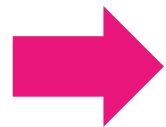


Outils de développement – suite

Stéphanie CHOLLET

En pratique...

- Le cycle de vie logiciel est un processus qui permet de produire un logiciel qui fonctionne.



Pas seulement coder !

Normalisation du code



Gestion de la documentation



Scripts pour la compilation,
le packaging et le déploiement
d'applications de manière reproductible



Gestion des versions
Version Control System



Plate-forme de tests



Suivi des problèmes
Bug tracker



Analyse statistique
du code



Et ...

Versions d'un logiciel

Qu'est-ce qu'une version d'un logiciel ?

Une **version d'un logiciel** correspond à un **état donné** de l'évolution d'un produit logiciel utilisant le versionnage (*versioning*). Elle est généralement associée à une numérotation qui permet de l'identifier, voire dans certains cas à un nom symbolique.

- ▶ Types d'évolution d'un logiciel :
 - ▶ Evolutions majeures :
 - ▶ Apport de nouvelles fonctionnalités
 - ▶ Restructuration de l'application
 - ▶ Evolutions mineures :
 - ▶ Correction de bugs
 - ▶ Ajouts de fonctionnalités secondaires

Numérotation des logiciels

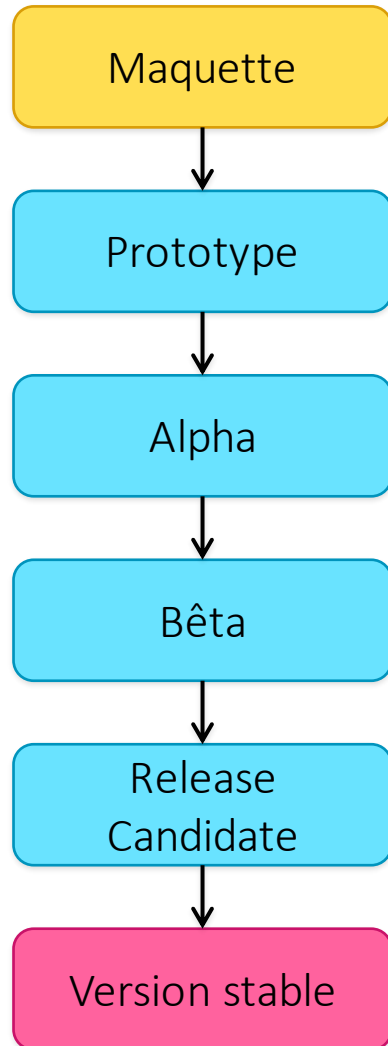
► Formalisation :

- Un ou plusieurs chiffres séparés par des points : 2.12.4
 - 2^{ème} version publiée, 12^{ème} ajout de fonctionnalités, 4^{ème} révision
 - Identification : majeur.mineur.micro
- Année de sortie du logiciel : Windows 2010
- Règle mathématique :
 - Version de TeX tend de manière asymptotique vers π
 - Version de Metafont tend de manière asymptotique vers e

► Règle de changement de la numérotation :

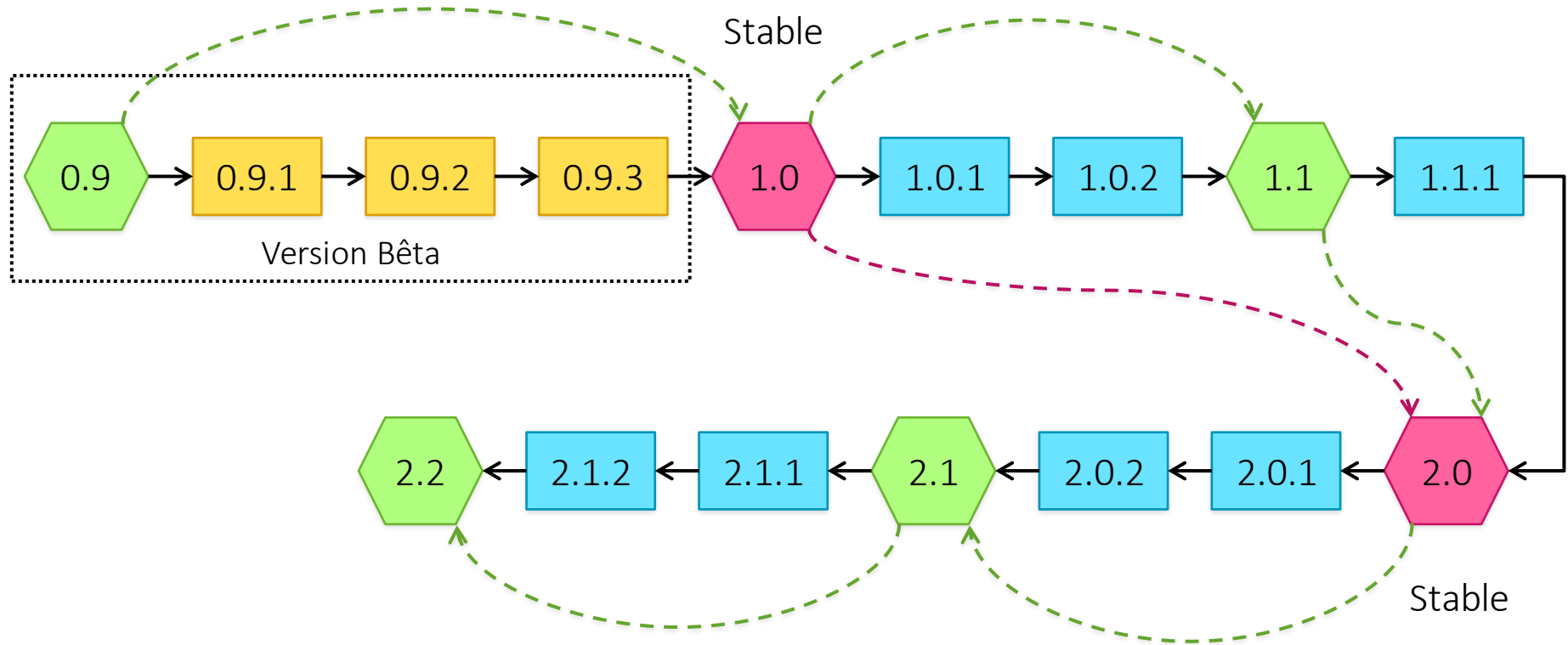
- Modification mineure : +1 sur la valeur micro
- Modification majeure : +1 sur la valeur mineure (ou majeure) + mise à 0 de la valeur micro (et mineure)

Phases d'un logiciel



- ▶ Maquette : aperçu visuel de l'objectif recherché sans réelle fonctionnalité
- ▶ Prototype : quelques fonctionnalités et sert de démonstrateur
- ▶ Alpha : version interne en phase de tests, manque certaines fonctionnalités et contient des bugs
- ▶ Bêta : version soumise à des utilisateurs
- ▶ RC : correspond à la version finale du logiciel, mise à disposition pour les qualifieurs
- ▶ Stable : version finale du logiciel avec les fonctionnalités prévues et « sans » bug

Arbre des versions d'un logiciel



Exemples de versions de logiciels – 1/2



- 2021-06, June 16, 2020
- 2021-03, March 17, 2020
- 2020-12, December 16, 2020
- 2020-09, September 16, 2020
- 2020-06, June 17, 2020
- 2020-03, March 18, 2020
- 2019-12, December 18, 2019
- 2019-09, September 19, 2019
- 2019-06, June 19, 2019
- 2019-03, March 20, 2019
- 2018-12, December 19, 2018
- 2018-09, September 19, 2018
- Photon, June 27, 2018
- Oxygen, June 28, 2017
- Neon, June 22, 2016
- Mars, June 24, 2015
- Luna, June 25, 2014
- Kepler, June 26, 2013
- Juno, June 27, 2012
- Indigo, June 22, 2011
- Helios, June 23, 2010
- Galileo, June 24, 2009
- Ganymede, June 25, 2008
- Europa, June 27, 2007
- Callisto, June 26, 2006

Eclipse 2021-06

- R Packages
- RC1 Packages
- M3 Packages
- M2 Packages
- M1 Packages

Eclipse Photon

- R Packages
- RC3 Packages
- RC2 Packages
- RC1 Packages
- M7 Packages
- M6 Packages
- M5 Packages
- M4 Packages
- M3 Packages
- M2 Packages

Exemples de versions de logiciels – 2/2



Cupcake
Android 1.5



Donut
Android 1.6



Eclair
Android 2.0/2.1



Froyo
Android 2.2/2.2.3



Gingerbread
Android 2.3/2.3.7



Honeycomb
Android 3.0/3.2



Ice Cream Sandwich
4.0



Android 4.1 Jelly Bean



Android 4.4 KitKat



android
Lollipop 5.0.1

android 6.0
Marshmallow



Android 7.0
NOUGAT



Android O



android

Gestionnaires de versions

Problématique

► Trace de l'évolution :

Calculatrice.java

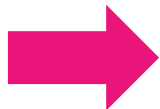
```
public Integer ajouter(Integer operande1, Integer operande2) {  
    return operande1 + operande2;  
}
```



Passage à une deuxième version

Calculatrice.java

```
public Float ajouter(Float operande1, Float operande2) {  
    return operande1 + operande2;  
}
```



Quelles modifications ? Pourquoi ? Par qui ?

Problématique

► Retour en arrière :

► Première version :

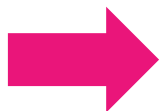
Calculatrice.java

```
public Float ajouter(Float operande1, Float operande2) {  
    return operande1 + operande2;  
}
```

► Deuxième version :

Calculatrice.java

```
public Integer ajouter(Integer operande1, Integer operande2) {  
    return operande1 + operande2;  
}
```



Impossible si on n'a pas gardé une copie du premier fichier

Problématique

- Equipe de développement :



```
public Integer ajouter(Integer  
operande1, Integer operande2) {  
    return operande1 + operande2;  
}
```

Calculatrice.java



```
public Integer soustraire(Integer  
operande1, Integer operande2) {  
    return operande1 - operande2;  
}
```

Calculatrice.java



Fusion

```
public Integer ajouter(Integer  
operande1, Integer operande2) {  
    return operande1 + operande2;  
}  
public Integer soustraire(Integer  
operande1, Integer operande2) {  
    return operande1 - operande2;  
}
```

Calculatrice.java

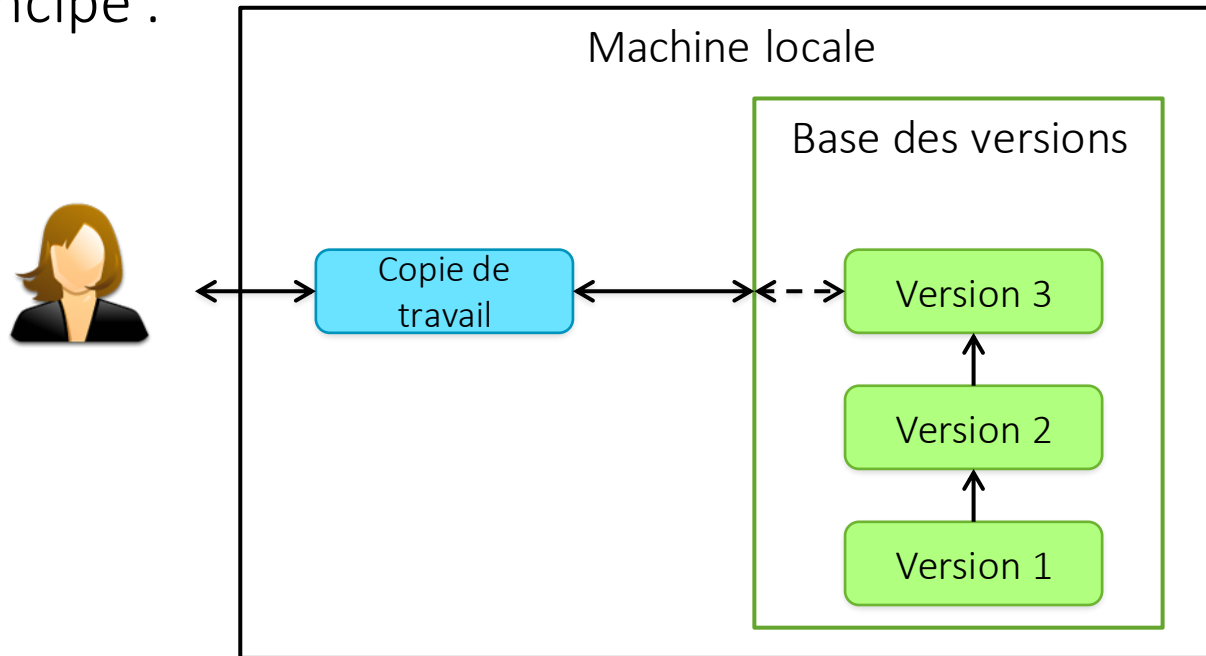
Gestion des versions

La **gestion de versions** (en anglais *version control* ou *revision control*) consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers (généralement en texte).

- ▶ Une **version** (ou révision) est une étape d'avancement
- ▶ Le passage d'une version à une autre se fait par des **modifications** : ajouts, suppressions, mises à jour de données
- ▶ Trois types de gestions de versions :
 - ▶ Locale, centralisée ou décentralisée

1- Gestion de versions locale

► Principe :



► Exemple : *Source Code Control System (SCCS)*



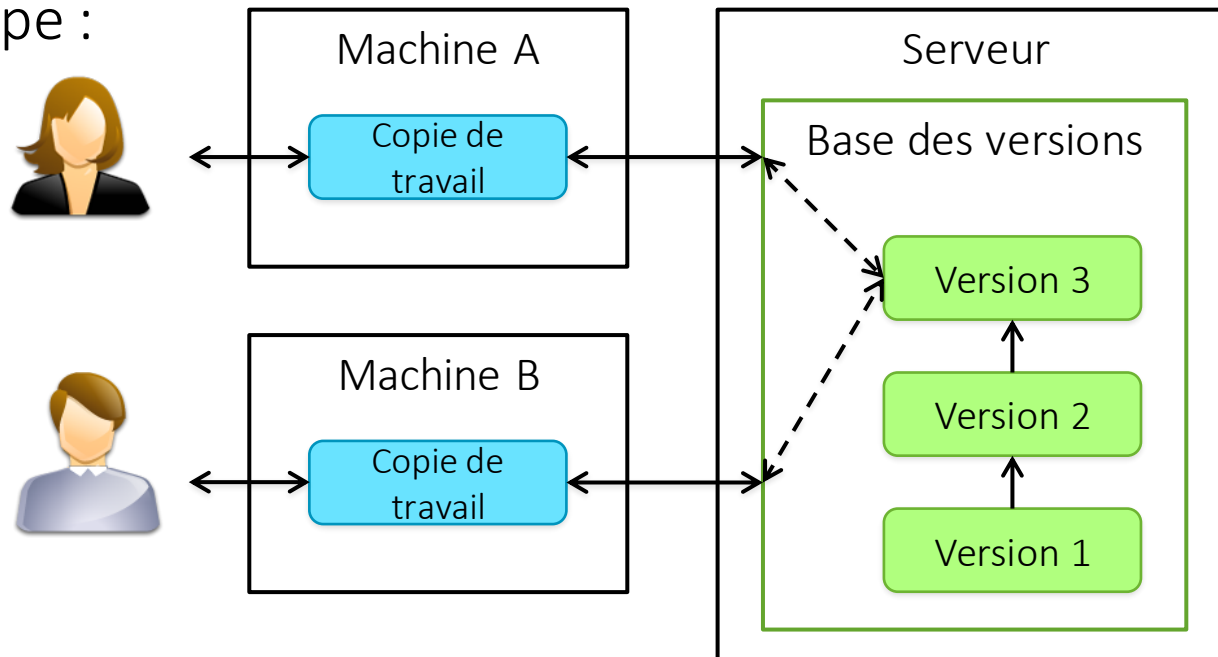
Gestion et utilisation très simples



Sensible aux pannes, pas de collaboration

2- Gestion de versions centralisée

► Principe :



► Exemples : CVS et Subversion



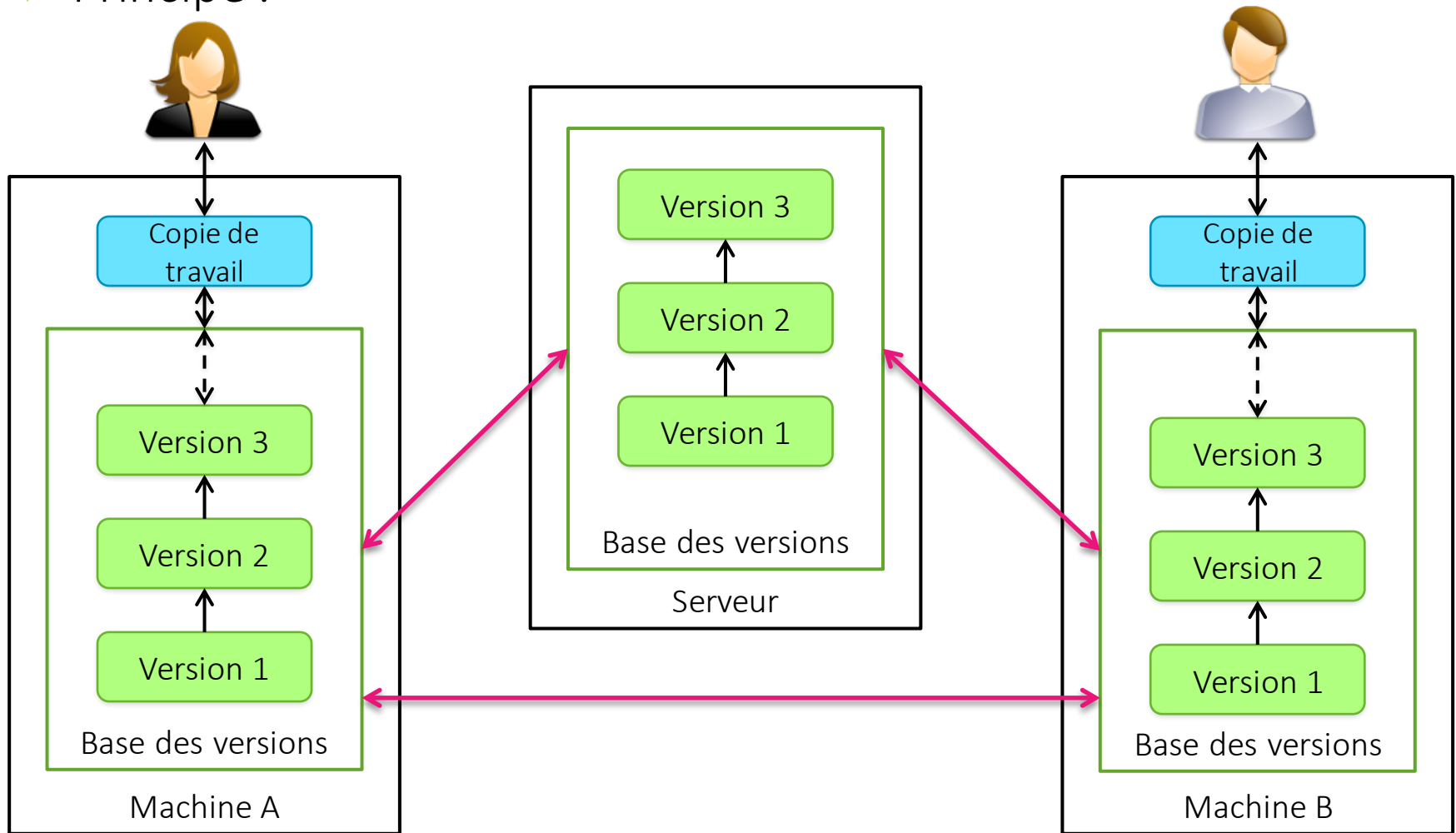
Gestion et utilisation simples



Sensible aux pannes, nécessite une connexion réseau

3- Gestion de versions décentralisée – 1/2

► Principe :



3- Gestion de versions décentralisée – 2/2

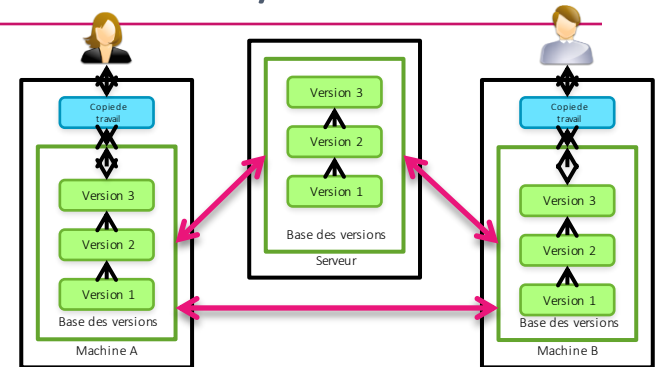
► Exemples : Git et Mercurial

► Avantages :

- Moins sensible aux pannes
 - Pas de dépendance à une seule machine
- Permet de travailler même si on n'est pas connecté au réseau
- Opérations du contributeur plus rapides car locales
- Permet le travail privé (brouillon) sans devoir publier ses modifications aux autres contributeurs
- Permet de garder un dépôt de référence contenant les versions livrées d'un projet

► Inconvénients :

- Opération longue pour le clonage d'un dépôt (tout l'historique est copié)
- Pas de système de lock donc il faut gérer les fusions (limites pour les données binaires)



Git

Historique (Wikipedia)

- ▶ De 1991 à 2002, le noyau Linux était développé sans utilisé de système de gestion de versions
- ▶ A partir de 2002, la communauté a commencé à utiliser BitKeeper, un système de gestion de versions propriétaire
- ▶ En 2005, suite à un contentieux, BitKeeper retire la possibilité d'utiliser gratuitement son produit. Linus Torvalds lance le développement de Git et après seulement quelques mois de développement, Git héberge le développement du noyau Linux
- ▶ Le magazine [*PC World*](#) nous apprend que « quand on lui a demandé pourquoi il avait appelé son logiciel “git”, qui est à peu près l'équivalent de “connard” en argot britannique^{8,9}, Linus Torvalds a répondu “je ne suis qu'un sale égoцентриque, donc j'appelle tous mes projets d'après ma propre personne. D'abord Linux, puis Git.”¹⁰ ».



Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre et distribué selon les termes de la licence publique générale GNU version 2.
Il est écrit en C, Shell Unix, Perl, Tcl et GNU Bash.

► <https://git-scm.com/>

Companies & Projects Using Git

Google

FACEBOOK

Microsoft



LinkedIn

NETFLIX



PostgreSQL



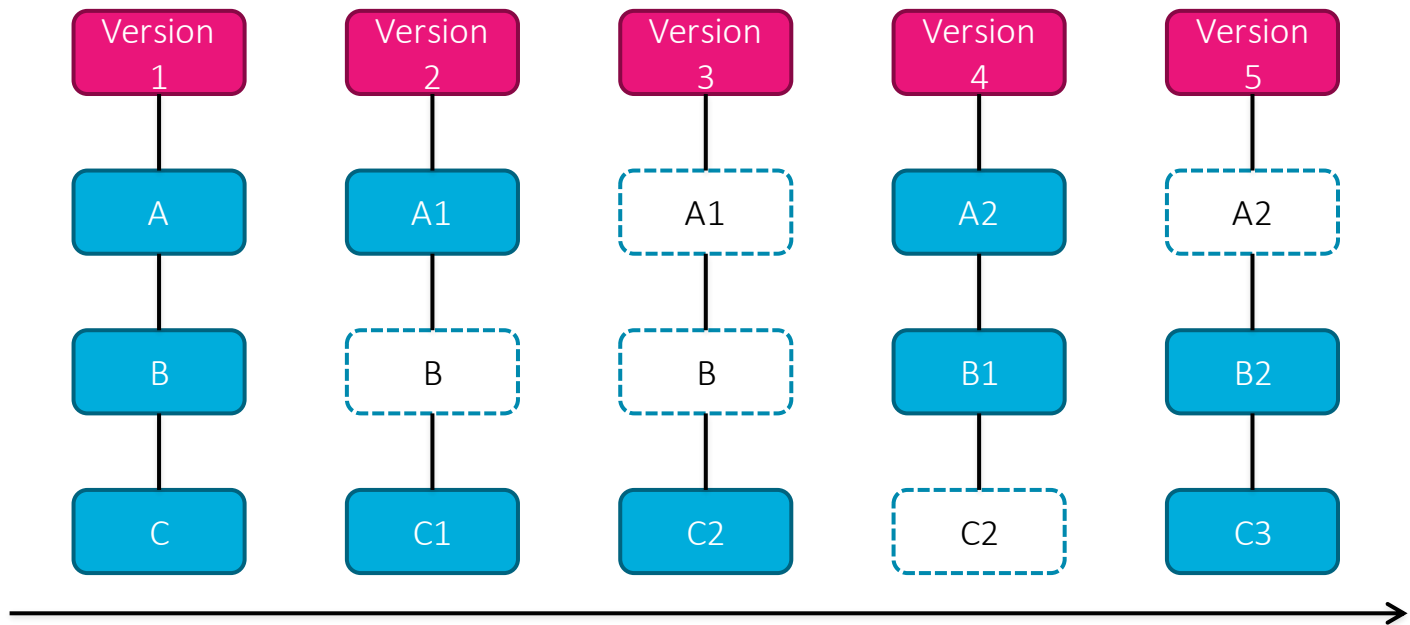
GNOME

eclipse



Principes de base

Un dépôt Git est une sorte de système de fichiers (base de données), enregistrant les versions de fichiers d'un projet à des moments précis au cours du temps sous forme d'instantanés.



Projet versionné localement avec Git

- ▶ Le **répertoire de travail** courant :

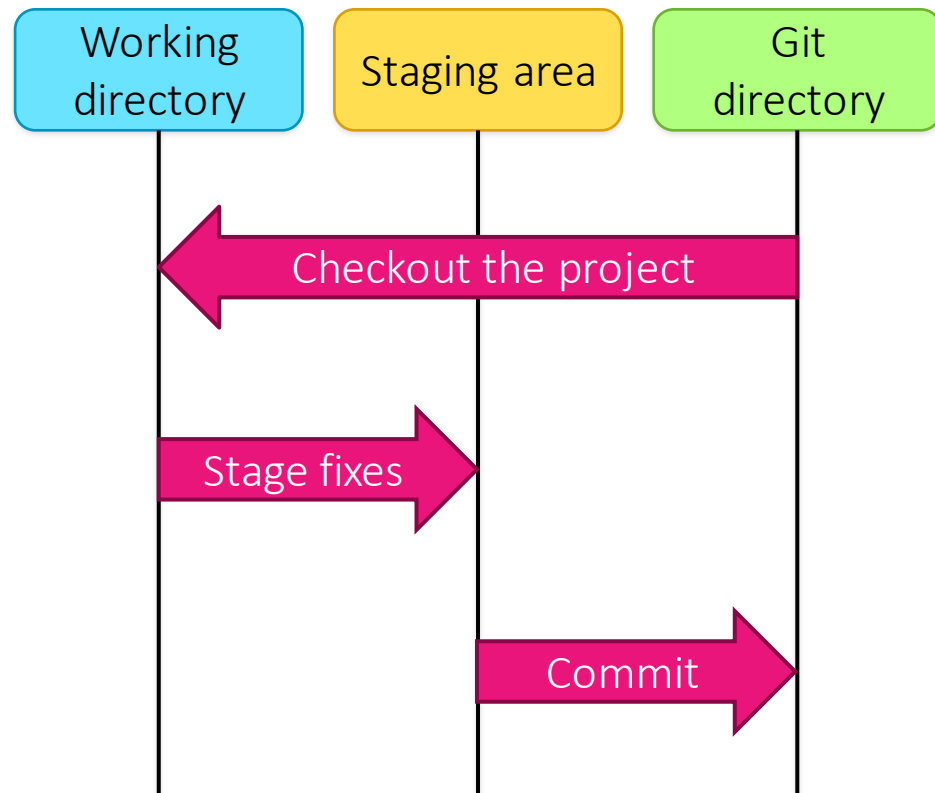
- ▶ Extraction unique d'une version du projet depuis la base de données du dépôt

- ▶ Le **répertoire Git/dépôt (.git)** :

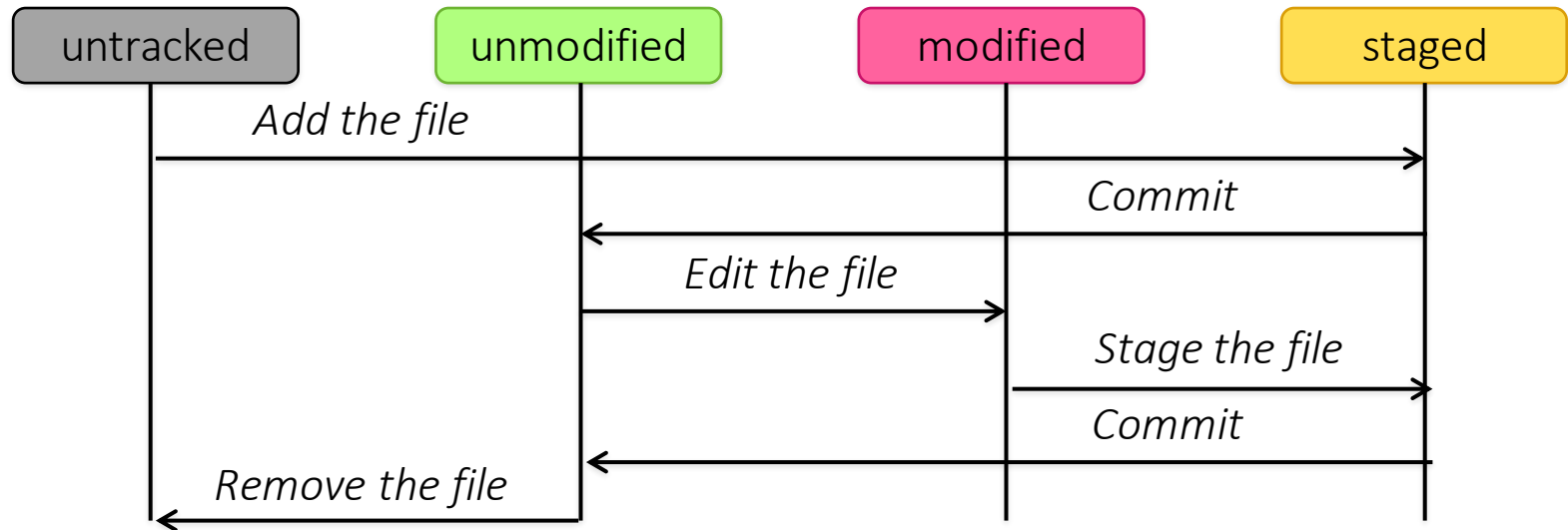
- ▶ Contient les méta-données et la base de données des objets du projet

- ▶ La **zone de transit/d'index** :

- ▶ Fichier contenant des informations à propos de ce qui sera pris en compte lors de la prochaine soumission



Etats d'un fichier



► Non versionné :

- Fichier n'étant pas ou plus géré par Git

► Non modifié :

- Fichier sauvegardé de manière sûre dans sa version courante dans la base de données du dépôt

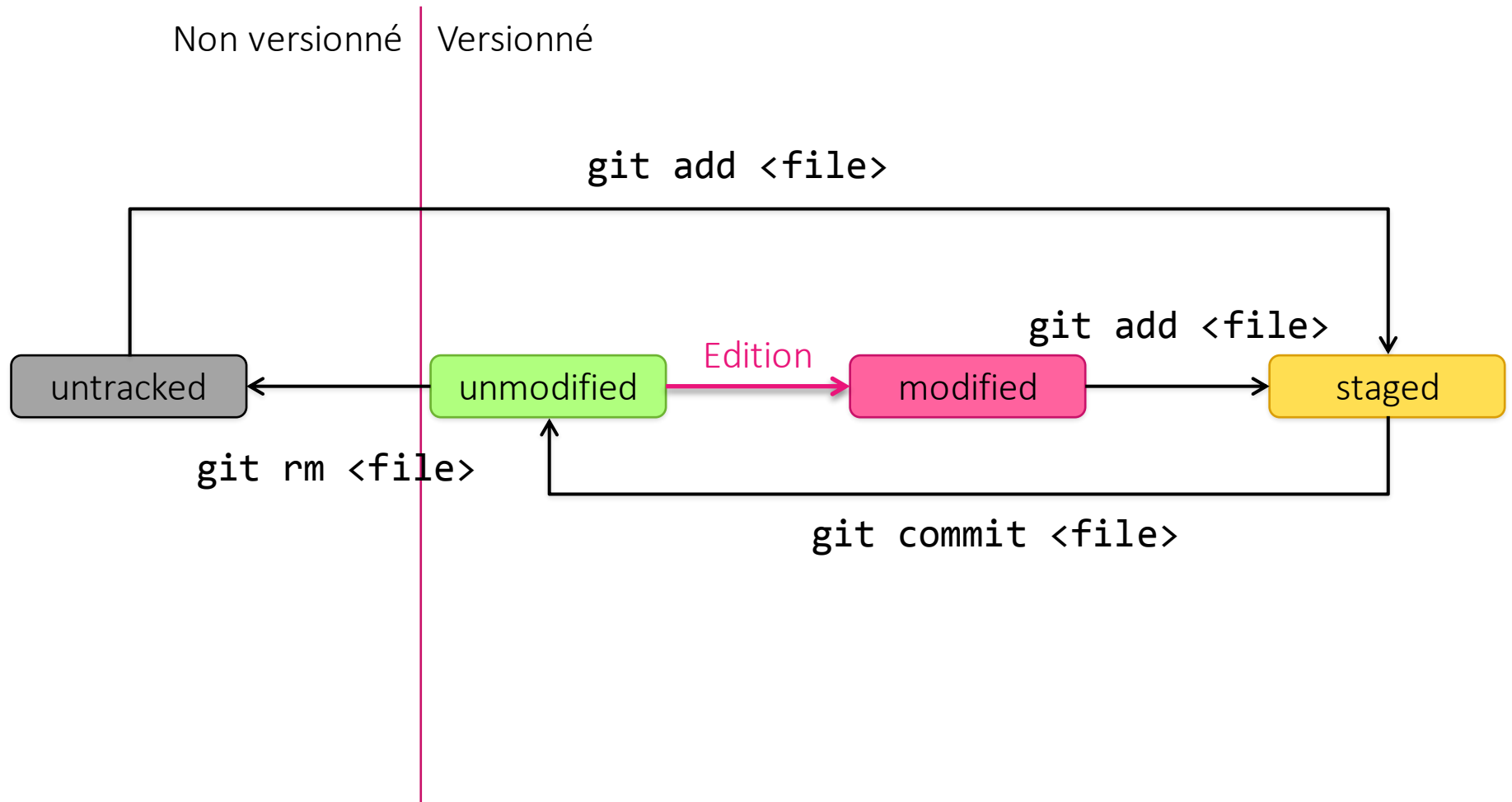
► Modifié :

- Fichier ayant subi des modifications depuis la dernière soumission

► Indexé (*staged*) :

- Idem, sauf qu'il en sera pris un instantané dans sa version courante lors de la prochaine soumission (commit)

Avec les commandes git



Commandes de base – 1/3

- ▶ Initialiser un dépôt :

- ▶ `git init`

- ▶ Afficher l'état des fichiers du répertoire courant :

- ▶ `git status`

- ▶ *Untracked files* : fichiers non versionnés
 - ▶ *Changes to be committed* : modifications (ajout, suppression, changement) chargées en zone de transit (staging area) ou indexées
 - ▶ *Changes not staged for commit* : modifications n'ayant pas été chargées en zone de transit (ou indexées)

Commandes de base – 2/3

- ▶ Indexer l'ajout ou les changements d'un fichier
 - ▶ `git add <fichier>`
- ▶ Annuler les modifications indexées d'un fichier
 - ▶ `git reset <fichier>`
- ▶ Annuler les modifications non encore indexées d'un fichier
 - ▶ `git checkout [--] <fichier>`
- ▶ Indexer la suppression d'un fichier
 - ▶ `git rm <fichier>`
- ▶ Déversionner un fichier
 - ▶ `git rm -cached <fichier>`

Commandes de base – 3/3

- ▶ Afficher le détail des modifications non indexées
 - ▶ `git diff`
- ▶ Afficher le détail des modifications indexées
 - ▶ `git diff --staged`
- ▶ Soumettre les modifications indexées en zone de transit
 - ▶ `git commit`
- ▶ Voir l'historique des soumissions
 - ▶ `git log`

Git

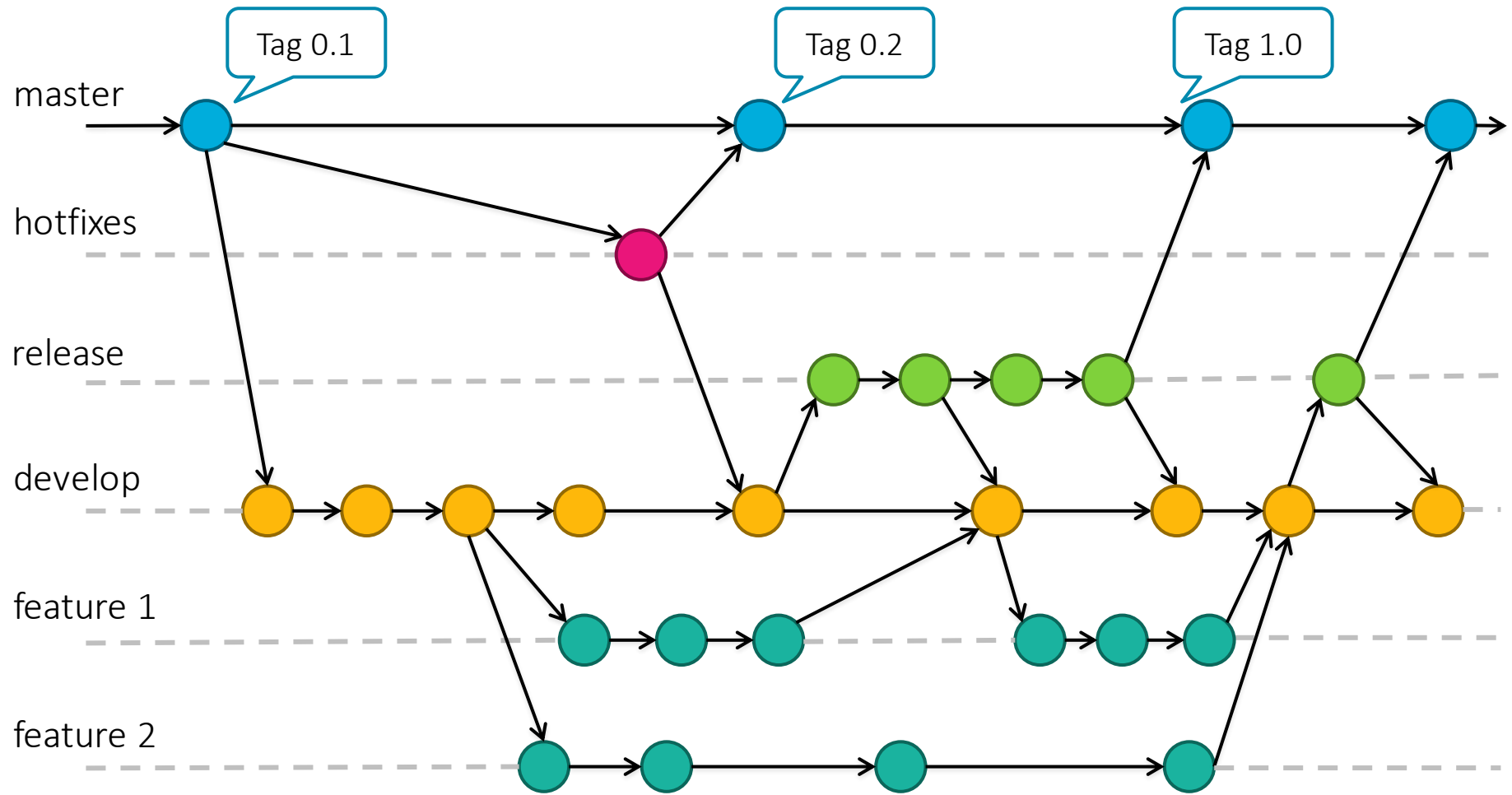
Les branches

Branche d'un projet

Une **branche d'un projet** est une ligne d'évolution divergeant de la ligne d'évolution courante, celles-ci se poursuivant **indépendamment** l'une de l'autre.

- ▶ Intérêt :
 - ▶ Possibilité d'avoir des évolutions de versions en parallèle
- ▶ Deux stratégies :
 - ▶ **Une branche par produit** (*Branch by Product*) : un client veut une modification qui lui est spécifique (contrainte externe)
 - ▶ **Une branche par objectif** (*Branch by Purpose*) : nécessité de continuer de développer tout en permettant la correction de bugs sur une branche en cours de tests (contrainte interne)

Exemple de git flow – 1/2



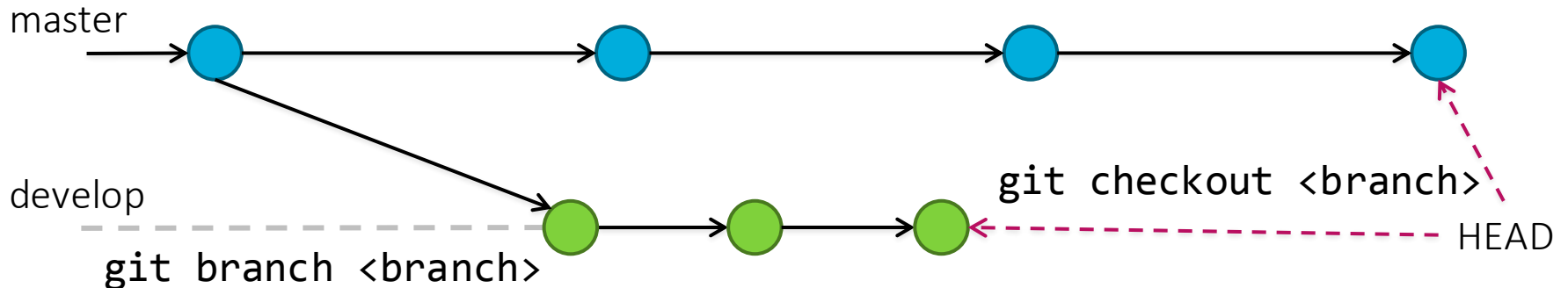
Exemple de git flow – 2/2

► Idée :

- Une branche develop pour le développement
- Une branche release pour préparer les livraisons
- Une branche master qui ne contient que des commits « stables »
- Autant de branches hotfix que de bugs sur ce qui a été livré
- Autant de branches (éphémères) que de features/développeurs

Branche avec Git

- ▶ Une branche est un pointeur vers un objet commit
- ▶ Par défaut, il existe une seule branche nommée « master »
- ▶ HEAD est un pointeur spécial vers la branche sur laquelle on travaille



Commandes pour les branches

- ▶ Créer une nouvelle branche :

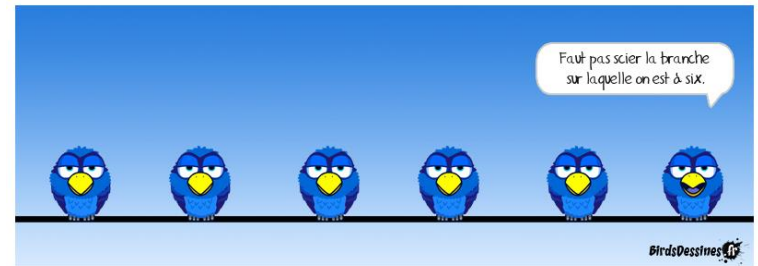
- ▶ `git branch <branche>`

- ▶ Voir les branches du dépôt local :

- ▶ `git branch`

- ▶ Supprimer une branche :

- ▶ `git branch -d <branche>`

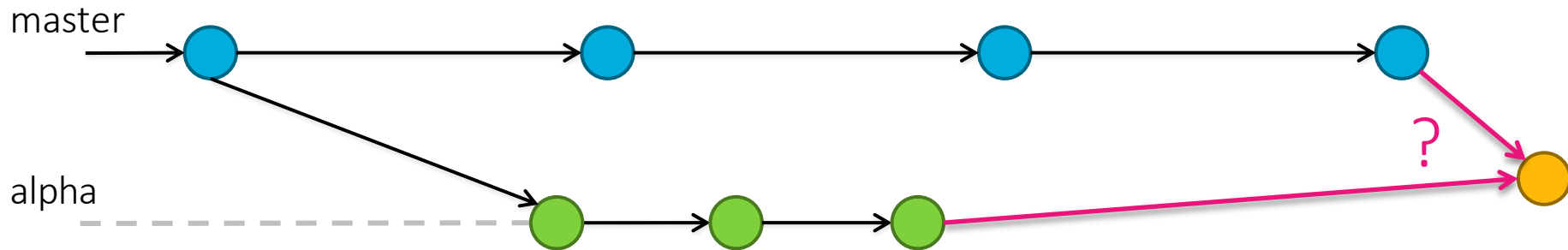


- ▶ Passer à une branche donnée (mise à jour de l'index et du répertoire de travail, ainsi que du pointeur HEAD) :

- ▶ `git checkout <branche>`

Problématique : incorporer des modifications

- Comment incorporer des modifications venant d'une autre branche ?

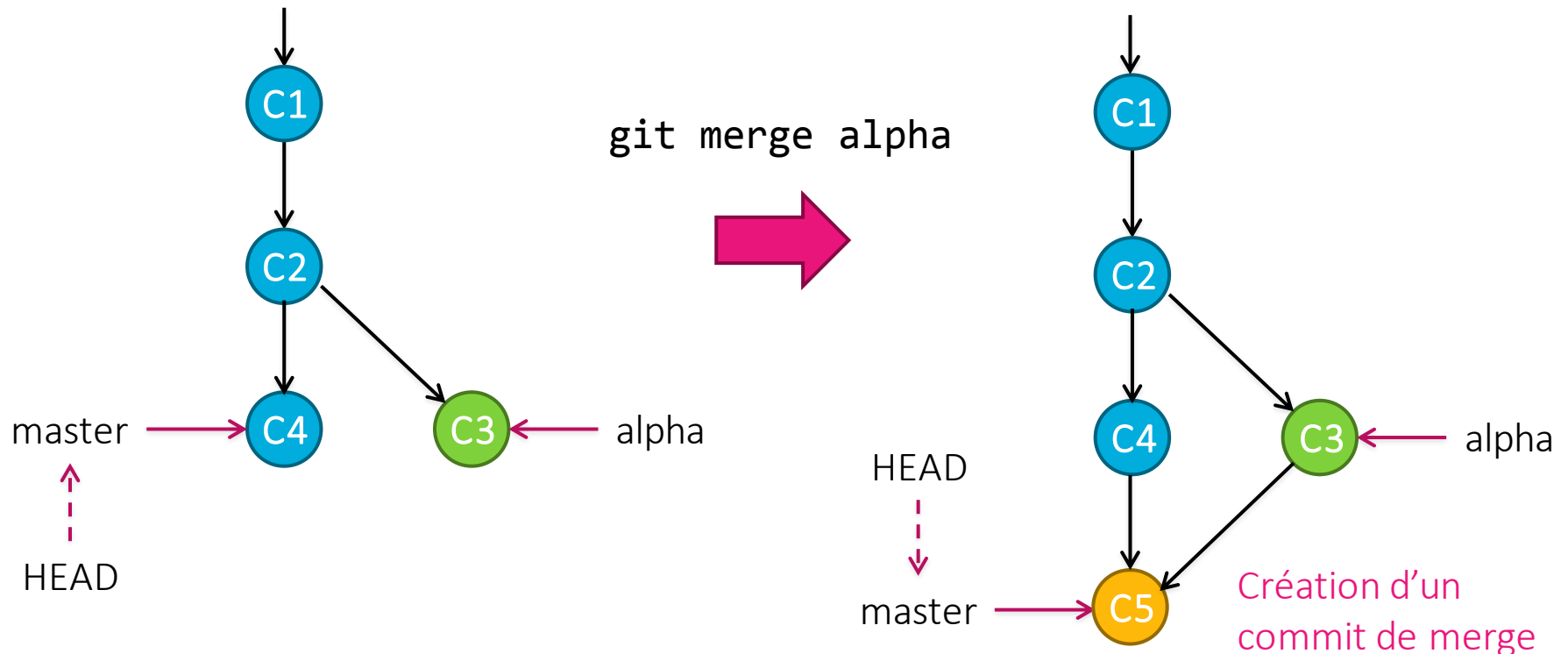


- Deux techniques :
 - Fusionner (merge)
 - Rebaser (rebase)

Fusionner deux branches : `git merge <branche>`

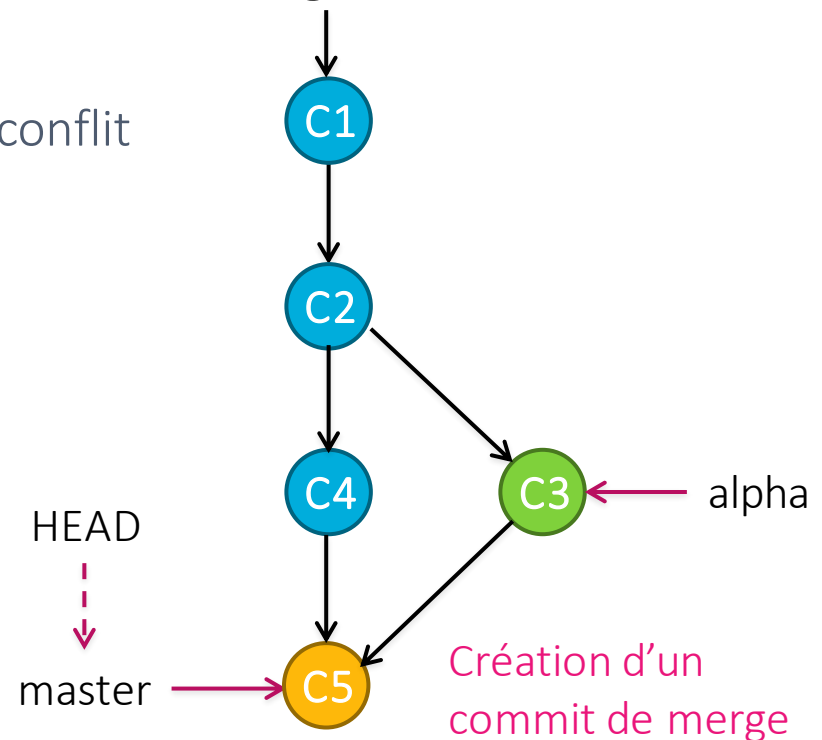
► Fusionner deux branches :

- Rattrier les modifications d'une branche dans une autre
- Par défaut, la fusion concerne tous les *commits* depuis la dernière fusion ou la création de la branche



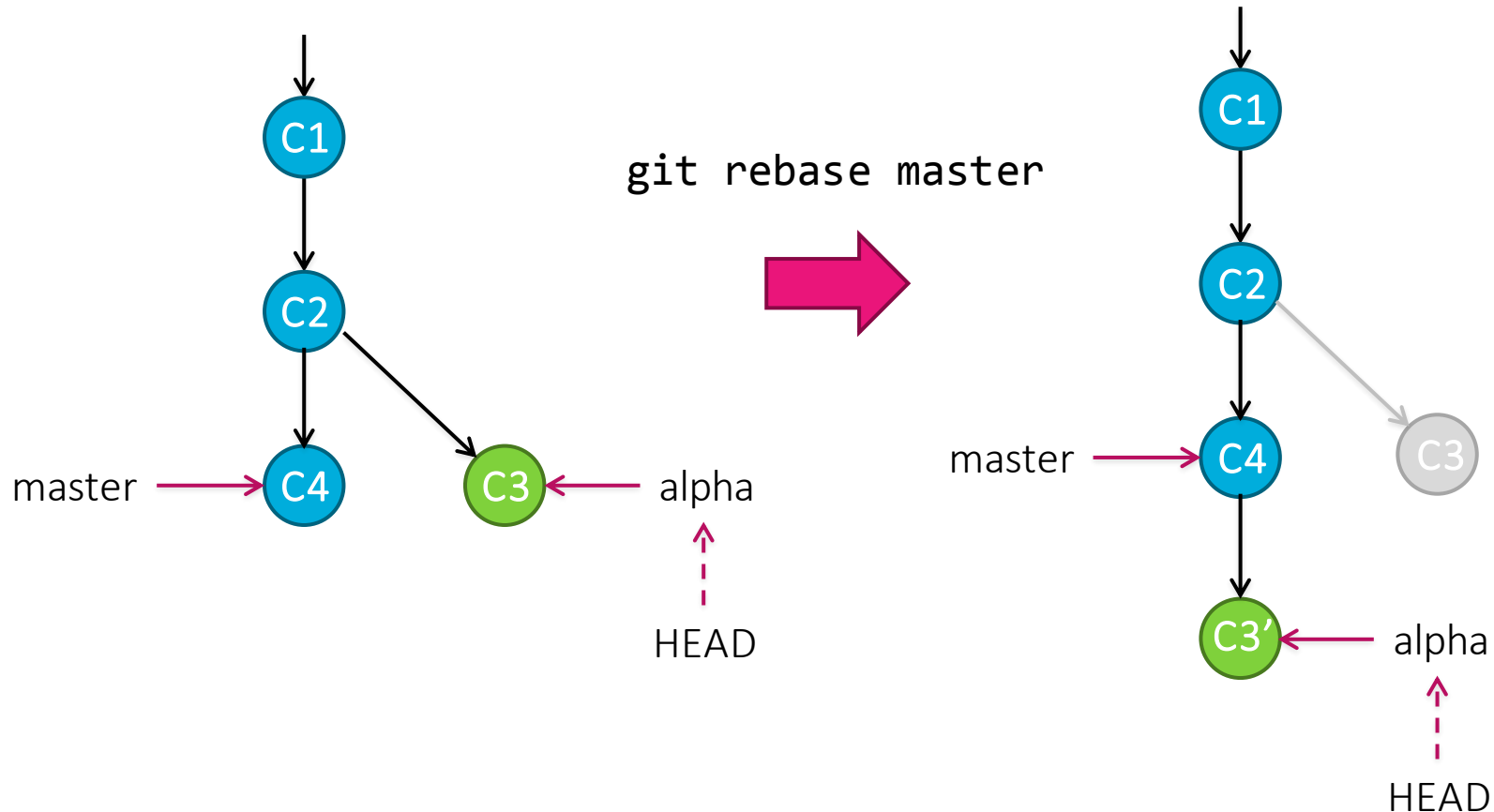
Fusionner deux branches : `git merge <branche>`

- ▶ En principe : c'est facile
- ▶ En réalité : gestion de conflits
- ▶ Exemple : si C3 et C4 modifient la même ligne d'un fichier
 - ▶ Git ne sait pas quoi choisir
 - ▶ Il faut manuellement résoudre le conflit



Rebaser deux branches : `git rebase <branche>`

- Modifie et/ou réécrit l'historique des commits



Merge vs. Rebase

► Merge :

- Pour la réintégration de branches de feature
- On veut garder l'historique des commits indépendants sans polluer l'historique de la branche principale

► Rebase :

- Pour la mise à jour de branches comme la correction d'anomalies
- Rend votre arbre de commits très propre puisqu'il ressemble à une ligne droite mais il modifie l'historique (apparent) de l'arbre des commits.
- Ne jamais rebaser des commits qui ont déjà été poussés sur un dépôt public

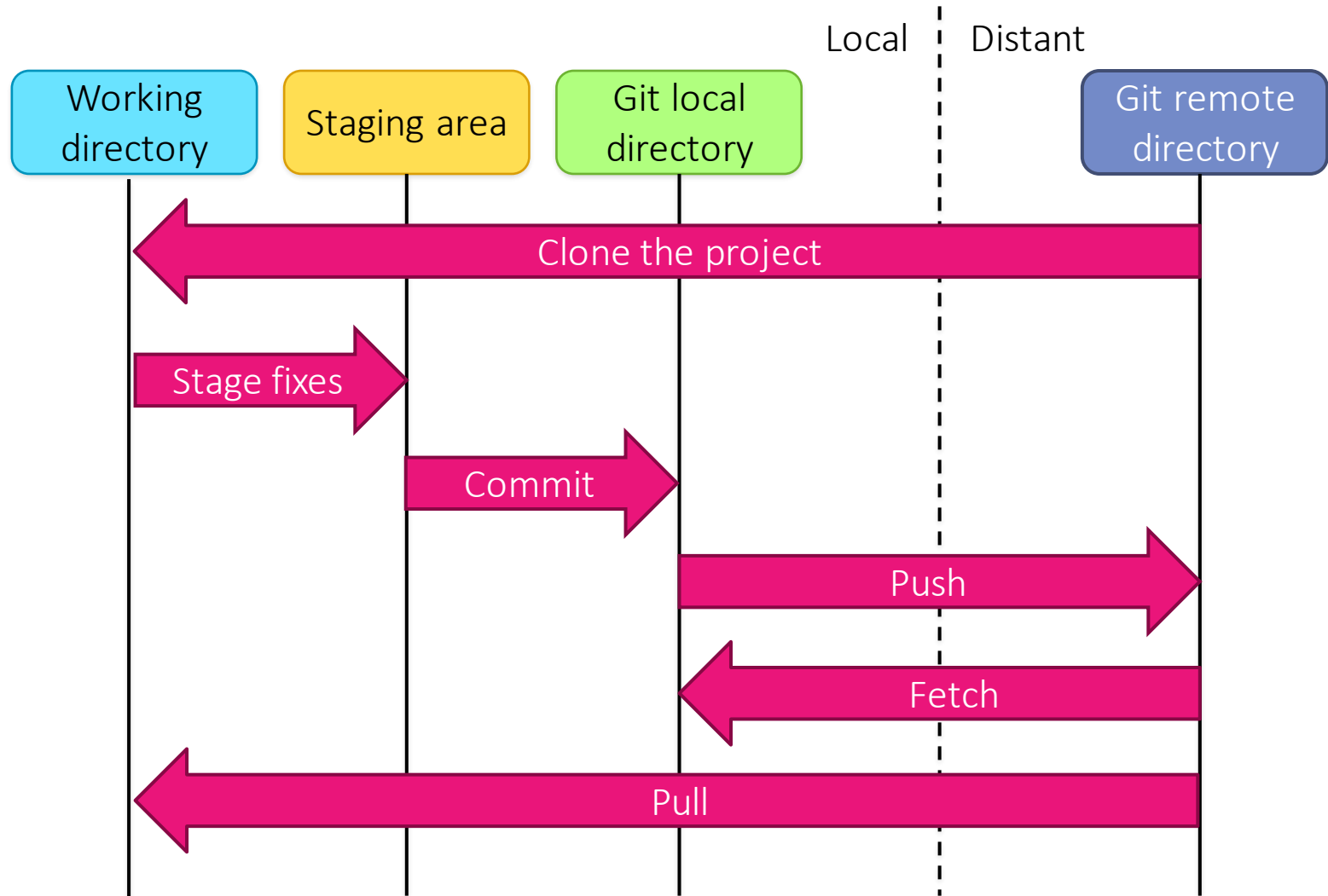
Revenir en arrière

- ▶ Cas de modifications commitées :
 - ▶ amend : modifier le dernier commit:
 - ▶ Ajout de fichiers au commit
 - ▶ Changement de message du commit
 - ▶ revert : annuler un commit par un autre commit
 - ▶ reset : rétablir la situation d'un ancien commit

Git

Dépôts distants

Dépôt distant



Commandes pour les dépôts distants

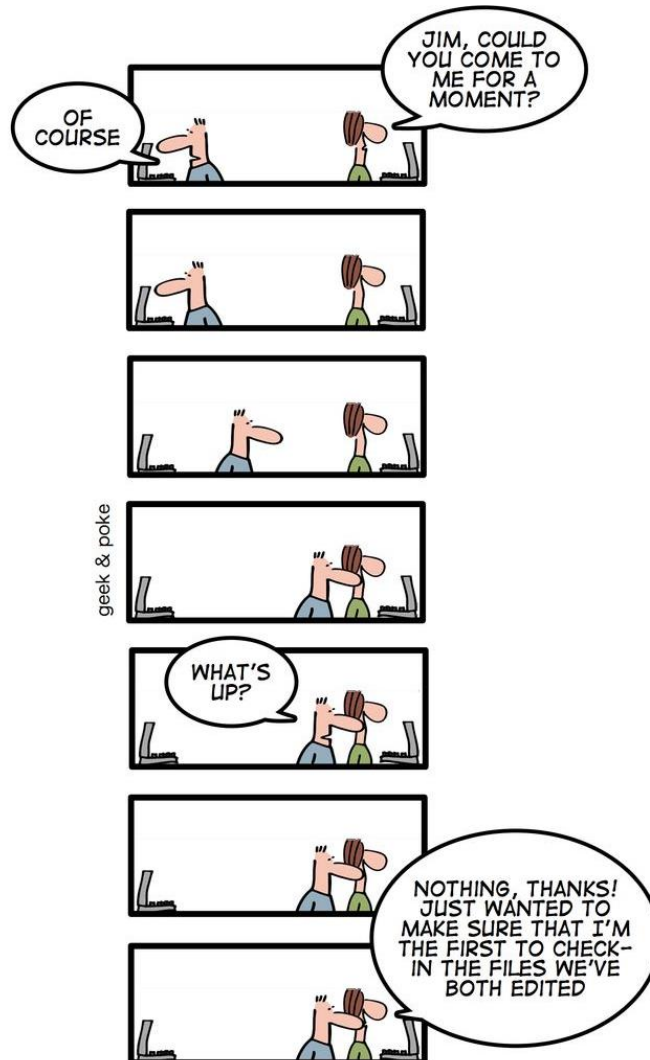
- ▶ Cloner un dépôt distant dans un nouveau répertoire :
 - ▶ `git clone <url_dépôt> <répertoire>`
- ▶ Récupérer/télécharger les branches ou les commits d'un autre dépôt
 - ▶ `git fetch <dépôt>`
- ▶ Rapatrier et intégrer un autre dépôt ou une branche locale
 - ▶ `git pull <dépôt>`
- ▶ Mettre à jour les références distantes ainsi que les objets associés
 - ▶ `git push <dépôt>`

Gestion des conflits – Bonne pratique



1. Je fais des modifications
2. Je peux faire *git fetch* pour comparer mais pas *git pull*
3. J'indexe mes modifications (*git add/git commit*)
4. Je fais *git pull*
5. J'ai un conflit
6. J'édite pour résoudre
7. J'indexe (*add/commit*) ce qui a été résolu
8. Je peux faire *git push* (on n'oublie pas le pull)

Gestion des conflits – Mauvaise pratique



<https://geekandpoke.typepad.com/geekandpoke/2010/10/being-a-code-made-easy-chapter-1.html>

Services d'hébergement de dépôts distants

► Github :

► <https://github.com/>



► GitLab :

► <https://about.gitlab.com/>

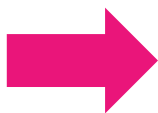


► Bitbucket :

► <https://bitbucket.org/>



Lequel choisir ?



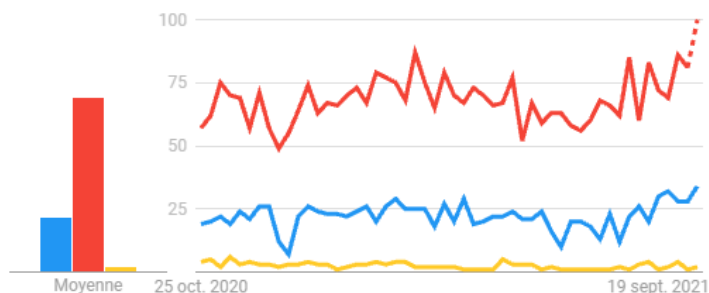
Dépôt GitLab hébergé à l'Ensimag :
https://gitlab.ensimag.fr/users/sign_in

Critères

► En version Cloud :

		Github	GitLab	Bitbucket
Open source	Jusqu'à 5 utilisateurs	0\$	0\$	0\$
	Supérieur à 5 utilisateurs	0\$	0\$	2\$/utilisateur/Mo
Répertoire privé	Jusqu'à 5 utilisateurs	25\$/Mo	0\$	0\$
	Supérieur à 5 utilisateurs	\$9/utilisateur/Mo	0\$	2\$/utilisateur/Mo

● gitlab ● github ● bitbucket



Facilité d'intégration avec les outils d'intégration continue (CI/CD) ?

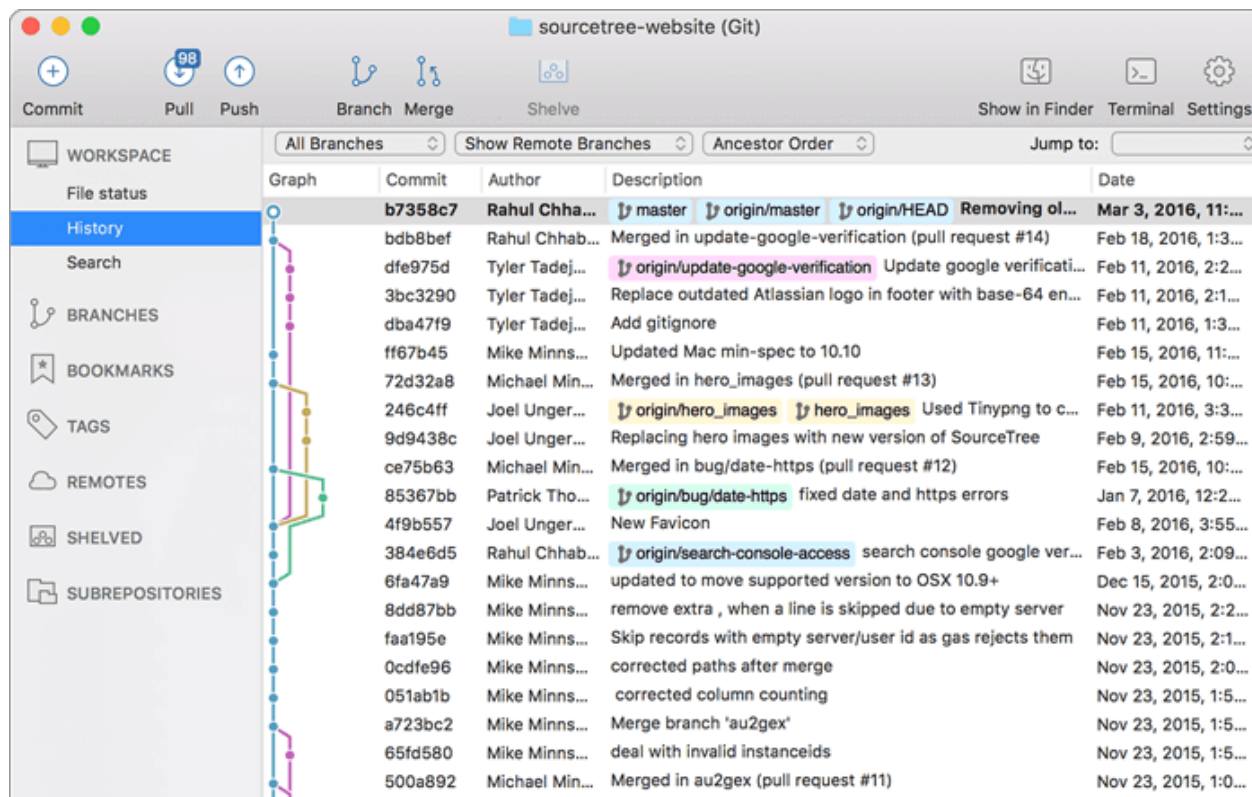
Git

Les outils

GUI Clients

► Sourcetree :

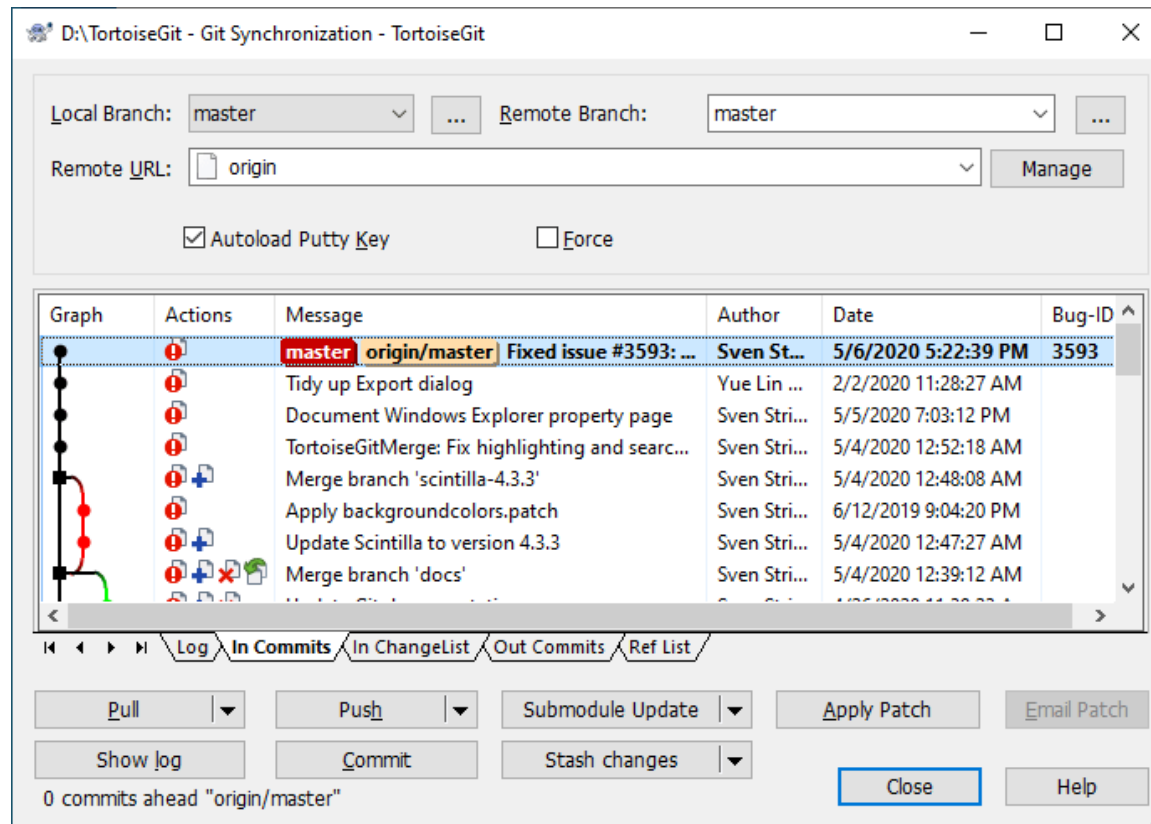
► <https://www.sourcetreeapp.com/>



GUI Clients

► TortoiseGit :

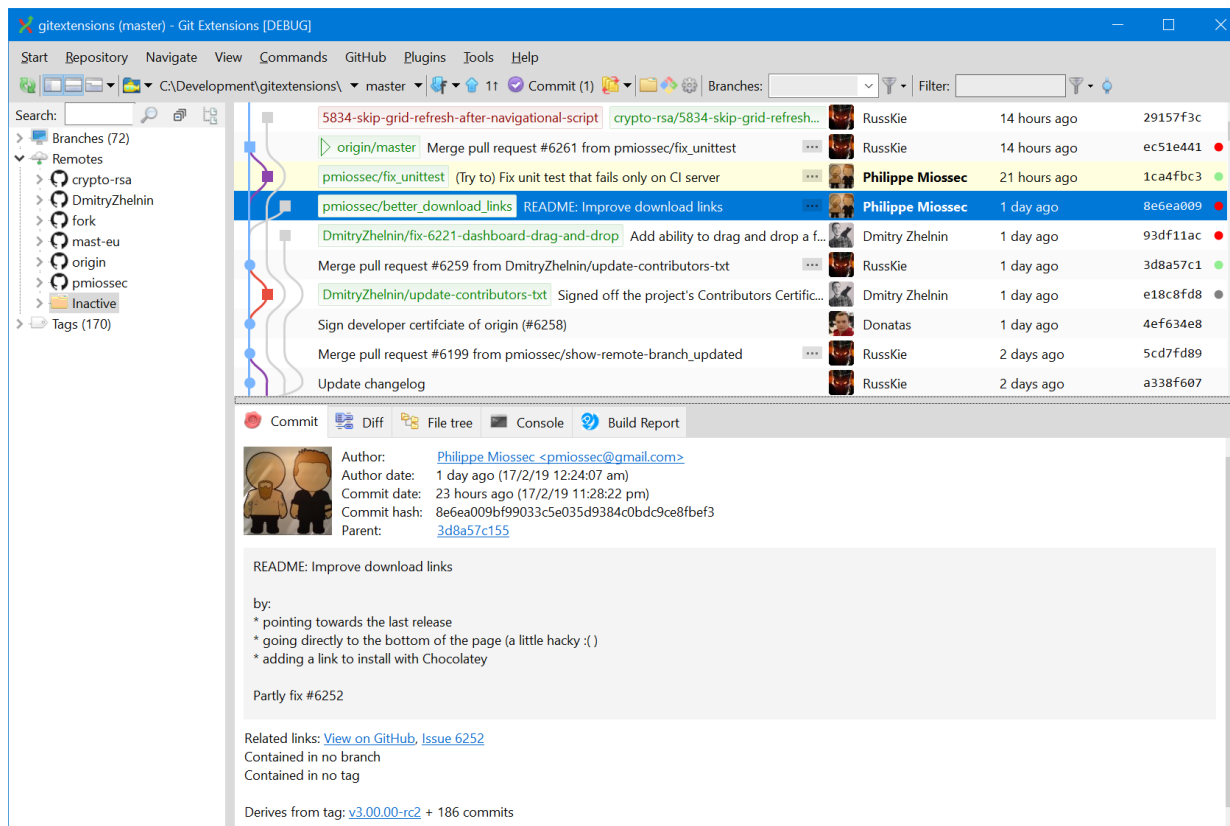
► <https://tortoisegit.org/>



GUI Clients

► Git Extensions :

► <https://gitextensions.github.io/>



GUI Clients

▶ GitKraken :

▶ <https://www.gitkraken.com/>

The screenshot displays the GitKraken application interface. On the left, a sidebar shows the 'LOCAL' repository structure with branches like 'fancier-refbar-changes', 'fancy-responsive-refbar-it...', 'graph-color-test', 'hopscootch', 'init-repo-gitignore-typeahead', 'invite-system', 'jars-view-file-history', 'master', 'remote-panel-redesign', 'settings-theme-styling', and 'view-file-history'. Below this, a 'REMOTE' section lists other repositories. The main area features a commit history graph with nodes and branches. The right sidebar shows a detailed view of a specific commit, including the commit message, author information, and a diff of the files 'package.json' and 'npm-shrinkwrap.json'.

Commit: cca151e6b9e32c39209c25131706740050
Parent: 8efe30a11761983173f844900fa5ec5c6be2
Author: John Haley <johnh@axosoft.com>
Author Date: September 30th 2015, 2:54 pm

Bump to version 0.1.40

0 added 0 deleted 2 modified

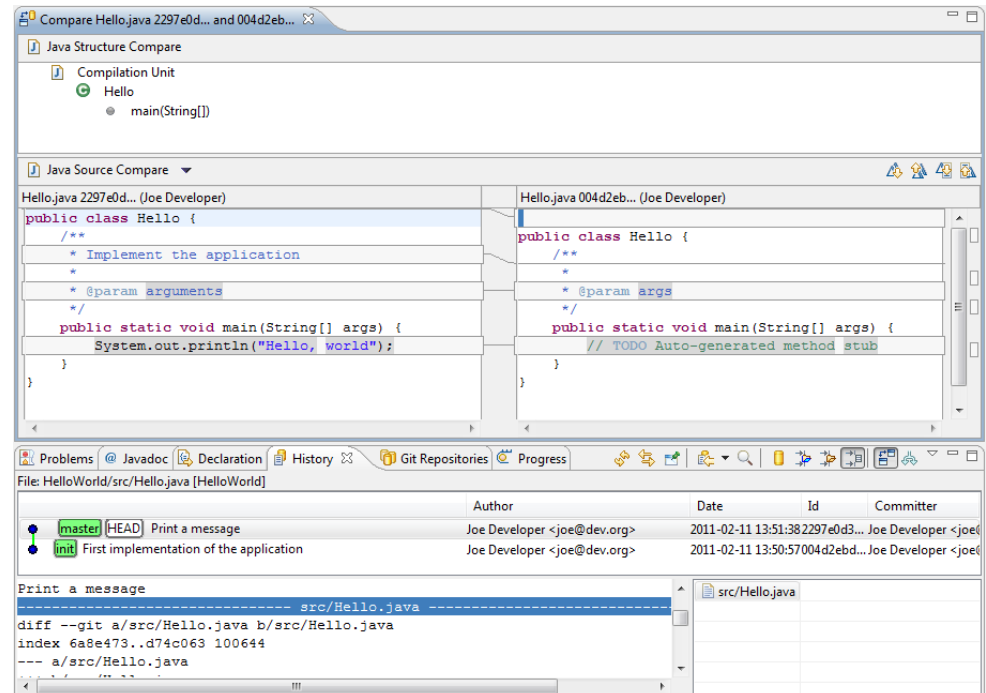
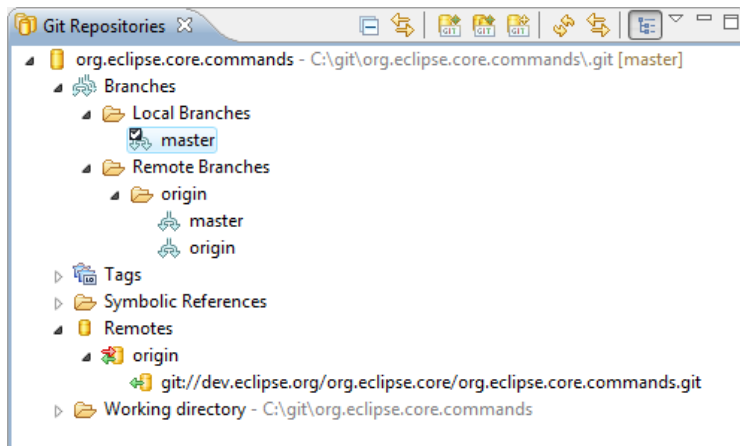
package.json

```
@@ -1,7 +1,7 @@
1 1 {
2 2   "name": "gitkraken",
3 3   "productName": "GitKraken",
-4 4   "version": "0.1.39",
+4 4   "version": "0.1.40",
5 5   "description": "An intuitive git cli
6 6   "main": "./src/appBootstrap/main.js"
```

Dans Eclipse

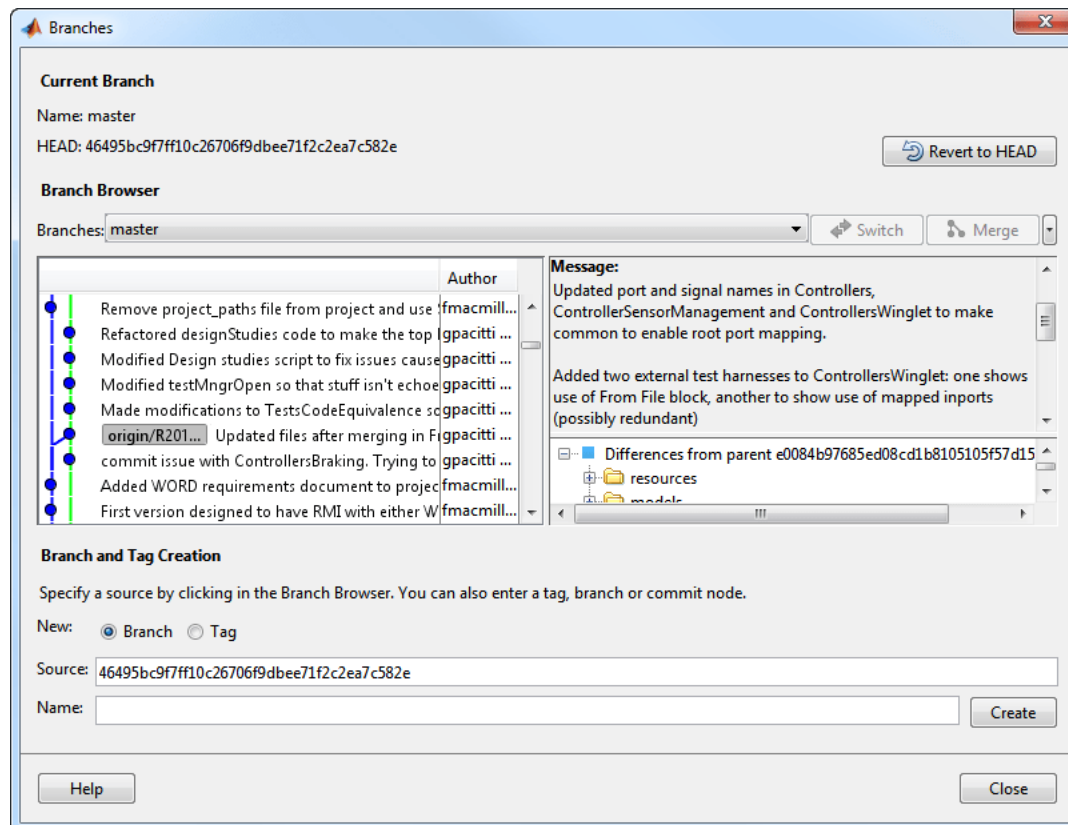
► Plugin EGit :

- <https://www.eclipse.org/egit/>
- Installé par défaut dans Eclipse
- Documentation : https://wiki.eclipse.org/EGit/User_Guide



Dans Matlab

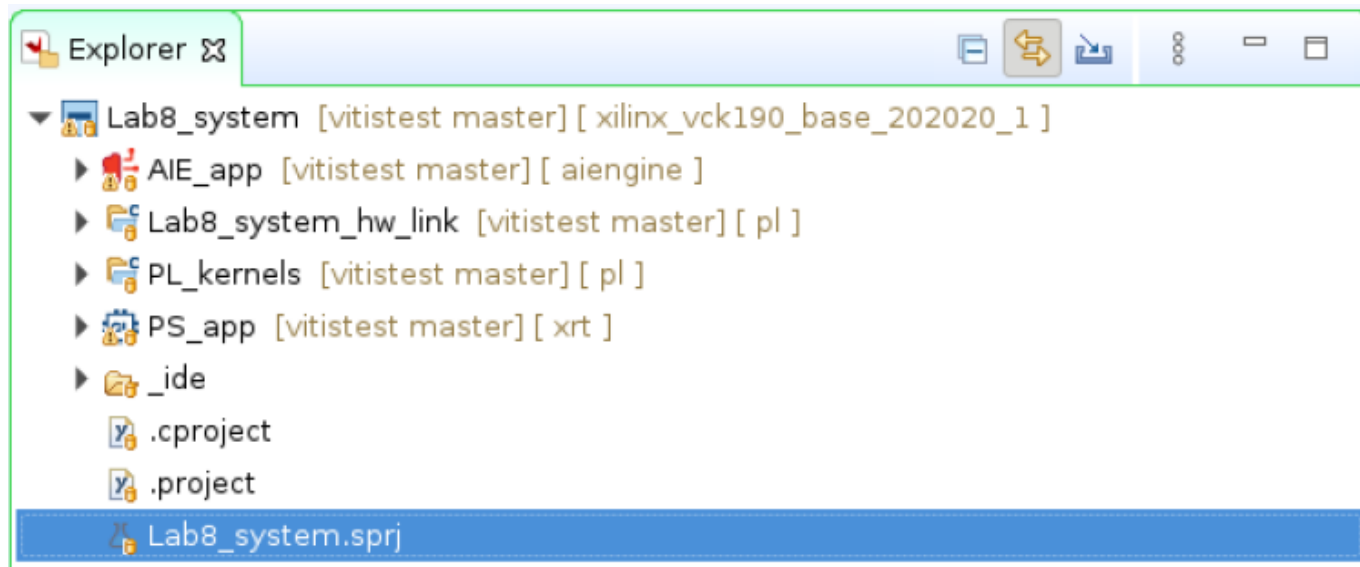
- ▶ Documentation :
- ▶ https://fr.mathworks.com/help/matlab/source-control.html?s_tid=CRUX_lftnav



Dans Xilinx

- Documentation :

https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/versioncontrol.html



Git

Conclusion

Conclusion

► Documentation officielle :

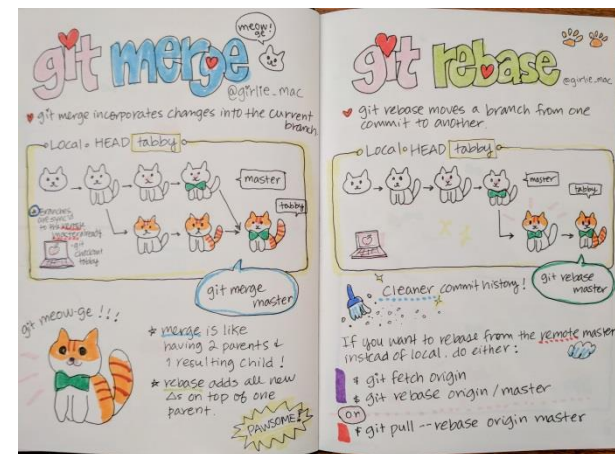
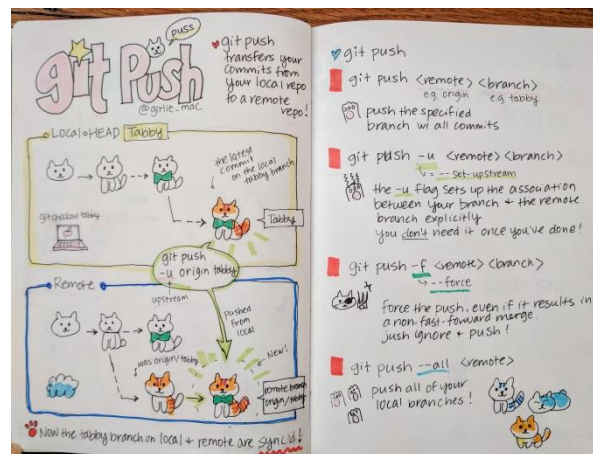
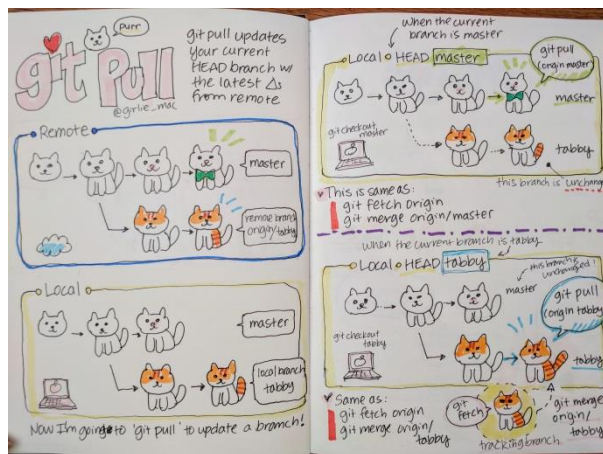
► <https://git-scm.com/>

► D'autres documentations :

► <https://training.github.com/downloads/fr/github-git-cheat-sheet.pdf?source=korben.info>

► <http://rogerdudler.github.io/git-guide/index.fr.html>

► <https://girliemac.com/blog/2017/12/26/git-purr/>



A faire

- ▶ Pas de TD sur Git mais à faire en travail personnel :

- ▶ https://learngitbranching.js.org/?locale=fr_FR

