Git - The stupid content tracker

...

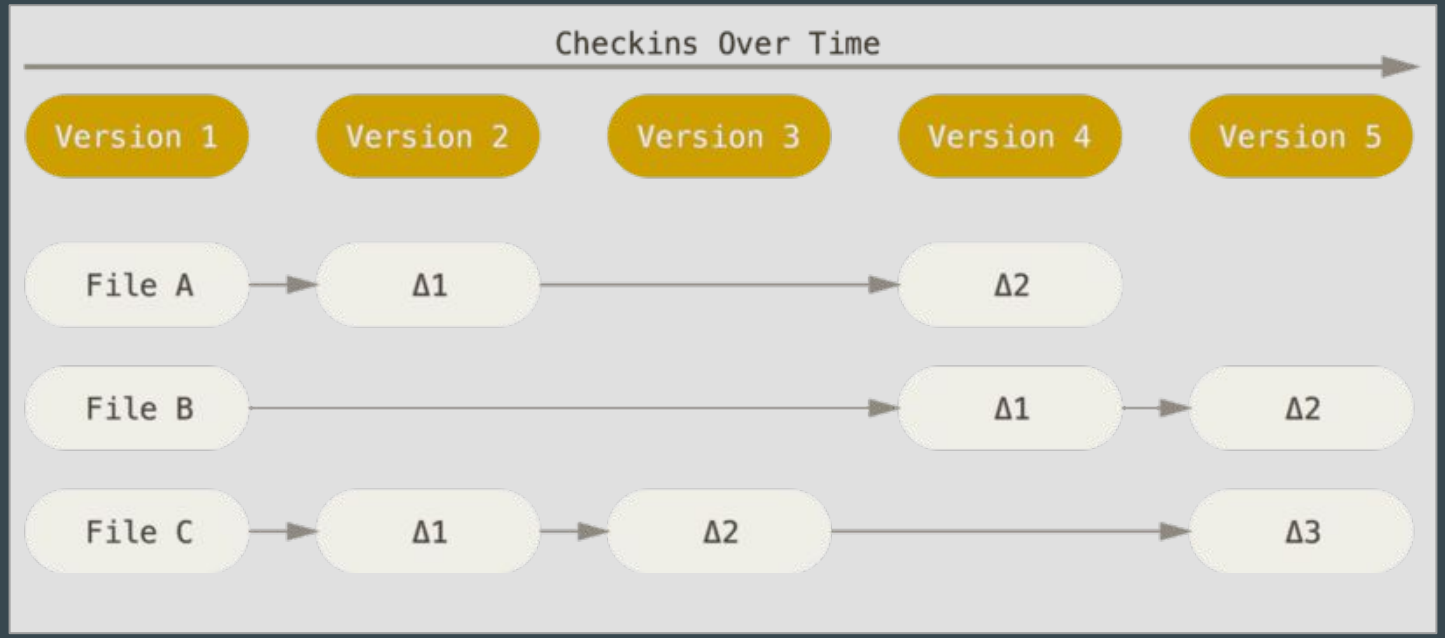
Démystifions Git !

Nicolas Nénon
21/02/2019

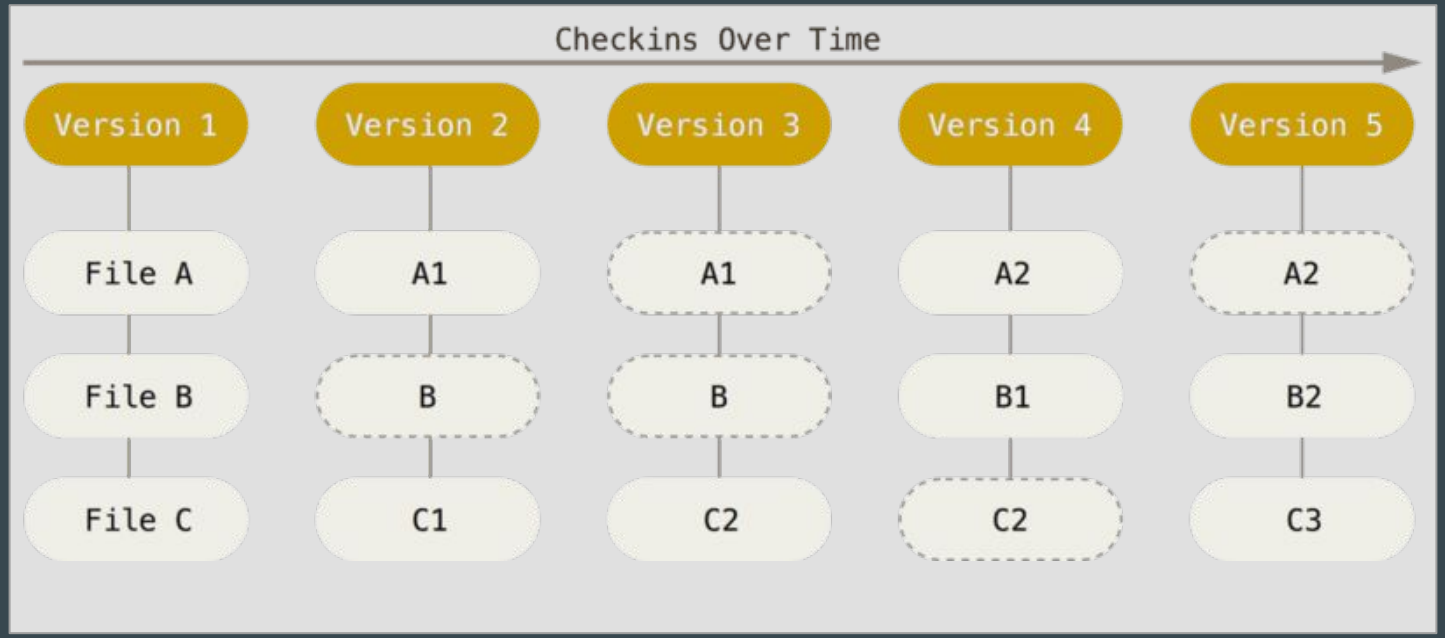
Sommaire

Préparation et commit

Des instantanés, pas des différences



Des instantanés, pas des différences



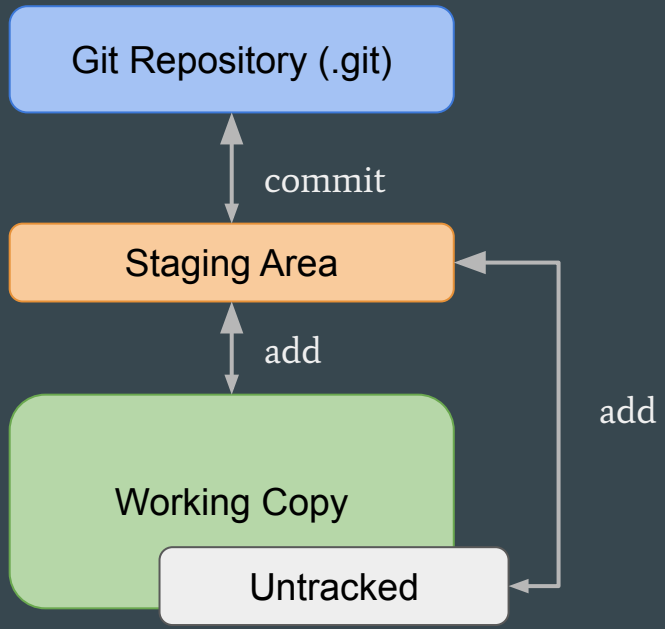
A savoir

- Opérations faites en local
- Gestion de l’intégrité des fichiers
- Git ne fait qu’ajouter des données

Commit

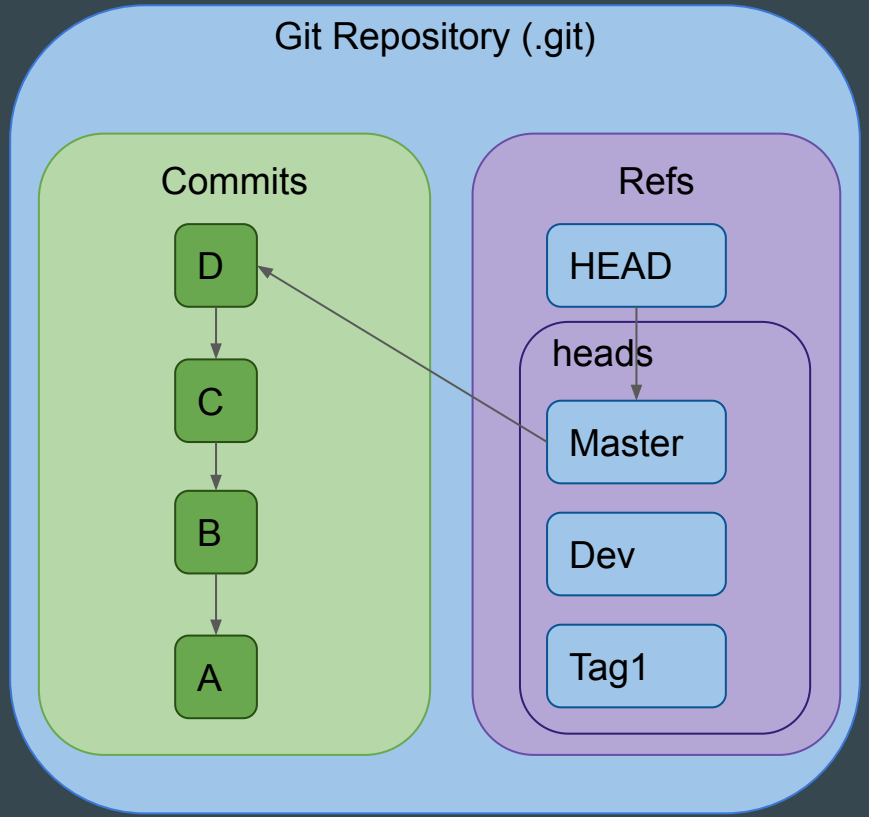
- Ensemble de snapshots
- Un auteur (adresse mail + nom prénom)
- Une date
- Un hash unique pour l’identifier

Zone de préparation (Stage)

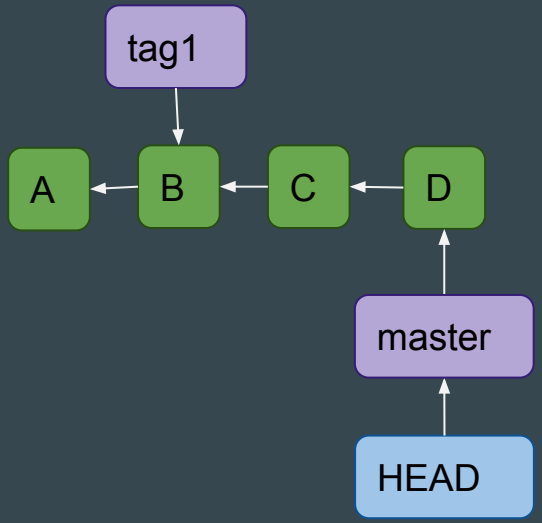


Refs et branches

Branches, HEAD & Tag



Branches, HEAD & Tag - Notation

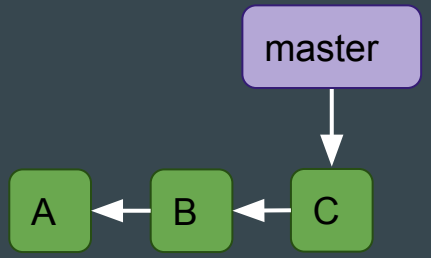


Branches, HEAD & Tag - Recap

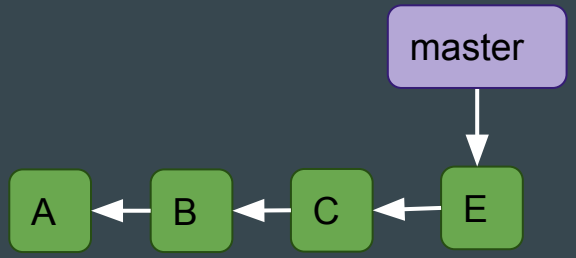
- Une branche c’est un pointeur sur un commit
- Créer un nouvelle branche c’est créer un nouveau pointeur
- Un tag sert à “marquer” un commit spécifique
- HEAD, pointe toujours vers une branche / un tag

Merge et Rebase

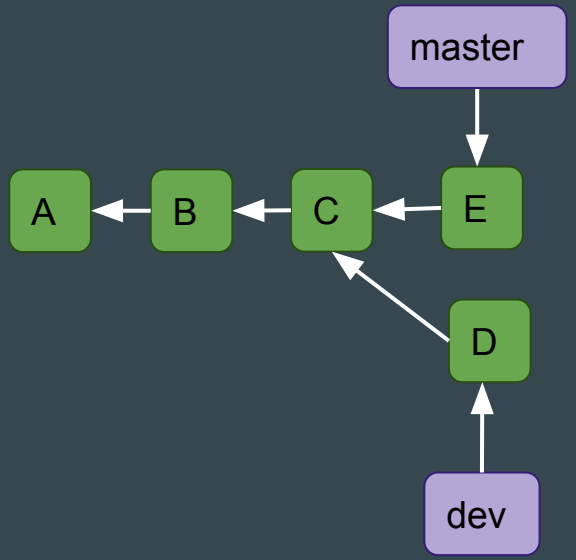
Merge



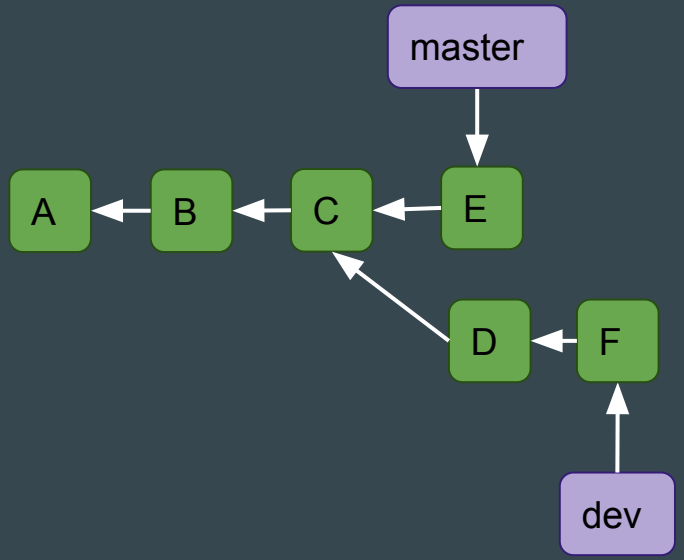
Merge



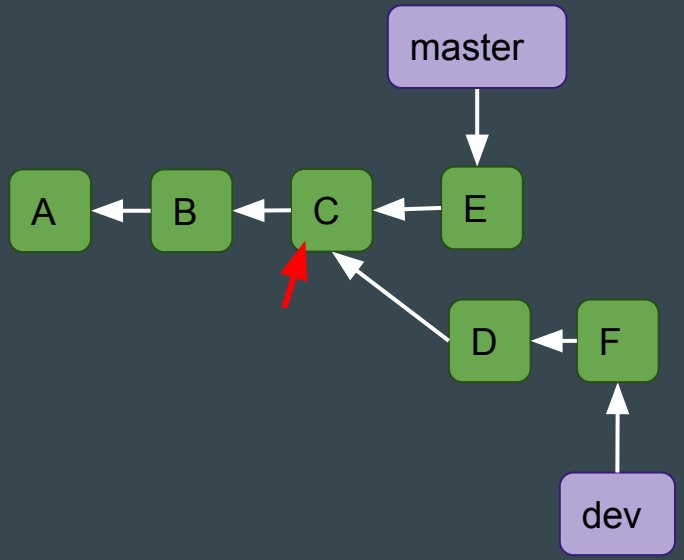
Merge



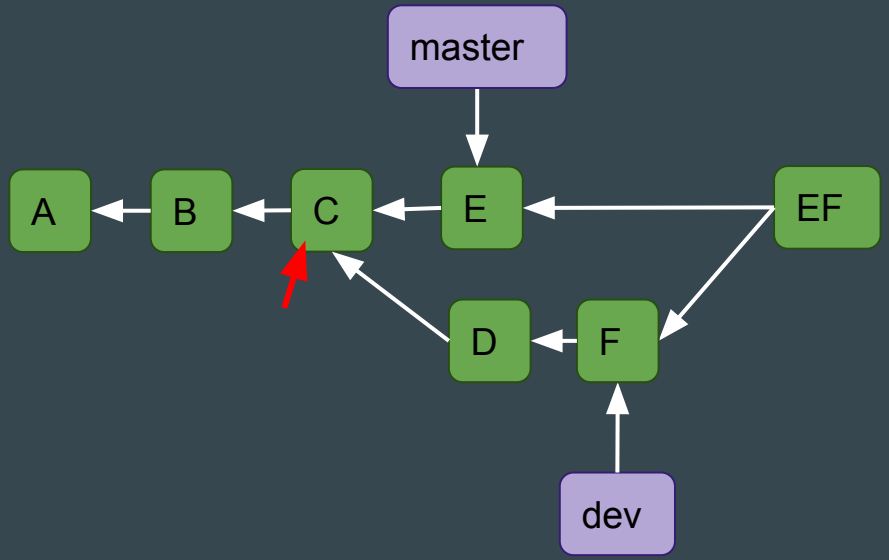
Merge



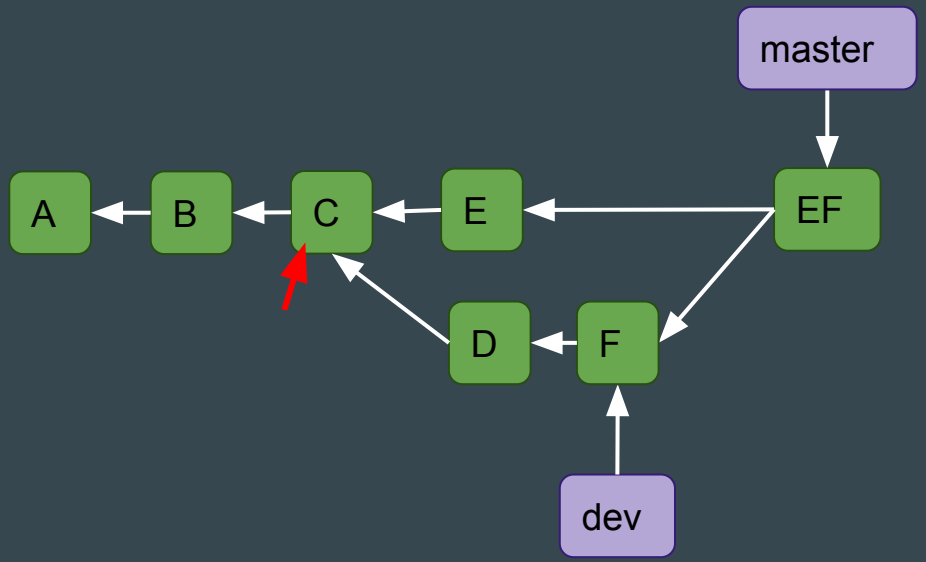
Merge



Merge



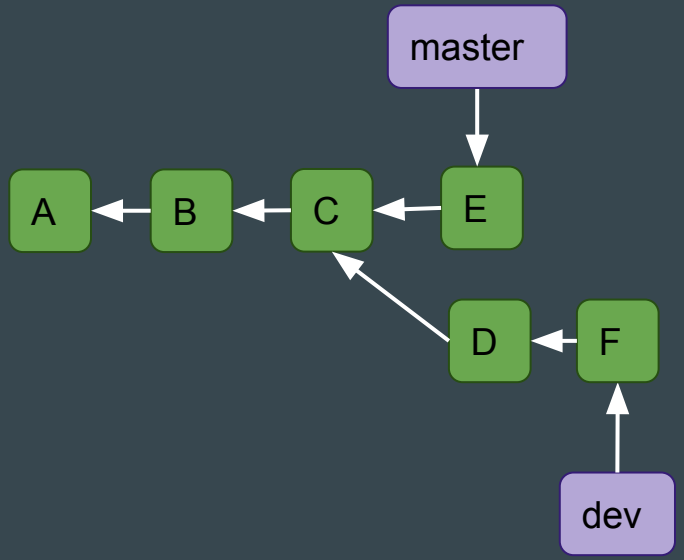
Merge



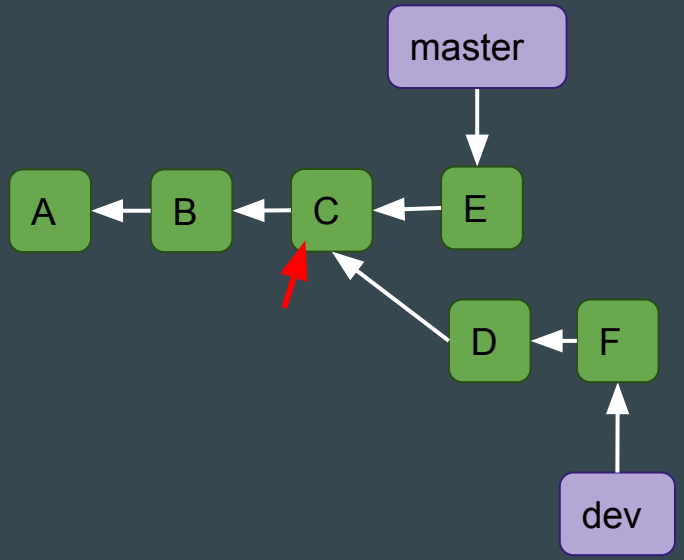
Merge - Recap

- Création d'un nouveau commit qui contient les snapshots des fichiers mergés
- Utilisation de la méthode du three-way merge (recherche de l'ancêtre commun)
- Après un merge, la branche dev peut être supprimée sans problème

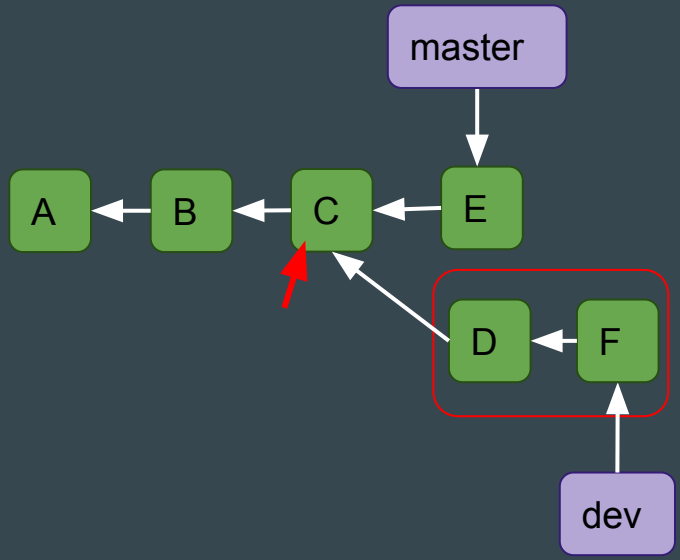
Rebase



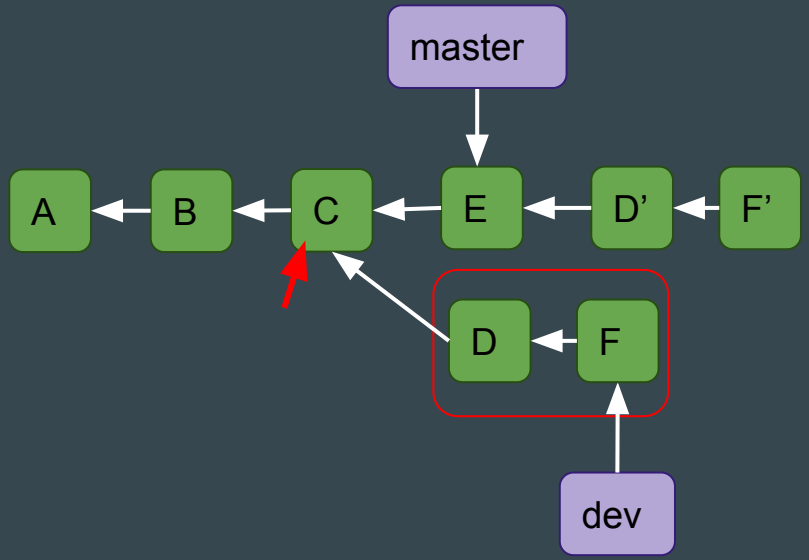
Rebase



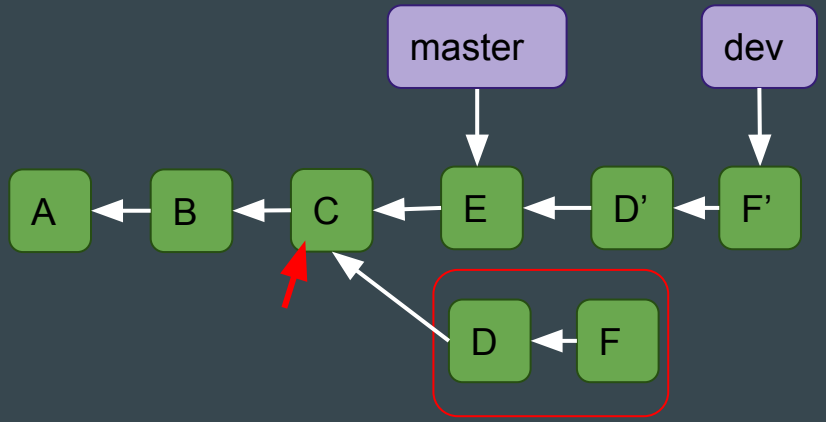
Rebase



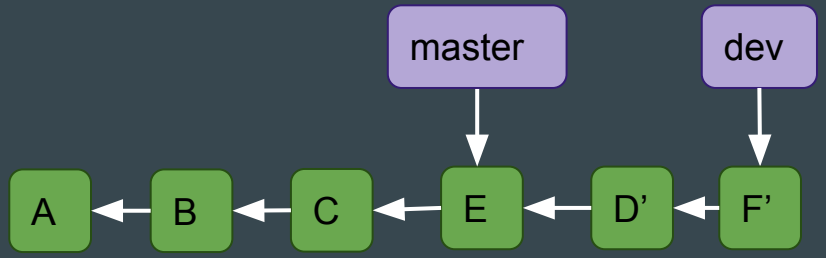
Rebase



Rebase



Rebase



Rebase - Recap

- Rejouer les commits sur la branche de destination
- Contrairement au merge, pas de commit
- Historique linéaire

Merge VS Rebase

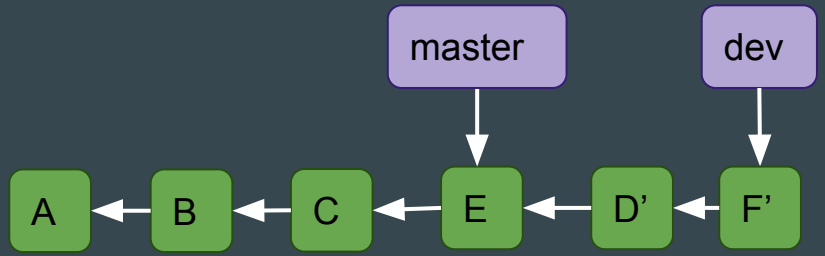
Quand faire un merge ?

- Merge des branches qui ont un sens
  - évolutions, merge de livraison
- Idéal dans le cas des branches distantes

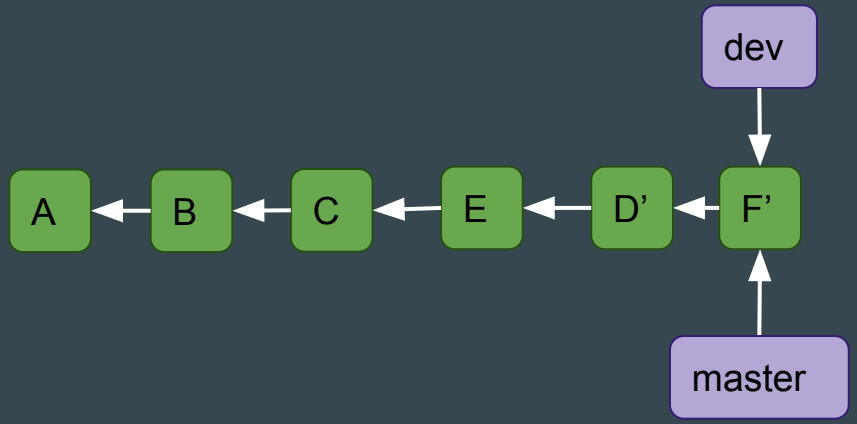
Quand faire un rebase ?

- La branche n'a pas de sens
  - expérimentation rapide, fix rapide
- Quand on a commencé un travail depuis longtemps

Cas du fast forward (FF)



Cas du fast forward (FF)



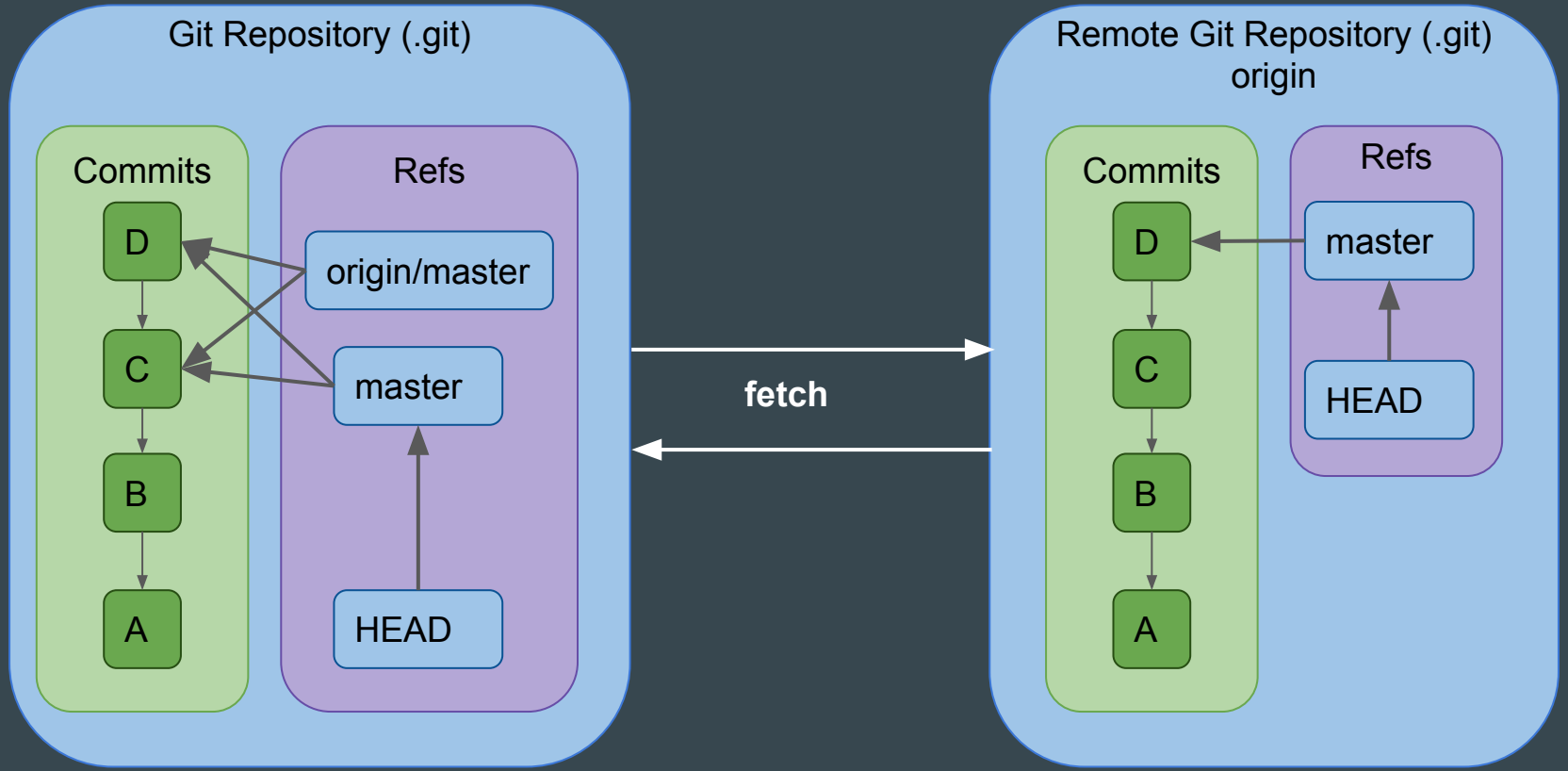
Travailler avec les branches distantes

Branches distantes

- Comme les branches locales, ce sont des refs
- Elles sont modifiées lors des actions réseaux
- origin est le remote par défaut, mais n'est pas spécial
- Une branche distante se note <remote>/<branch> ex : origin/master

Pull

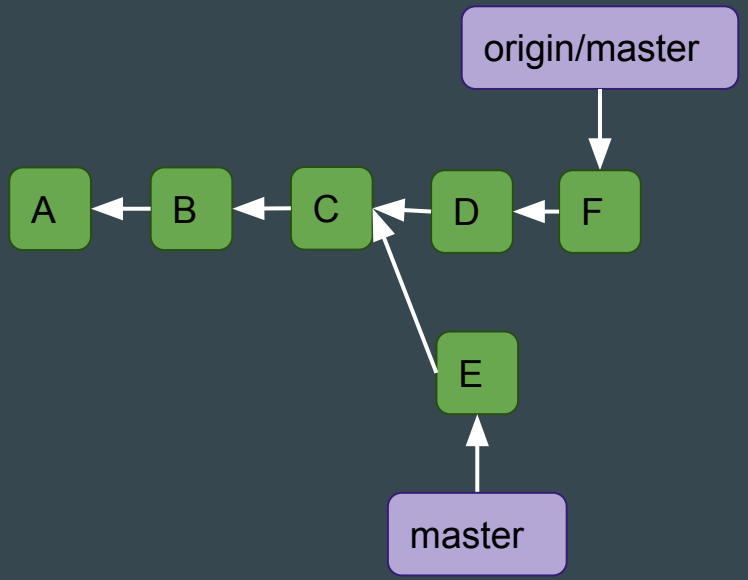
Pull


Git Repository (.git)
Commits
Refs
D
C
B
A
origin/master
master
HEAD
fetch
Remote Git Repository (.git)
origin
Commits
Refs
D
C
B
A
master
HEAD

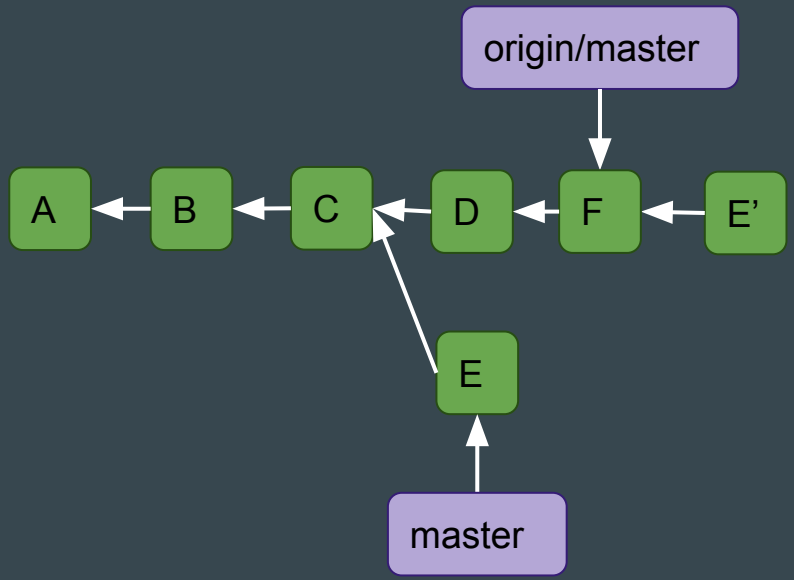
Pull - Recap

- Pull = Fetch + Merge
- Fetch : permet de récupérer les données du serveur
- Nouveaux commits, tags, branches et màj des branches origin/
- Possibilité de faire un pull rebase

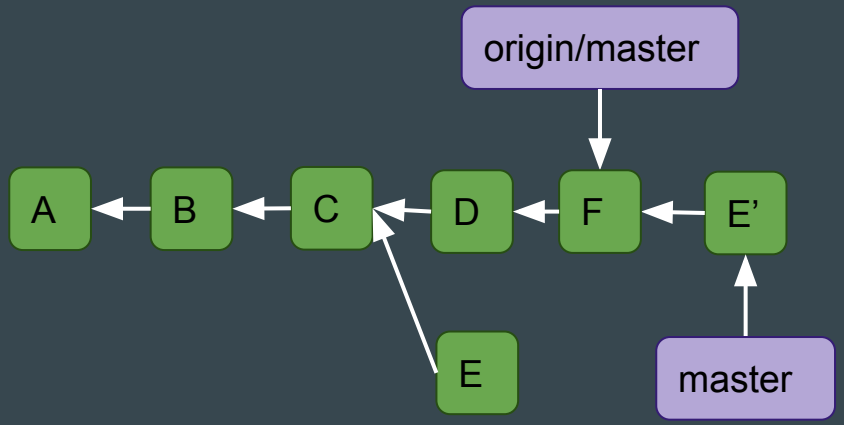
Pull - Rebase



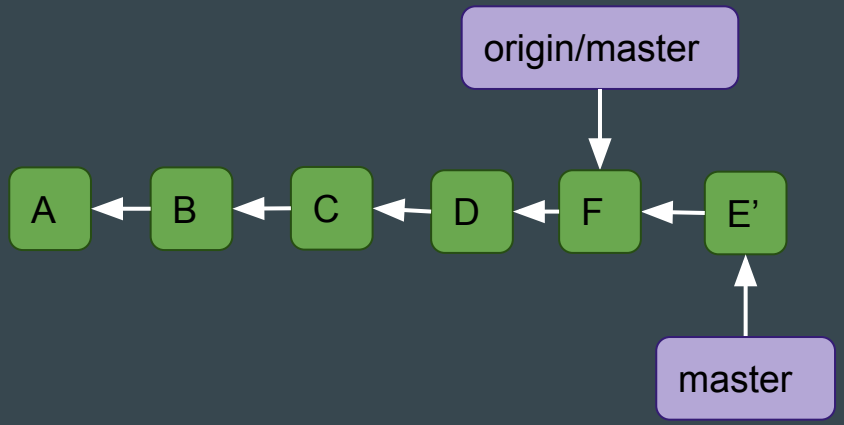
Pull - Rebase



Pull - Rebase

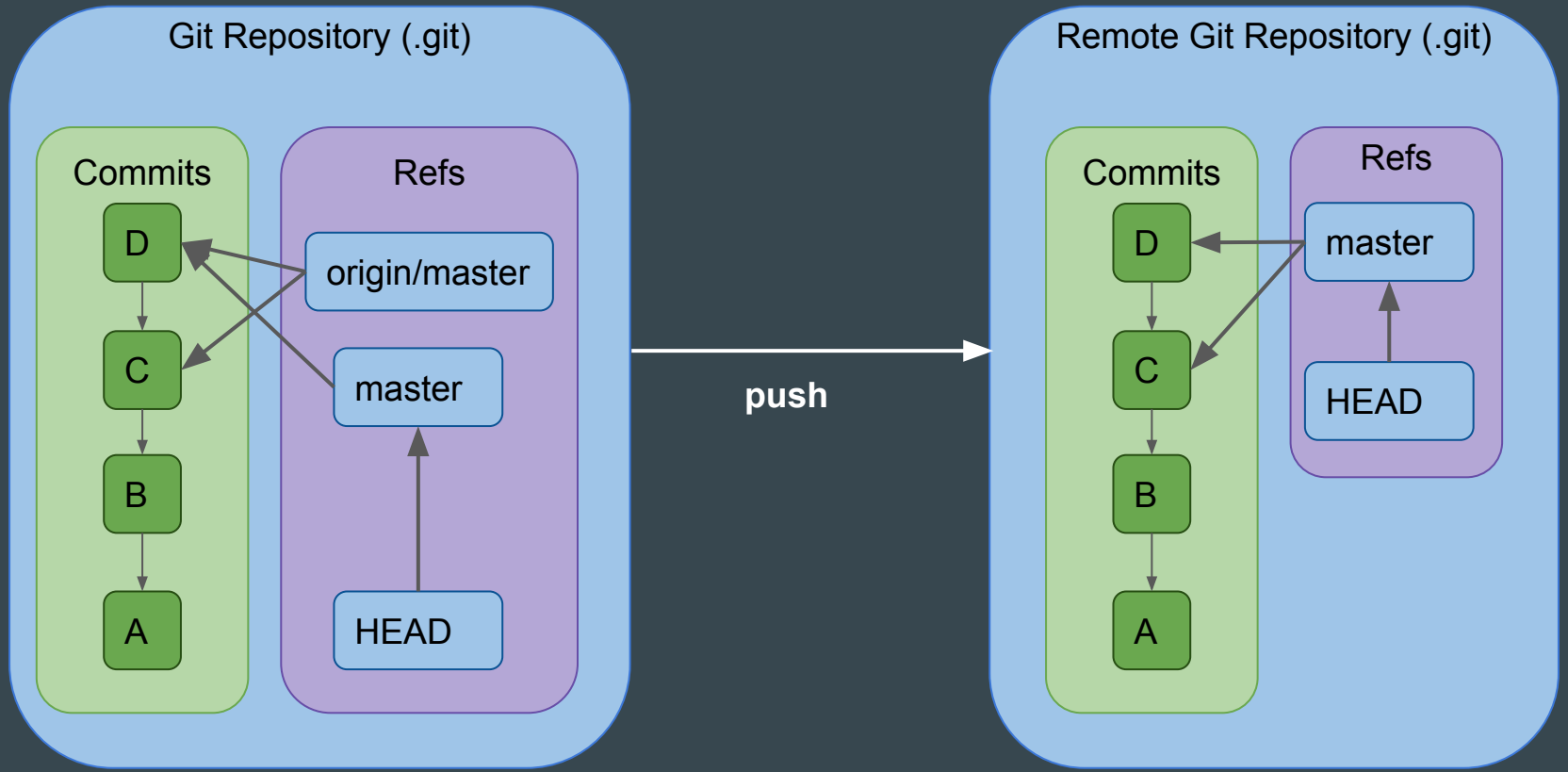


Pull - Rebase



Push

Push


Git Repository (.git)
Commits
D
C
B
A
Refs
origin/master
master
HEAD
push
Remote Git Repository (.git)
Commits
D
C
B
A
Refs
master
HEAD

Push - Recap

- Mise à jours de l'origin en local
- Envoie des données au serveur
- Aucune action possible sur la branche distante
- Toujours mettre à jour son travail avant d'envoyer

Stash - Remisage

- Permet de mettre de côté son travail courant
- Fonctionne comme une branche privée
- Fonctionnement de type pile
- Utilise des refs stash@{#}
  - stash@{0}

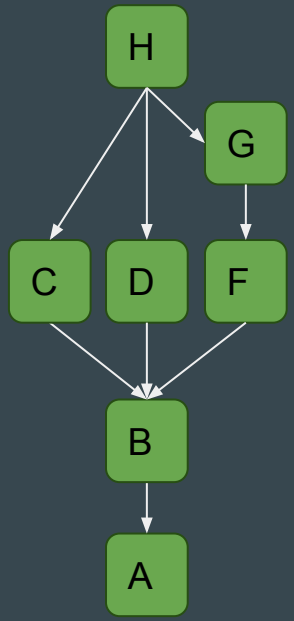
Notations magiques

^ et ~

- HEAD^n => n-ème parent direct
- HEAD~n => parent de la n-ème génération

^ et ~ - Exemple

- A = A^0
- B = A^ = A~
- C = A~2 = A^^
- D = B^2 = A~^2 = A^^2
- F = A^^3 = A~^3 = B^3
- G = A^^3^ = A~^3~
- H = G^ = A^^3^^ = A~^3~^

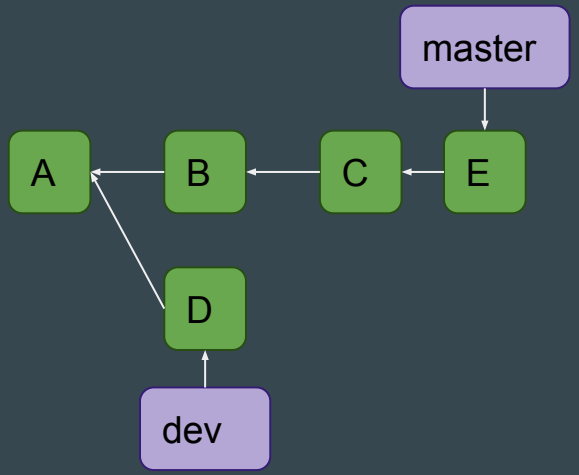


.. et ... - Range Notation et Symmetric Difference Notation

- master..dev signifie tous les commits de dev qui ne sont pas atteignables par master
  - git log ^master dev
  - git log dev --not master
- master...dev signifie tous les commits de dev et master sauf ceux en commun
  - git log r1 r2 --not $(git merge-base --all r1 r2)

.. et ... - Exemple

- git log master..dev : D
- git log dev..master : E C B
- git log master...dev : E C B D
- git log dev...master : D E C B



Workflow : retour d’expérience

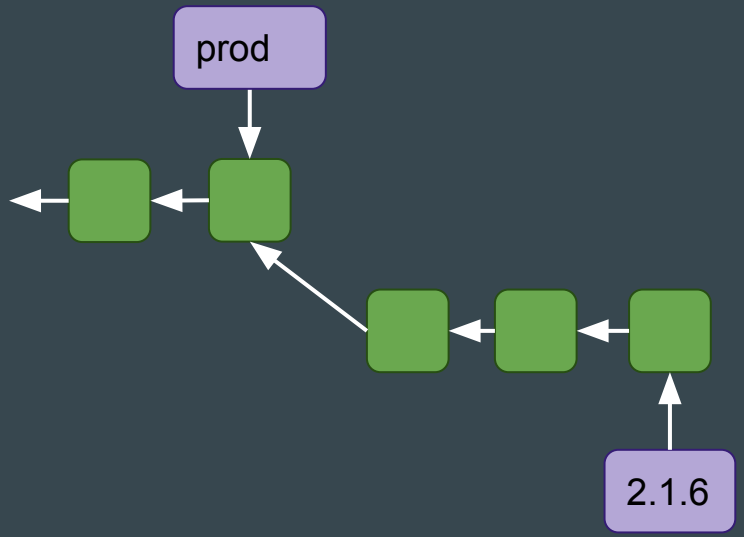
Contexte

- Équipe de 3 développeurs
- 3 environnements : dev, préprod, prod
- Pouvoir faire de l'avance de version (chevauchement)
- Gestion des retours de recette
- Maintenir un état cohérent entre les 3 environnements

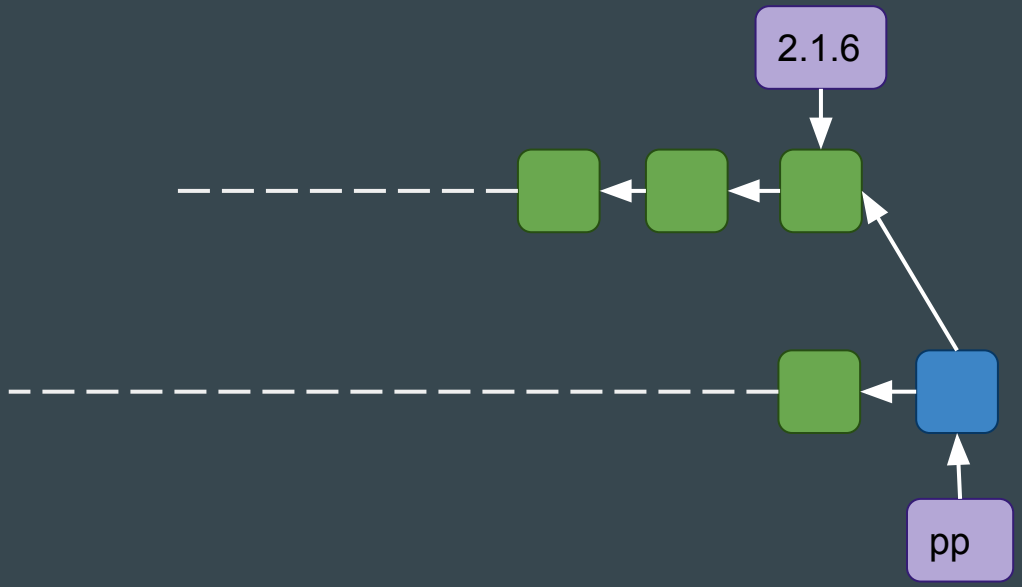
Début de réflexion

- Chacun sa copie de travail (son git)
- Une branche par feature (anomalie, évolution)
  - problème : explosion du nombre de branche
- Une branche par version, une branche par environnement
- La branche master utilisée uniquement pour l'historique

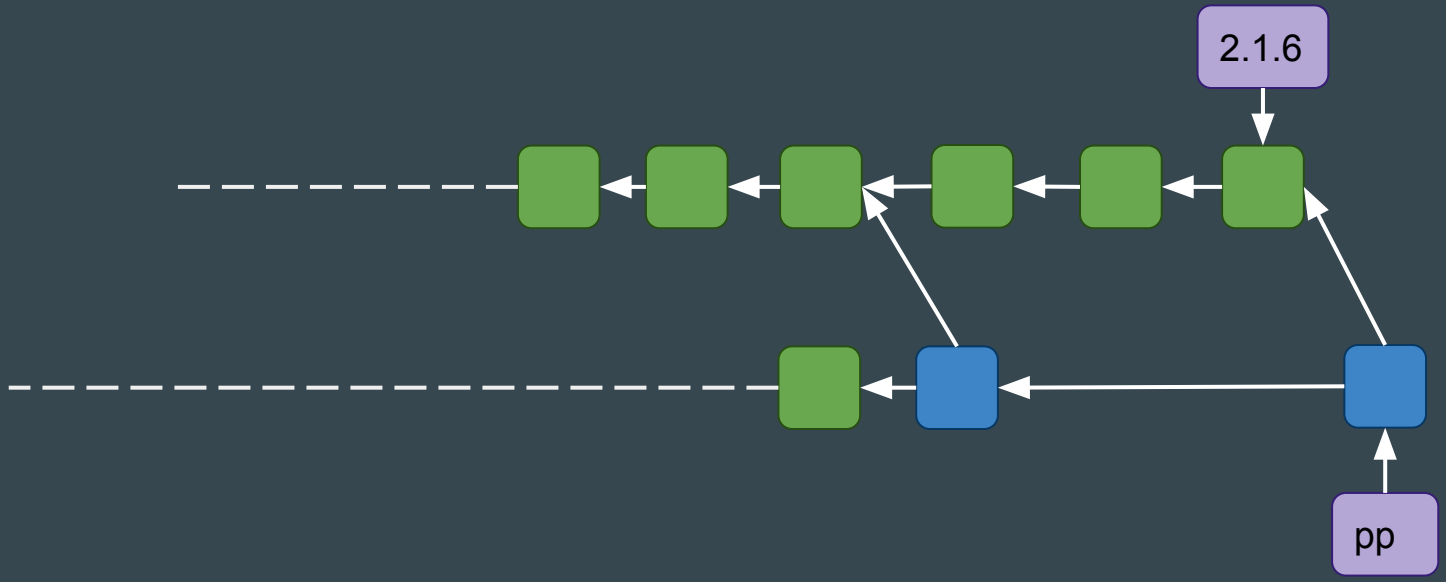
Cas d’une nouvelle version



Livraison en préproduction



Gestion des retours de recette



Livraison en production

- Simple merge de la pré-production

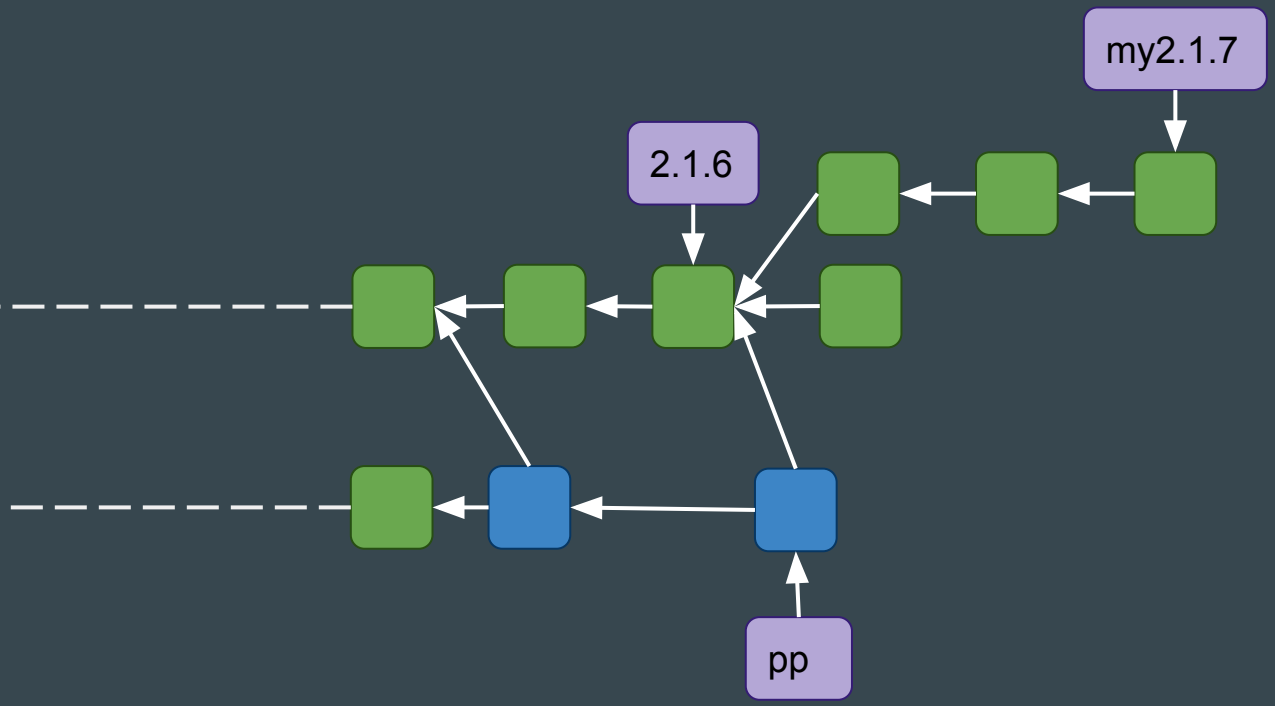
Avance de phase

- Comment faire pour avoir plusieurs versions ?
- Réussir à maintenir l'historique
- Avoir un historique le plus lisible possible

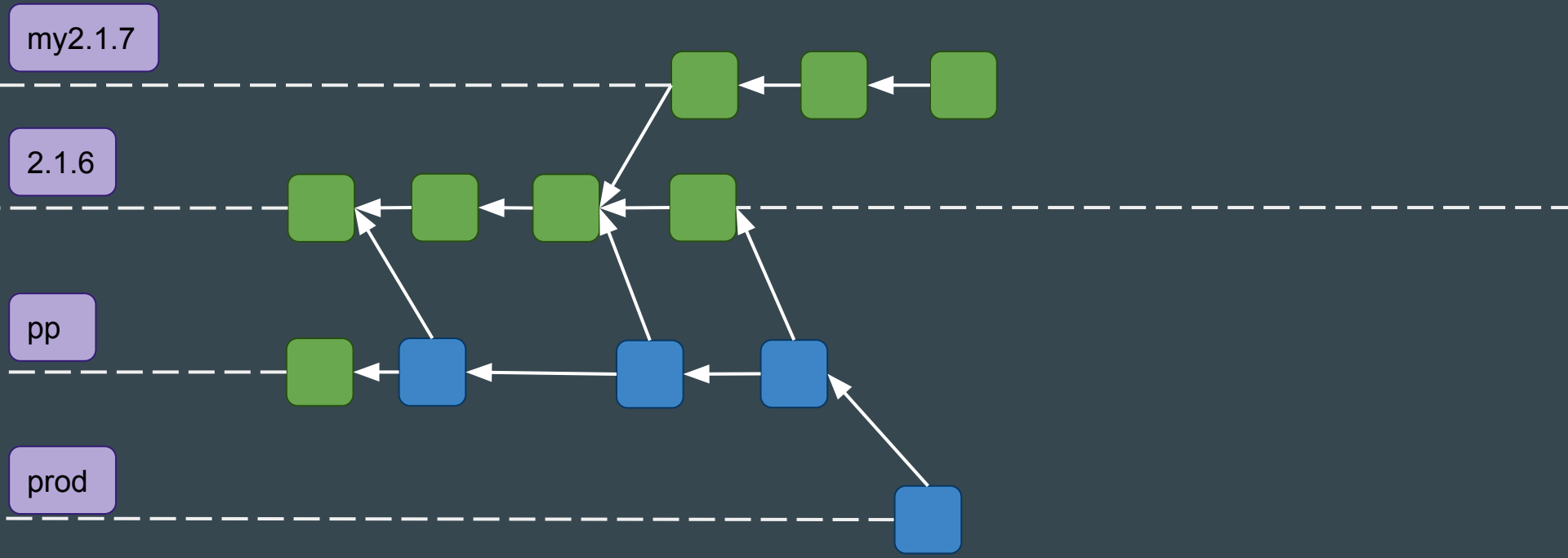
Avance de phase

- Solution : le rebase
- Chaque personne créer une branche locale
  - elle contient les commits pour la prochaine version
- Dès que la version précédente est en production
  - Création de la branche “officielle”
  - Rebase de ses développements + push

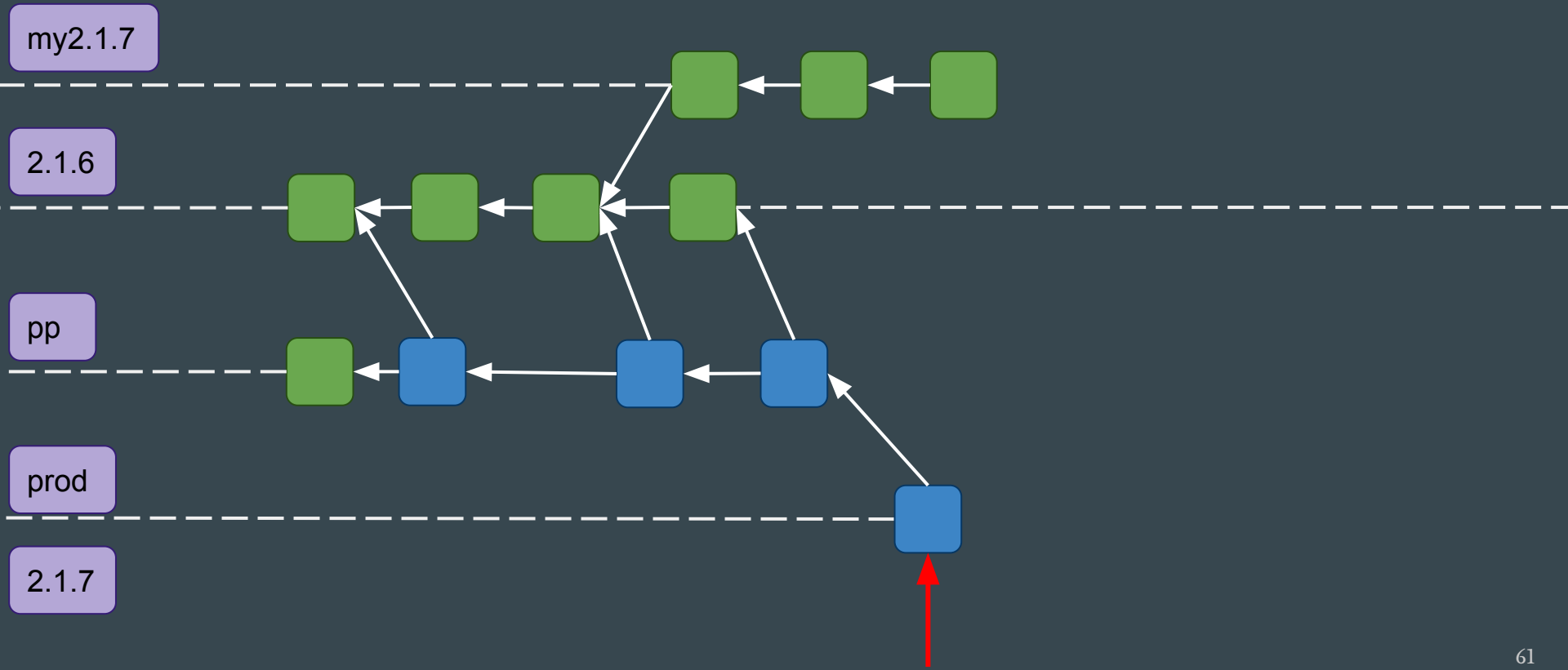
Avance de phase



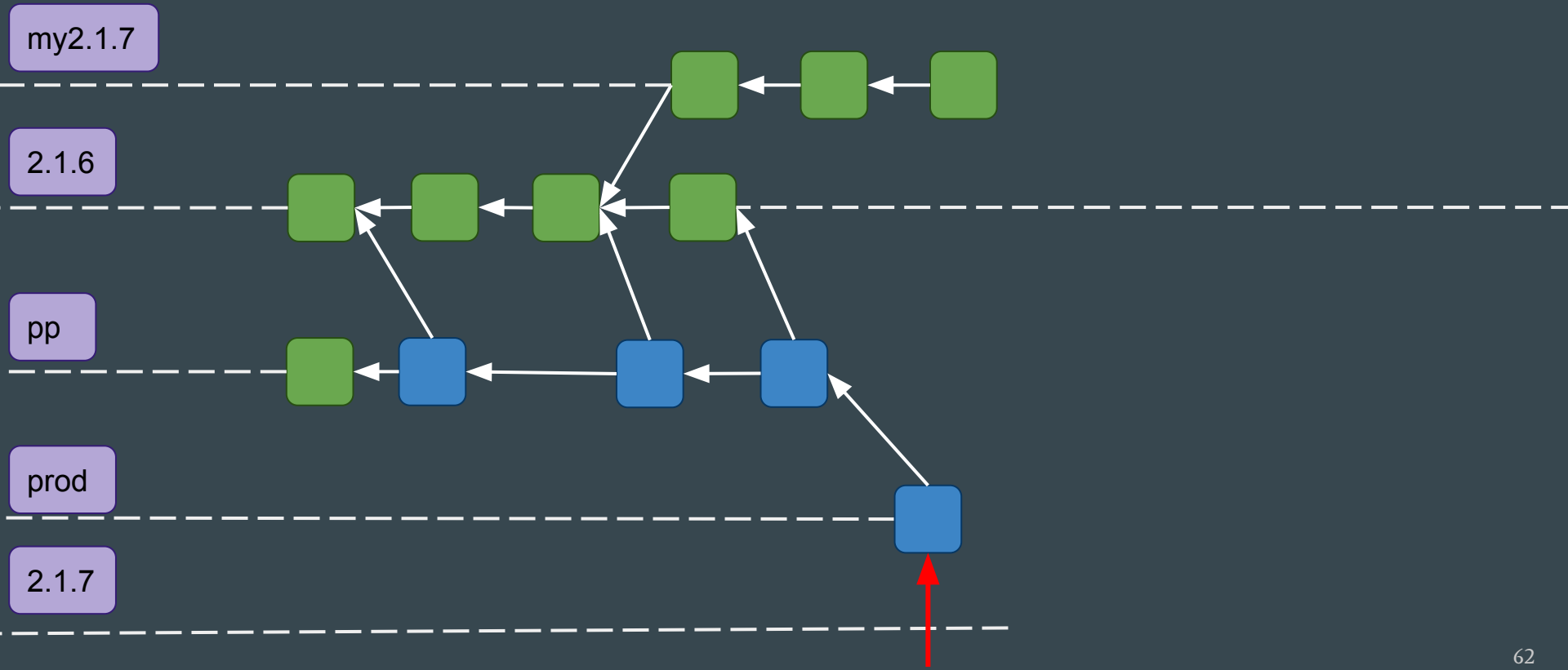
Avance de phase - Rebase



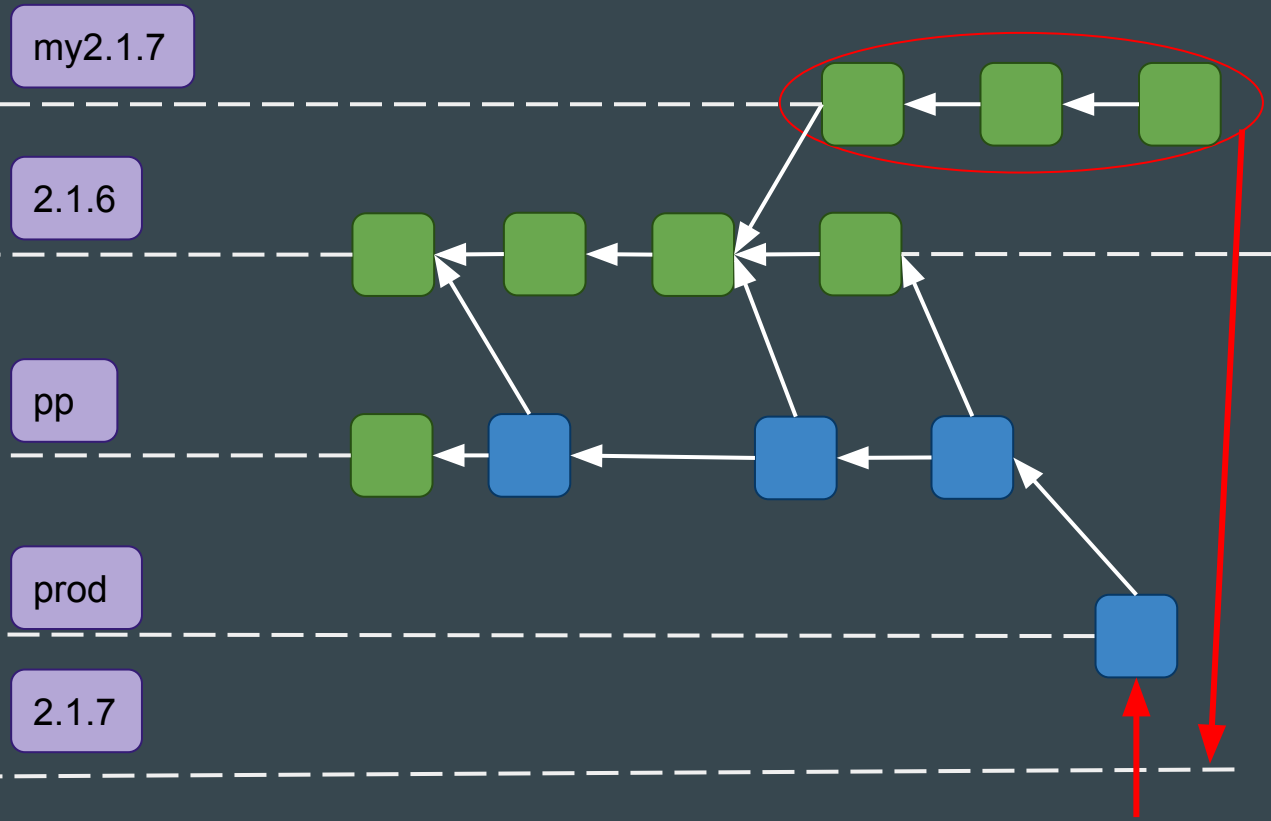
Avance de phase - Rebase



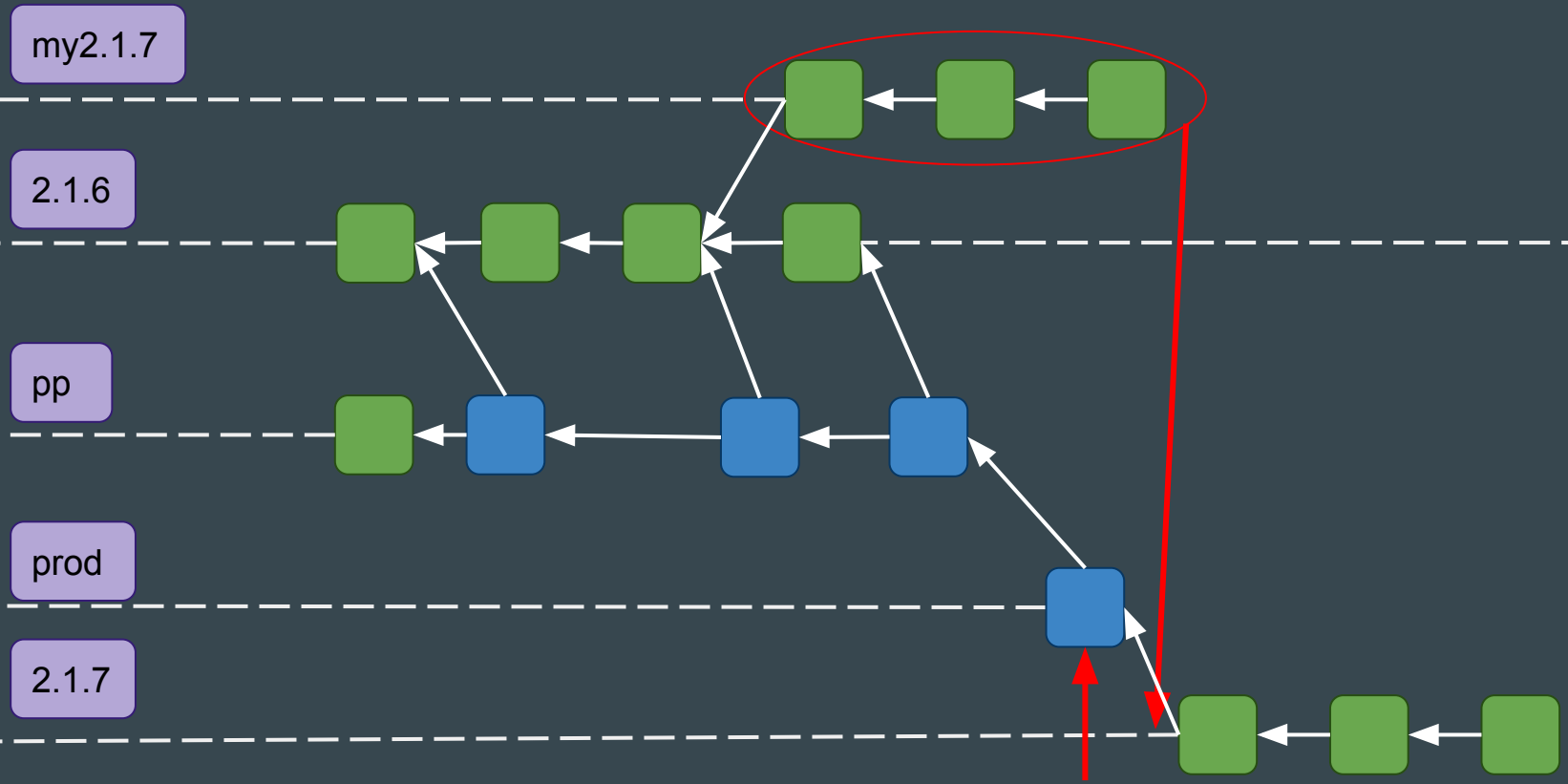
Avance de phase - Rebase



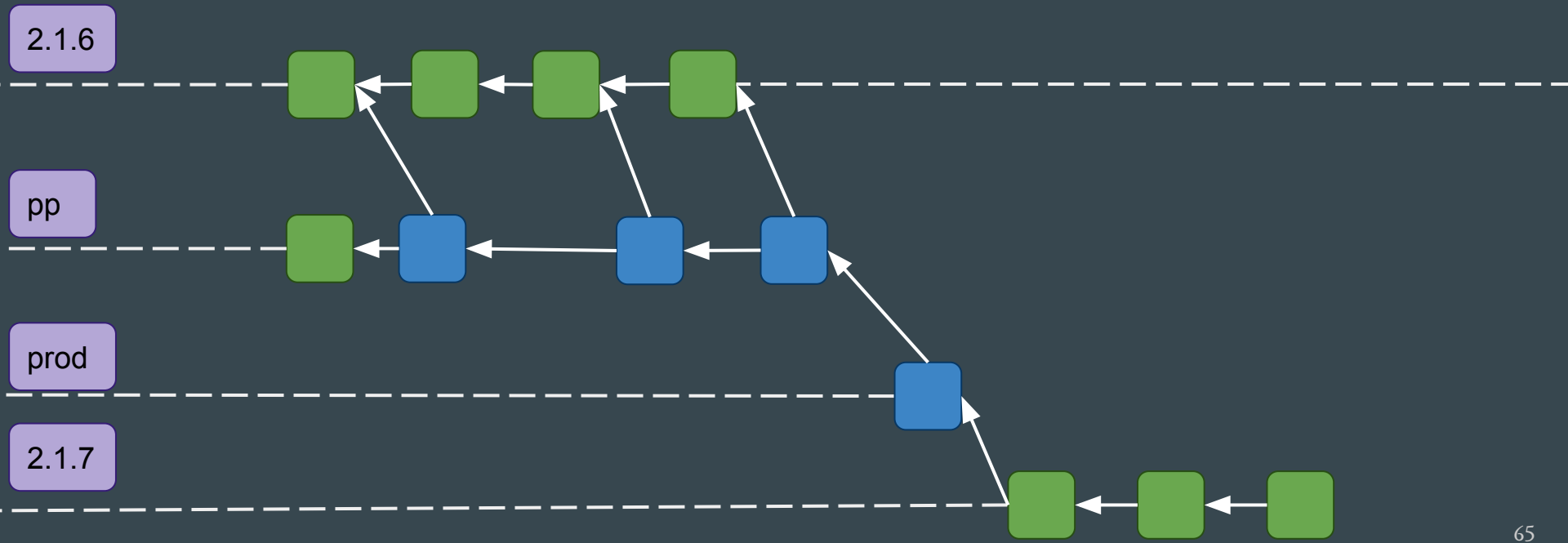
Avance de phase - Rebase



Avance de phase - Rebase



Avance de phase - Rebase



Tout ce qu’on a pas vu

Tout ce qu'on a pas vu

- Fork et pull request
- Hooks / Git as a lib
- Sous-module
- Plomberie
- Réécriture avancé de l'historique
  - Rebase interactif
  - cherry pick
  - ReReRe
  - --filter-branch
  - merge de deux historiques

Source

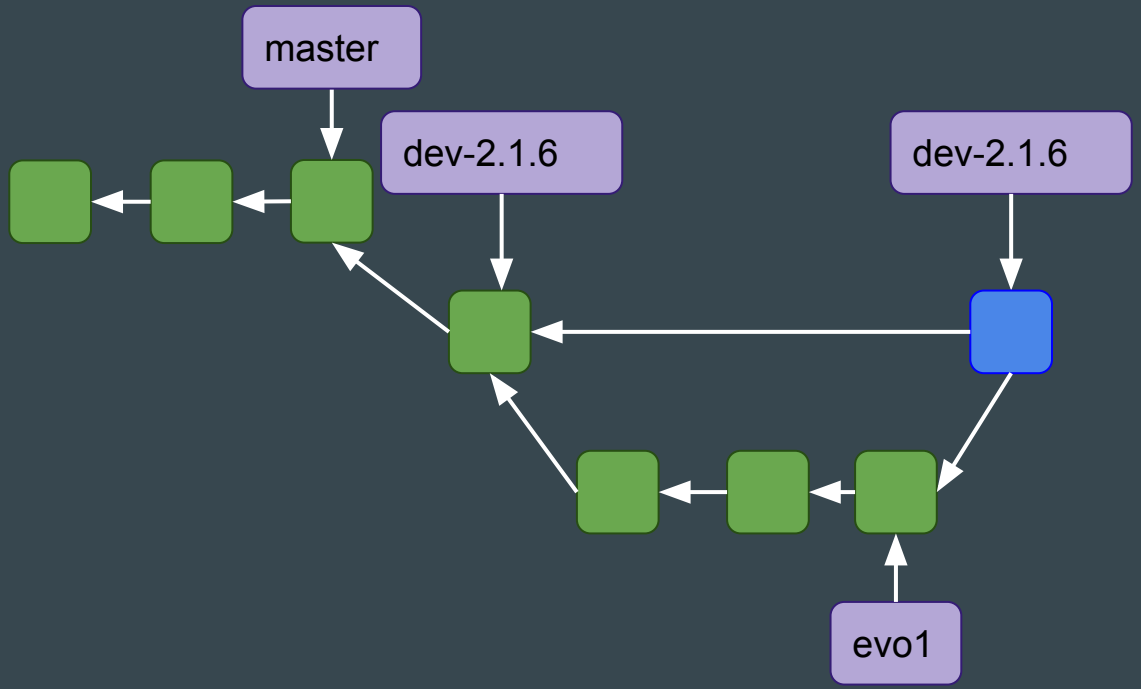
https://git-scm.com/book/en/v2

https://www.git-scm.com/docs/gitrevisions

https://www.git-scm.com/docs/gitglossary

http://loic-guibert.developpez.com/tutoriels/git/get-started/

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


master
dev-2.1.6
dev-2.1.6
evo1