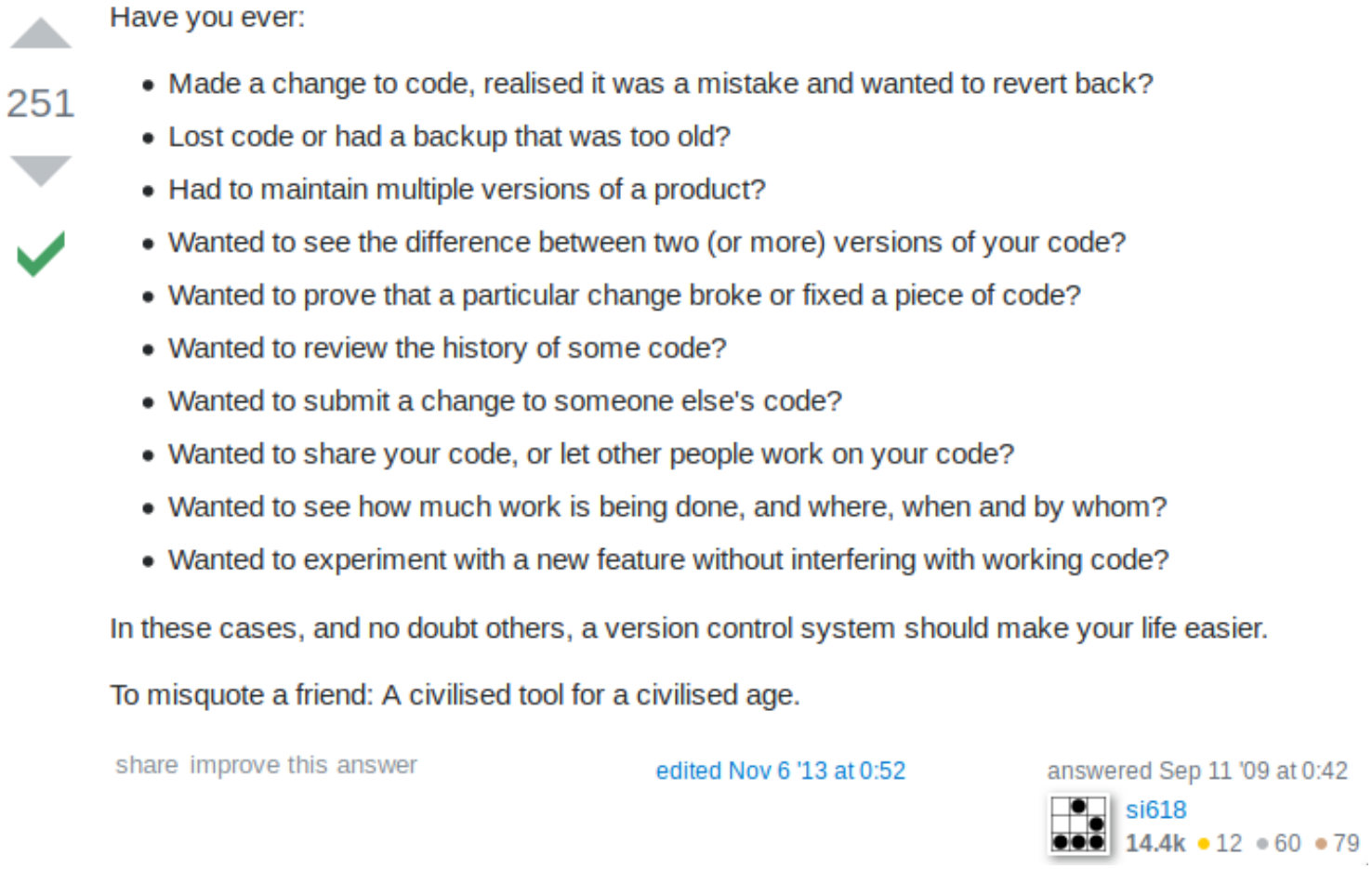
Git, Essayons de reprendre le contrôle !

Antoine de ROQUEMAUREL
satenske

Développeur Java consultant chez Tech Advantage

Meetup Java / C# du 28 Mars 2019

Le logiciel de gestion de versions



251

Have you ever:

- Made a change to code, realised it was a mistake and wanted to revert back?
- Lost code or had a backup that was too old?
- Had to maintain multiple versions of a product?
- Wanted to see the difference between two (or more) versions of your code?
- Wanted to prove that a particular change broke or fixed a piece of code?
- Wanted to review the history of some code?
- Wanted to submit a change to someone else's code?
- Wanted to share your code, or let other people work on your code?
- Wanted to see how much work is being done, and where, when and by whom?
- Wanted to experiment with a new feature without interfering with working code?

In these cases, and no doubt others, a version control system should make your life easier.

To misquote a friend: A civilised tool for a civilised age.

share improve this answer

edited Nov 6 '13 at 0:52

answered Sep 11 '09 at 0:42

si618

14.4k ● 12 ● 60 ● 79

Figure – Pourquoi devrais-je utiliser le contrôle de version ? [1]

1. https://stackoverflow.com/questions/1408450/why-should-i-use-version-control

Git

- Créé en 2005 par Linus Torvalds
- Décentralisé
- Excellente gestion des branches
- Efficace sur de gros projet

Git

- Créé en 2005 par Linus Torvalds
- Décentralisé
- Excellente gestion des branches
- Efficace sur de gros projet
  - Microsoft Windows :
    - 3 500 000 fichiers, soit 300 Go
    - 440 branches
    - 4 000 utilisateurs
    - 10 000 merges

Git

- Créé en 2005 par Linus Torvalds
- Décentralisé
- Excellente gestion des branches
- Efficace sur de gros projet

« *I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'.* »

WORKFLOW CENTRALISÉ

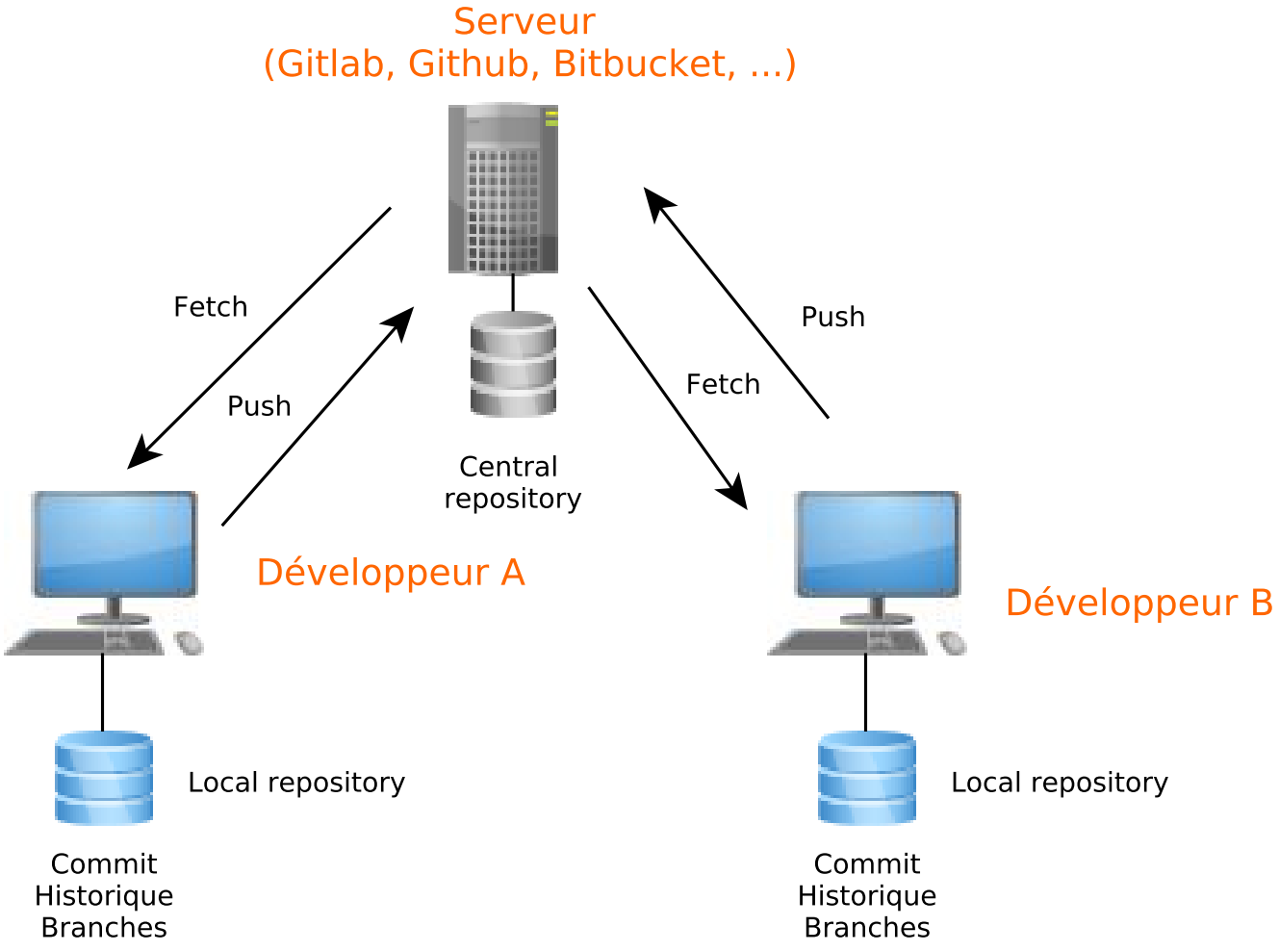


FIGURE – Système décentralisé

Décentralisé : Integration-Manager

- Souvent utilisé pour des projets publics
  - Github, Gitlab, ...

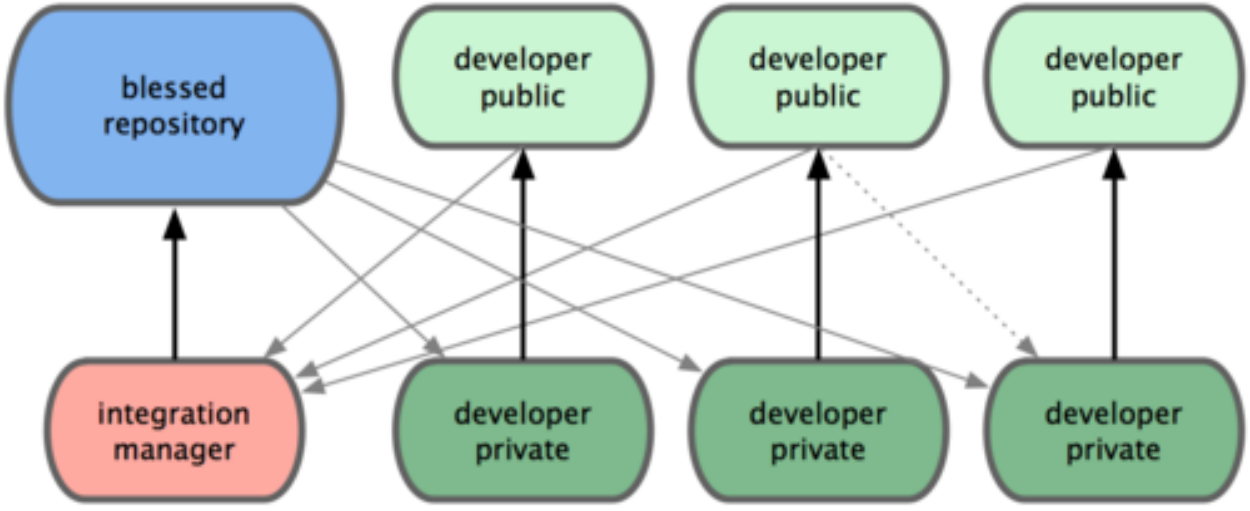


Figure – Integration-Manager Workflow [2]

2. https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows

Décentralisé : Dictateur et lieutenant

- Très gros projets avec des centaines de collaborateurs
  - Kernel Linux

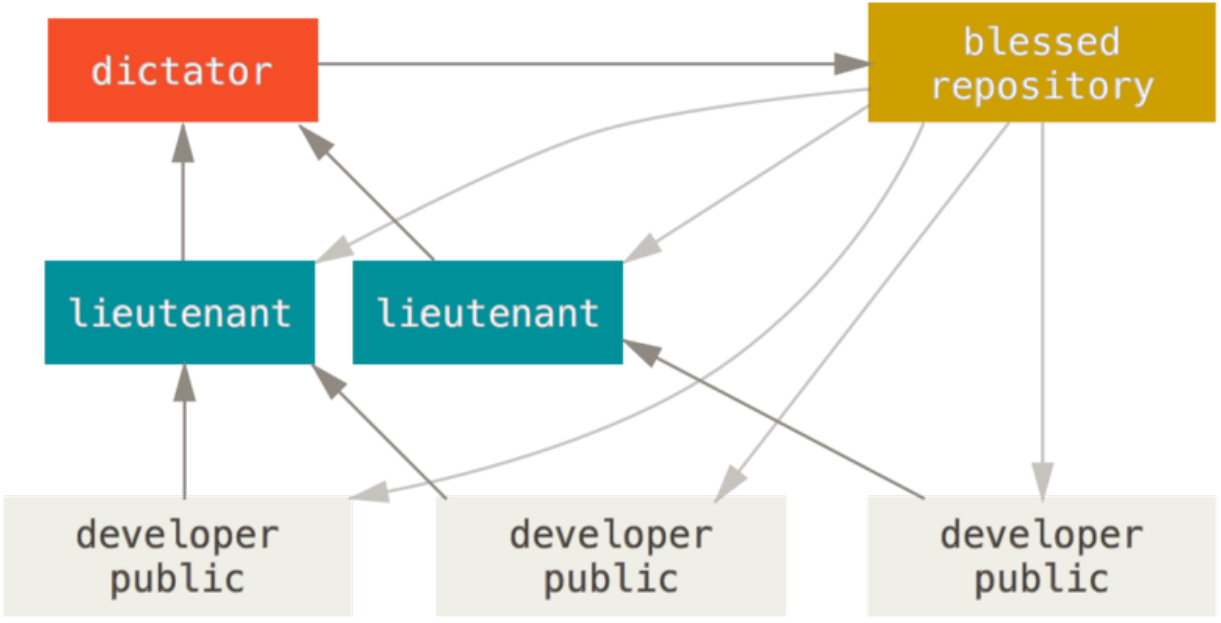


Figure – Dictateur et lieutenant Workflow [3]

3. https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows

La zone de transit (*staging area*)

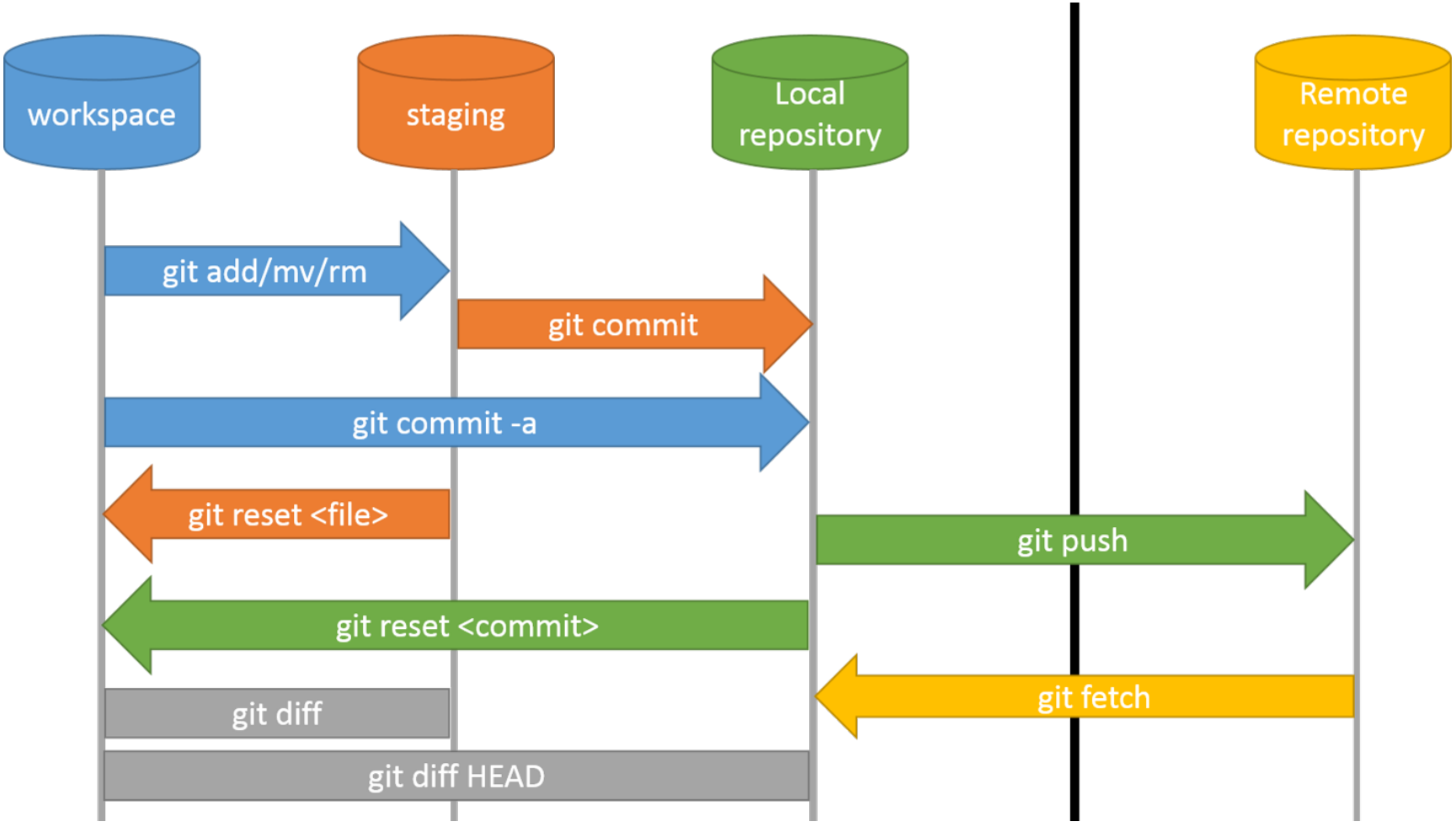


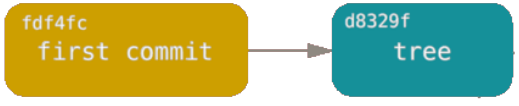
Figure – Fonctionnement de Git

Le stockage des données du dépôt : les objets

- Les changements de fichiers sont stockés dans le **commit**

Le stockage des données du dépôt : les objets

- Les changements de fichiers sont stockés dans le **commit**
- Un **tree** peut-être vu comme un répertoire
  - il référence d'autres tree ou des blobs

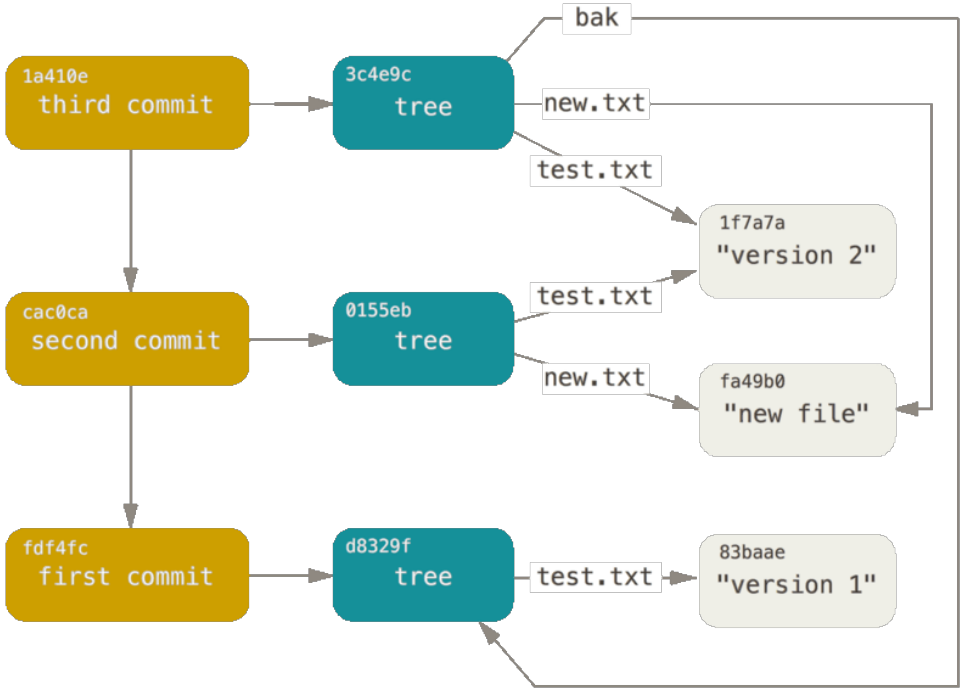


Le stockage des données du dépôt : les objets

- Les changements de fichiers sont stockés dans le **commit**
- Un **tree** peut-être vu comme un répertoire
  - il référence d'autres tree ou des blobs
- Chaque version de chaque fichier est un **blob**

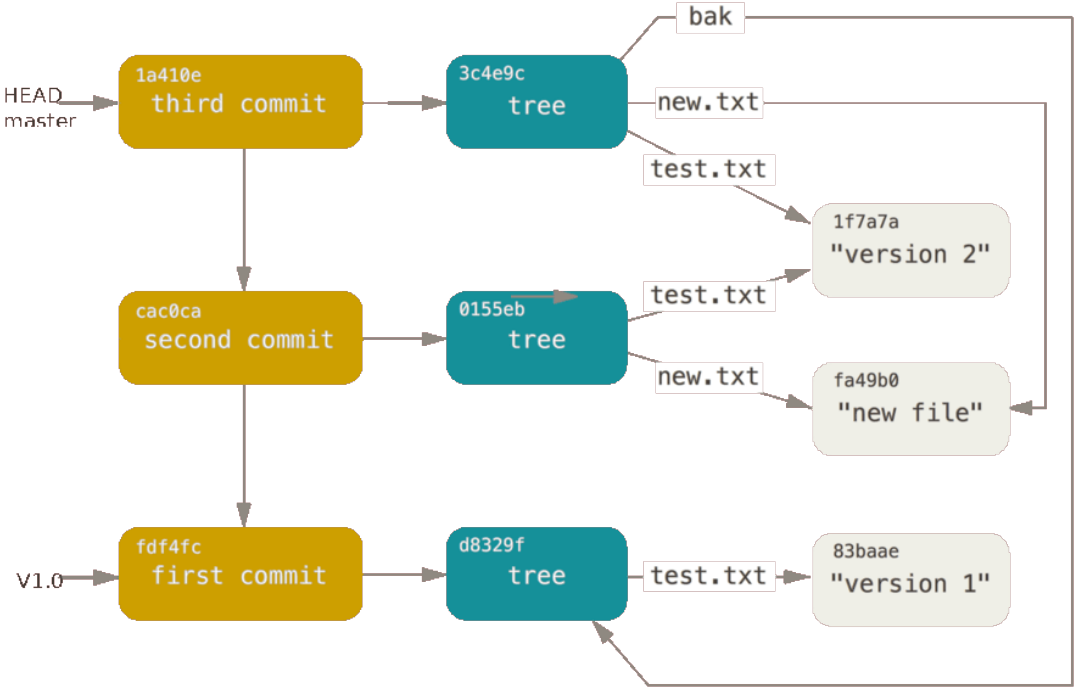
Le stockage des données du dépôt : les objets

- Les changements de fichiers sont stockés dans le **commit**
- Un **tree** peut-être vu comme un répertoire
  - il référence d'autres tree ou des blobs
- Chaque version de chaque fichier est un **blob**
- Les **commits** sont chainés entre eux



Le stockage des données du dépôt : les objets

- Les changements de fichiers sont stockés dans le **commit**
- Un **tree** peut-être vu comme un répertoire
  - il référence d'autres tree ou des blobs
- Chaque version de chaque fichier est un **blob**
- Les **commits** sont chainés entre eux
- Les **tags**, **branches** et **HEAD** sont des pointeurs de commit



Git débite à la hash !

- Le nom unique de chaque objet est obtenu avec `SHA-1`
  - Valeur sur 160 bits (nombre hexadécimal à 40 chiffres)

Git débite à la hash !

- Le nom unique de chaque objet est obtenu avec `SHA-1`
  - Valeur sur 160 bits (nombre hexadécimal à 40 chiffres)

- Tout objet du store est adressable par son contenu

Git débite à la hash !

- Le nom unique de chaque objet est obtenu avec `SHA-1`
  - Valeur sur 160 bits (nombre hexadécimal à 40 chiffres)
- Tout objet du store est adressable par son contenu
- Toute modification du contenu produira un changement du hash

Le stockage des infos du workspace : l'index

- Le fichier contient toutes les informations nécessaires à la génération d'un *tree object*
- Il permet la comparaison rapide entre un *tree object* et le *working tree*
- Il contient les informations sur les *merges conflicts*

Bien rédiger son message de commit

- Bien configurer son environnement : nom et adresse mail

Bien rédiger son message de commit

- Bien configurer son environnement : nom et adresse mail
- Séparer le sujet du corps avec une ligne vide

Bien rédiger son message de commit

- Bien configurer son environnement : nom et adresse mail
- Séparer le sujet du corps avec une ligne vide
- Limiter le sujet à 80 caractères

Bien rédiger son message de commit

- Bien configurer son environnement : nom et adresse mail
- Séparer le sujet du corps avec une ligne vide
- Limiter le sujet à 80 caractères
- Utiliser le corps pour expliquer quoi et pourquoi et non comment

RESTONS BRANCHÉS : LE GIT-FLOW

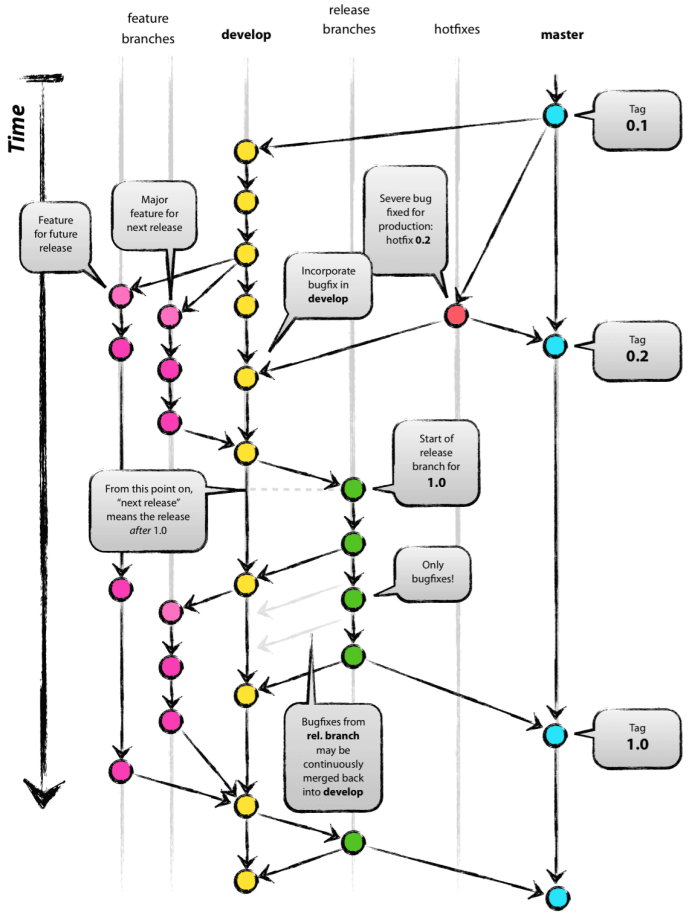
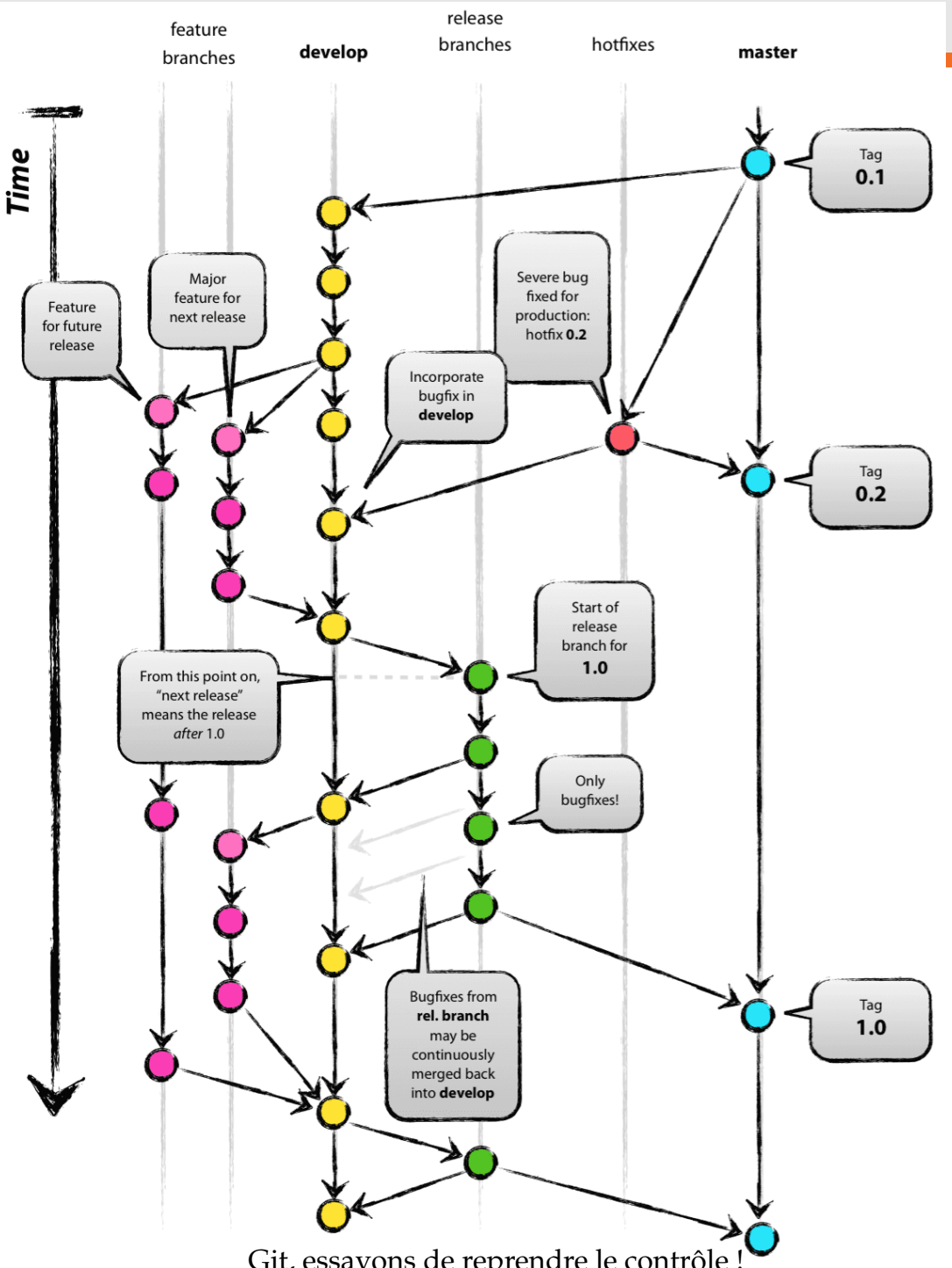


FIGURE – Un modèle de branchement [4]

4. https://nvie.com/posts/a-successful-git-branching-model/

RESTONS BI


feature branches
develop
release branches
hotfixes
master
Time
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.1
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

De l'importance de l'historique

Ingrédients :

- 200g de beurre
- 200g de chocolat
- 4 œufs
- 150g de sucre
- 60g de farine
- $\frac{1}{2}$ sachet de levure

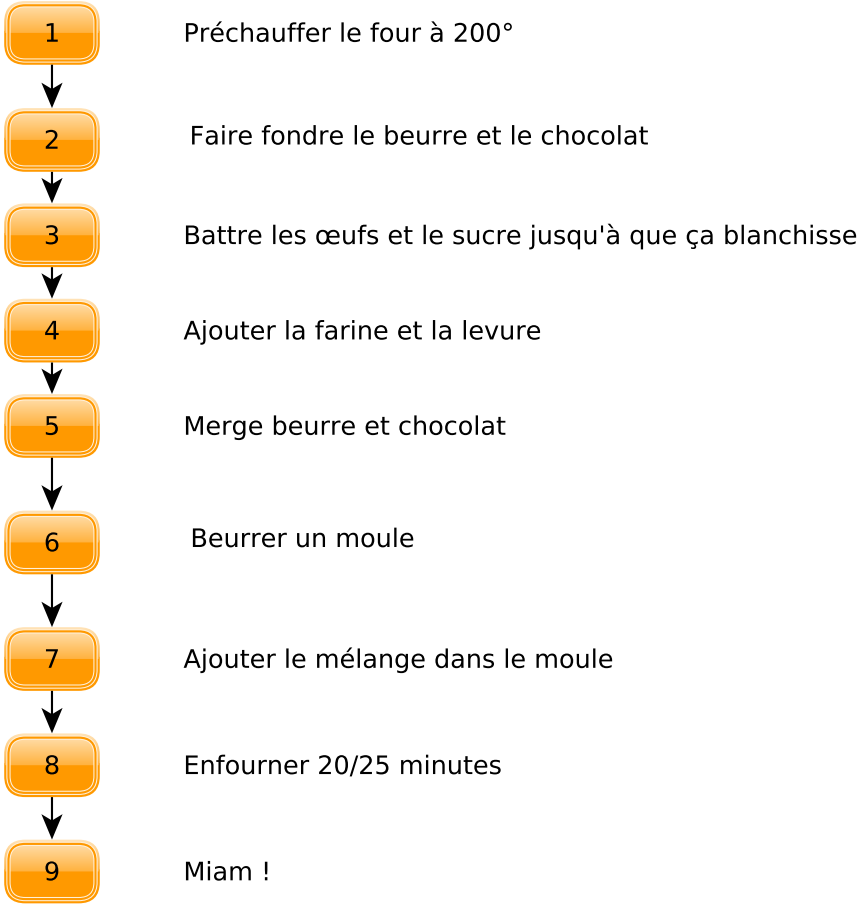
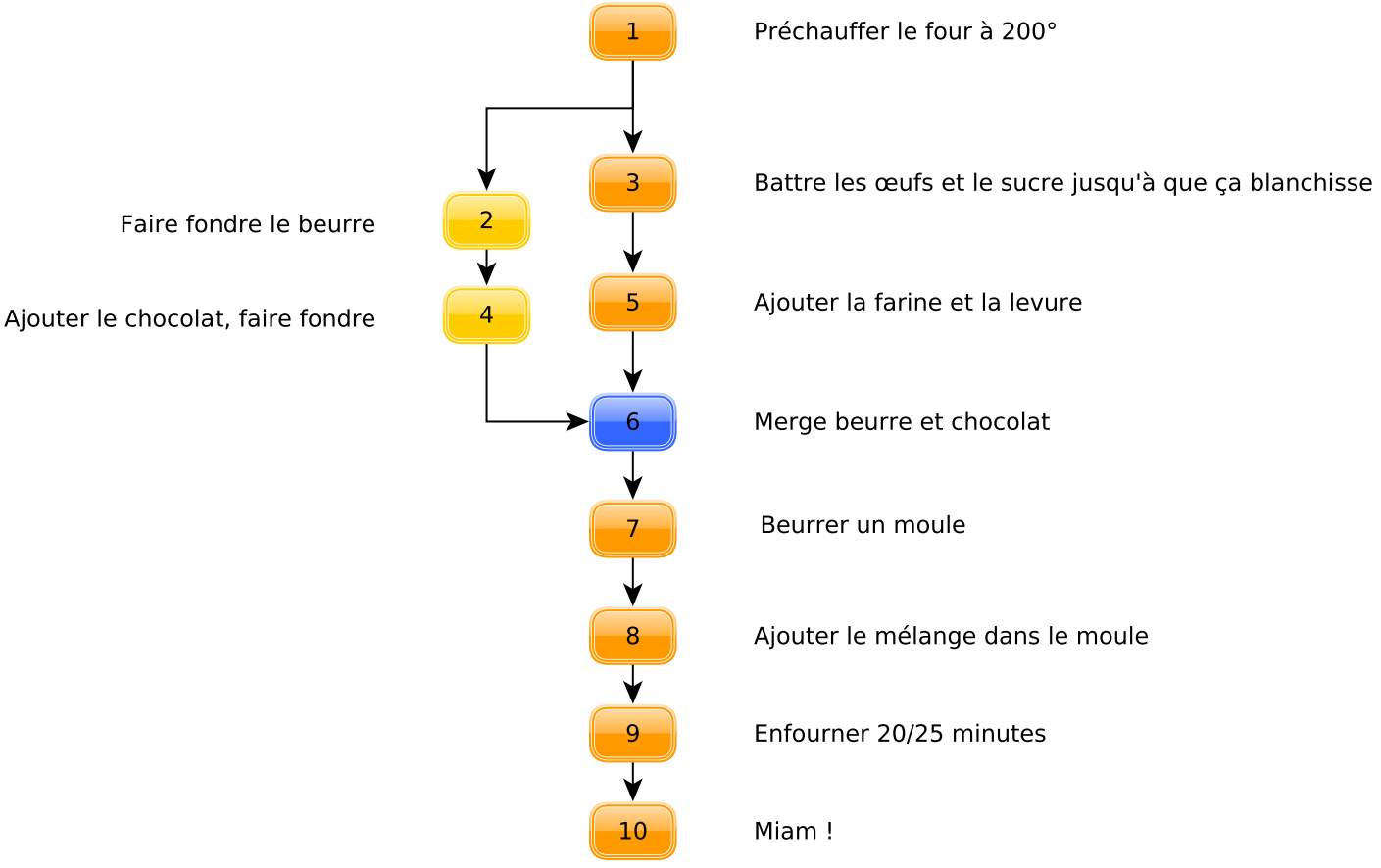


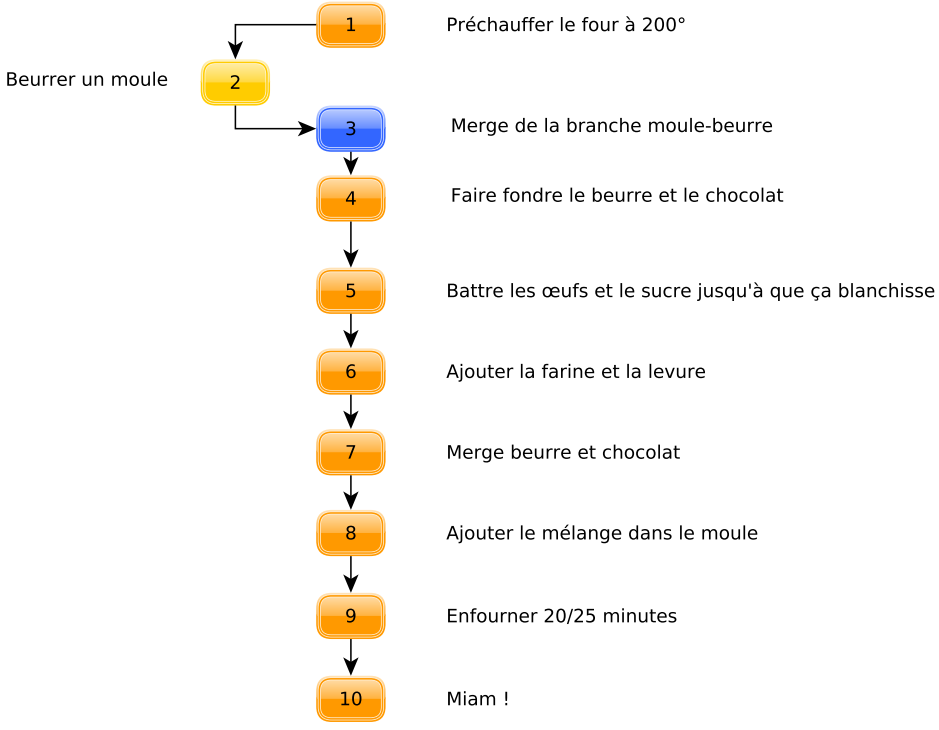
Figure – Historique sans collaboration

Le merge : on parralélise



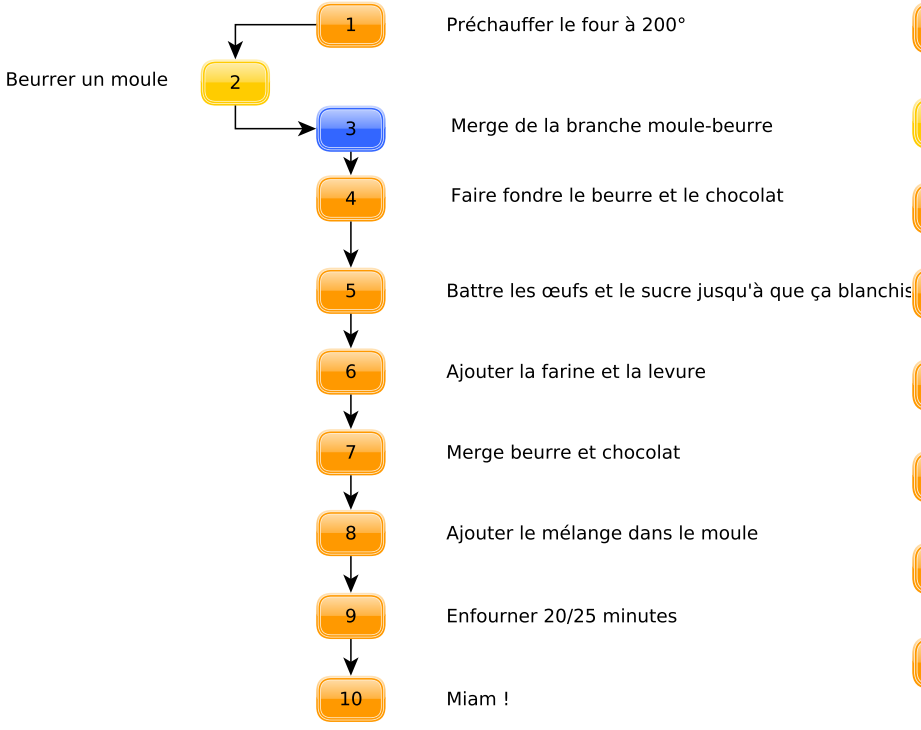

```
1 $ git merge beurre-chocolat
```

Le merge : le fast-forward




```
1 $ git merge --no-ff ↩
      moule-beurre
```

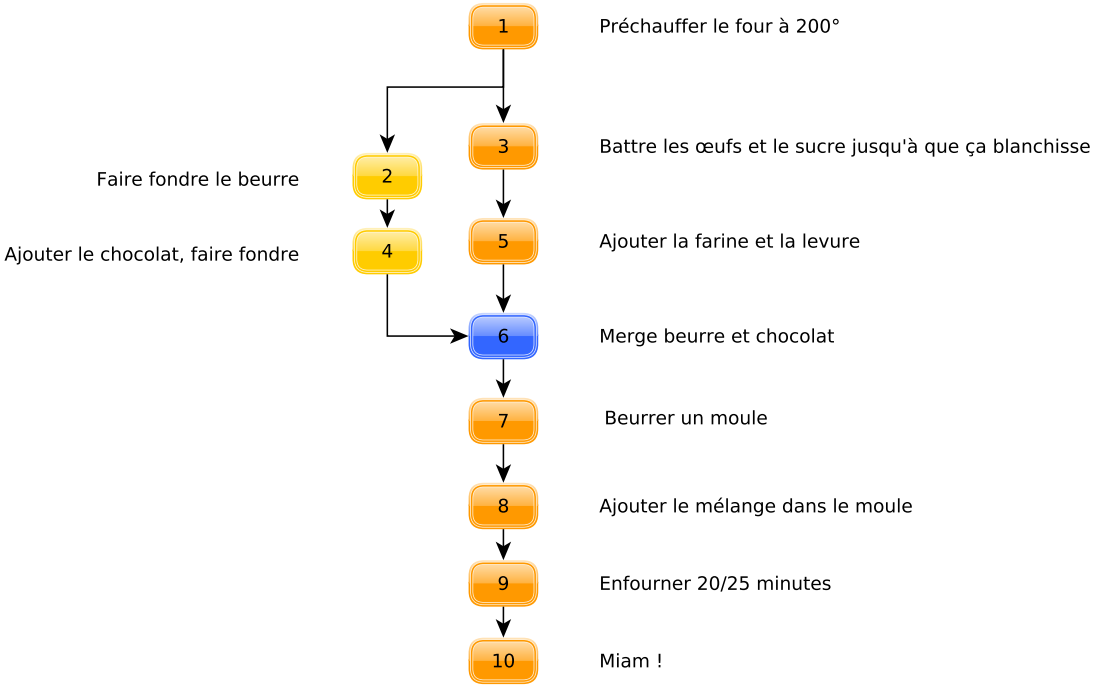
Le merge : le fast-forward




```
1 $ git merge --no-ff ↩
    moule-beurre
```

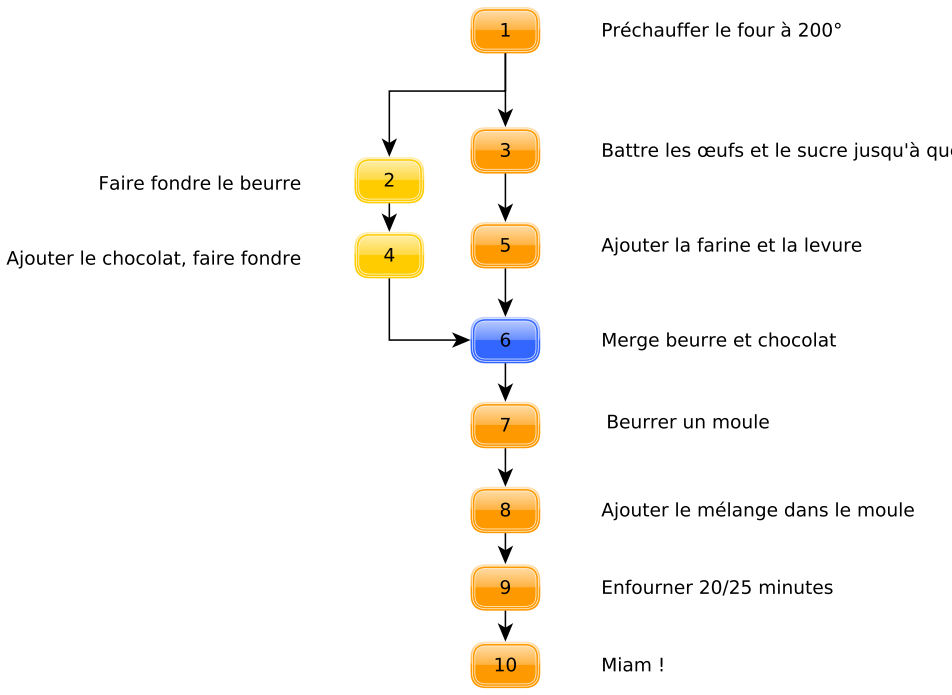

```
1 $ git merge moule-beurre
```

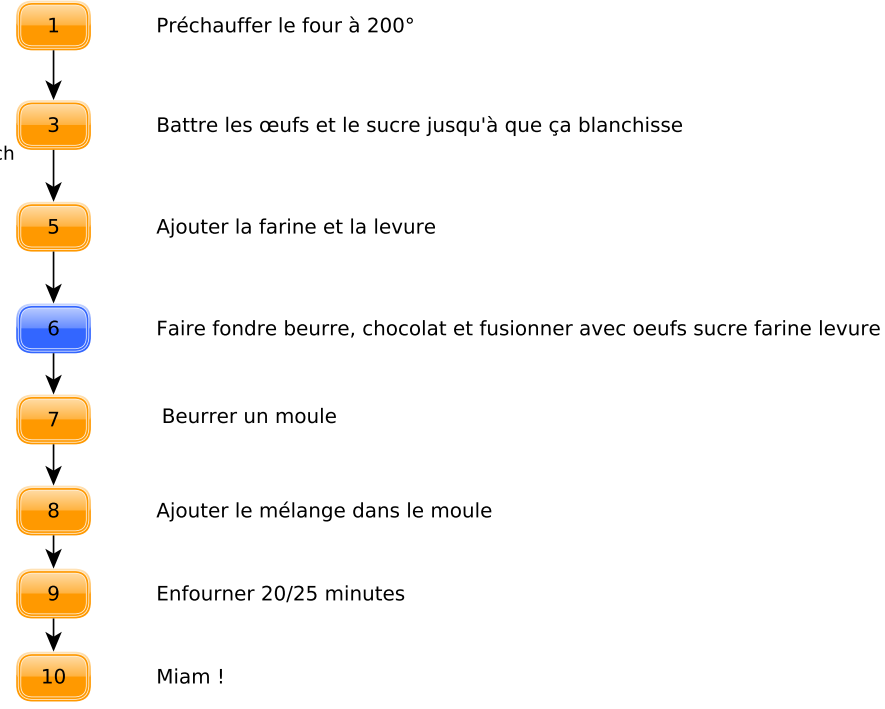
LE MERGE : LE SQUASH




```
1 $ git merge ↩
      beurre-chocolat
```

LE MERGE : LE SQUASH

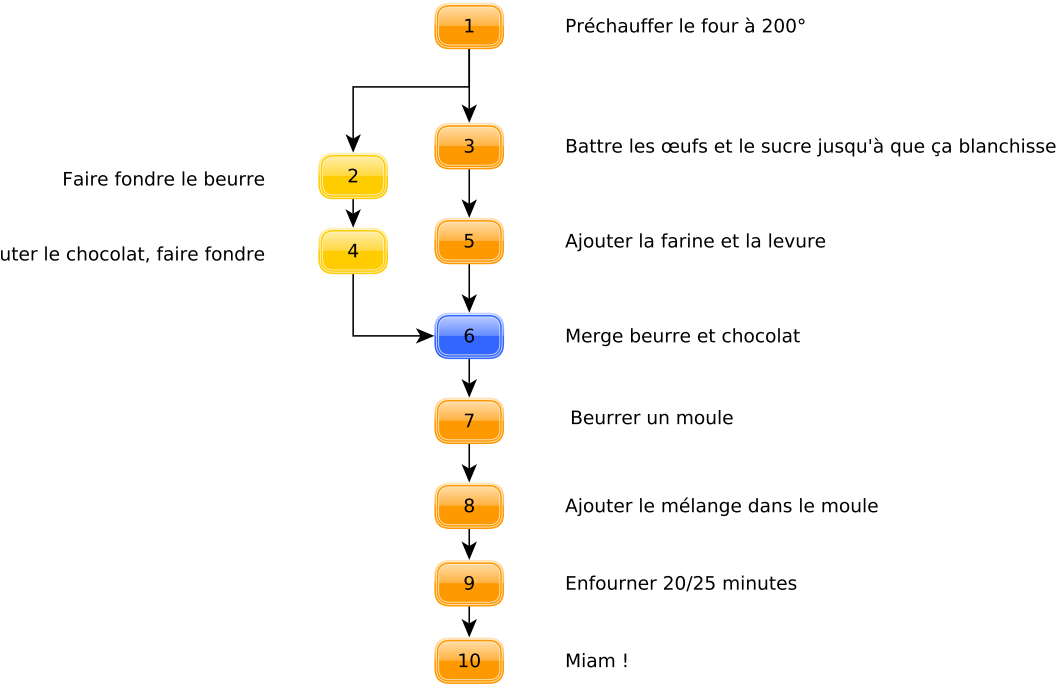





```
1 $ git merge ↩
      beurre-chocolat
```

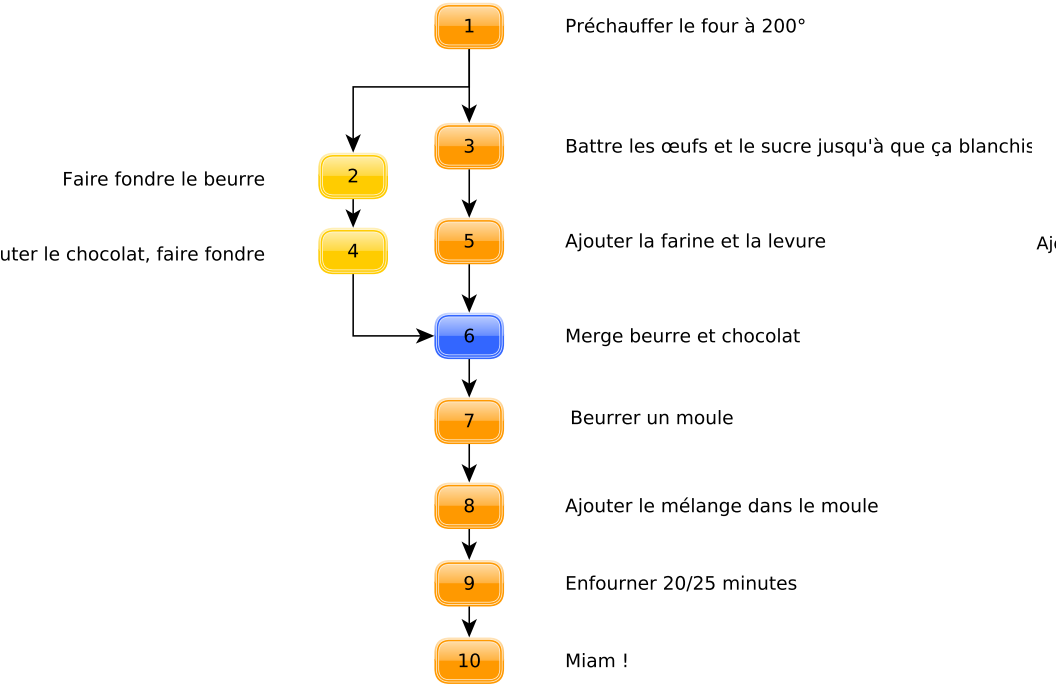
```
1 $ git merge --squash ↩
      beurre-chocolat
```

LE REBASE : ON VEUT UN HISTORIQUE LINÉAIRE

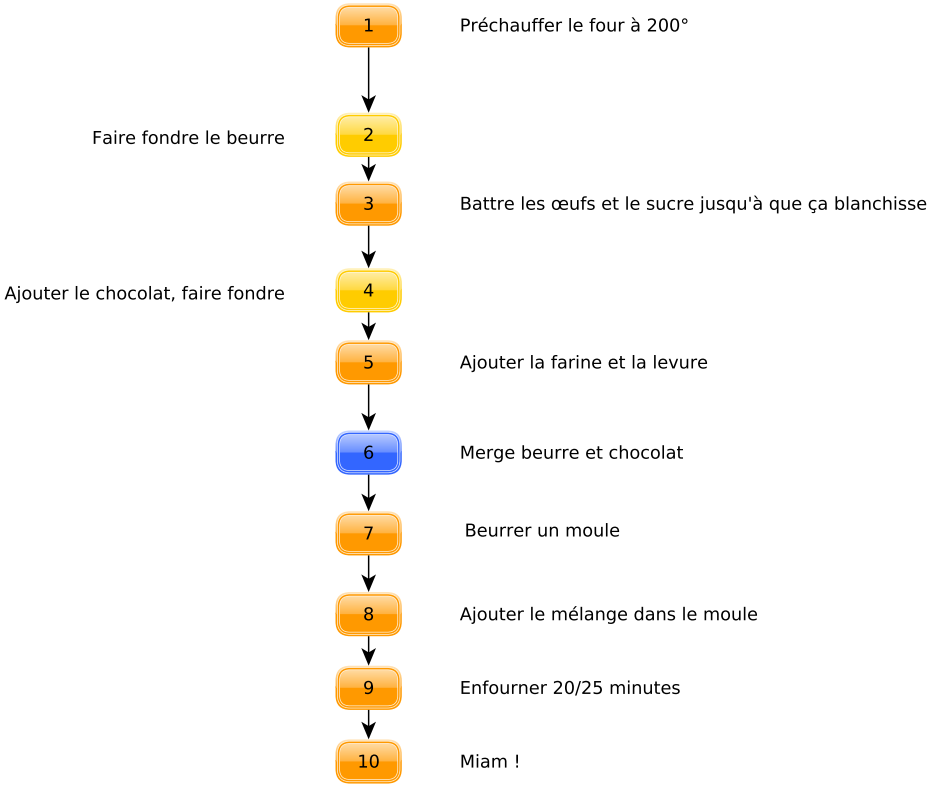



```
1 $ git merge ↩
      beurre-chocolat
```

LE REBASE : ON VEUT UN HISTORIQUE LINÉAIRE




```
1 $ git merge ↩
      beurre-chocolat
```




```
1 $ git rebase ↩
      beurre-chocolat
```

Le rebase : Attention à son utilisation

- Jamais sur une branche partagée
  - Un rebase réécrit l'historique et va donc changer les hash
- Si la branche à été poussée
  - Il faut faire un push force… Et donc écraser l'historique de la branche distante !
- Principalement à utiliser pour :
  - Mettre à jour sa branche par rapport à la branche mère
  - Réécrire son historique

Le stash : mettre des modifications de côté

- Utile pour pouvoir changer de brancher ou se mettre à jour sans commiter
- Gestion des conflits lors de l'application du stash

Le stash : mettre des modifications de côté

- Utile pour pouvoir changer de brancher ou se mettre à jour sans commiter
- Gestion des conflits lors de l'application du stash

- Le stash est une pile :
  - `git stash` pour empiler
  - `git stash pop` pour dépiler
- Possibilité d'y accéder comme une liste
  - `git stash list`
  - `git stash apply stash_name`

Le cherry-Pick : Réappliquer un commit

- Permet d'appliquer le « patch » d'un ou plusieurs commits sur une autre branche
- Cela va créer de nouveaux commits, avec des hashs différents
- Utile pour récupérer des modifications ponctuelles

Le cherry-Pick : Réappliquer un commit

- Permet d'appliquer le « patch » d'un ou plusieurs commits sur une autre branche
- Cela va créer de nouveaux commits, avec des hashs différents
- Utile pour récupérer des modifications ponctuelles

- `git cherry-pick commit`
  - applique le commit sur la branche courante
- `git cherry-pick commit1..commit2`
  - applique le range de commits sur la branche courante

LE PULL DEVRAIT ÊTRE INTERDIT. . .

- Par défaut, le pull peut être vu comme un alias :

```
1 $ git fetch
  $ git merge origin/master
```

Un git pull si on est sur master

LE PULL DEVRAIT ÊTRE INTERDIT. . .

- Par défaut, le pull peut être vu comme un alias :

```
$ git fetch
$ git merge origin/master
```

Un git pull si on est sur master

- Si on est pas en fast-forward, le merge va créer une branche temporaire

LE PULL DEVRAIT ÊTRE INTERDIT. . .

- Par défaut, le pull peut être vu comme un alias :

```
$ git fetch
$ git merge origin/master
```

Un git pull si on est sur master

- Si on est pas en fast-forward, le merge va créer une branche temporaire
- Ne faire un pull que si on est en fast forward

LE PULL DEVRAIT ÊTRE INTERDIT. . .

- Par défaut, le pull peut être vu comme un alias :

```
$ git fetch
$ git merge origin/master
```

Un git pull si on est sur master

- Si on est pas en fast-forward, le merge va créer une branche temporaire
- Ne faire un pull que si on est en fast forward
- Sinon, il faut faire

```
$ git fetch
$ git rebase origin/master
```

Se mettre à jour si on est sur master

Le rebase interactif : réécrire l'histoire

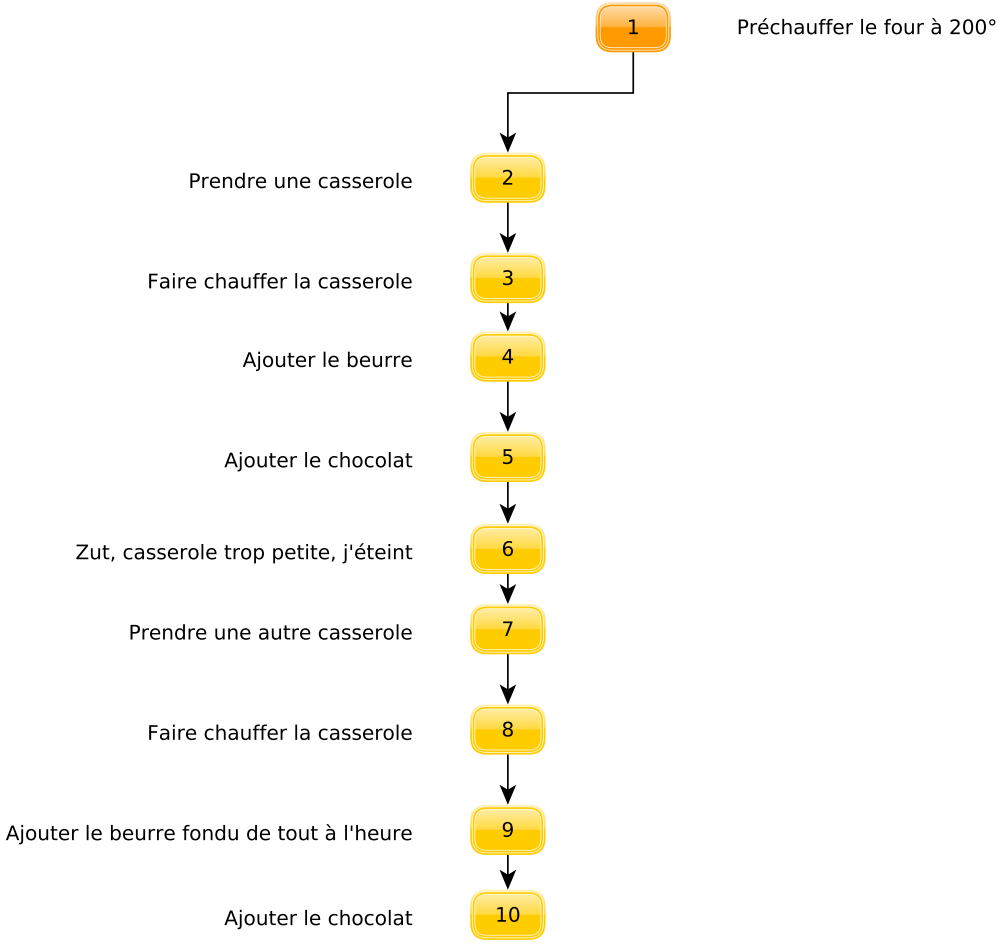


Figure – Un historique pourri

Le rebase interactif : réécrire l'histoire

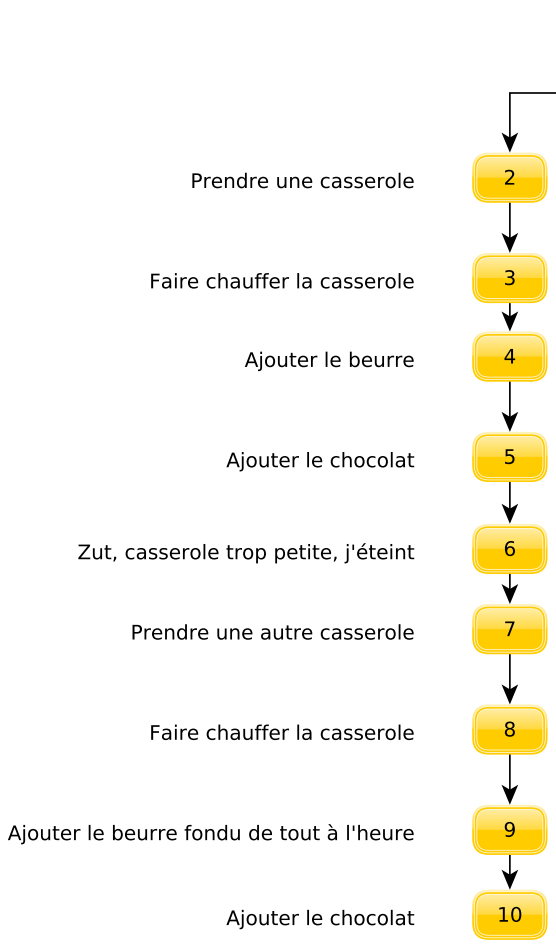


Figure – Un historique pourri

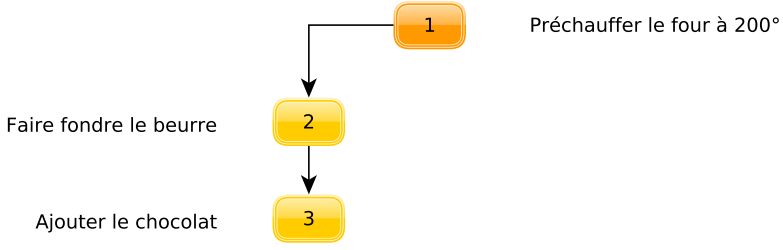


Figure – Et après un peu de nettoyage

Le rebase interactif : réécrire l'histoire

```
pick f7f3f6d changed my name a bit
pick 310154e updated README formatting and added blame
pick a5f4a0d added cat-file

# Rebase 710f0f8..a5f4a0d onto 710f0f8
```

Utilisation du rebase interactif

- **p, pick** : utiliser le commit (ne change rien)

Le rebase interactif : réécrire l'histoire

```
1 pick f7f3f6d changed my name a bit
  pick 310154e updated README formatting and added blame
3 pick a5f4a0d added cat-file

5 # Rebase 710f0f8..a5f4a0d onto 710f0f8
```

Utilisation du rebase interactif

- **p, pick** : utiliser le commit (ne change rien)
- **r, reword** : utilise le commit, mais permet de changer le message

Le rebase interactif : réécrire l'histoire

```
1 pick f7f3f6d changed my name a bit
  pick 310154e updated README formatting and added blame
3 pick a5f4a0d added cat-file

5 # Rebase 710f0f8..a5f4a0d onto 710f0f8
```

Utilisation du rebase interactif

- **p, pick** : utiliser le commit (ne change rien)
- **r, reword** : utilise le commit, mais permet de changer le message
- **e, edit** : utilise le commit et s'arrête pour pouvoir changer le contenu du commit

Le rebase interactif : réécrire l'histoire

```
1 pick f7f3f6d changed my name a bit
  pick 310154e updated README formatting and added blame
3 pick a5f4a0d added cat-file

5 # Rebase 710f0f8..a5f4a0d onto 710f0f8
```

Utilisation du rebase interactif

- **p, pick** : utiliser le commit (ne change rien)
- **r, reword** : utilise le commit, mais permet de changer le message
- **e, edit** : utilise le commit et s'arrête pour pouvoir changer le contenu du commit
- **s, squash** : fusionne avec le commit précédent

Le rebase interactif : réécrire l'histoire

```
1 pick f7f3f6d changed my name a bit
  pick 310154e updated README formatting and added blame
3 pick a5f4a0d added cat-file

5 # Rebase 710f0f8..a5f4a0d onto 710f0f8
```

Utilisation du rebase interactif

- **p, pick** : utiliser le commit (ne change rien)
- **r, reword** : utilise le commit, mais permet de changer le message
- **e, edit** : utilise le commit et s'arrête pour pouvoir changer le contenu du commit
- **s, squash** : fusionne avec le commit précédent
- **d, drop** : supprime le commit

BISECT : TROUVER D'OÙ VIENT LE BUG

- Savoir depuis quel commit l'application ne fonctionne plus

```
$ git bisect start <bad commit> <good commit>
# Quand la version courante est mauvaise :
$ git bisect bad
# Quand la version courante est bonne :
$ git bisect good
```

Utilisation de git bisect

BISECT : TROUVER D'OÙ VIENT LE BUG

- Savoir depuis quel commit l'application ne fonctionne plus

```
$ git bisect start <bad commit> <good commit>
# Quand la version courante est mauvaise :
$ git bisect bad
# Quand la version courante est bonne :
$ git bisect good
```

Utilisation de git bisect

- Et si on a des tests automatisés ?

```
$ git bisect start <bad commit> <good commit>
$ git bisect run auto-build-run-tests.sh
```

Utilisation de git bisect pour lancer les tests

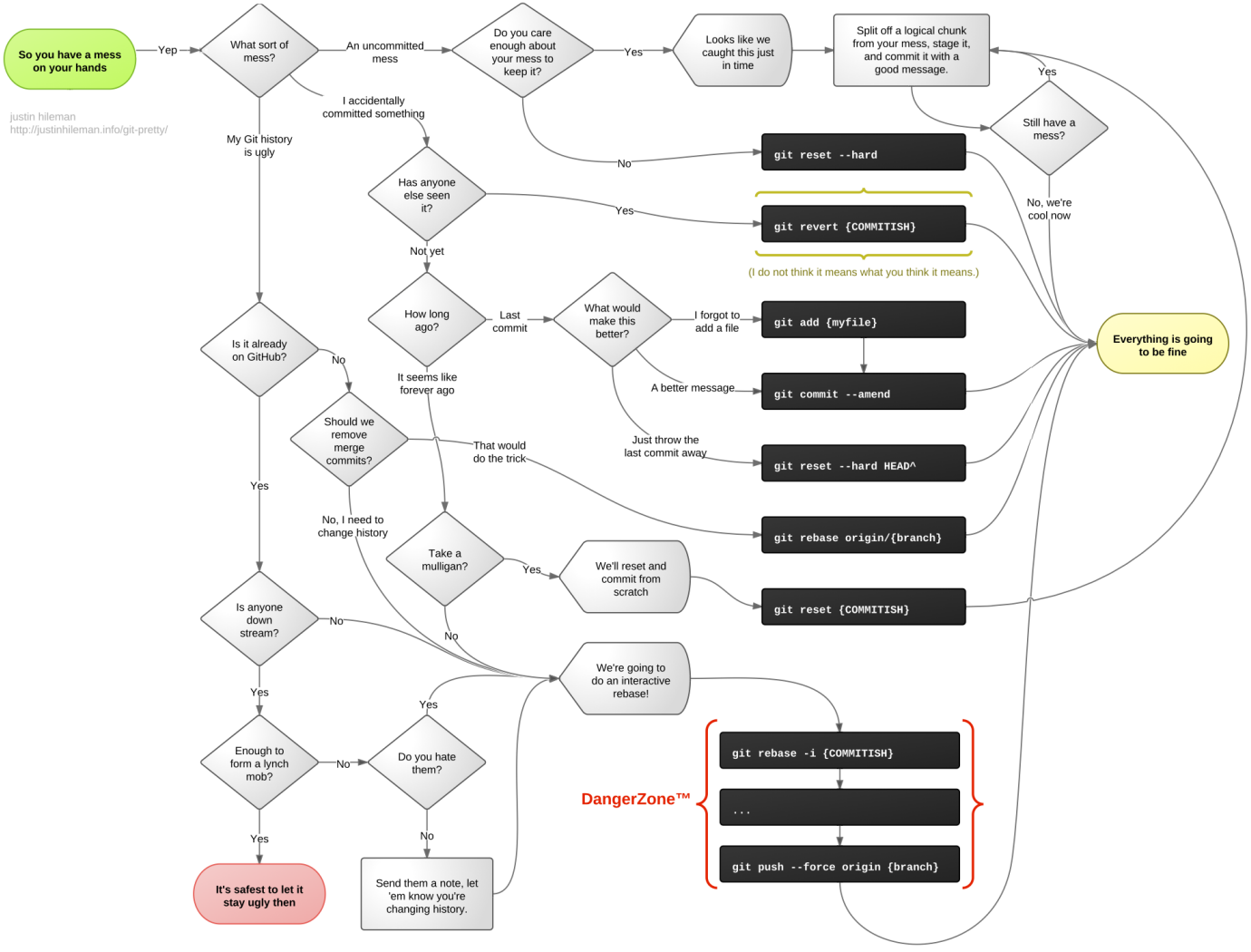
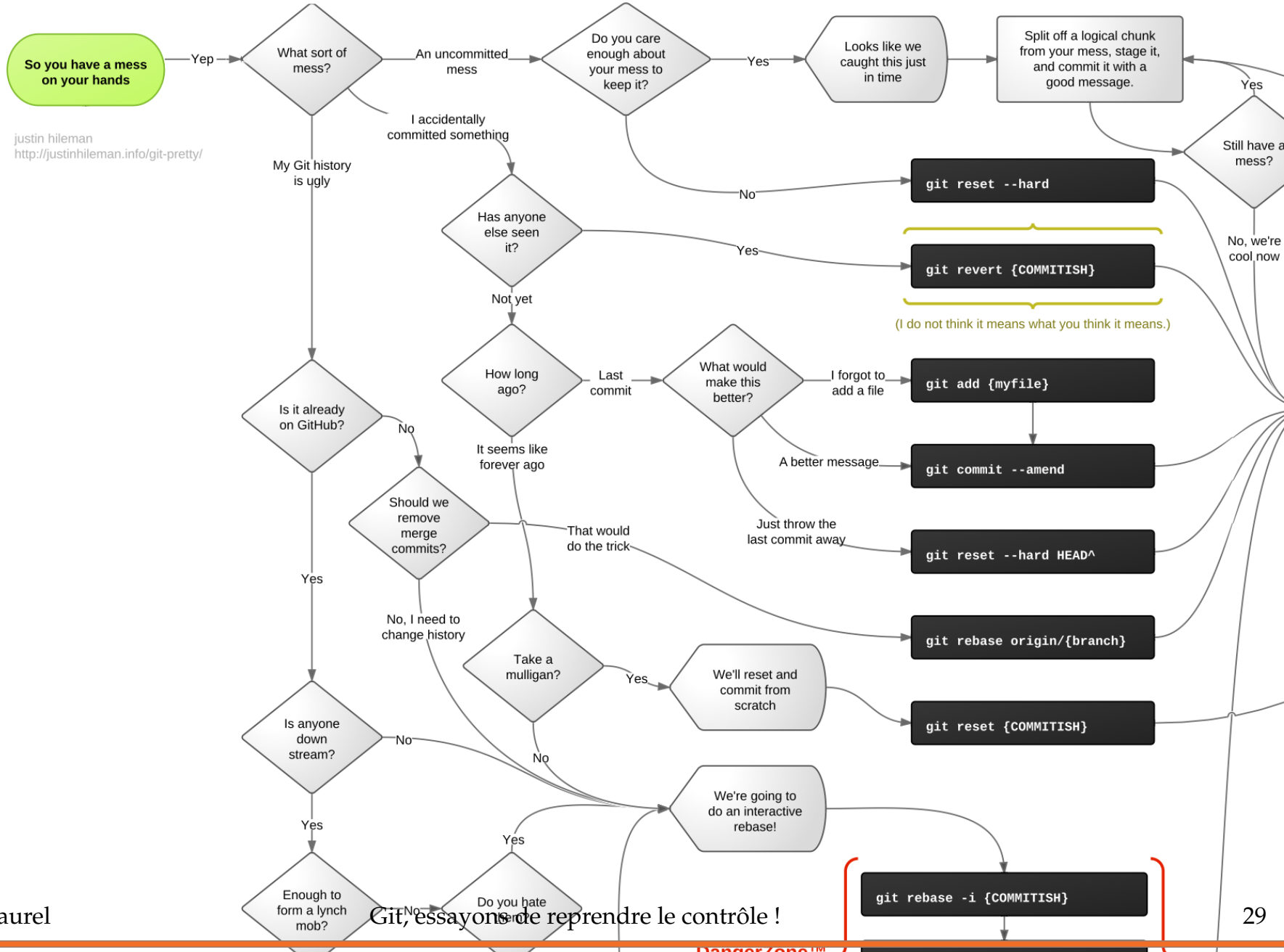


FIGURE – *So, you have a mess on your hands ?* [4]

4. `http://justinhileman.info/article/git-pretty/`


So you have a mess on your hands
Yep
What sort of mess?
An uncommitted mess
Do you care enough about your mess to keep it?
Yes
Looks like we caught this just in time
Split off a logical chunk from your mess, stage it, and commit it with a good message.
Yes
Still have a mess?
No, we're cool now
I accidentally committed something
My Git history is ugly
No
git reset --hard
Has anyone else seen it?
Yes
git revert {COMMITISH}
(I do not think it means what you think it means.)
Not yet
How long ago?
Last commit
What would make this better?
I forgot to add a file
git add {myfile}
A better message
git commit --amend
Just throw the last commit away
git reset --hard HEAD^
It seems like forever ago
That would do the trick
git rebase origin/{branch}
Is it already on GitHub?
No
Should we remove merge commits?
Yes
No, I need to change history
Take a mulligan?
Yes
We'll reset and commit from scratch
git reset {COMMITISH}
No
Is anyone down stream?
No
Yes
We're going to do an interactive rebase!
Yes
Enough to form a lynch mob?
No
Do you hate your guts?
git rebase -i {COMMITISH}
justin hileman
http://justinhileman.info/git-pretty/

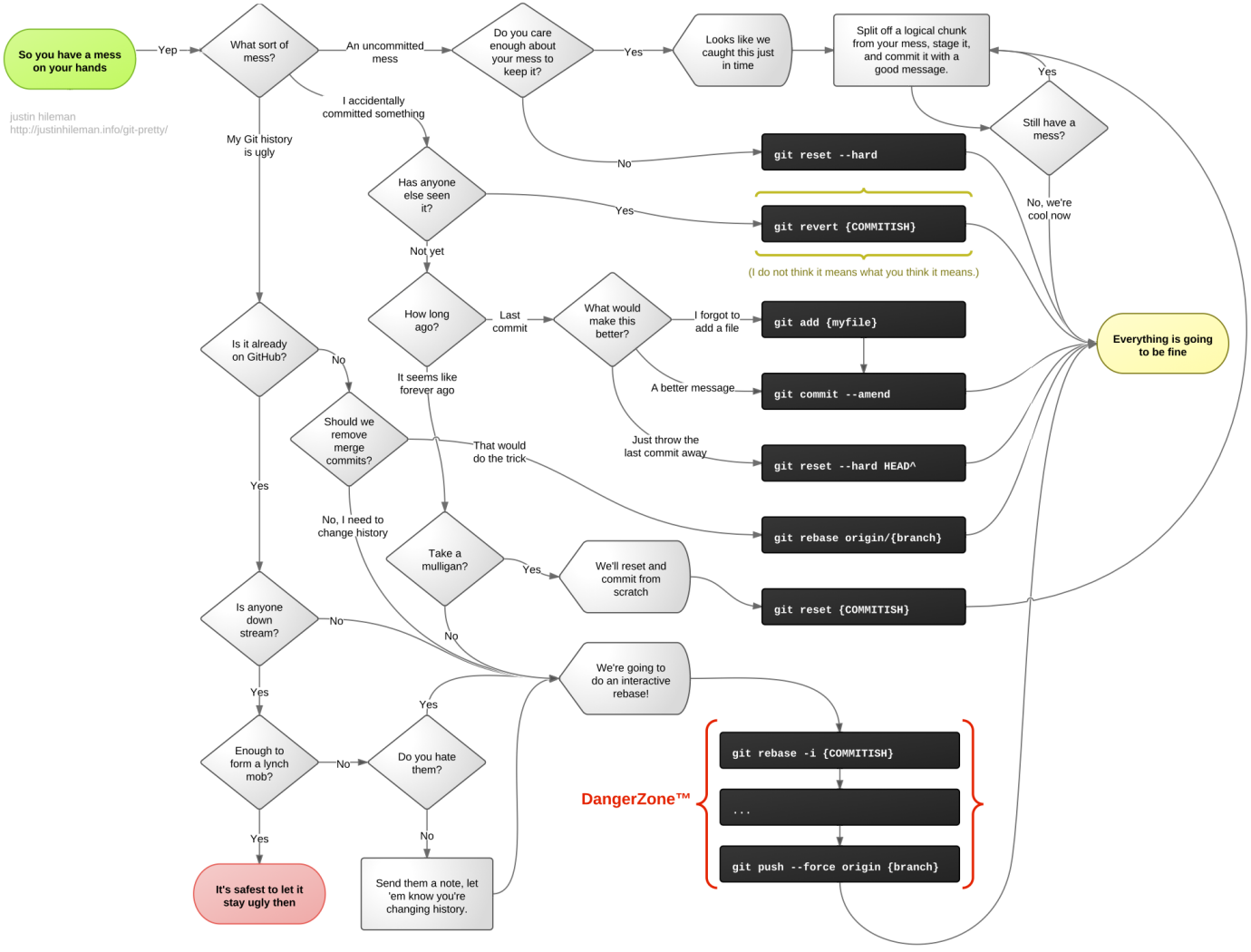
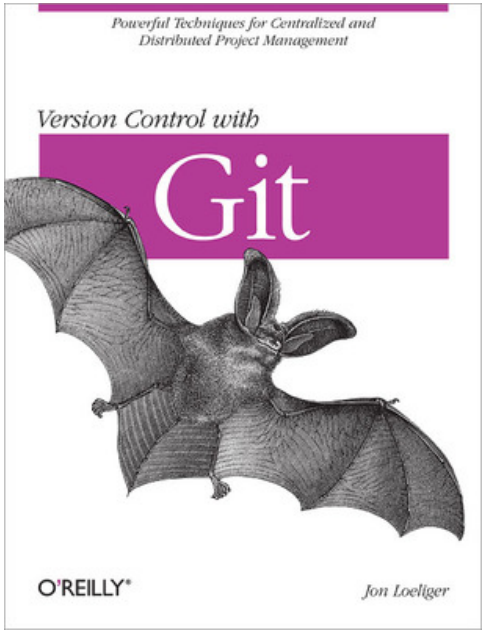
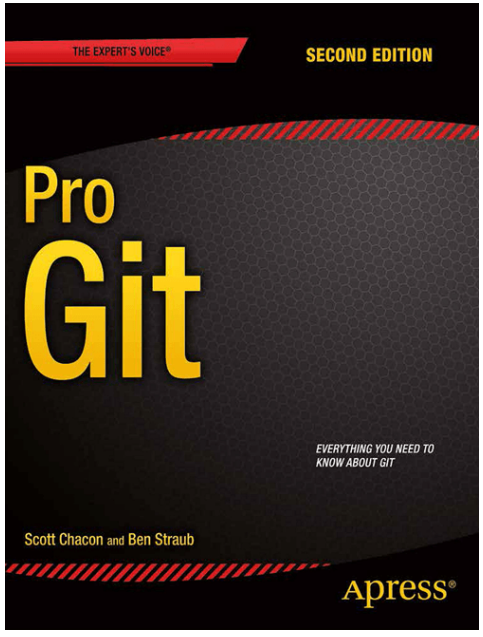


FIGURE – *So, you have a mess on your hands ?* [4]

4. `http://justinhileman.info/article/git-pretty/`

Références





- `git-scm.com`
  Site officiel
- `learngitbranching.js.org`
  Apprendre Git de manière ludique
- `github.com/aroquemaurel/Presentation-beamer-Git`
  Les sources LaTeX de cette présentation

Git, Essayons de reprendre le contrôle !

Antoine de ROQUEMAUREL
satenske

Développeur Java consultant chez Tech Advantage

Meetup Java / C# du 28 Mars 2019