

# Introduction au CI/CD

ENSG - Décembre 2021

- Présentation disponible à l'adresse: <https://cicd-lectures.github.io/slides/2021>
- Version PDF de la présentation :  Cliquez ici
- This work is licensed under a Creative Commons Attribution 4.0 International License
- Code source de la présentation:  <https://github.com/cicd-lectures/slides>



# Comment utiliser cette présentation ?





- Pour naviguer, utilisez les flèches en bas à droite (ou celles de votre clavier)
  - Gauche/Droite: changer de chapitre
  - Haut/Bas: naviguer dans un chapitre
- Pour avoir une vue globale : utiliser la touche "o" (pour "**O**verview")
- Pour voir les notes de l'auteur : utilisez la touche "s" (pour "**S**peaker notes")

# Bonjour !

# Damien DUPORTAL

- Señor 🌮 Software Engineer chez CloudBees sur le projet Jenkins ☐☐☐
- Freelancer
- Me contacter :
  - ✉ damien.duportal <chez> gmail.com
  - in Damien Duportal
  - 🐦 @DamienDuportal

# Julien LEVESY

- Senior Software Engineer @ Upfluence
- Me contacter :
  -  jlevesy <chez> gmail.com
  -  Julien Levesy
  -  @jlevesy
  -  @jlevesy

Et vous ?



# A propos du cours

- On a essayé de s'adapter à la situation et avons essayé de faire quelque chose d'interactif
- Il y aura donc une alternance de théorie et de pratique
- C'est la première fois qu'on le donne, il risque d'y avoir des soucis, be kind :-)
  - N'hésitez pas à ouvrir des PRs si vous en voyez [ici](#) (🙄 wink wink)

# Outils Nécessaires

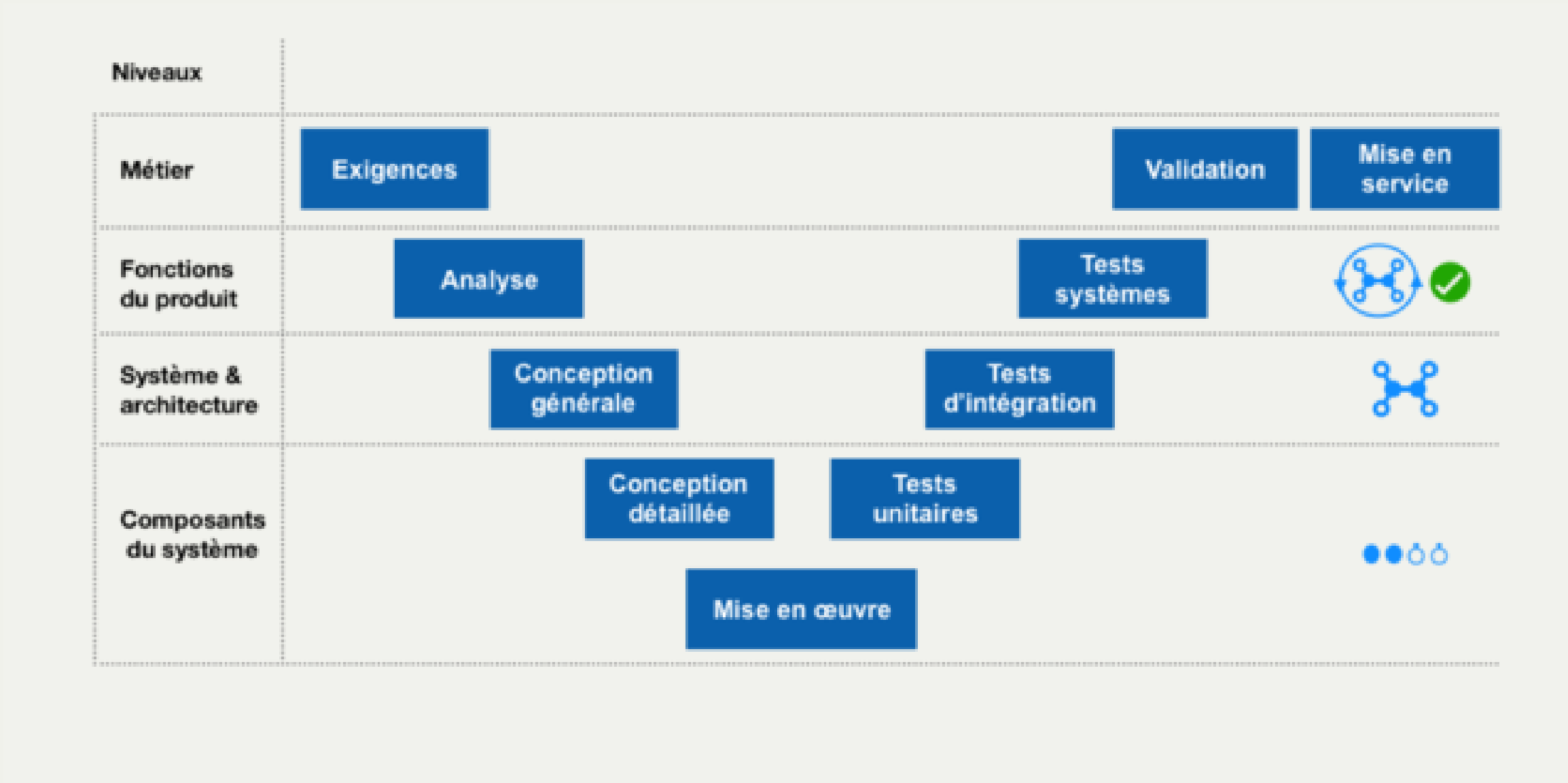
- Un navigateur récent (et décent)
- Un compte [GitHub](#)
- Un compte [GitPod.io](#), notre environnement de développement
- On va vous demander de travailler en binôme, commencez à réfléchir avec qui vous souhaitez travailler !



# Une petite histoire du génie logiciel

# Comment mener un projet logiciel?

# Avant : le cycle en V



# Que peut-il mal se passer?

- On specifie et l'on engage un volume conséquent de travail sur des hypothèses
  - ... et si les hypothèses sont fausses?
  - ... et si les besoins changent?
- Cycle trèèèèès long
  - Aucune validation à court terme
  - Coût de l'erreur décuplé

# Comment éviter ça?

- Valider les hypothèses au plus tôt, et étendre petit à petit le périmètre fonctionnel.
  - Réduire le périmètre fonctionnel au minimum.
  - Confronter le logiciel au plus tôt aux utilisateurs.
  - Refaire des hypothèses basées sur ce que l'on a appris, et recommencer!
- "Embrasser" le changement
  - Votre logiciel va changer en **continu**

# La clé : gérer le changement!

- Le changement ne doit pas être un événement, ça doit être la norme.
- Notre objectif : minimiser le coût du changement.
- Faire en sorte que:
  - Changer quelque chose soit facile
  - Changer quelque chose soit rapide
  - Changer quelque chose ne casse pas tout

# Heureusement, vous avez des outils à disposition!

Et c'est ce que l'on va voir ensemble aujourd'hui!

# Préparer votre environnement de développement



# TODO

# Les fondamentaux de git

# Tracer le changement dans le code

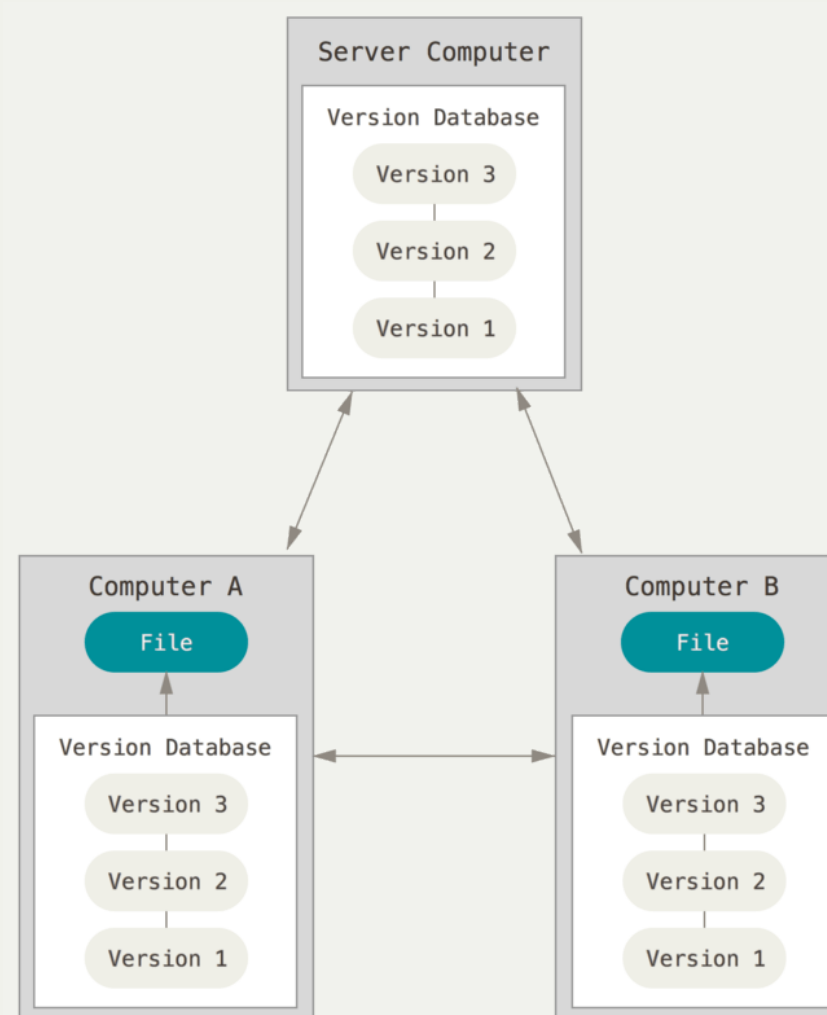
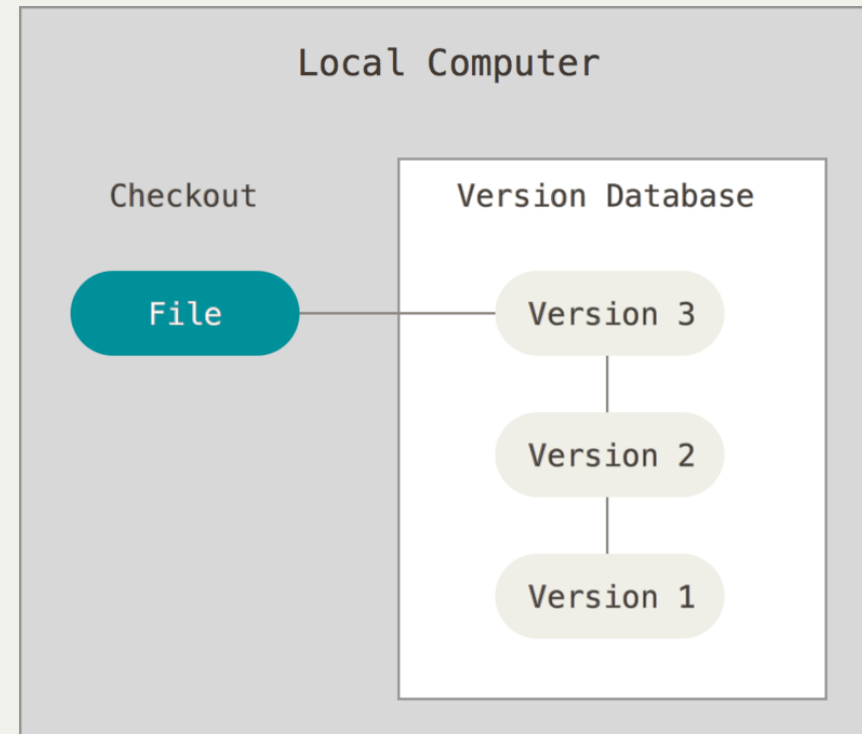
avec un **VCS** :  Version Control System

également connu sous le nom de SCM ( Source Code Management)

# Pourquoi un VCS ?

- Pour conserver une trace de **tous** les changements dans un historique
- Pour **collaborer** efficacement sur un même référentiel de code source

# Concepts des VCS



Source : <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

# Quel VCS utiliser ?



## Nous allons utiliser **Git**

# Git

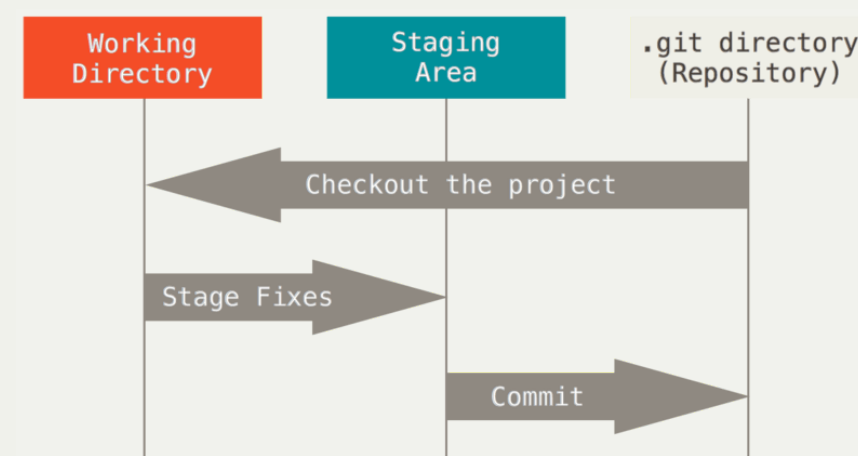
*Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.*

<https://git-scm.com/>



# Les 3 états avec Git

- L'historique ("Version Database") : dossier `.git`
- Dossier de votre projet ("Working Directory") - Commande
- La zone d'index ("Staging Area")



Source : [https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#les\\_trois\\_%C3%A9tats](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#les_trois_%C3%A9tats)



# Exercice avec Git - 1.1

- Rendez vous dans le répertoire `/workspace` (`cd /workspace`)
- Créez un dossier vide nommé `projet-vcs-1` puis positionnez-vous dans ce dossier

```
mkdir -p ./projet-vcs-1/ && cd ./projet-vcs-1/
```

[Copy](#)

- Est-ce qu'il y a un dossier `.git/` ?
  - Essayez la commande `git status` ?
- Initialisez le dépôt git avec `git init`
  - Est-ce qu'il y a un dossier `.git/` ?
  - Essayez la commande `git status` ?

# Solution de l'exercice avec Git - 1.1

```
cd /workspace
mkdir -p ./projet-vcs-1/
cd ./projet-vcs-1/
ls -la # Pas de dossier .git
git status # Erreur "fatal: not a git repository"
git init ./
ls -la # On a un dossier .git
git status # Succès avec un message "On branch master No commits yet"
```

[Copy](#)

# Exercice avec Git - 1.2

- Créez un fichier README .md dedans avec un titre et vos nom et prénoms
  - Essayez la commande `git status` ?
- Ajoutez le fichier à la zone d'indexation à l'aide de la commande `git add (...)`
  - Essayez la commande `git status` ?
- Créez un commit qui ajoute le fichier README .md avec un message,  
à l'aide de la commande `git commit -m <message>`
  - Essayez la commande `git status` ?

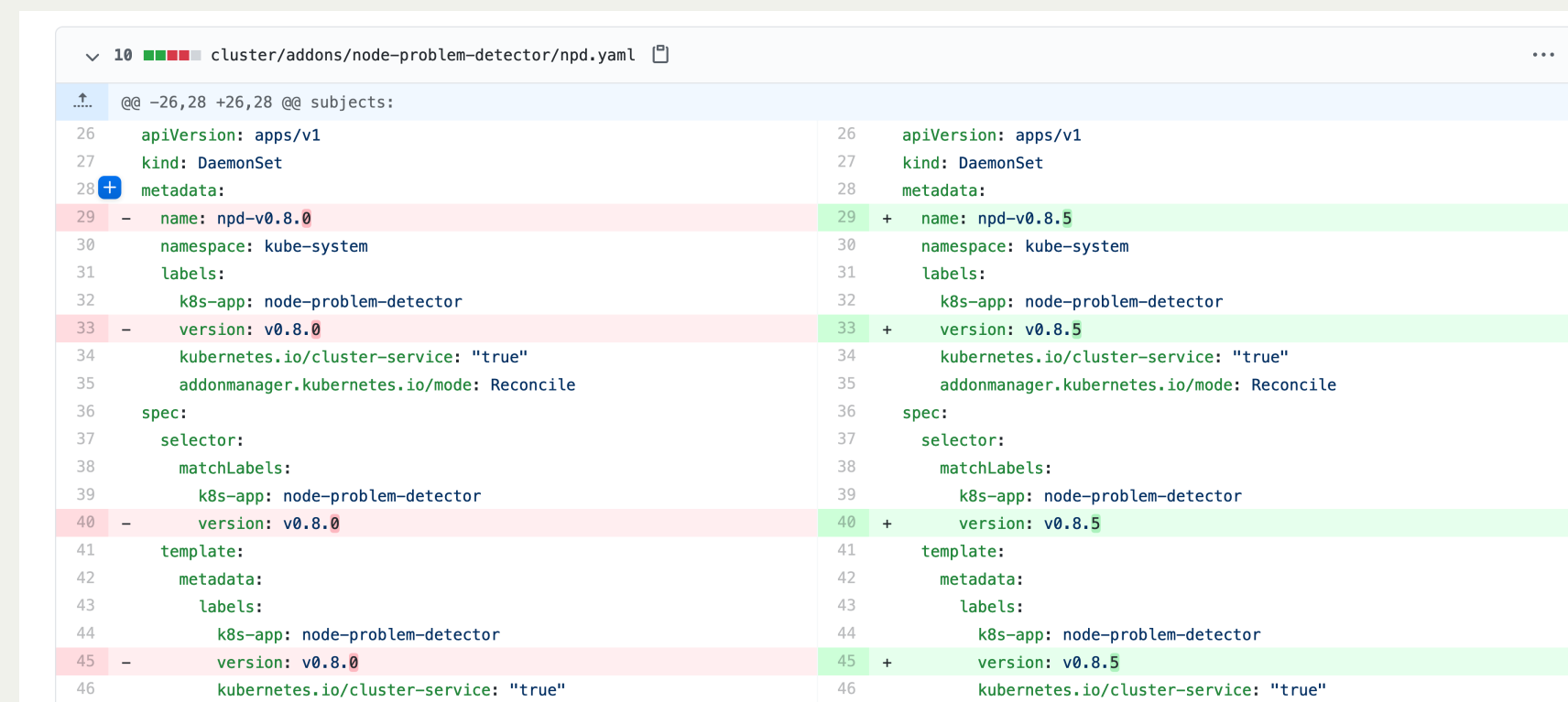
# Solution de l'exercice avec Git - 1.2

```
echo "# Read Me\n\nObi Wan" > ./README.md  
git status # Message "Untracked file"  
  
git add ./README.md  
git status # Message "Changes to be committed"  
git commit -m "Ajout du README au projet"  
git status # Message "nothing to commit, working tree clean"
```

[Copy](#)

# Terminologie de Git - Diff et changeset

**diff:** un ensemble de lignes "changées" sur un fichier donné

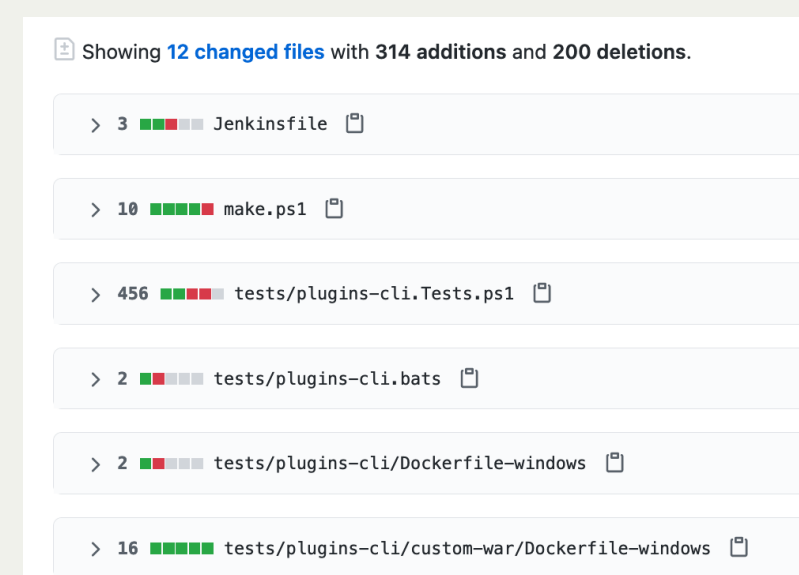


The screenshot shows a side-by-side comparison of a file named `cluster/addons/node-problem-detector/npd.yaml`. The left column represents the original file, and the right column represents the modified version. Lines are color-coded: green for additions, red for deletions, and white for unchanged content. The diff shows updates to the `name` field from `npd-v0.8.0` to `npd-v0.8.5`, the `version` field from `v0.8.0` to `v0.8.5`, and the `version` field in the `template.metadata.labels` section from `v0.8.0` to `v0.8.5`.

```
@@ -26,28 +26,28 @@ subjects:
26  apiVersion: apps/v1
27  kind: DaemonSet
28  metadata:
29  -   name: npd-v0.8.0
30    namespace: kube-system
31    labels:
32      k8s-app: node-problem-detector
33  -   version: v0.8.0
34    kubernetes.io/cluster-service: "true"
35    addonmanager.kubernetes.io/mode: Reconcile
36  spec:
37    selector:
38      matchLabels:
39        k8s-app: node-problem-detector
40  -   version: v0.8.0
41  template:
42    metadata:
43      labels:
44        k8s-app: node-problem-detector
45  -   version: v0.8.0
46    kubernetes.io/cluster-service: "true"

+   name: npd-v0.8.5
+   version: v0.8.5
+   version: v0.8.5
```

**changeset:** un ensemble de "diff" (donc peut couvrir plusieurs fichiers)



The screenshot shows a list of files that have been changed in a commit. The header indicates "Showing 12 changed files with 314 additions and 200 deletions." The list includes:

- > 3 Jenkinsfile
- > 10 make.ps1
- > 456 tests/plugins-cli.Tests.ps1
- > 2 tests/plugins-cli.bats
- > 2 tests/plugins-cli/Dockerfile-windows
- > 16 tests/plugins-cli/custom-war/Dockerfile-windows

# Terminologie de Git - Commit

**commit:** un changeset qui possède un (commit) parent, associé à un message

✓ Bump node-problem-detector to v0.8.5

Browse files

tosi3k committed 2 days ago

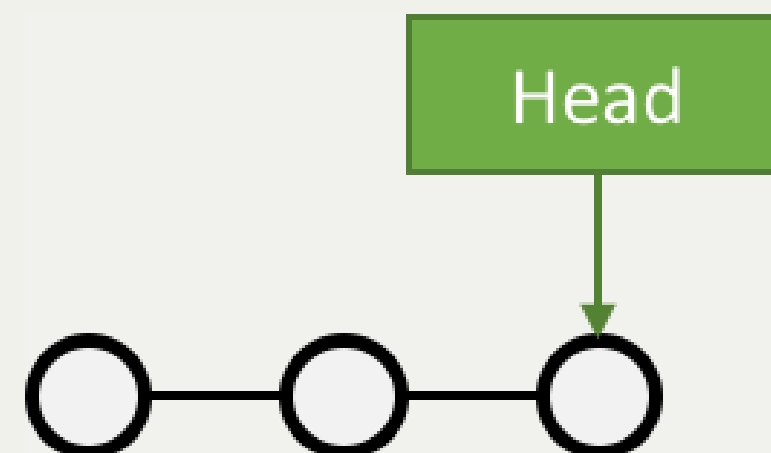
1 parent [e64ebe0](#) commit [8f2dd3aaab02c3b4c1c8233aa5f93bb439f23228](#)

Showing **3 changed files** with **8 additions** and **8 deletions**.

Unified Split

*"HEAD"*: C'est le dernier commit dans l'historique

○ : a commit



# Exercice avec Git - 2

- Afficher la liste des commits
- Afficher le changeset associé à un commit
- Modifier du contenu dans README .md et afficher le diff
- Annulez ce changement sur README .md

# Solution de l'exercice avec Git - 2

```
git log

git show # Show the "HEAD" commit
echo "# Read Me\n\nObi Wan Kenobi" > ./README.md

git diff
git status

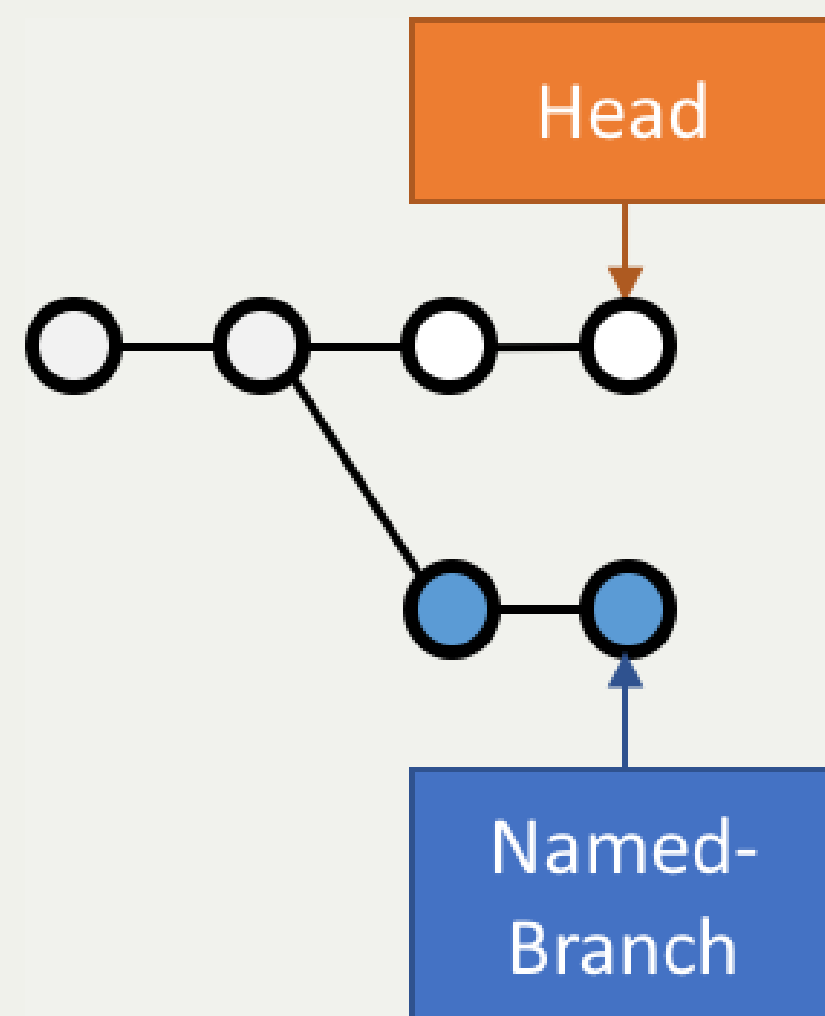
git checkout -- README.md
git status
```

[Copy](#)



# Terminologie de Git - Branche

- Abstraction d'une version "isolée" du code
- Concrètement, une **branche** est un alias pointant vers un "commit"



# Exercice avec Git - 3

- Créer une branche nommée `feature/html`
- Ajouter un nouveau commit contenant un nouveau fichier `index.html` sur cette branche
- Afficher le graphe correspondant à cette branche avec `git log --graph`

# Solution de l'exercice avec Git - 3

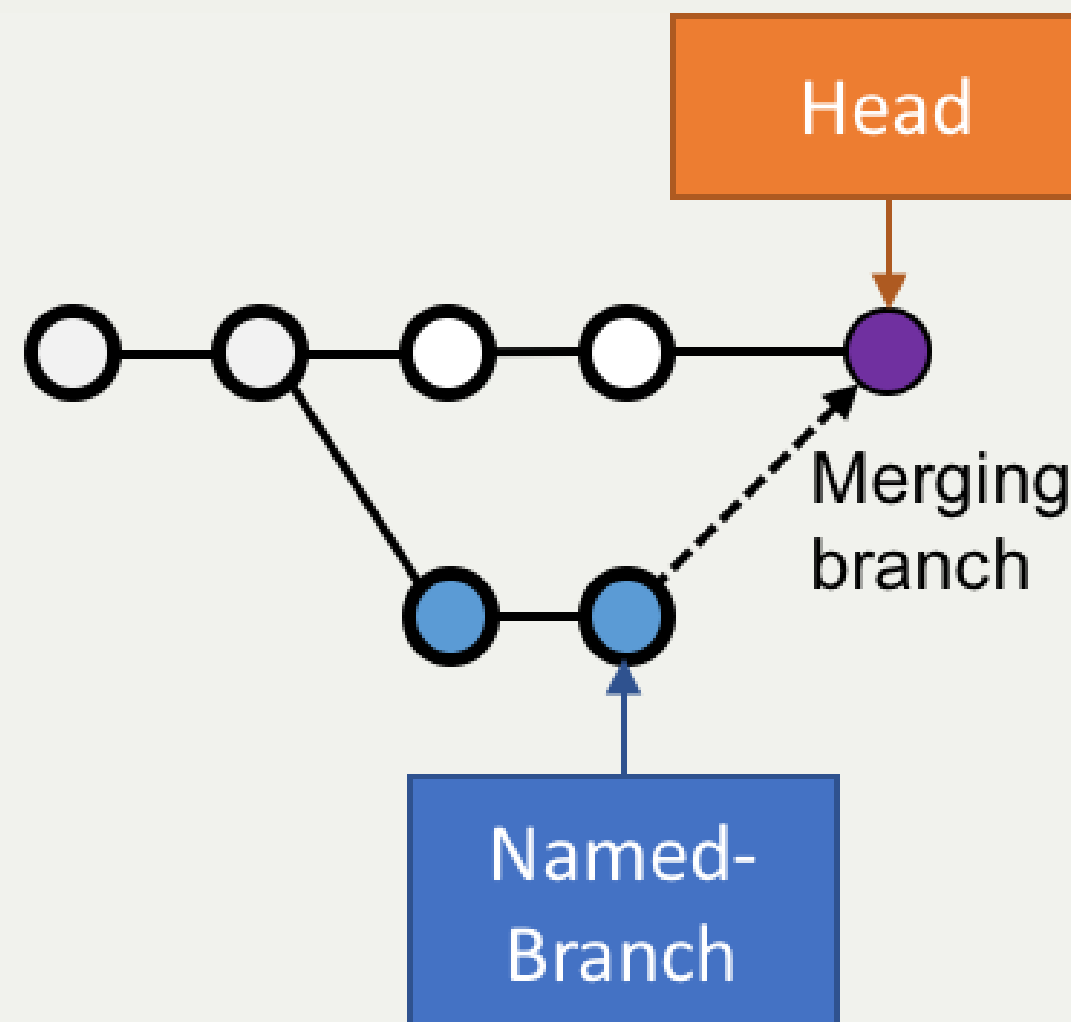
```
git branch feature/html && git checkout feature/html
# Ou git checkout -b feature/html
echo '<h1>Hello</h1>' > ./index.html
git add ./index.html && git commit -m "Ajout d'une page HTML par défaut"

git log --graph
# git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset '
```

Copy

# Terminologie de Git - Merge

- On intègre une branche dans une autre en effectuant un **merge**
  - Un nouveau commit est créé, fruit de la combinaison de 2 autres commits



# Exercice avec Git - 4

- Merger la branche `feature/html` dans la branche principale
  - ⚠ Pensez à utiliser l'option `--no-ff`
- Afficher le graphe correspondant à cette branche avec `git log --graph`

# Solution de l'exercice avec Git - 4

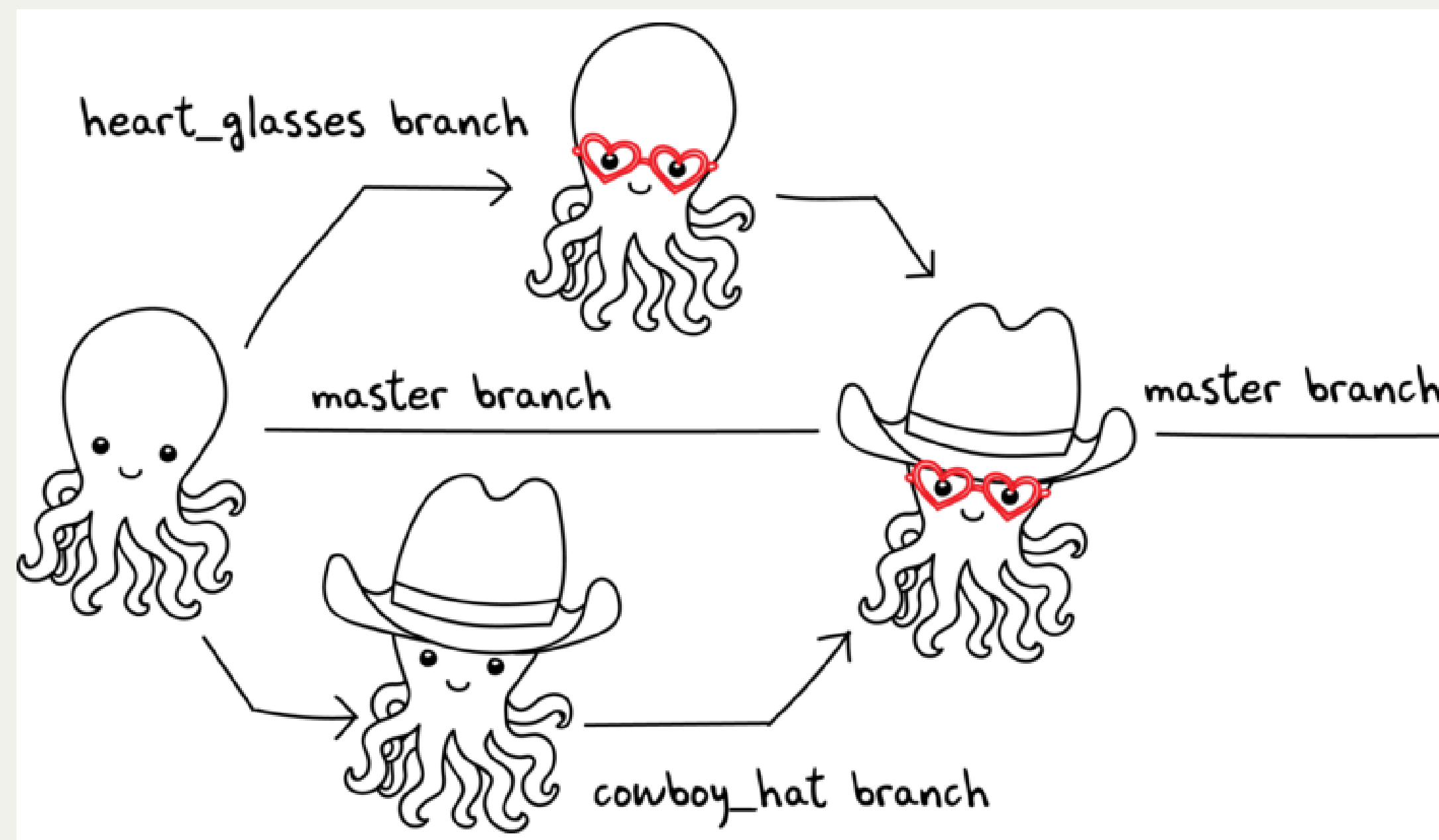
```
git checkout master  
git merge --no-ff feature/html  
git log --graph
```

Copy

```
# git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset'
```

# Feature Branch Flow

- **Une seule** branche **par** fonctionnalité



# Exemple d'usages de VCS

- "Infrastructure as Code" :
  - Besoins de traçabilité, de définition explicite et de gestion de conflits
  - Collaboration requise pour chaque changement (revue, responsabilités)
- Code Civil:
  - <https://github.com/steeve/france.code-civil>
  - <https://github.com/steeve/france.code-civil/pull/40>
  - <https://github.com/steeve/france.code-civil/commit/b805ecf05a86162d149d3d182e04074ecf72c066>



# Pour aller plus loin avec Git et les VCS...

Un peu de lecture :

- <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
- <http://martinfowler.com/bliki/VersionControlTools.html>
- <http://martinfowler.com/bliki/FeatureBranch.html>
- <https://about.gitlab.com/2014/09/29/gitlab-flow/>
- <https://www.atlassian.com/git/tutorials/comparing-workflows>
- <http://nvie.com/posts/a-successful-git-branching-model/>

# Présentation de votre projet

# TODO

# Cycle de vie de votre projet

# TODO

# Mettre son code en sécurité

# TODO

# Intégration continue (CI)



# TODO

# Git à plusieurs

# TODO

# Tests Automatisés

# TODO

# Versions

# TODO

# Mettre en production



TODO

# Conclusion

# TODO

# Merci !

- ✉ damien.duportal+pro <chez> gmail.com
- 🐦 @DamienDuportal
- ✉ jlevesy <chez> gmail.com
- 🐦 @jlevesy

Slides: <https://cicd-lectures.github.io/slides/2021>



Source on : <https://github.com/cicd-lectures/slides>