Introduction au Génie Logiciel

Séance 6 : Bonnes pratiques de programmation et DevOps

L. Laversa
laversa@irif.fr

Université Paris Cité

11 mars 2025

. Remerciements à E. Bigeon et J. Lefebvre

Les diagrammes de séquence

Objectif

Représentation graphique de la chronologie des échanges de messages entre les acteurs et le système

Les diagrammes de séquence

Objectif

Représentation graphique de la chronologie des échanges de messages entre les acteurs et le système

Interactions entre les acteurs et le système :

Pas de messages entre acteurs !

Les diagrammes de séquence

Objectif

Représentation graphique de la chronologie des échanges de messages entre les acteurs et le système

Interactions entre les acteurs et le système :

Pas de messages entre acteurs !

On ne peut pas avoir différents cas sur un même diagramme :

Un diagramme = un scénario

Dette technique

Quel est le temps nécessaire pour continuer à faire avancer le projet ?

- Choix à court terme pour résoudre un problème rapidement,
- mais entraînant des complications à long terme.

$\Rightarrow$ Difficultés à comprendre le code.

$\Rightarrow$ Perte de contrôle du projet.

Bonnes pratiques de code

Pourquoi ?

« Un développeur lit du code. Et parfois, il en écrit. »

Code lisible :

- Faire des changements simplement
- Être sûr de ne pas avoir introduit de nouvelles erreurs
- Gain de temps (- de temps à comprendre le code, + de temps à travailler sur l'implem')
- Partage de savoir / formation d'un nouveau membre de l'équipe

Bonnes pratiques de code

```
public static int function(int index){
    if (index < 0) throw new IndexOutOfBoundsException(
        index);
    if (index==0)
        return 1; if (index == 1)
    return 1; return function(index-1) + function(index-2);
}
```

Bonnes pratiques de code

```
public static int function(int index){
    if (index < 0) throw new IndexOutOfBoundsException(
        index);
    if (index==0)
        return 1; if (index == 1)
    return 1; return function(index-1) + function(index-2);
}
```

- Indentation
- Limites de blocs conditionnels
- Paramètres (ordre, nombre, disposition, ...)
- Noms utiles
- ...

Bonnes pratiques de code

```
public static int fibonacci(int entier){
    if (entier < 0) throw new IndexOutOfBoundsException(entier);
    if (entier == 0) return 1;
    if (entier == 1) return 1;
    return fibonacci(entier-1) + fibonacci(entier-2);
}
```

Bonnes pratiques de code

```
public static int f(int n){
    if (n < 0) throw new IndexOutOfBoundsException(n);
    if (n == 0) return 1;
    if (n == 1) return 1;
    return f(n-1) + f(n-2);
}
```

- Être concis... Mais pas trop !

Linters

Linters

Outil d'analyse statique du code permettant d'identifier et de corriger des potentielles sources d'erreurs syntaxiques (mais pas sémantiques).

Exemple

Pour Java :

`checkstyle, findbugs, sonarlist`

Autres sources de problèmes

- Magic Numbers

```java
public class Exemple {
    public static void main(String[] args) {
        int rayon = 5;
        double aire = 3.14159 * rayon * rayon;
        // 3.14159 est un magic number
        System.out.println("L'aire du cercle est : " + aire);
    }
}
```

Autres sources de problèmes

- Magic Numbers

```java
public class Exemple {
    // Declaration d'une constante pour pi
    public static final double PI = 3.14159;

    public static void main(String[] args) {
        int rayon = 5;
        double aire = PI * rayon * rayon;
        // Utilisation de la constante PI
        System.out.println("L'aire du cercle est : " + aire);
    }
}
```

Autres sources de problèmes

- Magic Numbers

```java
public class Exemple {
    // Declaration d'une constante pour pi
    public static final double PI = 3.14159;

    public static void main(String[] args) {
        int rayon = 5;
        double aire = PI * rayon * rayon;
        // Utilisation de la constante PI
        System.out.println("L'aire du cercle est : " + aire);
    }
}
```

- Complexité cyclomatique, *Too many return*
- Fuite mémoires potentielles

Bonnes pratiques de groupe

Uniformité

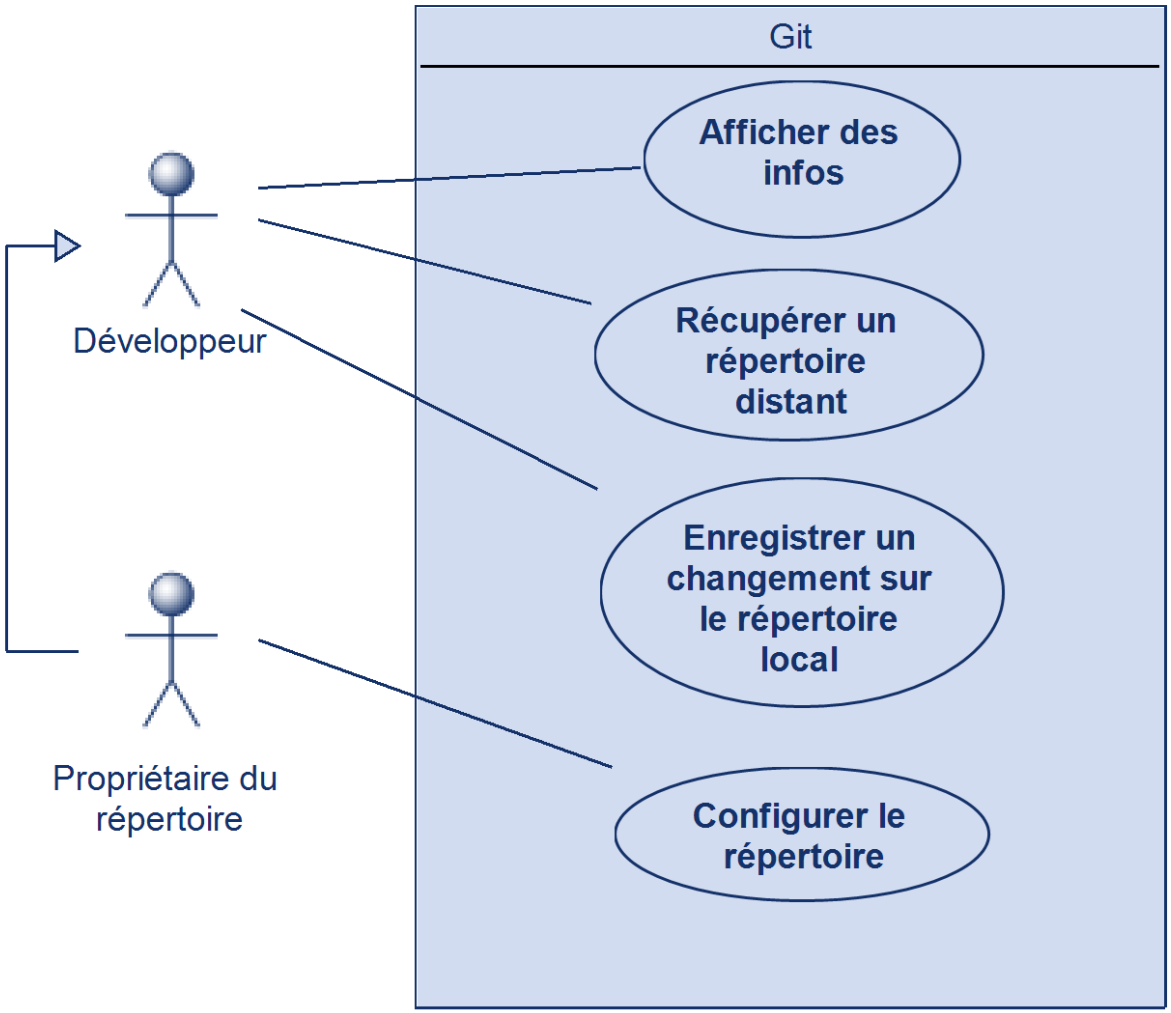
Choisir une convention et s'y tenir.

$\Rightarrow$ Consensus de groupe

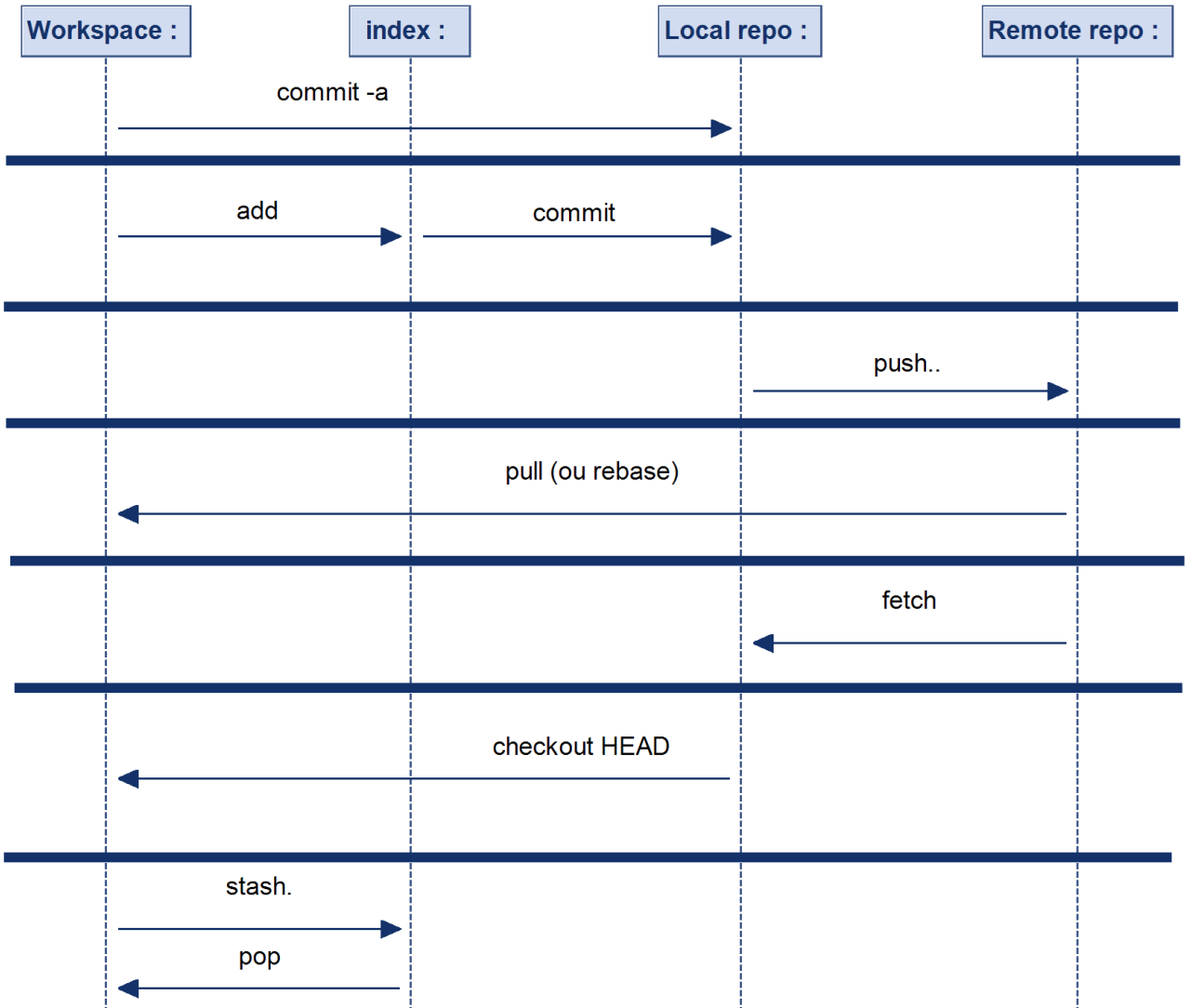
Pourquoi un gestionnaire de version ?

- Garder un historique
  - Résolution de problèmes
  - Construction du projet
- Décentralisation
  - Évite la perte de données
  - Travail à l'international

Git point de vue utilisateur - Exemples d'actions

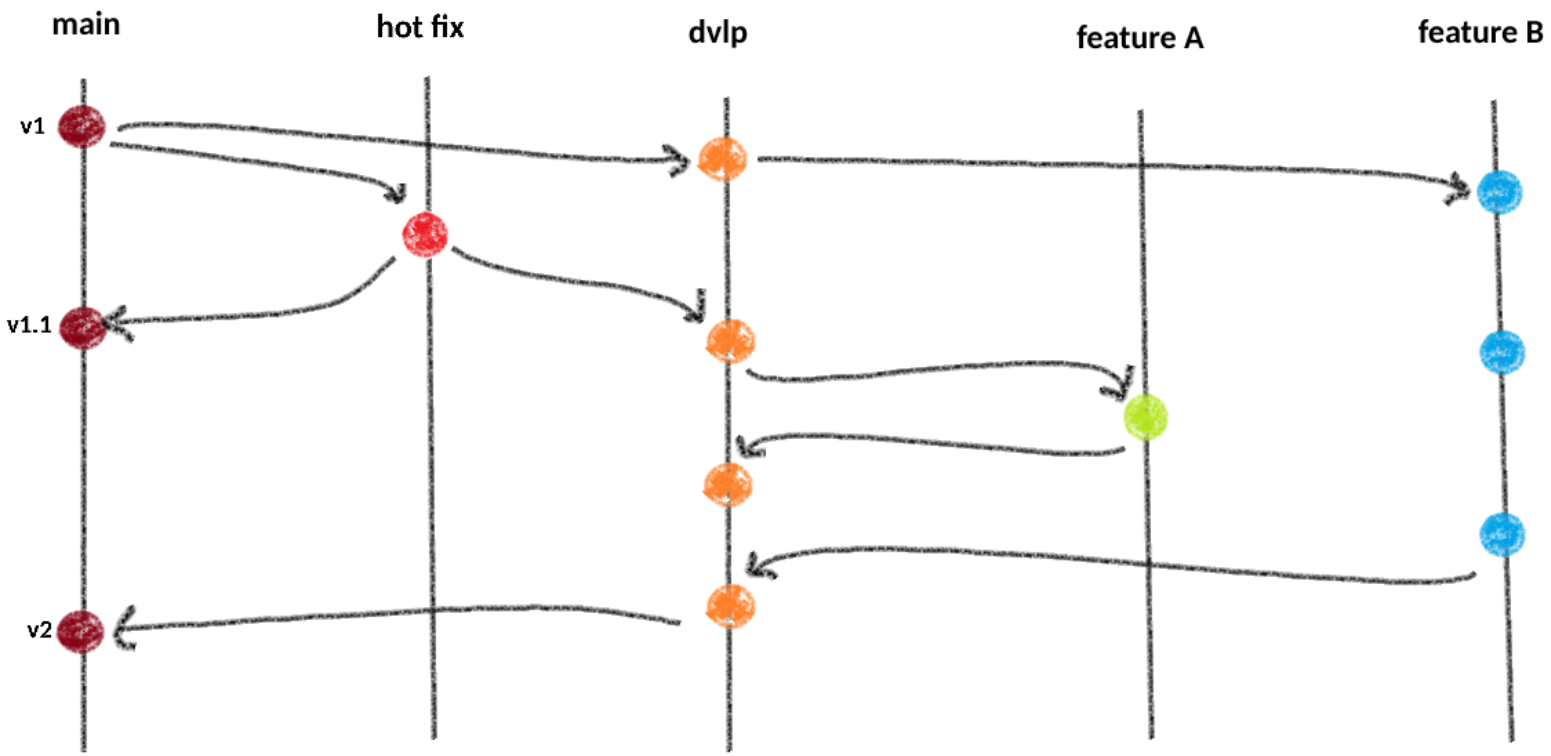


Git point de vue composants - Exemple de messages



Gitflow

Gestion du projet par branches, à chacune sa fonction



Outils de build

Objectif

Packager le code : mettre ensemble tout ce qu'il faut pour que le code soit utilisable.

- Compilation, tests, ...
- Gestion des dépendances, gestion d'environnement

Exemple

Pour Java :

`gradle, maven, ...`

CI / CD

Continuous Integration

Continuous Deployment

Intégration continue

Comment ?

- Compilation et tests sur machine distante *de façon récurrente*

Pourquoi ?

- Réduire temps des cycles de dévelopement
- Déléguer des tâches du développeur à un outil
- Détecter les erreurs *vite* !

Déploiement continu

Comment ?

- Faire tourner le code sur une autre machine
  - → analyse de qualité

Défi

- Avoir une machine à disposition, proche du système final
  - hardware
  - software

Exemple

Difficultés pour tester un serveur de mail

DevOps - Pourquoi ?

Historiquement :

- Développeurs codent
- Opérationnels monitorent

$\Rightarrow$ Volonté d'améliorer la collaboration entre ces équipes pour augmenter le niveau de connaissance globale du projet et accélérer la livraison.

DevOps - Comment ?

- Création de postes avec des points de vue plus globaux : + de vue d'ensemble, - de points de friction
- Automatisation : + de rapidité
- CI/CD : mise à jour plus fréquentes et plus fiables
- Remise en question du projet et du fonctionnement de l'équipe en continu