

# Orchestration de conteneurs et CI/CD

 ESGI Lyon - 2022/2023



- Présentation disponible à l'adresse: <https://dduportal.github.io/esgi-containers-ci-cd/2023>
- Version PDF de la présentation :  Cliquez ici
- Contenu sous licence Creative Commons Attribution 4.0 International License
  - Une partie du contenu provient de <https://github.com/cicd-lectures/slides> (auteurs: Julien Levesy et moi-même)
- Code source de la présentation:  <https://github.com/dduportal/esgi-containers-ci-cd>

# 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 "**Overview**")

# Bonjour !

La suite: vers le bas ↓

# Damien DUPORTAL

- Staff Software Engineer chez CloudBees pour le projet Jenkins 2011
- Freelancer
- Me contacter :
  -  damien.duportal <chez> gmail.com
  -  dduortal
  -  Damien Duportal
  -  @DamienDuportal

Et vous ?



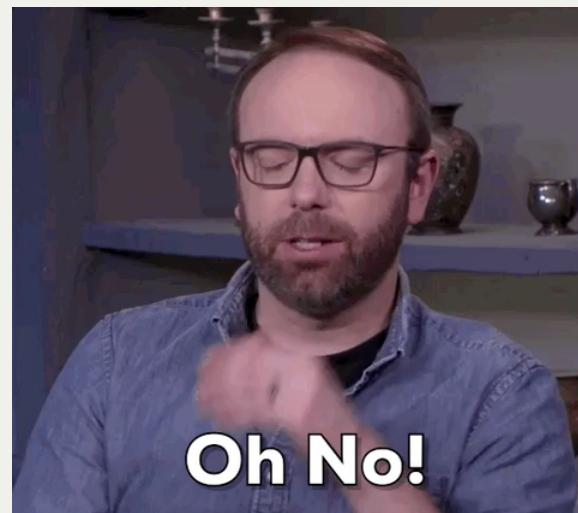
# A propos du cours

- Alternance de théorie et de pratique pour être le plus interactif possible
- Compatible Distanciel / Présentiel
- Contenu entièrement libre et open-source
  - N'hésitez pas ouvrir des Pull Request si vous voyez des améliorations ou problèmes: sur cette page (😉 wink wink)

# Calendrier

- **Distanciel**  Mercredi 01 février 2023 - 14h → 17h15
- **Présentiel**  Mardi 28 février 2023 - 14h → 17h15
- **Présentiel**  Mercredi 01 mars 2023 - 08h → 17h16
- **Présentiel**  Jeudi 02 mars 2023 - 08h → 13h
- **Distanciel**  Vendredi 28/04 - 14h → 17h15

# Evaluation



- Pourquoi ? s'assurer que vous avez acquis un minimum de concepts
- Quoi ? Une note sur 20, basée sur une liste de critères exhaustifs
- Comment ? Un projet GitHub (public) à me rendre (timing à déterminer ensemble)

# Plan

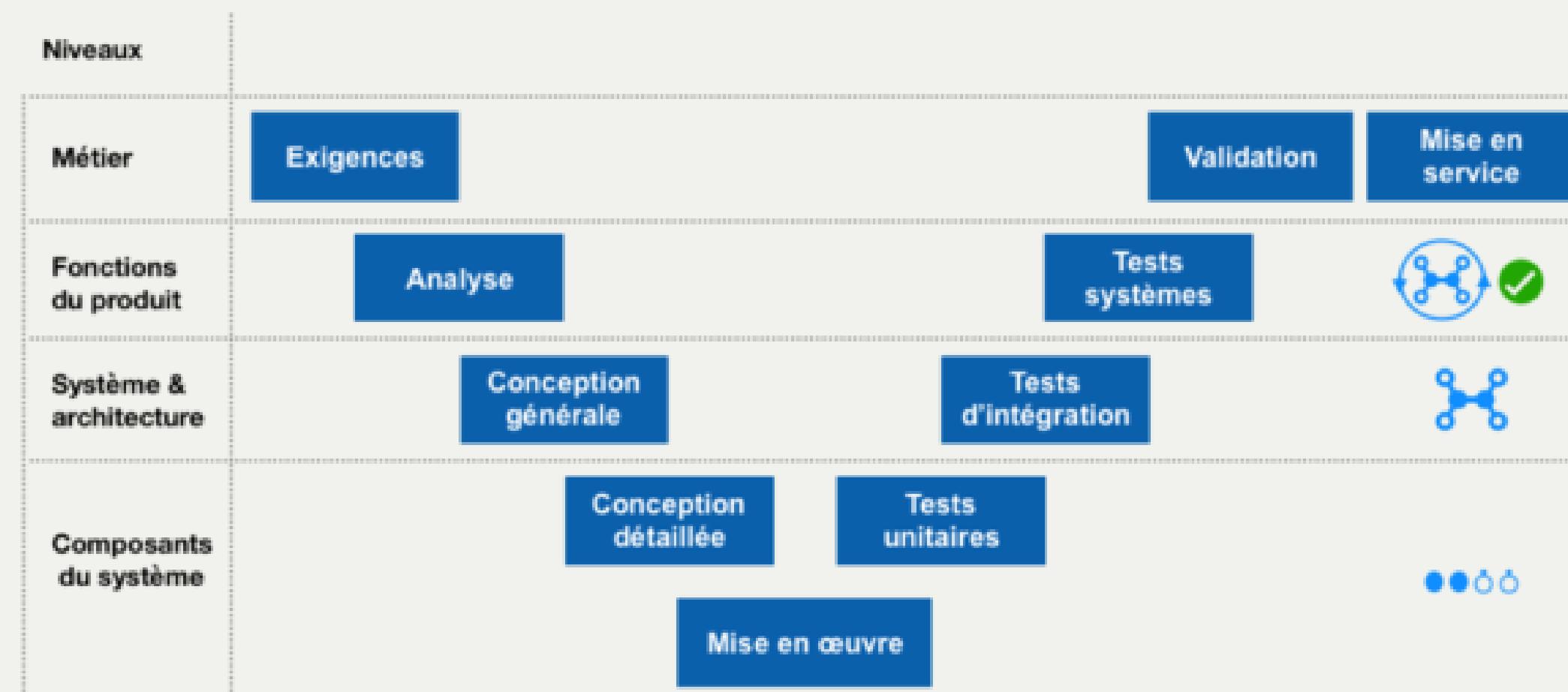


- Introduction
- Rappels (Ligne de commande, Git)
- Intégration Continue (CI)
- Rappels (Docker)
- Docker Avancé
- Orchestration de Containers
- Déploiement Continu (CD)

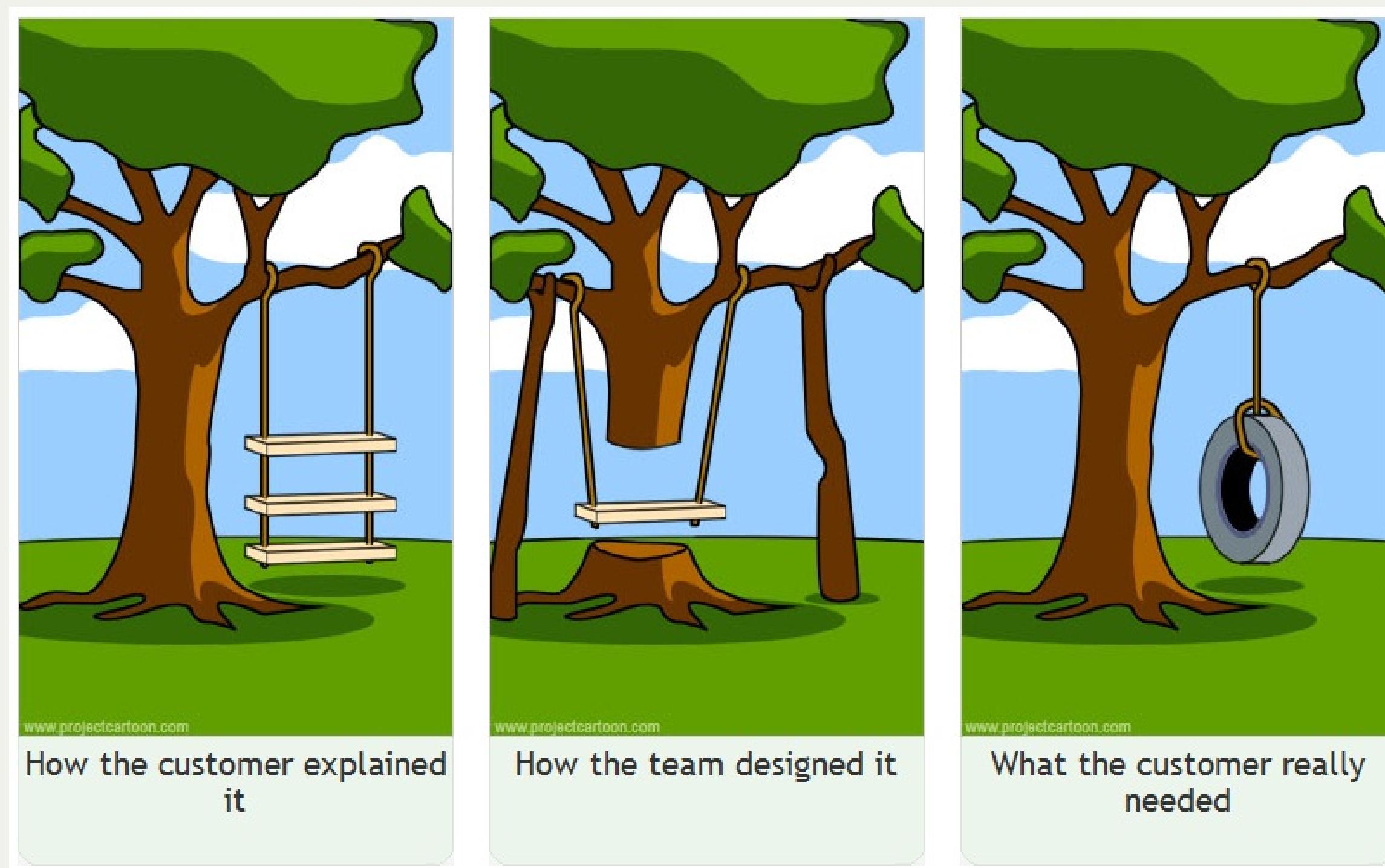
La suite: vers la droite ➔

Vous avez dit "DevOps" ?

# Avant : le cycle en V



# What could go right?



Copyright © <https://www.projectcartoon.com> under the Creative Commons Attribution 3.0 Unported License

# Et dans la vraie vie c'est pire !



Copyright © <https://www.projectcartoon.com> under the Creative Commons Attribution 3.0 Unported License

# Problématique

Travail basé sur les hypothèses de départ + Temps écoulé entre la capture du besoin et la mise à disposition en production

- Le besoin a forcément évolué dans ce laps de temps
- Erreur de capture du besoin == coût de l'erreur décuplé avec le temps

# Agile

- Travailler par petites itérations **complètes**
  - Commencer petit
  - Confronter le logiciel au plus tôt aux utilisateurs.
  - Refaire des hypothèses basées sur ce que l'on à appris, et recommencer !
- "Embrasser" le changement : Votre logiciel va changer en **continu**



Source

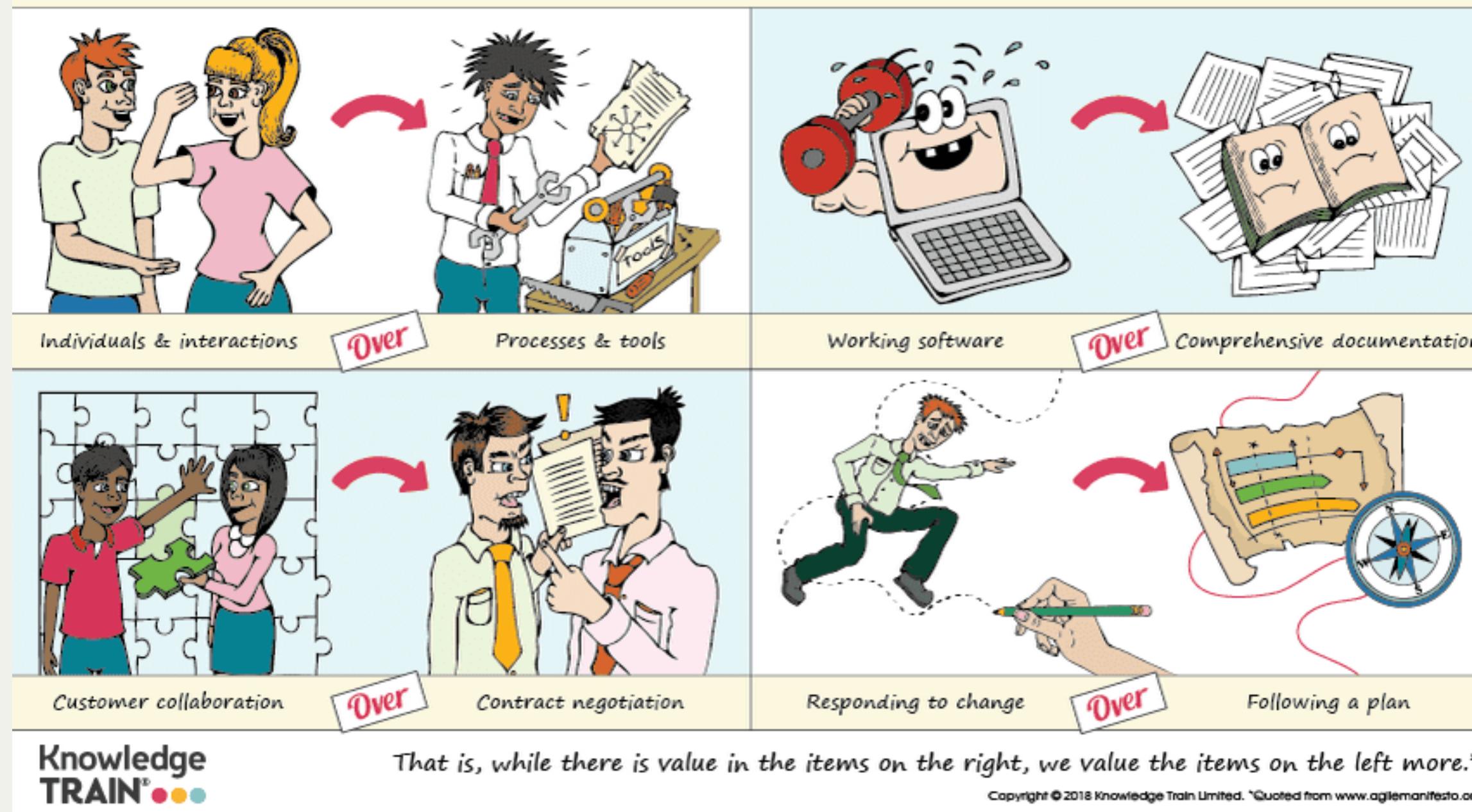
# Manifest Agile

- <https://agilemanifesto.org/iso/fr/manifesto.html>

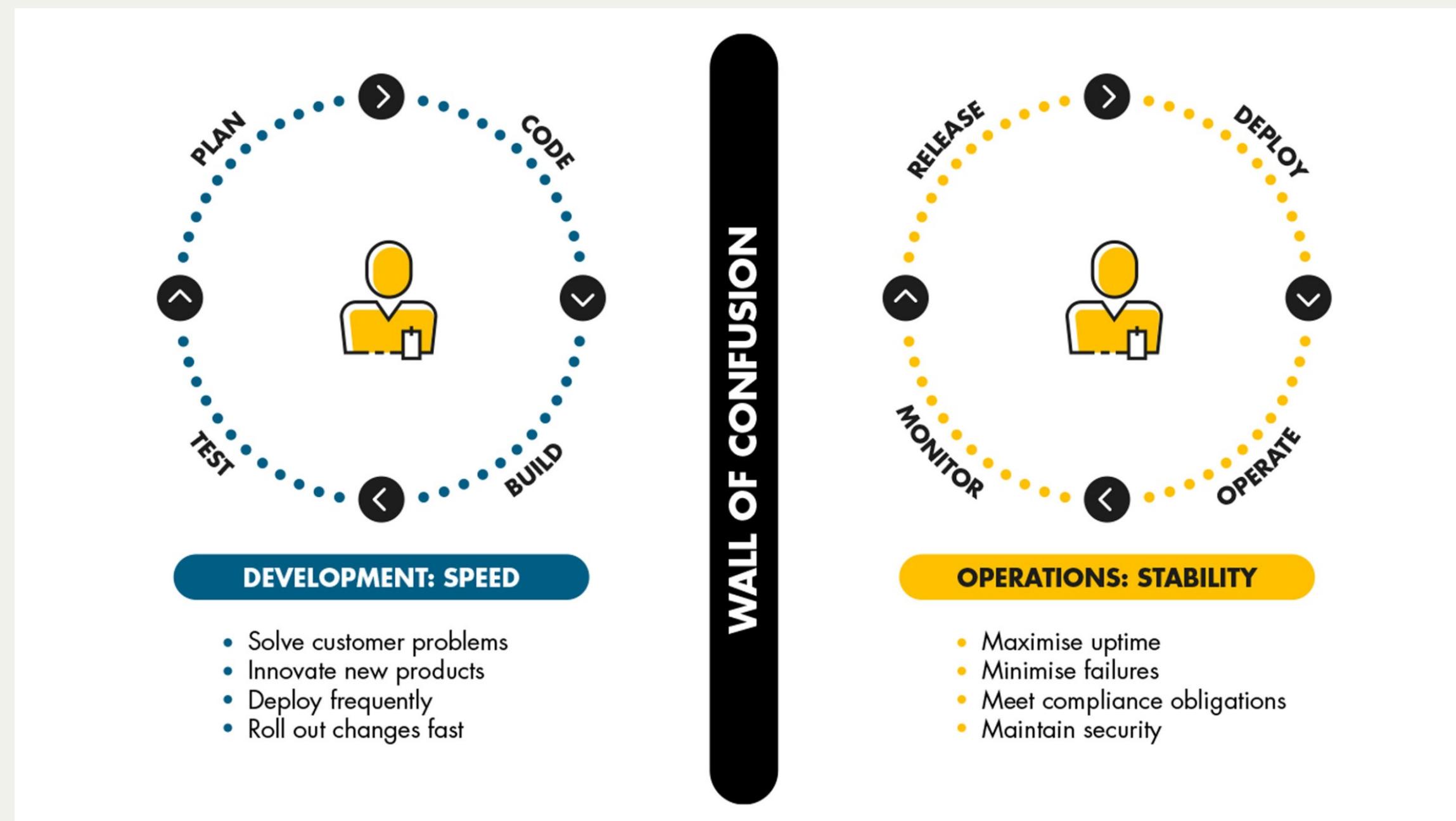
## ***Manifesto for Agile Software Development\****

"We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:



# Le mur de la confusion



Source

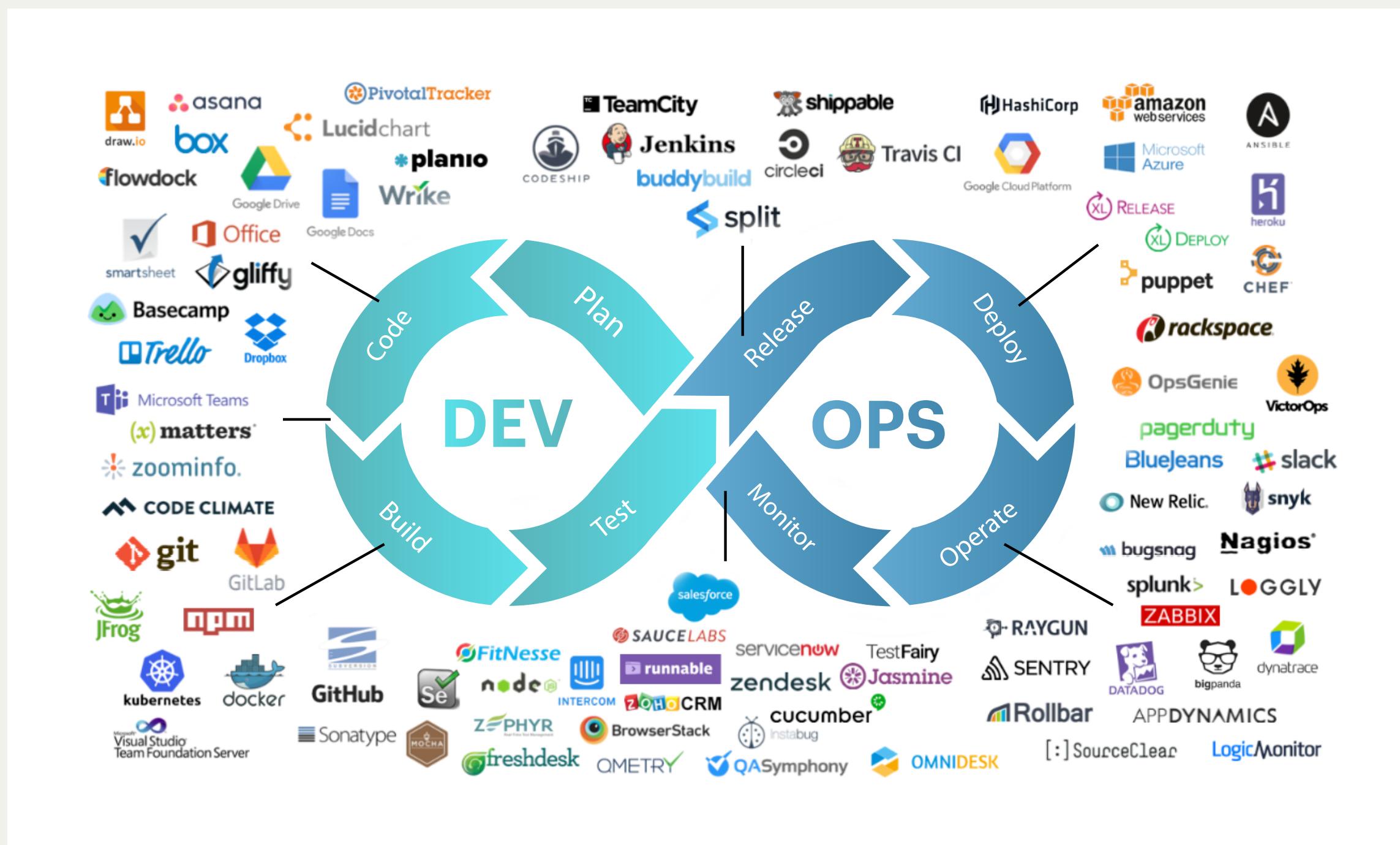
# La clé : gérer le changement!

- Le changement ne doit pas être un événement, ça doit être la **norme**
- Faire en sorte que changer quelque chose soit:
  - Facile
  - Rapide
  - Stable
  - Sûr

# Say Hello to DevOps



Heureusement, vous avez des outils à disposition !



# Préparer votre environnement de travail

# Outils Nécessaires



- Un navigateur web récent (et décent)
- Un compte sur GitHub

# GitPod

GitPod.io : Environnement de développement dans le  "nuage"

- **But:** reproductible
- Puissance de calcul sur un serveur distant
- Éditeur de code VSCode dans le navigateur
- Gratuit pour 50h par mois ()

# Démarrer avec GitPod



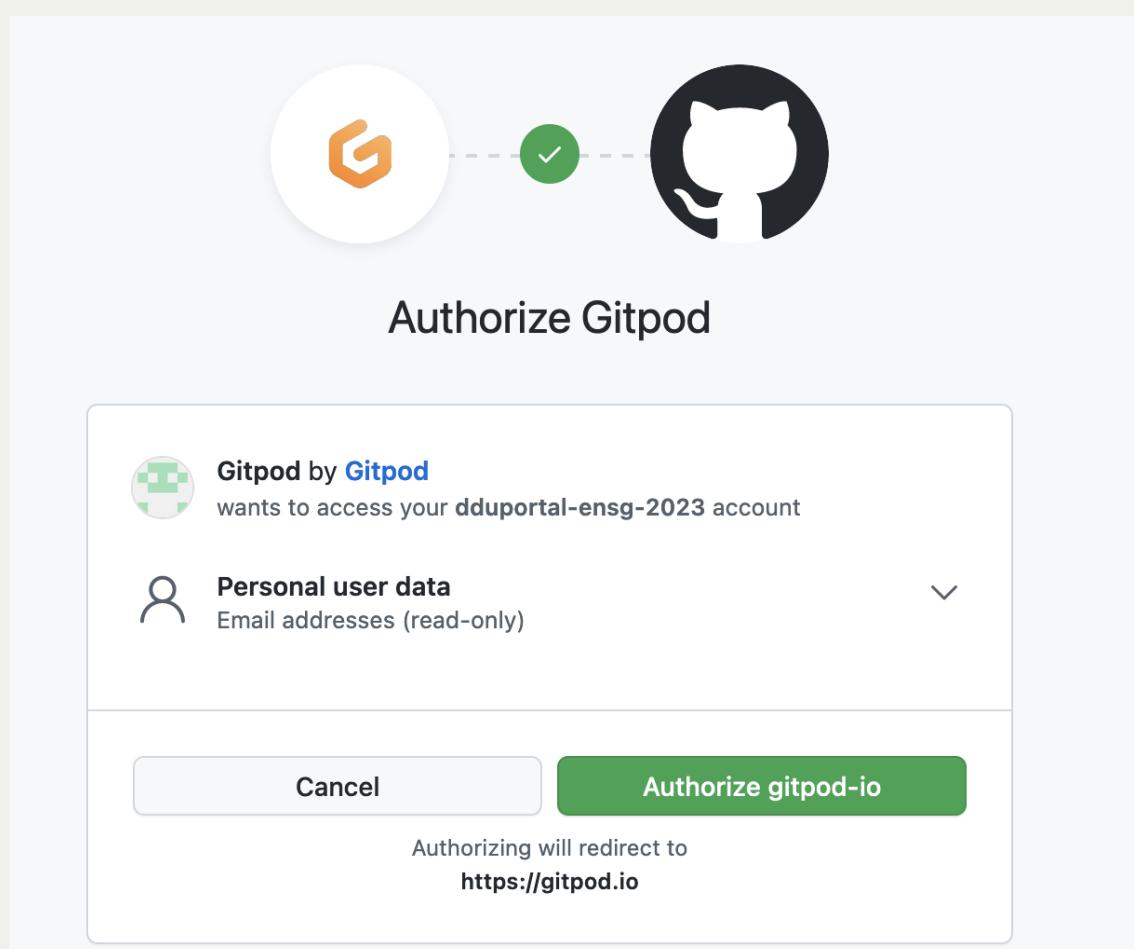
- Rendez vous sur <https://gitpod.io>
- Authentifiez vous en utilisant votre compte GitHub:
  - Bouton "Login" en haut à droite
  - Puis choisissez le lien " Continue with GitHub"

△ Pour les "autorisations", passez sur la slide suivante

# Autorisations demandées par GitPod



Lors de votre première connexion, GitPod va vous demander l'accès (à accepter) à votre email public configuré dans GitHub :



▲ Passez à la slide suivante avant d'aller plus loin

# Validation du Compte GitPod



GitPod vous demande votre numéro de téléphone mobile afin d'éviter les abus (service gratuit).  
Saisissez un numéro de téléphone valide pour recevoir par SMS un code de déblocage :

**User Validation Required**

**⚠ To use Gitpod you'll need to validate your account with your phone number. This is required to discourage and reduce abuse on Gitpod infrastructure.**

Enter a mobile phone number you would like to use to verify your account. Having trouble? [Contact support](#)

Mobile Phone Number

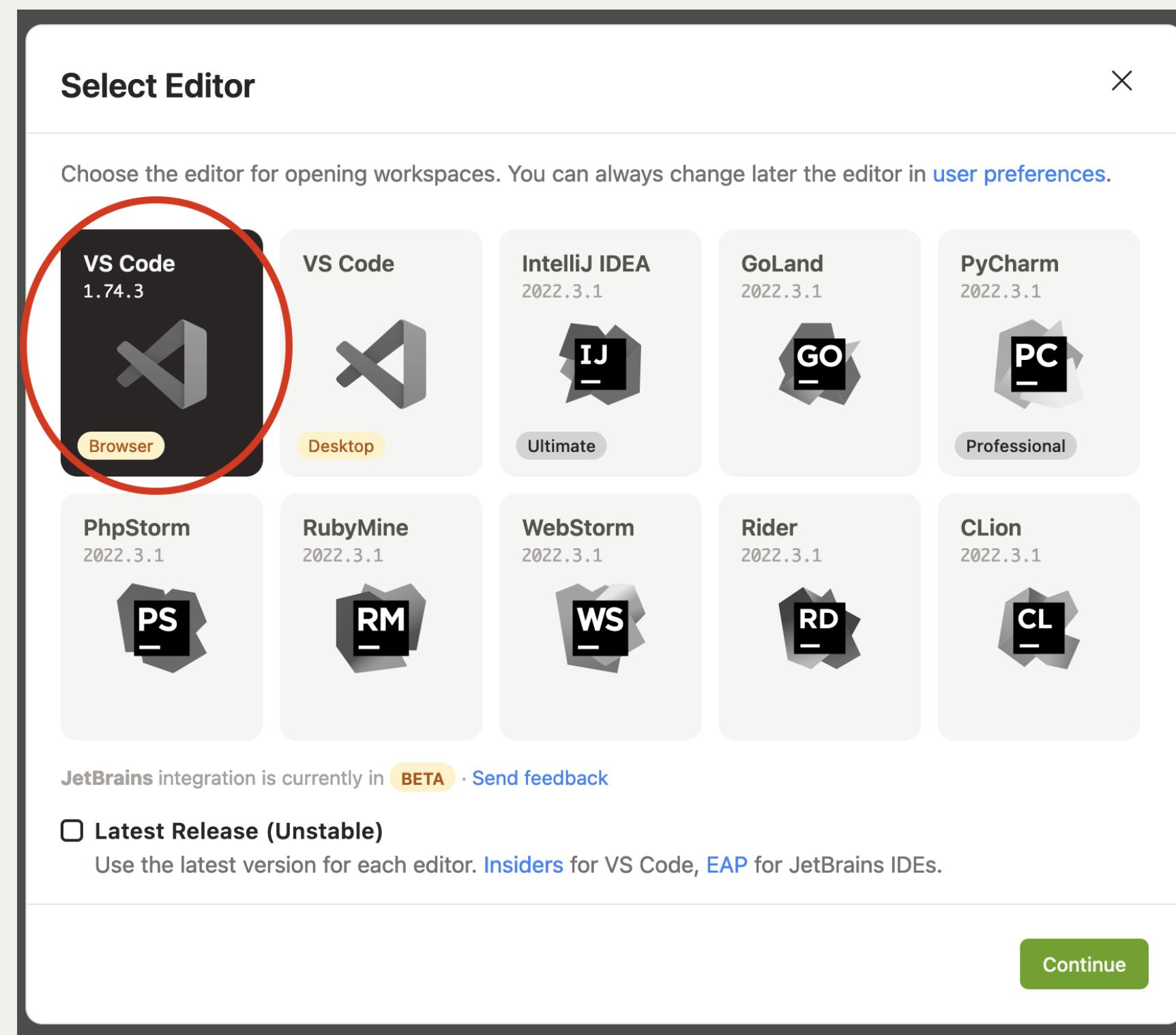
**Send Code via SMS**

▲ Passez à la slide suivante avant d'aller plus loin

# Choix de l'Éditeur de Code



Choisissez l'éditeur "VSCode Browser" (la première tuile) :



▲ Passez à la slide suivante avant d'aller plus loin

# Workspaces GitPod

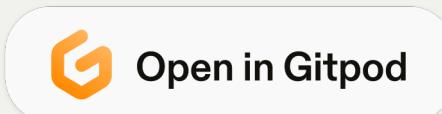


- Vous arrivez sur la page listant les "workspaces" GitPod :
- Un workspace est une instance d'un environnement de travail virtuel (C'est un ordinateur distant)
- ⚠ Faites attention à réutiliser le même workspace tout au long de ce cours⚠

A screenshot of the GitPod web interface. At the top, there's a header with a logo, a 'Workspaces' button, a 'New Team' button, and a 'Feedback' button. Below the header, the title 'Workspaces' is displayed with the subtitle 'Manage recent and stopped workspaces.' A search bar labeled 'Filter Workspaces' and a limit selector 'Limit: 50' are present. A green 'New Workspace' button is located on the right. The main area shows two workspace entries: one named 'cicdlectures-gitpod-jcu32jcv4q2' which is active (indicated by a green dot) and another named 'cicdlectures-gitpod-vv8g7mywidp'. Each entry includes a GitHub URL, the branch name 'main', a status message 'No Changes', and a timestamp ('3 minutes ago' for the first and '4 minutes ago' for the second). Each entry also has a three-dot menu icon on the right.

# Démarrer l'environnement GitPod

Cliquez sur le bouton ci-dessous pour démarrer un environnement GitPod personnalisé:



Après quelques secondes (minutes?), vous avez accès à l'environnement:

- Gauche: navigateur de fichiers ("Workspace")
- Haut: éditeur de texte ("Get Started")
- Bas: Terminal interactif
- À droite en bas: plein de popups à ignorer (ou pas?)

Source disponible dans : <https://github.com/dduportal/esgi-gitpod>

# Checkpoint

- Vous devriez pouvoir taper la commande `whoami` dans le terminal de GitPod:
  - Retour attendu: `gitpod`
- Vous devriez pouvoir fermer le fichier "Get Started" ...
  - ... et ouvrir le fichier `.gitpod.yml`

On peut commencer !

# Ligne de commande

Guide de survie



# Problématique

- Communication Humain <→ Machine
- Base commune de TOUS les outils

# CLI

-  CLI == "Command Line Interface"
-  "Interface de Ligne de Commande"

# REPL

Pour les théoriciens et curieux :

-  REPL == "Read–eval–print loop"

[https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print\\_loop](https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop)

# Anatomie d'une commande

```
ls --color=always -l /bin
```

Copy

- Séparateur : l'espace
- Premier élément (`ls`) : c'est la commande
- Les éléments commençant par un tiret – sont des "options" et/ou drapeaux ("flags")
  - "Option" == "Optionnel"
- Les autres éléments sont des arguments (`/bin`)
  - Nécessaire (par opposition)

# Manuel des commandes

- Afficher le manuel de <commande> :

```
man <commande> # Commande 'man' avec comme argument le nom de ladite commande
```

- Navigation avec les flèches haut et bas
  - Tapez / puis une chaîne de texte pour chercher
  - Touche n pour sauter d'occurrence en occurrence
- Touche q pour quitter



Essayez avec ls, chercher le mot color

- La majorité des commandes fournit également une option (--help), un flag (-h) ou un argument (help)
- Google c'est pratique aussi hein !

# Raccourcis

Dans un terminal Unix/Linux/WSL :

- CTRL + C : Annuler le process ou prompt en cours
- CTRL + L : Nettoyer le terminal
- CTRL + A : Positionner le curseur au début de la ligne
- CTRL + E : Positionner le curseur à la fin de la ligne
- CTRL + R : Rechercher dans l'historique de commandes

🎓 Essayez-les !

# Commandes de base 1/2

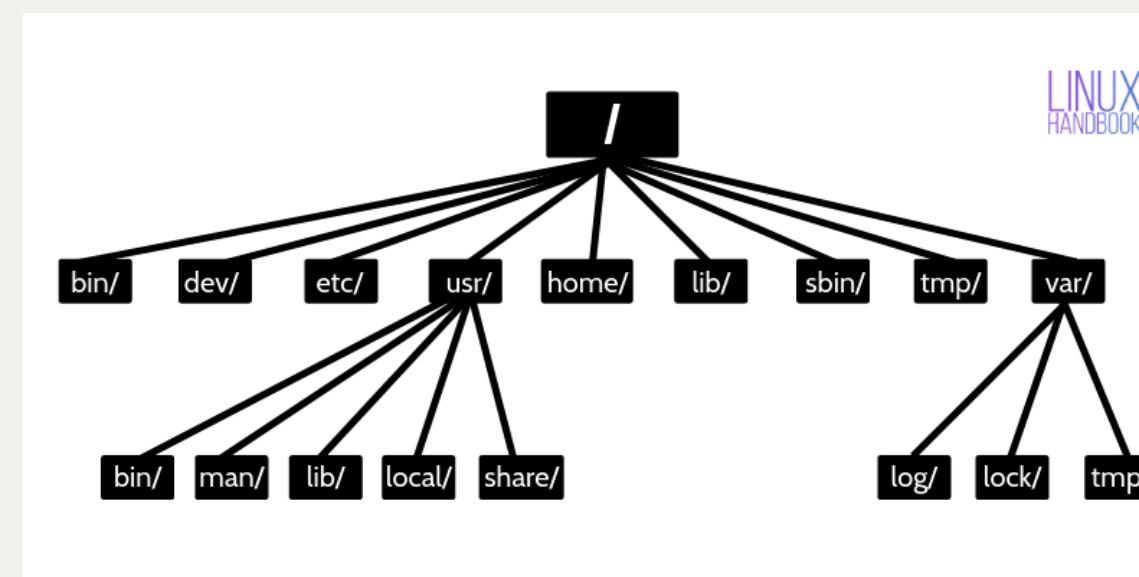
- `pwd` : Afficher le répertoire courant
  -  Option `-P` ?
- `ls` : Lister le contenu du répertoire courant
  -  Options `-a` et `-l` ?
- `cd` : Changer de répertoire
  -  Sans argument : que se passe t'il ?
- `cat` : Afficher le contenu d'un fichier
  -  Essayez avec plusieurs arguments
- `mkdir` : créer un répertoire
  -  Option `-p` ?

# Commandes de base 2/2

- echo : Afficher un (des) message(s)
- rm : Supprimer un fichier ou dossier
- touch : Créer un fichier
- grep : Chercher un motif de texte

# Arborescence de fichiers 1/2

- Le système de fichier a une structure d'arbre
  - La racine du disque dur c'est / : `ls -l /`
  - Le séparateur c'est également / : `ls -l /usr/bin`
- Deux types de chemins :
  - Absolu (depuis la racine): Commence par / (Ex. /usr/bin)
  - Sinon c'est relatif (e.g. depuis le dossier courant) (Ex . /bin ou local/bin/)



Source

# Arborescence de fichiers 2/2

- Le dossier "courant" c'est . : `ls -l ./bin # Dans le dossier /usr`
- Le dossier "parent" c'est .. : `ls -l ../ # Dans le dossier /usr`
- ~ (tilde) c'est un raccourci vers le dossier de l'utilisateur courant : `ls -l ~`
- Sensible à la casse (majuscules/minuscules) et aux espaces : `ls`

```
ls -l /bin  
ls -l /Bin  
mkdir ~/\"Accent tué\"\  
ls -d ~/Accent\ tué
```

Copy

# Un language (?)

- Variables interpolées avec le caractère "dollar" \$ :

```
echo $MA_VARIABLE
echo "$MA_VARIABLE"
echo ${MA_VARIABLE}

# Recommendation
echo "${MA_VARIABLE}"

MA_VARIABLE="Salut tout le monde"

echo "${MA_VARIABLE}"
```

Copy

- Sous commandes avec \$ (<command>) :

```
echo ">> Contenu de /tmp :\n$(ls /tmp)"
```

Copy

- Des if, des for et plein d'autres trucs (<https://tldp.org/LDP/abs/html/>)

# Codes de sortie

- Chaque exécution de commande renvoie un code de retour (🇬🇧 "exit code")
  - Nombre entier entre 0 et 255 (en **POSIX**)
- Code accessible dans la variable **éphémère** \$? :

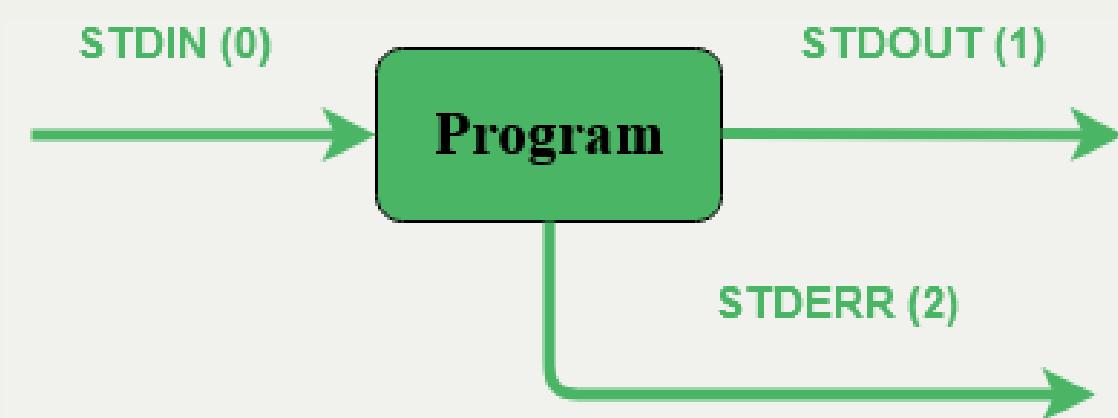
```
ls /tmp
echo $?

ls /do_not_exist
echo $?

# Une seconde fois. Que se passe-t'il ?
echo $?
```

Copy

# Entrée, sortie standard et d'erreur



```
ls -l /tmp
echo "Hello" > /tmp/hello.txt
ls -l /tmp
ls -l /tmp >/dev/null
ls -l /tmp 1>/dev/null

ls -l /do_not_exist
ls -l /do_not_exist 1>/dev/null
ls -l /do_not_exist 2>/dev/null

ls -l /tmp /do_not_exist
ls -l /tmp /do_not_exist 1>/dev/null 2>&1
```

Copy

# Pipelines

- Le caractère "pipe" | permet de chaîner des commandes
  - Le "stdout" de la première commande est branchée sur le "stdin" de la seconde
- Exemple : Afficher les fichiers/dossiers contenant le lettre d dans le dossier /bin :

```
ls -l /bin  
ls -l /bin | grep "d" --color=auto
```

Copy

# Exécution 1/2

- Les commandes sont des fichiers binaires exécutables sur le système :

```
command -v cat # équivalent de "which cat"  
ls -l "$(command -v cat)"
```

Copy

- La variable d'environnement \$PATH liste les dossiers dans lesquels chercher les binaires
  -  Utiliser cette variable quand une commande fraîchement installée n'est pas trouvée

# Exécution 2/2

- Exécution de scripts :
  - Soit appel direct avec l'interpréteur : sh ~/monscript.txt
  - Soit droit d'exécution avec un "shebang" (e.g. #!/bin/bash)

```
$ chmod +x ./monscript.sh
$ head -n1 ./monscript.sh
#!/bin/bash

$ ./monscript.sh
# Exécution
```

Copy

# Git

Guide de survie



# Problématique

- Support de communication
  - Humain → Machine
  - Humain <→ Humain
- Besoins de traçabilité, de définition explicite et de gestion de conflits
  - Collaboration requise pour chaque changement (revue, responsabilités)

# Tracer le changement dans le code

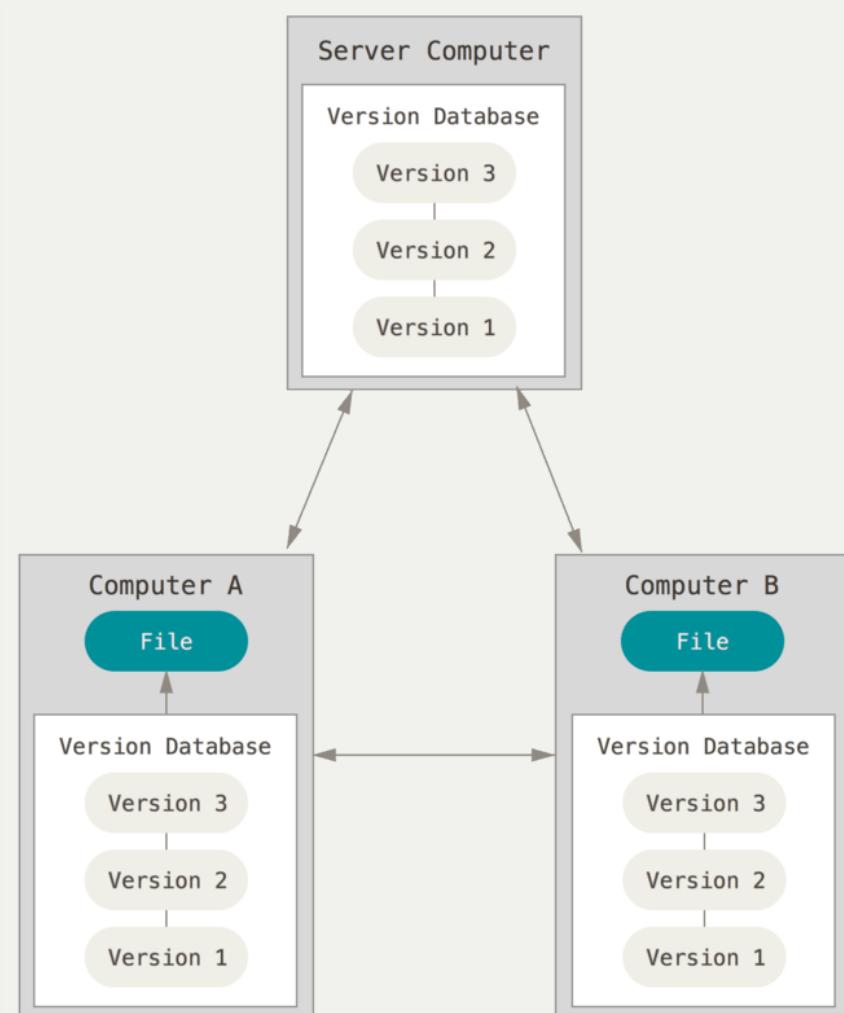
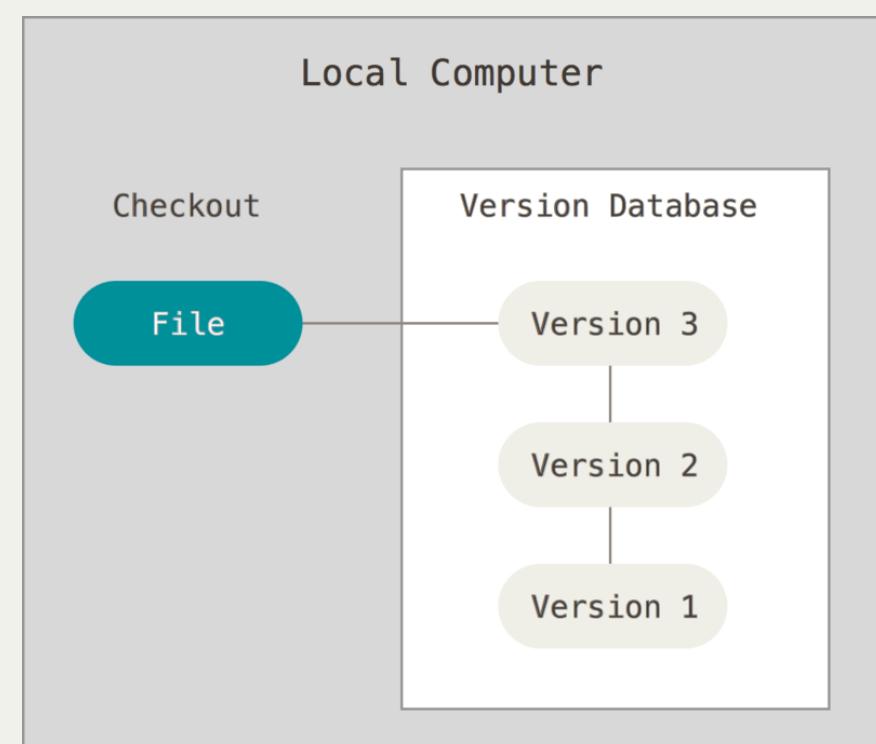
avec un **VCS** :  Version Control System

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

# Pourquoi un VCS ?

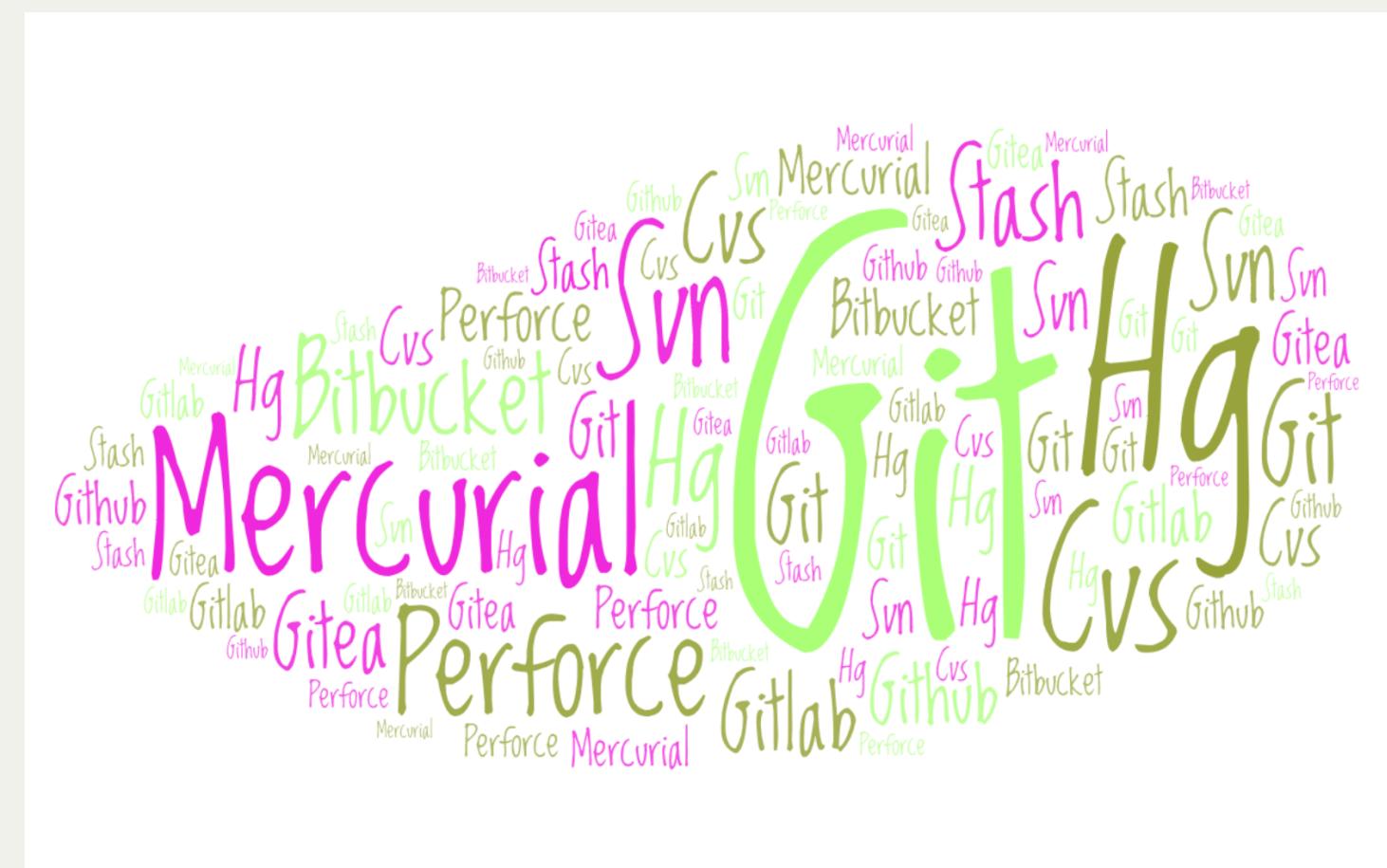
- Pour conserver une trace de **tous** les changements dans un historique
  - "Source unique de vérité" ( *Single Source of Truth*)
- Pour **collaborer** efficacement sur un même référentiel

# Concepts des VCS



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

# Quel VCS utiliser ?



Nous allons utiliser **Git**

# Git

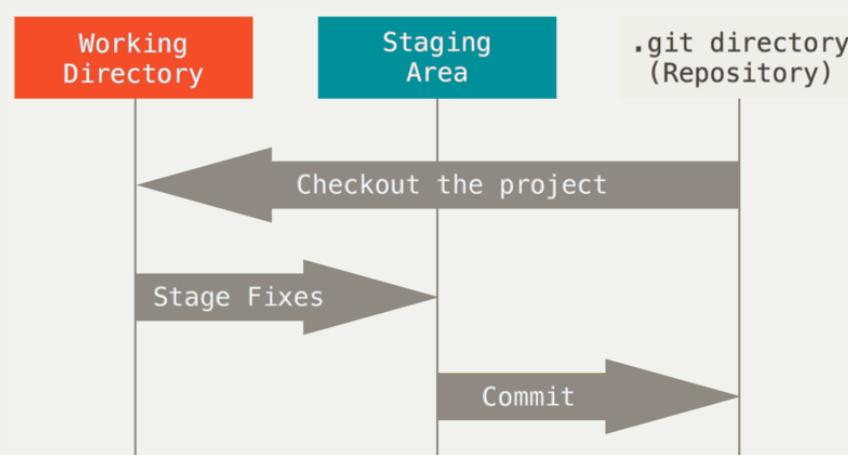
*Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.*

<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

- Dans le terminal de votre environnement GitPod:

- Créez un dossier vide nommé `projet-vcs-1` dans le répertoire `/workspace`, puis positionnez-vous dans ce dossier

```
mkdir -p /workspace/projet-vcs-1/  
cd /workspace/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

```
mkdir -p /workspace/projet-vcs-1/
cd /workspace/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

```
10 cluster/addons/node-problem-detector/npd.yaml
...
@@ -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"

26   apiVersion: apps/v1
27   kind: DaemonSet
28   metadata:
29 +   name: npd-v0.8.5
30   namespace: kube-system
31   labels:
32     k8s-app: node-problem-detector
33 +   version: v0.8.5
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: v0.8.5
41   template:
42     metadata:
43       labels:
44         k8s-app: node-problem-detector
45 +   version: v0.8.5
46   kubernetes.io/cluster-service: "true"
```

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

Showing 12 changed files with 314 additions and 200 deletions.

- > 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

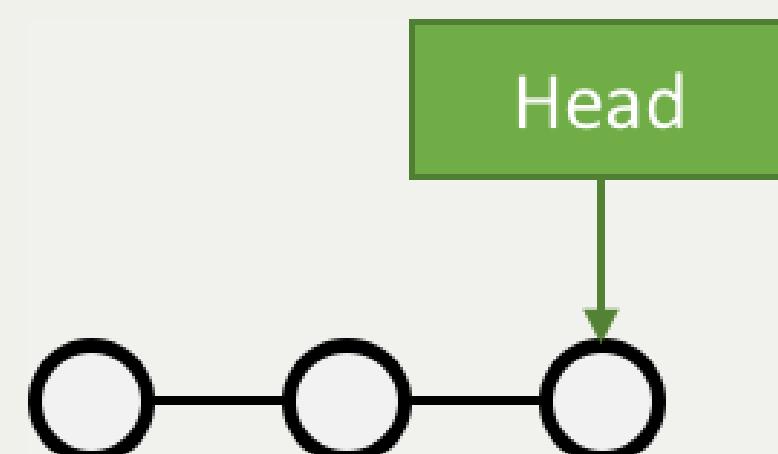
# Terminologie de Git - Commit

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

A screenshot of a GitHub commit page. The commit message is "✓ Bump node-problem-detector to v0.8.5". It was made by "master (#96716)" and committed by "tosi3k" 2 days ago. The commit has 1 parent, hash e64ebe0, and a commit message of 8f2dd3aaab02c3b4c1c8233aa5f93bb439f23228. It shows 3 changed files with 8 additions and 8 deletions. There are "Unified" and "Split" diff options.

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

O : 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
```

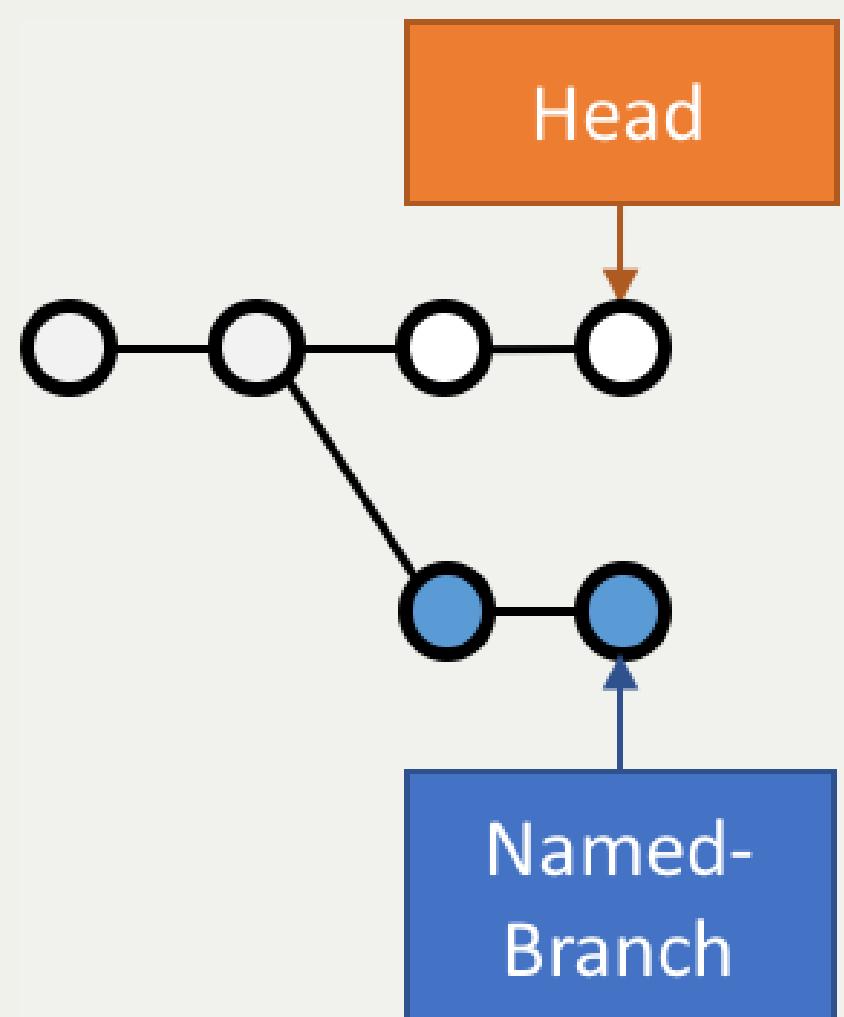
```
git diff  
git status
```

```
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

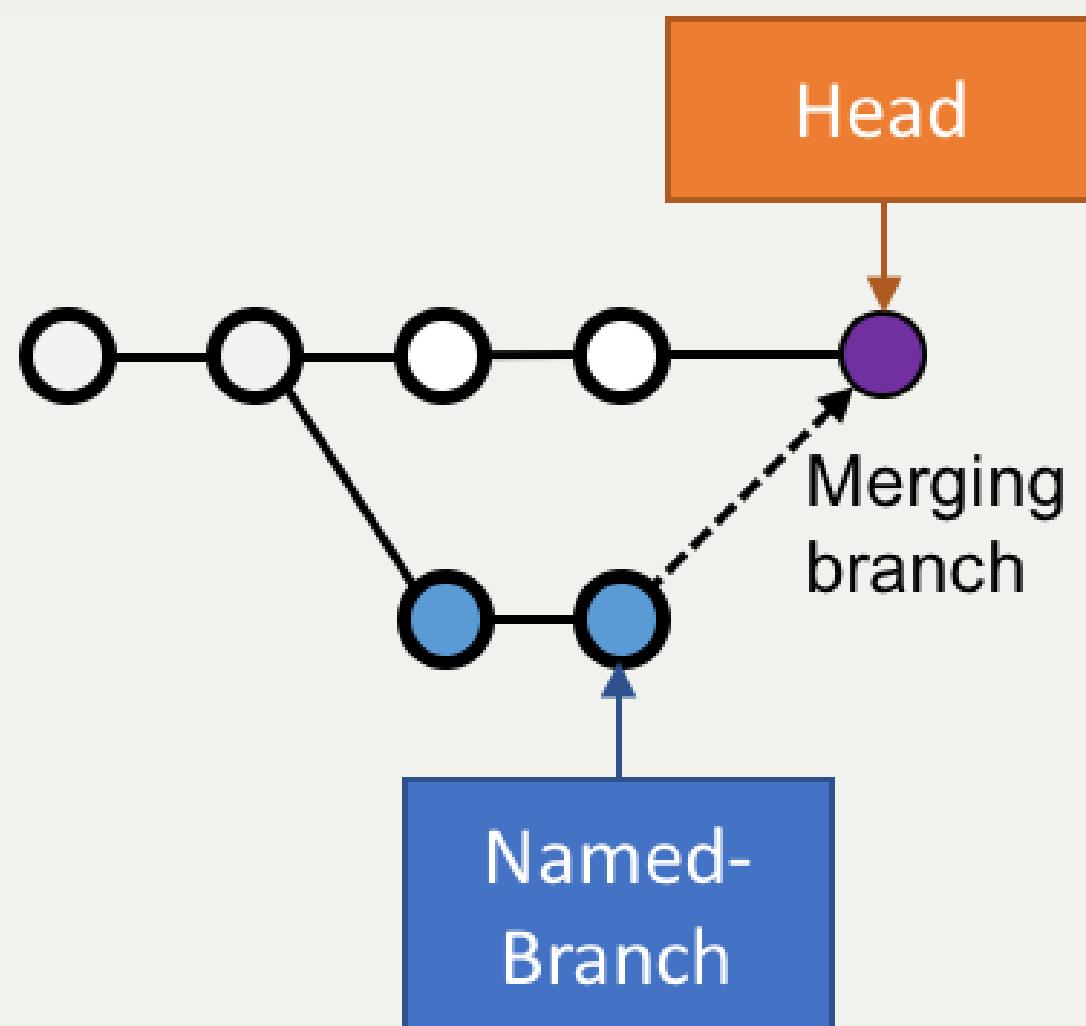
```
git branch feature/html && git switch feature/html
# Ou git switch --create feature/html
echo '<h1>Hello</h1>' > ./index.html
git add ./index.html && git commit --message="Ajout d'une page HTML par défaut" # -m / --message

git log
git log --graph
git lg # cat ~/.gitconfig => regardez la section section [alias], cette commande est déjà définie!
```

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
  - $\Delta$  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

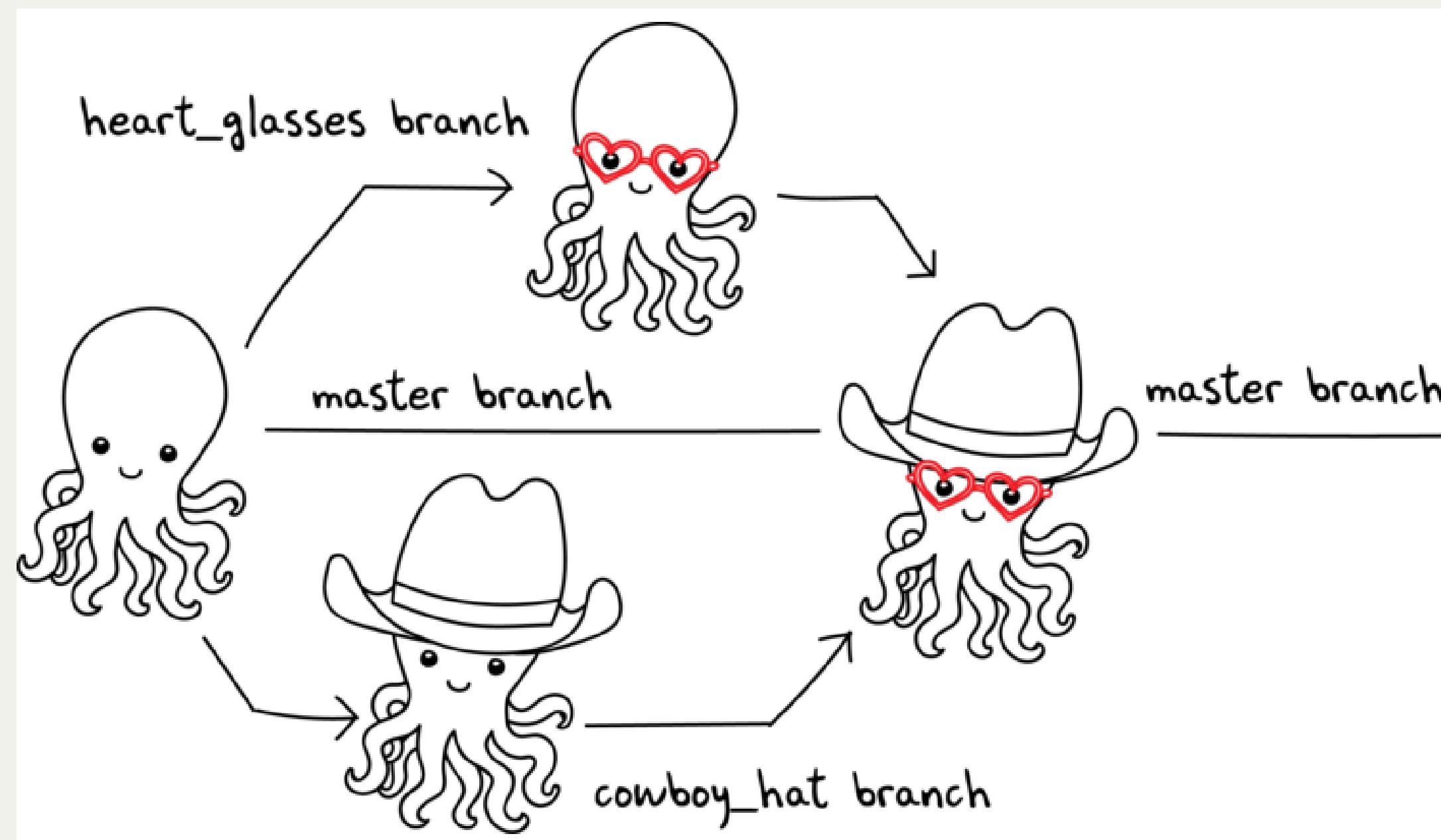
```
git switch main
git merge --no-ff feature/html # Enregistrer puis fermer le fichier 'MERGE_MSG' qui a été ouvert
git log --graph

# git lg
```

Copy

# 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/>

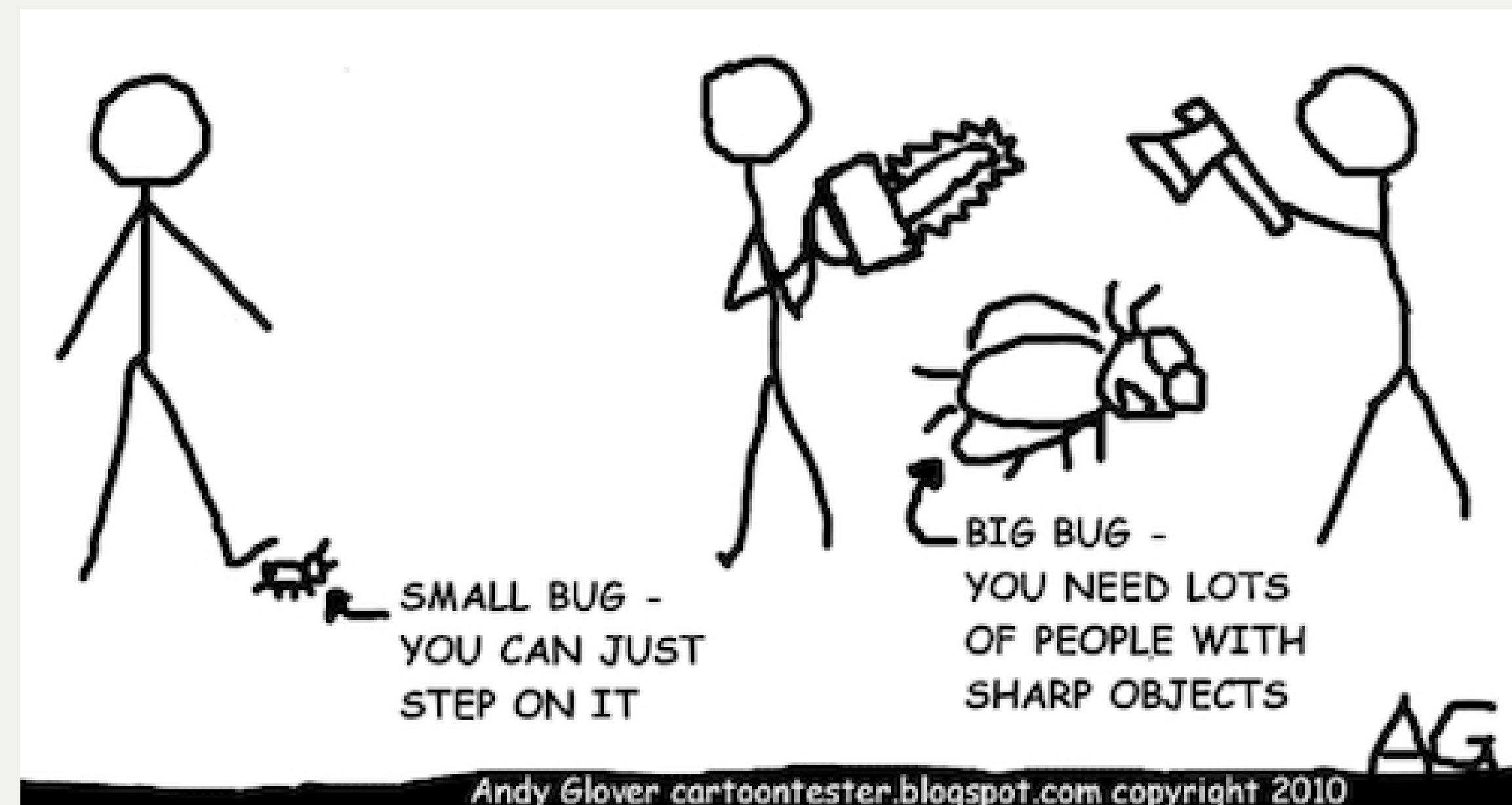
# Intégration Continue (CI)

*Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove.*

— Martin Fowler

# Pourquoi la CI ?

**But :** Déetecter les fautes au plus tôt pour en limiter le coût



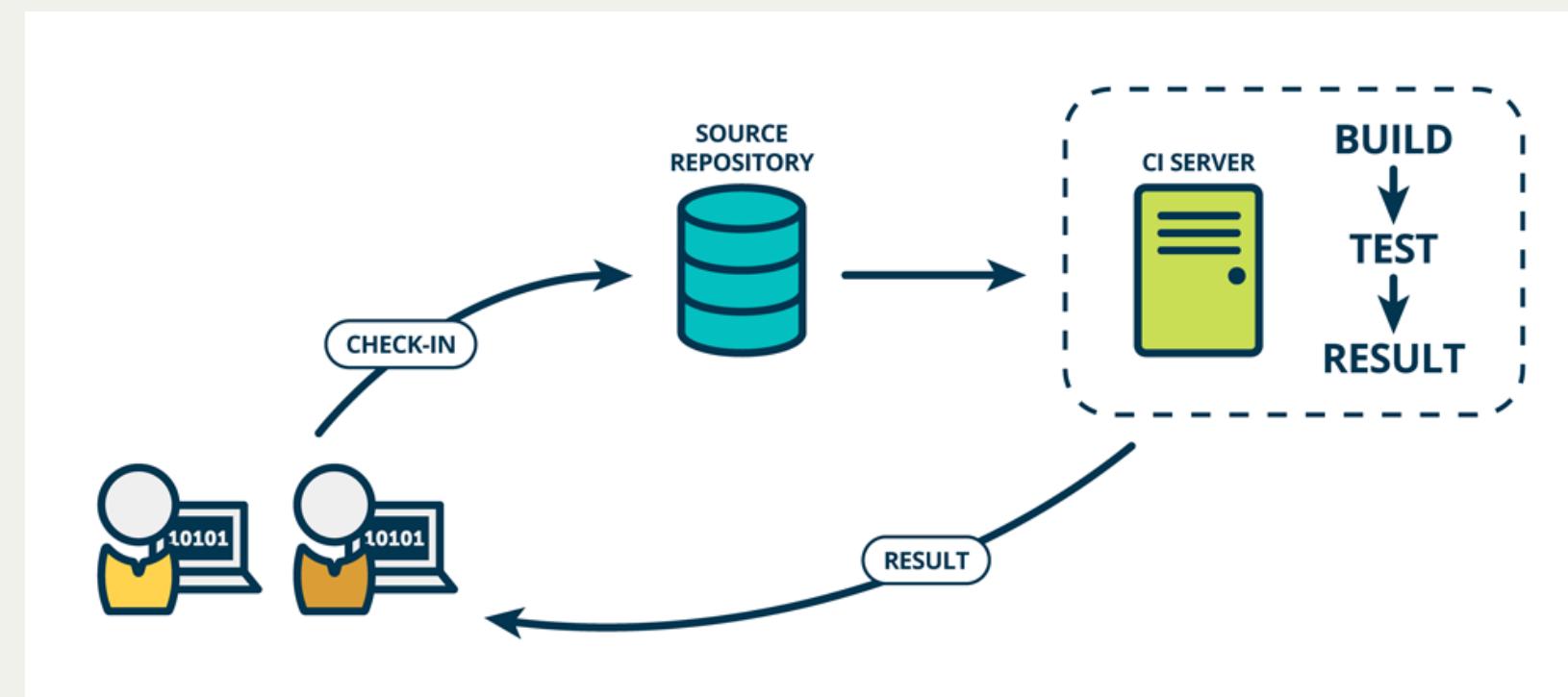
Source : <http://cartoontester.blogspot.be/2010/01/big-bugs.html>

# Qu'est ce que l'Intégration Continue ?

**Objectif** : que l'intégration de code soit un *non-événement*

- Construire et intégrer le code **en continu**
- Le code est intégré **souvent** (au moins quotidiennement)
- Chaque intégration est validée par une exécution **automatisée**

# Et concrètement ?



- Un•e dévelopeu•se•r ajoute du code/branche/PR :
  - une requête HTTP est envoyée au système de "CI"
- Le système de CI compile et teste le code
- On ferme la boucle : Le résultat est renvoyé au dévelopeu•se•r•s

# Quelques moteurs de CI connus

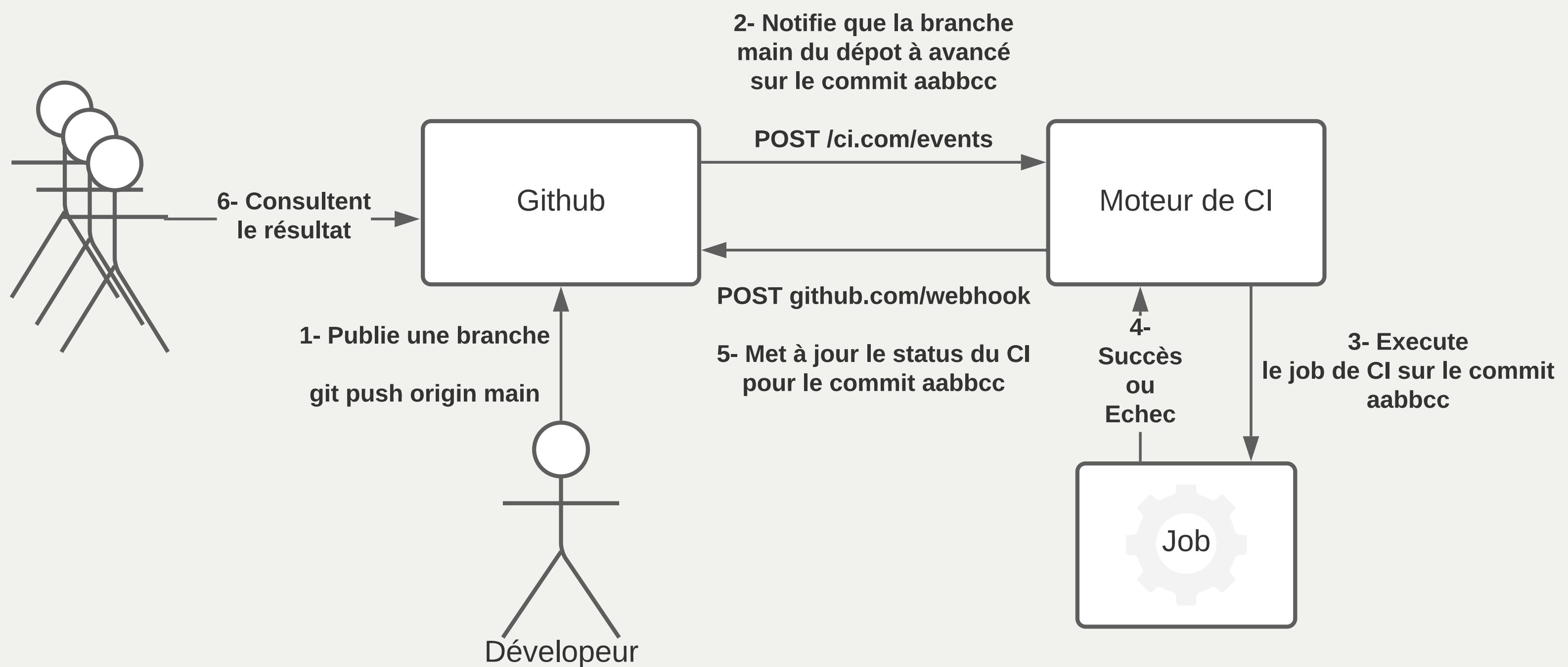
- A héberger soit-même : Jenkins, GitLab, Drone CI, CDS...
- Hébergés en ligne : Travis CI, Semaphore CI, Circle CI, Codefresh, GitHub Actions

# GitHub Actions

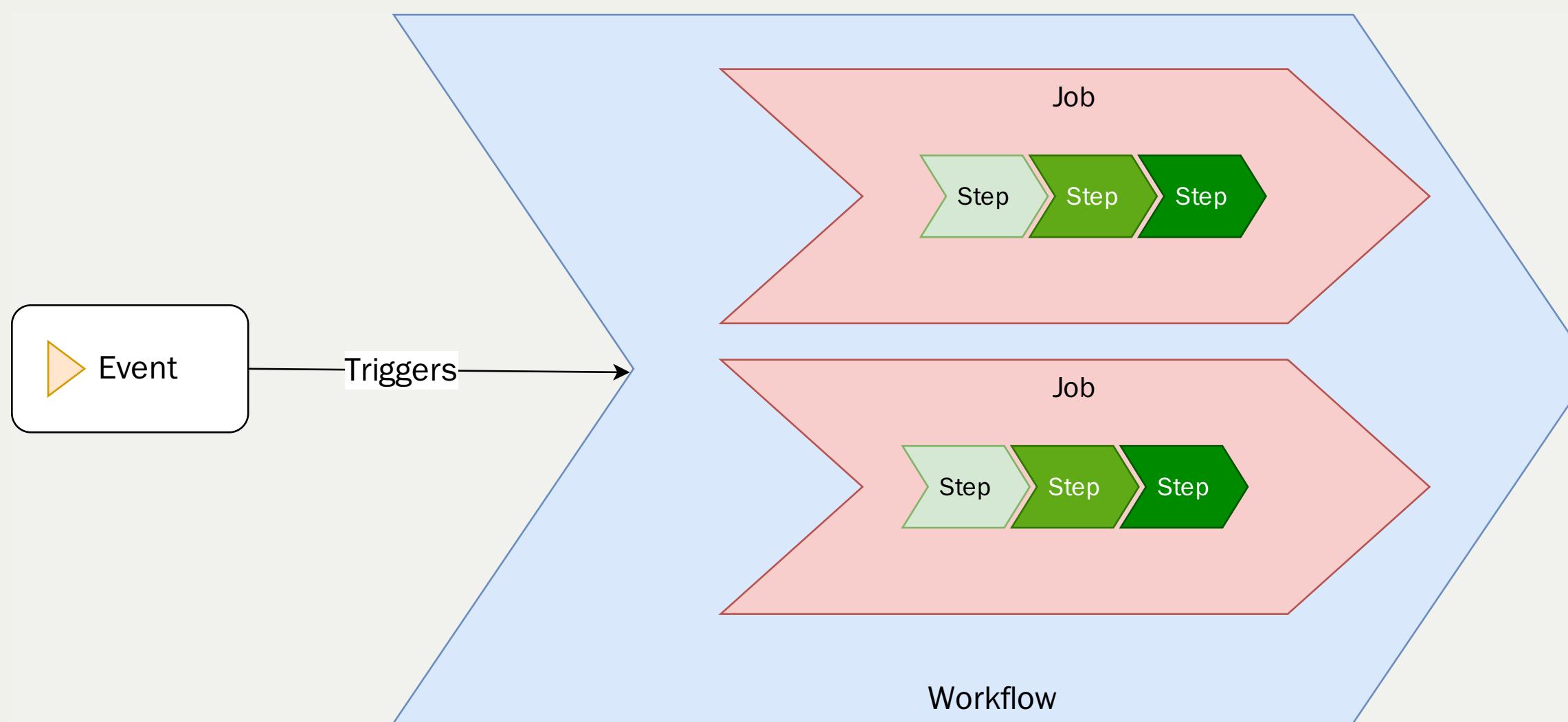
GitHub Actions est un moteur de CI/CD intégré à GitHub

- ✓ : Très facile à mettre en place, gratuit et intégré complètement
- ✗ : Utilisable uniquement avec GitHub, et DANS la plateforme GitHub

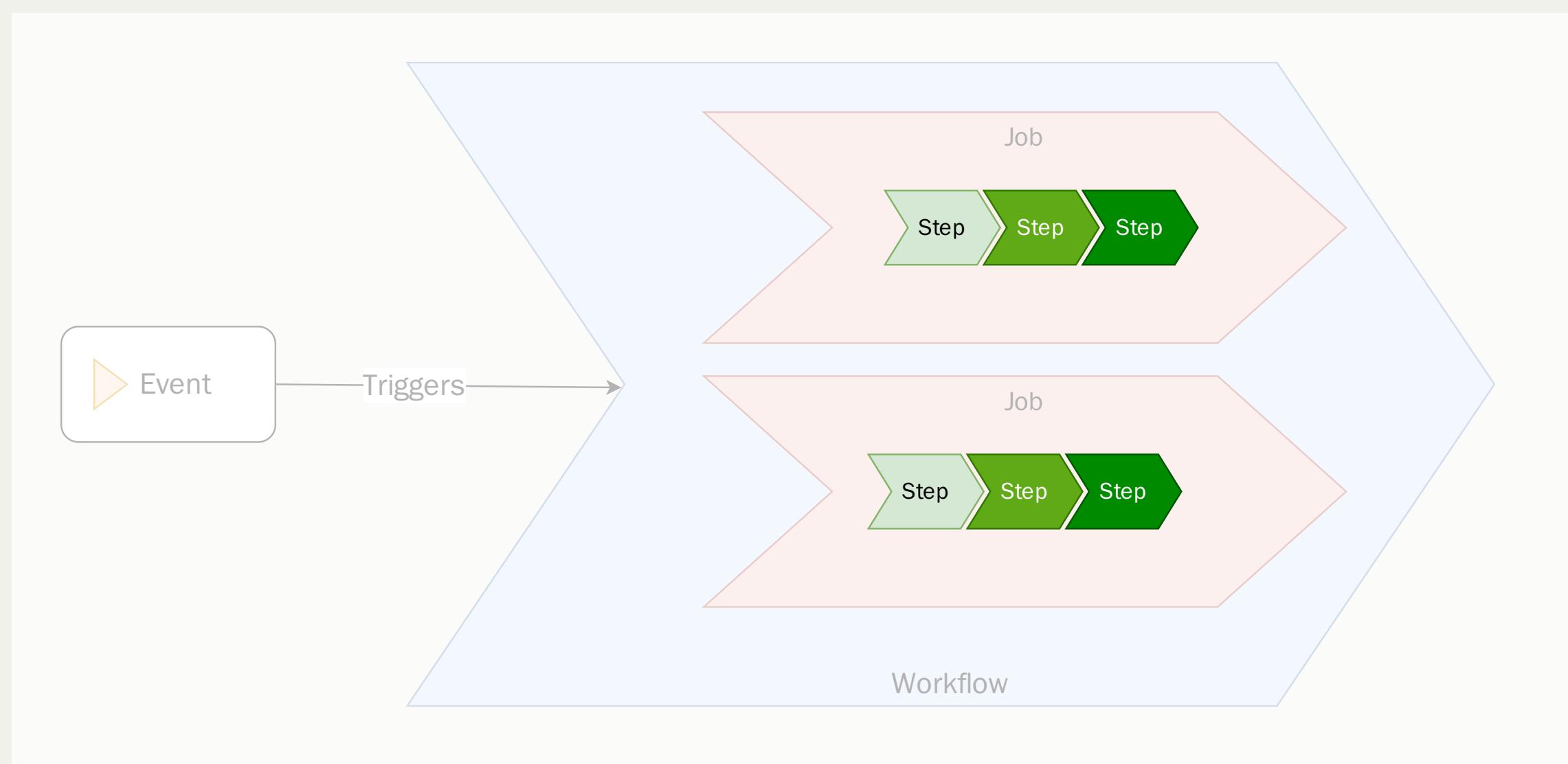
# Anatomie de déclenchement de GitHub Actions



# Concepts de GitHub Actions



# Concepts de GitHub Actions - Step 1/2



# Concepts de GitHub Actions - Step 2/2

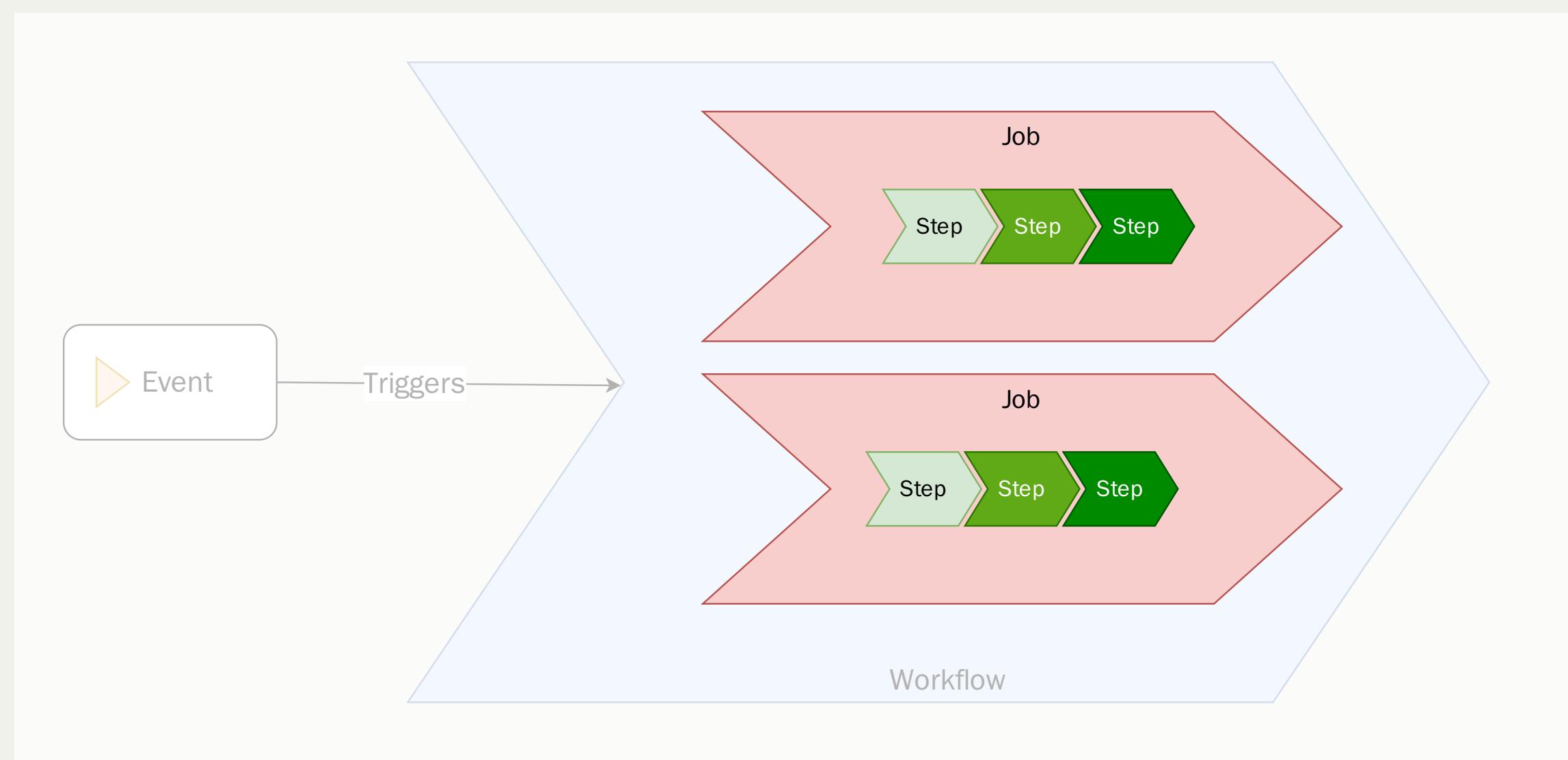
Une **Step** (étape) est une tâche individuelle à faire effectuer par le CI :

- Par défaut c'est une commande à exécuter - mot clef `run`
- Ou une "action" (quel est le nom du produit déjà ?) - mot clef `uses`
  - Réutilisables et partageables

```
steps: # Liste de steps
  # Exemple de step 1 (commande)
  - name: Say Hello
    run: echo "Hello ENSG"
  # Exemple de step 2 (une action)
  - name: 'Login to DockerHub'
    uses: docker/login-action@v1 # https://github.com/marketplace/actions/docker-login
    with:
      username: ${{ secrets.DOCKERHUB_USERNAME }}
      password: ${{ secrets.DOCKERHUB_TOKEN }}
```

Copy

# Concepts de GitHub Actions - Job 1/2



# Concepts de GitHub Actions - Job 2/2

Un **Job** est un groupe logique de tâches :

- Enchaînement *séquentiel* de tâches
- Regroupement logique : "qui a un sens"
  - Exemple : "compiler puis tester le résultat de la compilation"

```
jobs: # Map de jobs
  build: # 1er job, identifié comme 'build'
    name: 'Build Slides'
    runs-on: ubuntu-22.04 # cf. prochaine slide "Concepts de GitHub Actions - Runner"
    steps: # Collection de steps du job
      - name: 'Build the JAR'
        run: mvn package
      - name: 'Run Tests on the JAR file'
        run: mvn verify
  deploy: # 2nd job, identifié comme 'deploy'
    # ...
```

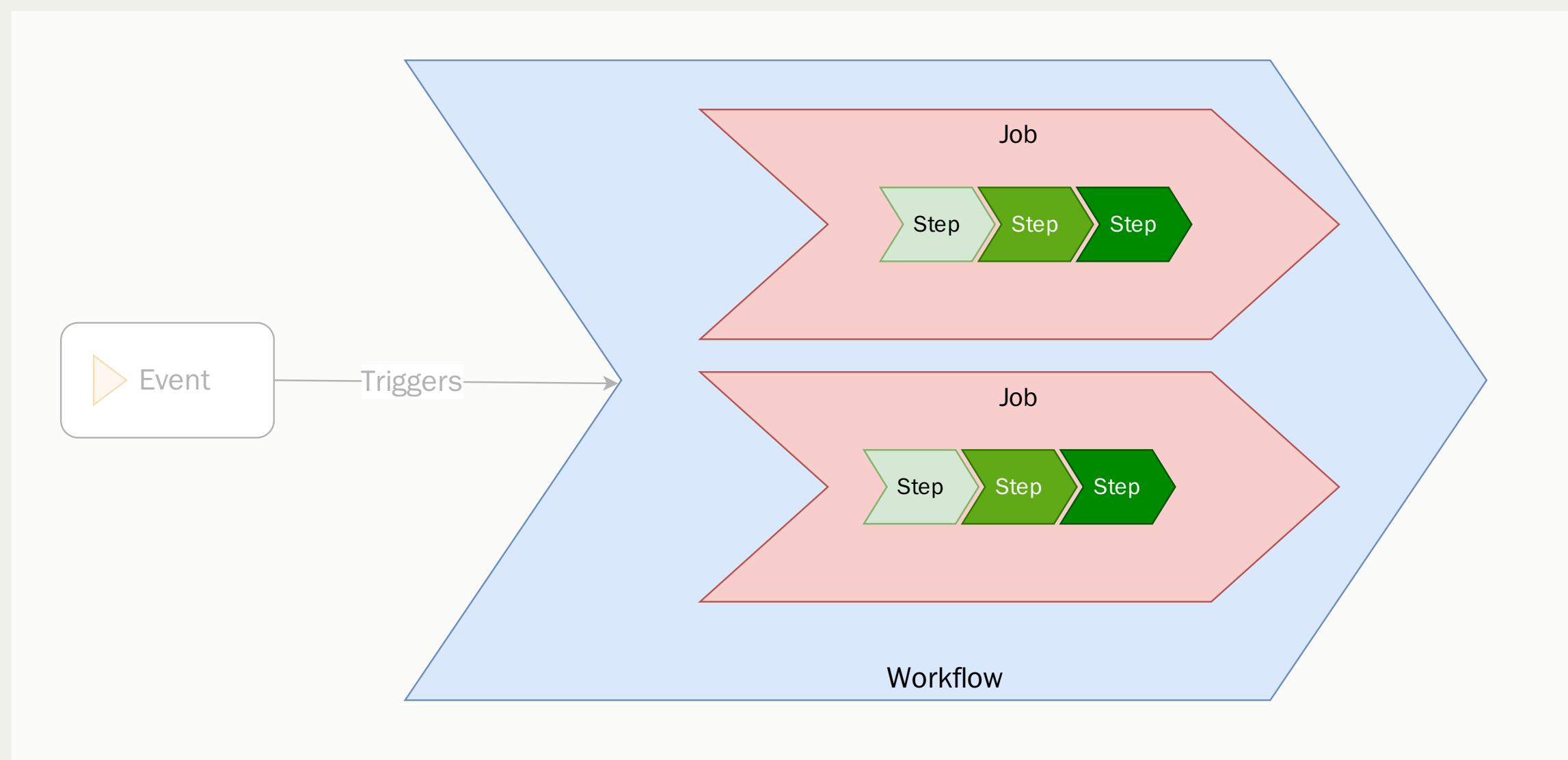
Copy

# Concepts de GitHub Actions - Runner

Un **Runner** est un serveur distant sur lequel s'exécute un job.

- Mot clef `runs-on` dans la définition d'un job
- Défaut : machine virtuelle Ubuntu dans le cloud utilisé par GitHub
- D'autres types sont disponibles (macOS, Windows, etc.)
- Possibilité de fournir son propre serveur

# Concepts de GitHub Actions - Workflow 1/2



# Concepts de GitHub Actions - Workflow 2/2

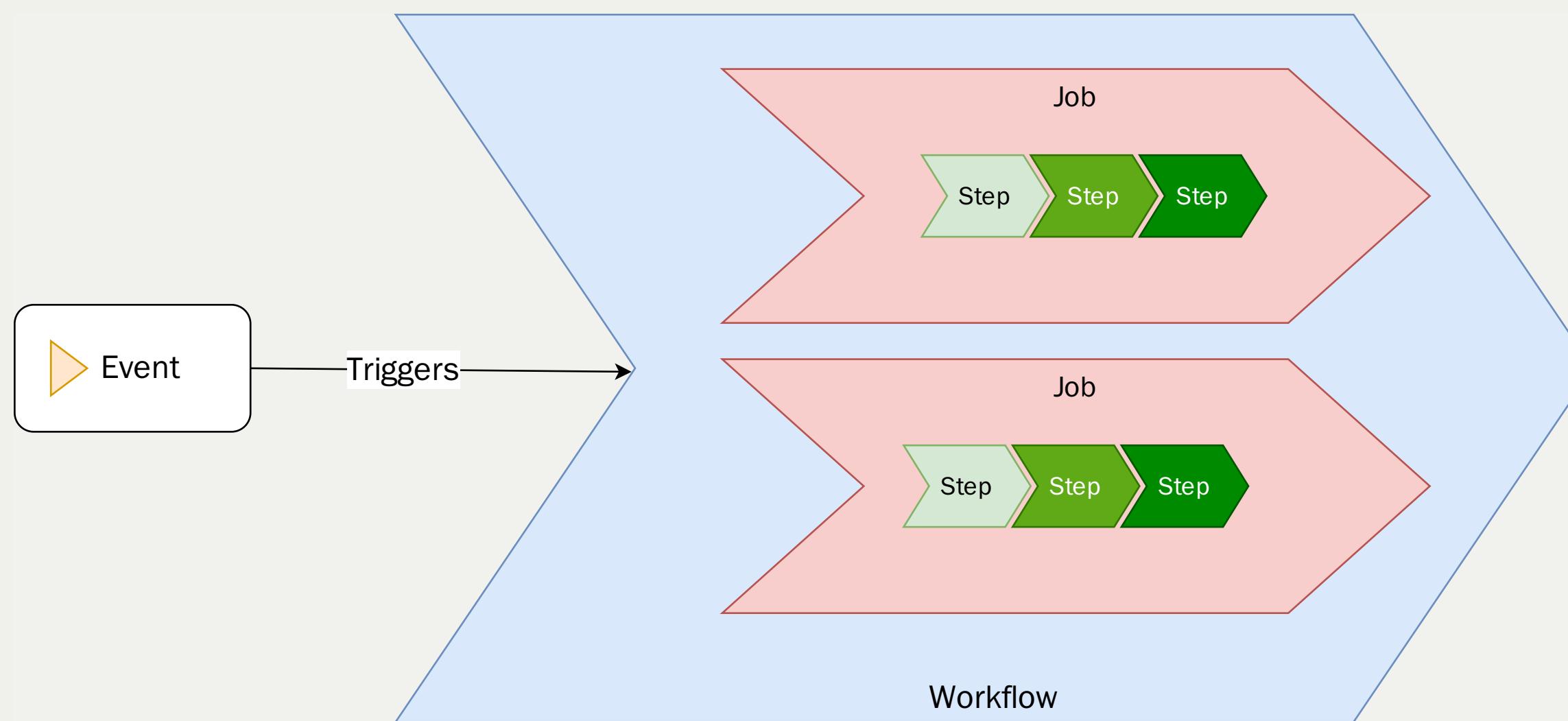
Un **Workflow** est une procédure automatisée composée de plusieurs jobs, décrite par un fichier **YAML**.

- On parle de "Workflow/Pipeline as Code"
- Chemin : `.github/workflows/<nom du workflow>.yml`
- On peut avoir *plusieurs* fichiers donc *plusieurs* workflows

```
.github/workflows
├── ci-cd.yaml
├── bump-dependency.yaml
└── nightly-tests.yaml
```

Copy

# Concepts de GitHub Actions - Évènement 1/2



# Concepts de GitHub Actions - Évènement 2/2

Un **évenement** du projet GitHub (push, merge, nouvelle issue, etc. ) déclenche l'exécution du workflow

- Plein de type d'évènements : push, issue, alarme régulière, favori, fork, etc.
  - Exemple : "Nouveau commit poussé", "chaque dimanche à 07:00", "une issue a été ouverte" ...
- Un workflow spécifie le(s) évènement(s) qui déclenche(nt) son exécution
  - Exemple : "exécuter le workflow lorsque un nouveau commit est poussé ou chaque jour à 05:00 par défaut"

# Concepts de GitHub Actions : Exemple Complet

Workflow File :

```
name: Node.js CI
on: # Événements déclencheurs
  - push:
    branch: main # Lorsqu'un nouveau commit est poussé sur la branche "main"
  - schedule:
    - cron: */15 * * * * # Toutes les 15 minutes
jobs:
  test-linux:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v3
      - run: npm install
      - run: npm test
  test-mac:
    runs-on: macos-12
    steps:
      - uses: actions/checkout@v3
      - run: npm install
      - run: npm test
```

Copy

# Essayons GitHub Actions

- **But :** nous allons créer notre premier workflow dans GitHub Actions
- N'hésitez pas à utiliser la documentation de GitHub Actions:
  - Accueil
  - Quickstart
  - Référence



# Exercice: Créez un dépôt (git dans) GitHub

- En étant authentifié dans GitHub,
- Créez un nouveau dépôt nommé esgi-devops-2023
  - Pas de "template" (modèle)
  - Visibilité publique
  - Initialisation avec un fichier README.md



# Exercice: Récupérez le dépôt dans Gitpod

- Obtenez l'URL (HTTPS) du dépôt GitHub fraîchement créé
  - ⚡ Depuis la page du dépôt, cliquez sur le bouton vert intitulé "**Code**"
- Dans Gitpod,
  - Depuis un terminal, positionnez-vous dans le dossier `/workspace`,
  - Clonez le dépôt avec la commande `git clone https://github.com/xxx/esgi-devops-2023`
  - ⚡ Si le dépôt n'apparaît pas dans l'"Explorer" à gauche : `code -a /workspace/esgi-devops-2023`



# Exercice: Exemple simple avec GitHub Actions

- Dans le dépôt esgi-devops-2023, sur la branch main,
  - Créez le fichier `.github/workflows/bonjour.yml` avec le contenu suivant :

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - run: echo "Bonjour 🙋"
```

Copy

- Commitez puis poussez le fichier sur GitHub:

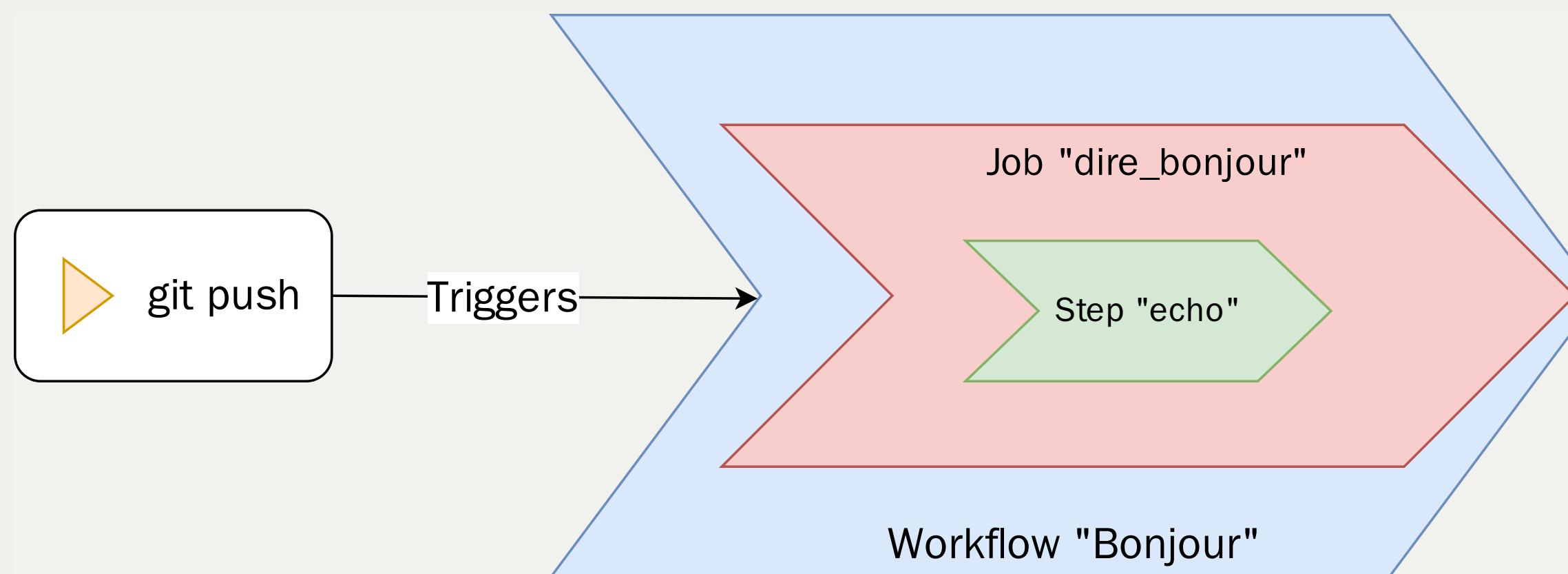
```
git add .github/workflows/bonjour.yml
git commit -m 'Create GitHub workflow Bonjour'
git push origin main
```

Copy



# Exercice: Exemple simple avec GitHub Actions : Récapète

- Revenez sur la page GitHub de votre projet et naviguez dans l'onglet "Actions" :
  - Voyez-vous un workflow ? Et un Job ? Et le message affiché par la commande echo ?



# Exemple GitHub Actions : Checkout

- Supposons que l'on souhaite utiliser le code du dépôt...
  - Essayez: modifiez le fichier `bonjour.yml` pour afficher le contenu de `README.md`:

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - run: ls -l # Liste les fichiers du répertoire courant
      - run: cat README.md # Affiche le contenu du fichier `README.md` à la base du dépôt
```

Copy

- Est-ce que l'étape se passe bien ? (SPOILER: non ✗ )



# Exercice GitHub Actions : Checkout

- **But** : On souhaite récupérer ("checkout") le code du dépôt dans le job
- 🚧 C'est à vous d'essayer de *réparer* 🔧 le job :
  - L'étape doit être conservée et doit fonctionner
  - Utilisez l'action "checkout" (Documentation) du marketplace GitHub Action
  - Vous pouvez vous inspirer du Quickstart de GitHub Actions

# ✓ Solution GitHub Actions : Checkout

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v3 # Récupère le contenu du dépôt correspondant au commit du workflow en cours
        - run: ls -l # Liste les fichiers du répertoire courant
        - run: cat README.md # Affiche le contenu du fichier `README.md` à la base du dépôt
```

Copy

# Exemple : Environnement d'exécution

- Notre workflow doit s'assurer que "la vache" 🐄 doit nous lire 🗂 le contenu du fichier README.md
  - WAT 😳 ?
- Essayez la commande `cat README.md | cowsay` dans GitPod
  - Modifiez l'étape du workflow pour faire la même chose dans GitHub Actions
  - SPOILER: ✘ (la commande `cowsay` n'est pas disponible dans le runner GitHub Actions)

# Problème : Environnement d'exécution

- **Problème** : On souhaite utiliser les mêmes outils dans notre workflow ainsi que dans nos environnement de développement
- Plusieurs solutions existent pour personnaliser l'outillage, chacune avec ses avantages / inconvénients :
  - Personnaliser l'environnement dans votre workflow: ( $\triangle$  sensible aux mises à jour,  $\checkmark$  facile à mettre en place)
  - Spécifier un environnement préfabriqué pour le workflow ( $\triangle$  complexe,  $\checkmark$  portable)
  - Utiliser les fonctionnalités de votre outil de CI ( $\triangle$  spécifique au moteur de CI,  $\checkmark$  efficacité)



# Exercice : Personnalisation dans le workflow

- **But :** exécuter la commande `cat README.md | cowsay` dans le workflow comme dans GitPod
- 🚀 C'est à vous de mettre à jour le workflow pour personnaliser l'environnement :
  - 🔎 Cherchez comment installer `cowsay` dans le runner GitHub (`runs-on...`)

# ✓ Solution : Personnalisation dans le workflow

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v3 # Récupère le contenu du dépôt correspondant au commit du workflow en cours
      - run: |
          sudo apt-get update
          sudo apt-get install -y cowsay
      - run: cat README.md | cowsay
```

Copy



# Exercice : Environnement préfabriqué

- **But** : exécuter la commande `cat README.md | cowsay` dans le workflow comme dans GitPod
  - En utilisant le même environnement que GitPod (même version de cowsay, java, etc.)
- 🚀 C'est à vous de mettre à jour le workflow pour exécuter les étapes dans la même image Docker que GitPod :
  - 🌐 Image utilisée dans GitPod
  - 🌐 Utilisation d'un container comme runner GitHub Actions
  - 🌐 Contraintes d'exécution de container dans GitHub Actions (`--user=root`)

# ✓ Solution : Environnement préfabriqué

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    container:
      image: ghcr.io/dduportal/esgi-gitpod
      options: --user=root
    steps:
      - uses: actions/checkout@v3 # Récupère le contenu du dépôt correspondant au commit du workflow en cours
        - run: cat README.md | cowsay
```

Copy

# Checkpoint



- Quel est l'impact en terme de temps d'exécution du changement précédent ?
- **Problème :** Le temps entre une modification et le retour est crucial
- Docker ? 😊



# Work in progress

🚧 Revenez plus tard