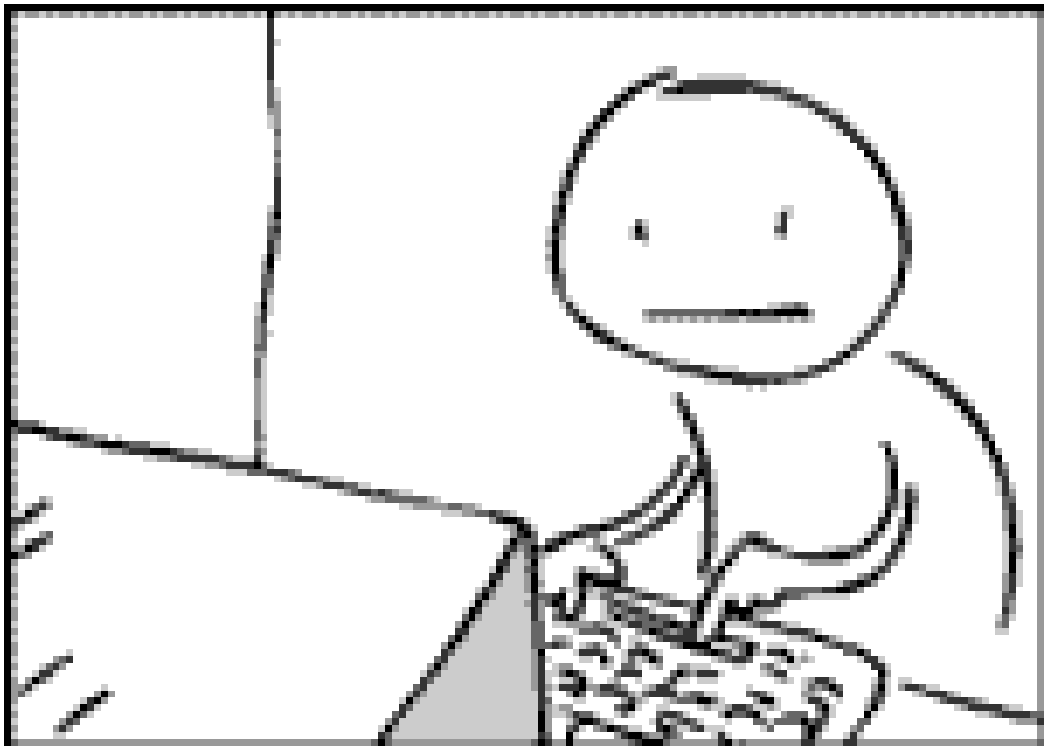
git en pratique

WTF ! I don't get how it works !



Sommaire

hupstream

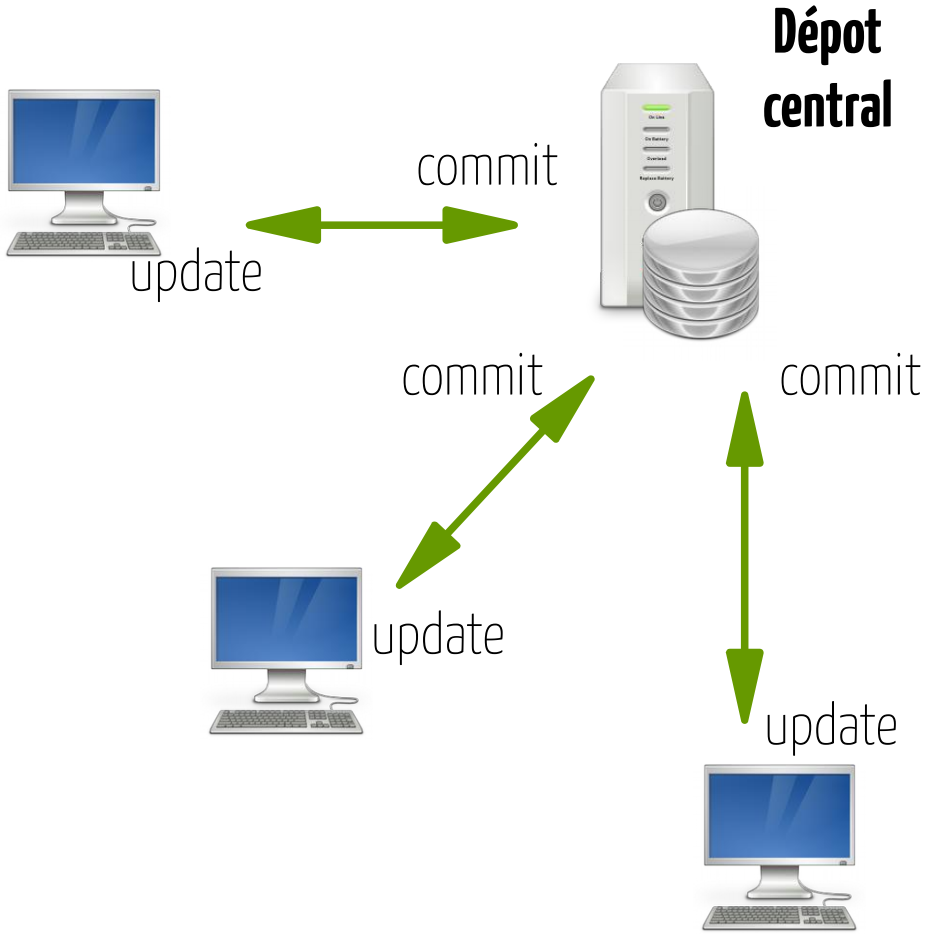
1. Retour sur les entrailles de git

hupstream

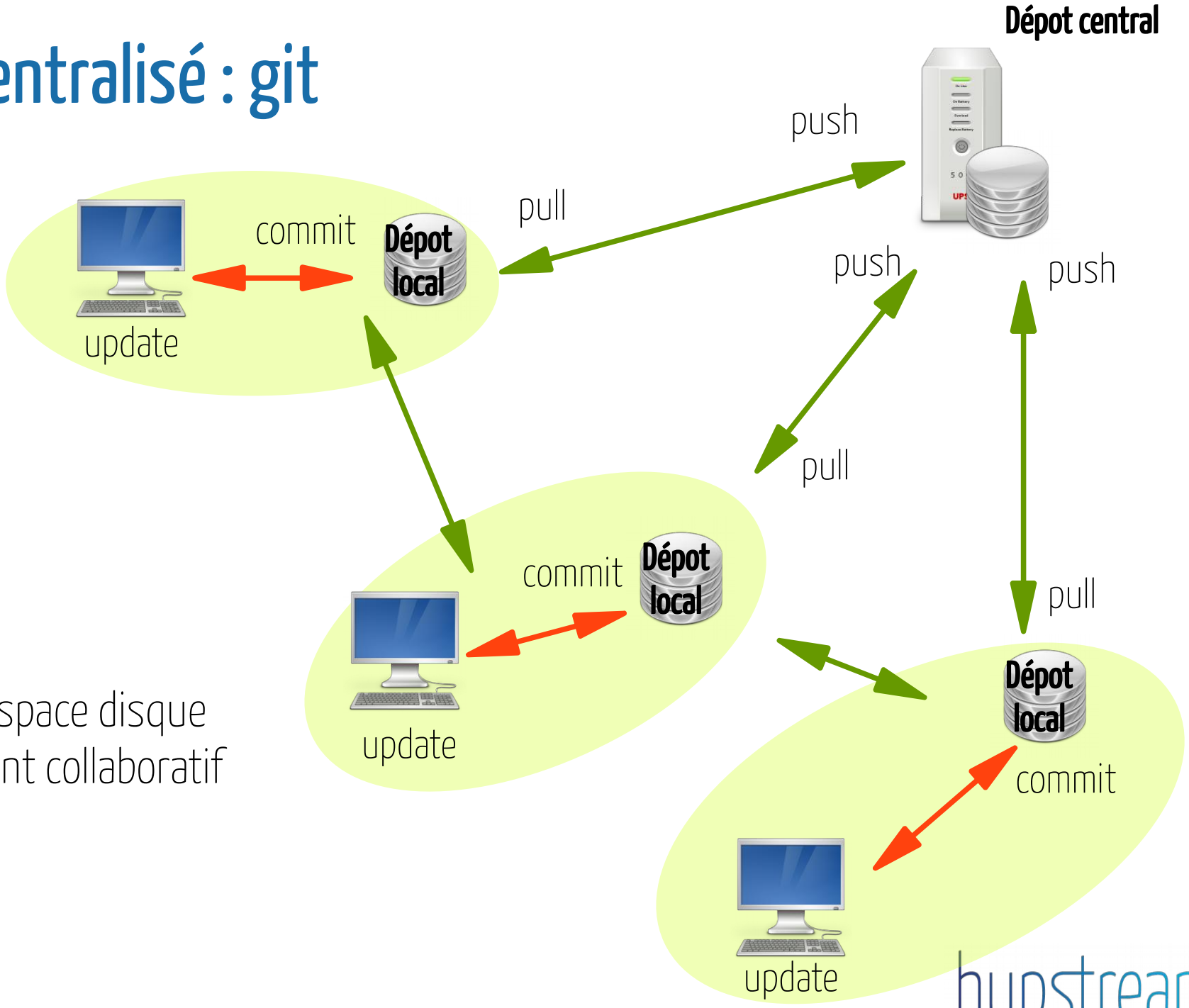
VCS centralisé : svn, cvs

- compréhension facilitée

- disponibilité (réseau)
- single point of failure
- gestion de l'espace disque



VCS décentralisé : git

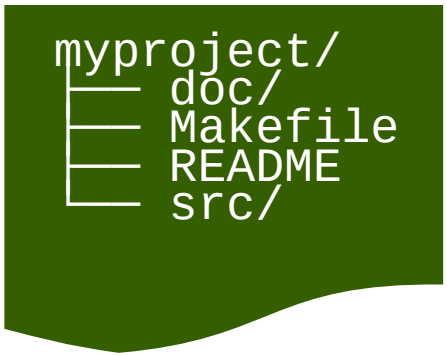


- disponibilité
- redondance
- gestion de l'espace disque
- développement collaboratif

- complexité

hupstream

Les 3 zones locales de git



Dépot git

- Appelée aussi base de données ou arbre git
- Stocke les data et metadata jusqu'au dernier commit

Zone de cache

- Appelée aussi index
- Stocke un snapshot des changements

Répertoire de travail

- Le disque local
- Modifications et suppressions de fichiers

Les 3 zones locales de git

Zone de travail

Fichier non suivi

`untracked`

Zone de cache

Fichier suivi
modifié
Nouveau fichier

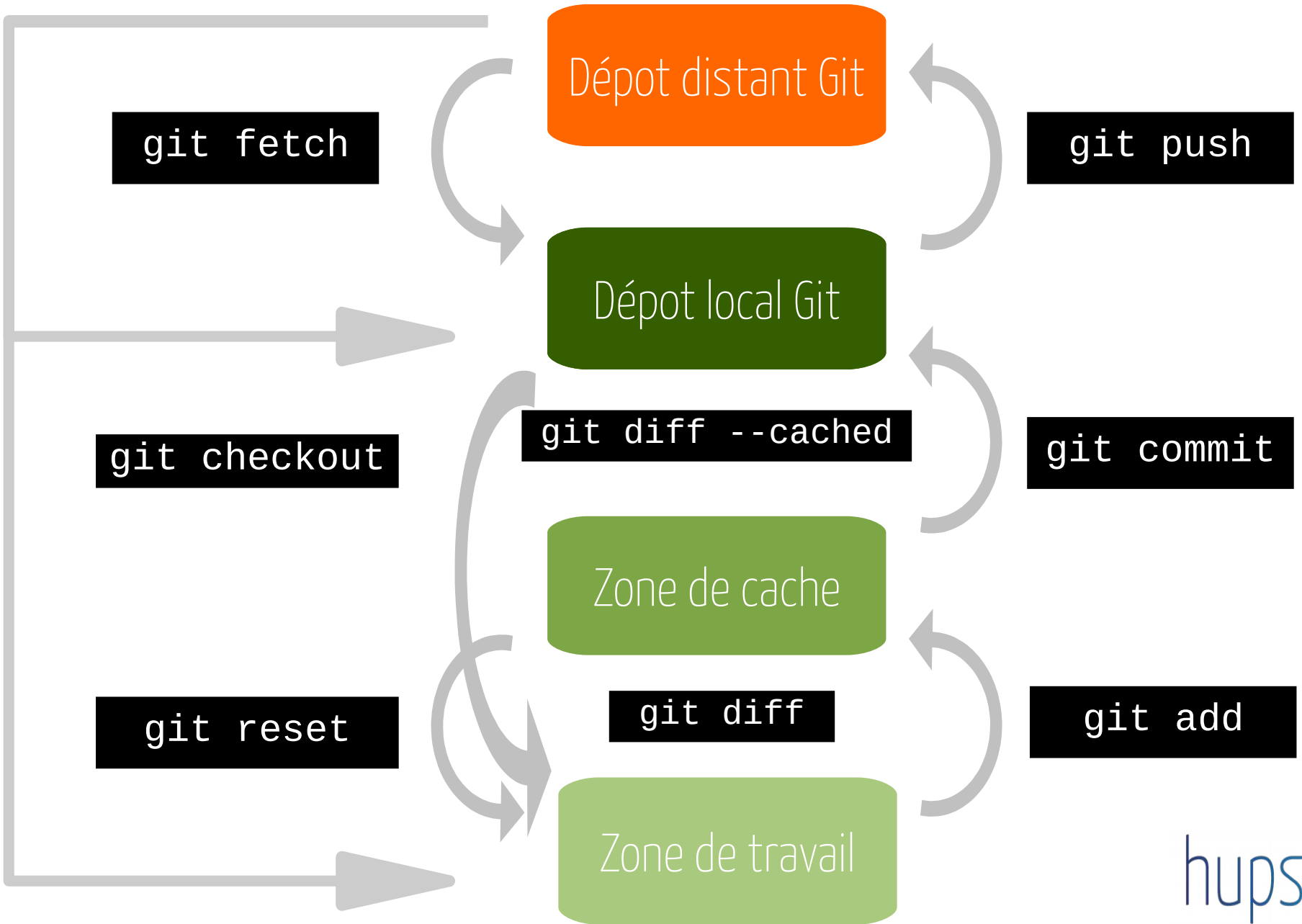
`staged`

Dépot local Git

Fichier suivi
non modifié

`tracked`

Commandes et zones de travail

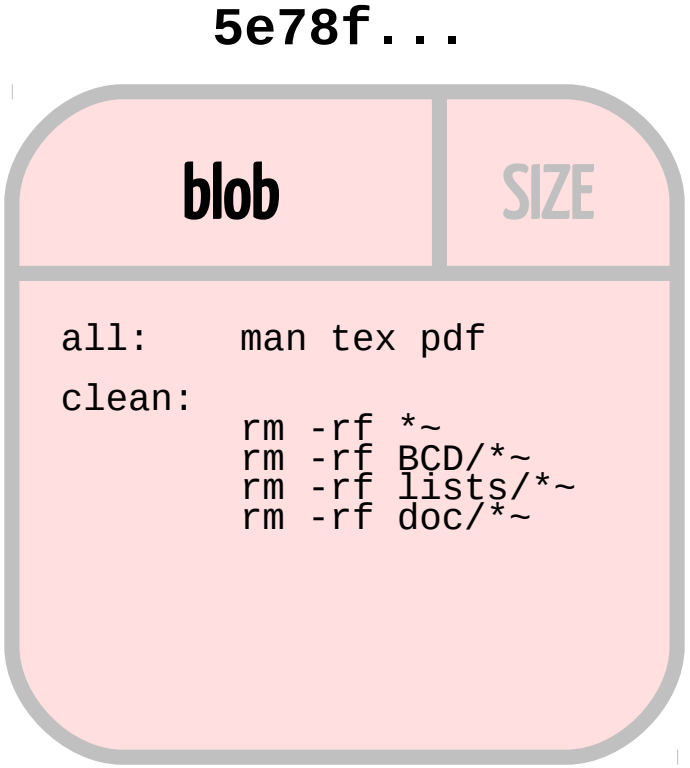


hupstream

Objets Git

Blobs

- Stocke le contenu d'un fichier
- Référencé par un SHA1
- 2 fichiers avec le même contenu dans un dépot ont le même SHA1



hupstream

Objets Git

Arbres

- Liste des pointeurs vers les blobs (fichiers) et/ou trees (répertoires), leurs noms, leurs ID, leurs permissions
- Référencé par un SHA1, modifié lors du moindre changement d'un des composants de l'arbre

12af7...

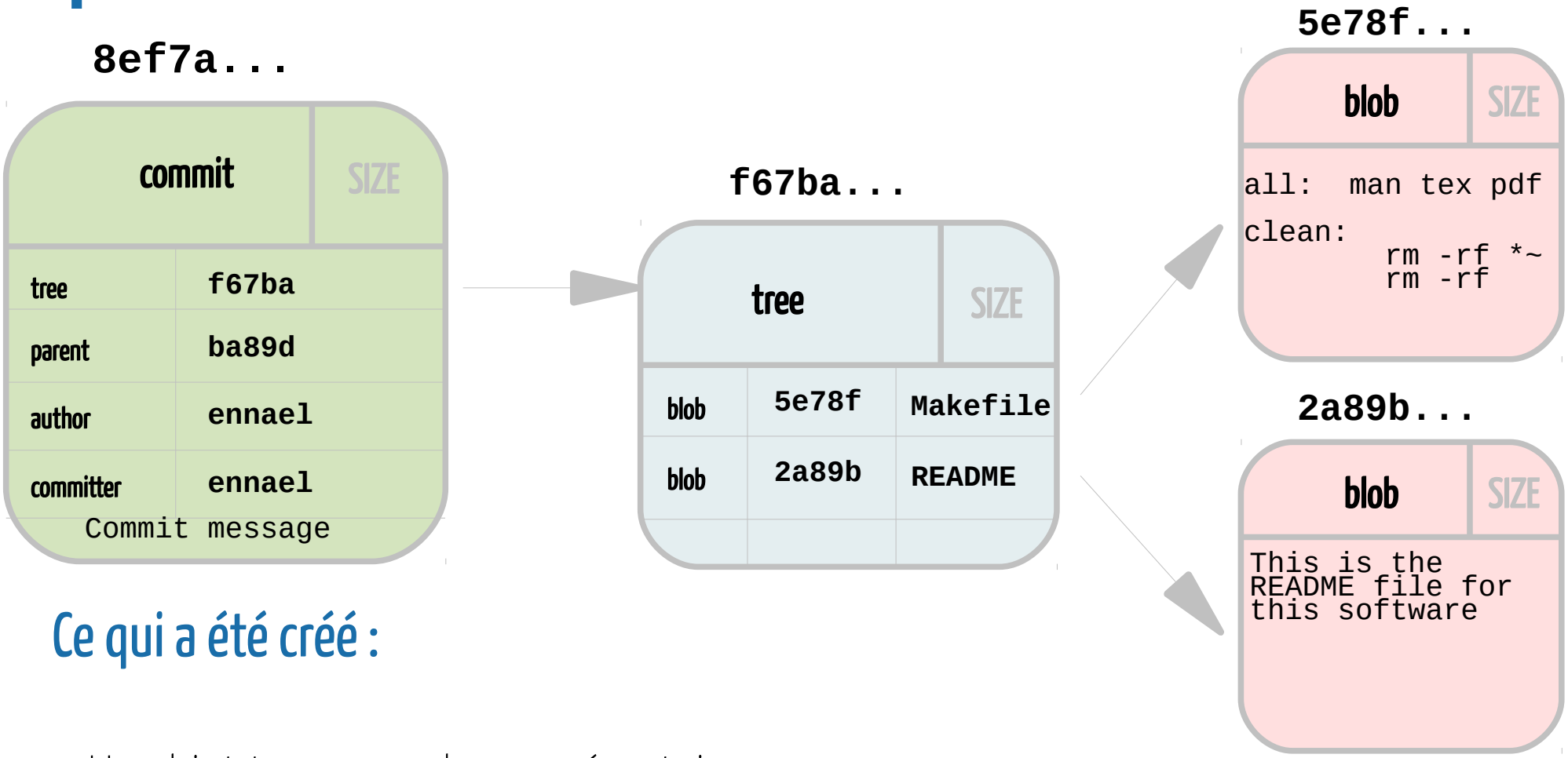
tree		SIZE
blob	5e78f	Makefile
blob	2a89b	README
tree	cd20c	src
blob	36ca9	config

Objets Git

Commits

- Référence l'état du dépot à un moment donné (tree)
- Contient le parent (état précédent)
- Contient un message décrivant la différence
- Référencé par un SHA1

Les commits ne sont pas des deltas



Ce qui a été créé :

- Un objet tree pour chaque répertoire
- Un objet blob pour chaque fichier
- Un objet commit pour pointer à la racine

hupstream

Objets Git

Définir le type d'un objet

```
git cat-file -t <SHA1>
```

Afficher le contenu d'un objet

```
git cat-file -p <SHA1>
```

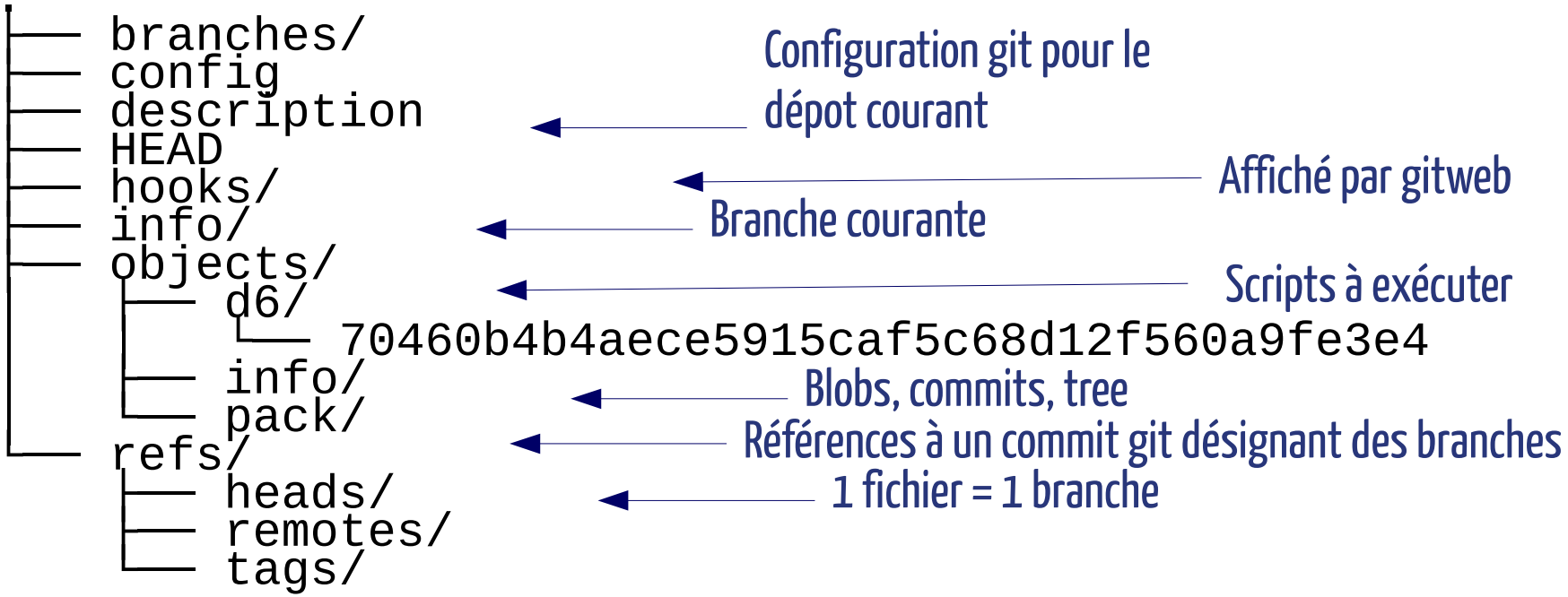
La zone de cache

Une zone temporaire pour préparer les commits

- création des blobs relatifs aux modifications indexées
- liste ordonnée des chemins de fichiers + permission + SHA1 blobs
- sera utilisée pour préparer le futur commit (construction du tree)

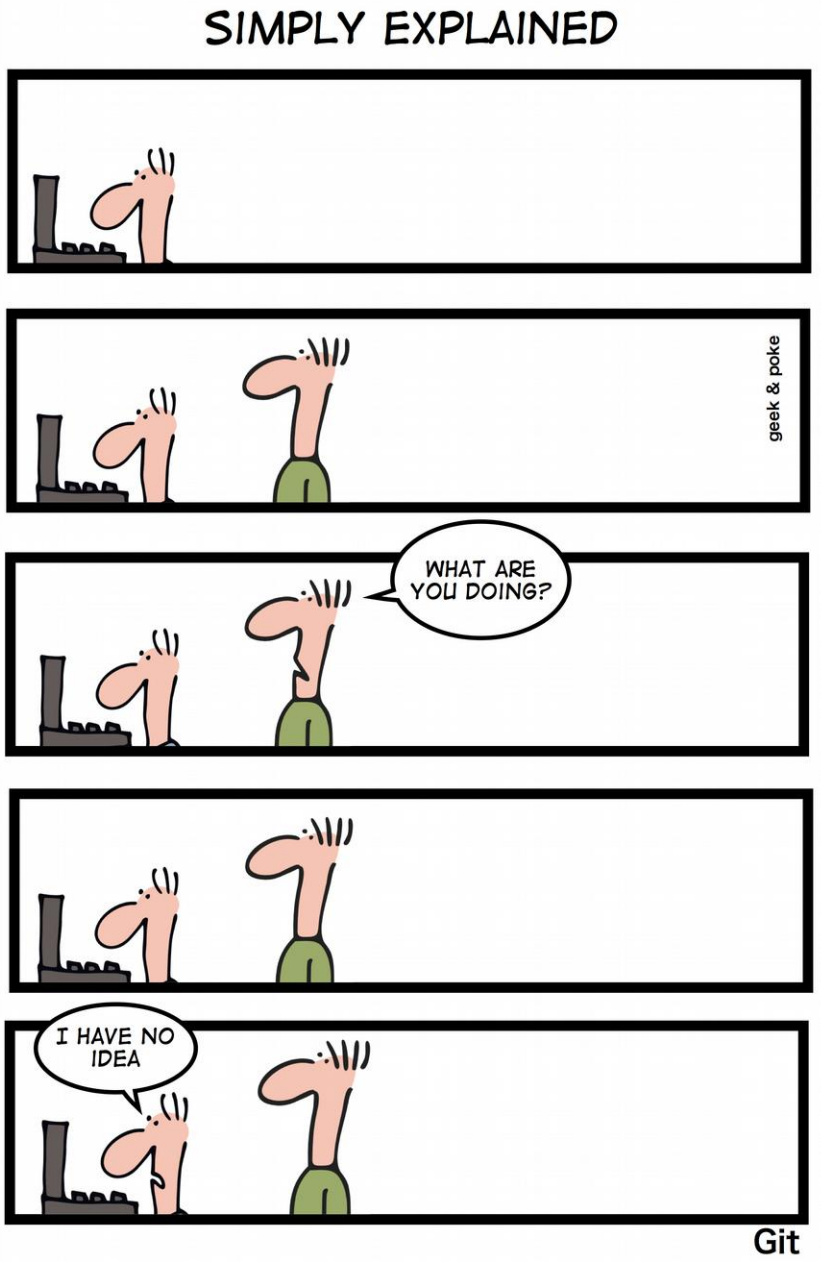
```
$ git ls-files --stage
100644 e8b7c915cf81135b25e2662a72356339ae5cb774 0       fic1
```

Arborescence Git



hupstream

Simply explained



2. Configurer et optimiser son environnement git

hupstream

Configurer votre environnement git

Personnaliser son environnement pour tous les dépots

- Linux, MAC OS

```
~/.gitconfig
```

- Windows

```
C:\Documents and Settings\$USER
```

ou

```
C:\Users\$USER
```

Lister les paramètres de configuration existant

```
git config –-global --list
```

hupstream

Configurer votre environnement git

Personnaliser son environnement pour un dépot donné

- Fichier de configuration

```
<depot>/.git/config
```

- Lister les paramètres de configuration existant

```
git config --list
```

Modifier votre environnement

Structure du fichier de configuration

```
[section]
    param1 = <valeur>
...
```

Modification avec git config

```
git config section.param1 <valeur>
```

Configurer son environnement git

Votre nom

```
[user]
    email = firstname.lastname@hupstream.com
    name = Firstname Lastname
```

Colorisation, options avancées

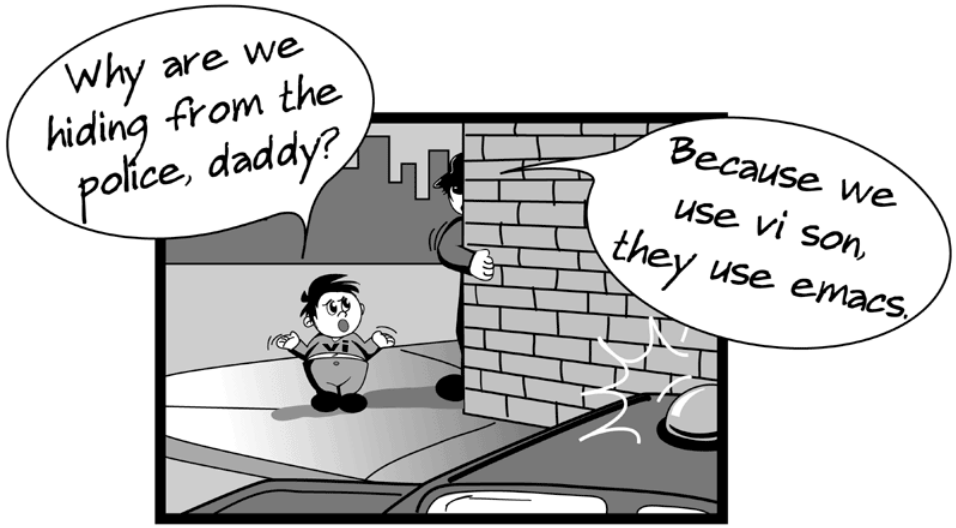
visualisation des divisions de patches et report des espaces de fin

```
[color]
    ui = auto
[color "diff"]
    whitespace = red reverse bold
    meta = white blue
```

Optimiser son environnement git

Editeurs et pagers favoris

```
[core]
    editor = vim
    pager = less
```



hupstream

Création d'alias de commandes

Personnaliser les commandes git

```
[alias]
        <nom_alias> = <attributs de commande>
```

Créer un alias

```
git config alias.<nom_alias> <attributs de commande>
```

Optimiser son environnement git

Ignorer des fichiers globalement ou pour un dépot

- à la racine du répertoire utilisateur
- à la racine du dépot

```
[ennael@localhost test (master)]$ cat ~/.gitignore
*~
*.[oa]
[ennael@localhost test (master)]$ cat <depot>/.gitignore
*~
*.[oa]
```

hupstream

3. Commandes de base... et plus

hupstream

Créer un dépot local

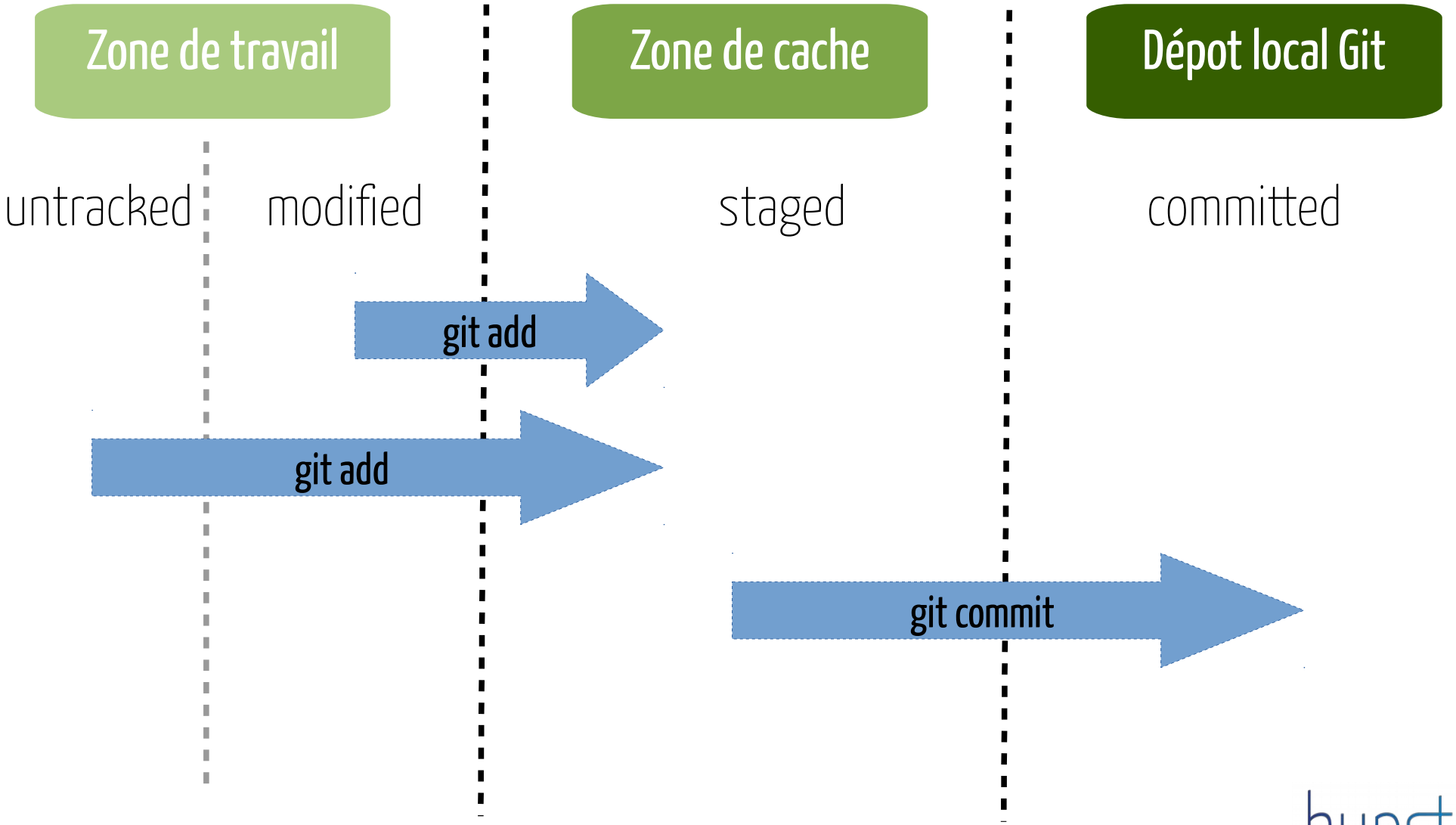
Création du dépot

```
git init [--bare] <répertoire>
```

`--bare` : dépot serveur

le dépot n'est pas adhérent au système de fichiers (renommage, déplacement)

Le processus de commit



Ajouter à la zone de cache

Indexation des modifications

```
git add <fic1>...<ficn>
```

- ajouter de nouveaux fichiers non encore trackés
- ajouter des fichiers trackés et modifiés

A éviter

```
git add *
```

Finaliser le commit

Commit des modifications

```
git commit [-m <message>]
```

- utilise le contenu de la zone de cache pour réaliser le commit

A éviter

```
git commit -a
git commit *
```

Finaliser le commit

Etat du dépot

```
git status [-s] [-b branche]
```

- état de la zone de travail
- état du cache
- conflit
- solutions

```
# Modifications qui seront validées :
#   (utilisez "git reset HEAD <fichier>..." pour désindexer)
#
#       modifié :   fic1
# Fichiers non suivis:
#   (utilisez "git add <fichier>..." pour inclure dans ce qui
# sera validé)
#
#       essai
```

Supprimer un fichier

La commande rm

```
rm <fichier>
git add <fichier>
git commit
```

La commande git rm

```
git rm <fichier>
git commit
```

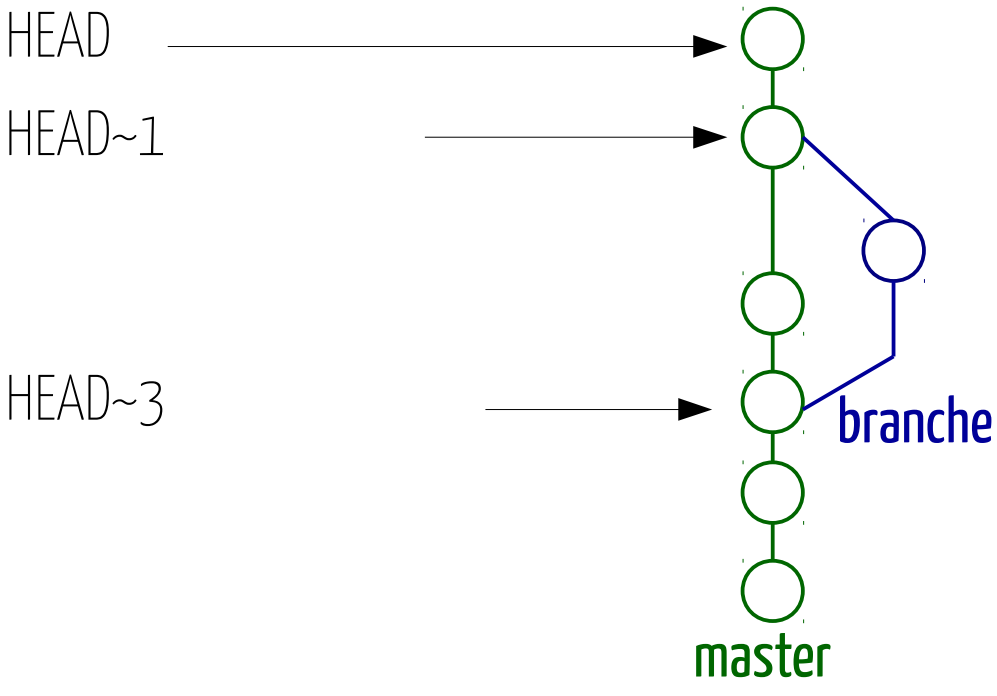
Les révisions git

HEAD

- référence vers le dernier commit de la branche courante

[HEAD]~x ou [SHA1]~x

- désigne le xième commit avant HEAD, sans tenir compte des opérations de merge



hupstream

Filtrage avancé sur les logs git

Afficher les logs en arbre

```
[ennael@localhost BCD (master)]$ git log --graph --all --oneline
*   cdd63ee Merge branch 'test'
|\
| * a1a981d fix typo
* | 9d0e767 fix path
* | 215095c create a new variable for tests
* |   83173a7 Merge branch 'test'
|\ \
| |/
| * c0d17bc add new variable for test
* | 6e20954 add new variable for new tests
|/
* fc34af2 remove unneeded list
* ac20415 add comment on last modification (user test)
```

hupstream

Recueillir de l'information

Etat des modifications

```
git diff [--cached] [--name-status] [SHA1...] [fichier]
```

- différence répertoire de travail / zone de cache
- différence zone de cache / commit le plus récent
- différence entre 2 commits

```
[ennael@localhost test (master)]$ git diff --cached
diff --git a/include/configs/mx53ard.h b/include/configs/mx53ard.h
index fdb4134..25a4630 100644
--- a/include/configs/mx53ard.h
+++ b/include/configs/mx53ard.h
@@ -31,6 +31,8 @@
 #define CONFIG_MXC_GPIO

+#define CONFIG_CMD_BOOTZ
```

Recueillir de l'information

Suivi des modifications par auteur

```
git blame [-L x,+y] <fichier> [<SHA1>]
```

- annote les lignes d'un fichier avec modifications par auteur
- sur l'ensemble du fichier, une ou plusieurs lignes spécifiées, un intervalle spécifié

```
[ennael@localhost u-boot (master)]$ git blame -L 1,+3 Makefile
7ebf7443 (wdenk         2002-11-02 23:17:16 +0000 1) #
eca3aeb3 (Wolfgang Denk 2013-06-21 10:22:36 +0200 2) # (C) Copyright 2000-2013
7ebf7443 (wdenk         2002-11-02 23:17:16 +0000 3) # Wolfgang Denk, DENX Software
Engineering, wd@denx.de.
```

Consulter une ancienne version d'un fichier

git show

vérifier le contenu d'un objet git sans exécuter de checkout au préalable (tag, commit, blob, tree)

```
git show <SHA1>:<fichier>
```

```
[ennael@localhost u-boot (master)]$ git show HEAD~1:Makefile
#
# (C) Copyright 2000-2013
# Wolfgang Denk, DENX Software Engineering, wd@denx.de.
#
# SPDX-License-Identifier:      GPL-2.0+
#

VERSION = 2014
PATCHLEVEL = 04
```

Récupérer un fichier supprimé de la zone de travail

git checkout

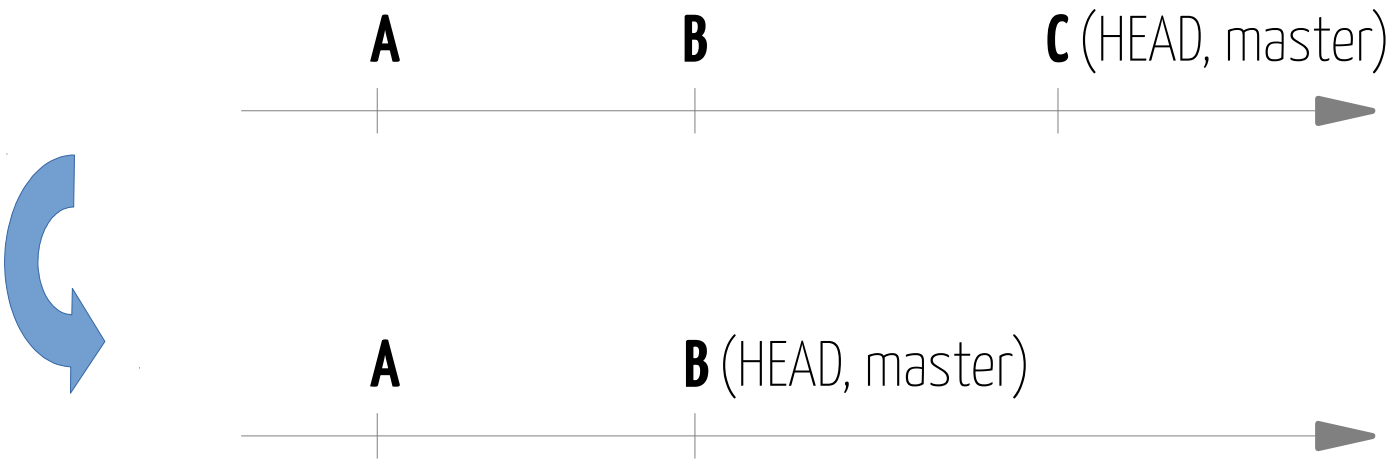
récupérer un fichier dans une version donnée et le copier dans la zone de travail

```
git checkout <fichier> [<SHA1>]
```

Les différents modes de git reset

Ce que fait git reset

```
git reset [options] B
```



- Déplacement de HEAD
- L'option utilisée détermine le traitement qui sera appliqué au commit, à la zone de travail et la zone de cache pendant l'exécution de git reset

Les différents modes de git reset : mixed

- Mode par défaut de git reset
- Déplace la zone de cache vers la zone de travail
- Ne touche pas les modifications présentes en zone de travail
- Le commit reseté est transféré en zone de travail

```
git reset [--mixed] [SHA1]
```

Pourquoi l'utiliser ?

Indexation d'une modification erronée : conserver le contenu afin de le corriger et le commiter à nouveau

Les différents modes de git reset : soft

- Le commit reseté est ajouté à la zone de cache
- Aucun impact sur la zone de travail

```
git reset [--soft] SHA1
```

Pourquoi l'utiliser ?

Contenu du commit erroné

Les différents modes de git reset : hard

- Modifie la zone de travail et la zone de cache : toutes les modifications sont supprimées

```
git reset [--hard] [SHA1]
```

Pourquoi l'utiliser ?

Revenir en arrière sur des modifications erronées pour se positionner sur l'étape précédant les modifications

Attention : vous pouvez perdre votre travail car la commande modifie votre zone de travail

Nettoyer sa zone de travail

- Supprimer tous les fichiers non trackés

```
git clean
```

Supprime uniquement les fichiers non encore trackés par git, sauf les fichiers explicitement ignorés (.gitignore)

Staging partiel sur un fichier

Décomposer son commit en petits blocs, pour affiner le contenu du commit :

```
git add -p <file>
```

- y mettre en attente ce bloc
- n ne pas mettre en attente ce bloc
- q quitte ; ne met pas en attente ce bloc ni aucun de ceux restant
- a mettre en attente ce bloc et tous les blocs suivant du fichier
- d ne met pas en attente ce bloc ni aucun bloc suivant du fichier
- s divise le bloc courant en plus petits blocs
- e édite manuellement le bloc courant

Staging partiel sur un fichier

Identifier le ou les bloc s à conserver pour le commit à venir

```
Split into 3 hunks.
...
-#export:
-#	svn export -q -rBASE . $(NAME)-$(VERSION)
 export:
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y
@@ -55,3 +52,6 @@
 export:
-	@cd .. ; git archive --prefix=$(NAME)-$(VERSION)/ HEAD * |
xz >images/$(NAME)-$(VERSION).tar.xz;
+	svn export -q -rBASE . $(NAME)-$(VERSION)
+#export:
+#	@cd .. ; git archive --prefix=$(NAME)-$(VERSION)/ HEAD * |
xz >images/$(NAME)-$(VERSION).tar.xz;
Stage this hunk [y,n,q,a,d,/,K,j,J,g,e,?]? y
...
+# use dist to manage upstream packages
 dist: cleandist export tar
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? n
```

hupstream

Utilisation des tags

Pourquoi utiliser des tags

- Identifier un commit spécifique en utilisant une étiquette au lieu d'un hash
- Permet de mettre en évidence un commit important dans l'historique – souvent utilisé pour mettre en avant une release
- 2 types de tags: lightweight (fichier plat), annotated (objet)

Objets Git : tag annoté

Tags

- Informations à propos du tag : auteur, pourquoi il a été créé, ...
- Référencé par un SHA1
- Les tags sont des références immuables

Comment tagger les commits

Lister les tags

```
git tag
```

Créer un tag lightweight

```
git tag  <tag> [<SHA1>]
```

Créer un tag annoté

```
git tag -a <tag> [<SHA1>] -m “message”
```

Supprimer un tag local

```
git tag -d <tag>
```

Création d'un tag annoté

```
git tag -a <tag> [<SHA1>] -m “message”
```

Exemple

```
git tag -a v1.0 -m ‘my message to explain this tag’
git show v1.0
tag v1.0
Tagger: Anne Nicolas <anicolas@hupstream.com>
Date:   Sun Jan 5 21:53:32 2014 +0100

my message to explain this tag

commit 9e0b67c59a1a20c1193d4eec1acd1a4e9b8506c1
Author: Anne Nicolas <anicolas@hupstream.com>
Date:   Sun Jan 5 01:23:57 2014 +0100

    update plan
...
```

hupstream

Visualiser les tags dans les logs

Ajouter les références à la commande git log

```
git log --decorate=full --oneline
```

```
[ennael@localhost u-boot (master)]$ git log –-decorate=full --oneline
3fe1a85 (HEAD, refs/remotes/origin/master, refs/remotes/origin/HEAD,
refs/heads/master) powerpc: hiddendragon: remove orphan board
7edb1f7 powerpc: debris: remove orphan board
2868f86 powerpc: kvme080: remove orphan board
0116f40 (tag: refs/tags/v2014.07-rc2) Prepare v2014.07-rc2
54c5d08 dm: rename device struct to udevice
```

Utilisation des tags : bonnes pratiques

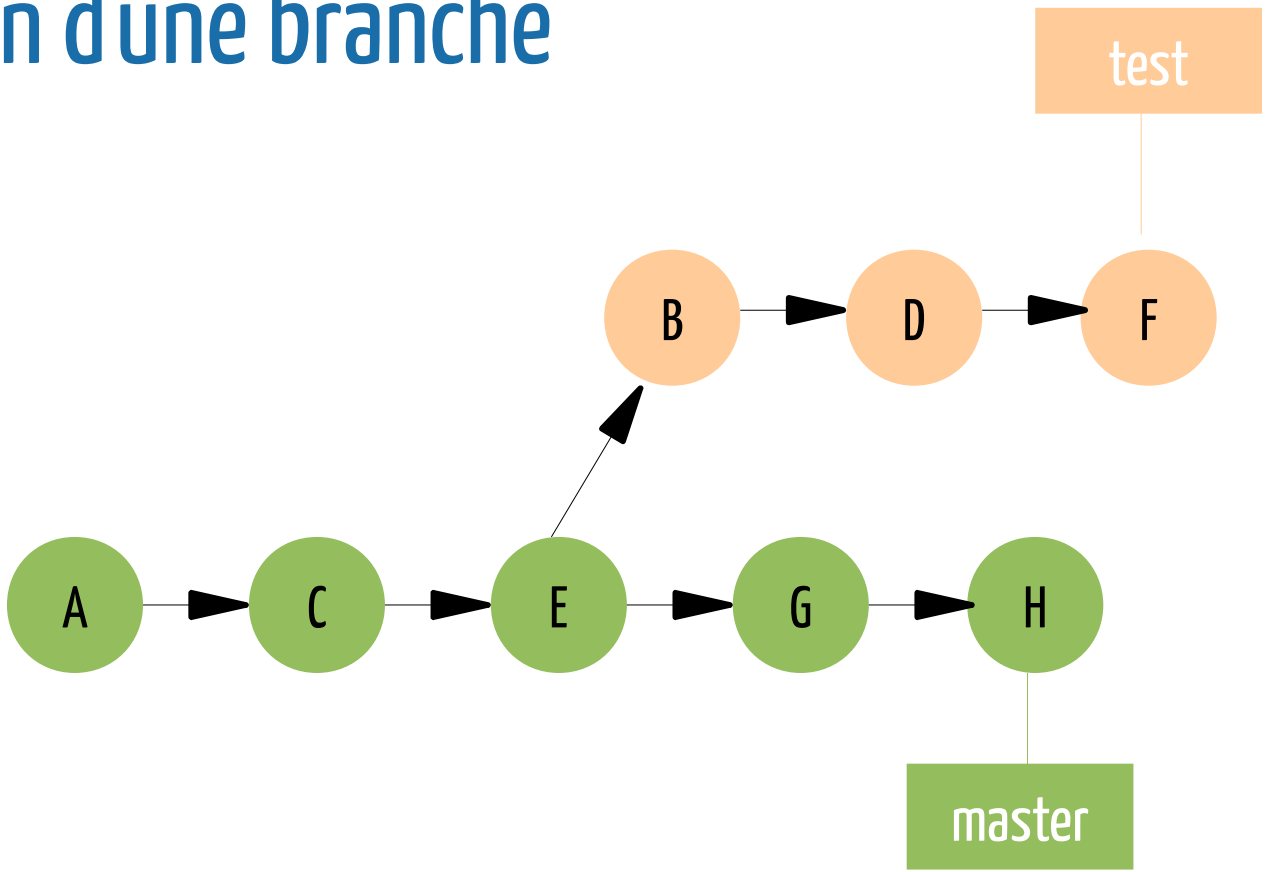
- Utiliser des chaînes compréhensibles
- Versioning : convention couramment utilisée

```
v.[major].[minor].[patch]
```

4. Collaboration, branches et conflits

hupstream

Définition d'une branche

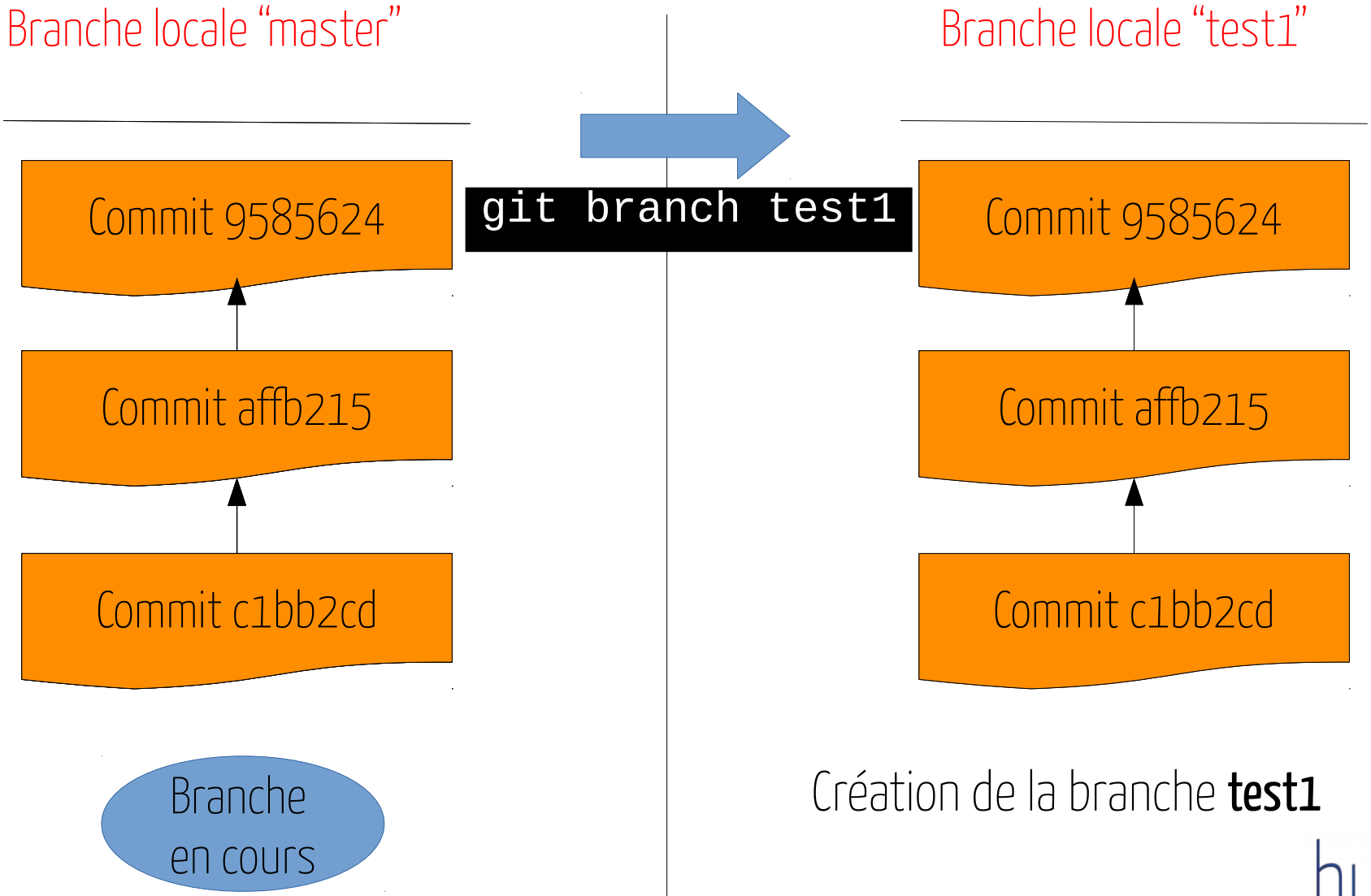


- **Une branche** est une référence au commit HEAD, contenu dans un fichier `.git/refs/heads/master`

- **HEAD** est une référence au dernier commit réalisé

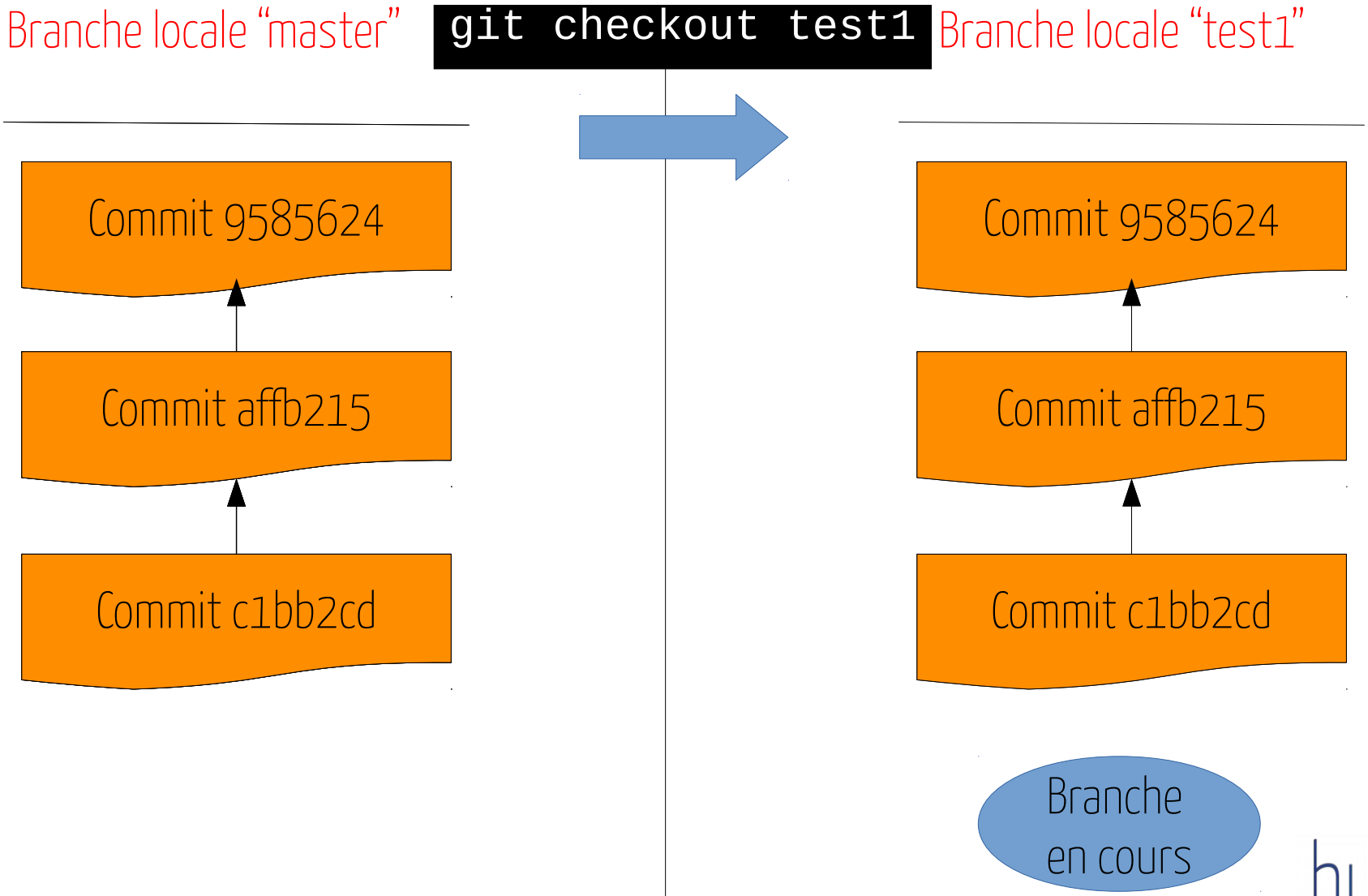
hupstream

Créer une branche

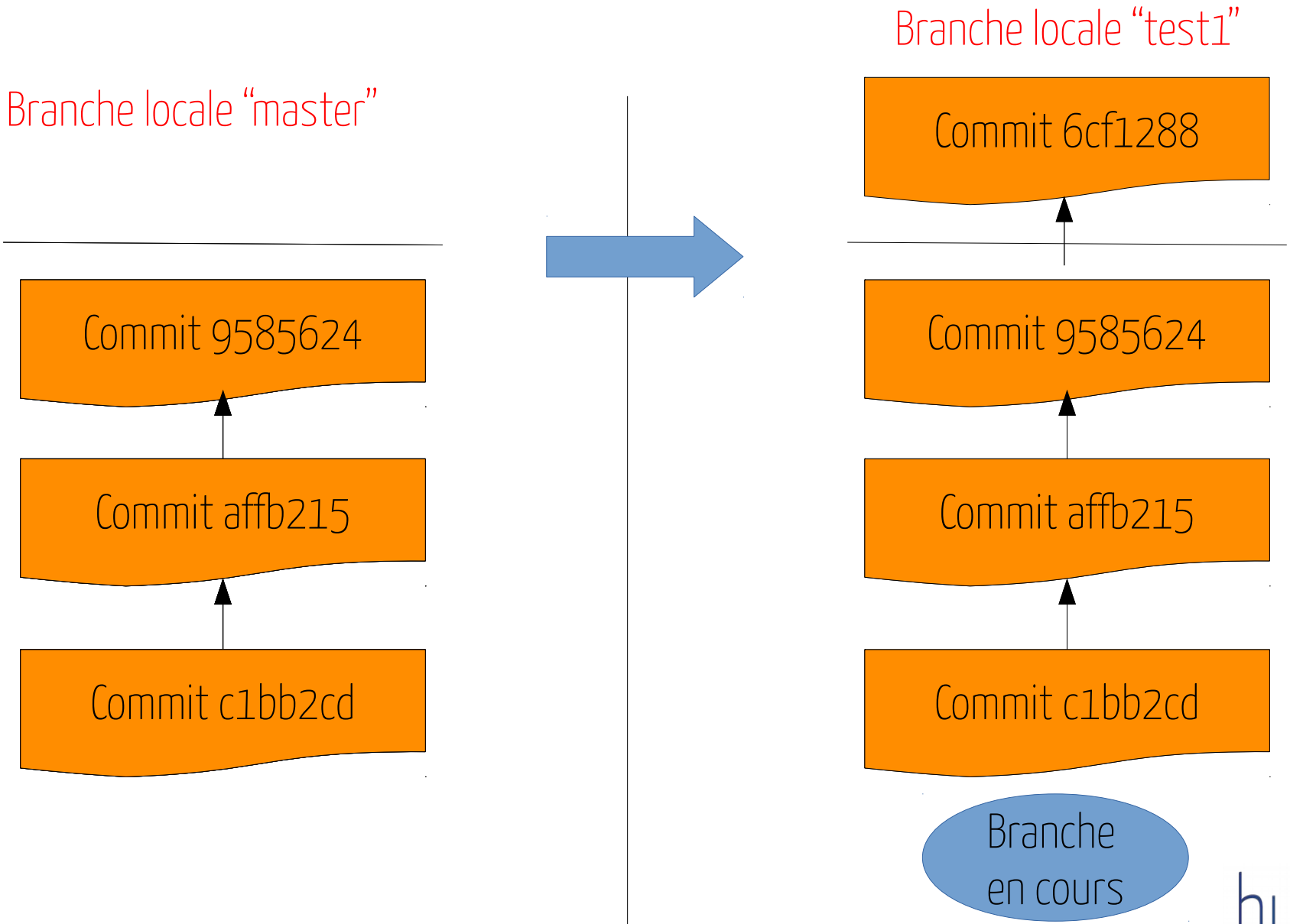


hupstream

Changer de branche



Génération d'un historique spécifique



hupstream

Commandes de gestion de branches

- Créer une branche

```
git branch <branche>
git checkout -b <branche>
```

- Lister les branches locales

```
git branch [-v]
```

- Renommer une branche

```
git branch -m <ancien> <nouveau>
```

- Supprimer une branche

```
git branch -d <branche>
```

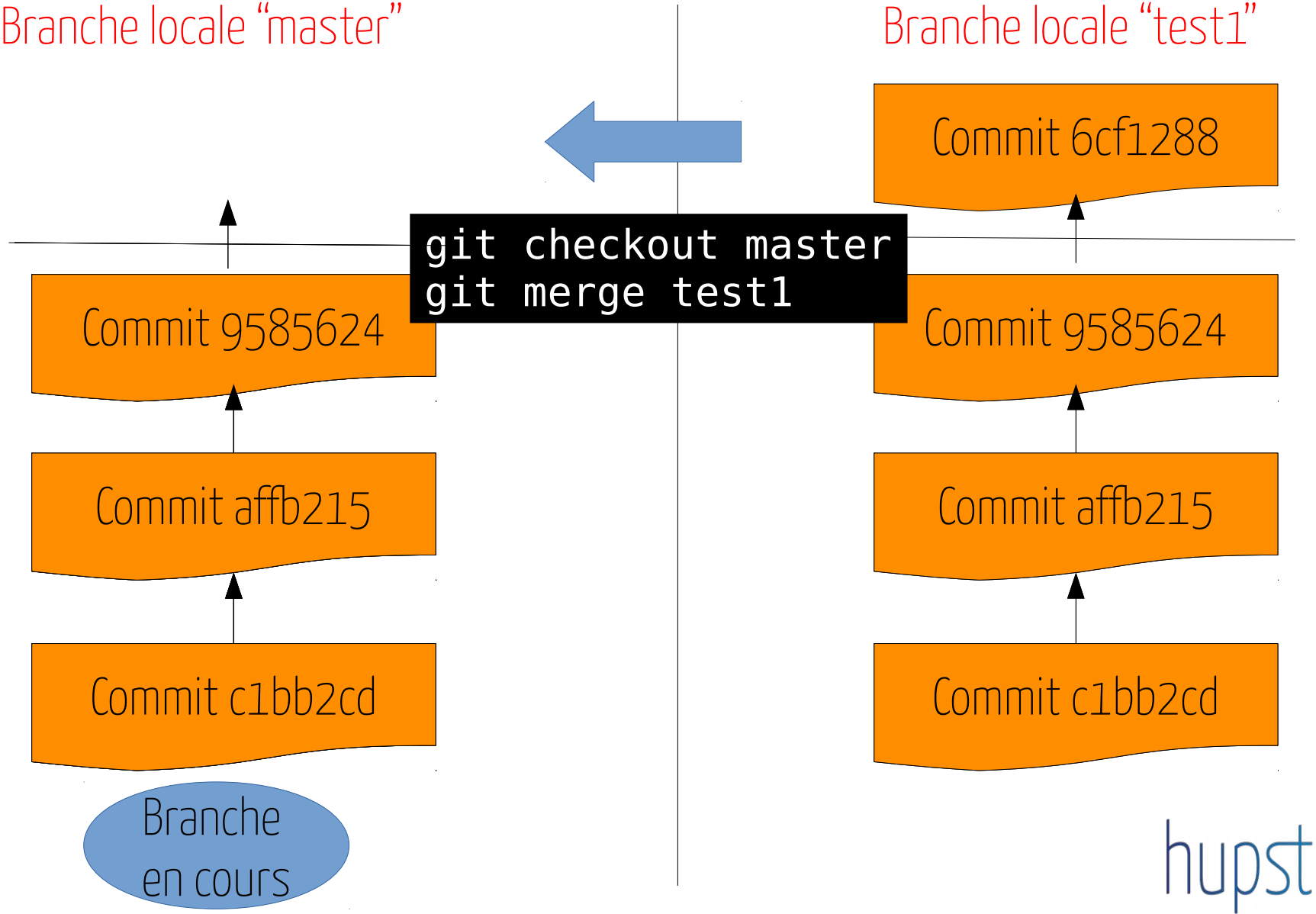
Visualiser les références

Utiliser git log pour visualiser les branches

```
 git log --all --oneline --graph --decorate=full
* 315be38 (refs/heads/experiment) Play icon for starter button
* 6448856 Centered footer
* cdfdf5d Further rebasing links in the footer
* 50794a5 (HEAD, refs/heads/master) Updated ex2 steps
* ebc1592 Link to GitHub teacher repo
* 9bcbd40 Centered footer
* 9040392 move to Licence (MIT) then moved to CC BY-NC-SA 3.0 license
| * 5c789b5 (refs/remotes/origin/master, refs/remotes/origin/HEAD)Update
| * 18b8a3c Link to GitHub repo
| * 44b8ae5 Revert "NO INCOMING LINKS!"
```

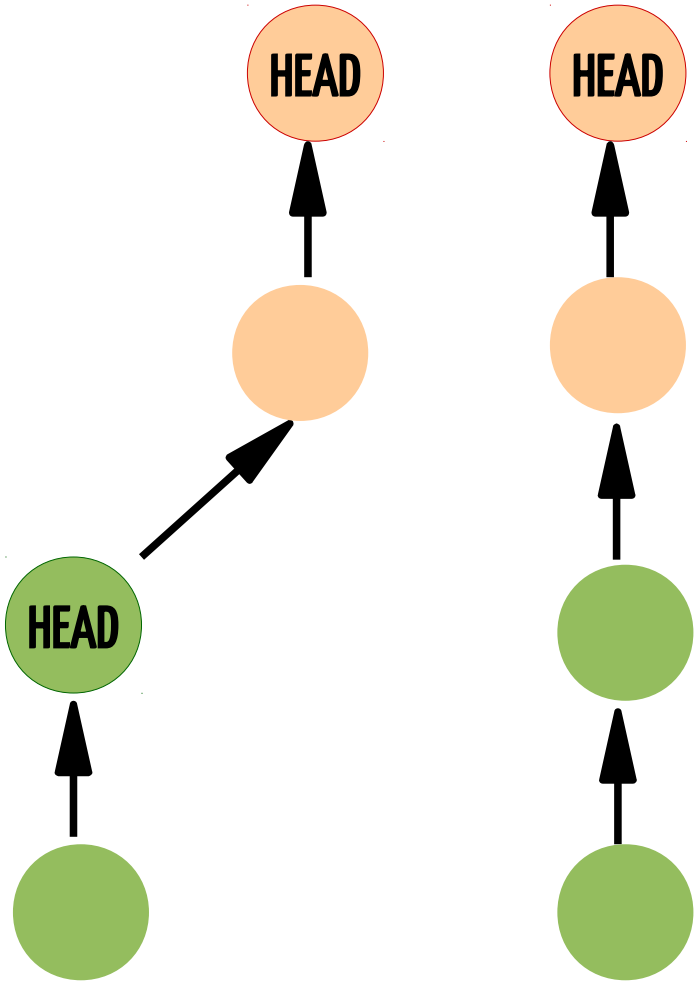
hupstream

Fusionner des branches

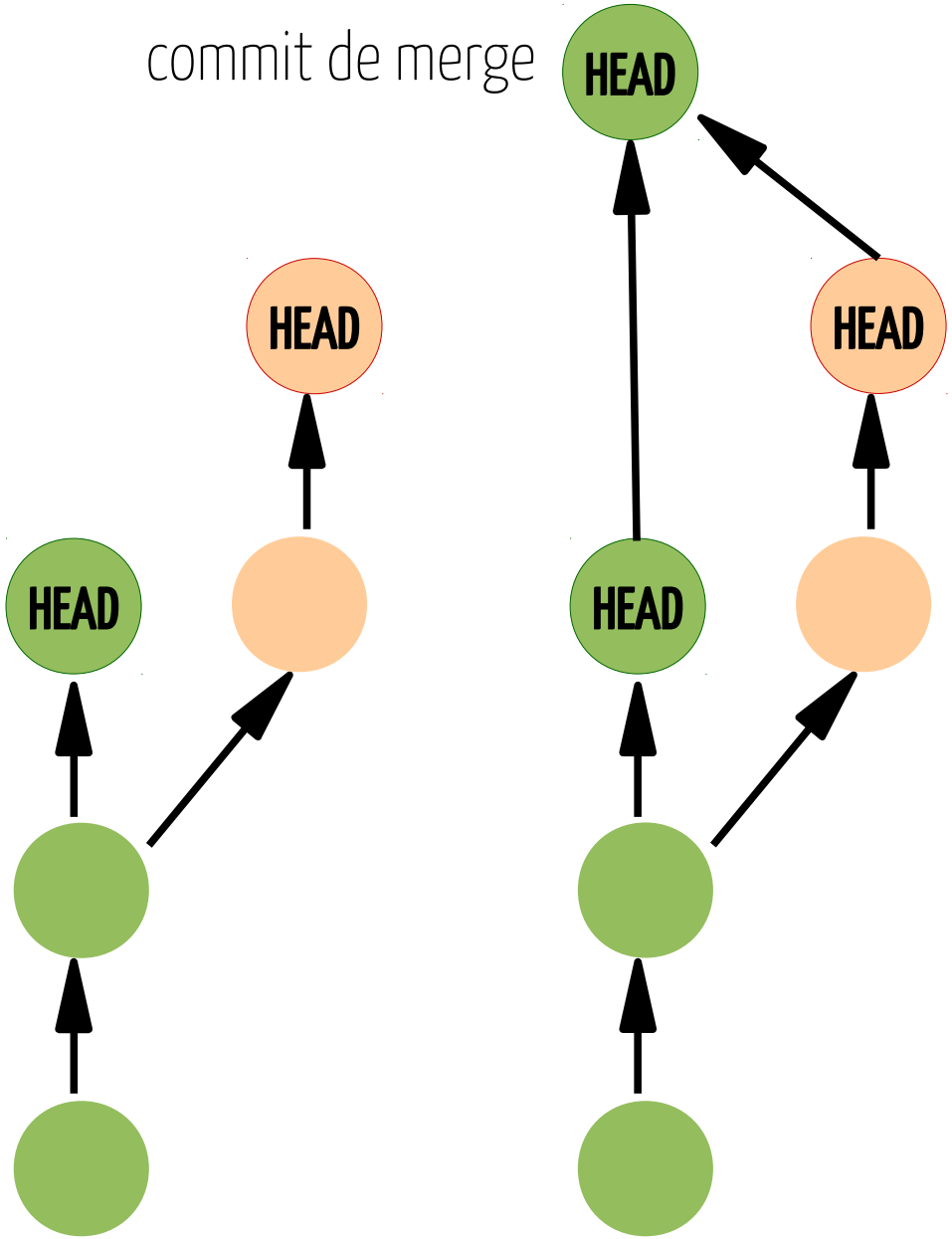


hupstream

Branches fast-forward



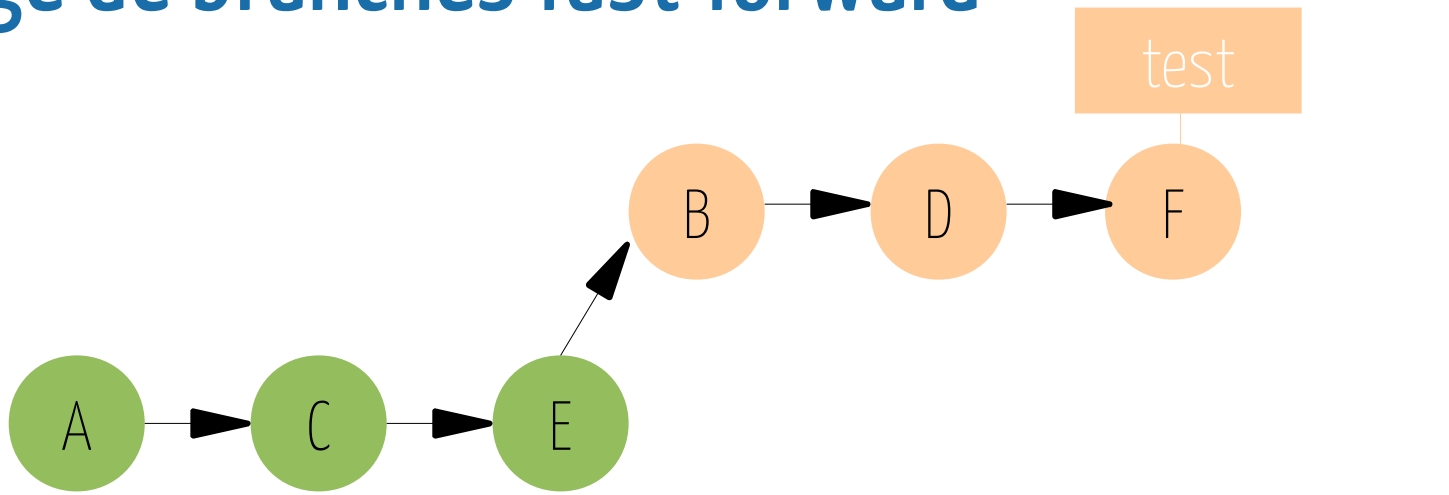
fast-forward



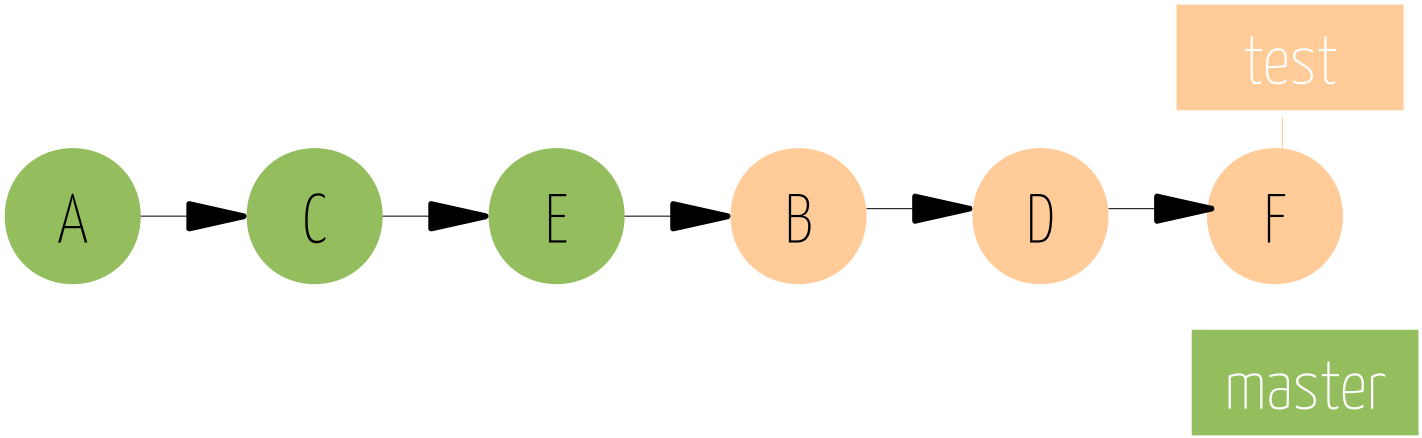
non fast-forward

hupstream

Merge de branches fast-forward

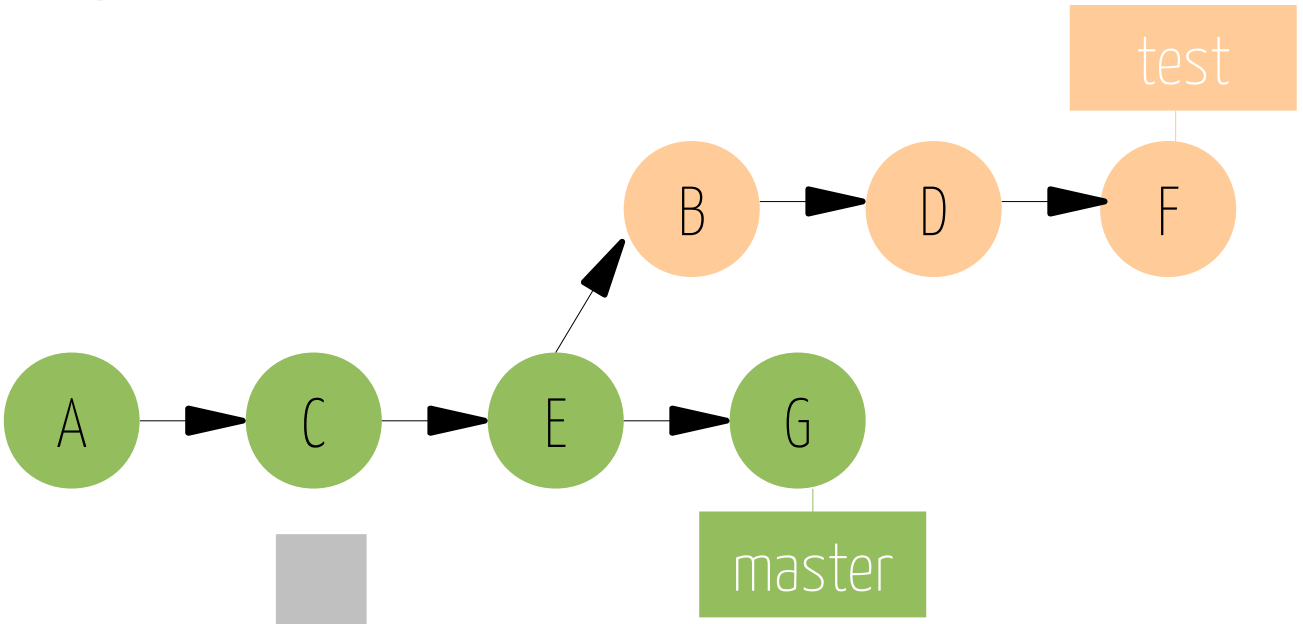



```
git checkout master
git merge test
```

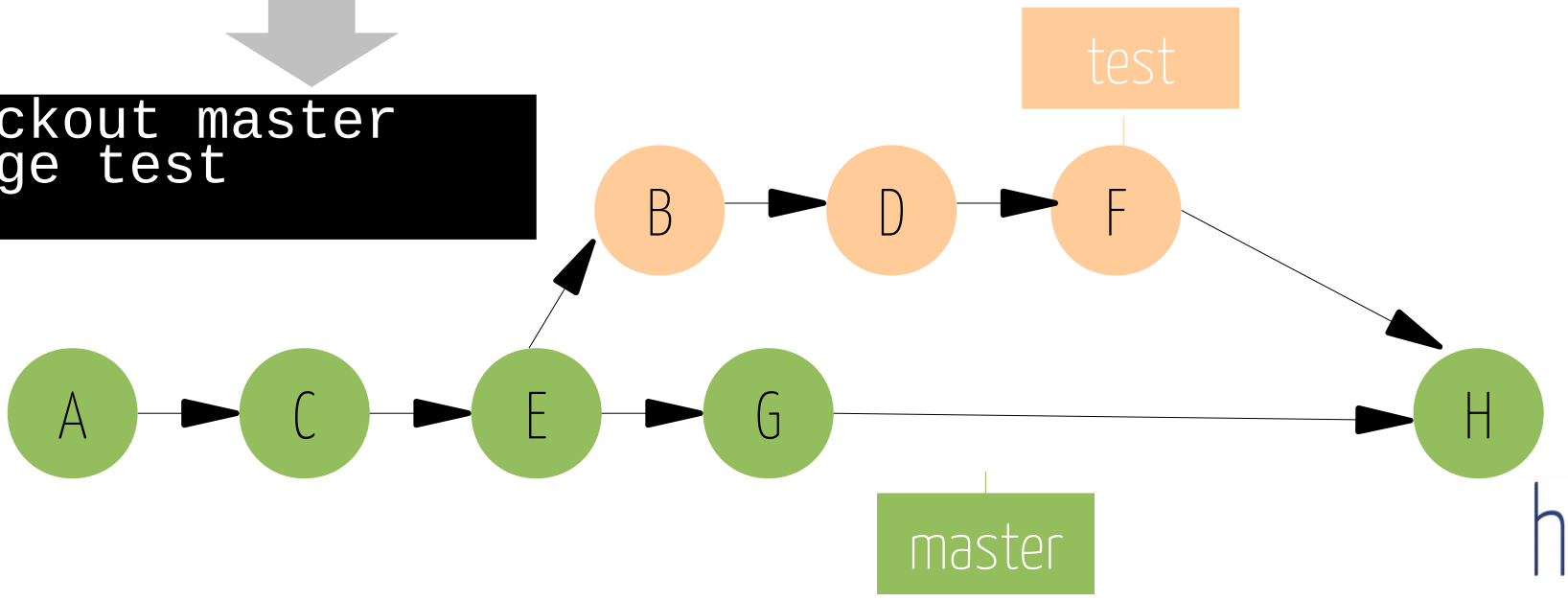


hupstream

Merge de branches non fast-forward




```
git checkout master
git merge test
```



hupstream

Le commit de merge

Un commit spécifique avec deux parents

```
$ git cat-file -p ac83084

tree 50657a22a97e9cd86e9bf8c65b8ba42fa2155d22
parent 6e20b5b02604e9567479e08cbe1c2c79af6afe78
parent 0eca865bb3db8f05b3bf9167184ab47416263196
author Anne Nicolas <ennael@mageia.org> 1455552916 +0100
committer Anne Nicolas <ennael@mageia.org> 1455552916 +0100

Merge branch 'dev'
```

Résolution manuelle d'un conflit

Un message typique de conflit

```
[ennael@localhost BCD (master)]$ git merge test
Auto-merging create_dvd.sh
CONFLICT (content): Merge conflict in create_dvd.sh
Automatic merge failed; fix conflicts and then commit the result.
```

Résolution manuelle d'un conflit

Afficher le status d'un merge : git status

```
[ennael@localhost BCD (master|MERGING)]$ 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:      create_dvd.sh
```

Résolution manuelle d'un conflit

Trouver la cause du problème

```
[ennael@localhost BCD (master|MERGING)]$ vi create_dvd.sh
...
<<<<<<< HEAD
test=$homebcd/build
=======
test=$homebcd/pieces
>>>>>>> test
```

Vérifier le contenu d'un fichier dans une branche donnée

```
[ennael@localhost BCD (master|MERGING)]$ git show test:create_dvd.sh
```

Résolution manuelle d'un conflit

Afficher les différentes version du fichier incriminé

```
[ennael@localhost BCD (master|MERGING)]$ git ls-files -u
100755 74b6be3071f1f4ccc50fef6583200719 0b64f76b 1       create_dvd.sh
100755 df2c0b97b36da492688bcc1ac34e9daaeed7a463 2       create_dvd.sh
100755 c68e912bb118dc7c0aed3de3662b429438b3bce4 3       create_dvd.0sh
```

Obtenir le contenu d'une de ces versions

```
[ennael@localhost BCD (master|MERGING)]$ git show :1:create_dvd.sh
```

hupstream

Résolution manuelle d'un conflit

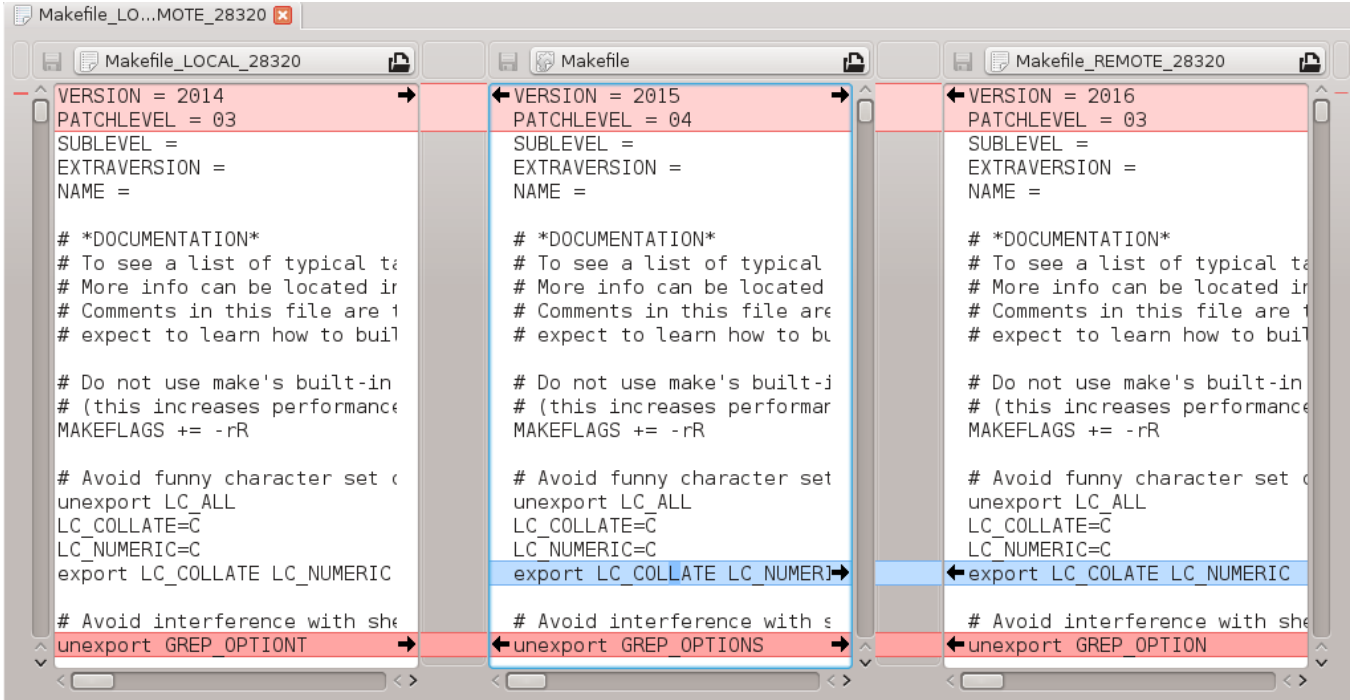
Utilisation d'outils pour visualiser le contenu d'un conflit

outils disponibles / utilisables

```
git mergetool --tool-help
```

Lancer l'outil de gestion des diff

```
git mergetool -t <outil> [<fichier>]
```



hupstream

Résolution manuelle d'un conflit

Résoudre le conflit et partager

1. modifier le fichier pour résoudre le conflit
2. ajouter le fichier à l'index
3. committer le merge

Annuler un merge en cours

```
git merge --abort
```

Gérer un conflit avec une stratégie

Choisir de conserver systématiquement la version d'une branche donnée

Utiliser une stratégie pour conserver soit la version de la branche courante (ours) soit celle de la branche entrante (theirs),

```
git merge <branch> --strategy-option theirs|ours
```

```
[ennael@localhost BCD (master)]$ git merge test
Auto-merging create_dvd.sh
CONFLICT (content): Merge conflict in create_dvd.sh
Automatic merge failed; fix conflicts and then commit the result.
[ennael@localhost BCD (master)]$ git merge test --strategy-option theirs
Auto-merging create_dvd.sh
Merge made by the 'recursive' strategy.
 create_dvd.sh | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

hupstream

Créer une archive du code

Lister les formats d'archive disponibles

```
git archive -l
```

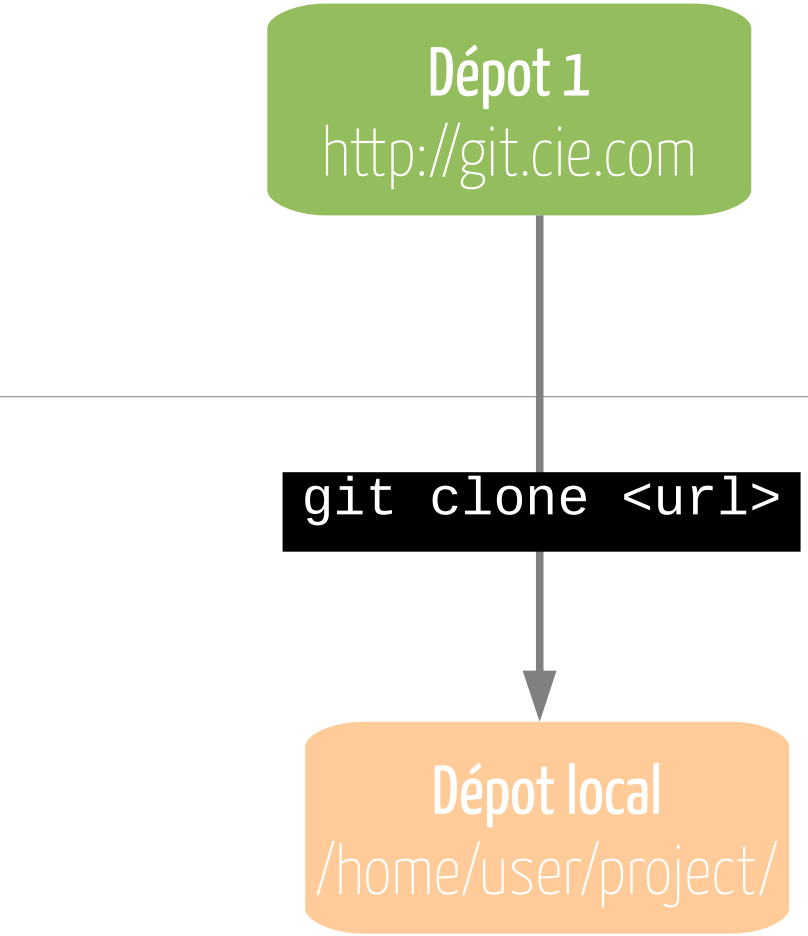
Générer une archive du code à un moment donné de l'historique

```
git archive --format=<format> --prefix=<répertoire/> -o <arvhive> <SHA1>
```

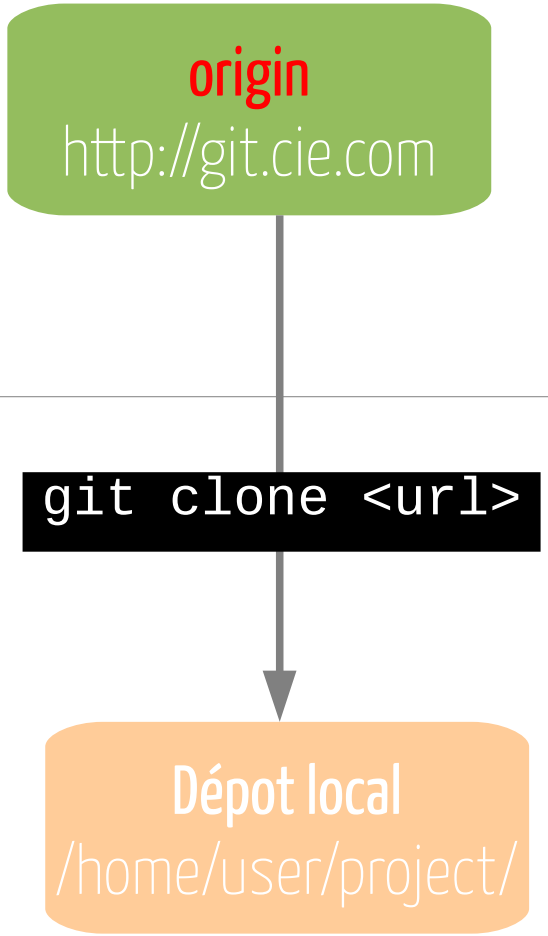
5. Dépot distant, branches distantes

hupstream

Gérer le dépot git distant



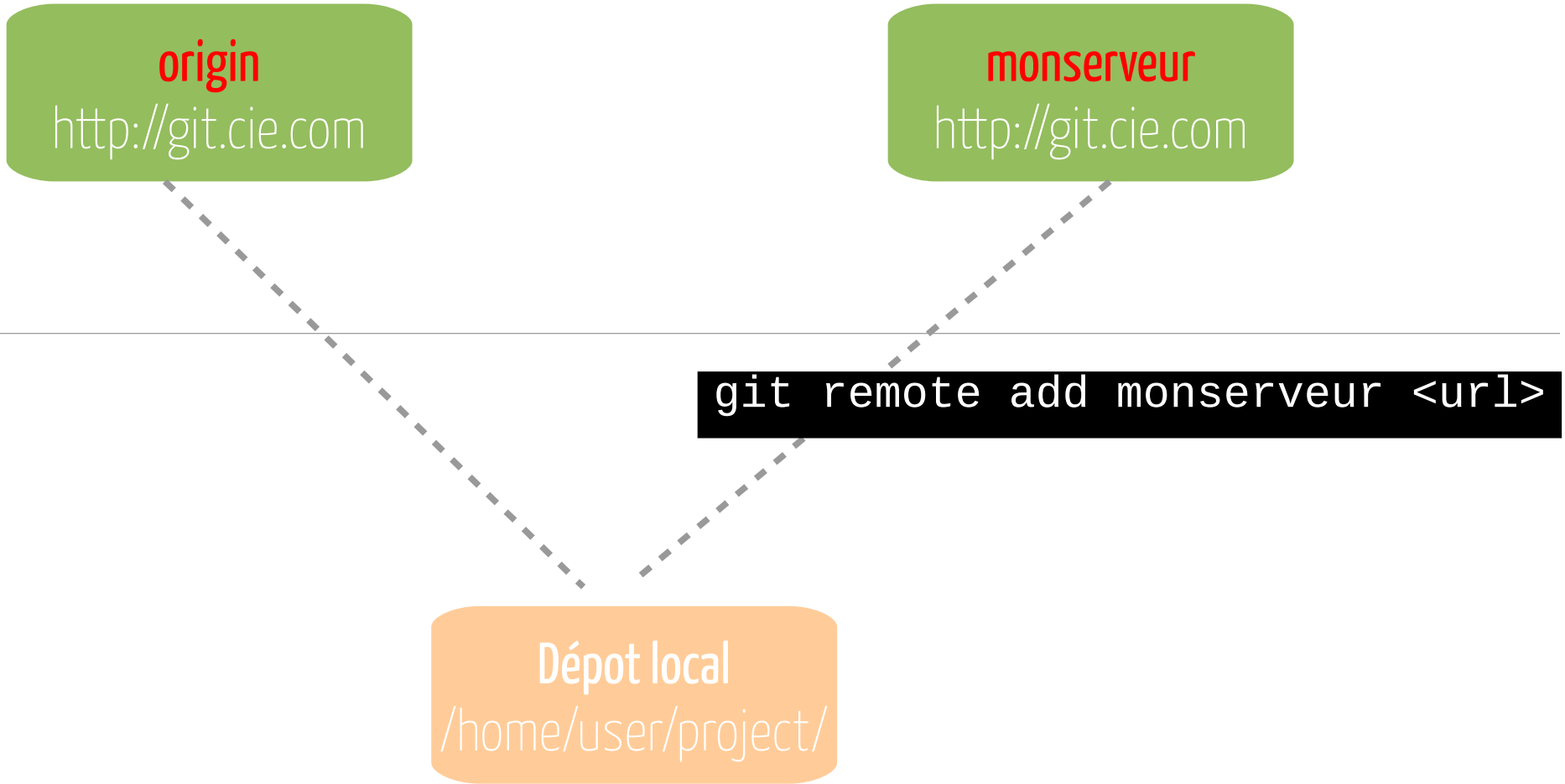
Gérer le dépot git distant



Le premier dépot est toujours appellé “**origin**”

hupstream

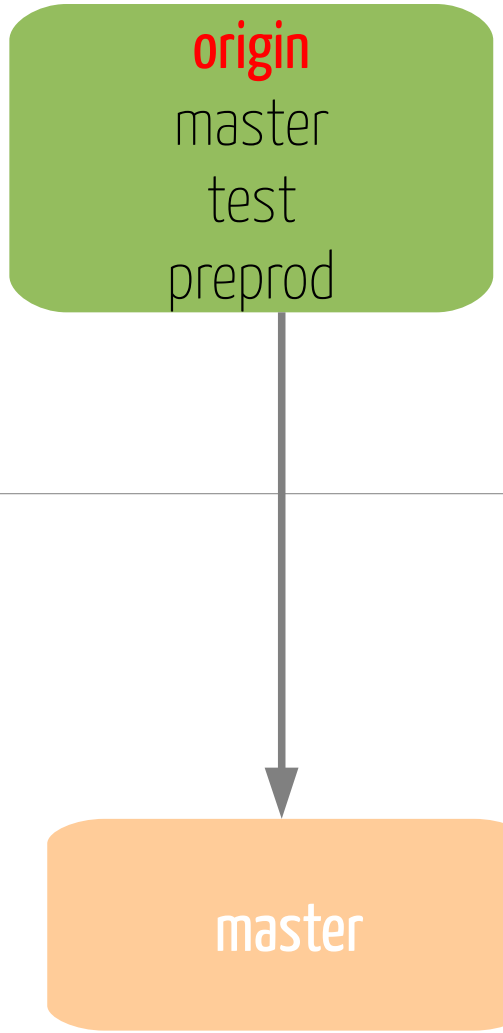
Gérer d'autres dépots git distants



Le premier dépot est toujours appellé "**origin**"

hupstream

Lister les branches distantes




```
# git branch -r

origin/master
origin/test
origin/preprod
```

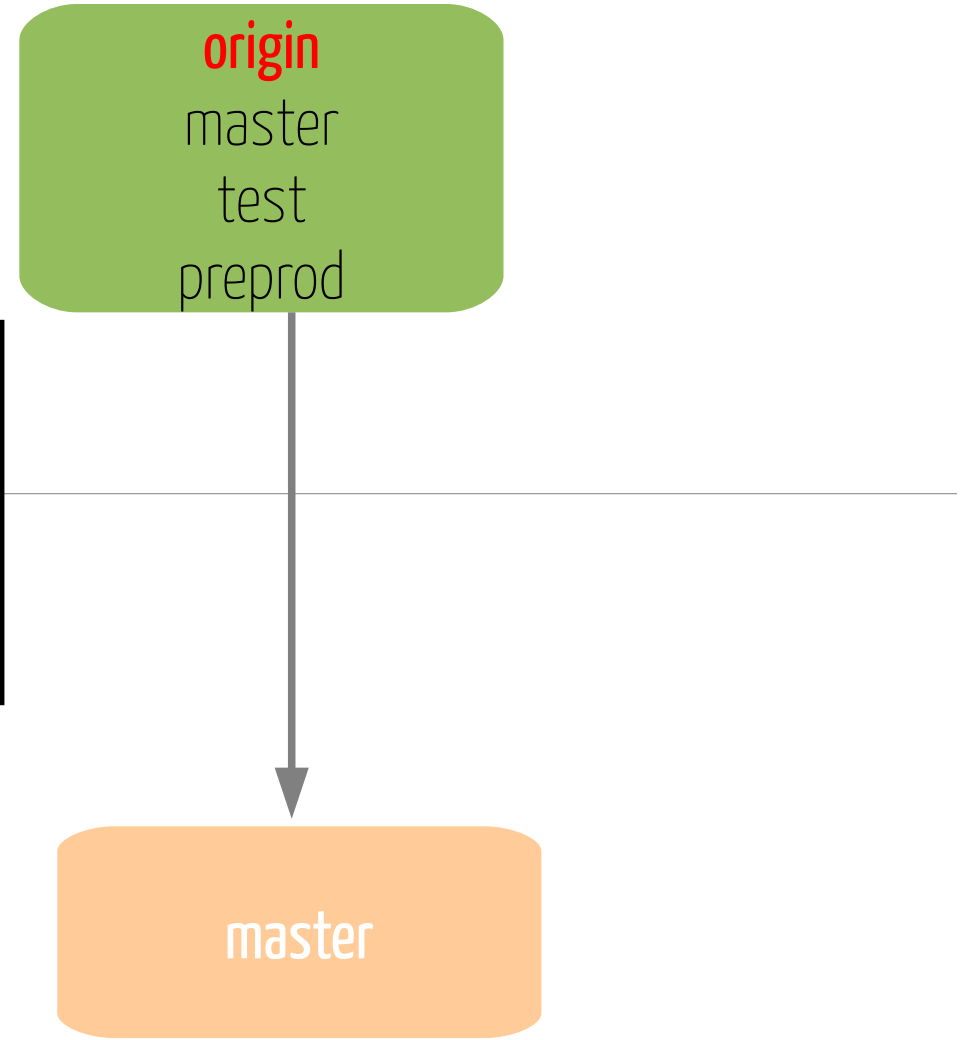
Récupérer une branche distante

Processus de récupération

1. création d'une branche locale
2. merge de la branche distante dans la branche locale
3. enregistrement du suivi de la branche dans le fichier `.git/config`

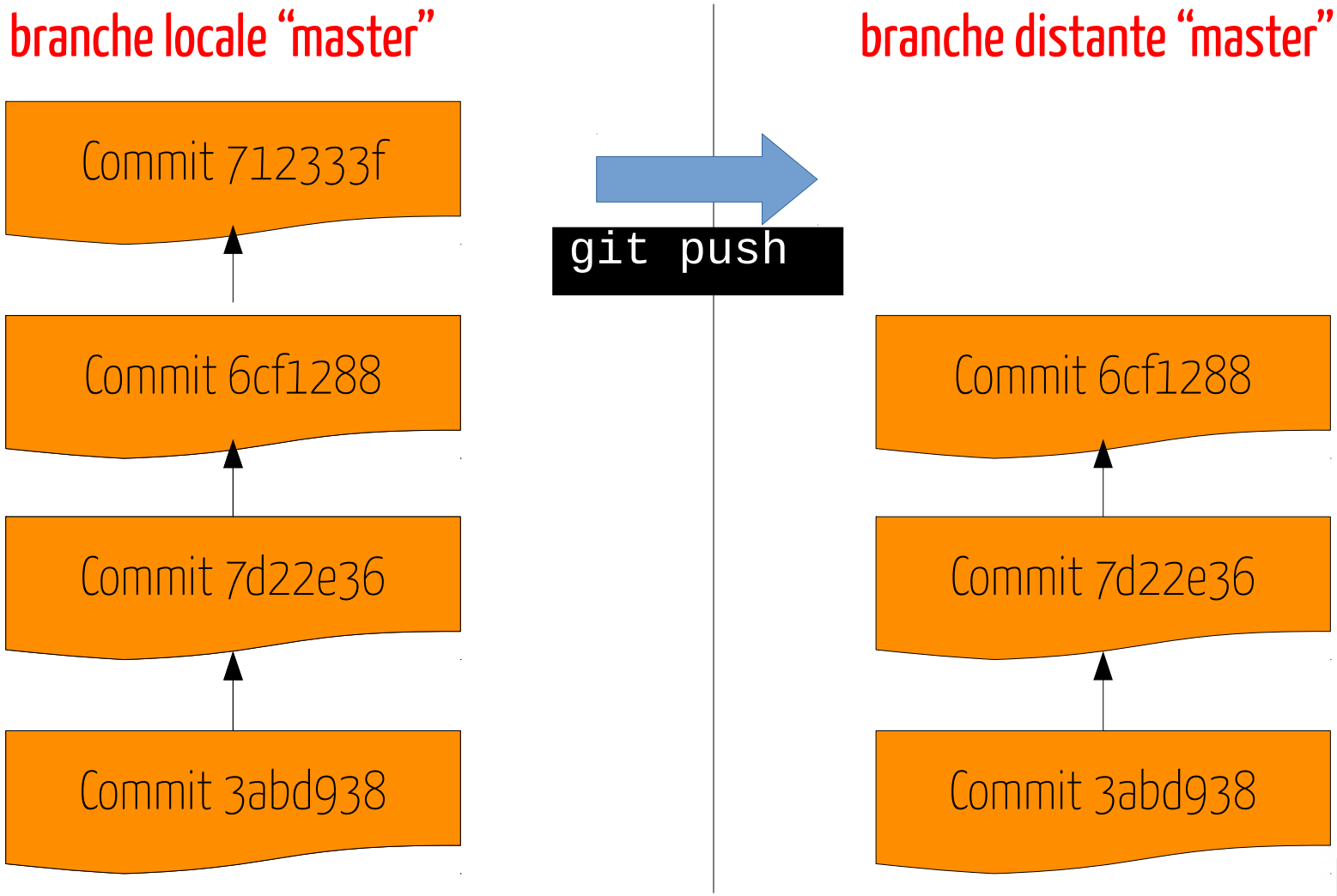
Récupérer une branche distante

```
# git checkout -b preprod origin/preprod
# git branch
* preprod
master
# git branch -a
* preprod
master
remotes/origin/master
remotes/origin/preprod
```



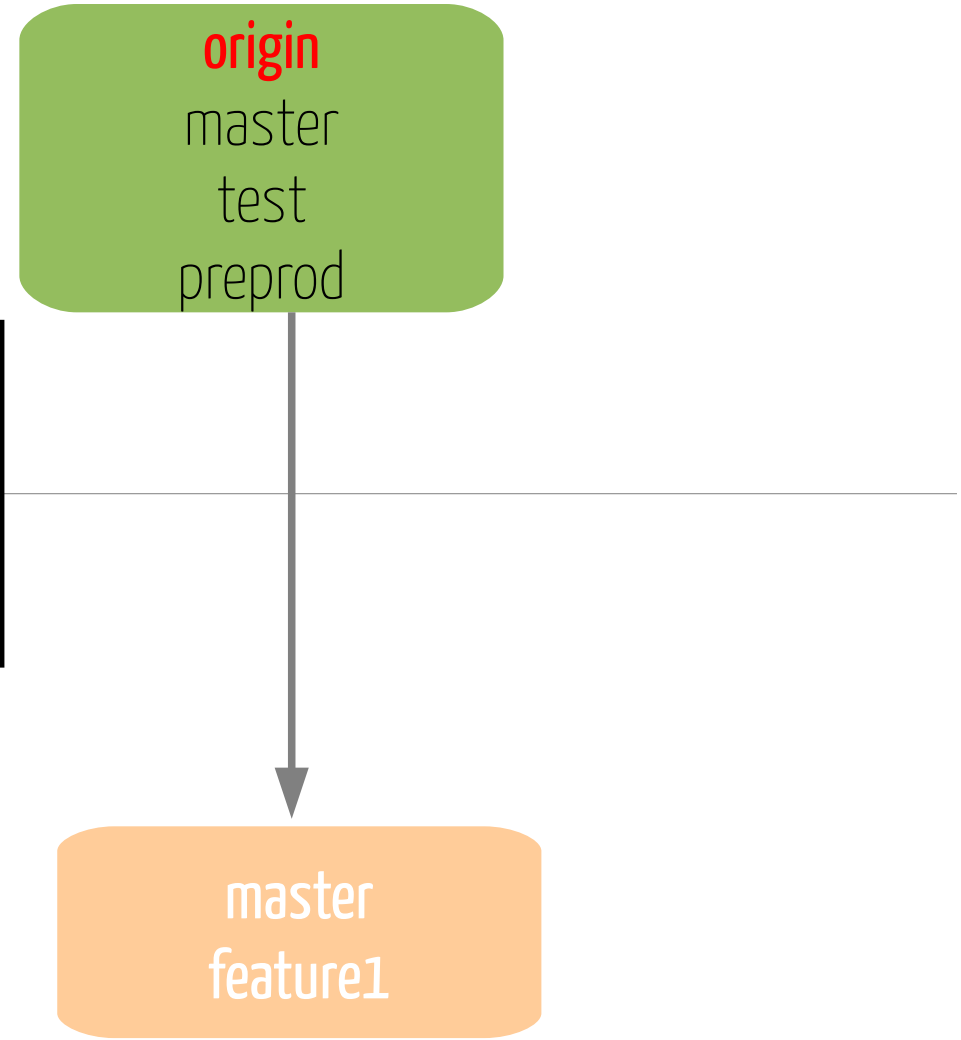
hupstream

Partager ses commits



Partager une branche locale

```
# git push -u origin feature1
# git branch -a
* preprod
feature1
master
remotes/origin/feature1
remotes/origin/master
remotes/origin/preprod
```



hupstream

Commandes de gestion de branches

- Liste de toutes les branches (locales + distantes)

```
git branch -a
```

- Récupérer une branche distante

```
git checkout -b <branche_locale> <dépot>/<branche>
```

ou

```
git checkout <branche_locale>
```

- Associer une branche locale à une branche distante

```
git branch --set-upstream locale origin/distante
```

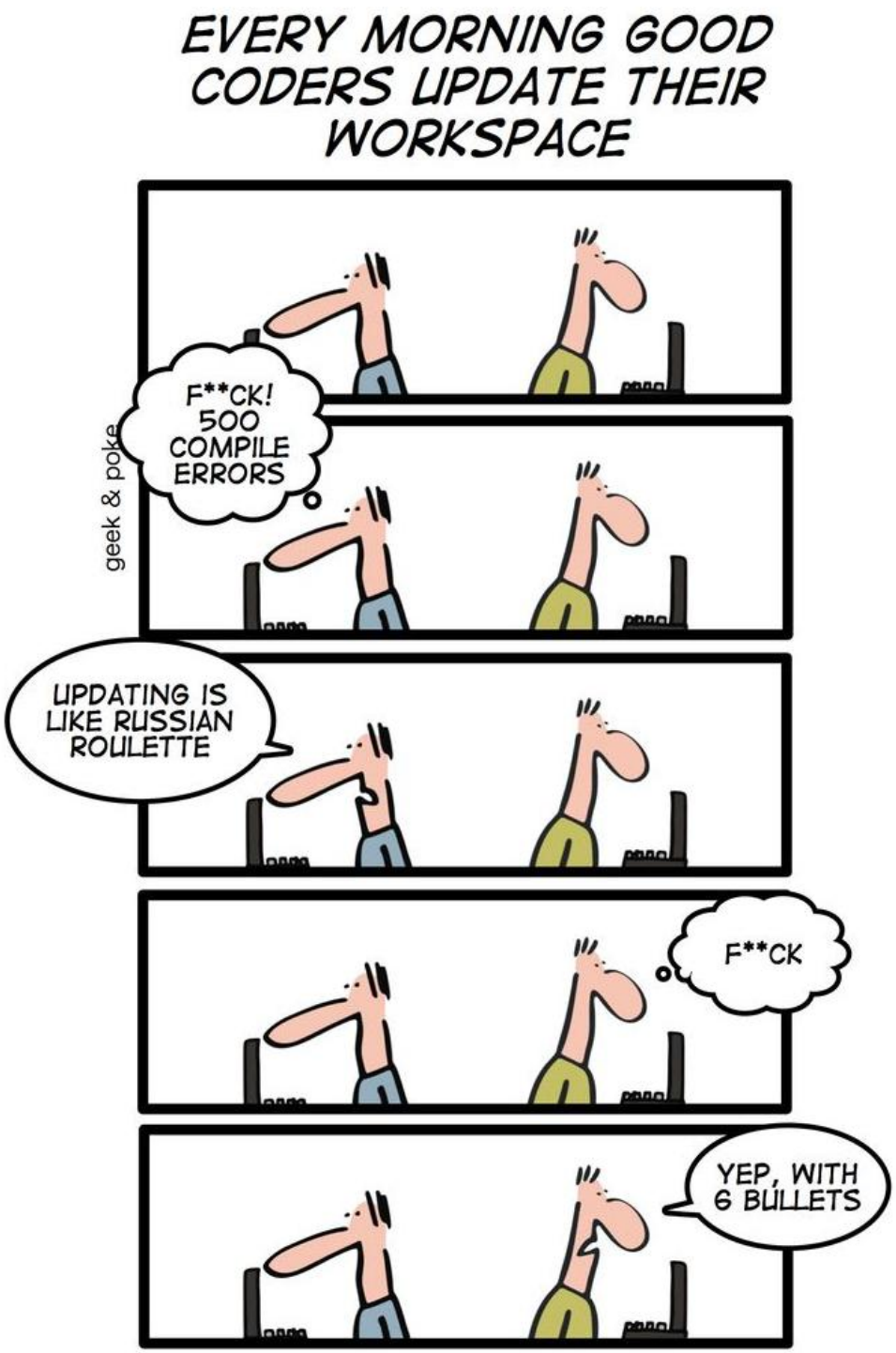
- Pousser une branche sur le serveur distant

```
git push -u origin <branche>
```

- Supprimer une branche distante

```
git push origin --delete <branche_distante>
```

Update


EVERY MORNING GOOD CODERS UPDATE THEIR WORKSPACE
F**CK! 500 COMPILE ERRORS
geek & poke
UPDATING IS LIKE RUSSIAN ROULETTE
F**CK
YEP, WITH 6 BULLETS

Suivi de branches à distance : git fetch

- git pull = git fetch + git merge
- Mettre à jour le suivi des branches distantes sans rien modifier dans les branches locales ni la zone de travail locale
- Passer en revue les modifications disponibles sur le serveur distant

```
[ennael@localhost sandgit (master)]$ git fetch
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From ssh://git.hupstream.com/git/users/sandgit
 * [new branch]      test1      -> origin/test1
```

hupstream

Suivi de branches à distance : reccueillir de l'information

- liste des commits disponibles dans la branche distante mais pas dans la branche locale

```
git log HEAD..origin/<branche>
```

descrtiption du dépot distant et des branches

```
git remote show origin
```

Bonnes pratiques : du bon usage des branches

Eviter de détruire une branche distante qui n'a pas été mergée

- Détruire une branche détruit également le code définitivement
- Conserver toutes les branches rend les dépots inutilisables

```
$ git branch -d dev
error: The branch 'dev' is not fully merged.
If you are sure you want to delete it, run 'git branch -D dev'.
$ git branch -D dev
```

Partage des tags

Pousser des tags sur le serveur

- pousser tous les tags disponibles en local

```
git push --tags
```

- pousser uniquement ceux dont le commit a déjà été poussé su le serveur

```
git push --follow-tags
```

- pousser un tag en particulier

```
git push origin <tag>
```

Récupérer un tag distant

```
git pull --tags
```

Supprimer un tag distant

```
git push origin :refs/tags/<tag>
```

Tags et refspec

Inclure les tags dans les données à pousser récupérer

- sans modifier la configuration du dépôt local

```
git fetch --all
```

- en modifiant les refspec

```
[remote "upstream"]
   url = <url>
   fetch = +refs/heads/*:refs/remotes/upstream/*
   fetch = +refs/tags/*:refs/tags/*
```

6. Workflows et modes d'organisation

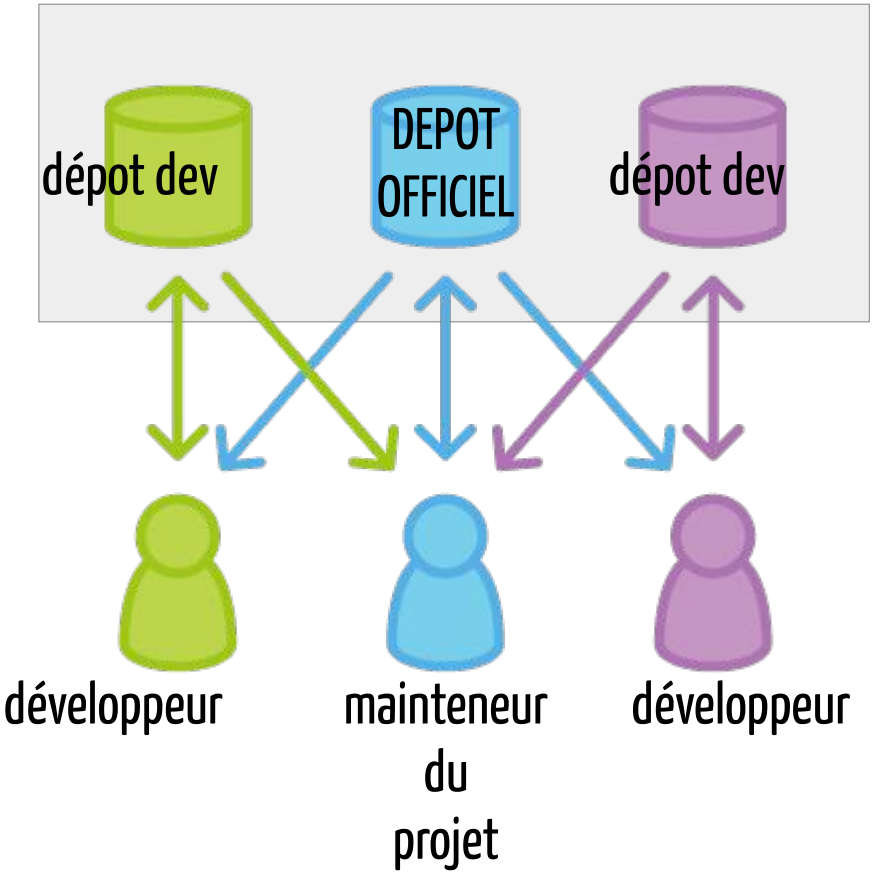
hupstream

Workflow : fork et pull request

Déroulé d'une pull request

- Création d'une branche basée sur master
- Apport des modifications
- Envoi d'une pull request via l'interface prévue à cet effet, contenant les changements à merger
- Nouvelles modifications si demande en ce sens
- Merge de la pull request
- Nettoyage des branches en utilisant la fonctionnalité prévue à cet effet

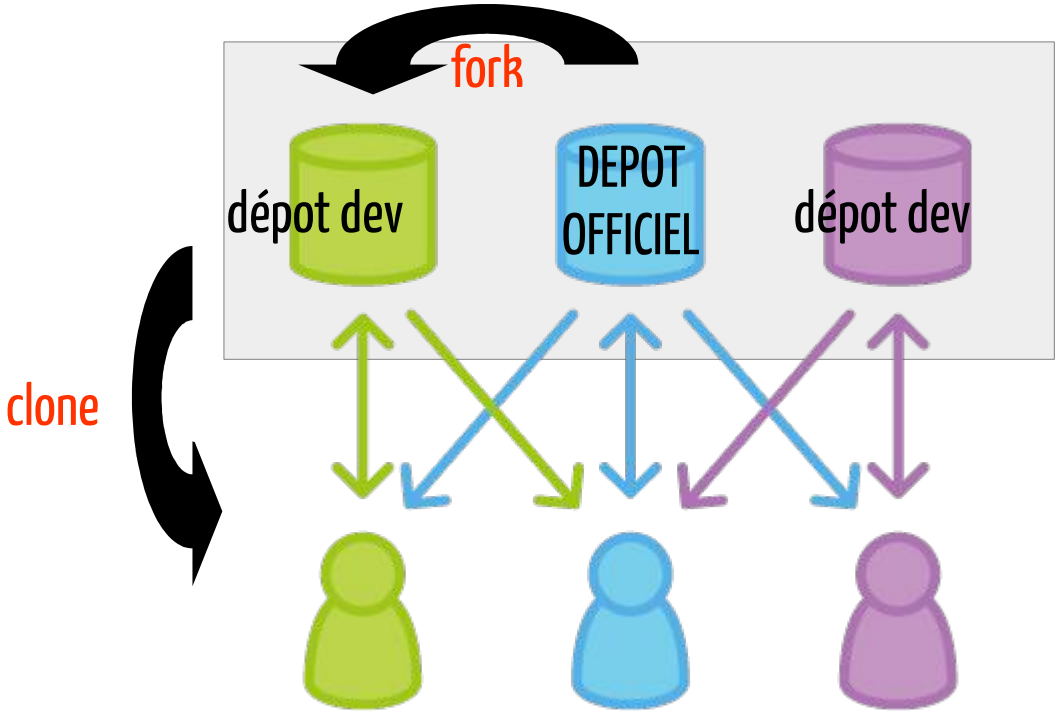
Workflow à base de fork



- développeur = 1 dépot privé local + 1 dépot distant
- mainteneur = dépot officiel du projet
- permet de gérer des intervenants extérieurs sans donner des droits d'accès au dépot officiel

hupstream

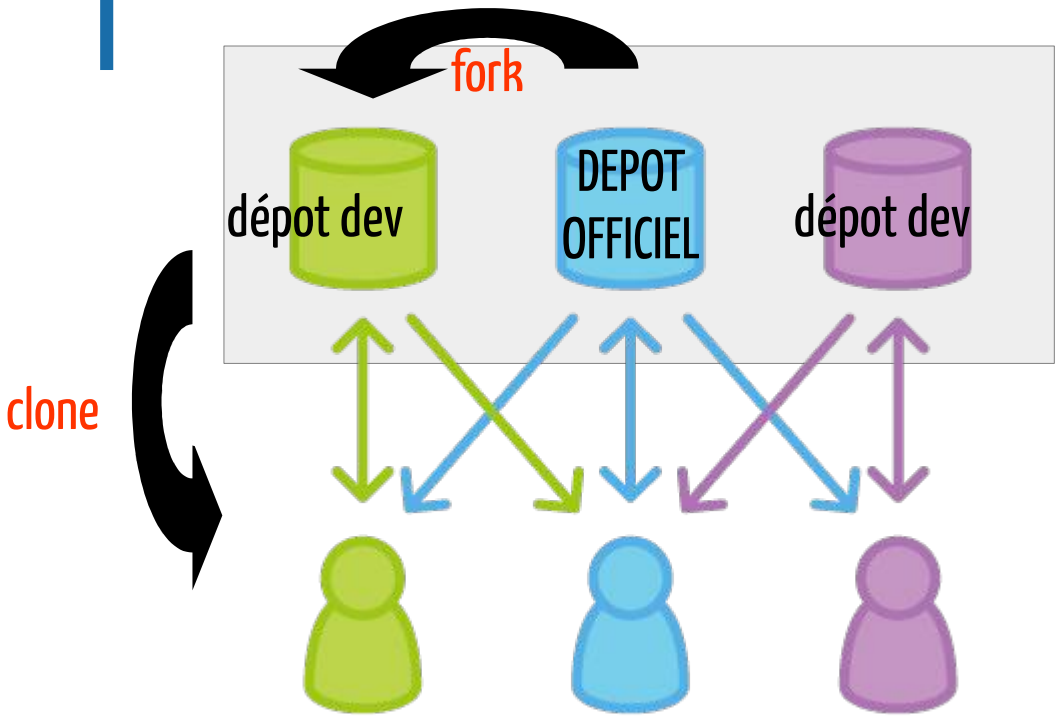
Workflow à base de fork



- dépot public dev est un fork du dépot officiel : un clone côté serveur (git clone --bare)
- utilisation de 2 remotes : origin (dev) et upstream (officiel) avec git remote add
- dev peut mettre à jour depuis le dépot officiel (git pull upstream master)
- dev pousse ses modifications sur le dépot distant dev

hupstream

Workflow à base de fork




```
Dev
git pull upstream master
git push origin feature-branch

Mainteneur
git pull upstream master
git push origin master
```

1) envoi des commits sur le dépot public distant
2) envoi d'une pull request
3) récupération par le mainteneur dans son dépot local ou examen direct de la pull request
4) review
5) merge dans la branche master local
6) envoi sur la branche master du dépot officiel

hupstream

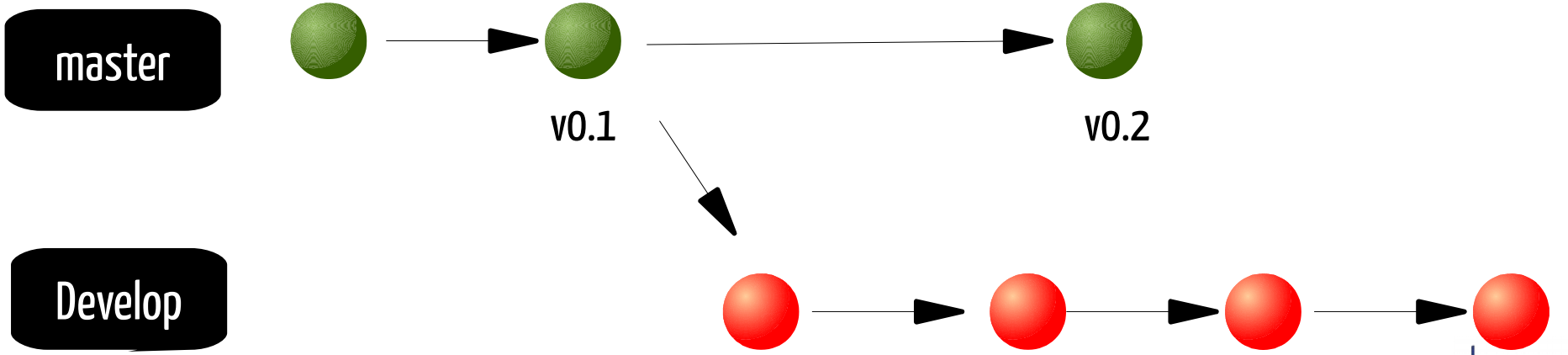
git flow

La branche master

- historique officiel du projet
- versions finales
- utilisation de tags

La branche Develop

- branche d'intégration pour les features développées

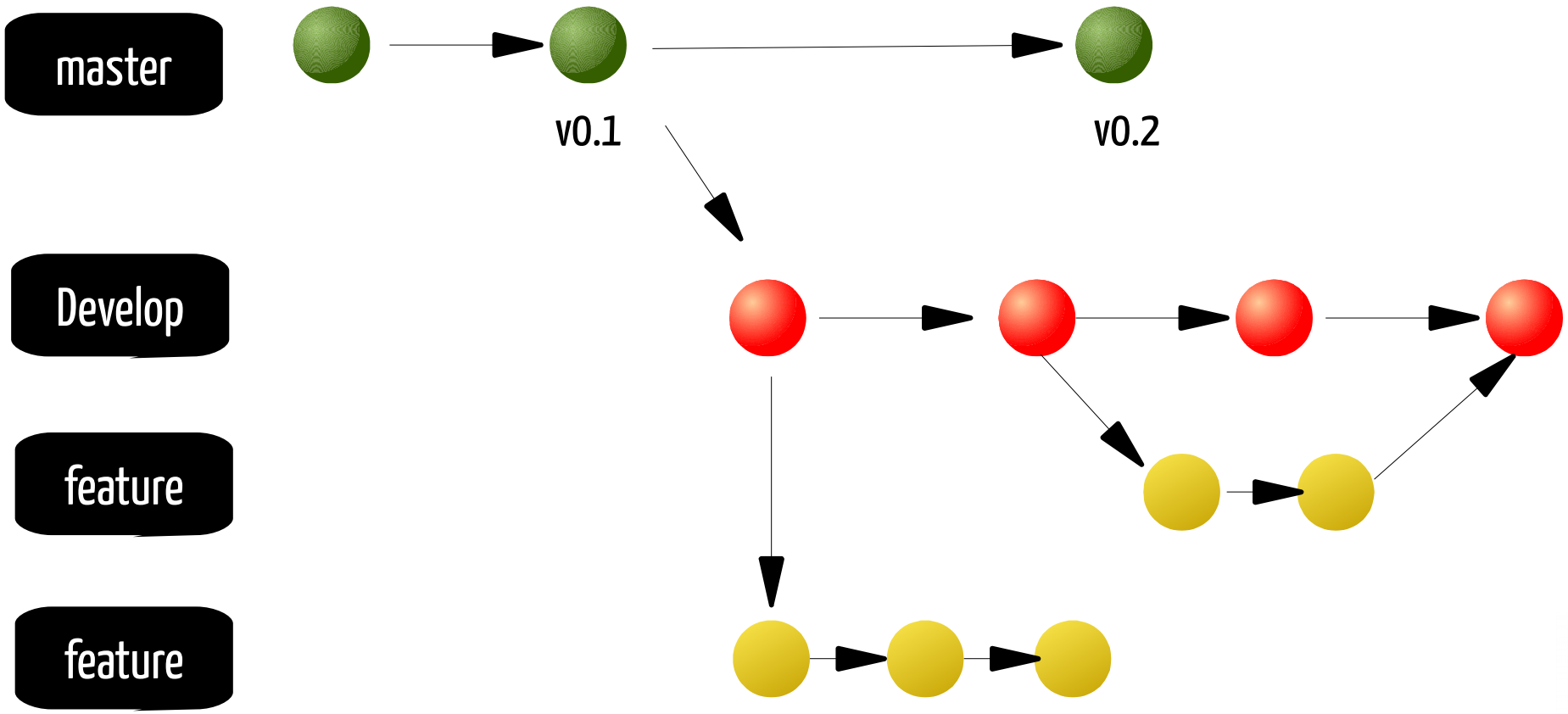


hupstream

git flow

Les branches de feature

- les features ne sont jamais reversées dans master

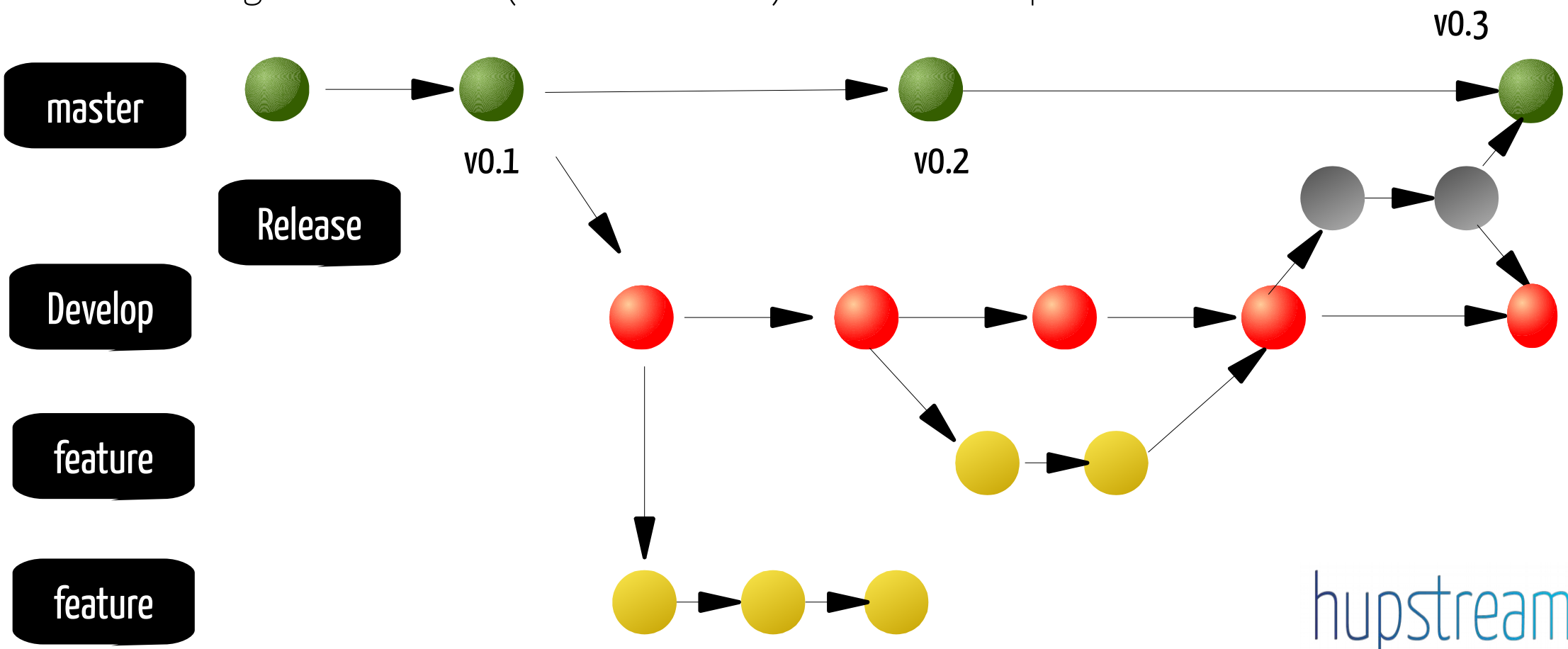


hupstream

git flow

Les branches de release

- création de la branche de release depuis Develop
- merge dans master (nouvelle version) et dans Develop

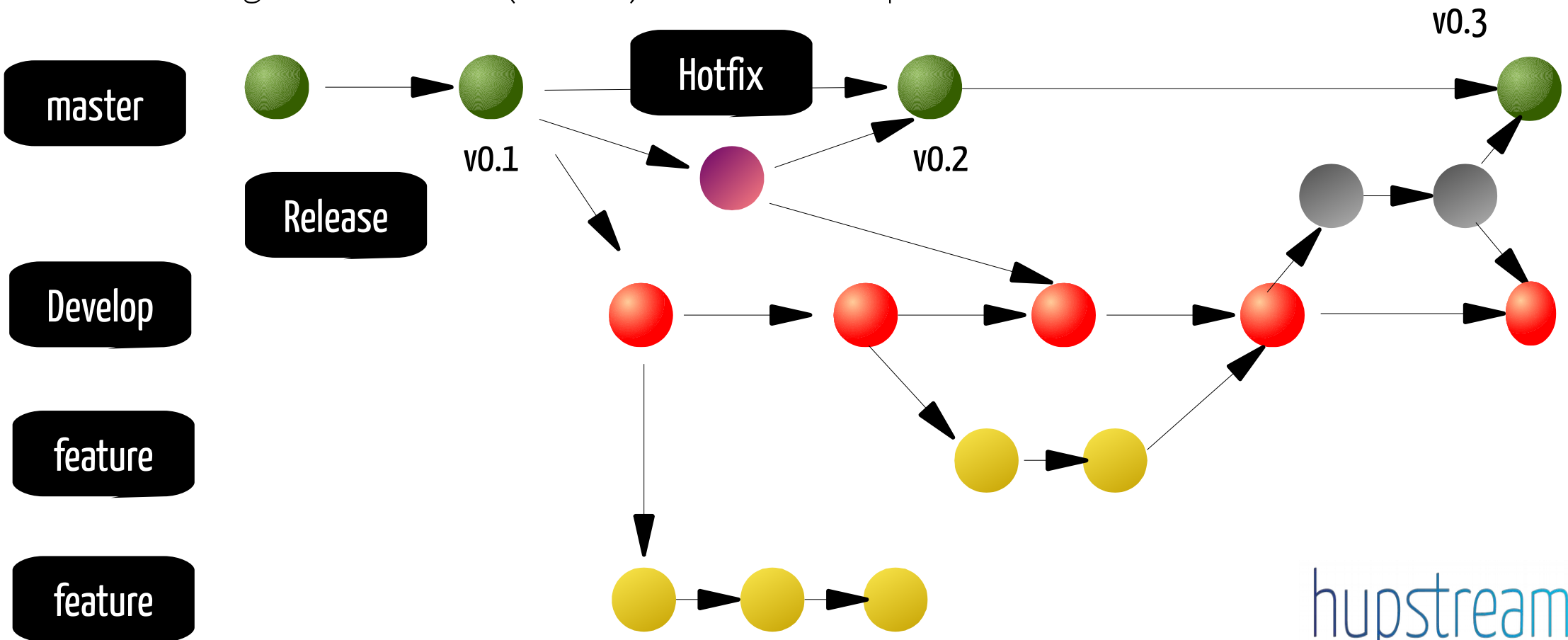


hupstream

git flow

Les branches de maintenance

- les seules créées depuis master, bug fix sur la branche de production
- merge dans master (version) et dans Develop

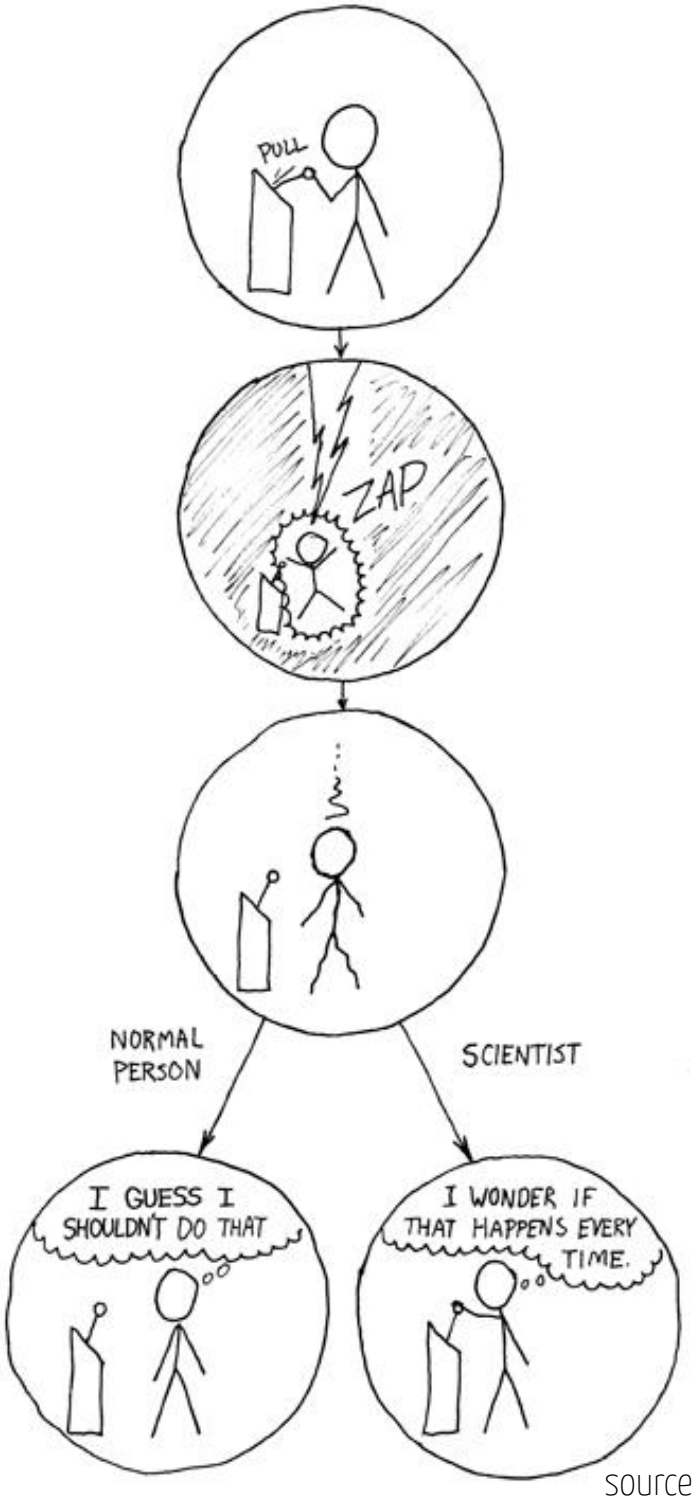


hupstream

7. Le rebase sous toutes ses formes

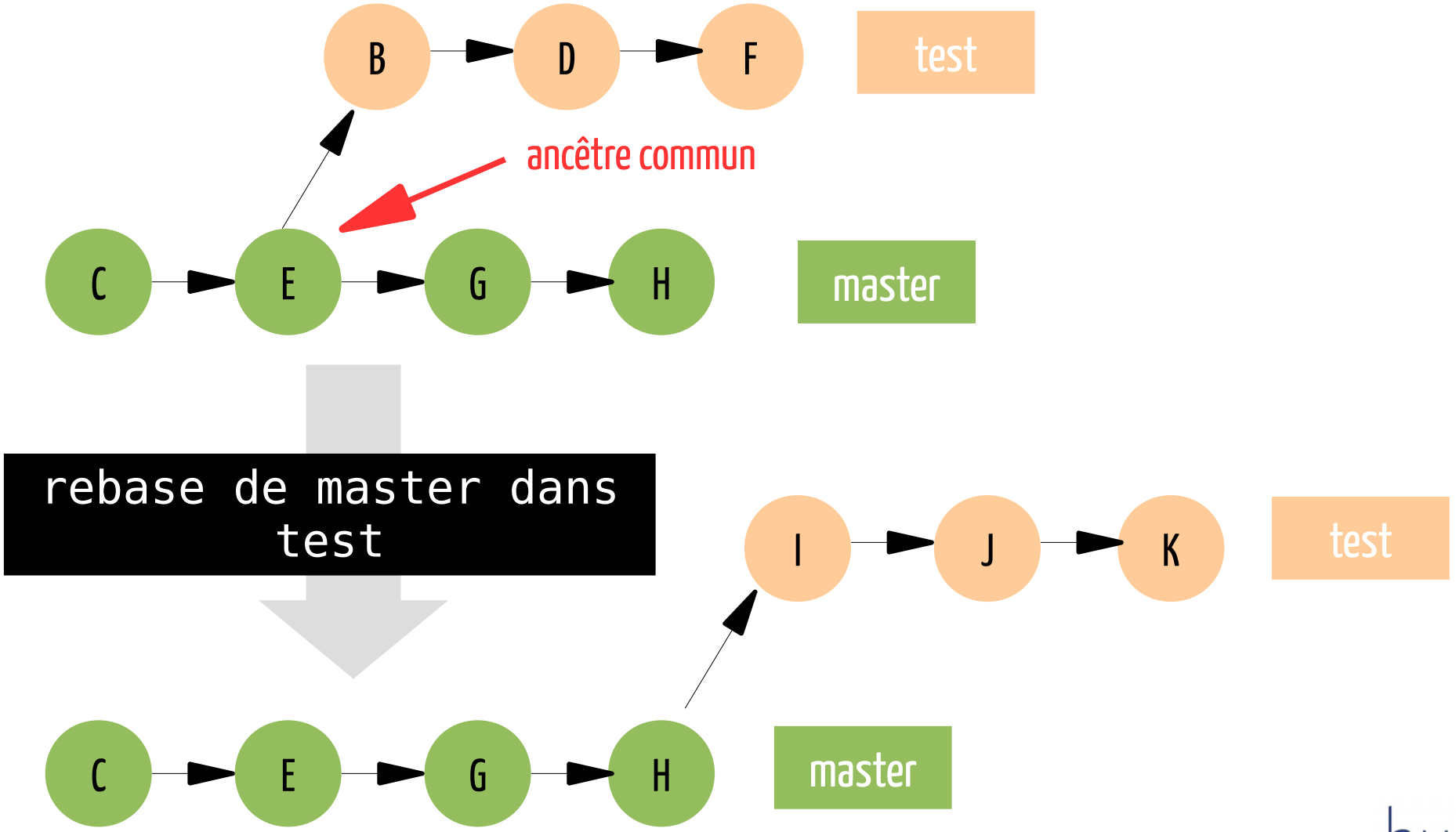
hupstream

Le rebase n'est pas à proscrire s'il est fait correctement



source : xkcd

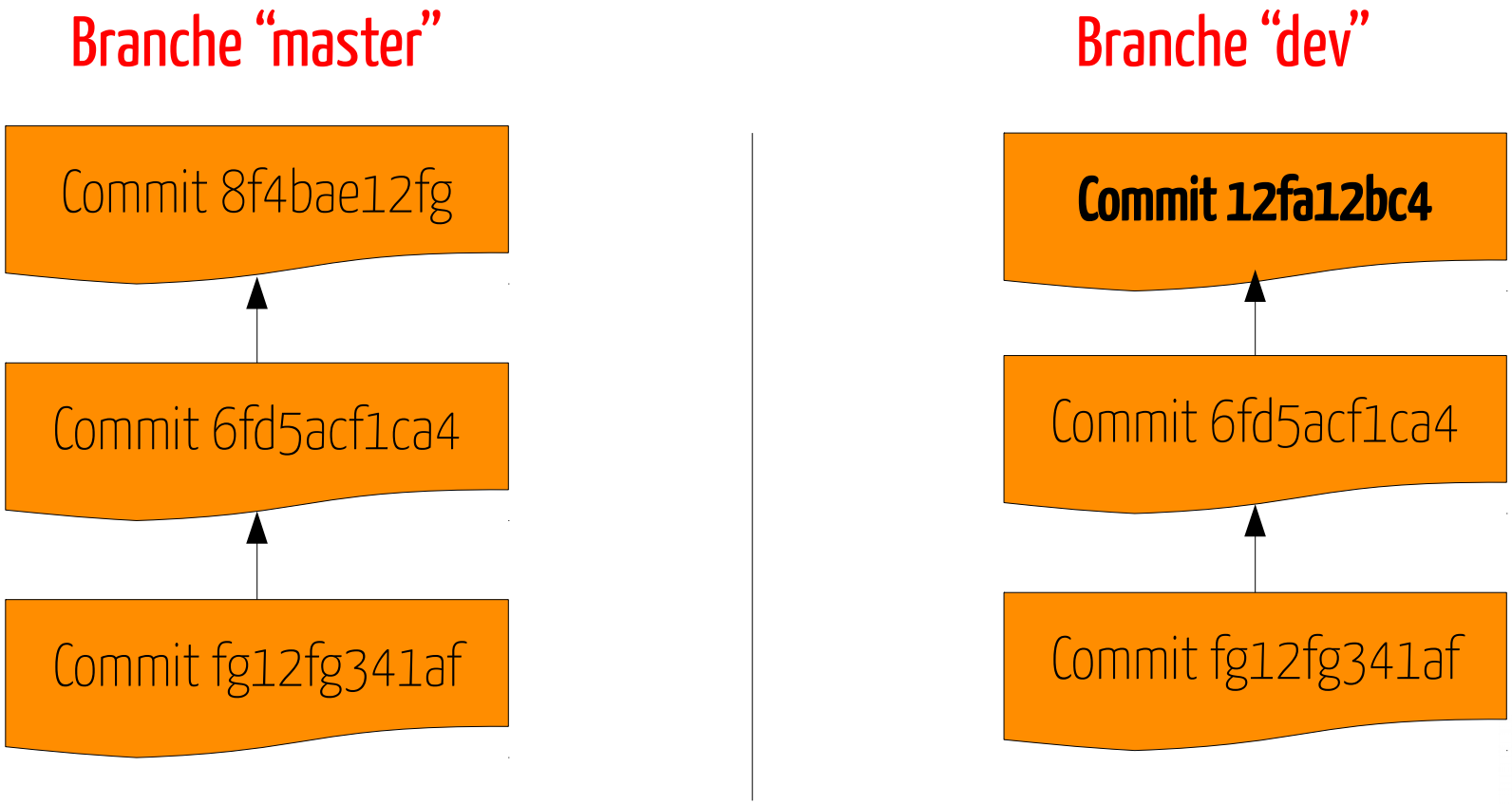
Le rebase pour réorganiser l'historique



hupstream

Le rebase de branches locales

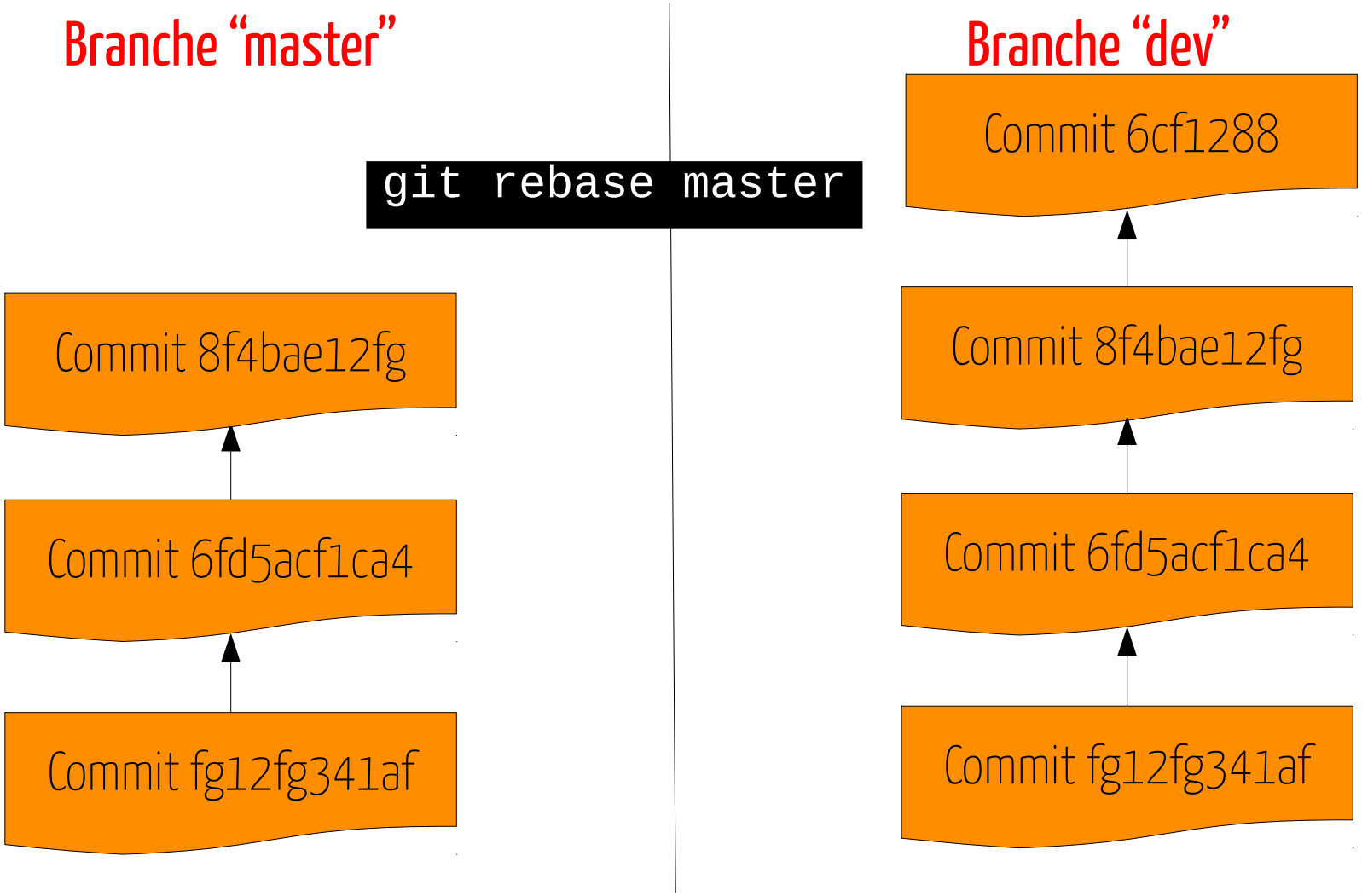
- un commit a été réalisé pendant que le commit 8f4ba était exécuté
- les branches ne sont pas “fast-forward”
- objectif : mise à jour de la branche dev



hupstream

Le rebase de branches locales

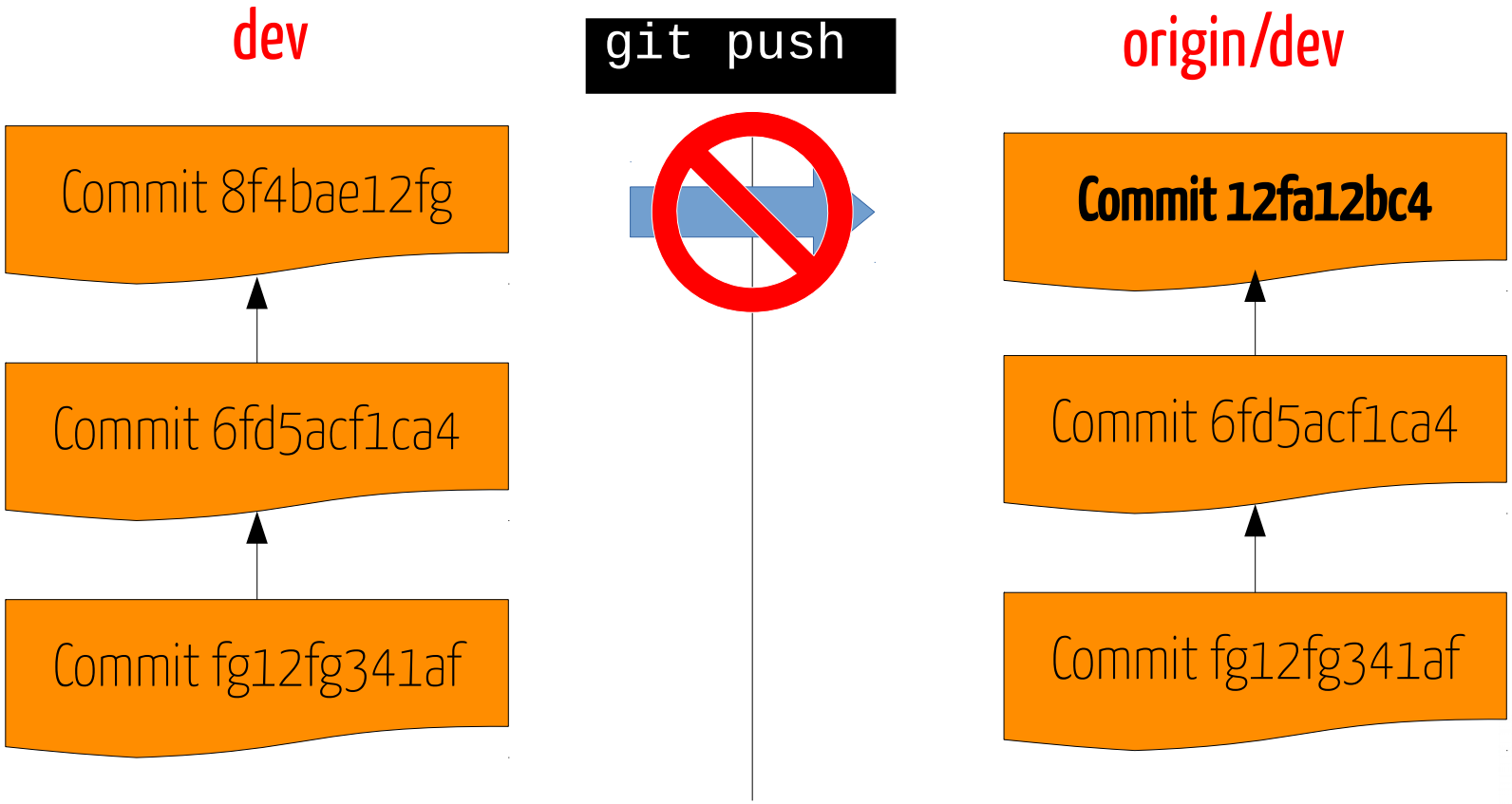
- après rebase : branches fast-forward



hupstream

Le rebase de branches locale / distante

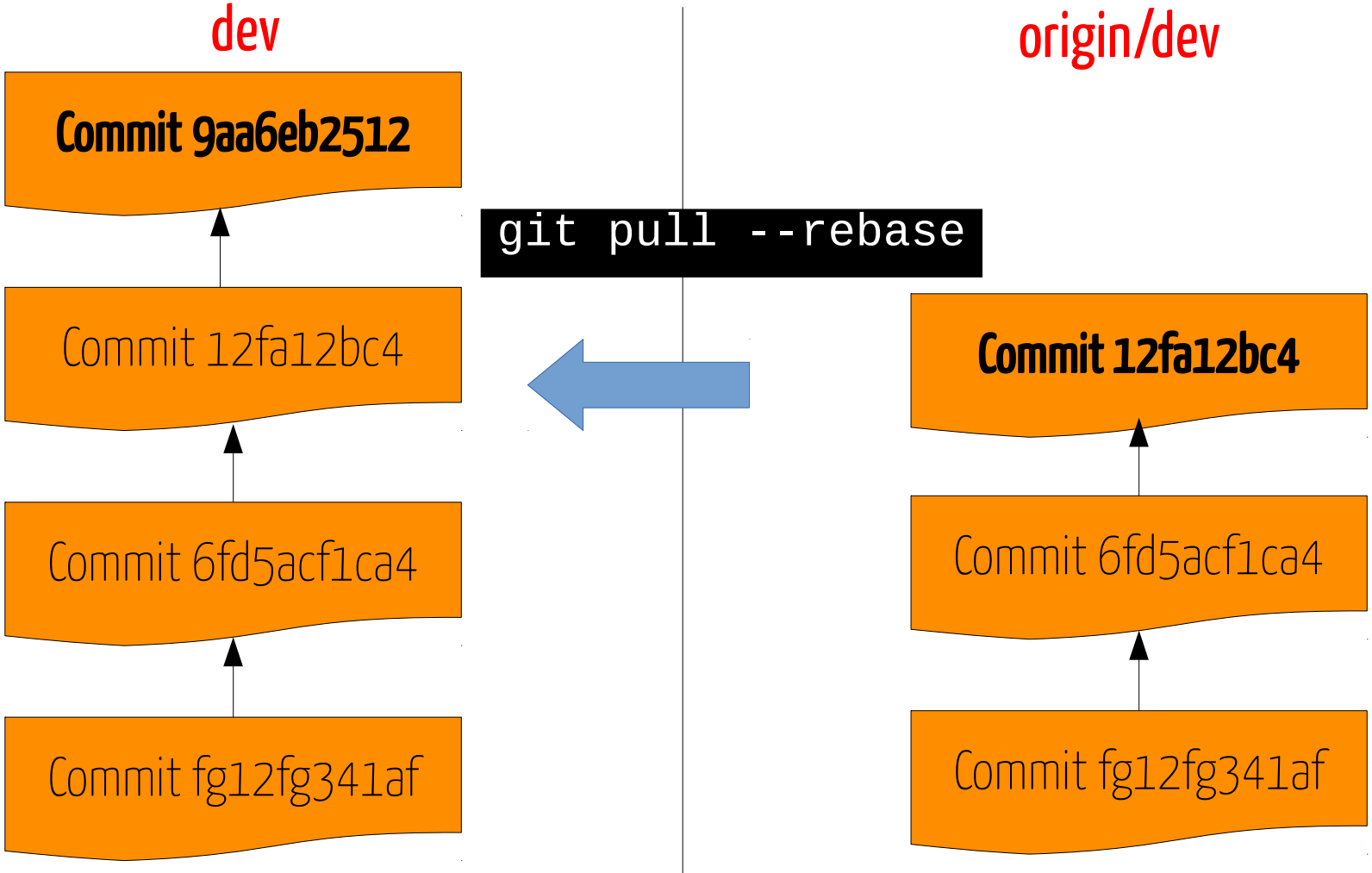
- un commit a été réalisé pendant que le commit 8f4ba était exécuté
- un push échouera car l'historique n'est pas cohérent
- les branches ne sont pas non-fast-forward



hupstream

Le rebase de branches locale / distante

- `git pull –rebase` réécrit l'historique et positionne le commit 12fa12 avant le commit 9aa6eb2512 (anciennement 8f4bae12fg)



hupstream

Résoudre un conflit lié au rebase

Message typique de conflit

```
error: could not apply fa39187... something to add to patch A

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing,
run "git rebase --abort".
Could not apply fa39187f3c3dfd2ab5faa38ac01cf3de7ce2e841...
Change fake file
```

Résoudre le conflit

- éditer le fichier
- ajouter au cache
- finaliser le rebase

```
git rebase --continue
```

- abandonner le rebase

```
git rebase --abort
```

Rebase interactif

Rebase simple / interactif

- rebase simple : réorganisation des commits
- rebase interactif : édition de branches (suppression, édition, fusion, modification de l'ordre des commits)

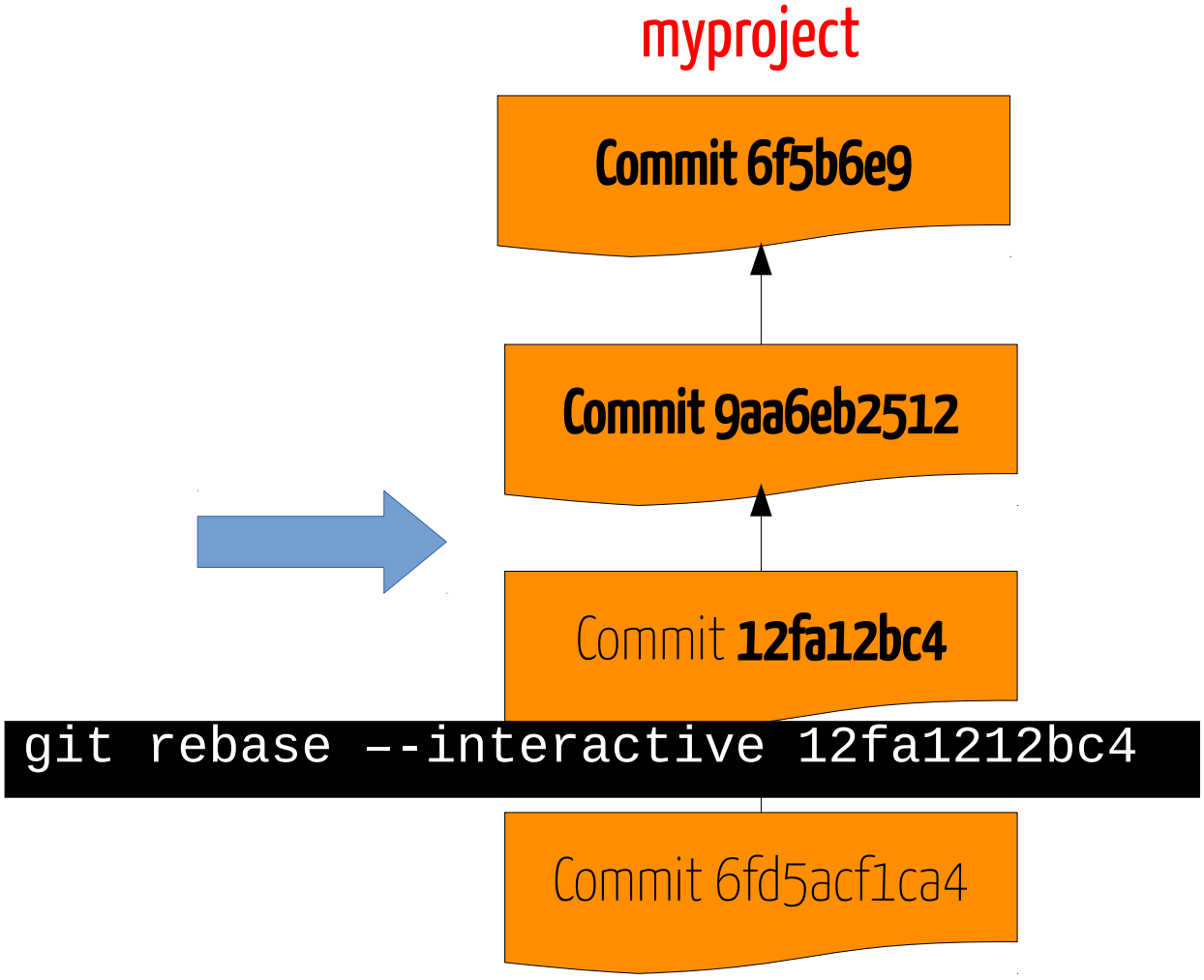
Attention : seuls les commits non partagés doivent être rebasés

Les différents modes

- **pick** inclure le commit :
- **reword** reformuler le message de commit
- **edit** éditer le contenu du commit, spliter...
- **squash** fusionner 2 commits ou plus et réécrire le message de commit
- **fixup** idem que squash mais seul un message de commit est conservé

Rebase interactif : lancement

- objectif : modifier le commit 9aa6eb2512
- Sélectionner le commit précédent celui à modifier

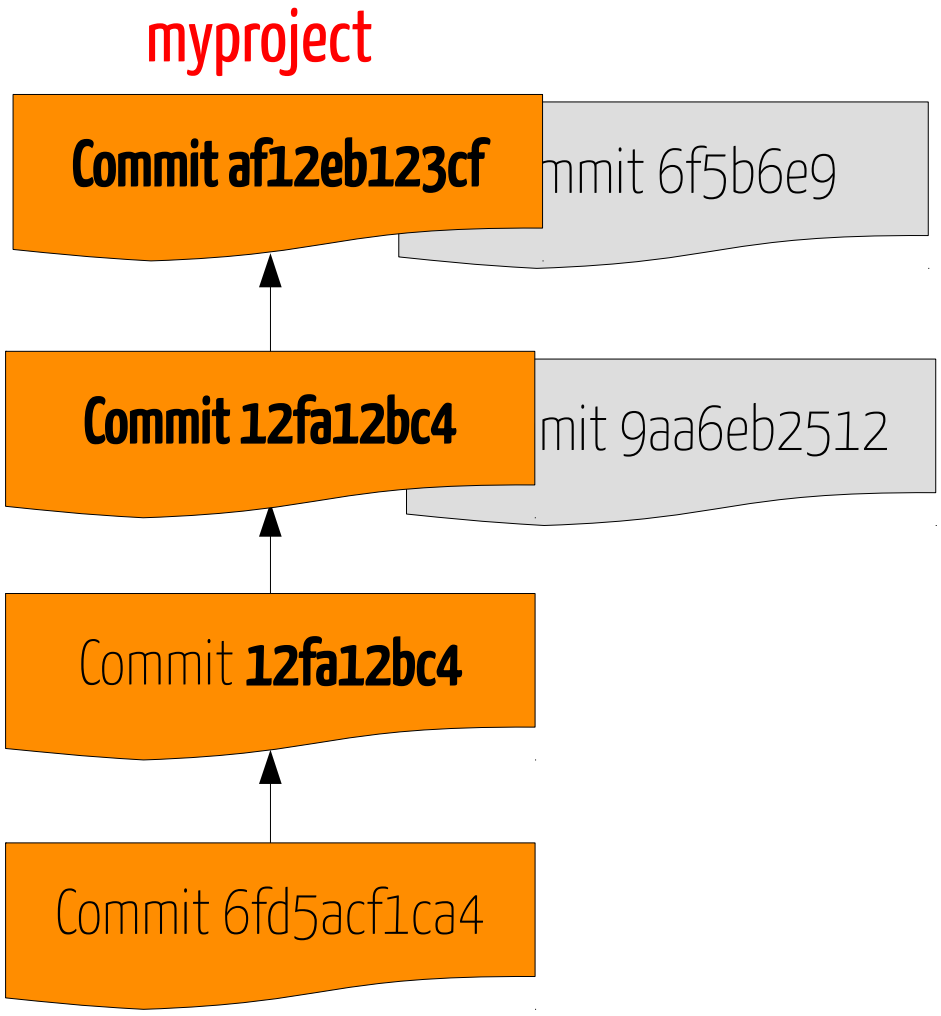


hupstream

Rebase interactif : exécution

- modification du message de commit

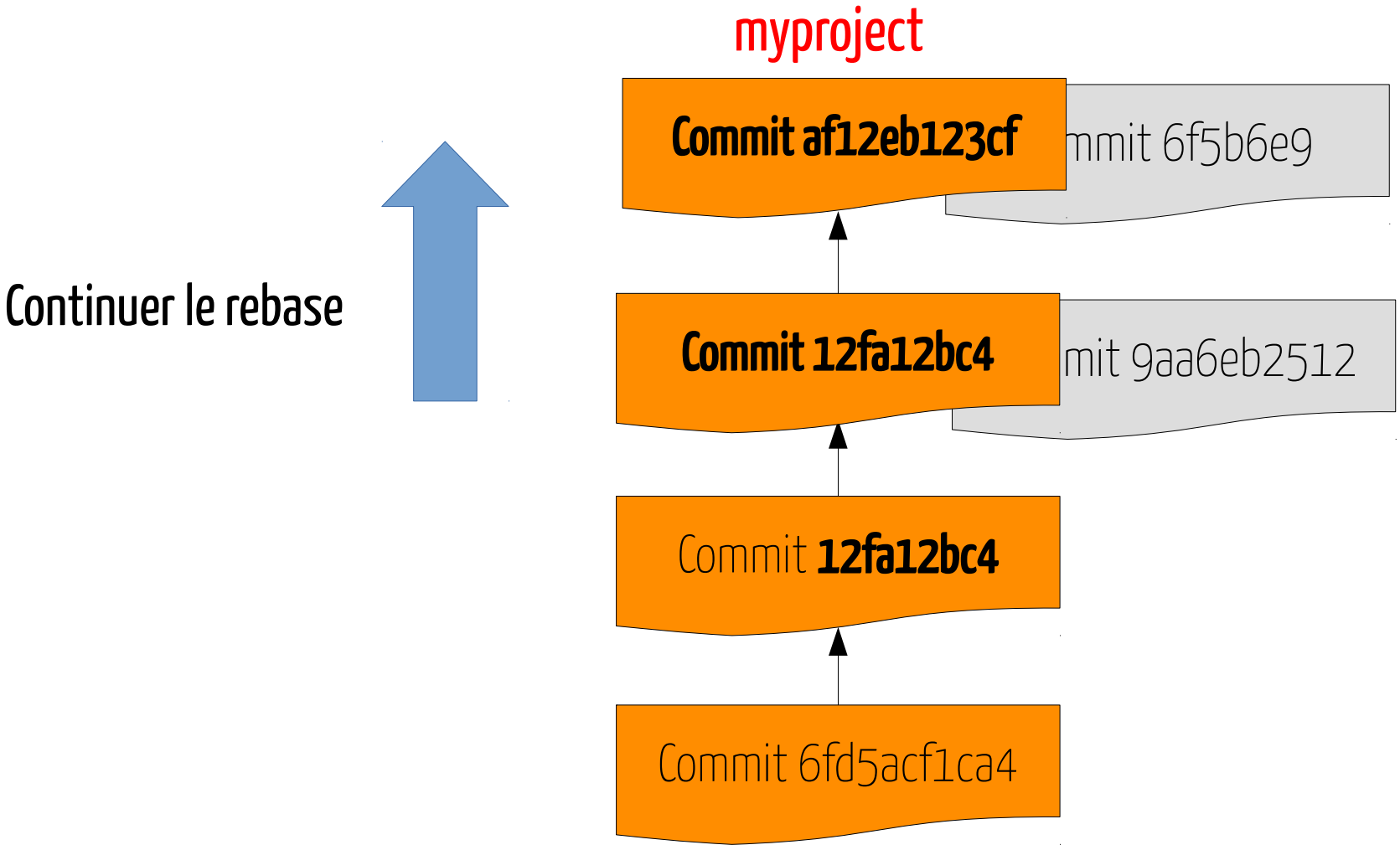
```
reword
commit 9aa6eb
```



hupstream

Rebase interactif : exécution

- modification du message de commit



hupstream

Résoudre un conflit lié au rebase

Message typique de conflit

```
error: could not apply fa39187... something to add to patch A

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing,
run "git rebase --abort".
Could not apply fa39187f3c3dfd2ab5faa38ac01cf3de7ce2e841...
Change fake file
```

Résoudre le conflit

- éditer le fichier et ajouter au cache
- finaliser le rebase

```
git rebase --continue
```

- abandonner le rebase

```
git rebase --abort
```

hupstream

Rebase interactif : résumé

```
git rebase --interactive <commit_id>
```

- ne pas rebaser des commits qui sont déjà dans la branche de destination
- ne rebaser que le travail non mergé

- Rebase et bonnes pratiques :
  https://lkml.org/lkml/2008/2/12/377

Rebase interactif : résumé

- Exécuter un rebase **sur une branche séparée**
  Si le rebase n'est pas correct, repartir de la branche originale
- Utile pour **retravailler le message de commit** ou le titre
- Utile pour **repenser le split des commits**
  Certains commits doivent être fusionnés pour plus de cohérence / atomicité
- Un bon moyen de **préparer une branche** pour un pull request

8. Git commit - Pratiques avancées

hupstream

Message de commit

Mettre son travail de côté : git stash

Le stash est un moyen

- de stopper temporairement le code en cours de développement et y revenir plus tard
- de permettre un pull avec la présence de modifications locales

- Ajouter les changements en cours à la zone de cache

```
[ennael@localhost BCD (master)]$ git add .
```

- Ajouter les changements en cours à la zone de stash

```
[ennael@localhost BCD (master)]$ git stash
Saved working directory and index state WIP on master: 6cf1288 add comment
HEAD is now at 6cf1288 add comment
```

- Vérifier le status courant de la zone de stash

```
[ennael@localhost BCD (master)]$ git stash list
stash@{0}: WIP on master: 6cf1288 add comment
```

Mettre son travail de côté : git stash

- De retour au code, récupération de ce qui a été mis de côté

```
[ennael@localhost BCD (master)]$ git stash apply
Auto-merging create_dual.sh
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   create_dual.sh
#
no changes added to commit (use "git add" and/or "git commit -a")
```

hupstream

Mettre son travail de côté : git stash

- ajouter un stash

```
git stash
```

ou

```
git stash save “message stash”
```

- lister les entrées de stash

```
git stash list
```

Mettre son travail de côté : git stash

- récupérer le contenu du stash

```
git stash apply stash@{x}
```

- détruire une entrée du stash

```
git stash drop stash@{x}
```

- appliquer et supprimer

```
git stash pop stash@{x}
```

- récupérer une zone de stash dans une branche

```
git stash branch <nom_de_la_branche> stash@{x}
```

Récupérer un ou plusieurs commits : cherry picking

Pourquoi ?

Applique les changements générés par un commit réalisé dans une branche vers une autre branche

Comment ça fonctionne ?

- Récupérer un commit
- Rejouer le commit
- Construire un nouveau commit avec un ID différent (même changement mais parent différent)

Attention : cela peut rendre les 2 branches non mergeables

hupstream

Cherry picking : utilisation

Comment

- déterminer le hash du commit à appliquer dans la branche souhaitée
- appliquer ce commit à la branche finale

```
git cherry-pick [--no-commit] <SHA1>...
```

En cas de conflit

- résolution du problème

```
git cherry-pick –-continue
git cherry-pick –-abort
git cherry-pick --quit
```

Bonnes pratiques : le message de commit

Commit Often,

Perfect Later,

Publish Once

Bonnes pratiques : le message de commit

Un header soigné

- Décrire le commit en une ligne sans rentrer dans les détails
- La seconde partie du message contient tous les détails
- Ajouter des informations à propos du fichier ou de la partie du fichier qui a été modifiée

Bonnes pratiques : le message de commit

Etre compréhensible

- Détailler la problématique, la solution apportée et l'implémentation de cette solution
- Il n'y a jamais trop de détails
- Toujours donner de l'information hormis pour les merges

Bonnes pratiques : le message de commit

Longueur des lignes

- Lisibilité
- Utilisation des paragraphes
- 80 colonnes

Bonnes pratiques : le message de commit

Des commits atomiques

- Simplifie les revues de code
- Simplifie le debug en utilisant `git diff` ou `git bisect`
- Trouver un changement beaucoup plus rapidement pour toute opération comme un revert
- 1 commit = 1 fonctionnalité

Corriger son dernier commit

Modifier votre dernier commit

- modifier le message de commit
- ajouter le contenu du cache

```
git commit --amend
```

Défaire un commit

Supprimer un ou plusieurs commits

- joue à l'envers le commit
- utilisable sur les commits déjà partagés

```
git revert <SHA1>...
```

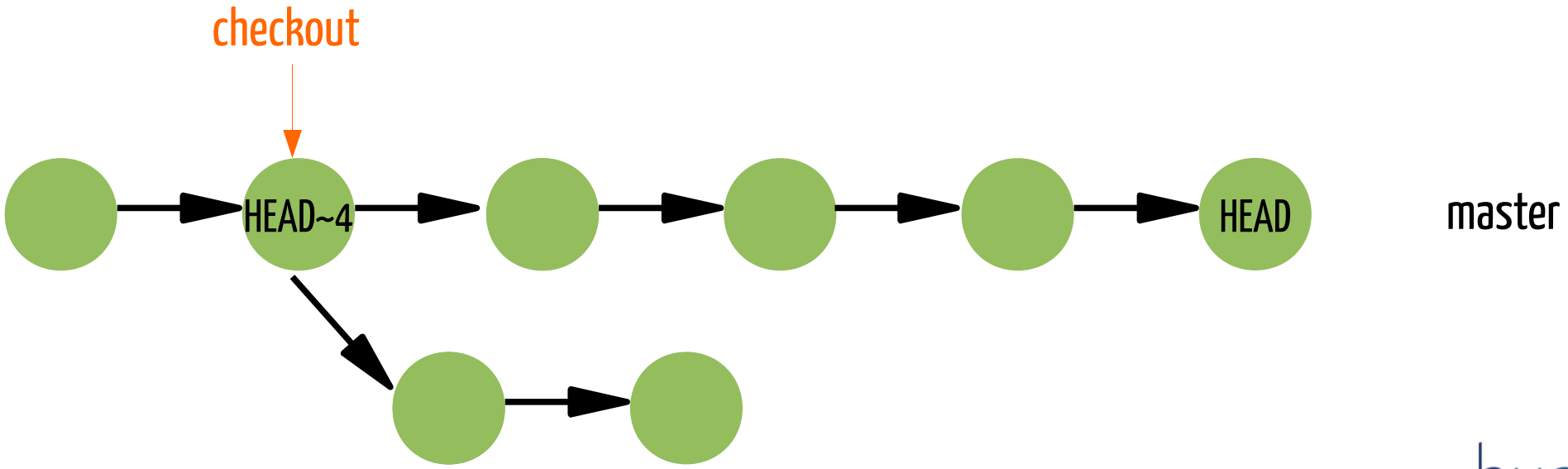
9. Debugger son code

hupstream

Head détaché

Se positionner sur un commit non référencé dans l'historique

```
[ennael@localhost bcd]$ git checkout HEAD~4
...
HEAD est maintenant sur a67a04b... ajout de la variable ENV
[ennael@localhost bcd ((a67a04b...))]$
```

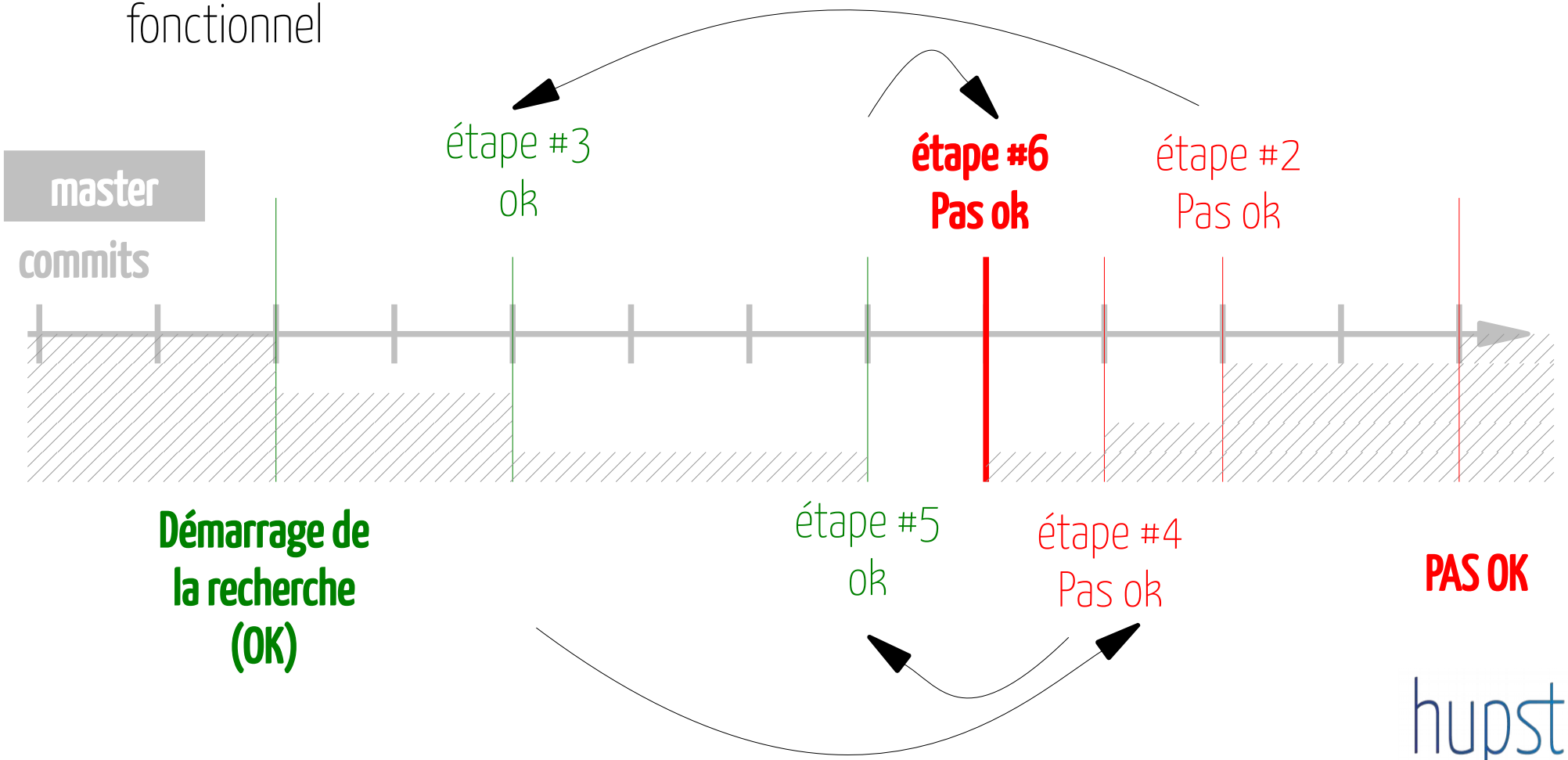


hupstream

Git et la recherche dichotomique

Fonctionnement de la recherche dichotomique

Utiliser les logs de git pour trouver un numéro de commit où le code était fonctionnel



hupstream

git bisect

Ce qu'il permet

- Passer automatiquement d'un commit à un autre
- Pas de recherche de commit dans l'ordre, mais "couper la poire en 2"
- Travaille automatiquement dans une branche séparée sans aucun risque pour le code courant
- Permet des tests automatiques

hupstream

Git bisect

- trouver un commit correspondant à un code fonctionnel
- trouver un commit correspondant à un code non fonctionnel

Démarrer l'utilisation de bisect pour trouver le commit erroné

- initialisation de git bisect

```
[ennael@localhost BCD (master)]$ git bisect start HEAD 175b8ed
```

- enregistrement du commit correspondant au code fonctionnel

```
[ennael@localhost BCD (master|BISECTING)]$ git bisect good
```

- et du commit correspondant au code non fonctionnel

```
[ennael@localhost BCD (master|BISECTING)]$ git bisect bad
Bisecting: 2 revisions left to test after this (roughly 1 step)
[e4808387253d2a7d6dfe07f16db33a08981e5088] clean Makefile
```

hupstream

Git bisect

Récupération du commit erroné

Dernier round

```
[ennael@localhost BCD ((cb61c72...)|BISECTING)]$ git bisect bad
e4808387253d2a7d6dfe07f16db33a08981e5088 is the first bad commit
commit e4808387253d2a7d6dfe07f16db33a08981e5088
Author: Anne Nicolas <ennael@mageia.org>
Date:   Thu Dec 19 10:02:20 2013 +0100

    clean Makefile

:100644 100644 3bb87f41120596734203c24fc2a32afa445b6249
2c180507d224d38093c98f413f81d07e298357ec M      Makefile
```

hupstream

Git bisect

Sortir de git bisect

```
[ennael@localhost BCD ((cb61c72...)|BISECTING)]$ git bisect reset
Previous HEAD position was cb61c72... comment svn extract as it's not used
anymore
Switched to branch 'master'
[ennael@localhost BCD (master)]$
```

hupstream

Automatiser git bisect

Utiliser un script pour automatiser la recherche du commit erroné

Prérequis

- Inclure toutes les conditions de tests dans le script
- Le script testera le code en utilisant les codes de retour d'erreurs suivant :

0	succès
!= 0	échec

Automatiser git bisect

Démarrer avec les étapes précédentes

- Démarrer git bisect en fournissant l'intervalle de travail

```
[ennael@localhost BCD (master)]$ git bisect start HEAD 175b8ed
```

- exécuter votre script et laisser git bisect trouver le coupable

```
[ennael@localhost BCD ((188f1a3...)|BISECTING)]$ git bisect run ./test.sh
running ./test.sh
...
5ffb62d24cb317e4c754e5c65843647397063ff4 is the first bad commit
commit 5ffb62d24cb317e4c754e5c65843647397063ff4
Author: Anne Nicolas <ennael@mageia.org>
Date:   Thu Dec 19 11:15:56 2013 +0100

    add comment on plop file test
```

git est maintenant votre ami !