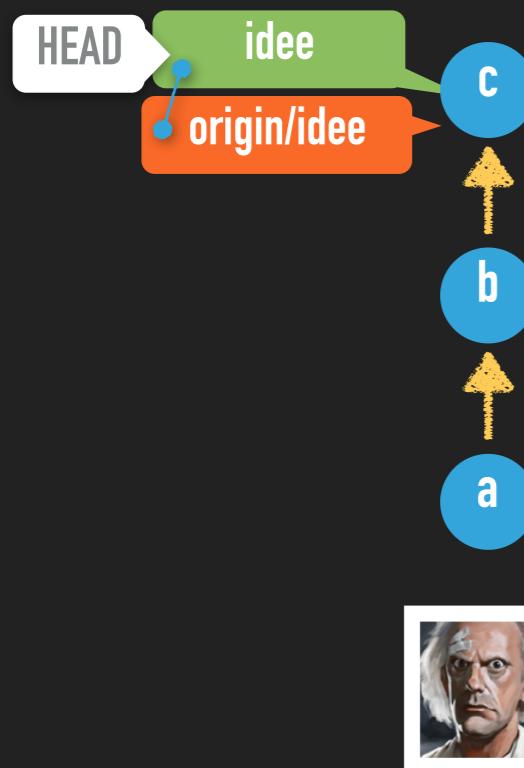
BOSSER À DEUX SUR UNE BRANCHE



git commit
git push



git commit
git push
push rejected : pull first
git pull merge phase
git push



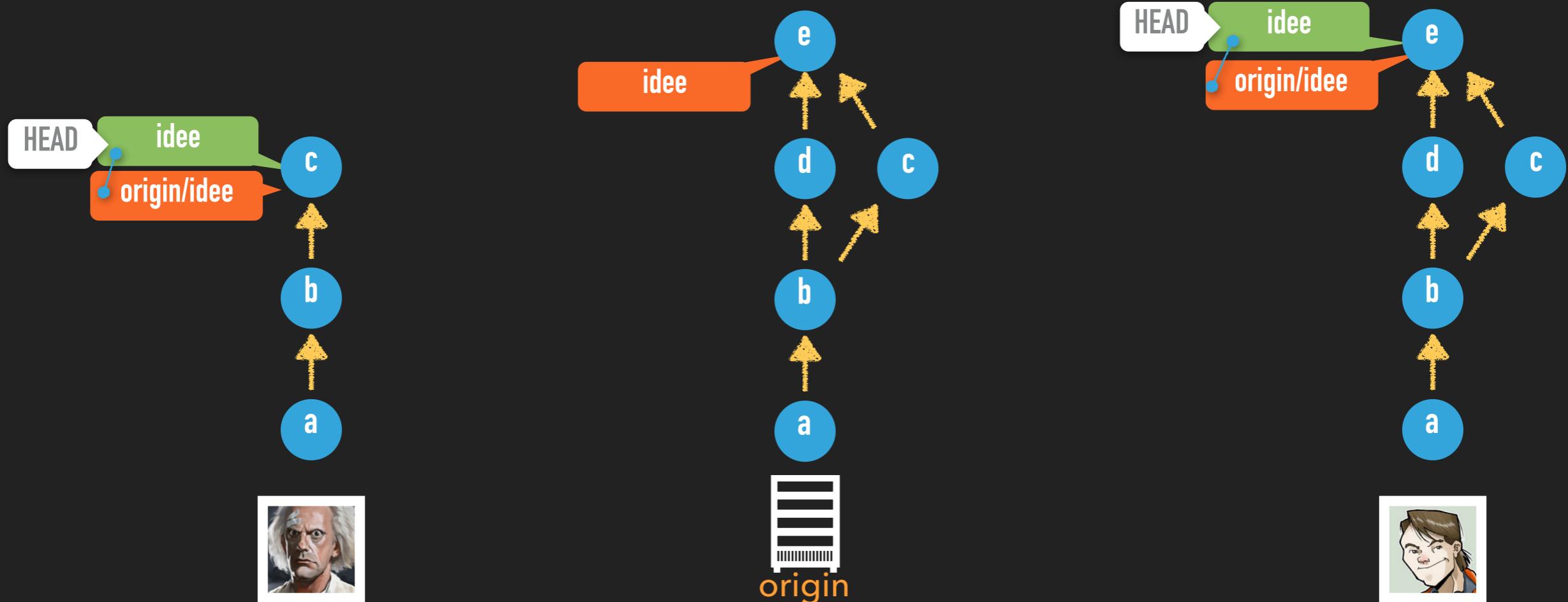
BOSSER À DEUX SUR UNE BRANCHE



git commit
git push
git pull



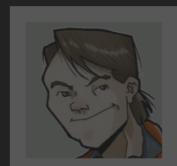
git commit
git push
push rejected : pull first
git pull merge phase
git push



BOSSEZ À DEUX SUR UNE BRANCHE



```
git commit  
git push  
git pull
```

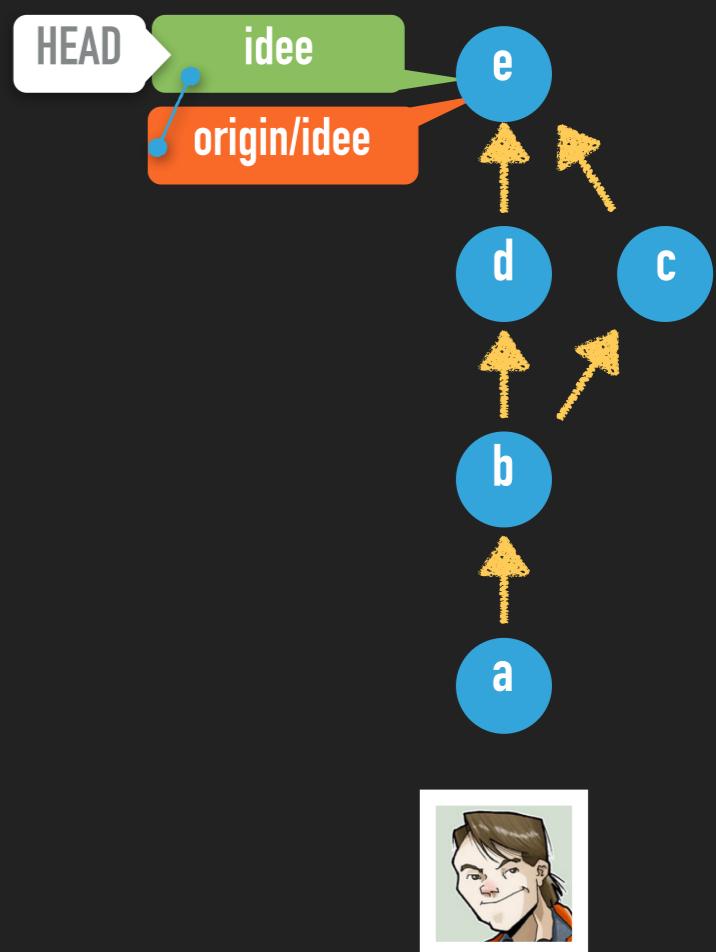
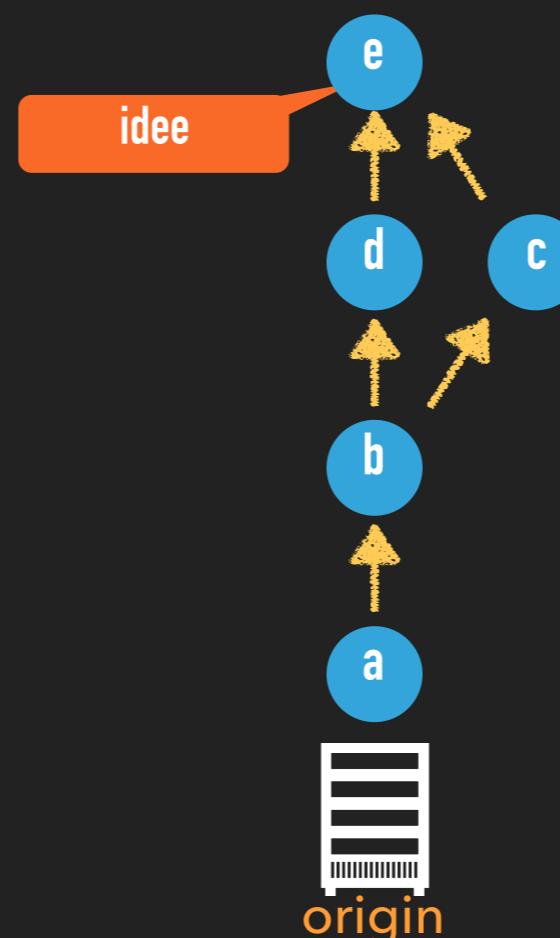
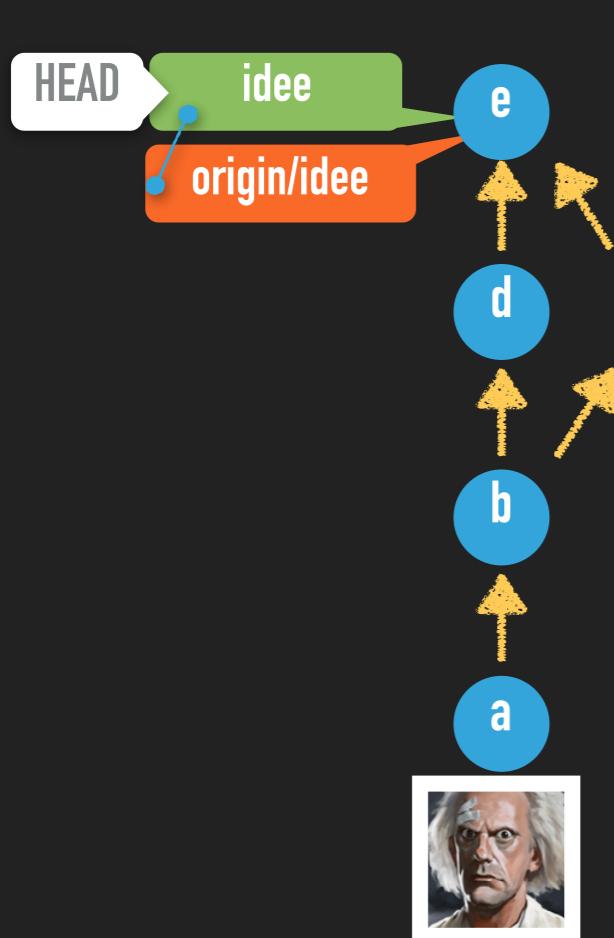


git commit
git push

push rejected : pull first

git pull merge phase

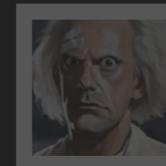
git push



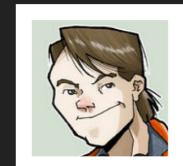
git pull --rebase

- ▶ remplace le merge de la phase de merge par un rebase
- ▶ Permet de garder un historique linéaire
- ▶ Peut mener à un conflit
- ▶ à utiliser si on est à deux sur une même branche

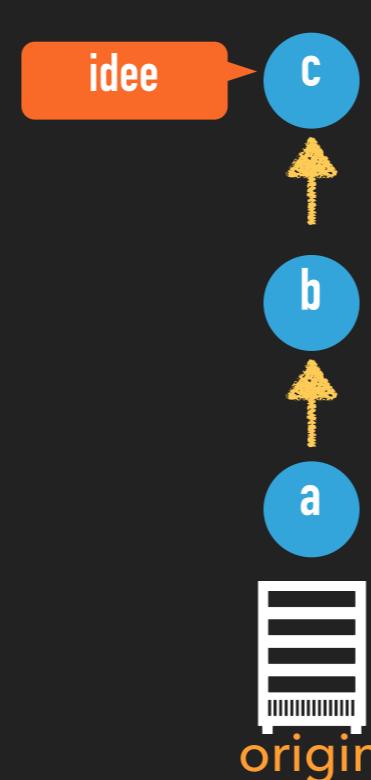
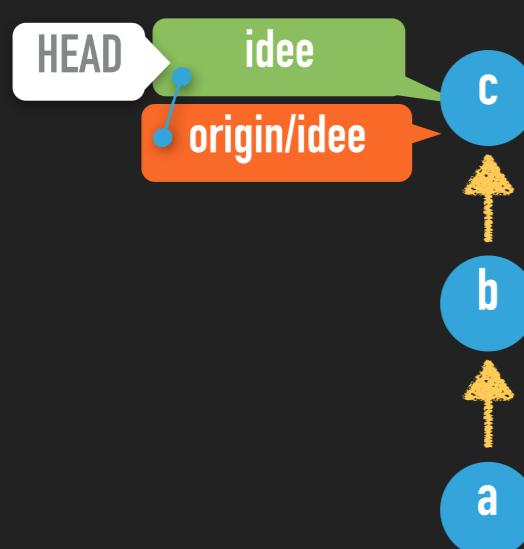
GIT PULL AVEC REBASE



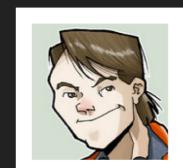
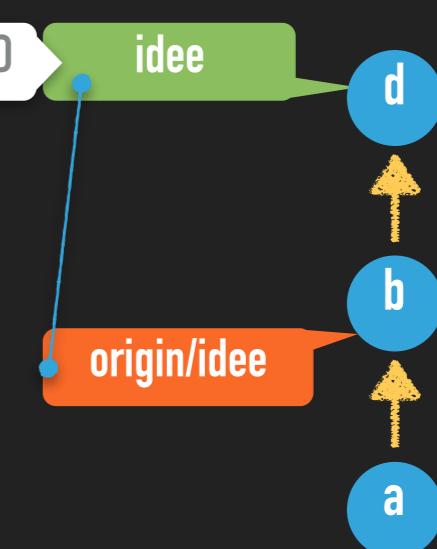
git commit
git push



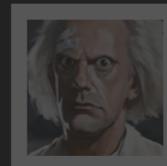
git commit
git push
push rejected : pull first
git pull --rebase



$c \neq b$
CONFLIT



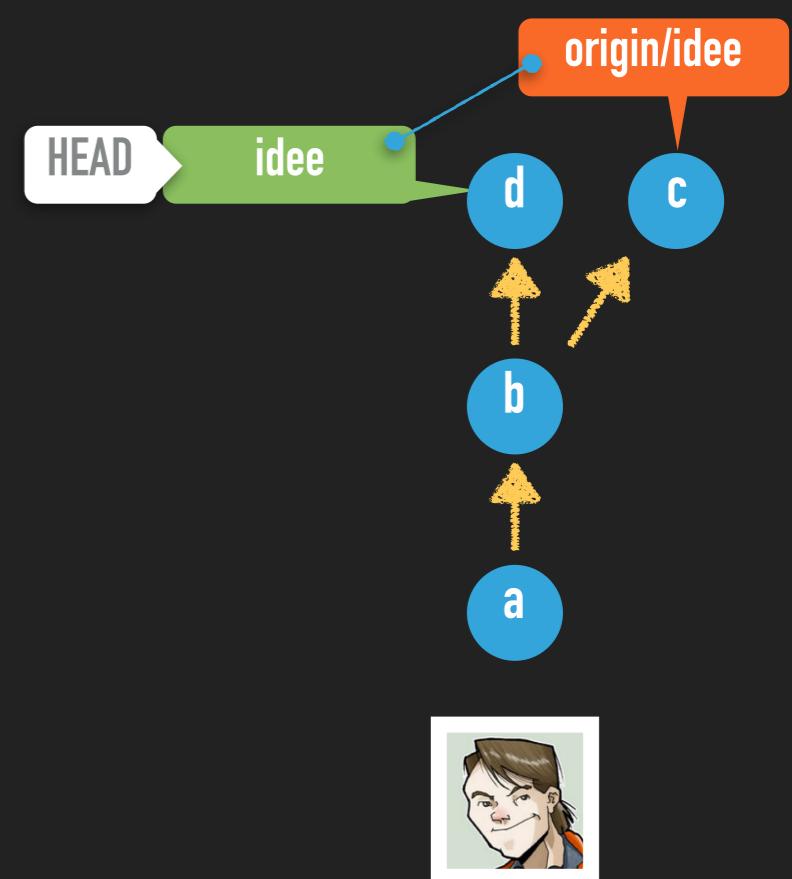
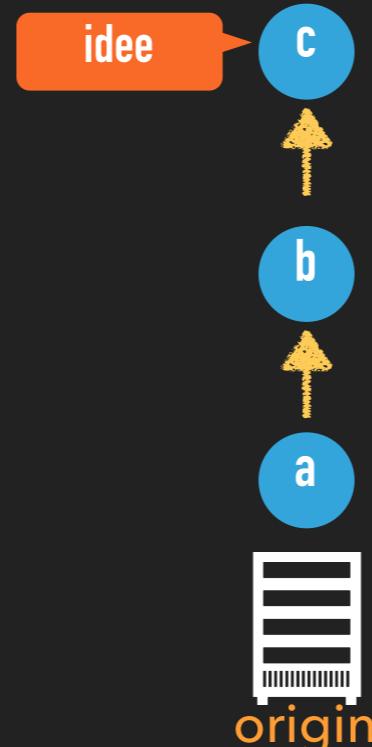
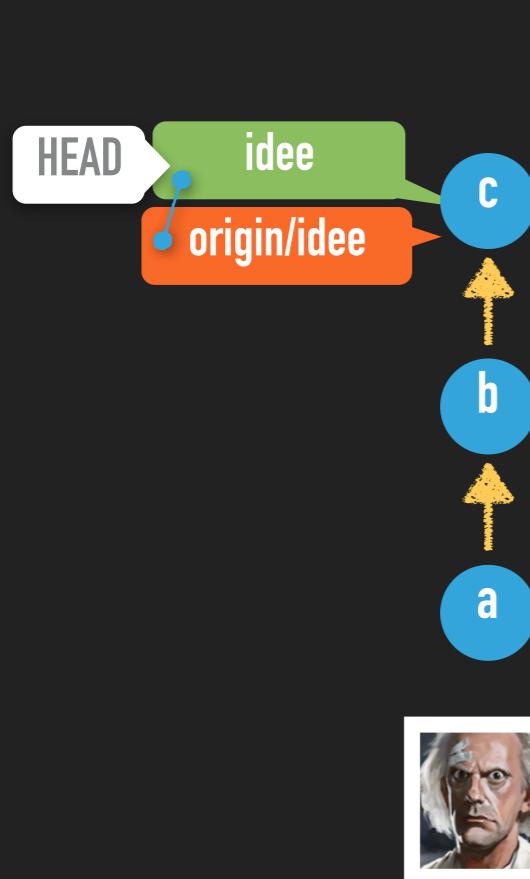
GIT PULL AVEC REBASE



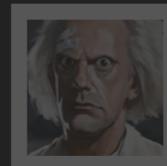
git commit
git push



git commit
git push
push rejected : pull first
git pull --rebase **fetch phase**



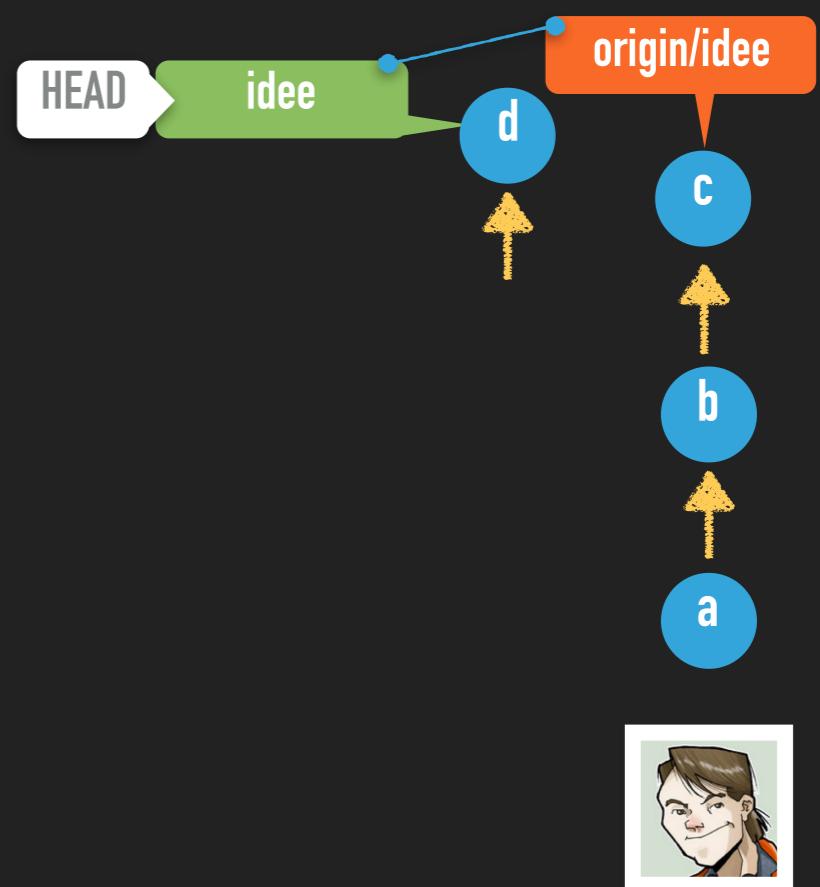
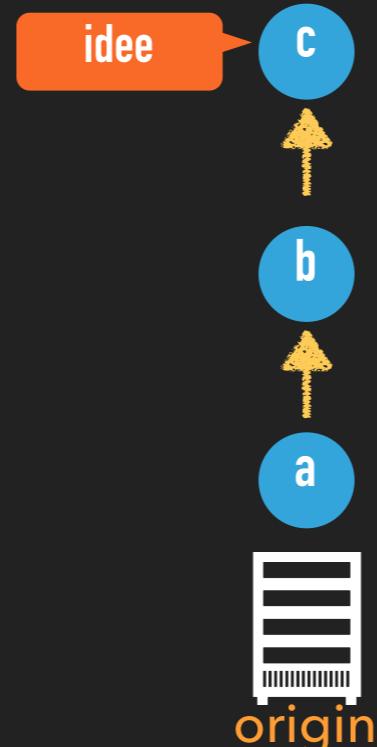
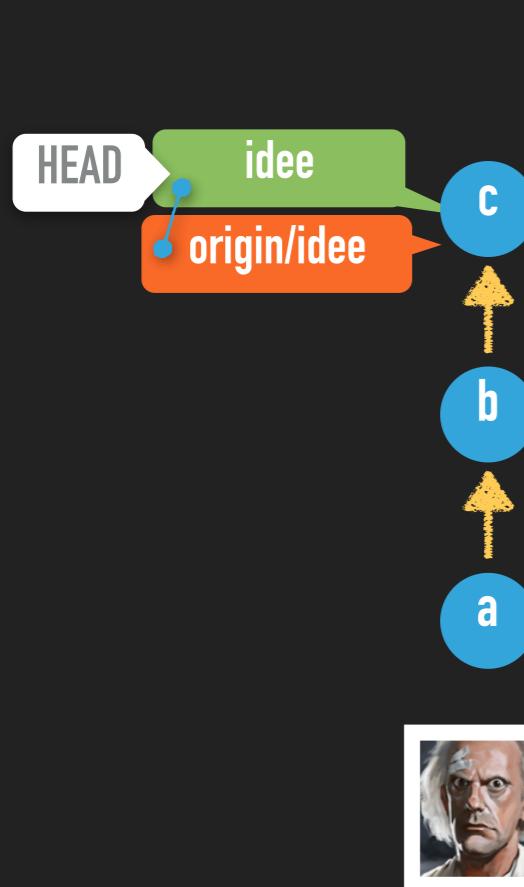
GIT PULL AVEC REBASE



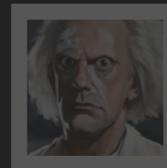
git commit
git push



git commit
git push
push rejected : pull first
git pull --rebase **rebase phase**



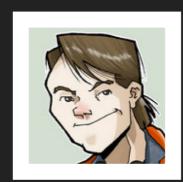
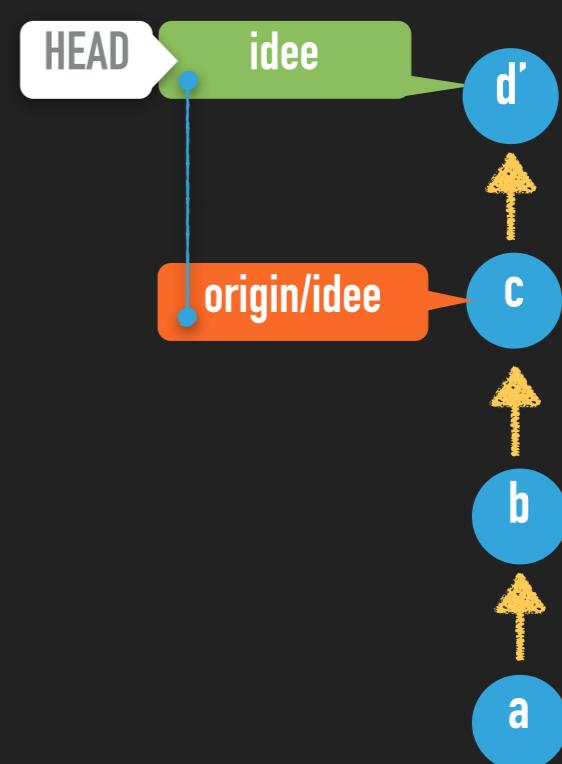
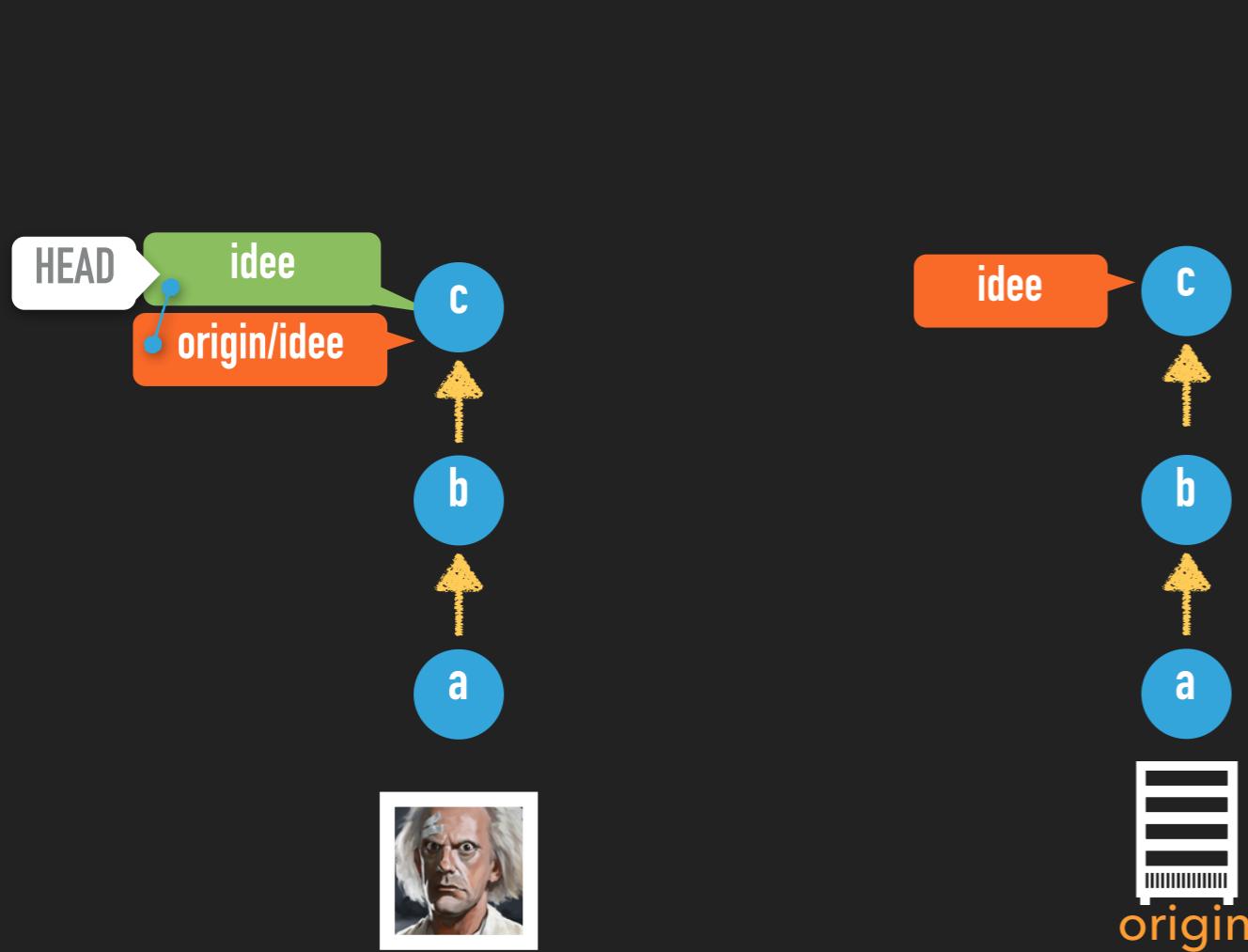
GIT PULL AVEC REBASE



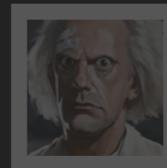
git commit
git push



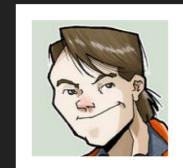
git commit
git push
push rejected : pull first
git pull --rebase **rebase phase**
git push



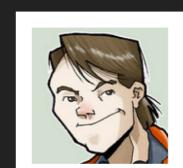
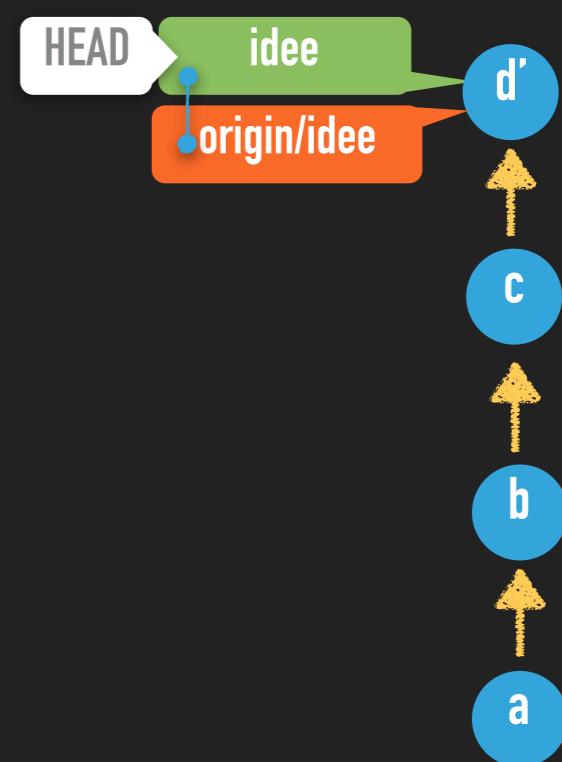
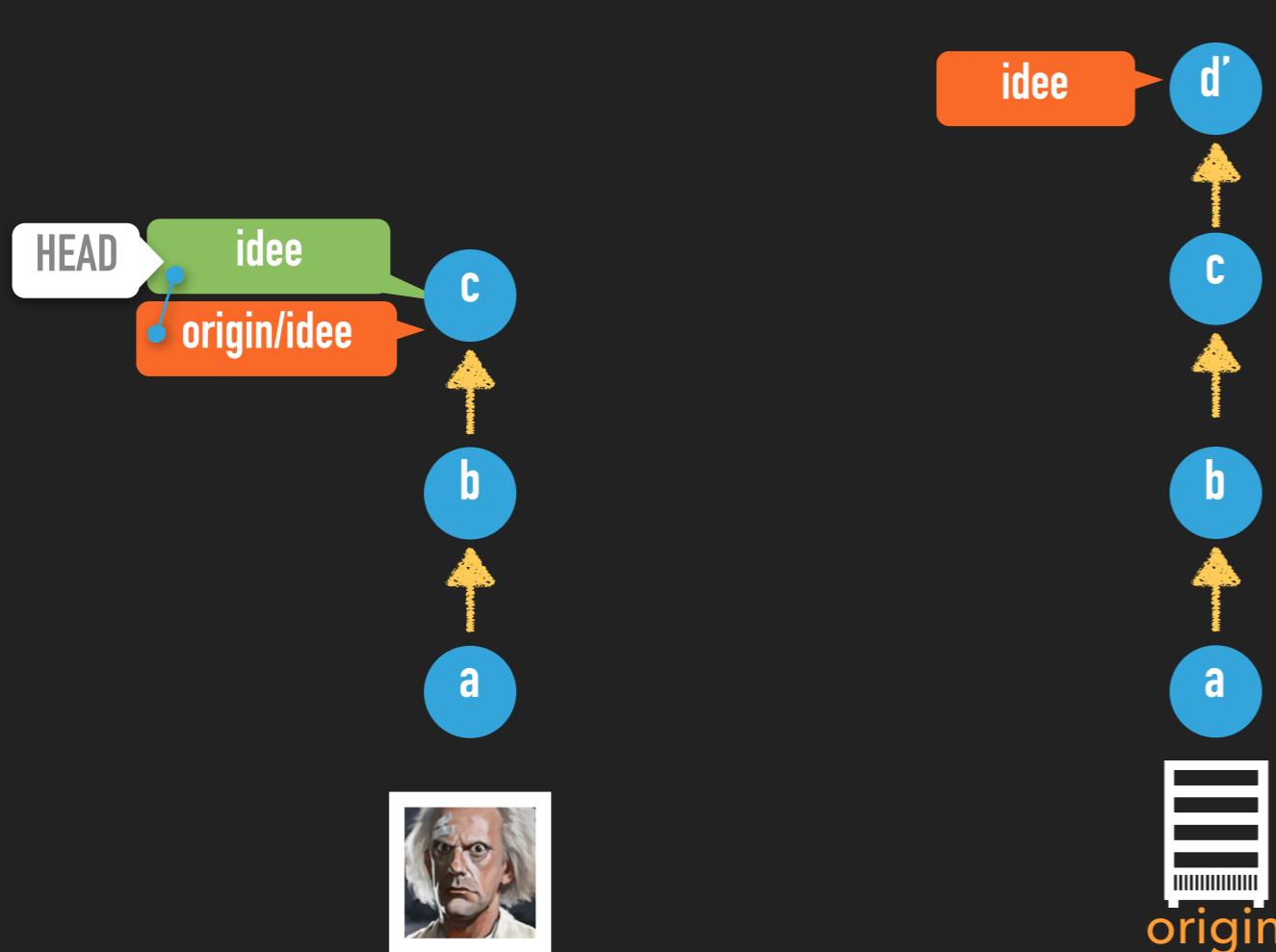
GIT PULL AVEC REBASE



git commit
git push



git commit
git push
push rejected : pull first
git pull --rebase **rebase phase**
git push



CLONE/PUSH/PULL

- ▶ Cloner un repo depuis une url:

```
git clone url
```

- ▶ Envoyer uniquement la branche courante feature-name:

```
git push -u origin feature-name
```

```
git push
```

- ▶ Récupérer toutes les modifications (sans danger):

```
git fetch
```

- ▶ Récupérer toutes les modifications ET merger la branche courante (risque de **conflit**):

```
git pull
```



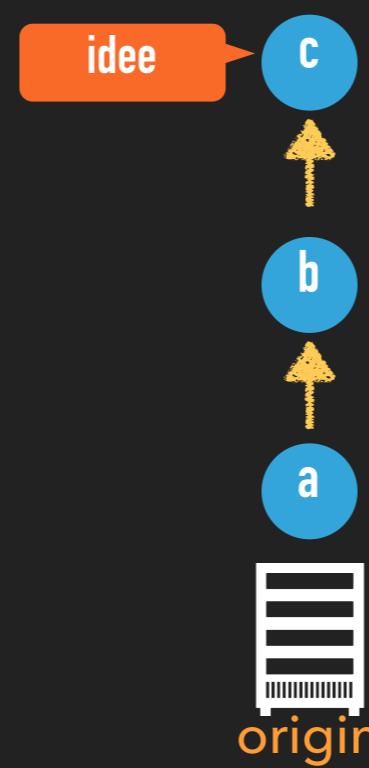
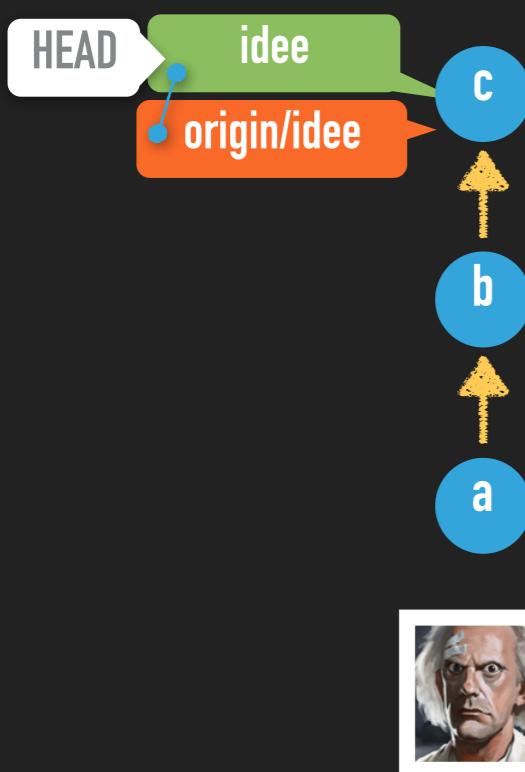
FAUT RÉFLÉCHIR, MCFLY, FAUT RÉFLÉCHIR !

RE-PUSH D'UN RESET

PUSHER UN RETOUR EN ARRIÈRE : DANGER ?



git reset --hard HEAD^



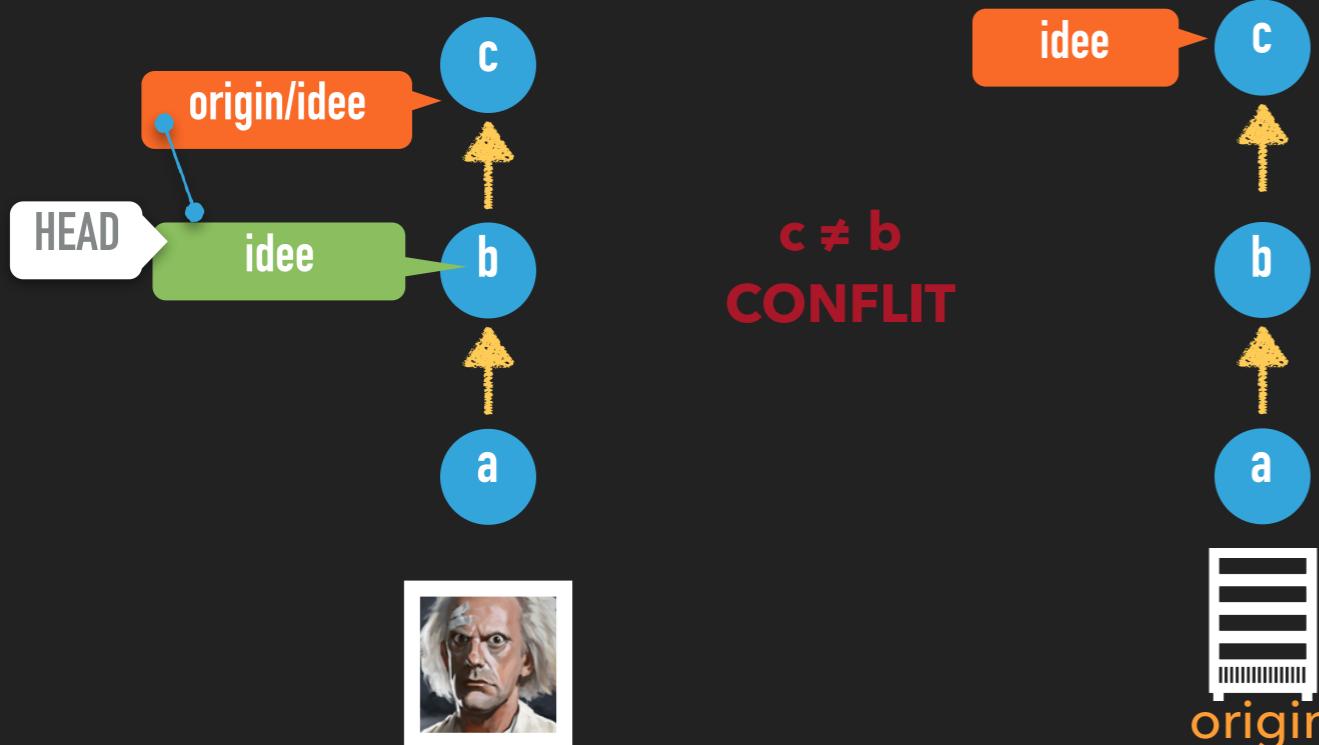
PUSHER UN RETOUR EN ARRIÈRE : DANGER ?



git reset --hard HEAD^
git push **push rejected : pull first**

```
doc@labo$ git push
To ssh://git@algec.iut-blagnac.fr:2200/git_training
 ! [rejected]      develop -> develop (non-fast-forward)
error: failed to push some refs to 'ssh://git@algec.iut-blagnac.fr:2200/git_training/basic.git'
hint: Updates were rejected because the tip of your local branch is behind
hint: its remote counterpart. Integrate the remote changes before pushing again.
hint: 'git pull ...' before pushing again.
hint: See the 'Note about fast-forwards'
```

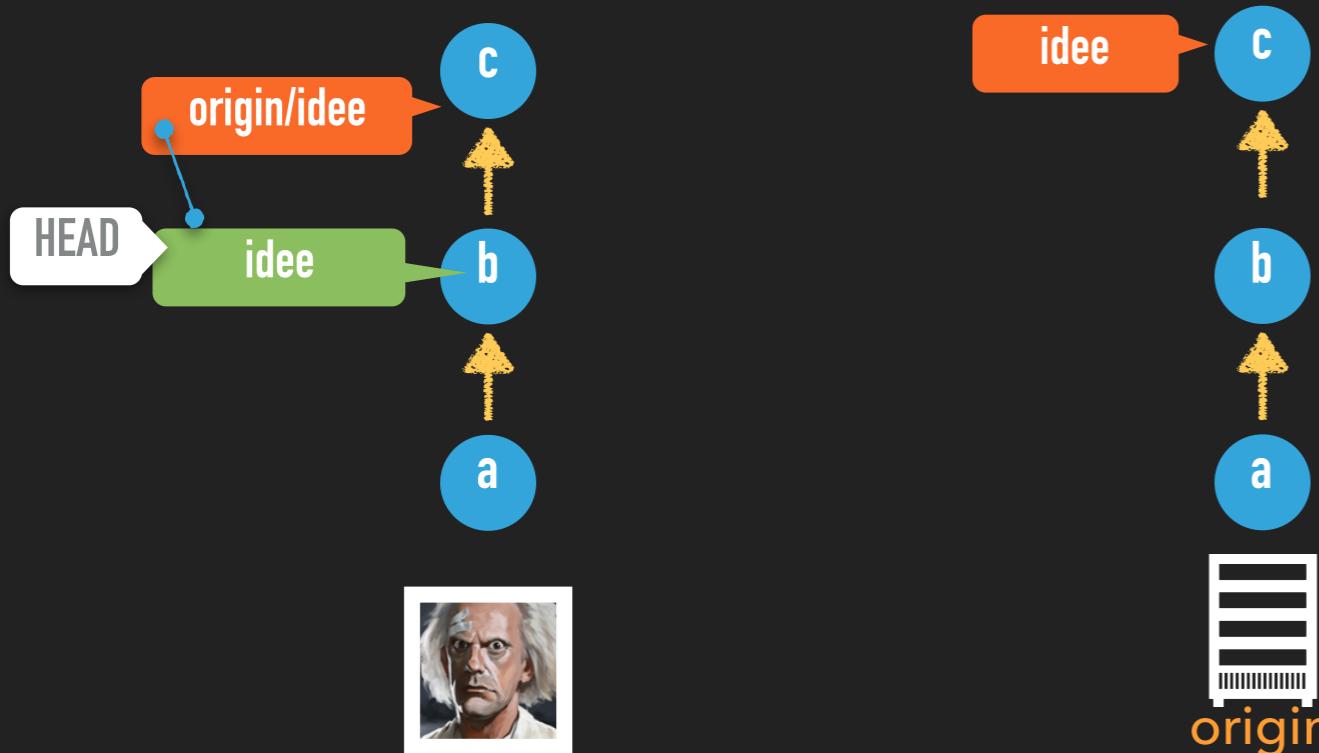
NE
PAS
FAIRE
PULL !



git push -f



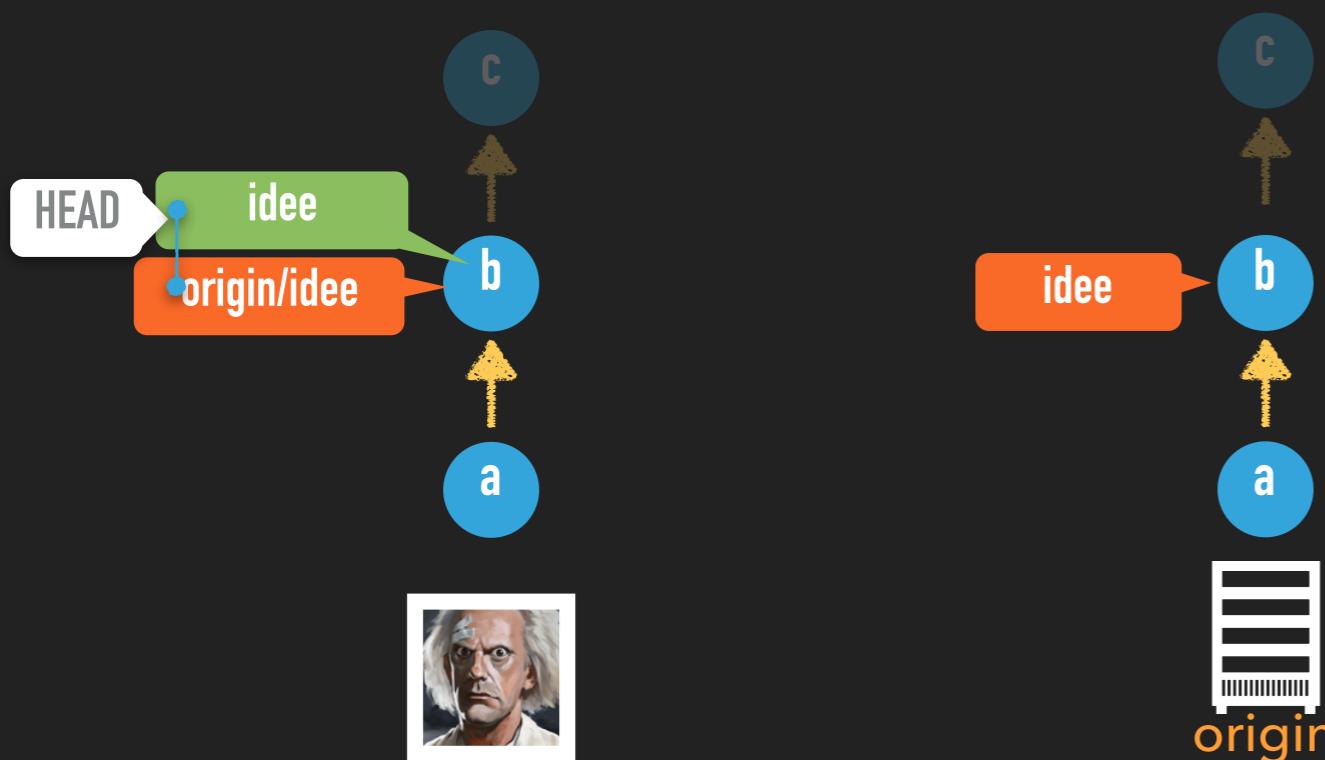
```
git reset --hard HEAD^  
git push -f
```



git push -f



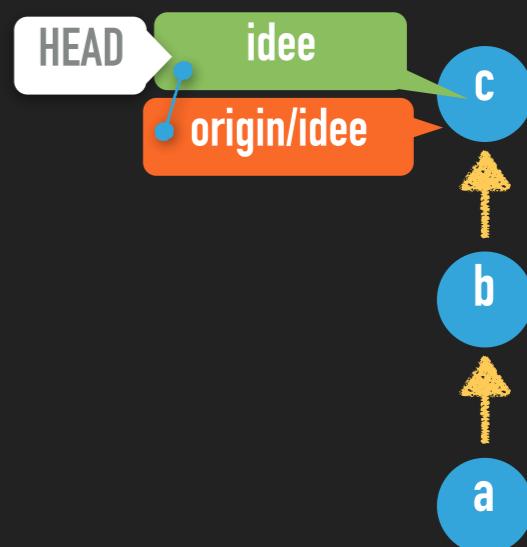
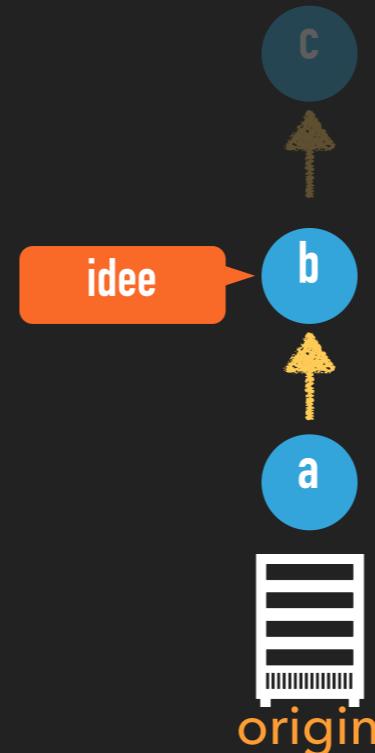
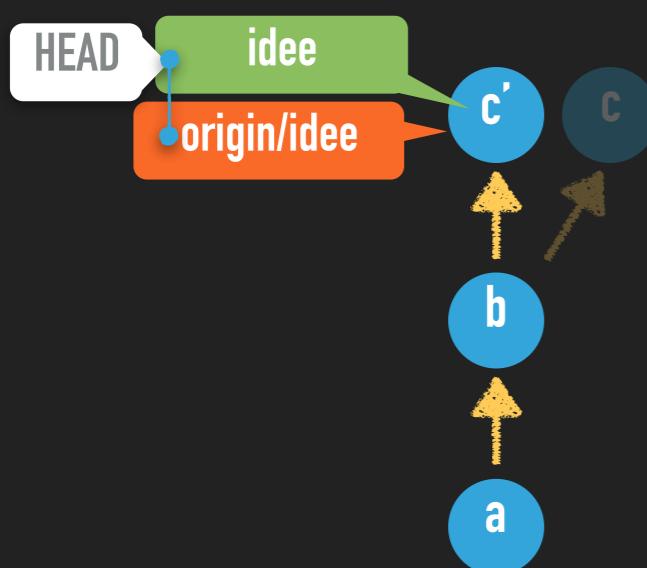
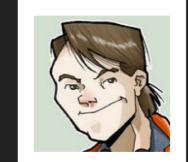
```
git reset --hard HEAD^  
git push -f
```



OÙ EST LE DANGER?



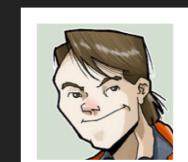
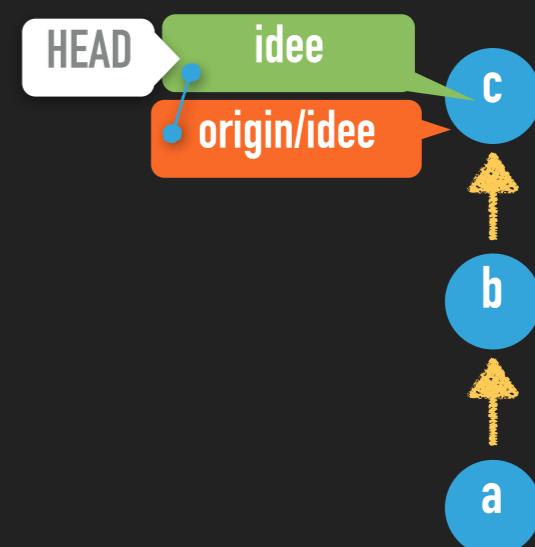
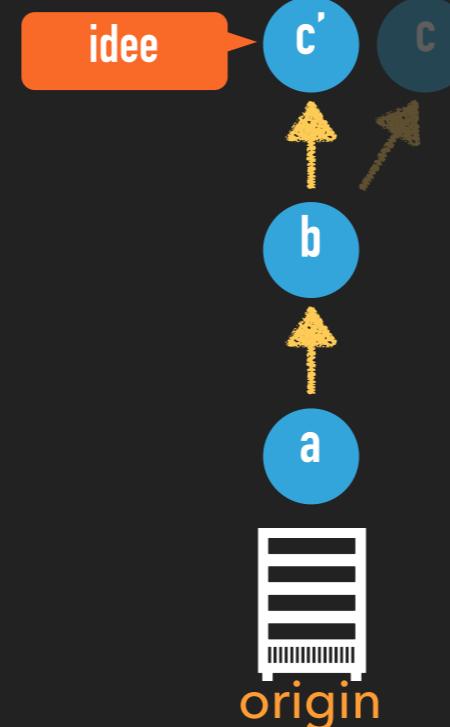
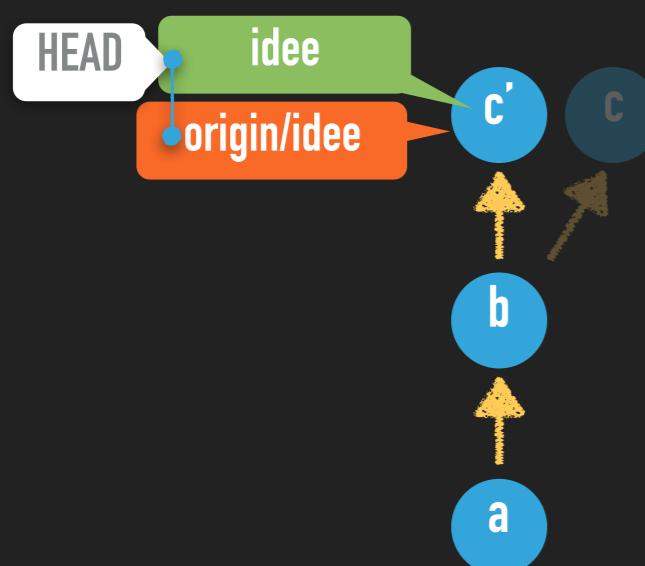
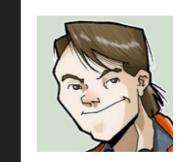
```
git reset --hard HEAD^  
git push -f  
git commit -m "je recommence"  
git push
```



OÙ EST LE DANGER?



```
git reset --hard HEAD^  
git push -f  
git commit -m "je recommence"  
git push
```



OÙ EST LE DANGER?

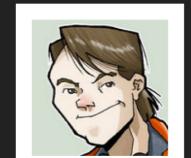


```
git reset --hard HEAD^
git push -f
git commit -m "je recommence"
git push
```

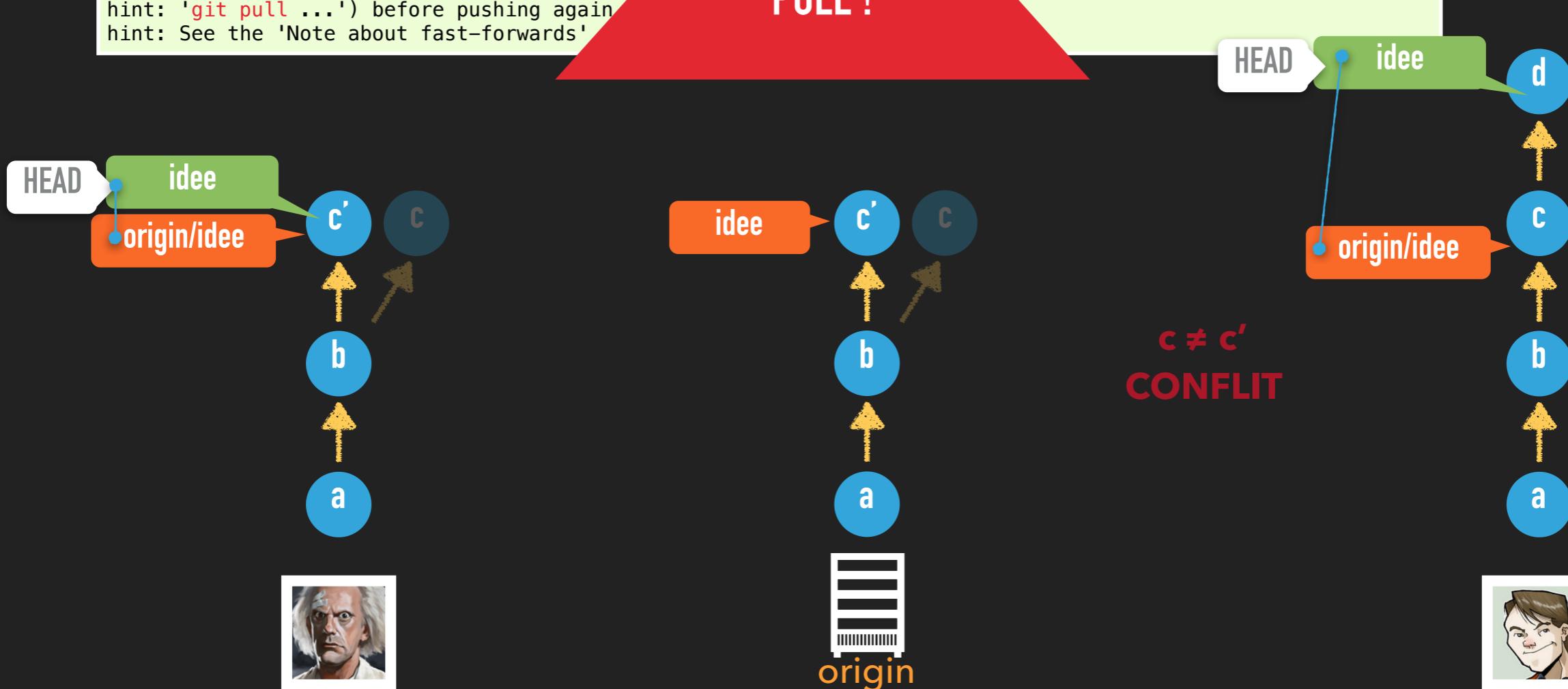
```
marty@lycee$ git push
To ssh://git@algec.iut-blagnac.fr:2200/git_training
 ! [rejected]      develop -> develop (non-fast-forward)
error: failed to push some refs to 'ssh://git@algec.iut-blagnac.fr:2200/git_training/basic.git'
hint: Updates were rejected because the tip of your local branch is behind
hint: its remote counterpart. Integrate the remote changes before pushing again.
hint: 'git pull ...' before pushing again.
hint: See the 'Note about fast-forwards'
```

**NE
PAS
FAIRE
PULL !**

git commit -m "a moi"



git push**push rejected : pull first**

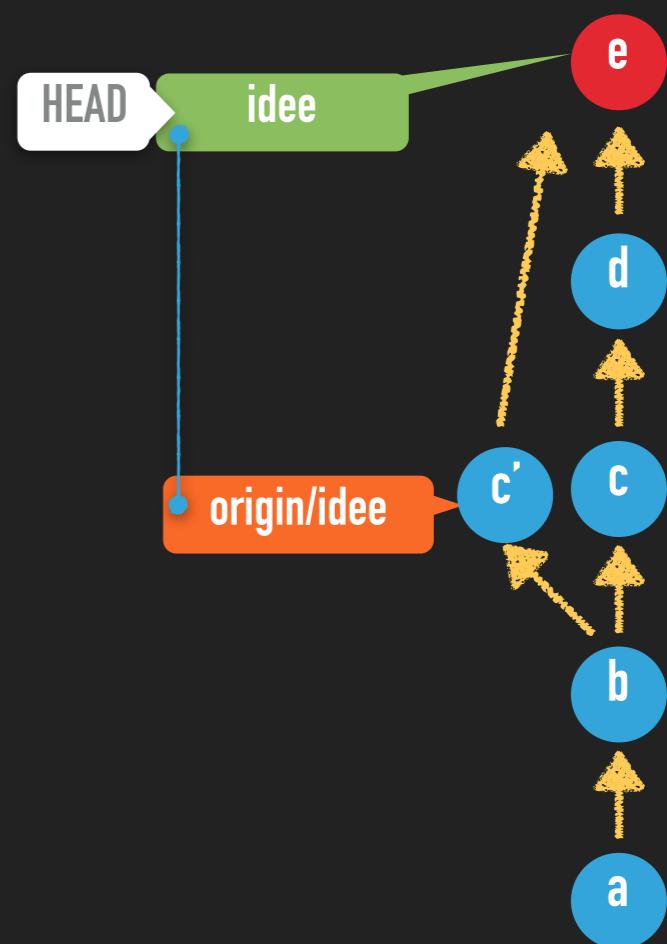
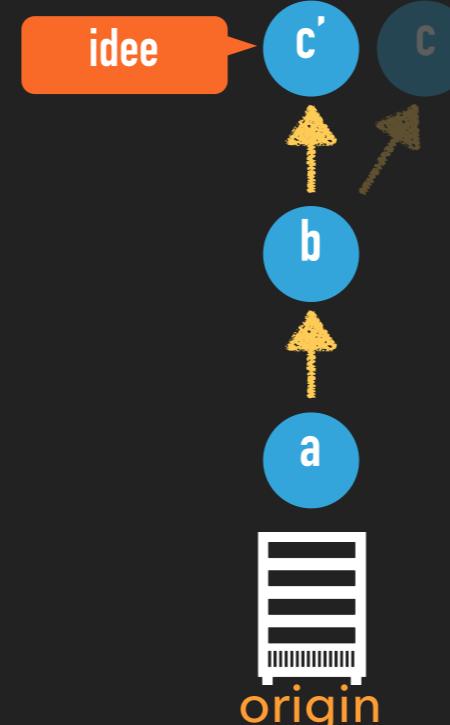
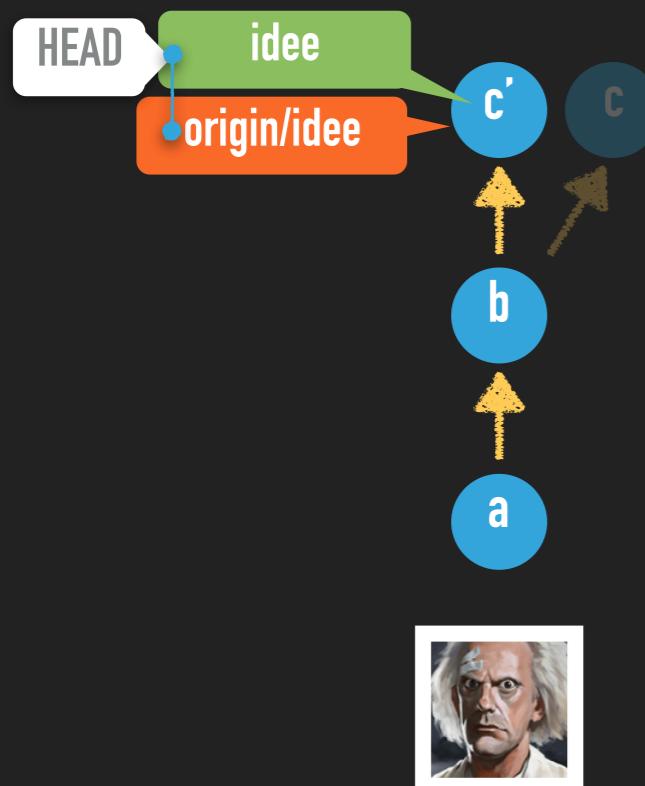
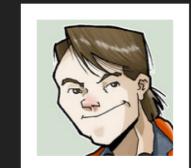


OÙ EST LE DANGER?



```
git reset --hard HEAD^  
git push -f  
git commit -m "je recommence"  
git push
```

```
git commit -m "a moi"  
git push  
git pull
```



LES COMMANDES QUI NECESSITENT UN PUSH -F

- ▶ :
 - git **branch** -f
 - git **reset**
 - git **commit** --amend
 - git **rebase**
 - git **pull** --rebase
- ▶ Ne **PAS** faire git **pull** ensuite
- ▶ Faire tourner 7 fois son clavier dans sa bouche
- ▶ Puis faire git **push** -f
- ▶ Prévenir les collègues sur la branche impactée



FAUT RÉFLÉCHIR, MCFLY, FAUT RÉFLÉCHIR !

DES COMMANDES EN VRAC 2

git remote [add|set-url|remove]

- ▶ **add:** ajoute un nouveau **remote**

```
git remote add mon_serveur ssh://serveur.url.com/project.git
```

- ▶ **set-url:** change l'url d'un **remote** existant

```
git remote set-url mon_serveur ssh://uneautre.url.com/  
project.git
```

```
git remote set-url origin      ssh://uneautre.url.com/  
nouveau_nom.git
```

- ▶ **remove:** enlève un **remote** existant

```
git remote remove mon_serveur
```

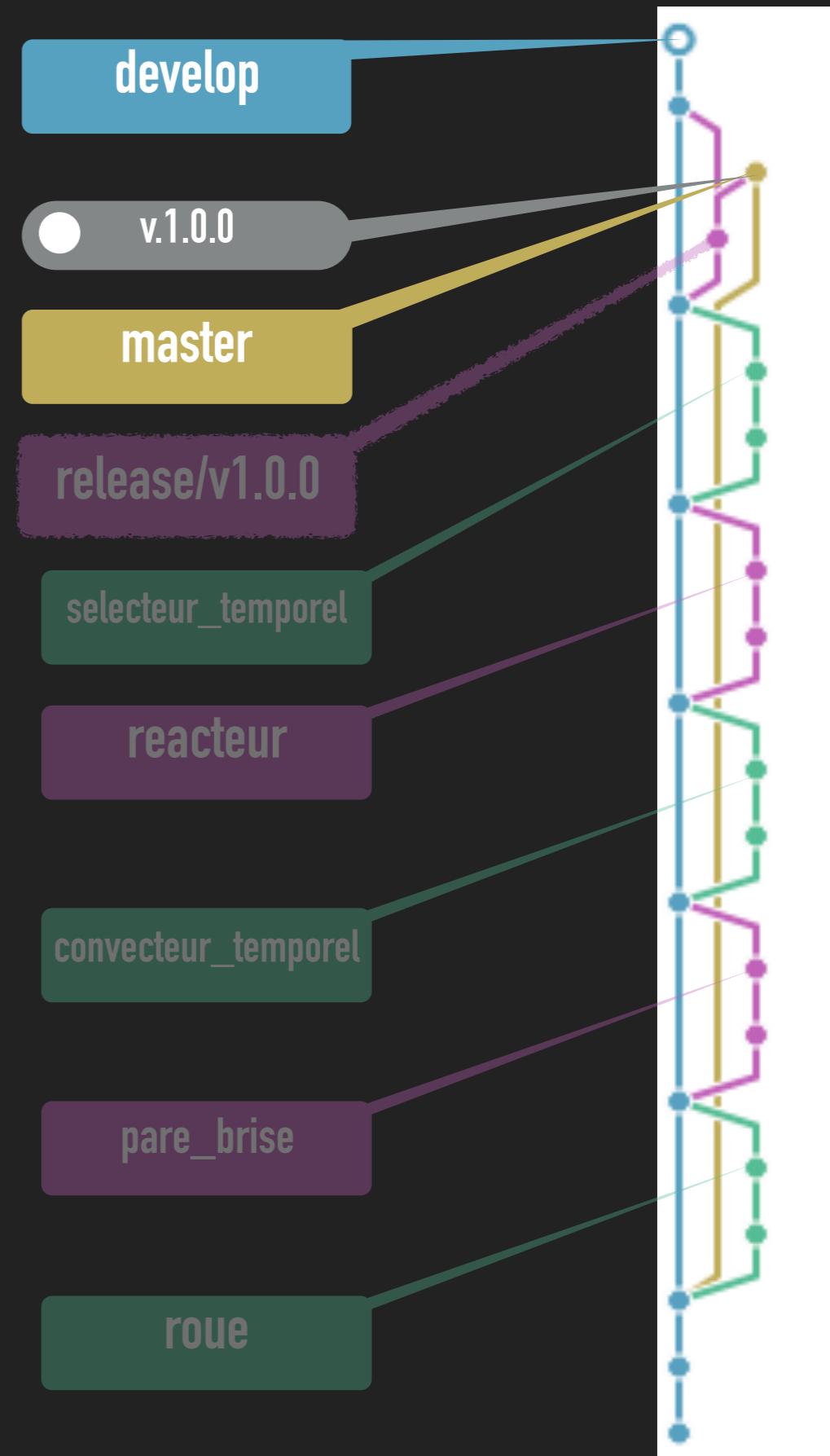
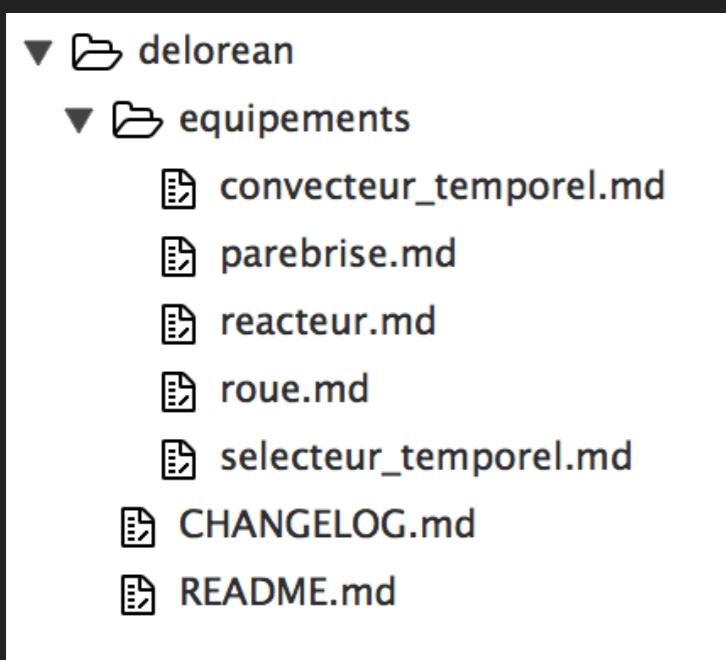


LES VOYAGES DANS LE TEMPS SONT BEAUCOUP TROP DANGEREUX.

DELorean V2

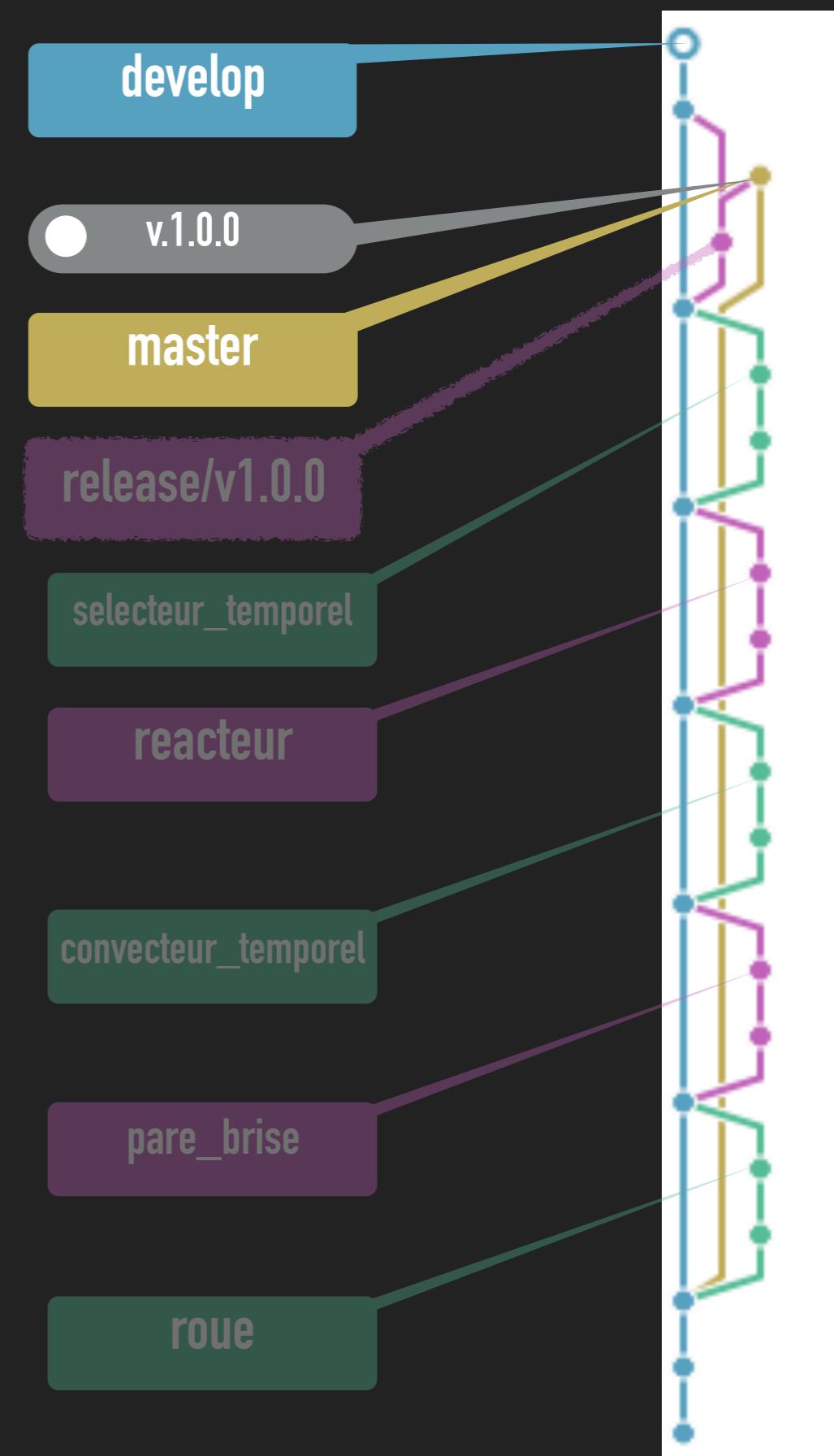
DESCRIPTION PROJET DELOREAN

- ▶ Un fichier README.md , un CHANGELOG.md
- ▶ Un dossier équipements/ qui contient un fichier par équipement
- ▶ Une v1.0.0 déjà taguée



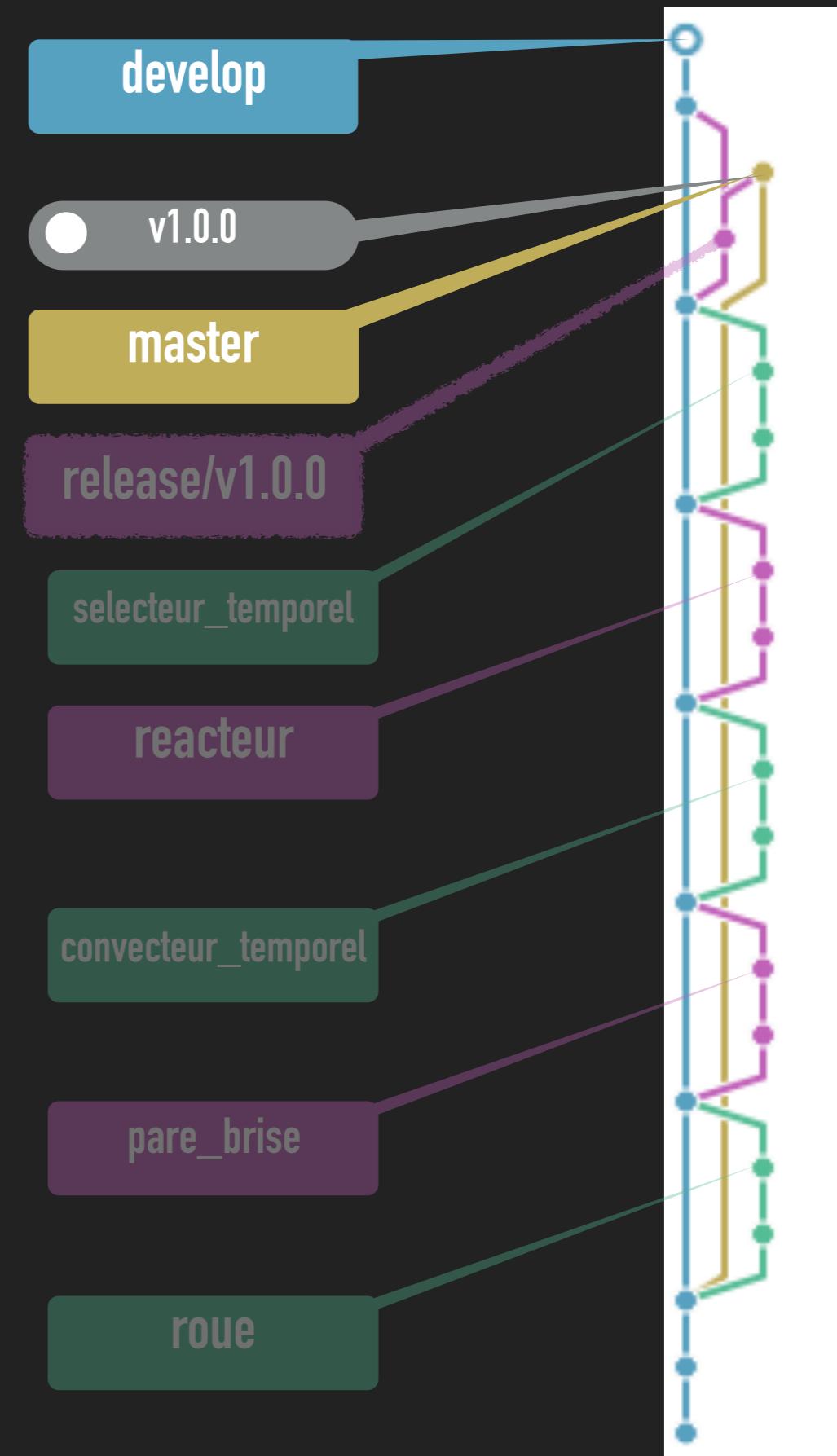
LA DELOREAN V1.1.0

- ▶ Ajout de la climatisation
- ▶ Ajout d'une lentille anti gravitationnelle
- ▶ Amélioration du réacteur en réacteur à détritus
- ▶ Amélioration du pare brise en pare brise anti-UVB
- ▶ Correction de bugs



LA DELOREAN V1.1.0

- ▶ Respect des règles de nommage des branches
 - ▶ feature/XXXX
 - ▶ release/vX.X.X
 - ▶ develop
 - ▶ master
- ▶ L'erreur est ~~permise~~ conseillée!



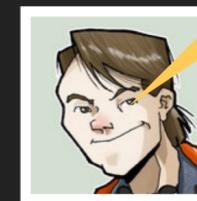
LA DELOREAN V1.1.0

LES ACTIONS

DELOREAN V2

MARTY

- ▶ Crée la branche **climatisation**
- ▶ Ajoute un équipement *climatisation.md* sur la branche **climatisation**
- ▶ Fait deux commits sur la branche **climatisation**



LE PERSO

IL N'EST PAS DIT QUAND FAIRE PUSH/PULL
A VOUS DE VOIR

GENERER DES CLÉS SSH

- ▶ Générez une paire de clés ssh
ssh-keygen.exe
-> id_rsa et id_rsa.pub dans \$HOME/.ssh
- ▶ Ajoutez votre clé ssh id_rsa.pub sur le compte gitlab
- ▶ https://gitlab_server/profile/keys

CLONER LE PROJET

- ▶ En binôme, deux par projet, clonez:
 - ▶ `ssh://git@algec.iut-blagnac.fr:2200/git_training_2016/delorean-{01-20}.git`
- ▶ Choisissez un rôle: Doc ou Marty
- ▶ Veillez à bien avoir configuré nom prénom et mail pour que je puisse attribuer les notes

NOUVELLE FEATURE



- ▶ Crée la branche `reacteur_dechet` à partir de `develop`
- ▶ Change le fichier `reacteur.md` pour qu'il consomme des détritus
- ▶ Commit le fichier `reacteur.md` sur la branche `reacteur_dechet`

REINTEGRATION DE FEATURE



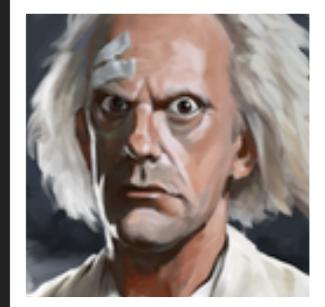
- ▶ Réintègre la branche `reacteur_dechet` dans `develop`

NOUVELLE FEATURE



- ▶ Crée la branche **climatisation**
- ▶ Ajoute un équipement *climatisation.md* sur la branche **climatisation**
- ▶ Fait deux commits sur la branche **climatisation**

QUICKFIX



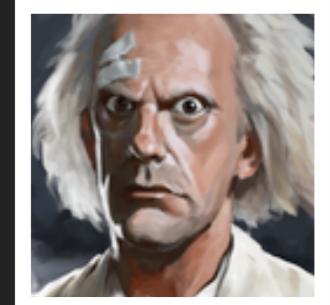
- ▶ Sur la branche `develop`, ajoute une caractéristique sur la roue.md (un mode hiver par exemple)
- ▶ Commit le changement de fichier sur la roue

REINTEGRATION DE FEATURE



- ▶ Réintègre la branche **climatisation** dans **develop**

NOUVELLE FEATURE



- ▶ Crée la branche `lentille_gravitationnelle`
- ▶ Ajoute un équipement
`lentille_gravitationnelle.md` sur la branche
`lentille_gravitationnelle`
- ▶ Fait deux commits sur la branche
`lentille_gravitationnelle`

TRAVAIL À DEUX SUR UNE BRANCHE



- ▶ Reprend le travail du doc sur la branche **lentille_gravitationnelle**
- ▶ Ajoute un commit supplémentaire sur la branche **lentille_gravitationnelle**

RÉINTÉGRATION DE BRANCHE



- ▶ Réintègre la branche `lentille_gravitationnelle` dans `develop`

NOUVELLE FEATURE



- ▶ Crée la branche `pare_brise_plus`
- ▶ Ajoute une caractéristique dans `parebrise.md`
 - ▶ filtre anti UVB
- ▶ commit l'amélioration sur `pare_brise_plus`

CORRECTION DE BUG DANS DEVELOP



- ▶ Corrige le bug dans parebrise.md sur la branche develop
- ▶ commit le bug fix sur develop

RÉCUPÉRATION DE LA CORRECTION DANS LA BRANCHE



- ▶ Récupère le bugfix sur le parebrise.md du doc depuis `develop` dans sa branche `pare_brise_plus`
- ▶ Change une autre caractéristique du parebrise.md et la commit dans la branche `pare_brise_plus`

RÉINTÉGRATION DE BRANCHE



- ▶ Réintègre la branche `pare_brise_plus` dans `develop`

C'EST L'HEURE DE LA LIVRAISON!



- ▶ Crée la branche de livraison `release/v1.1.0`
- ▶ Remplit le fichier `CHANGELOG.md`
- ▶ Commit le `CHANGELOG.md`

NOUVELLE FEATURE



- ▶ Crée la branche `toit_ouvrant` depuis `develop`
- ▶ Ajoute l'équipement `toit_ouvrant.md`
- ▶ Commit le fichier sur `toit_ouvrant`
- ▶ Ajoute un autre commit sur `toit_ouvrant`

QUICK FIX DE DERNIÈRE MINUTE



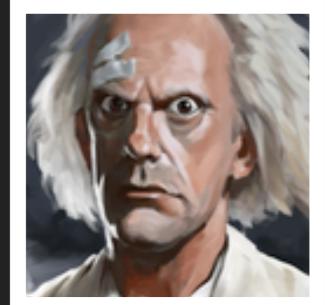
- ▶ sur `release/v1.1.0`
- ▶ Corrige le bug du `selecteur_temporel`
- ▶ Ajoute la correction du bug dans `CHANGELOG.md`
- ▶ Commit `CHANGELOG.md` et le bug

RÉINTÉGRATION DE BRANCHE



- ▶ Réintègre la branche `toit_ouvrant` dans `develop`

FIN DE RELEASE

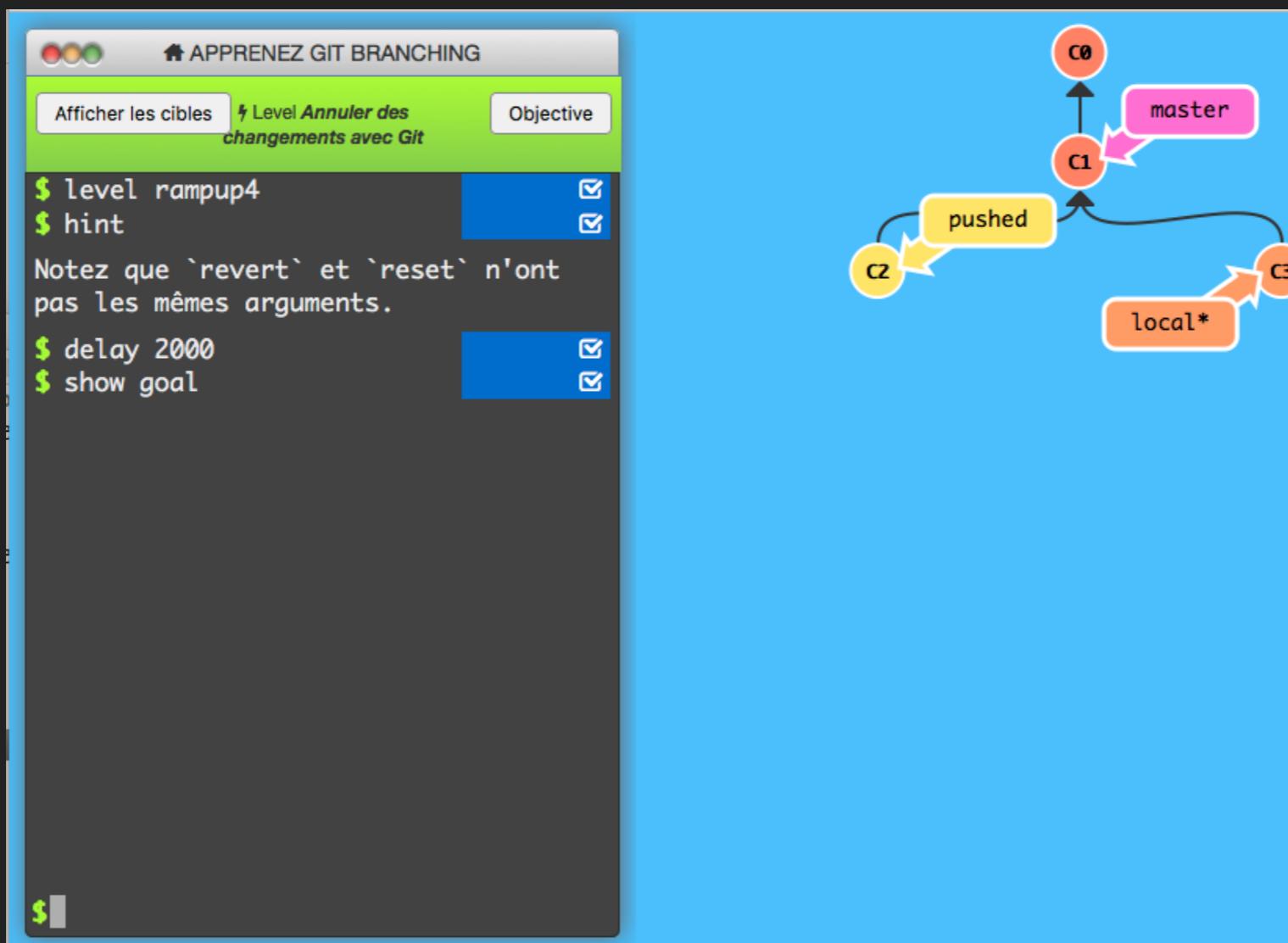


- ▶ Merge la branche `release/v1.1.0` dans `master`
- ▶ Pose le tag `v1.1.0` sur `master`
- ▶ Merge la branche `release/v1.1.0` dans `develop`

POUR CEUX QUI VEULENT S'AMUSER (OUI OUI)

► Games:

<http://pcottle.github.io/learnGitBranching/>





LE FUTUR N'EST JAMAIS ÉCRIT À L'AVANCE POUR PERSONNE

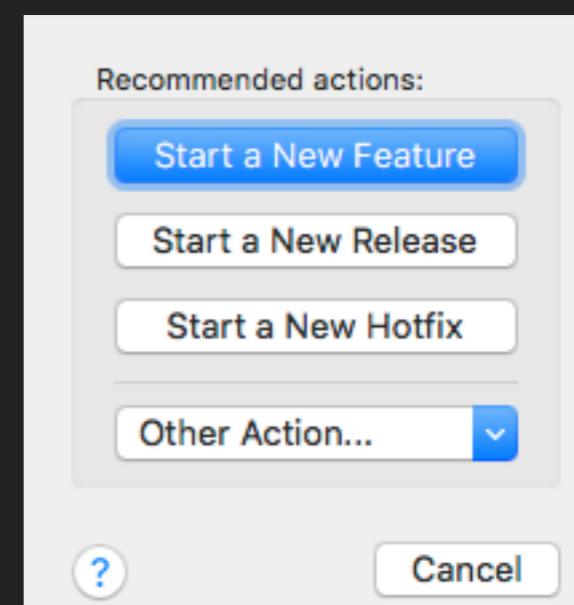
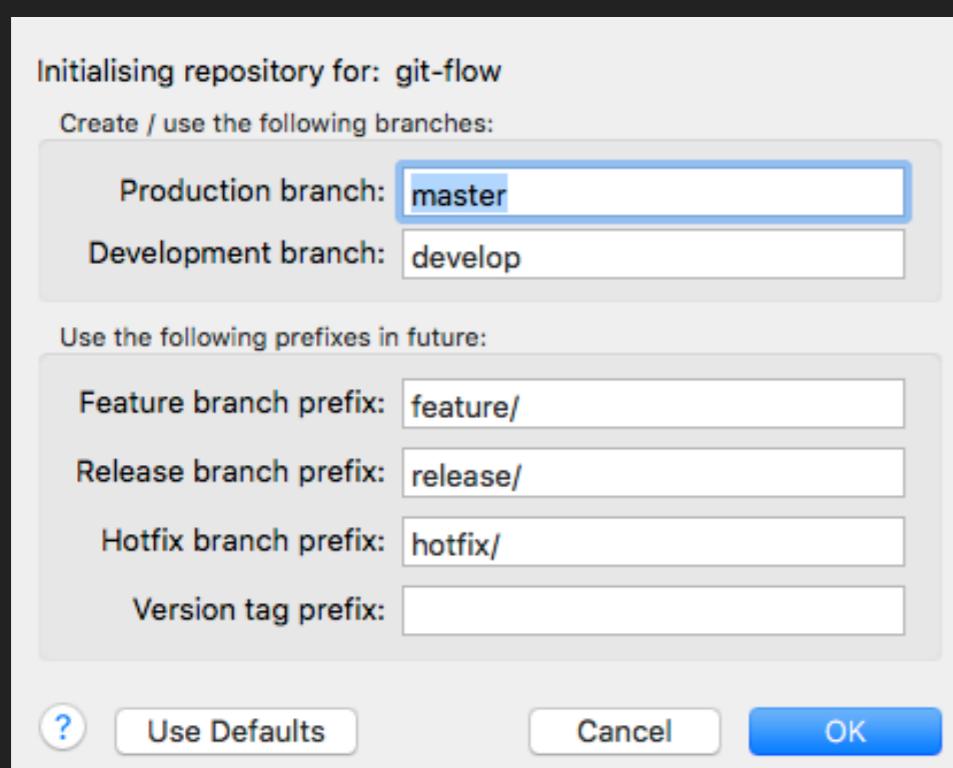
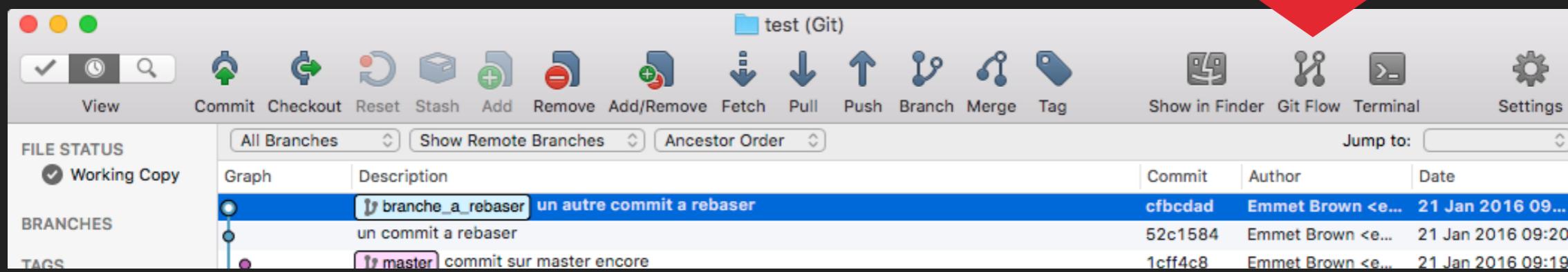
GITFLOW

UNE COMMANDE POUR FAIRE DU NVIE SIMPLEMENT

- ▶ <https://github.com/nvie/gitflow>
- ▶ une surcouche à git
 - ▶ git flow feature start / finish / rebase
 - ▶ git flow release start / finish

UNE COMMANDE POUR FAIRE DU NVIE SIMPLEMENT

► Bouton gitflow dans SourceTree



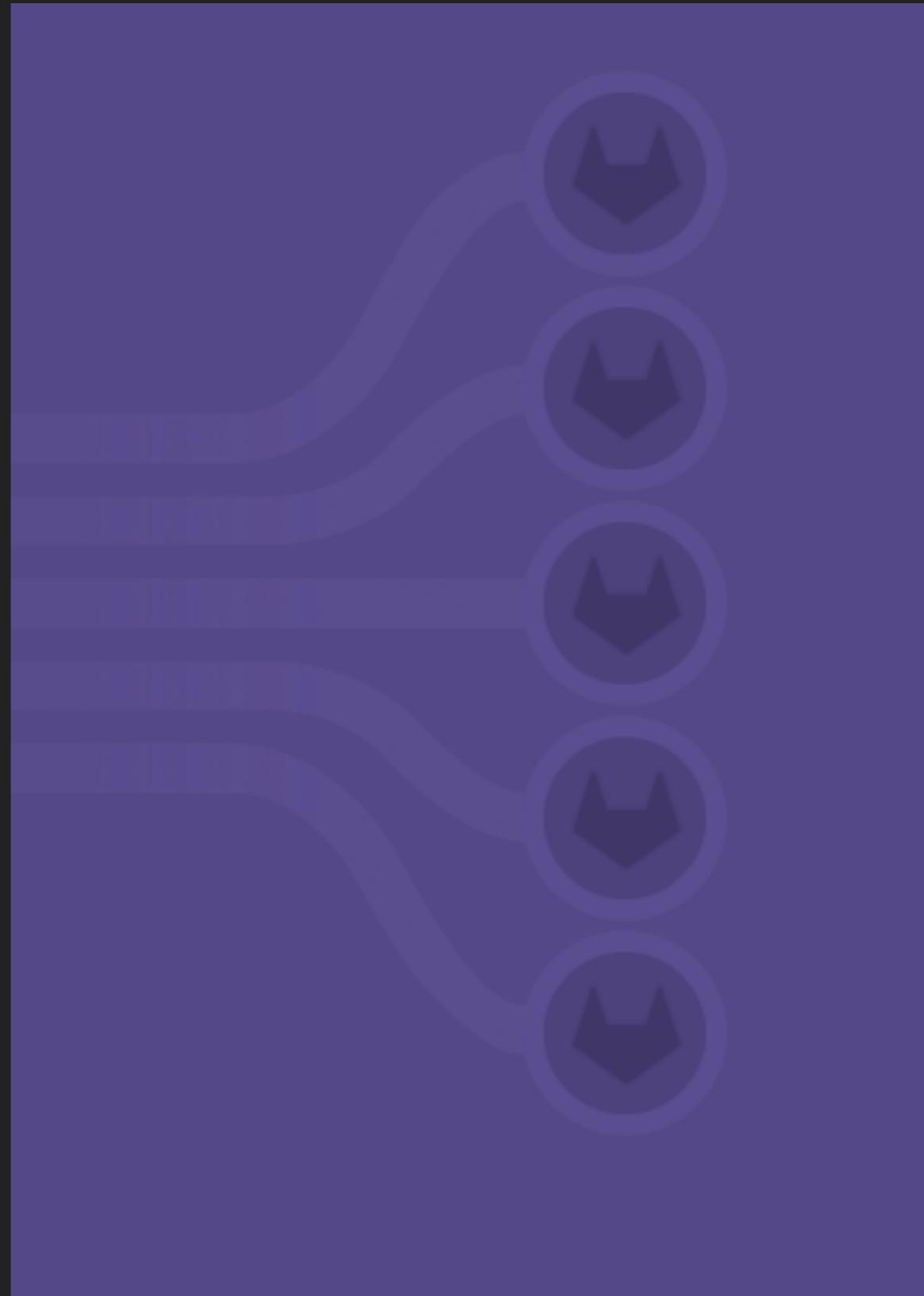


LES PRINCIPES DE

GITLAB

GITLAB

- ▶ open source (dans sa version CE)
- ▶ <https://about.gitlab.com/>
- ▶ gestion de repository
- ▶ projets, groupes de projets
- ▶ utilisateurs, équipe
- ▶ intégration continue
- ▶ revue de code



GESTION DE REPOSITORY

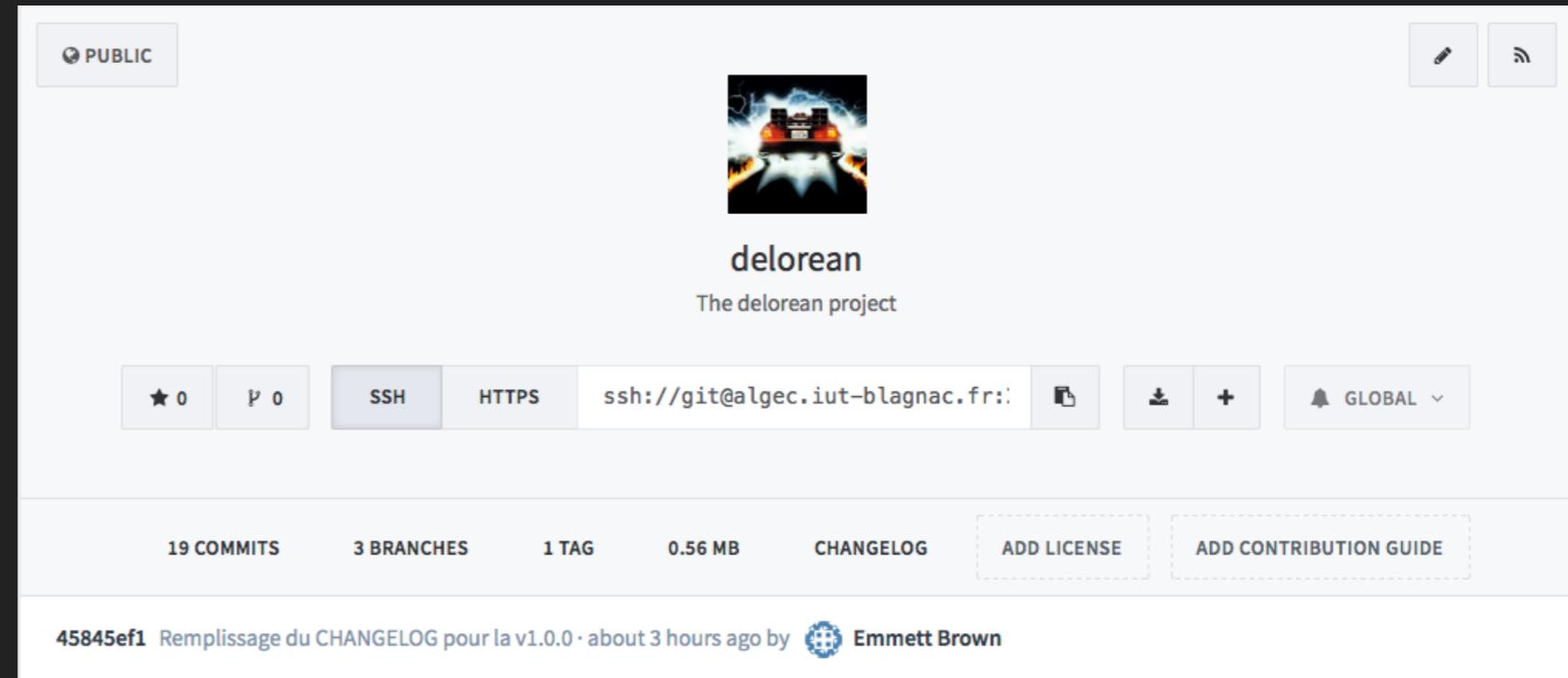
- ▶ Les projets sont organisés par groupes ou namespaces
- ▶ Chaque utilisateur à un namespace personnel
- ▶ On peut créer des groupes de projets

The screenshot shows the GitLab web interface. On the left is a dark sidebar with the GitLab logo at the top. Below it are several menu items: 'Projects' (with a house icon), 'Activity' (with a person icon), 'Groups' (with a people icon, highlighted with a yellow box), 'Milestones' (with a clock icon), 'Issues' (with a bug icon), 'Merge Requests' (with a merge icon), 'Snippets' (with a code icon), 'Help' (with a question mark icon), and 'Profile Settings' (with a user icon). To the right of the sidebar is a light-colored main area titled 'Projects'. At the top of this area is a search bar labeled 'Filter by name'. Below the search bar is a table listing four projects. Each project row has a small circular icon on the left, a project name in blue text, and a brief description in grey text.

- D git_training_2016 / delorean Le projet delorean pour...
- git_training / delorean The delorean project
- B git_training / basic Un projet basique pour...
- git_training / hill_valley The hill valley storyline

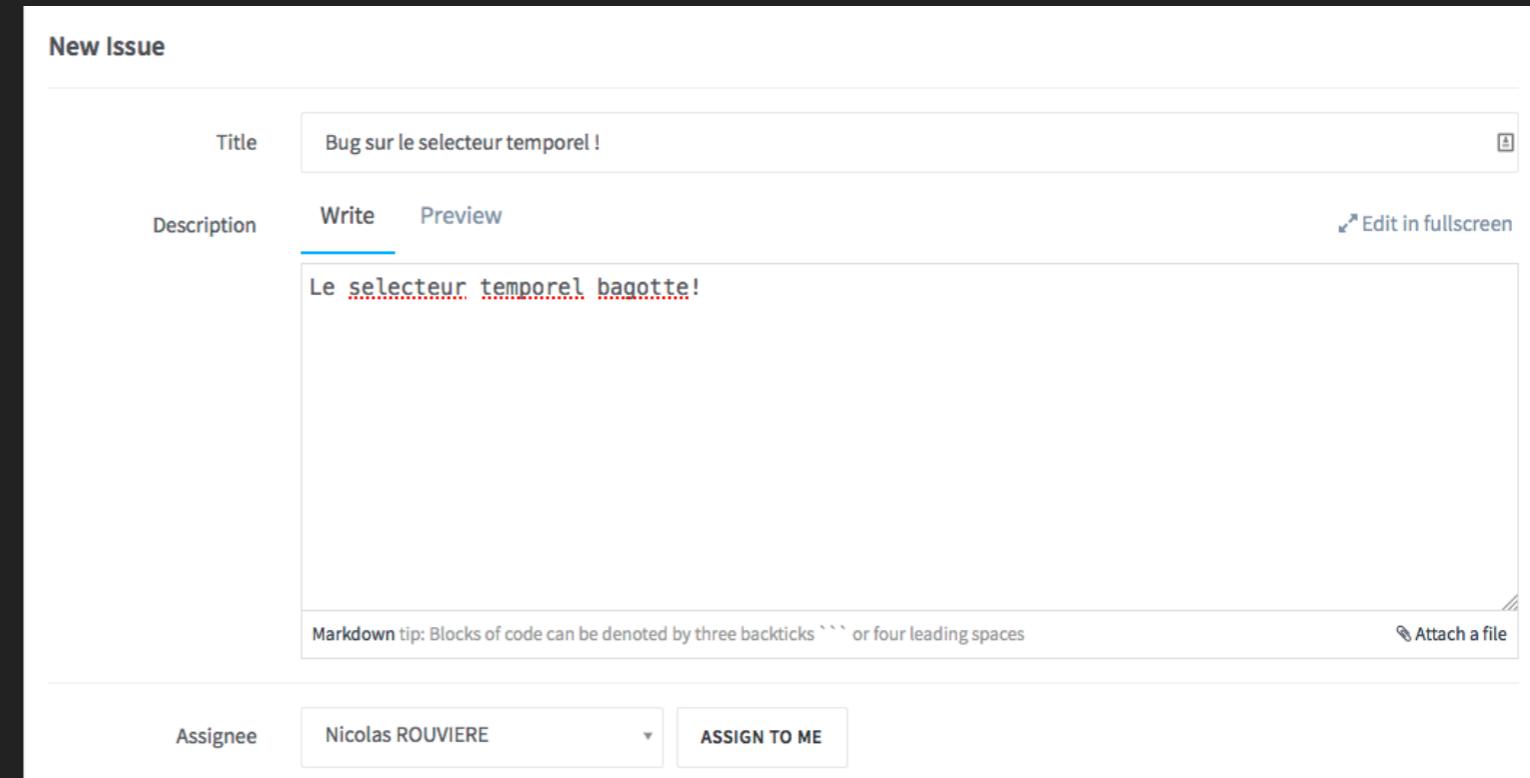
PROJET

- ▶ une URL d'accès
- ▶ Un README.md
- ▶ Un flux d'activité
- ▶ Un gestionnaire d'anomalies, un wiki, des snippets, des builds
- ▶ Des stats
- ▶ Des merge requests



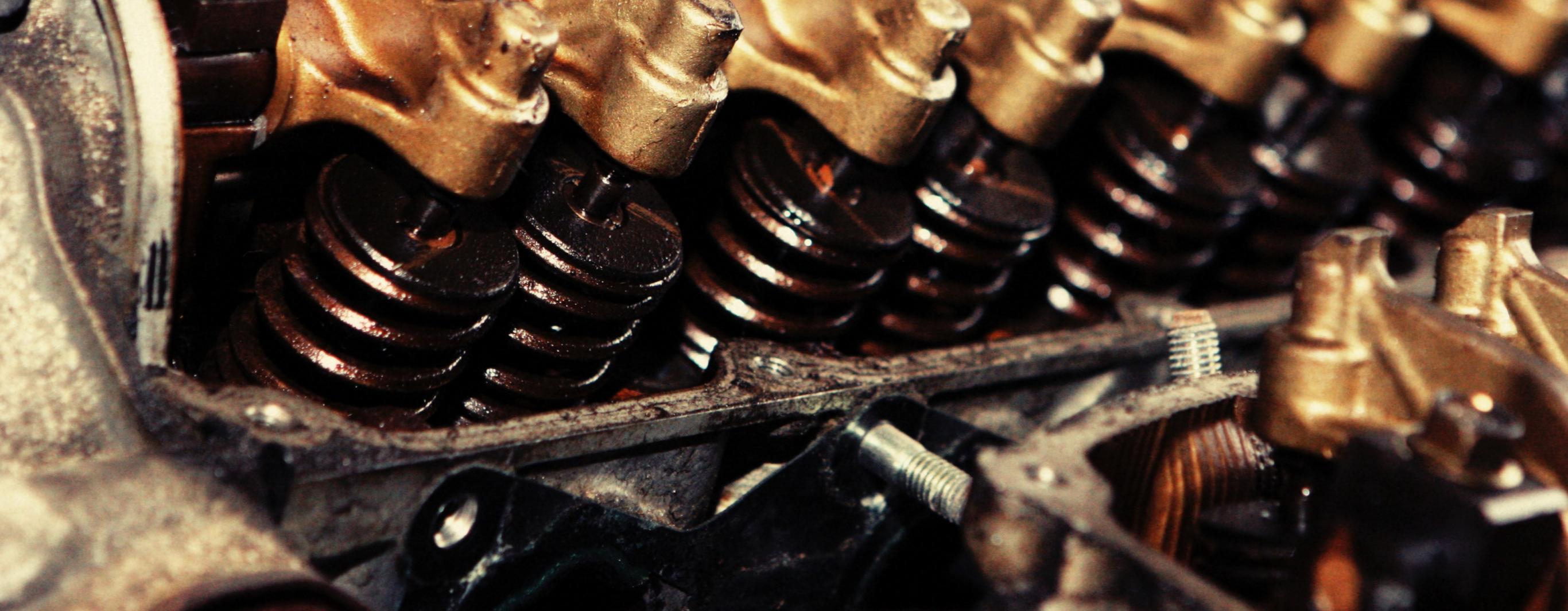
GESTION D'ISSUES

- ▶ simple: OUVERT/FERME
- ▶ Assignation
- ▶ description au format markdown
- ▶ Lien avec le code:
`git commit -m '#1 un commentaire en lien avec l'issue 1'`
`git commit -m 'fixes #1 ce commentaire fermera l'issue 1 dès son merge dans develop'`



MERGE REQUEST

- ▶ Les branches master et develop peuvent être protégées
Seul le owner du projet peut pusher dessus
Seul le owner du projet merge les branches dessus
- ▶ Les développeurs doivent demander une merge-request
- ▶ Revue de code
- ▶ Merge en ligne!



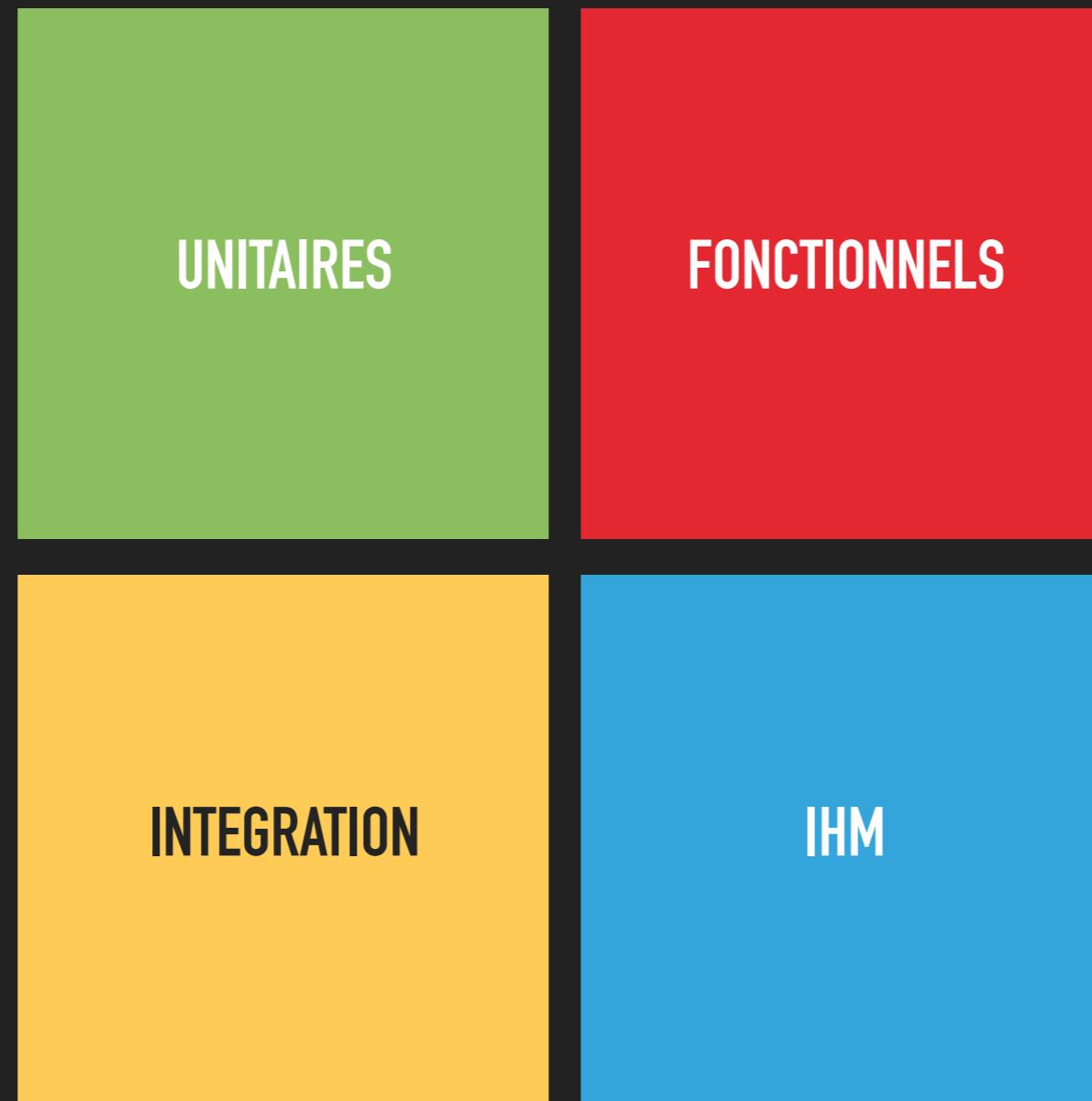
LES PRINCIPES DE

GITLAB-CI

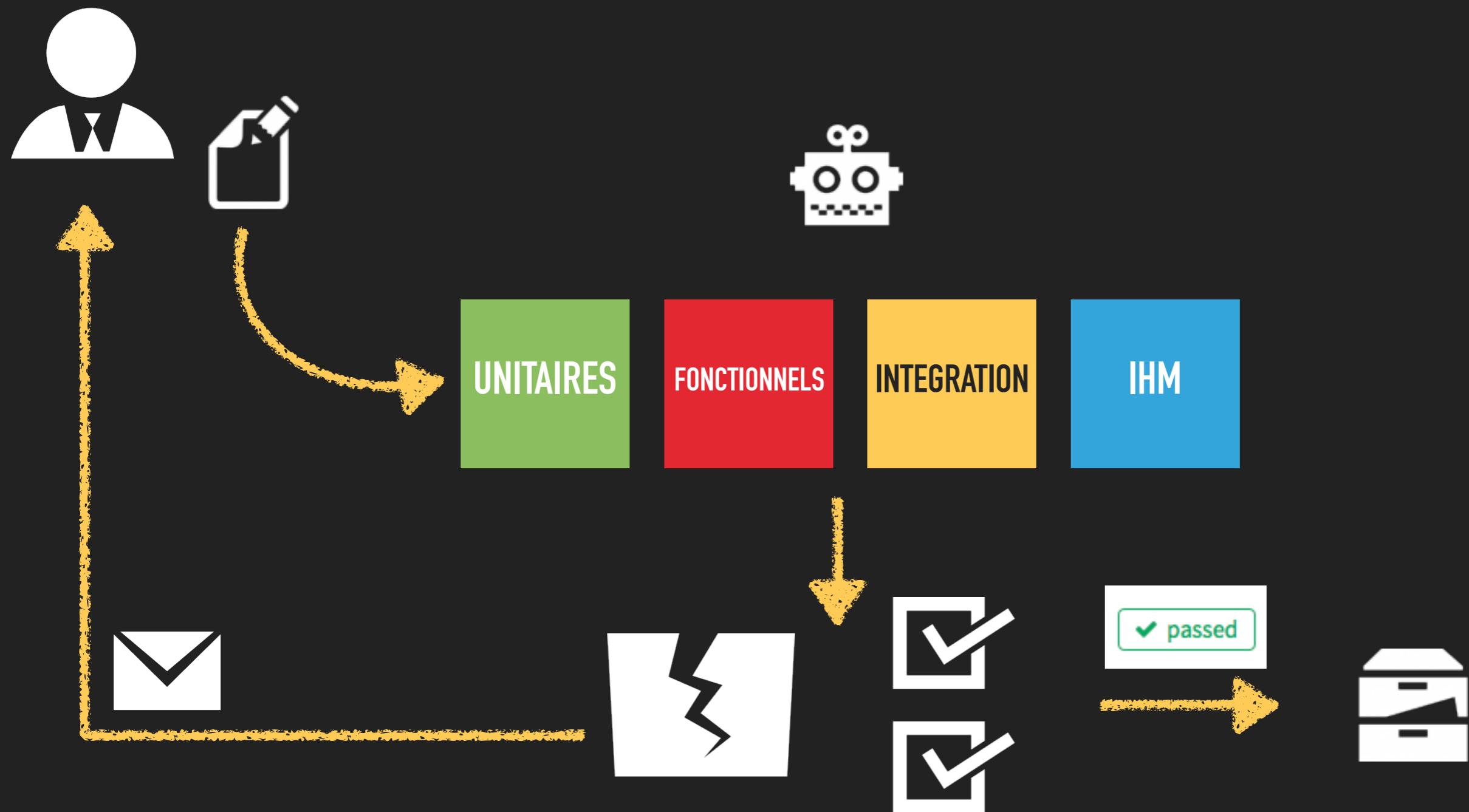
TESTER SON CODE À CHAQUE COMMIT

- ▶ automatiser ses tests
- ▶ ajouter un fichier de description des tests dans son repo
- ▶ gitlab le détecte et mandate un runner pour executer les tests
- ▶ chaque commit est décoré avec les résultats de tests

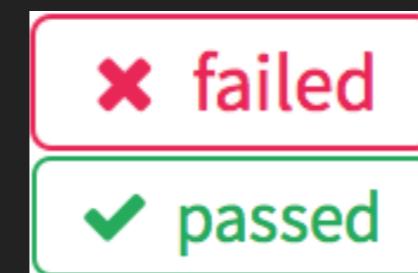
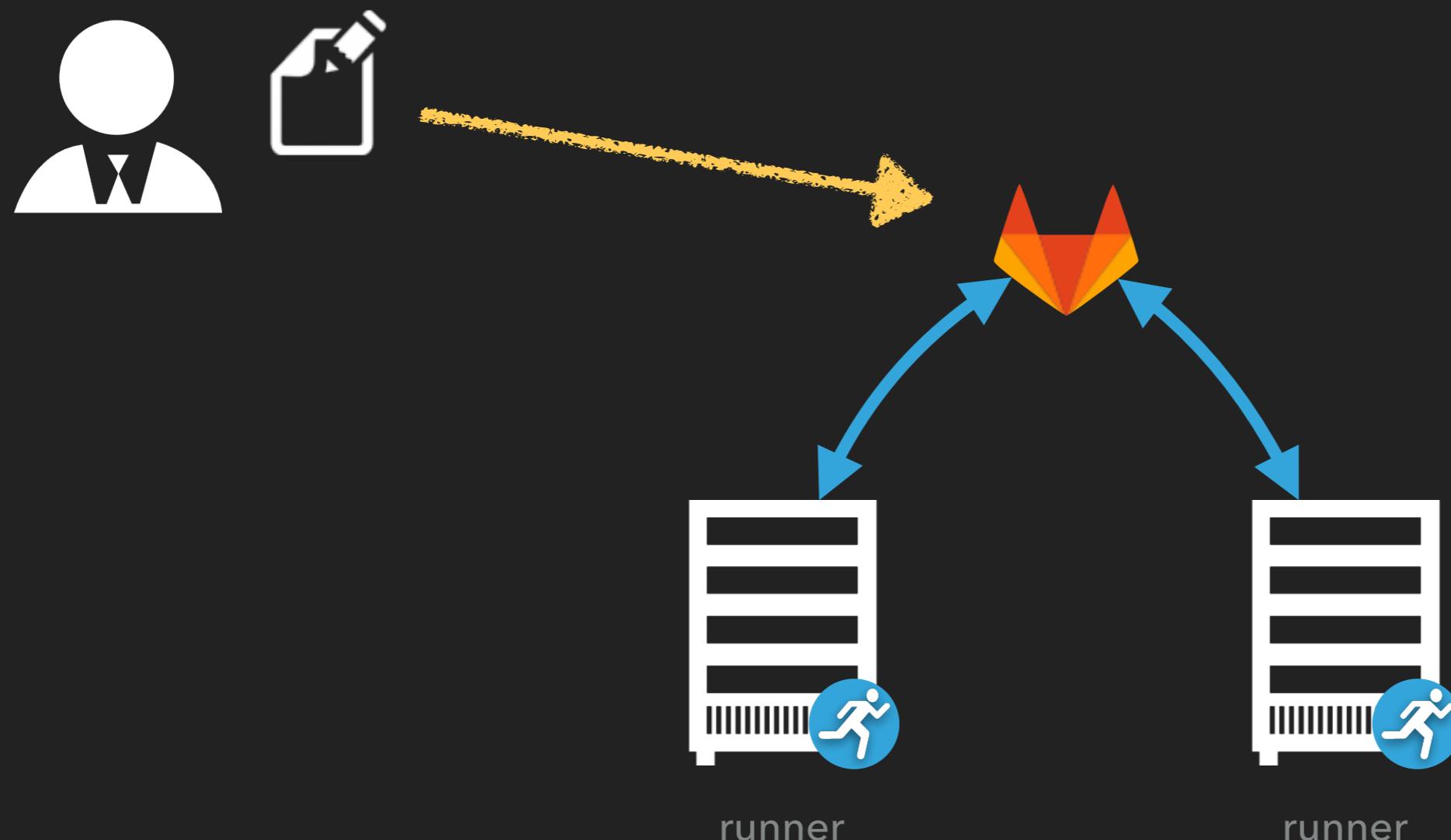
LES TYPES DE TEST



LA RELEASE CONTINUE



GITLAB CI UN SERVEUR DES RUNNERS



.gitlab-ci.yml

.gitlab-ci.yml

```
stages:
- build
- test
- deploy
- cleanup

build_job:
  stage: build
  script:
  - make build

test_job:
  stage: test
  script:
  - make test

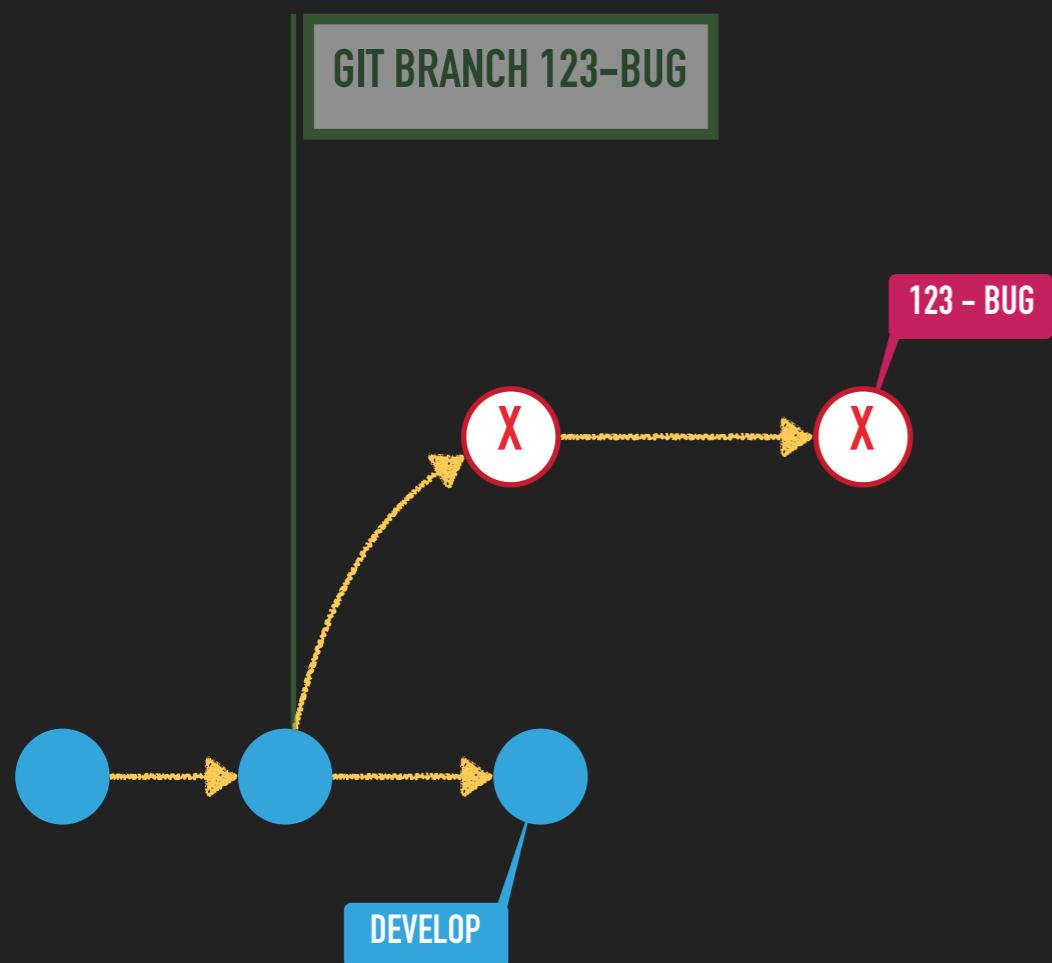
deploy_job:
  stage: deploy
  script:
  - make deploy

cleanup_job:
  stage: cleanup
  script:
  - cleanup after builds
when: always
```

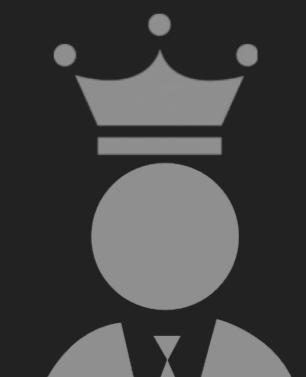
MANAGE ISSUE 123 WITH GITLAB



MANAGE ISSUE 123 WITH GITLAB

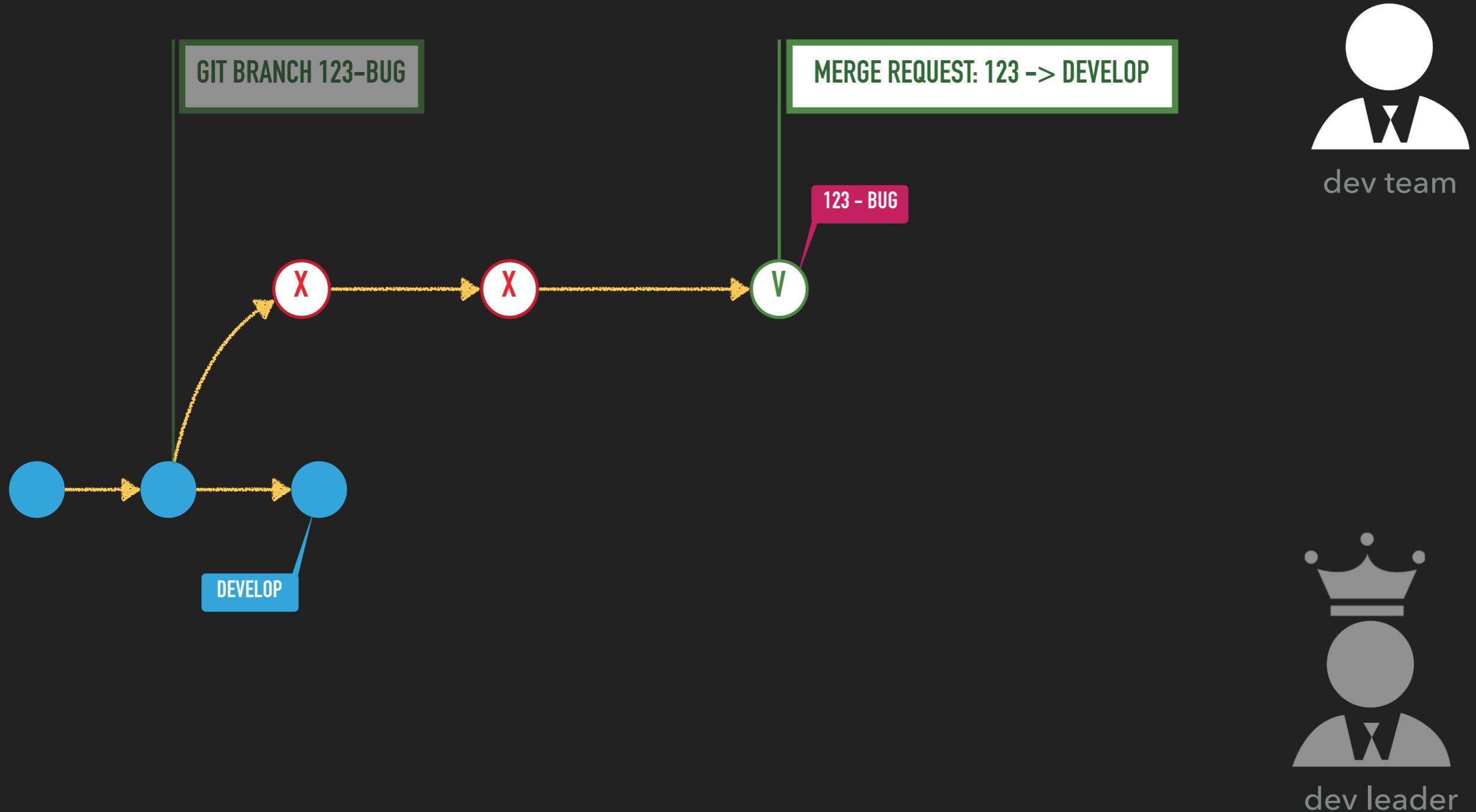


dev team

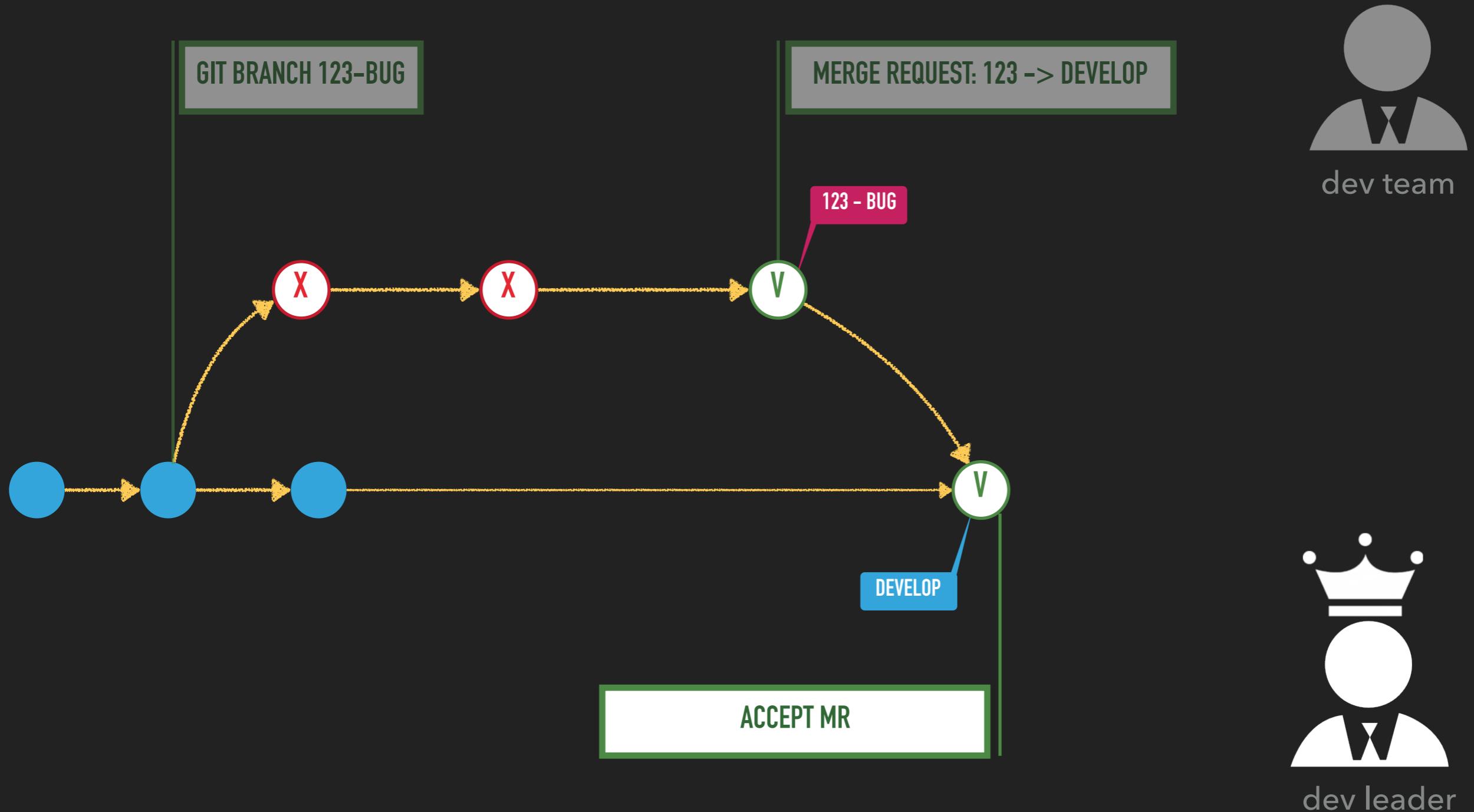


dev leader

MANAGE ISSUE 123 WITH GITLAB



MANAGE ISSUE 123 WITH GITLAB





LES VOYAGES DANS LE TEMPS SONT BEAUCOUP TROP DANGEREUX.

DELOREAN V2

CLONER LE PROJET

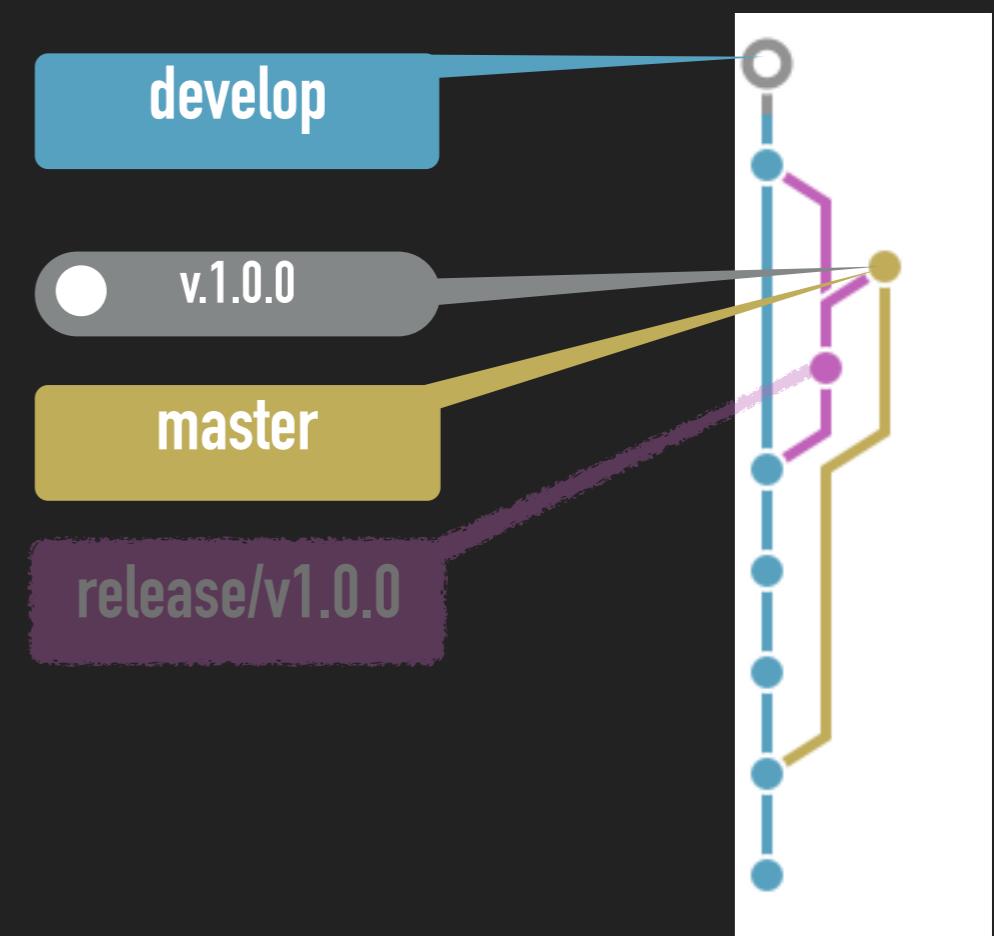
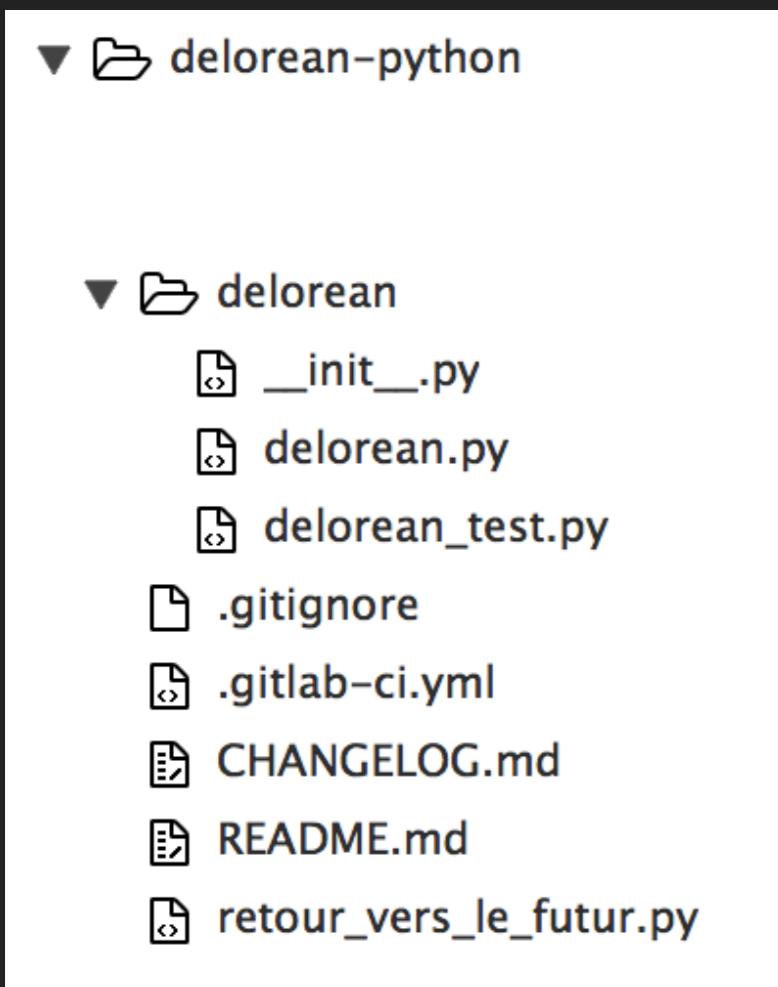
- ▶ En binôme, deux par projet, clonez:
- ▶ `ssh://git@algec.iut-blagnac.fr:2200/git_training_2017/delorean-python-{01-20}.git`
- ▶ Indiquez vos noms-mails dans le `README.md`
commit + push dans **develop**

Réalisé par:

- | - PRENOM NOM ELEVE 1, prenom.nom@monmail.com
- PRENOM NOM ELEVE 2, prenom2.nom2@monmail.com

DESCRIPTION PROJET DELOREAN

- ▶ Un fichier README.md , un CHANGELOG.md
- ▶ Une v1.0.0 déjà taguée



delorean/delorean.py

```
class Delorean(object):
    """A delorean class"""
    + def __init__(self): ...
    + def set_speed(self, speed): ...
    + def get_speed(self): ...
    + def insert_plutonium(self): ...
    @property
    + def has_plutonium(self): ...
    + def set_target_date(self, target_date): ...
    + def get_date(self): ...
    + def __speed_changed(self): ...
```

DELOREAN V2

(develop)\$ python retour_vers_le_futur.py

```
from delorean.delorean import Delorean

print 'Doc construit une delorean'
delo = Delorean()
print 'Nous sommes le %s' % delo.get_date()

print 'Doc refait le plein de plutonium'
delo.insert_plutonium()

print 'Doc programme un voyage dans le temps pour retourner en 1955'
delo.set_target_date('05/11/1955')

print 'Doc accelere jusqu a 89mph'
delo.set_speed(89)

print ##### VOYAGE DANS LE TEMPS #####
print 'Nous sommes le %s' % delo.get_date()
```

```
/usr/bin/python /Users/booba/git/delorean-python/retour_vers_le_futur.py
Doc construit une delorean
Nous sommes le 2016-01-29
Doc refait le plein de plutonium
Doc programme un voyage dans le temps pour retourner en 1955
Doc accelere jusqu a 89mph
Delorean traveled in time to 05/11/1955
##### VOYAGE DANS LE TEMPS #####
Nous sommes le 1955-11-05
```

Process finished with exit code 0

RECOMMANDATIONS TP

- ▶ Lever des issues pour chaque item de travail (4 issues)
- ▶ Indiquer le numéro d'issue dans le commentaire
git commit -m "#1: correction du bug"
- ▶ Tester en local avant de pusher
- ▶ Suivre l'état des tests après un push
- ▶ Travaillez en séquence, pas en parallèle
- ▶ Livrez la v1.1.0
- ▶ N'oubliez pas de PUSHER votre travail avant de partir

EVALUATION

- ▶ 100% des méthodes testées
- ▶ Tests OK
- ▶ 100% de couverture de code
- ▶ workflow NVIE suivi (branches de feature, release, master)
- ▶ Tag v1.1.0 posé sur master

A FAIRE (VISIBLE DANS LE README.MD)

TODOs V1.1.0

BUG selecteur

- Corriger le bug du selecteur temporel pour que les tests passent

Ajout lentille gravitationnelle

- ajouter la méthode `enable_antigravity_lens(self)` dans la classe `Delorean`
- ajouter la méthode `disable_antigravity_lens(self)` dans la classe `Delorean`
- ajouter la méthode `is_flying(self)` dans la classe `Delorean` qui renvoie `True` quand la lentille est activée.
- ajouter le test `test_delorean_antigravity()` dans `delorean_test.py`

Amélioration du réacteur en reacteur à déchets

- remplacer la méthode `insert_plutonium(self)` par la méthode `insert_waste(self)` dans la classe `Delorean`
- remplacer la méthode `has_plutonium(self)` par la méthode `has_waste(self)` dans la classe `Delorean`
- remplacer la méthode `test_delorean_insert_plutonium()` par la méthode `test_delorean_insert_waste(self)` dans `delorean_test.py`

Ajout climatisation

- ajouter la méthode `set_temperature(self, temperature)` dans la classe `Delorean`
- ajouter la méthode `get_temperature(self)` dans la classe `Delorean`
- ajouter le test `test_delorean_temperature()` dans `delorean_test.py`

BONUS

- ▶ Ajouter un stage doc dans .gitlab-ci.yml
- ▶ Générer la documentation python du module

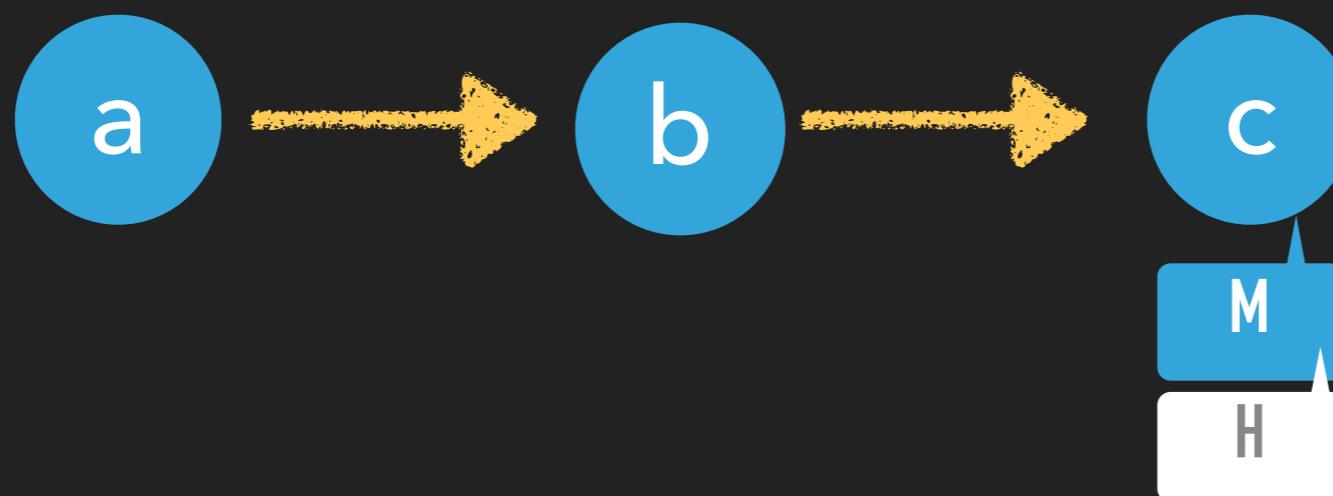
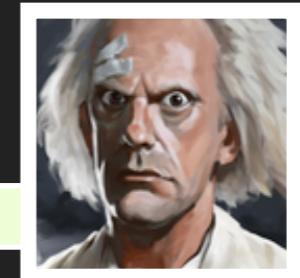


POUR ALLER PLUS LOIN

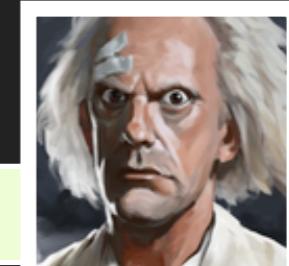
ET APRES

git checkout -b name [ref]

```
doc@labo$ git checkout -b ideefolle
```

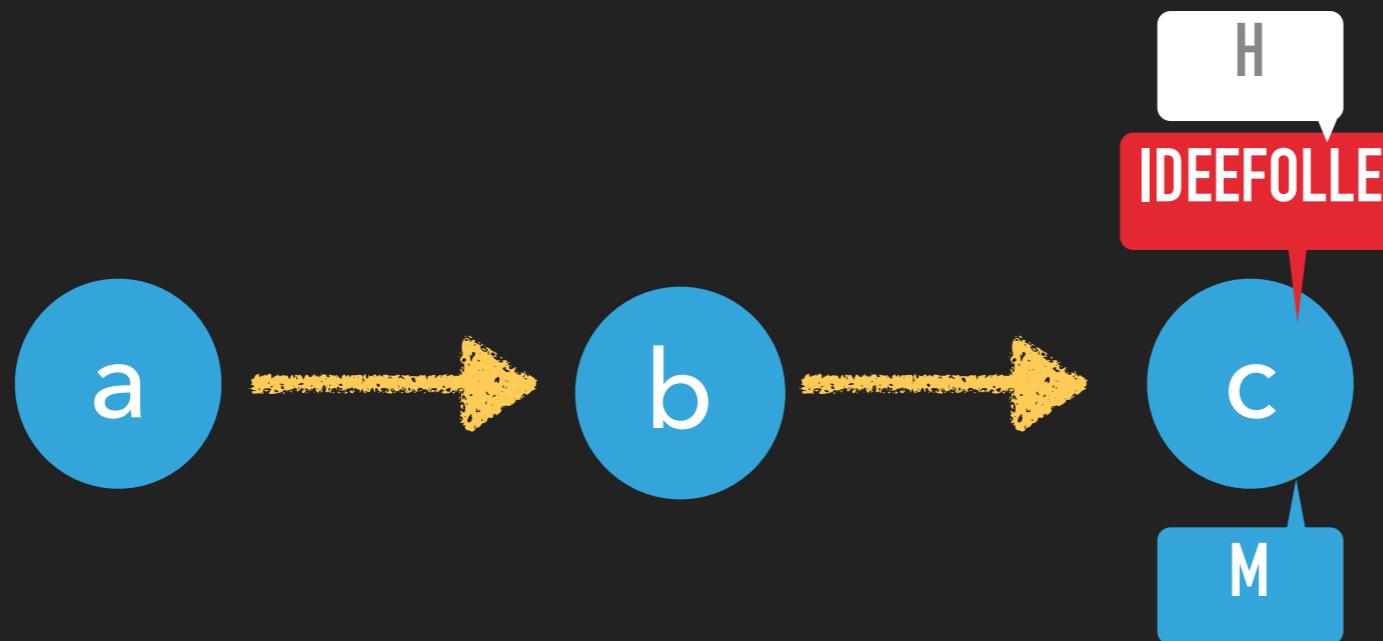


git checkout -b name [ref]



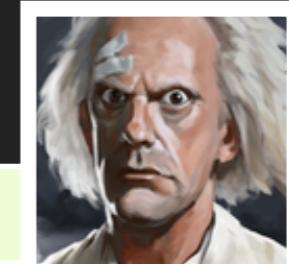
```
doc@labo$ git checkout -b ideefolle  
Switched to a new branch 'ideefolle'
```

```
doc@labo$ echo "cette commande est folle" >> commandes/ideefolle.txt  
doc@labo$ git add commandes/ideefolle.txt  
doc@labo$ git commit -m "ajout d'une idée folle"  
[ideefolle 759a26a] ajout d'une idée folle  
 1 file changed, 1 insertion(+)  
 create mode 100644 commandes/ideefolle.txt
```

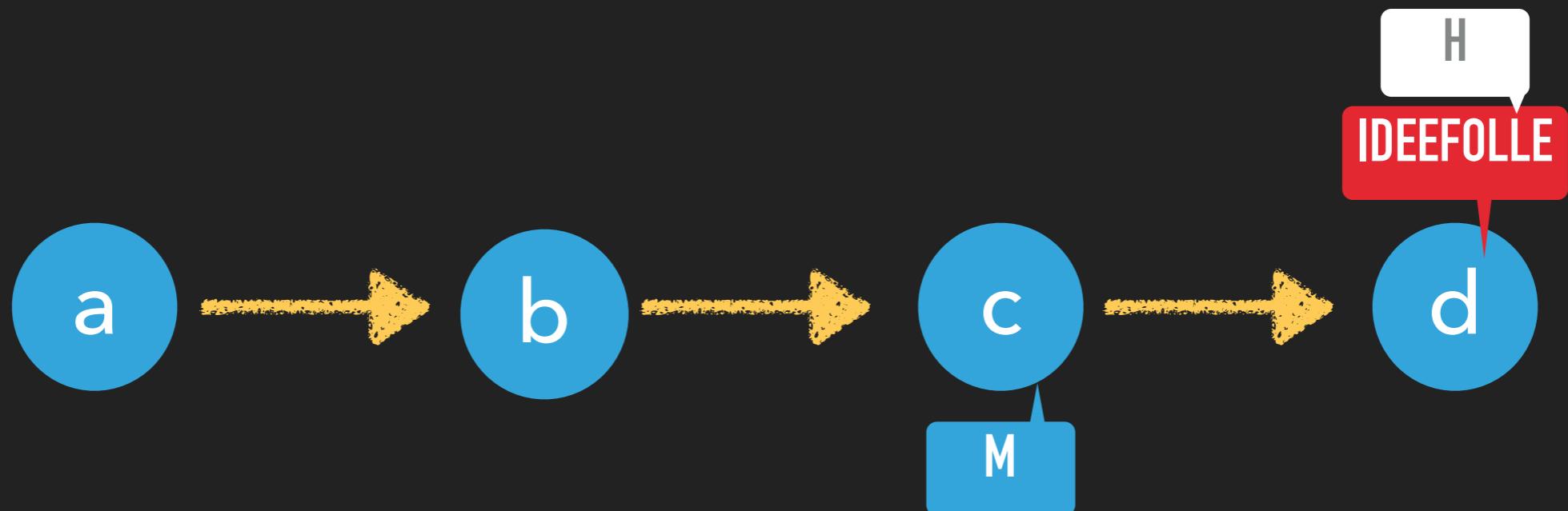


git checkout -b name [ref]

```
doc@labo$ git checkout -b ideefolle  
Switched to a new branch 'ideefolle'
```



```
doc@labo$ echo "cette commande est folle" >> commandes/ideefolle.txt  
doc@labo$ git add commandes/ideefolle.txt  
doc@labo$ git commit -m "ajout d'une idée folle"  
[ideefolle 759a26a] ajout d'une idée folle  
 1 file changed, 1 insertion(+)  
 create mode 100644 commandes/ideefolle.txt
```



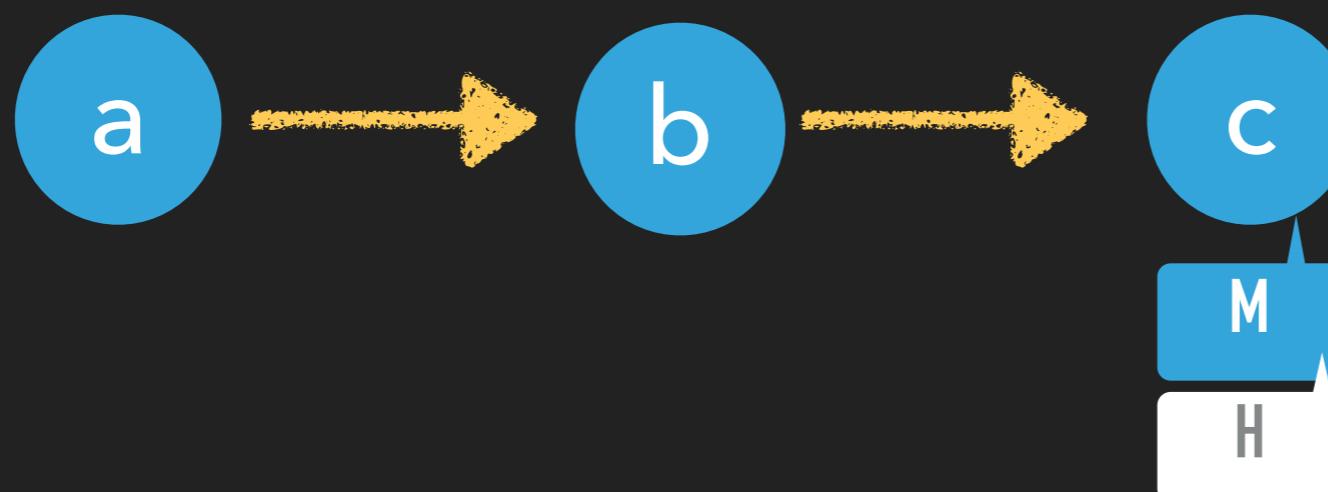
git reset [ref] -- file...

- ▶ Reset le ou les fichiers dans l'état où ils étaient à ref

- ▶ Ex:

```
git reset a -- README.md
```

```
git reset HEAD~~ -- README.md
```



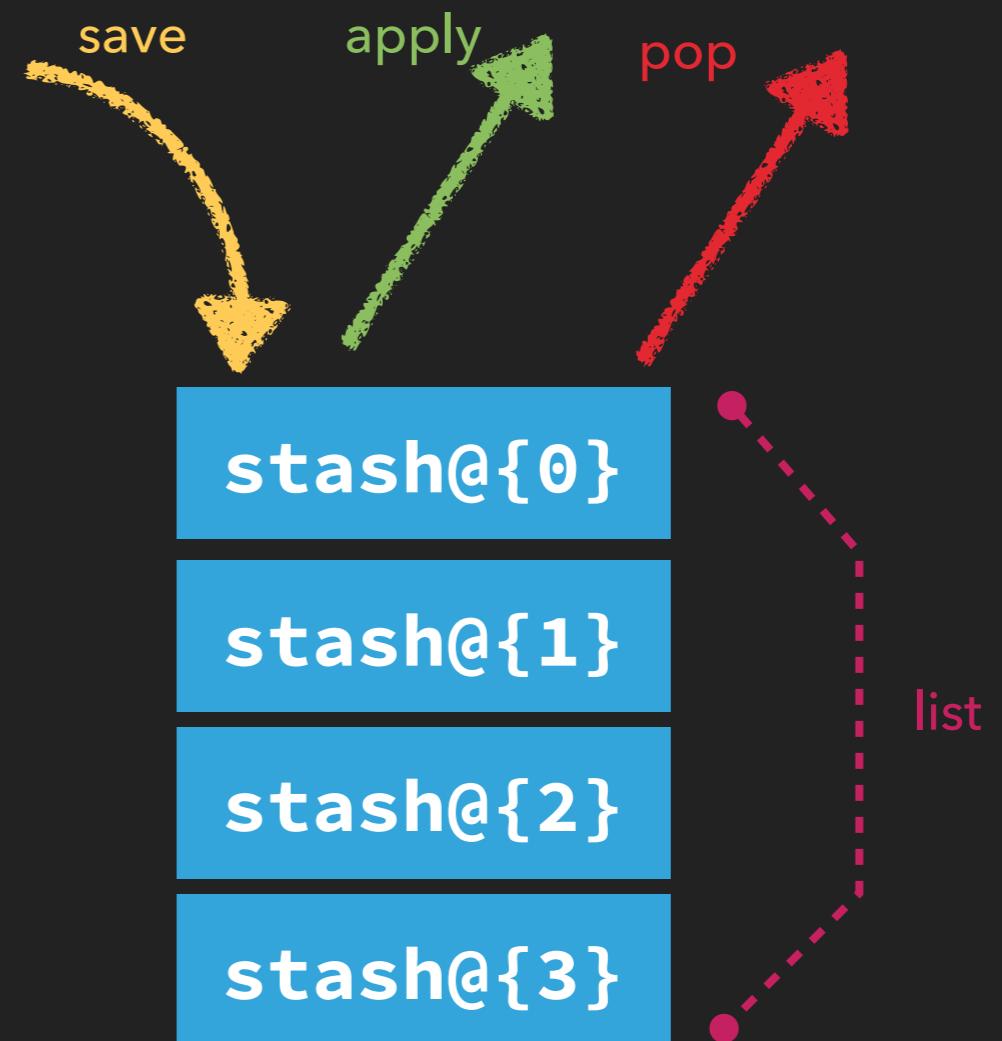
git bundle bundlefile fromref..toref

- ▶ TBC

git stash [save (--all) [<message>] [pop|apply|list]

- ▶ Permet de:

- ▶ **save**: sauver les modifications en cours dans un stash
- ▶ **apply**: ré-appliquer les modifications du dernier stash
- ▶ **pop**: ré-applique les modifications du dernier stash ET le supprime
- ▶ **list**: liste les stash en cours
- ▶ Utile quand on veut mettre ses modifications de coté avant un merge/rebase/checkout





INFOZESK@GMAIL.COM

MERCI!

CREDITS

- ▶ Icons made on <http://flaticons.net/>
- ▶ <https://git-scm.com/>
- ▶ A successful git branching model nvie.com
- ▶ stock images from <https://stocksnap.io/>

CREDITS

- ▶ Marty avatar <http://paulorocker.deviantart.com>
- ▶ Doc avatar: impossible de trouver l'auteur de l'image, mais merci!

