

GITLAB CI/CD

Maîtriser la gestion du cycle de vie des développements logiciels



Qui suis-je ?

Thomas Saquet

(Non. Pas comme le hobbit.)

✉️ thomas@solution-libre.fr

🐦 X : [@tsaquet](https://twitter.com/tsaquet) / [@qtgeekes](https://twitter.com/qtgeekes)

🐦 BlueSky : [@tsaquet.bsky.social](https://blue.sky/@tsaquet.bsky.social)

🎙️ Twitch : [qtgeekes](https://twitch.tv/qtgeekes)

▶️ Youtube : [Qu'est-ce que tu GEEKes ?](https://www.youtube.com/c/QuestcequeGEEKes)

Formateur / Vulgarisateur

Formation DevOps depuis 2015

Vidéos Youtube depuis 2017

Streaming Twitch depuis 2020

Expert Dev / DevOps / Linux

Professionnel depuis 2008

Développement, conseil, accompagnement

9h00

Début du cours
(9h30 le premier jour)

10h30

Pause
(15 minutes)

12h30

Pause Déjeuner
(1 heure / 1 heure 30)

15h30

Pause
(15 minutes)

17h00

Fin du cours
(17h30 si nécessaire)
(17h au plus tard le dernier jour)

Objectifs pédagogiques

À l'issue de la formation, vous serez en mesure de :

- Connaître l'offre GitLab
- Pratiquer la gestion de versions avec Git et collaborer avec GitLab
- Mettre en place l'intégration continue (CI) et le déploiement continu (CD) avec GitLab
- Appréhender les éléments constitutifs d'une usine logicielle DevOps

Organisation du cours

 N'hésitez pas à prendre des notes, à réécrire les explications avec vos mots.

 Prenez la parole, posez des questions 

 Vous avez le support de cours, essayez de ne pas aller voir les pages suivantes en avance 

A vous !

- Votre nom.
- Votre entreprise.
- Votre métier.
- Vos compétences sur les différents sujets de la formation.
- Votre projet, vos attentes du cours.

Thèmes abordés dans le cours

- Chapitre 1 : Git, le socle de Gitlab
 - Dans lequel nous aborderons la base de git
- Chapitre 2 : Docker, le porte-conteneurs
 - Dans lequel nous aborderons les concepts de base de Docker
- Chapitre 3 : Gitlab un outil DevOps
 - Où l'on découvre l'usine Gitlab
- Chapitre 4 : Gitlab CI/CD²
 - Où l'on apprends ce que sont intégration, livraison et déploiement continus
- Chapitre 5 : Gitlab et au-delà
 - Où l'on découvre des fonctionnalités avancées

CHAPITRE 1

GIT, LE SOCLE DE GITLAB

GIT, LE SOCLE DE GITLAB

- **Concepts de git** nous permettra de découvrir git.
- **Les zones et les états** nous présentera les zones de travail et les états des fichiers.
- **Les modèles de branches** nous présentera les branches et un modèle pour les utiliser.
- **Les branches, des pointeurs** nous expliquera comment fonctionnent les branches.
- **Merge Vs. Rebase** nous présentera le mécanisme de merge et de rebase
- **Pour aller plus loin** nous permettra de parler des sujets dont on ne va pas parler.
- **GIT - TP - Jeux** nous permettra de manipuler les concepts vus en cours à l'aide de jeux.

GIT, LE SOCLE DE GITLAB

Concepts de git

- Les zones et les états
- Les modèles de branches
- Les branches, des pointeurs
 - Merge Vs. Rebase
 - Pour aller plus loin
 - GIT - TP - Jeux

CONCEPTS DE GIT

Gestion de version à l'ancienne

- Copier-coller le code-source dans un répertoire : (Monprogramme-version2.good.new.final/)
- Outil historique : CVS (Le premier utilisé massivement !)
- Outils centralisés : SVN, ClearCase, Perforce
 - Un serveur stocke l'historique, les copies de travail ne contiennent qu'une version
 - Utilisation partielle sans réseau
- Outils dé-centralisés : **Git**, Mercurial, Darcs, Fossil
 - Chaque copie de travail comporte aussi l'historique
 - Mécanismes de synchro entre deux dépôts

Concepts de git

Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

CONCEPTS DE GIT

Gestion de version

La gestion de version a de nombreux avantages :

- **Retours en arrière** et corrections toujours possibles
- **Historique** de toutes les opérations
- Indispensable pour le **travail en équipe**
- Travaux en **parallèle** sur **plusieurs branches**
- Pour du **code** mais aussi un **site web**, de la **doc**, de la **conf**, etc.

Concepts de git

Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

CONCEPTS DE GIT

Pourquoi indispensable ?

Le développement de logiciel est un processus itératif :

- Il y a très souvent de **nouvelles versions** (corrections, ajouts)

L'historique des développements est important :

- Comprendre d'où provient un bug
- Identifier comment une modification a été faite
- **Tester** dans une version antérieure encore déployée chez un client

Concepts de git

Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

CONCEPTS DE GIT

Concept d'historique

- Chaque modification du code-source (**révision**) est enregistrée dans un «dépôt» local
- L'ensemble des **révisions** forme l'historique

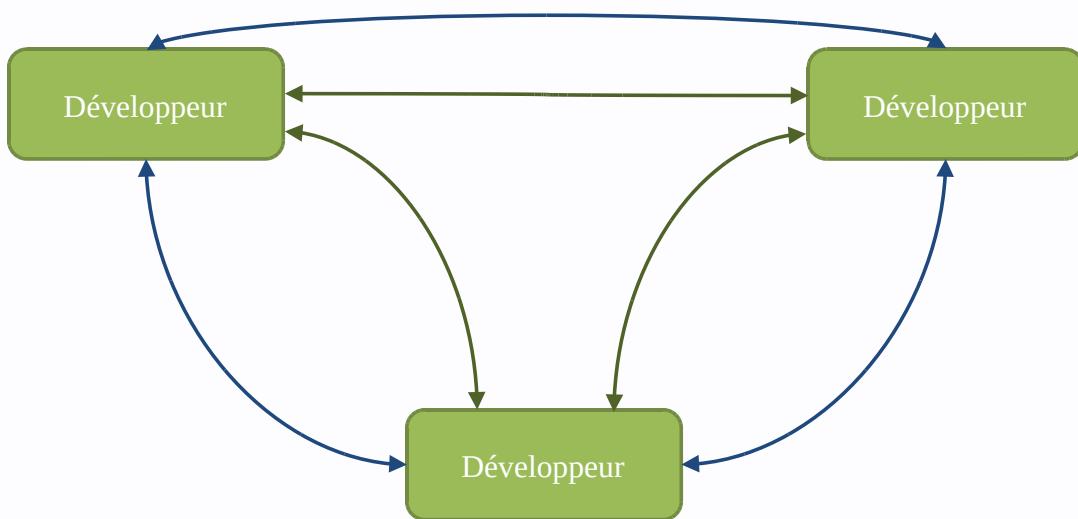
Concepts de git

Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

CONCEPTS DE GIT

Un système distribué

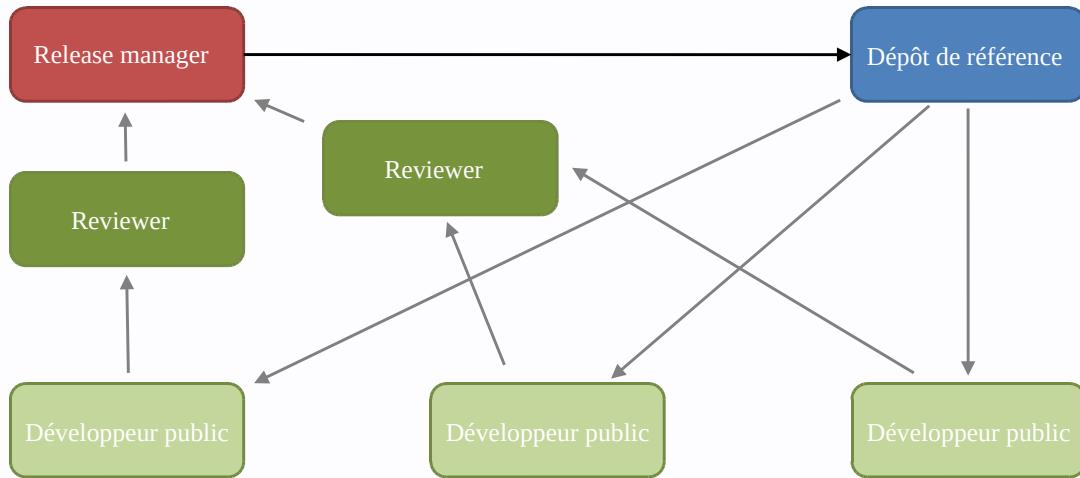
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



CONCEPTS DE GIT

Un système distribué

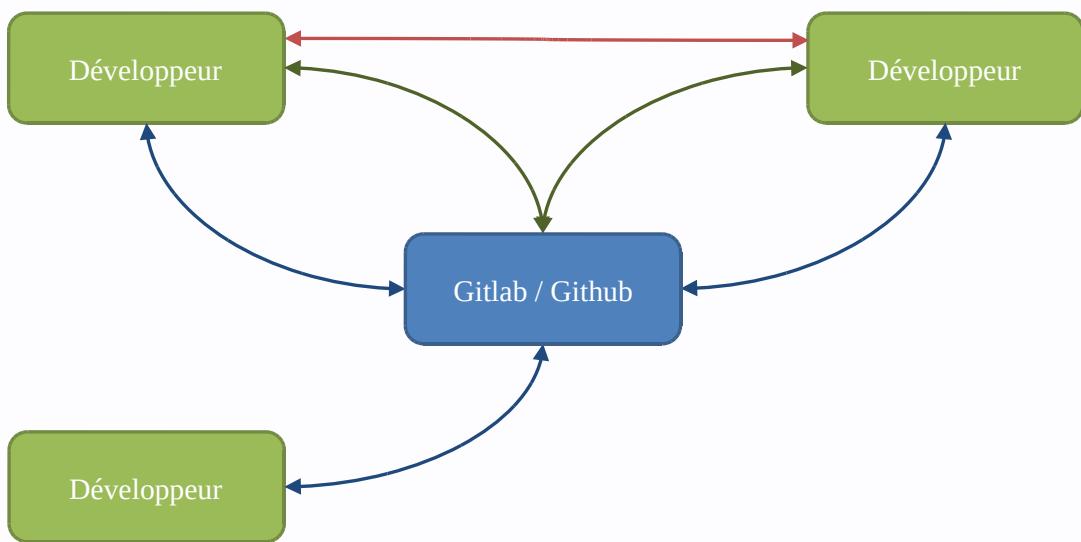
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



CONCEPTS DE GIT

Un système distribué

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



CONCEPTS DE GIT

Concrètement

- Tout est géré dans un répertoire .git à la base du projet
- Un système de branches permet de gérer des versions en parallèle
- Il y a des solutions de centralisations (Gitlab, Github)
- A distance on est connecté via SSH ou HTTPS
- Utilisation de l'empreinte SHA-1 pour gérer l'intégrité

Concepts de git

Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

GIT, LE SOCLE DE GITLAB

Concepts de git

Les zones et les états

Les modèles de branches

Les branches, des pointeurs

Merge Vs. Rebase

Pour aller plus loin

GIT - TP - Jeux

LES ZONES ET LES ÉTATS

Les 3 zones

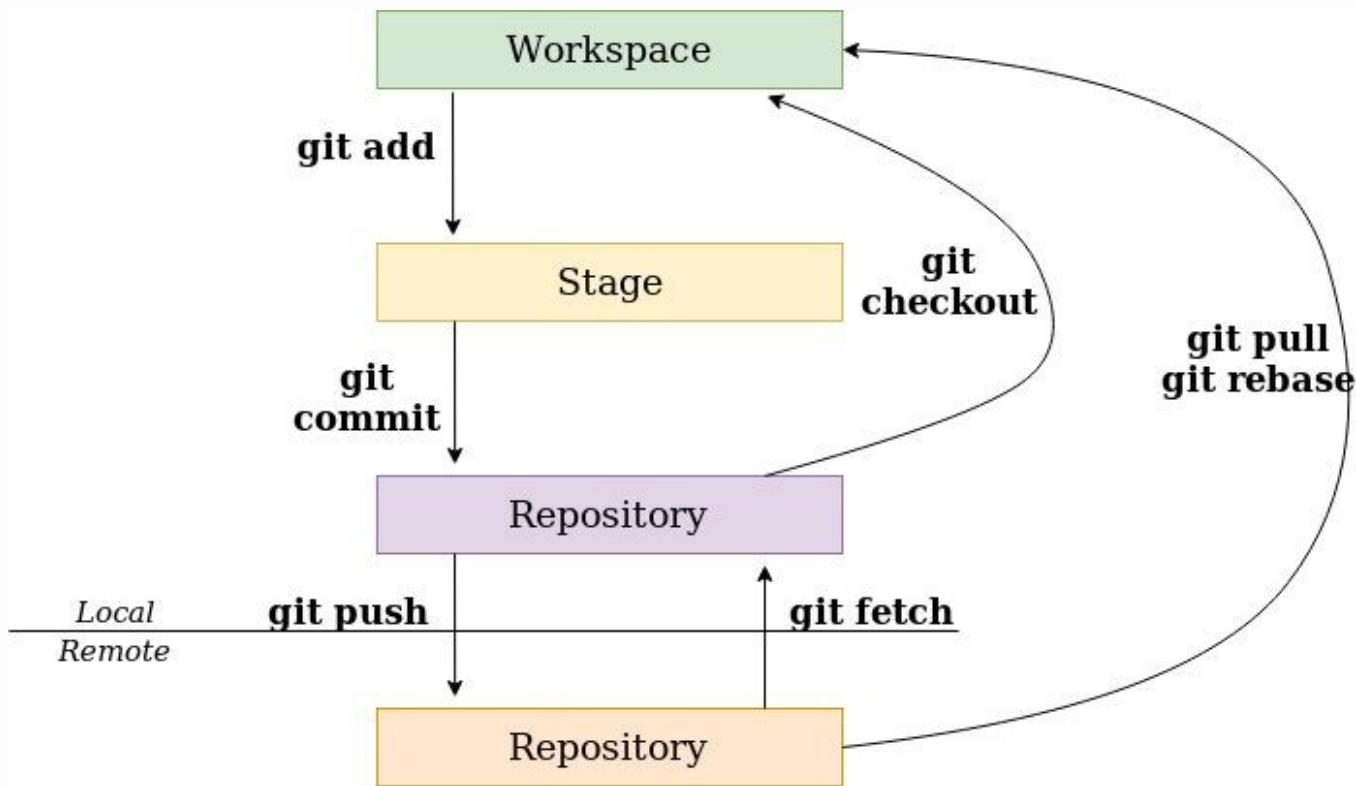
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

- **Working directory**
 - C'est ici qu'on travaille
 - On y modifie nos fichiers « normalement »
- **Index / stage**
 - Zone de « sauvegarde » des modifications qu'on souhaite partager
 - Les modifications y sont en attente.
- **Local repository**
 - C'est notre dépôt local, où se trouve la modification une fois qu'on a fait notre commit
 - C'est ce qui est prêt à être envoyé sur le serveur distant

LES ZONES ET LES ÉTATS

Changement de zone

Concepts de git
[Les zones et les états](#)
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



LES ZONES ET LES ÉTATS

Manipulations basiques

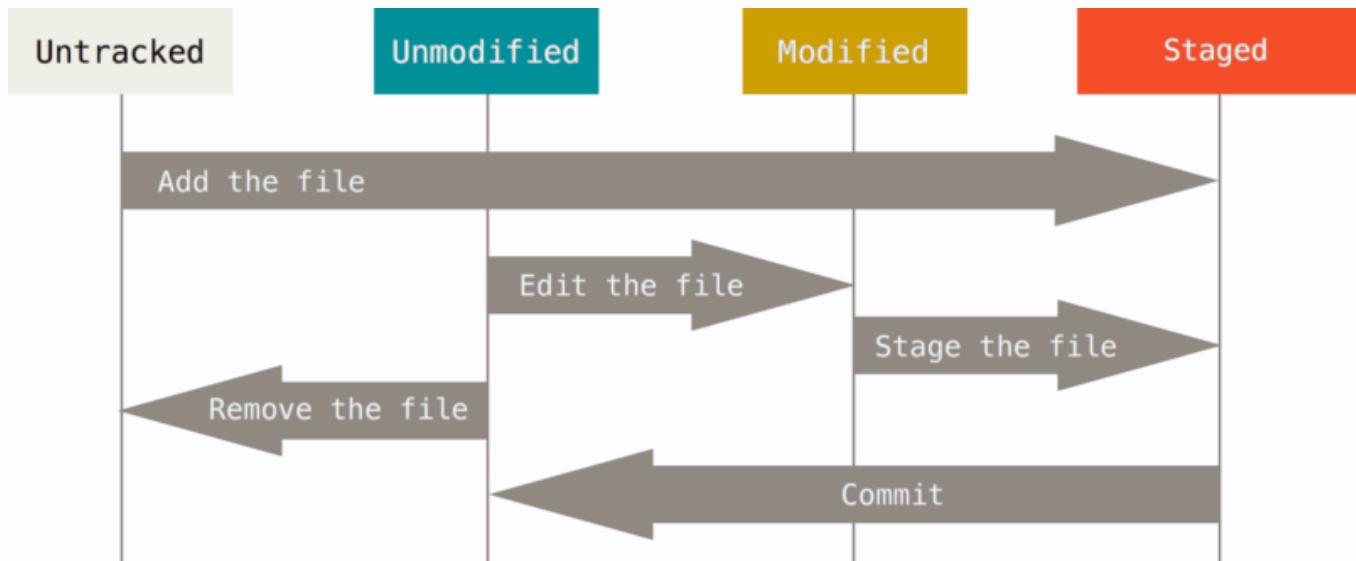
- Initialiser un dépôt :
 - git init
- Récupérer (cloner) un dépôt :
 - git clone
- Ajouter un dépôt distant :
 - git remote add
- Pousser votre version sur un dépôt :
 - git push
- Récupérer la dernière mise à jour d'un dépôt :
 - git fetch (+ git merge)
- Récupération des changements :
 - git pull

Concepts de git
[Les zones et les états](#)
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

LES ZONES ET LES ÉTATS

Etat des fichiers

Concepts de git
[Les zones et les états](#)
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



LES ZONES ET LES ÉTATS

Où en suis-je ?

- Connaître l'état du répertoire
- Regarder les différences par rapport à l'index (la zone de staging)

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

```
git status

On branch master

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git checkout -- <file>..." to discard changes in working
     directory)

      modified: views/add_commentaire.php

no changes added to commit (use "git add" and/or "git commit -a")
```

LES ZONES ET LES ÉTATS

En direction du commit

- Ajouter les modifications à l'index (zone de staging)

```
git add views/add_commentaire.php
git status

On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    modified: views/add_commentaire.php
```

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

LES ZONES ET LES ÉTATS

Désindexer des fichiers (1/2)

- Bien lire la réponse à **git status**

```
git status

On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    modified: views/add_commentaire.php
```

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

LES ZONES ET LES ÉTATS

Désindexer des fichiers (2/2)

- ... Et s'en servir

```
git restore --staged monfichier  
git status
```

```
On branch master  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
    (use "git checkout -- <file>..." to discard changes in  
     working directory)  
  
      modified: views/add_commentaire.php
```

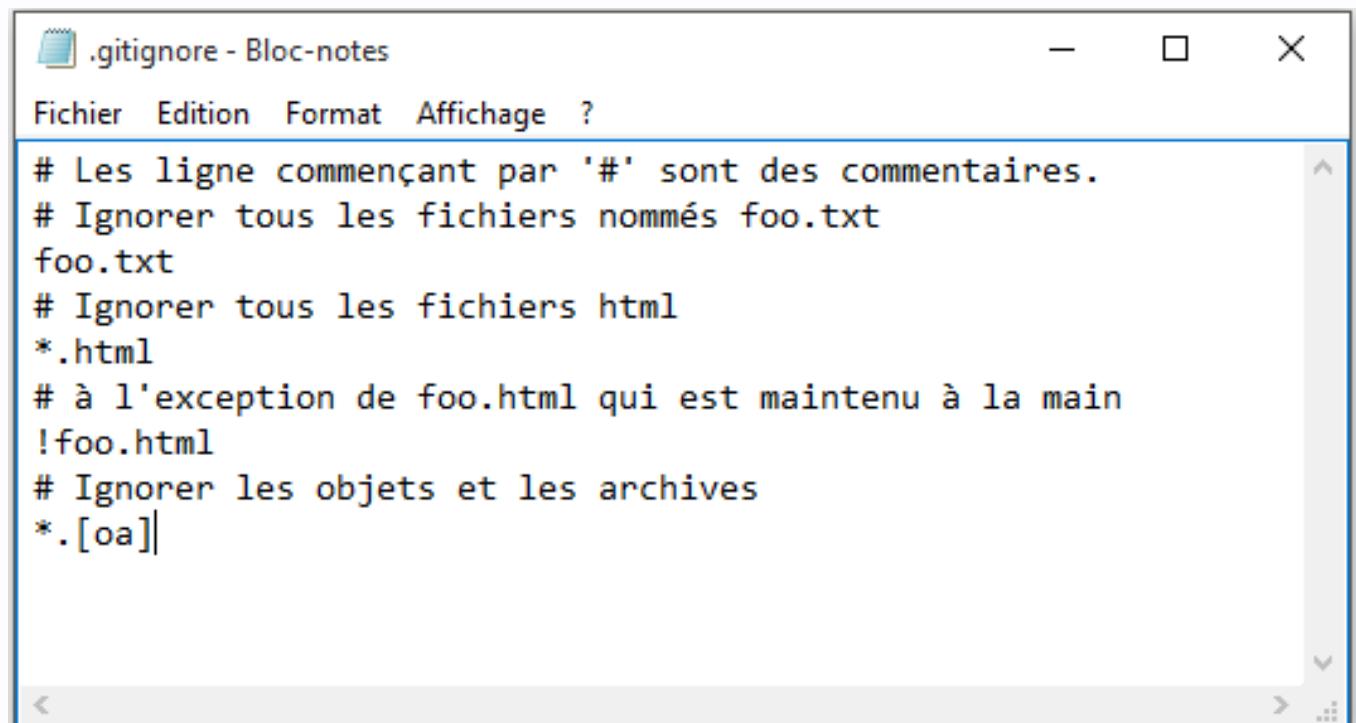
```
no changes added to commit (use "git add" and/or "git commit -a")
```

Concepts de git
[Les zones et les états](#)
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

LES ZONES ET LES ÉTATS

Ignorer des fichiers

Pour que Git ignore des fichiers (binaires, fichiers de conf, etc.), il faut un fichier **.gitignore** à la racine du dépôt. Il doit contenir les chemins à ignorer (expressions régulières).



The screenshot shows a Windows Notepad window with the title ".gitignore - Bloc-notes". The menu bar includes Fichier, Edition, Format, Affichage, and ?. The main content area contains the following text:

```
# Les lignes commençant par '#' sont des commentaires.  
# Ignorer tous les fichiers nommés foo.txt  
foo.txt  
# Ignorer tous les fichiers html  
*.html  
# à l'exception de foo.html qui est maintenu à la main  
!foo.html  
# Ignorer les objets et les archives  
*[oa]
```

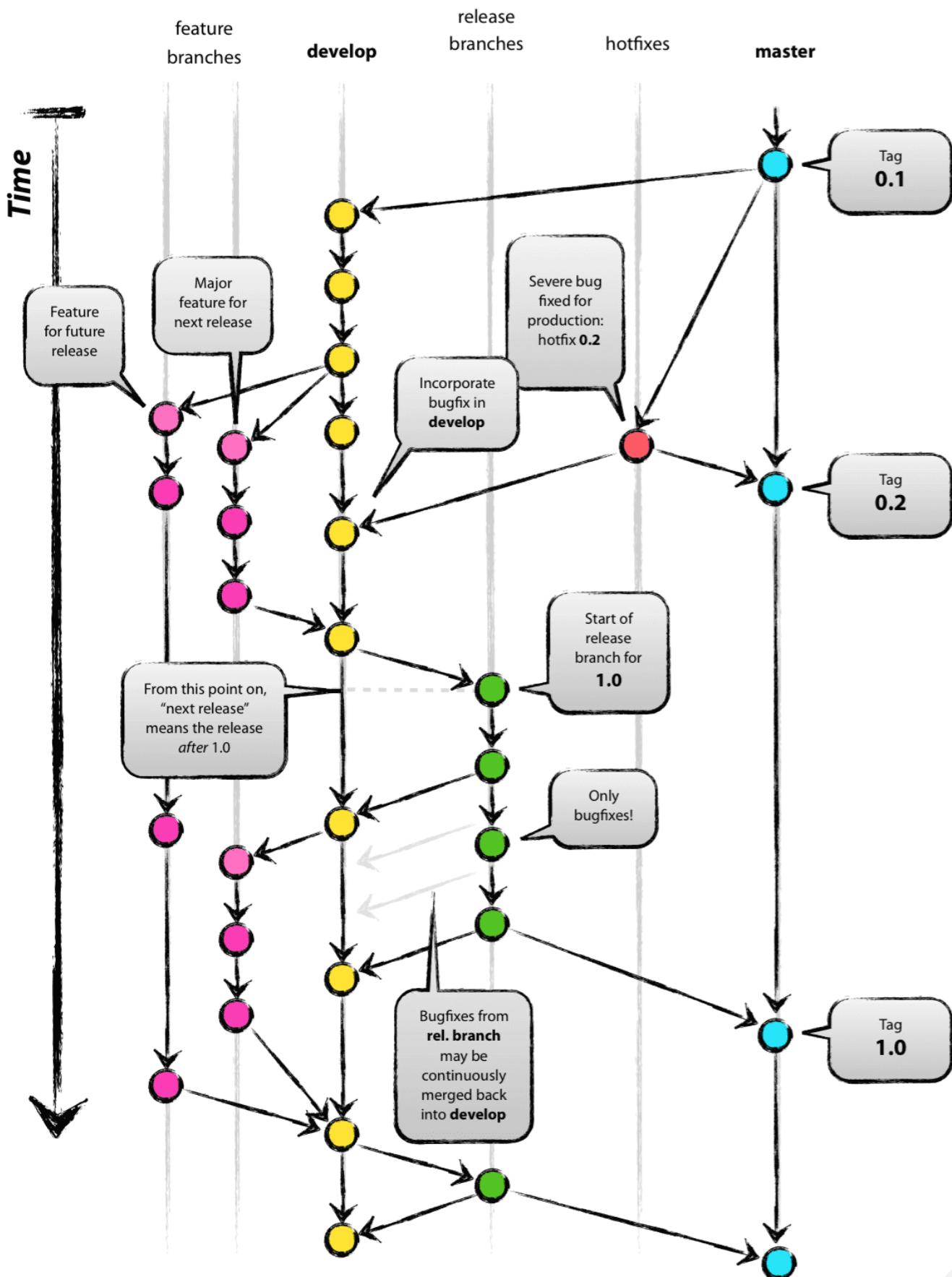
Concepts de git
[Les zones et les états](#)
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

GIT, LE SOCLE DE GITLAB

- Concepts de git
- Les zones et les états
- Les modèles de branches**
- Les branches, des pointeurs
 - Merge Vs. Rebase
 - Pour aller plus loin
 - GIT - TP - Jeux

LES MODÈLES DE BRANCHES

Concepts de git
 Les zones et les états
Les modèles de branches
 Les branches, des pointeurs
 Merge Vs. Rebase
 Pour aller plus loin
 GIT - TP - Jeux



LES MODÈLES DE BRANCHES

Un modèle de branches efficace

- main (anciennement master)
- develop

origin/main (nom alternatif : master) est la branche principale où le code source de HEAD reflète l'état prêt à être déployé **en production**

origin/develop (noms alternatifs : staging / intégration) est la branche où le code source de HEAD reflète les derniers changements livrés **pour la prochaine version**. C'est à partir de cet emplacement que sont compilées les versions quotidiennes.

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

LES MODÈLES DE BRANCHES

Un modèle de branches efficace

Les branches de support :

- les branches pour les **fonctionnalités**
- les branches pour les **versions**
- les branches de **correctifs**

Contrairement aux branches principales, ces branches ont toujours une durée de vie limitée, puisqu'elles seront **effacées** à terme.

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

LES MODÈLES DE BRANCHES

Les branches de fonctionnalité

Concepts de git
 Les zones et les états
Les modèles de branches
 Les branches, des pointeurs
 Merge Vs. Rebase
 Pour aller plus loin
 GIT - TP - Jeux

Peuvent provenir de :

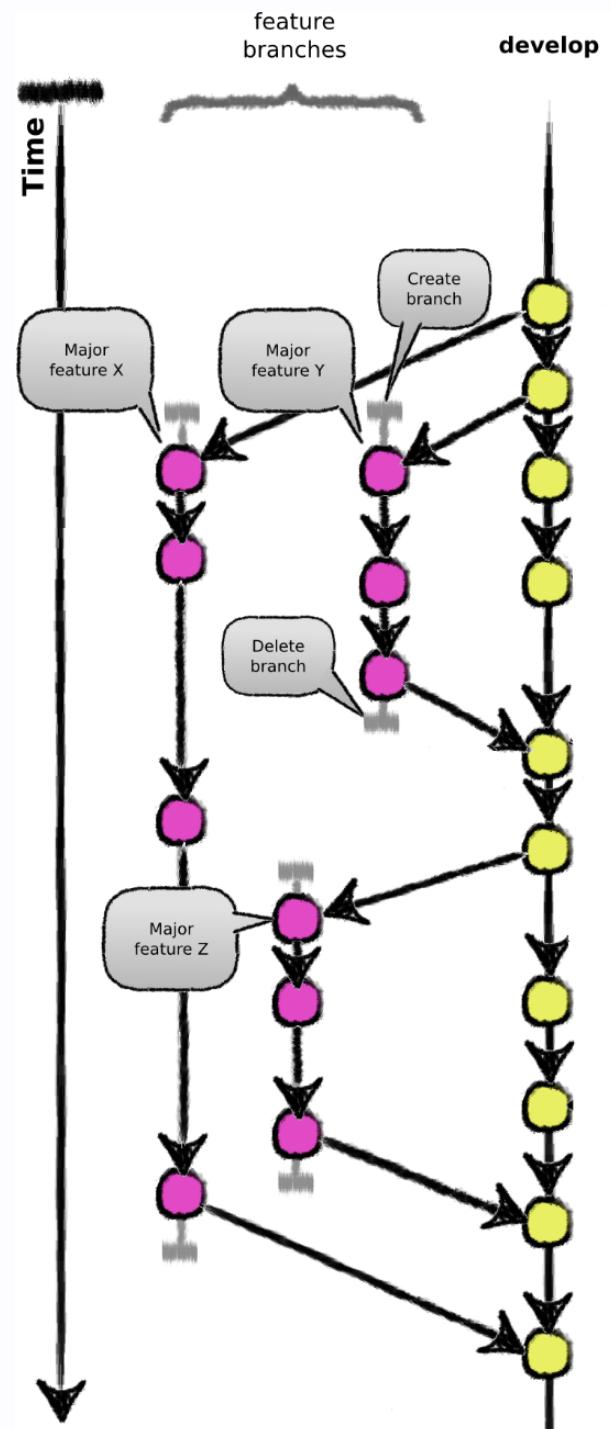
- develop
- une autre branche feature

Doivent être fusionnées dans :

- develop

Convention de nommage :

- feat-



LES MODÈLES DE BRANCHES

Les branches de fonctionnalité

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

```
$ git checkout -b feat-myFeature develop

# Développement de la fonctionnalité (commits)

$ git checkout develop
$ git pull origin develop
$ git merge feat-myFeature

# Résolution d'éventuels conflits

$ git push origin develop
$ git branch -d feat-myFeature
```

LES MODÈLES DE BRANCHES

Les branches de livraison

Concepts de git
 Les zones et les états
Les modèles de branches
 Les branches, des pointeurs
 Merge Vs. Rebase
 Pour aller plus loin
 GIT - TP - Jeux

Peuvent provenir de :

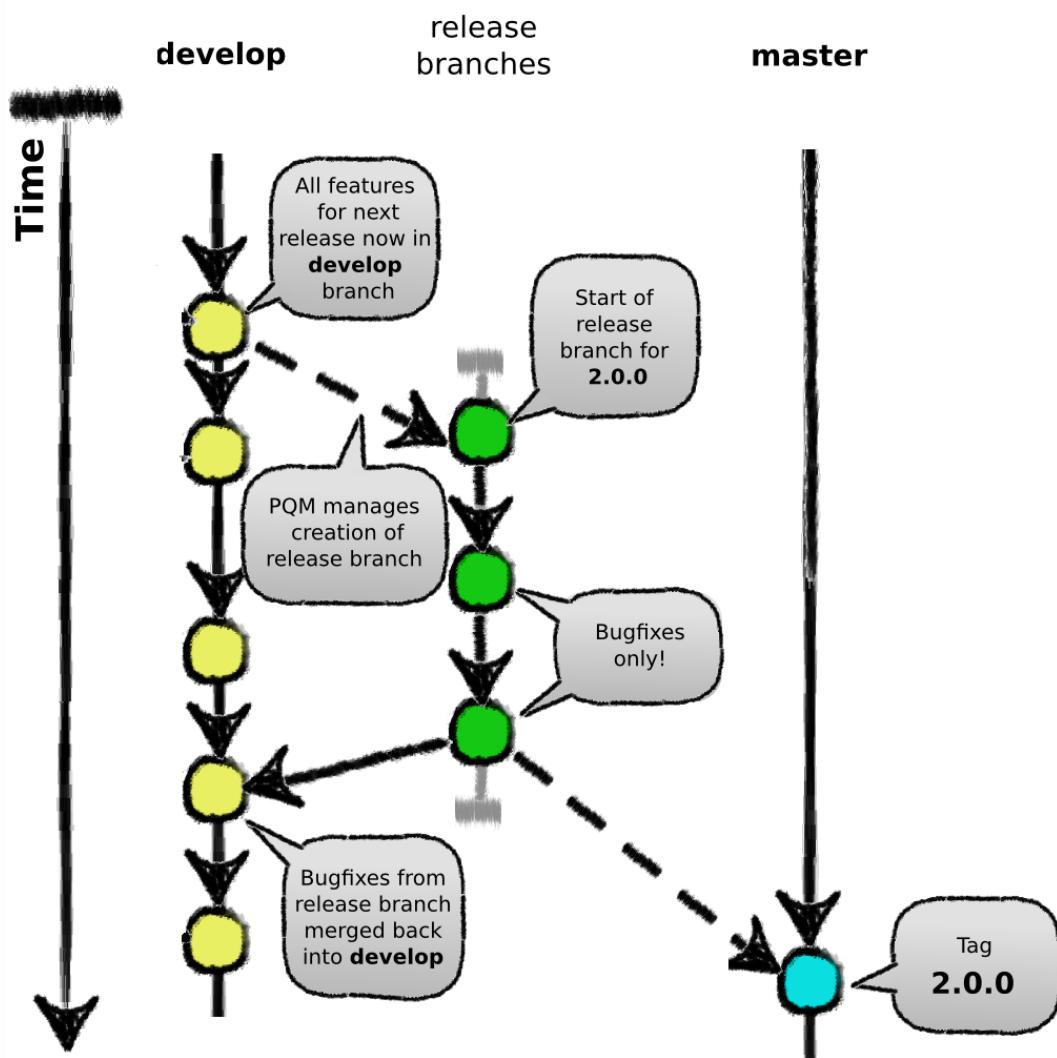
- develop

Doivent être fusionnées dans :

- develop
- main (master)

Convention de nommage de la branche :

- release-



LES MODÈLES DE BRANCHES

Les branches de livraison

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

```
$ git checkout -b release-v1.2 develop

# Mise à jour des scripts/documentation (commits)
$ git checkout main
$ git merge release-v1.2
$ git tag v1.2
$ git push origin main --tags

$ git checkout develop
$ git pull develop
$ git merge release-v1.2
$ git push origin develop

$ git branch -d release-v1.2
```

LES MODÈLES DE BRANCHES

Les branches de correctifs

Concepts de git
 Les zones et les états
Les modèles de branches
 Les branches, des pointeurs
 Merge Vs. Rebase
 Pour aller plus loin
 GIT - TP - Jeux

Peuvent provenir de :

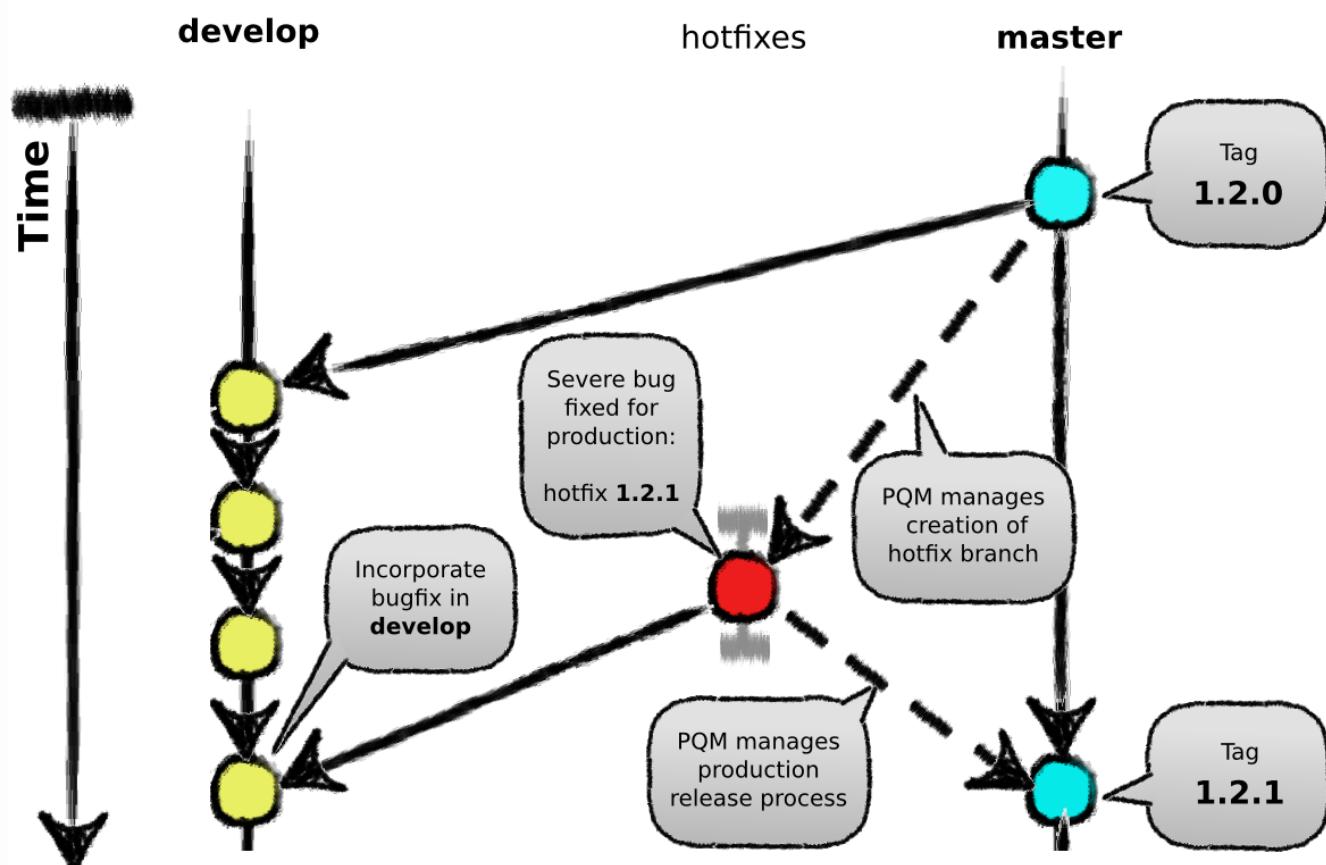
- main (master)

Doivent être fusionnées dans :

- develop
- main (master)

Convention de nommage de la branche :

- hotfix-



LES MODÈLES DE BRANCHES

Les branches de correctifs

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

```
$ git checkout -b hotfix-v1.2.1 main

# Correction de l'anomalie + mise à jour des scripts/documentation (commits)

$ git checkout main
$ git merge hotfix-v1.2.1
$ git tag v1.2.1
$ git push --tags origin main
$ git checkout develop
$ git merge hotfix-v1.2.1
$ git push --tags origin develop
$ git branch -d hotfix-v1.2.1
```

LES MODÈLES DE BRANCHES

Les commandes principales

- Créer une branche locale :

```
git branch <newBranch>
```

- Charger la branche dans le workspace :

```
git checkout <maBranch>
git switch <maBranch>
```

- Merger 2 branches :

```
git merge <otherBranch>
```

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

LES MODÈLES DE BRANCHES

Les commandes principales

- Lister les branches locales :

```
git branch
```

- Supprimer une branche :

```
git branch -d <oldBranch>
```

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

GIT, LE SOCLE DE GITLAB

Concepts de git

Les zones et les états

Les modèles de branches

Les branches, des pointeurs

Merge Vs. Rebase

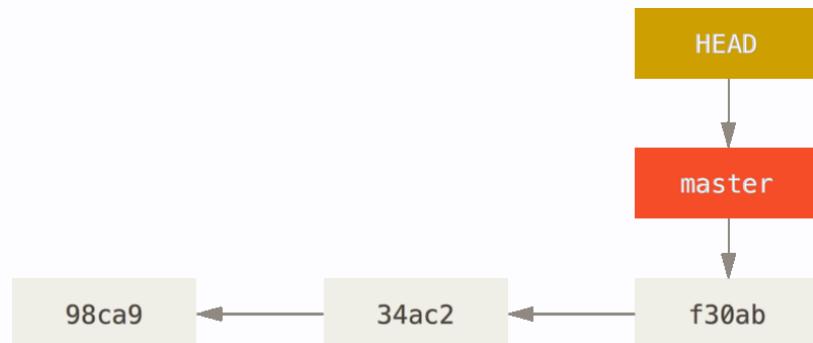
Pour aller plus loin

GIT - TP - Jeux

LES BRANCHES, DES POINTEURS

HEAD, tête de lecture

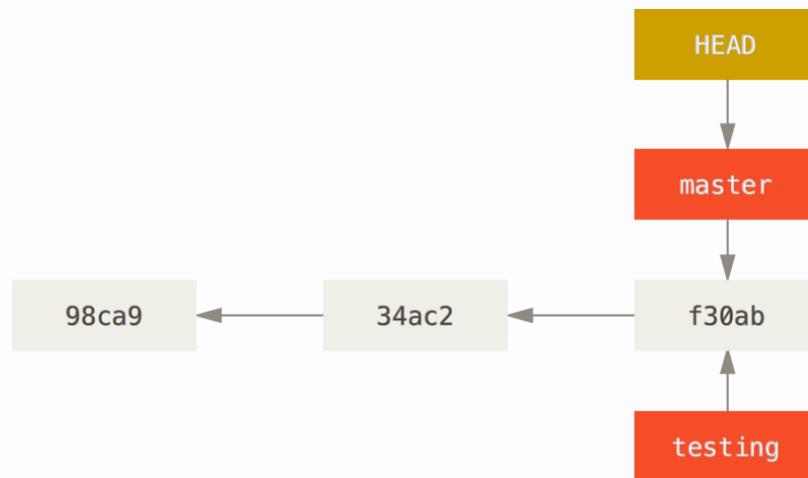
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



LES BRANCHES, DES POINTEURS

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

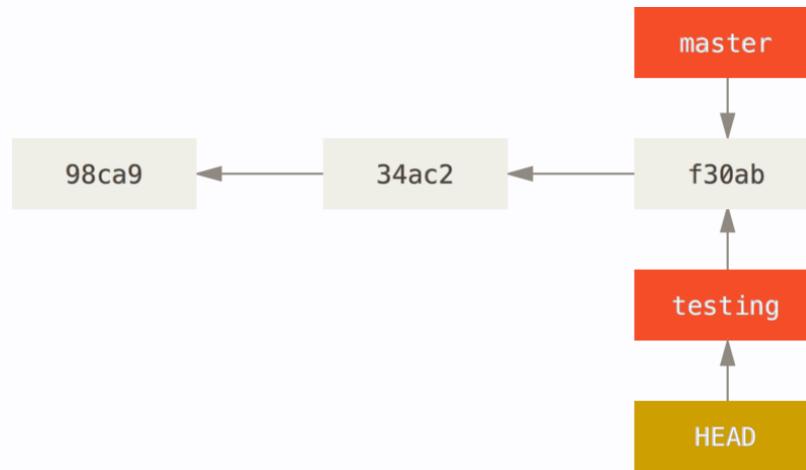
```
$ git branch testing
```



LES BRANCHES, DES POINTEURS

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

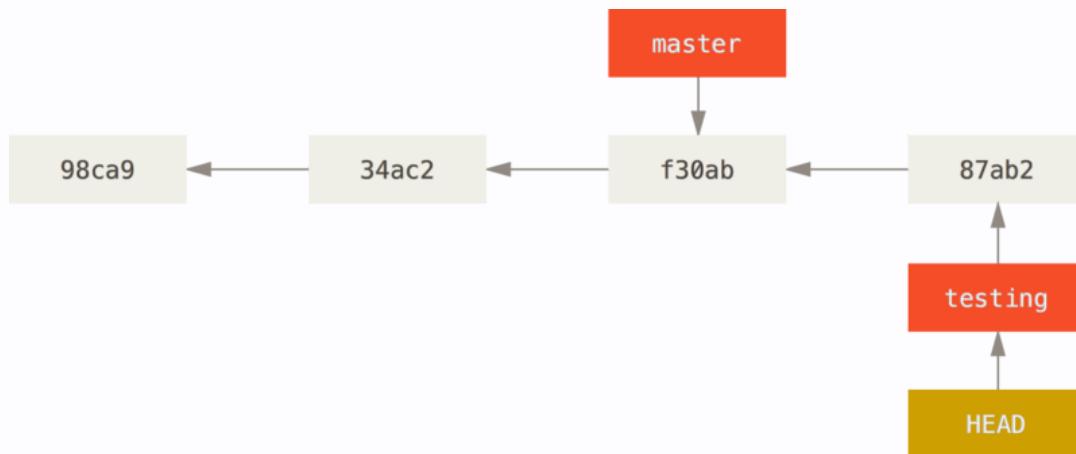
```
$ git checkout testing
```



LES BRANCHES, DES POINTEURS

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

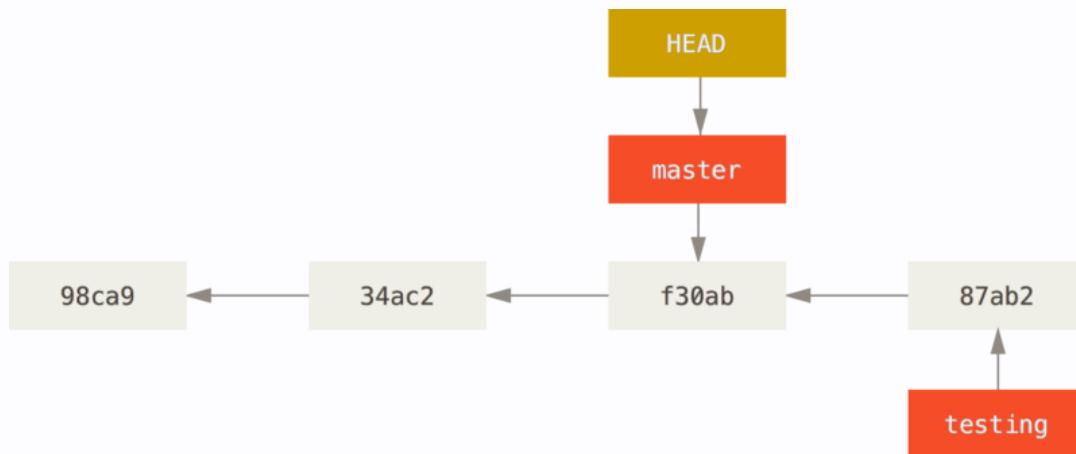
```
$ vim fichier #ici on fait des changements  
$ git commit -m 'changements dans le fichier !'
```



LES BRANCHES, DES POINTEURS

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

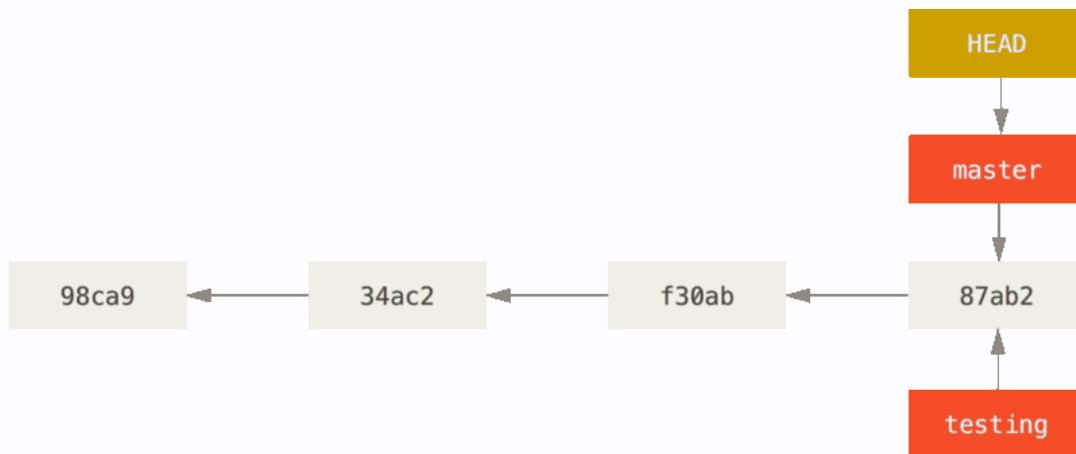
```
$ git checkout master
```



LES BRANCHES, DES POINTEURS

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

```
$ git merge testing
```



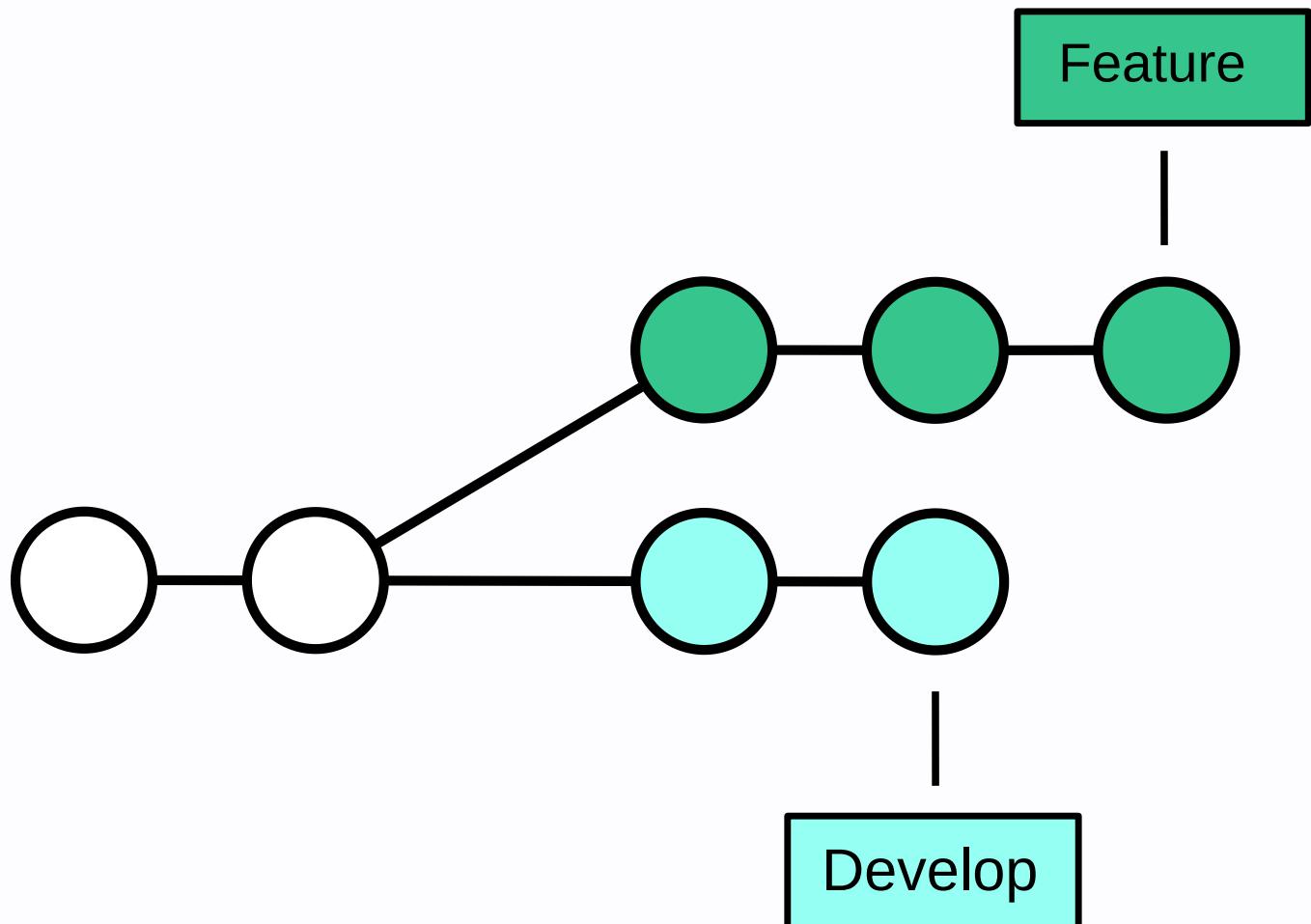
GIT, LE SOCLE DE GITLAB

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

MERGE VS. REBASE

Situation initiale

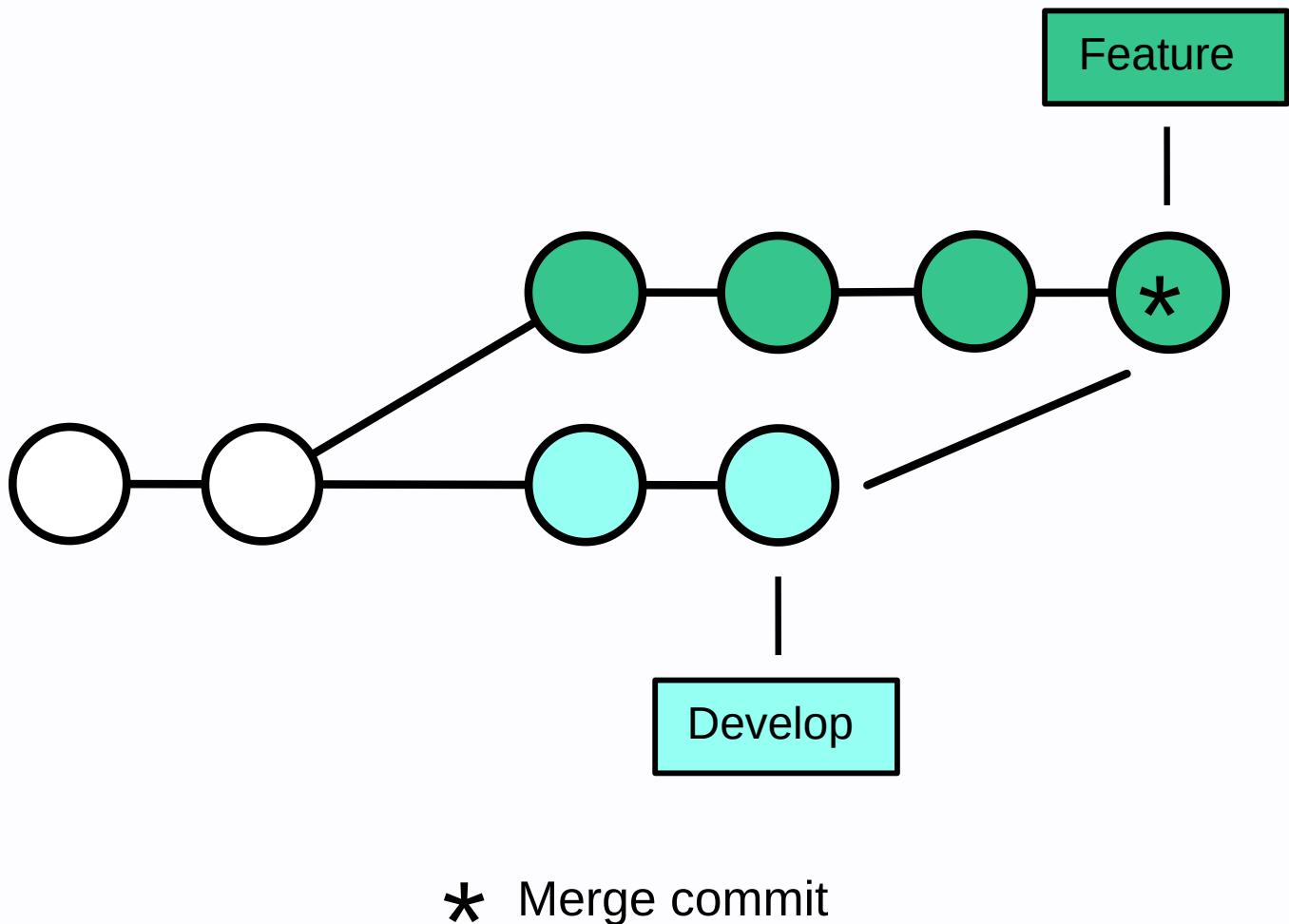
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



MERGE VS. REBASE

Merge

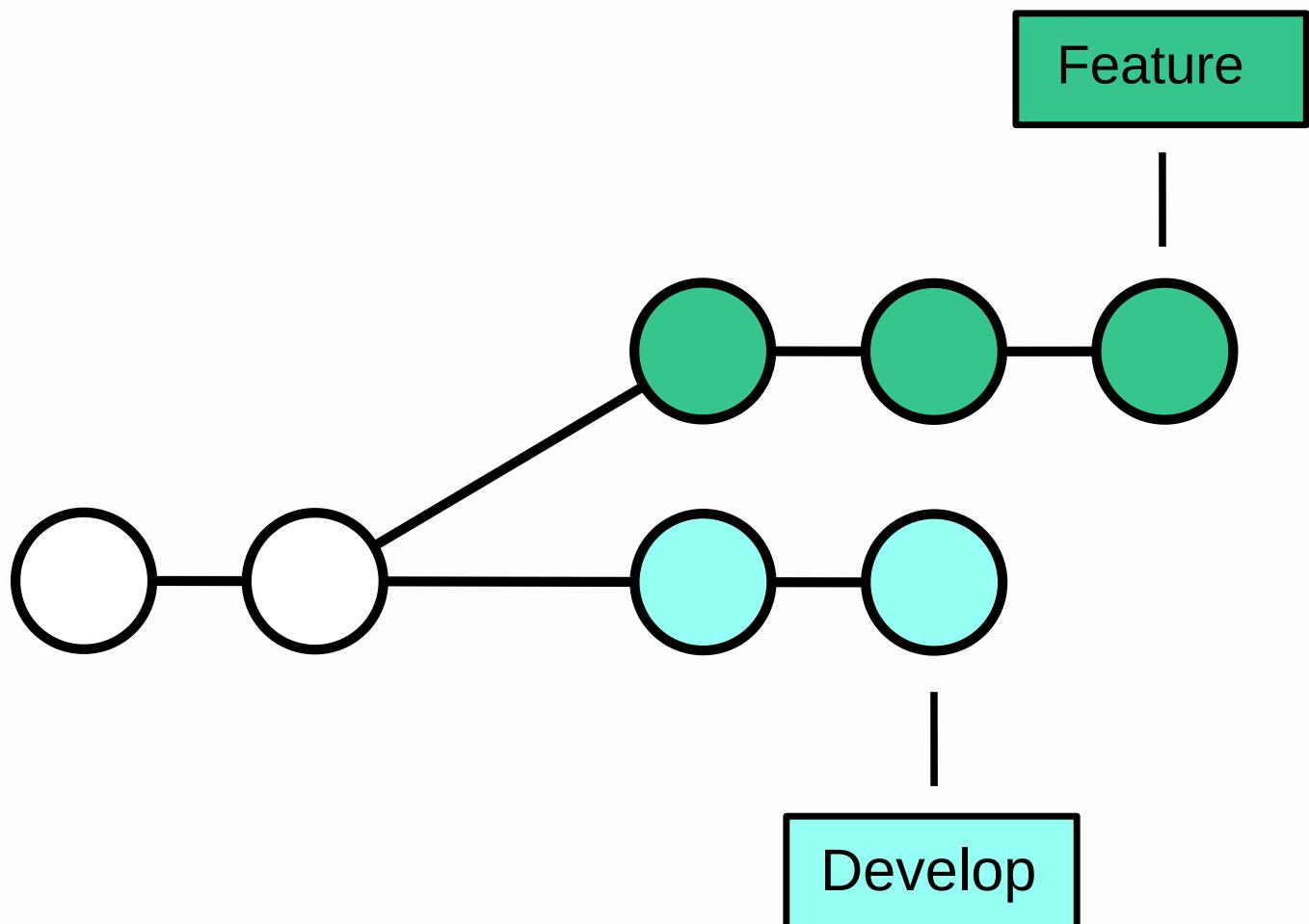
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



MERGE VS. REBASE

Retour à la situation initiale

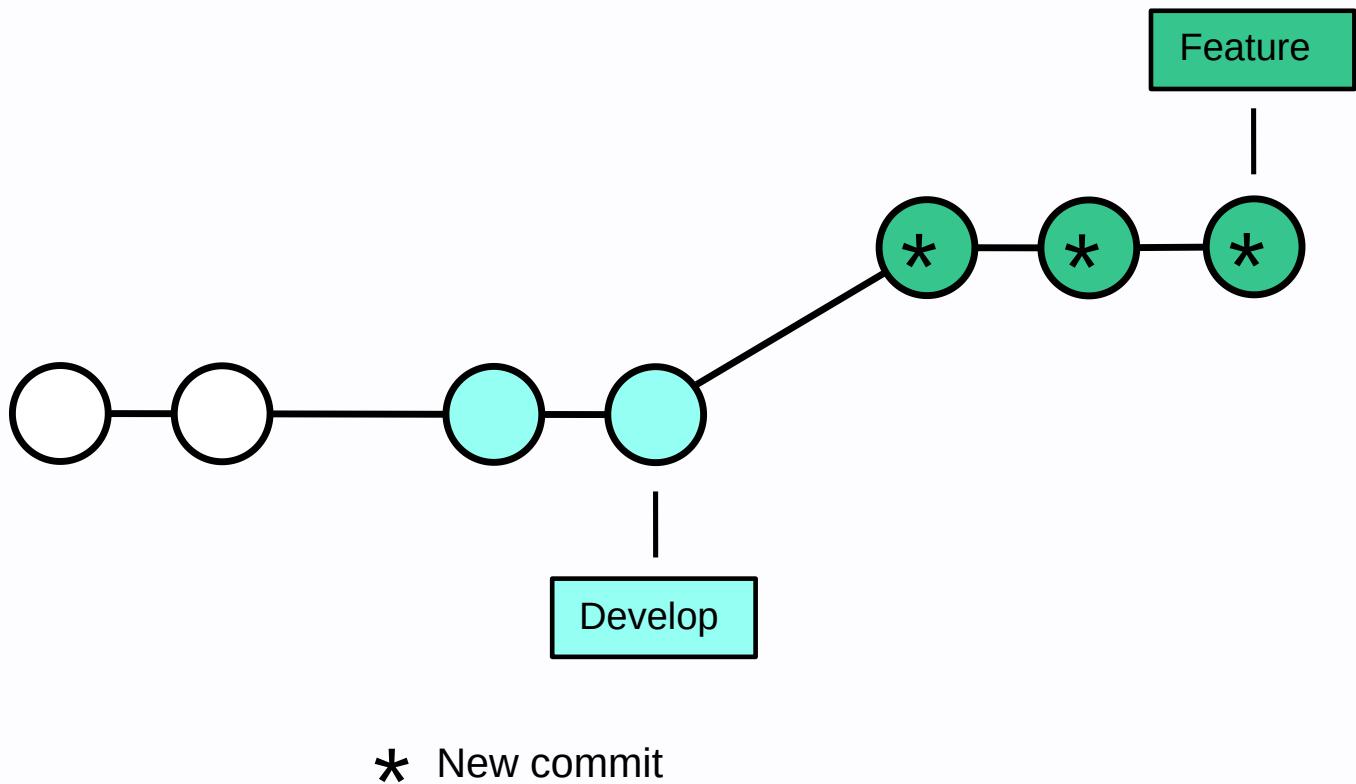
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



MERGE VS. REBASE

Rebase

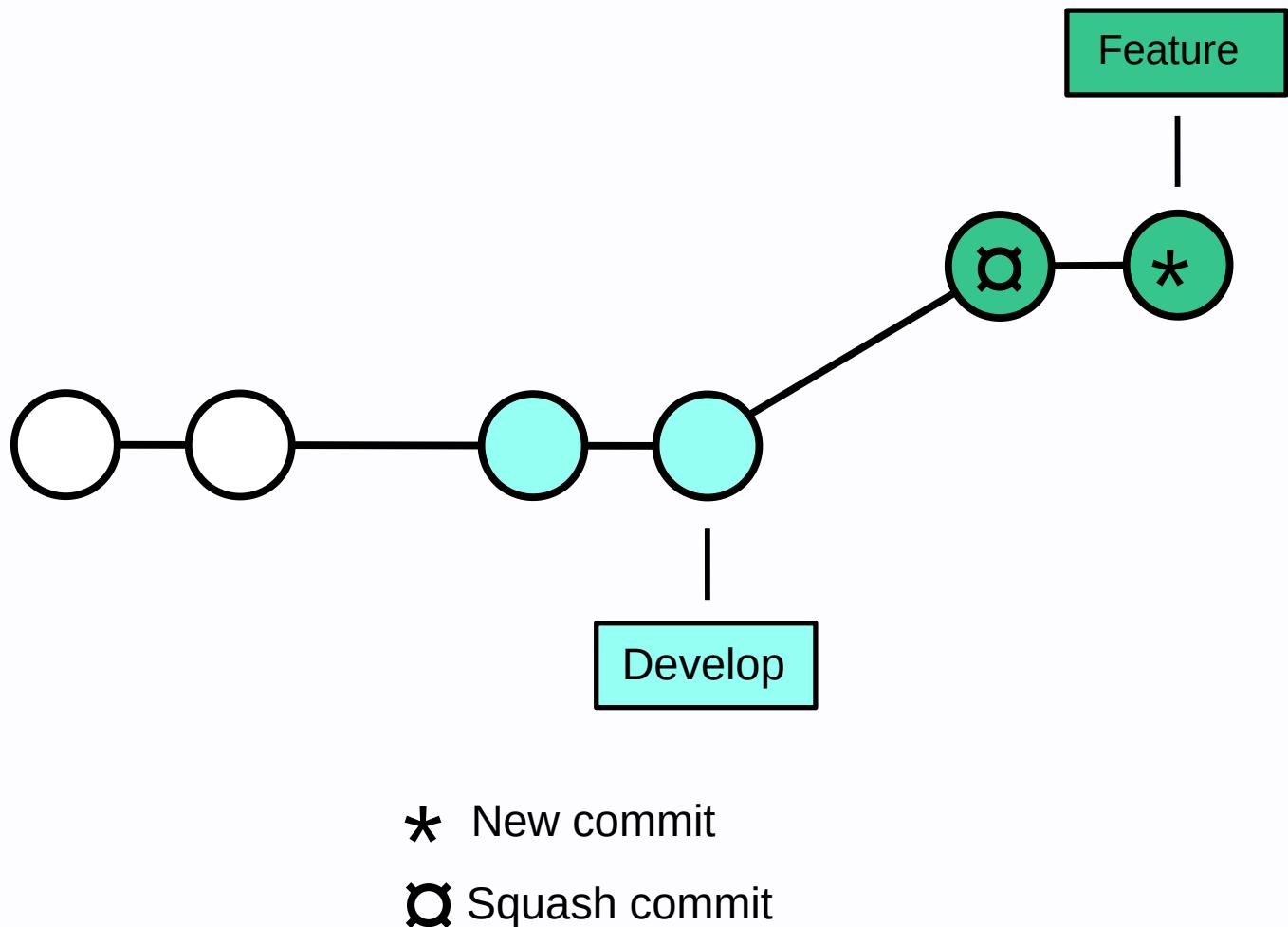
Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



MERGE VS. REBASE

Rebase Interactif

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

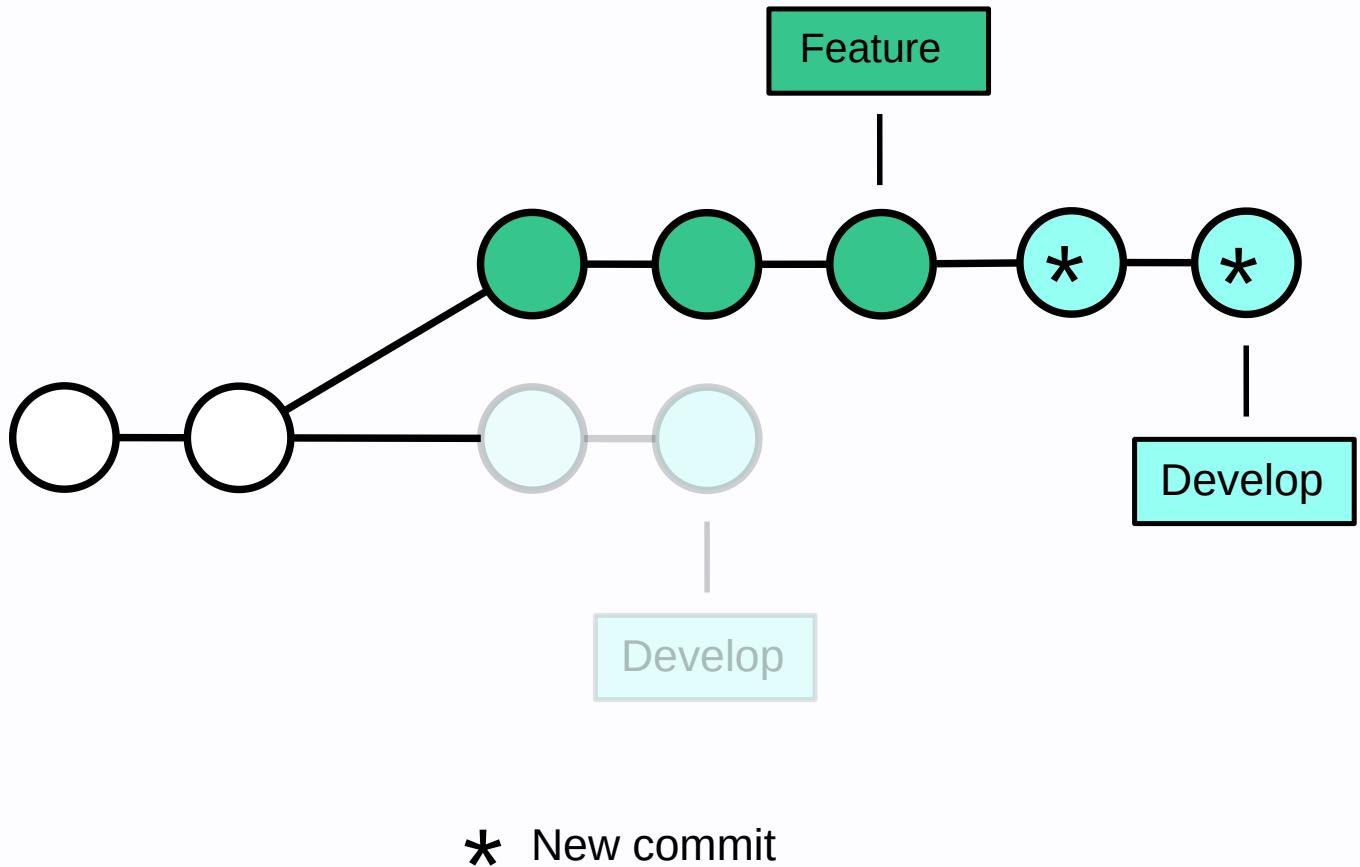


MERGE VS. REBASE

Rebase règle d'or

Ne jamais l'utiliser sur des branches publiques

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



MERGE VS. REBASE

Merge : dans quel cas ?

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

- Pour avoir une branche identifiable dans le graphe
- Pour visualiser une branche « connue », identifiée par l'équipe, le bugtracker ou le gestionnaire de projet (sprint, story, bug...)
- Pour conserver la date de départ de la tâche

MERGE VS. REBASE

Rebase : dans quel cas ?

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

- Pour avoir un graphe linéaire
- Dans le cas d'une branche locale temporaire partant d'une base obsolète
- Pour nettoyer mon historique local et le mettre au propre avant de le partager
- Quand la date de départ de la tâche n'a pas d'importance

MERGE VS. REBASE

Parfois, lorsque l'on merge des branches... Un conflit sauvage apparaît !

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

```
[ETMB]: ~/Documents/test$ git merge master
Auto-merging README.txt
CONFLICT (content): Merge conflict in README.txt
Recorded preimage for 'README.txt'
Automatic merge failed; fix conflicts and then commit the result.
[ETMB]: ~/Documents/test$ git status
# On branch knock-knock
# You have unmerged paths.
#   (fix conflicts and run "git commit")
#
# Unmerged paths:
#   (use "git add <file>..." to mark resolution)
#
#       both modified:      README.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
[ETMB]: ~/Documents/test$ █
```

MERGE VS. REBASE

Le diff

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

```
README.txt      *
1 Hello World!
2
3 Knock, knock.
4 Who's there?
5 Git.
6 Git who?
7 <<<<< HEAD
8 Git this joke over with.
9 =====
10 Git on with the assignment!
11 >>>>> master
12 |
```

MERGE VS. REBASE

Résolution !

- Annuler un merge/rebase :

```
git merge/rebase --abort
```

- Rechercher tous les fichiers qui contiennent le mot TODO dans le code source :

```
git grep "TODO"
```

- Voir qui sont les dernières personnes à avoir modifié les différentes lignes d'un fichier :

```
git blame myFile
```

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

MERGE VS. REBASE

- Voir l'historique d'un fichier :

```
git log -p myFile
```

- Voir les modifications sur un fichier entre deux commits :

```
git diff oldId newId
```

- Voir les détails d'un commit :

```
git show id
```

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

MERGE VS. REBASE

Pour minimiser les conflits :

- Faire des commits réguliers et de petite taille
- Éviter les branches qui vivent trop longtemps (voir si il est possible de répartir la fonctionnalité de cette branche en plusieurs sous-fonctionnalités et donc en plusieurs branches dont la durée de vie sera moins importante)
- Régulièrement se remettre à jour avec la branche distante (avant chaque commit, avec git pull)
- Respecter un système de branche
- Utiliser les bons outils pour vous aider

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

GIT, LE SOCLE DE GITLAB

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

POUR ALLER PLUS LOIN

Ce qu'on n'aborde pas

(... Mais qu'on peut faire avec git !)

- Modifier l'historique
- Mettre ses modifications de côté temporairement
- Récupérer des commits esseulés
- git mergetool
- Résoudre automatiquement des conflits déjà rencontrés
- Trouver un bug avec bisect

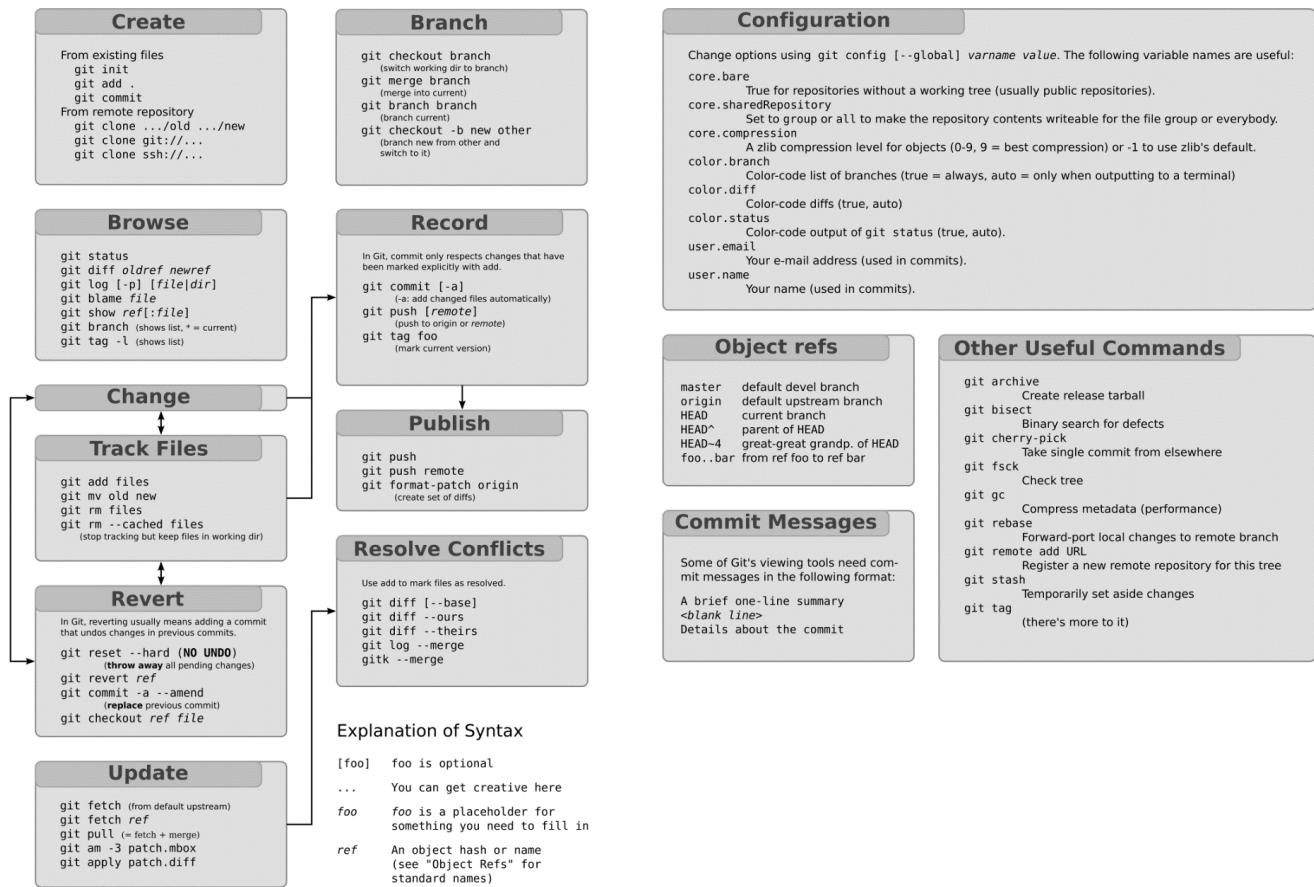
Plein d'autres choses, git est très puissant !

<https://git.goffinet.org/>

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

POUR ALLER PLUS LOIN

Concepts de git
 Les zones et les états
 Les modèles de branches
 Les branches, des pointeurs
 Merge Vs. Rebase
Pour aller plus loin
 GIT - TP - Jeux



GIT, LE SOCLE DE GITLAB

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux

GIT - TP - JEUX

Testons et observons

Concepts de git
 Les zones et les états
 Les modèles de branches
 Les branches, des pointeurs
 Merge Vs. Rebase
 Pour aller plus loin
GIT - TP - Jeux

<https://onlywei.github.io/explain-git-with-d3/>

Basic Commands	Undo Commits	Combine Branches	Remote Server
<code>git commit</code>	<code>git reset</code>	<code>git merge</code>	<code>git fetch</code>
<code>git branch</code>	<code>git revert</code>	<code>git rebase</code>	<code>git pull</code>

We are going to skip instructing you on how to add your files for commit in this explanation. Let's assume you already know how to do that. If you don't, go read some other tutorials.

Pretend that you already have your files staged for commit and enter `git commit` as many times as you like in the terminal box.

```
Type git commit a few times.

$ enter git command
```

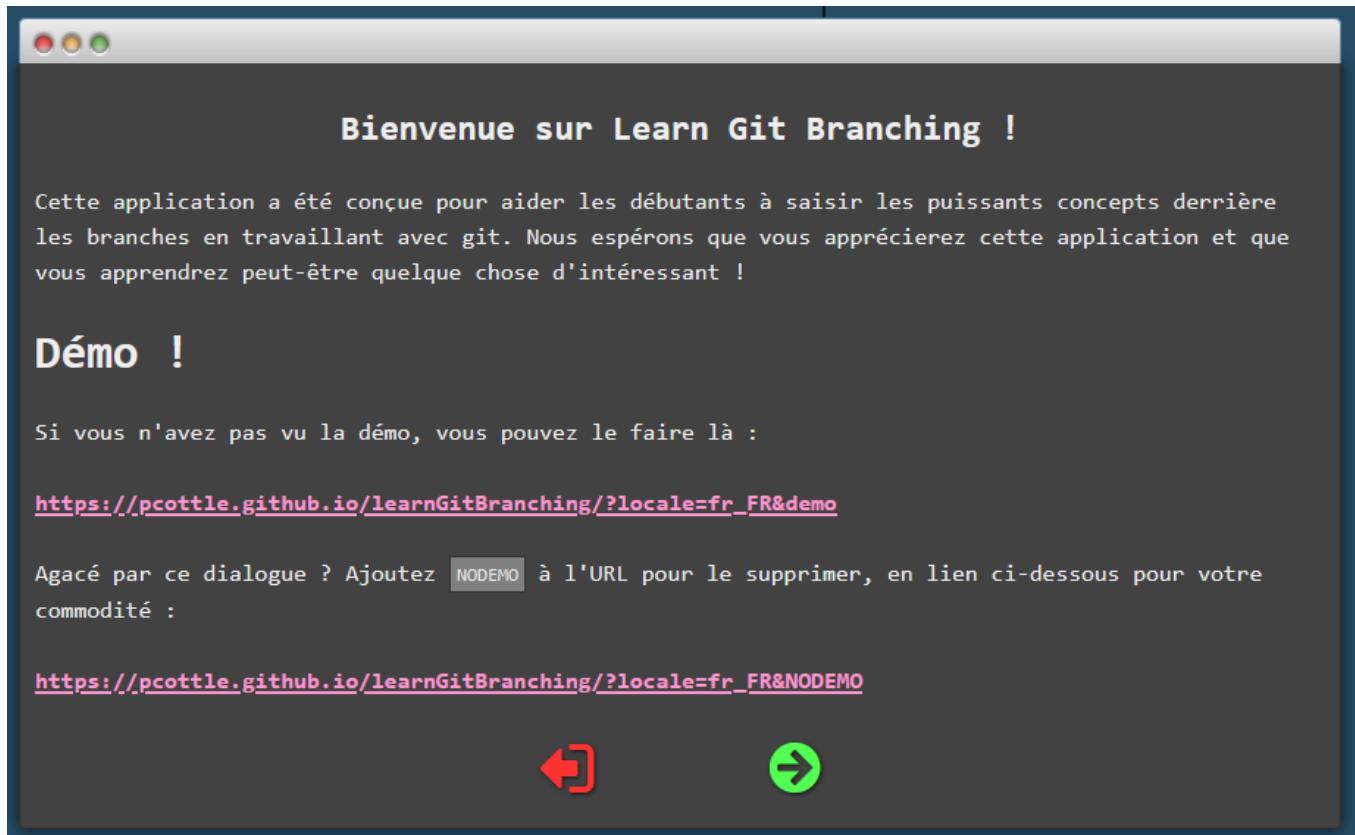
Local Repository
 Current Branch: master

GIT - TP - JEUX

Learn Git Branching

<https://learngitbranching.js.org>

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
GIT - TP - Jeux



GIT - TP - JEUX

Oh My Git

<https://ohmygit.org/>

Concepts de git
Les zones et les états
Les modèles de branches
Les branches, des pointeurs
Merge Vs. Rebase
Pour aller plus loin
[GIT - TP - Jeux](#)



GIT, LE SOCLE DE GITLAB

Dans ce chapitre nous avons :

- Vu les principes de base de git.
- Découvert le concept de zone et d'état de fichier.
- Découvert le concept de branches et de modèle de branches.
- Découvert que les branches ne sont que des pointeurs.
- Compris la différence entre un merge et un rebase
- Découvert tout ce qu'il reste à découvrir à propos de git !
- Fait le tour des fonctionnalités de base de git grâce à des jeux !

CHAPITRE 2

DOCKER, LE PORTE-CONTENEURS

DOCKER, LE PORTE-CONTENEURS

- **Docker** nous présentera ce qu'est Docker.
- **Architecture de Docker** nous présentera comment fonctionne Docker et le vocabulaire associé.
- **Les concepts de Docker** nous présentera les concepts principaux de Docker.
- **Docker - TP Bases & Dockerfile** nous fera manipuler les concepts basiques de Docker.
- **Docker - TP Réseau & Volumes** Nous fera manipuler les réseaux et les volumes avec Docker.

DOCKER, LE PORTE-CONTENEURS

Docker

- Architecture de Docker
- Les concepts de Docker
- Docker - TP Bases & Dockerfile
- Docker - TP Réseau & Volumes

DOCKER

En quelques mots

L'utilisation de containers a de nombreux avantages :

- **Encapsuler** les applications
- **Virtualisation** légère
- Consommation **faible** de ressources
- Accélération du **déploiement** des applications
- Amélioration de la **portabilité** des applications
- Gestion des ressources **simplifiées**

Docker

Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER

Comment ça a commencé

- dotCloud, fondée par Solomon Hykes {Epitech, 2006} en France en 2008 puis relancée à San Francisco en 2010
- Première release open source de Docker par dotCloud en mars 2013
- Déjà en 2014 :
 - 300 000 pulls
 - 18 600 stars sur github
 - 740 contributeurs significatifs
 - 300 projets construits sur Docker
- Des milliers d'applications « Dockerisées »
- Intégré avec... Tout ?

Docker

Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER

Aujourd'hui

- 318 milliards de "**pulls**"
- 7,3 millions d'utilisateurs
- Un écosystème qui continue de croître

(source : <https://www.docker.com/company/>)

Docker

Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER

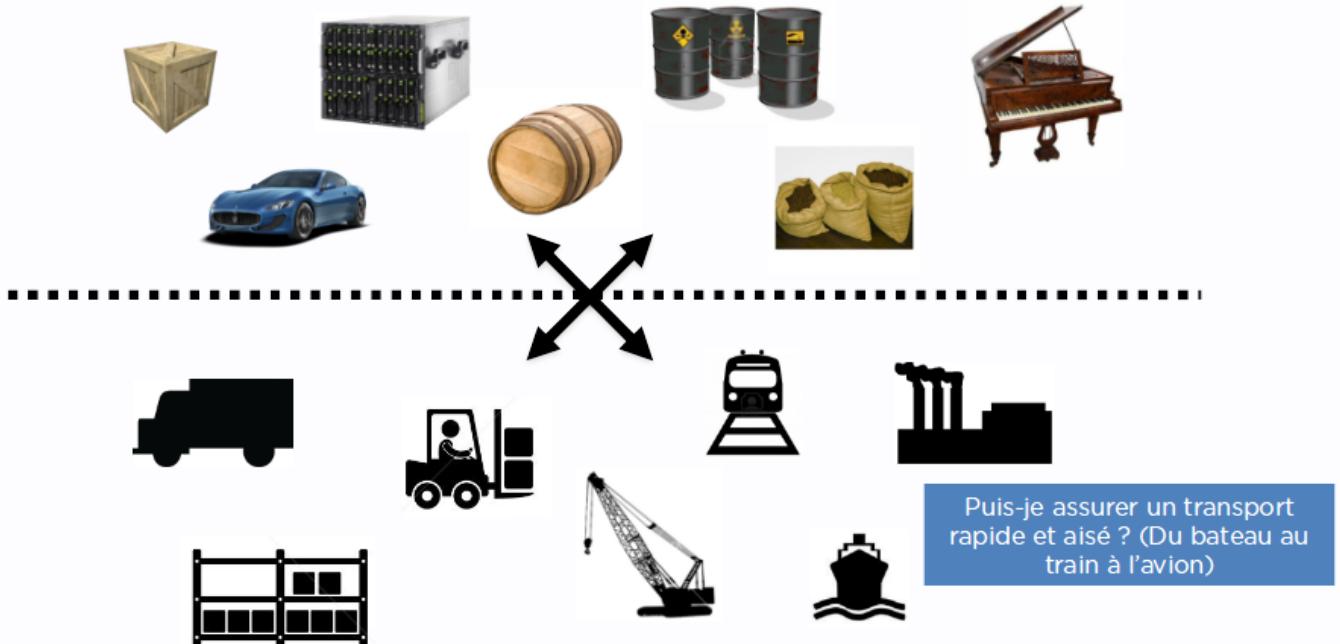
Pourquoi cet engouement ?

Docker

Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

Multiplicité des biens

Les biens doivent-ils interagir ?
(les bananes à côté des épices)



DOCKER

Docker

Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

Une matrice de l'enfer !

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?

DOCKER

Solution !

Docker

Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes



DOCKER, LE PORTE-CONTENEURS

Docker

Architecture de Docker

Les concepts de Docker

Docker - TP Bases & Dockerfile

Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

Client - Serveur

L'utilisation de containers a de nombreux avantages :

- Docker Engine : le Docker serveur au centre de l'architecture
- Client : CLI pour interagir avec le serveur
- Des applications avec leurs environnement complet
- Image : image de binaires et de librairies pour exécuter des applications
- Container : une instance d'une image en exécution
- Registry : bibliothèque d'images

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

Docker Engine

- C'est le serveur au centre de l'architecture
- Il permet d'exécuter les conteneurs
- C'est un « **daemon** »
- Il utilise les namespaces du noyau Linux et les cgroups
- Les namespaces garantissent l'isolation des conteneurs

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

Le client

- Le client Docker se connecte au Docker Engine
- Il peut être installé sur la même machine
- Ou sur une machine différente
- Il y a deux types de clients
 - CLI
 - GUI

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

Images

- Modèles en **lecture seule** utilisés pour créer les conteneurs
- Construites par vous ou d'autres utilisateurs
- Stockées dans **Docker hub** ou votre propre registre
- Basées sur une ou plusieurs images

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

Docker Hub

- Permet de centraliser les images
- Configuration par défaut
- Permet d'explorer les images
- Permet de récupérer des images
- Permet de stocker ses propres images

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

Conteneurs

- Espace d'exécution d'application isolé
- Contient tout ce qui est nécessaire à l'exécution
 - Binaires de l'application
 - Bibliothèques requises par l'application

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

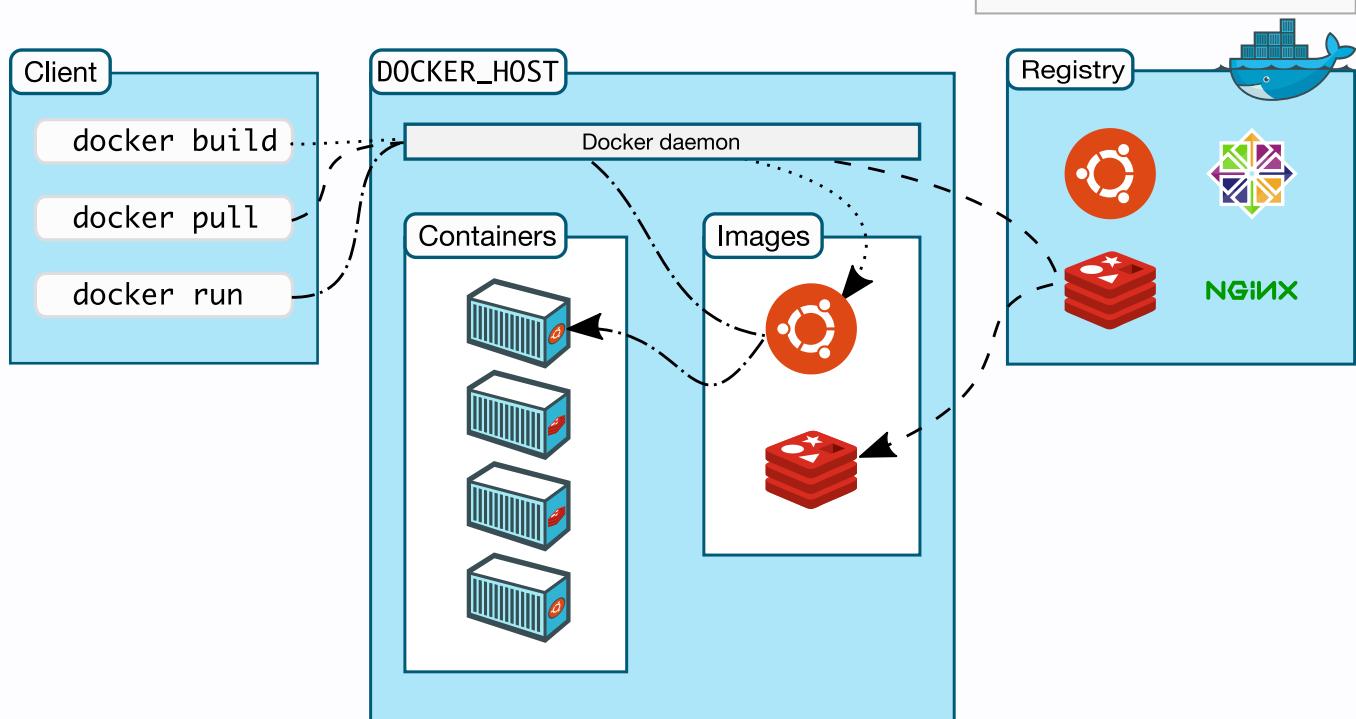
Registre

- Stockage des images
 - Registre local
 - Registre distant
- Accès aux images
 - Registre privé
 - Registre public

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

ARCHITECTURE DE DOCKER

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes



ARCHITECTURE DE DOCKER

Orchestration

- Automatisation du déploiement
 - Mise à l'échelle
 - Gestion
 - Mise en réseau
- De multiples solutions :
 - Docker swarm mode
 - Docker compose
 - Kubernetes

Docker
[Architecture de Docker](#)
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER, LE PORTE-CONTENEURS

Docker

Architecture de Docker

Les concepts de Docker

Docker - TP Bases & Dockerfile

Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

La vie d'un conteneur

- Le conteneur est porté par le processus avec lequel il est lancé
- La commande qui permet de lancer le conteneur à partir d'une image est `docker run`
- La commande qui permet de lister les conteneurs qui sont en train d'être exécutés est `docker ps`

Docker
Architecture de Docker
[Les concepts de Docker](#)
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

Options importantes de run

- L'option " -i " indique qu'on souhaite se connecter à stdin
- L'option " -t " permet d'obtenir un terminal
- L'option " -d " permet de lancer le conteneur en arrière-plan
- L'option " -p " permet de mapper un port réseau sur l'hôte
- L'option " -v " permet de lier un volume

Docker
Architecture de Docker
[Les concepts de Docker](#)
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

Commandes utiles

- `docker logs` : permet d'afficher les logs d'un conteneur qu'on lui passe en paramètre
- `docker exec` : permet d'exécuter un processus supplémentaire à l'intérieur d'un conteneur
 - Très utile pour "aller voir" à l'intérieur d'un conteneur et comprendre ce qui s'y passe
- `docker stop` : permet d'arrêter un conteneur (SIGTERM)
- `docker kill` : permet de forcer l'arrêt d'un conteneur (SIGKILL)
- `docker inspect` : permet d'obtenir des informations sur le conteneur qu'on lui passe en paramètre
- `docker rm` : permet de supprimer un conteneur, avec l'option " -f " ça permet de le faire quand il tourne

Docker
Architecture de Docker
[Les concepts de Docker](#)
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

Les images Docker

- Une succession de couches qui sont elles mêmes des images
- Copy On Write
- Couche supérieure en écriture, le reste en lecture seule
- On peut les créer
 - A la main: `docker commit`
 - A partir d'un Dockerfile: `docker build`

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

Dockerfile

- Contient les instructions qui permettre de construire l'image
 - L'image de base
 - Les programmes à installer, les modifications à effectuer sur l'image de base
 - La commande à exécuter au lancement

Docker
Architecture de Docker
[Les concepts de Docker](#)
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

Le build

- Système de cache
 - Importance de l'ordre des instructions dans le Dockerfile
 - Le moindre changement annule le cache à partir de cette étape là
- Contexte de build : répertoire où se trouve le Dockerfile
 - Envoyé au docker daemon
 - Utilisé pendant le build pour construire l'image

Docker
Architecture de Docker
[Les concepts de Docker](#)
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

Le réseau

- Un réseau de type bridge est créé par défaut, représenté par une interface sur l'hôte.
- Pour créer un réseau de type bridge :

```
docker network create <network name>
```

- Un DNS est intégré au démon docker : ça permet aux conteneurs de communiquer en utilisant leur nom lorsqu'ils sont sur le même bridge
- Pour qu'un conteneur soit accessible à l'extérieur, il faut exposer les ports du conteneur et les mapper sur des ports de l'hôte

Docker
Architecture de Docker
[Les concepts de Docker](#)
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

LES CONCEPTS DE DOCKER

Mapping des ports

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

- Manuel :

- `docker run -p [host port]:[container port] <image>`
- On peut utiliser plusieurs fois l'option `-p` pour mapper plusieurs ports
- On peut visualiser le mapping avec `docker port <conteneur>`

- Automatique :

- `docker run -P <image>`
- Mappe automatiquement les ports exposés dans le conteneur sur un numéro de port de l'hôte

LES CONCEPTS DE DOCKER

Les volumes

- Un répertoire dans un conteneur
- Dédié à la persistance des données indépendamment du cycle de vie du conteneur
 - Persistant si conteneur supprimé
 - Peut être mappé sur un répertoire sur le hôte
 - Pas affecté par le COW
 - Si on crée une image à partir d'un conteneur, le contenu des volumes n'est pas intégré à l'image

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

Les volumes, commandes importantes

- Créer un volume
 - `docker volume create [--name nom_volume]`
- Lister les volume
 - `docker volume ls`
- Lancer le conteneur avec un volume
 - `docker run -it -v monvolume:/www/monvolume ubuntu bash`
- Obtenir les infos sur le volume
 - `docker volume inspect monvolume`
- Supprimer le volume (uniquement si inutilisé)
 - `docker volume rm monvolume`

LES CONCEPTS DE DOCKER

Volumes d'hôte

- Lors de l'exécution d'un conteneur, vous pouvez mapper des dossiers de l'hôte sur un volume
- Les fichiers du dossier hôte seront présents dans le volume
- Les modifications d'un côté affectent l'autre
- S'il n'existe pas, le chemin sera créé
- Si le dossier existe, les fichiers sont remplacés par ceux de l'hôte

Docker
Architecture de Docker
[Les concepts de Docker](#)
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER, LE PORTE-CONTENEURS

Docker

Architecture de Docker

Les concepts de Docker

Docker - TP Bases & Dockerfile

Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

Installation

- Vérifiez si Docker est installé
- Démarrez le démon Docker s'il ne l'est pas
- Affichez la version de Docker

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

Utilisation de Docker Hub

- Téléchargez la dernière image **ubuntu** à partir du Hub
- Listez les images présentes localement

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

Exécuter un conteneur

- Exécutez un conteneur à partir de l'image **ubuntu**
- Listez les conteneurs en exécution
- Listez tous les conteneurs

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

Accéder au terminal d'un conteneur

- Créez un conteneur à l'aide de l'image **ubuntu** et connectez-vous au terminal
- Créez un fichier dans le conteneur puis sortez du conteneur
- Relancez la commande run écrite au premier point de cette diapo
Qu'est-il arrivé au fichier ?

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

Exécution en mode détaché

- Lancer un conteneur basé sur l'image `rockylinux/rockylinux` en exécutant la commande `ping 127.0.0.1 -c 60` en mode détaché
- Listez les conteneurs
- Attendez quelques secondes et listez à nouveau les conteneurs
Que s'est il passé ?

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

La commande exec

- Exécutez un conteneur en arrière plan en lançant **bash**
- Exédez la commande **ps -ef**
 - à l'intérieur du conteneur
 - depuis l'extérieur
- Que constatez-vous ?
- Observez les PPID : que constatez-vous ?

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

La commande logs

- Exécutez un conteneur en arrière-plan
- Récupérez l'id du conteneur et consultez les logs
- Consultez les logs en ajoutant l'option -f
- Consultez les logs en ajoutant l'option -f et l'option --tail 10

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

La commande inspect

- Testez la commande inspect, lisez les différentes informations que vous obtenez

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

Supprimer un conteneur

- Listez les conteneurs arrêtés à l'aide de l'option --filter
- Supprimez un conteneur arrêté
- Supprimez tous les conteneurs arrêtés

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP BASES & DOCKERFILE

Dockerfile

- Créez un fichier Dockerfile dans un nouveau répertoire
- Remplissez le Dockerfile pour qu'il utilise [rockylinux/rockylinux](#) comme image de base, qu'il y ait une mise à jour de la distribution et que wget soit installé
- Construisez la nouvelle image
- Listez les images
- Modifiez le Dockerfile pour ajouter l'installation d'un nouveau paquet, zip
- Construisez la nouvelle image
Observez l'utilisation du cache
- Visualisez l'historique de la construction des images
- Modifiez le fichier Dockerfile pour que les deux installations soient faites avec la même commande
Que remarquez-vous lors de la construction ?

Docker
Architecture de Docker
Les concepts de Docker
[Docker - TP Bases & Dockerfile](#)
Docker - TP Réseau & Volumes

DOCKER, LE PORTE-CONTENEURS

Docker

Architecture de Docker

Les concepts de Docker

Docker - TP Bases & Dockerfile

Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

Réseau par défaut

- Lancez un conteneur en arrière-plan
- Utilisez la commande suivante pour regarder le réseau bridge

```
[root@centos1 ~]# docker network inspect bridge
```

- Lancez un autre conteneur et testez le réseau à l'aide de la commande ping et des adresses IP des conteneurs

DOCKER - TP RÉSEAU & VOLUMES

Création d'un bridge alternatif

- Créez un bridge
- Lancez plusieurs conteneurs nommés puis testez le réseau sur le nouveau bridge

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Connexion à plusieurs réseaux

- Lancez un nouveau conteneur nommé (à l'aide de l'option `--name`) en mode interactif et essayez de « pinguer » le précédent
- Connectez-vous au même réseau, retestez le ping

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Le mapping de ports

- Lancez un conteneur basé sur l'image **nginx** avec un mapping manuel
- Listez les conteneurs
- Regardez les infos sur les ports avec la commande adaptée
- Lancez un conteneur basé sur l'image **nginx** avec un mapping automatique
- Sur quel port le port 80 de nginx est-il mappé ?
- Créez une nouvelle image qui expose directement le port 80 automatiquement.

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Création des volumes

- Créez un volume **vol1**
- Listez les volumes
- Montez le **vol1** dans un conteneur ubuntu
- Créez un fichier dans un conteneur ubuntu
- Quittez le conteneur avec **exit**
- Créez une image à partir du conteneurs
- Lancez un conteneur à partir de la nouvelle image.
Que remarquez-vous ?

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Utilisation des volumes

- Listez les volumes
 - Affichez les caractéristiques d'un volume
 - Vérifiez la persistance des données dans le volume
 - Lancez un nouveau conteneur et accédez au même volume, créez un autre fichier
 - Passez le conteneur en arrière plan puis connectez vous au premier en utilisant la commande **exec**
- Qu'observez-vous ?

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Suppression des volumes

- Listez les volumes
- Supprimez le volume vol1

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Montez un volume host

- Montez un volume host, par exemple un dossier que vous aurez créé et dans lequel vous aurez placé un fichier
- Montez un volume pour les logs d'un conteneur nginx
- Accédez au serveur à l'aide de la commande **curl**
- Inspectez le volume des logs
- Regardez dans le dossier sur l'hôte

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER - TP RÉSEAU & VOLUMES

Les volumes dans un Dockerfile

- Testez l'instruction **VOLUME** dans un Dockerfile.
- Lancez un conteneur à partir de l'image ainsi construite
- Trouvez les répertoires correspondants aux volumes sur l'hôte

Docker
Architecture de Docker
Les concepts de Docker
Docker - TP Bases & Dockerfile
Docker - TP Réseau & Volumes

DOCKER, LE PORTE-CONTENEURS

Dans ce chapitre nous avons :

- Découvert Docker.
- Découvert l'architecture de Docker.
- Vu les concepts principaux de Docker.
- Découvert comment commencer à manipuler Docker.
- Découvert comment commencer à manipuler les réseaux et les volumes avec Docker.

CHAPITRE 3

GITLAB UN OUTIL DEVOPS

GITLAB UN OUTIL DEVOPS

- **DevOps** nous expliquera les grands principes derrière le mot DevOps
- **Github - Gitlab** nous introduira ce que sont ces plateformes
- **Introduction** nous présentera Gitlab
- **Introduction - TP** nous permettra de découvrir Gitlab.com
- **Installation** nous détaillera l'installation de Gitlab
- **Installation - TP** nous apprendra à installer Gitlab
- **Administration** nous présentera les différents aspects de l'administration de Gitlab
- **Administration - TP** nous permettra de tester quelques aspects de l'administration de Gitlab
- **Gestion des dépôts** nous montrera comment gérer les dépôts git

GITLAB UN OUTIL DEVOPS

DevOps

Github - Gitlab

Introduction

Introduction - TP

Installation

Installation - TP

Administration

Administration - TP

Gestion des dépôts

DEVOPS

DevOps, quésaco ?

- Une philosophie (!= métier)
- Un but : réduire le time to market
- Une extension de l'agilité
- Mise en avant de la collaboration
- Détruire les silos
- DevOps, DataOps, DevSecOps etc...

DevOps

Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

DEVOPS

Petite histoire du DevOps

- ~2000 : le web prend de l'ampleur
- 2003 : Google crée le terme SRE (Site Reliability Engineering)
- 2009 : le terme DevOps apparaît
- 2014 : il y en a partout, le DevOps accélère !

DevOps

Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

DEVOPS

CI / CD²

DevOps

Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts



Continuous Integration

Exemple

Merge Request



Continuous Delivery

Exemple

Image Docker



Continuous deployment

Exemple

App en prod

GITLAB UN OUTIL DEVOPS

DevOps

Github - Gitlab

Introduction

Introduction - TP

Installation

Installation - TP

Administration

Administration - TP

Gestion des dépôts

GITHUB - GITLAB

Github

Une révolution

- Le « Google » du développeur : Outil majeur de montée en compétence
- Centré sur l'humain
- Encourage l'amateurisme

DevOps
[Github - Gitlab](#)
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GITHUB - GITLAB

Gitlab

Le petit frère de GitHub

- Quelques fonctions en moins :
 - La recherche parmi toutes les contributions
 - Github actions
- Quelques fonctions en plus :
 - Version self-hosted open source
 - Gitlab CI

DevOps
[Github - Gitlab](#)
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GITHUB - GITLAB

Gitlab

- Le produit
 - Serveur de gestion de source Git et de CI/CD
 - Existe en version open source
- Principales fonctionnalités
 - Serveur de versionning Git
 - CI / CD / CD (Auto DevOps)
 - Intégration LDAP / AD
 - Gestion de projets et tickets
- Avantages
 - Pas d'add-on à maintenir (vs forge multi-outils)
 - Synergie inter-outils élevée

DevOps
[Github - Gitlab](#)
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GITLAB UN OUTIL DEVOPS

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

INTRODUCTION

Les versions

- Gitlab CE
 - Open source (MIT)
 - Auto-hébergeable
 - Fonctionnalités de base
 - Gratuit
- Gitlab EE
 - Auto-hébergeable
 - Plus de fonctionnalités
 - Licence payante
- Gitlab.com
 - SaaS
 - Basé sur EE
 - Accessible gratuitement (fonctionnalités limitées)

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

INTRODUCTION

Fonctionnalités

- **Plan** : planification de tâches, tableau gestion de projet, milestones
- **Create** : création du code, des branches des tags, des merges requests
- **Verify** : Analyse du code, analyse de sécurité, test dynamique
- **Package** : Création et stockage des images Docker et des artefacts
- **Secure** : Tests de type SAST & DAST, scan de container
- **Release** : Livraison continue, pipeline delivery, flag, audit
- **Configure** : Auto DevOps, IaC, Kubernetes
- **Monitor** : Gestion des incidents, Métriques intégrées (Grafana / Prometheus)
- **Govern** : Gestion des dépendances, des vulnérabilités, politique de sécurité
- **Manage** : suivi des contributions, tickets, métriques sur la vélocité

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts



GITLAB UN OUTIL DEVOPS

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

INTRODUCTION - TP

- Créer un compte sur gitlab.com
- Configurer son compte avec une clé ssh
- Configurer la double authentification
- Créer un projet
- Créer un **board** et ajouter des issues
 - Tester les labels
- Tester les commits et les push
- Tester les Merge Request
 - Crées à partir d'une branche
 - Crées à partir d'une issue

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

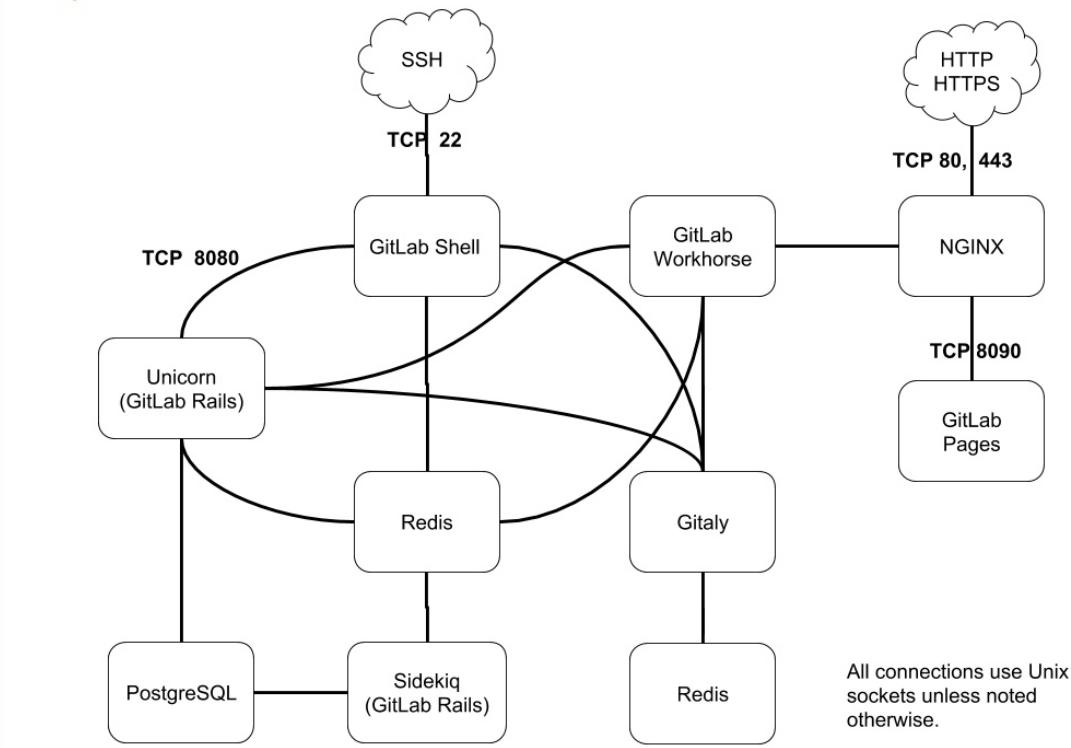
GITLAB UN OUTIL DEVOPS

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

INSTALLATION

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GitLab Application Architecture



INSTALLATION

Package Omnibus

- Monolithique par défaut
- Packagé pour GNU / Linux
- Le plus utilisé, notamment sur les images Docker
- Pas de HA par défaut

```
sudo apt update
sudo apt install -y curl openssh-server ca-certificates
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh|sudo bash
sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-ce
```

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

INSTALLATION

Les types d'installation

- Container / Kubernetes
- Package système Debian, Ubuntu, CentOS
- Cloud provider AWS, GCP, Azure...
- A partir des sources
- Gestionnaire de configuration Puppet, Ansible ...

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GITLAB UN OUTIL DEVOPS

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

INSTALLATION - TP

A l'aide de conteneurs :

- Installer et démarrer Gitlab
- Accéder au compte d'administration

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GITLAB UN OUTIL DEVOPS

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

ADMINISTRATION

Gratuit

- **Monitoring** : Informations système, jobs, logs, logs, checks...
- **Messages** : Gérer les messages aux utilisateurs
- **System Hooks** : Permet de créer des hooks HTTP sur des events GitLab
- **Applications** : Configurer OAuth pour l'intégration avec d'autres services
- **Abuse Reports** : Gérer les rapports d'abus
- **Kubernetes** : Connecter Gitlab à un cluster Kubernetes et le manager
- **Deploy keys** : Créer des jetons pour les projets privés depuis l'extérieur
- **Integrations** : Gérer les intégrations avec d'autres outils
- **Labels** : Gérer les labels pour tous les nouveaux projets
- **Appearance** : Gérer l'apparence de GitLab
- **Settings** : Gérer les paramètres globaux de l'instance (CI/CD, limites etc...)

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

ADMINISTRATION

Payant

- **License** : Gérer les licences de l'instance Gitlab
- **Push rules** : Gérer les règles de push (tail de commit etc...)
- **Geo** : Permet gérer des réplicats distant
- **Credentials** : Voir tous les tokens et clés SSH pour audit notamment

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

ADMINISTRATION

REST

- Client-server
- Stateless
- Système découpé en layer
- Uniforme
- Cache possible (serveur)
- Méthodes HTTP standard (cf RFC 1945)

```
curl --header "Authorization: Bearer
<your_access_token>""
"https://gitlab.example.com/api/v4/projects"
```

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

ADMINISTRATION

GraphQL

- Langage pour interroger l'API (Query Language)
- Client-serveur
- Uniquement méthode POST
- Cache possible (client)

```
curl 'https://gitlab.com/api/graphql' --header  
"Authorization: Bearer $GRAPHQL_TOKEN"  
--header "Content-Type: application/json" --request  
POST --data "{\"query\": \"query {currentUser  
{name}}\"}"
```

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

ADMINISTRATION

gitlab-ctl

- CLI
- Présent par défaut
- Permet de gérer les composants Gitlab

```
gitlab-ctl status
gitlab-ctl stop sidekiq
gitlab-ctl restart
gitlab-ctl tail
gitlab-ctl tail gitlab-rails
```

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GITLAB UN OUTIL DEVOPS

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

ADMINISTRATION - TP

- Ajoutez un message général pour avertir d'une maintenance
- Créez 2 utilisateurs (un admin, un standard)
- Vérifiez le bon fonctionnement du service Gitlab
- Consultez les logs de Nginx
- Redémarrez le composant Nginx
- Vérifiez le status (EN CLI)
- Redémarrez tout le Gitlab (avec gitlab-ctl)
- Exécutez 2 appels APIs
 - lister les groupes
 - créer un nouveau groupe "Mon groupe"
- Créez un backup de Gitlab...
- ... et restaurez-le

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GITLAB UN OUTIL DEVOPS

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GESTION DES DÉPÔTS

Ajout des membres

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

Formations > formations_marp > Members

Project members

You can invite a new member to **formations_marp** or invite another group.

Members 4

		Filter members			Account	
Account	Source	Access granted	Max role	Expiration	Created on	Last activity
 Florent Poinsaut  @fpoinsaut	Formations	6 months ago by Florent Poinsaut	Owner	Expiration date 	10 Jul, 2018	27 Oct, 2022
 Karin @Kr1	Direct member	6 months ago by Thomas Saquet	Maintainer	Expiration date 	27 May, 2020	17 Oct, 2022
 Thomas Saquet  @tsaquet	Formations	9 months ago	Owner	Expiration date 	10 Apr, 2018	31 Oct, 2022
 Yves Rougy @yrougy	Formations	7 months ago by Thomas Saquet	Maintainer	Expiration date 	25 Mar, 2022	26 Sep, 2022

GESTION DES DÉPÔTS

Clonage

main
formations_marp /
+ ▾
History
Find file
Web IDE
Download
Clone ▾



cours gitlab gln ok à 80%
Thomas Saquet authored 5 days ago

Name	Last commit	
.vscode	added intro section for LXM and fixed crash ...	
authors	git 1 day + sdv theme adjustement for linux ...	
courses	cours gitlab gln ok à 80%	
coursesV2	cours gitlab gln ok à 80%	
generator	feat: add kai as author	3 weeks ago
marp_engine	everything is launched from golang program...	1 month ago

Clone with SSH
git@usine.solution-libre.fr:for

Clone with HTTPS
https://usine.solution-libre.fr

Open in your IDE

- Visual Studio Code (SSH)
- Visual Studio Code (HTTPS)
- IntelliJ IDEA (SSH)
- IntelliJ IDEA (HTTPS)

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

GESTION DES DÉPÔTS

Gestion des branches

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

Branch	Last Commit	Commits	Merge Request	Compare	Delete	
11-encrypt-pvc	Sef2leb7 · Use a generic var for vol encryption #11	0.2	<input checked="" type="checkbox"/>	Merge request	Compare	Delete
12-compatibility-with-the-vl-3-0	036ca494 · Update terraform min ver + remove expe #12	0.1	<input checked="" type="checkbox"/>	Merge request	Compare	Delete
3-possibility-to-use-longhorn	d3095620 · Merge branch 'main' into 3-possibility-to-use-longhorn	0.2	<input checked="" type="checkbox"/>	Merge request	Compare	Delete
2-ability-to-add-a-nextcloud	09e52501 · Merge branch 'main' into 2-ability-to-add-a-nextcloud	0.8	<input checked="" type="checkbox"/>	Merge request	Compare	Delete
main	laceaaa4 · Fix ordering for the first installation	1 month ago	<input checked="" type="checkbox"/>	Merge request	Compare	Delete
Show more active branches						
4-ability-to-add-a-vault	051f3d92 · move longhorn code on a branch	3 months ago	<input checked="" type="checkbox"/>	Merge request	Compare	Delete

GESTION DES DÉPÔTS

Settings

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

French High Availability Multi-Cloud Hosting >  Terraform Module > Repository Settings

Search page

Default branch

[Expand](#)

Set the default branch for this project. All merge requests and commits are made against this branch unless you specify a different one.

Mirroring repositories

[Expand](#)

Set up your project to automatically push and/or pull changes to/from another repository. Branches, tags, and commits will be synced automatically. [How do I mirror repositories?](#)

Protected branches

[Expand](#)

Keep stable branches secure and force developers to use merge requests. [What are protected branches?](#)

Protected tags

[Expand](#)

Limit access to creating and updating tags. [What are protected tags?](#)

Deploy tokens

[Expand](#)

Deploy tokens allow access to packages, your repository, and registry images.

Deploy keys

[Expand](#)

Add deploy keys to grant read/write access to this repository. [What are deploy keys?](#)

Repository cleanup

[Expand](#)

Clean up after running `git filter-repo` on the repository. [?](#)

GESTION DES DÉPÔTS

Options CI/CD

DevOps
Github - Gitlab
Introduction
Introduction - TP
Installation
Installation - TP
Administration
Administration - TP
Gestion des dépôts

Formations > formations_marp > CI/CD Settings

 Search page

General pipelines

[Expand](#)

Customize your pipeline configuration.

Auto DevOps

[Expand](#)

Automate building, testing, and deploying your applications based on your continuous integration and delivery configuration. [How do I get started?](#)

Runners

[Expand](#)

Runners are processes that pick up and execute CI/CD jobs for GitLab. [What is GitLab Runner?](#)

Artifacts

[Expand](#)

A job artifact is an archive of files and directories saved by a job when it finishes.

Variables

[Expand](#)

Variables store information, like passwords and secret keys, that you can use in job scripts. [Learn more.](#)

Variables can be:

- **Protected:** Only exposed to protected branches or protected tags.
- **Masked:** Hidden in job logs. Must match masking requirements. [Learn more.](#)

Pipeline triggers

[Expand](#)

Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions. [Learn more.](#)

Deploy freezes

[Expand](#)

GITLAB UN OUTIL DEVOPS

Dans ce chapitre nous avons :

- Vu ce que DevOps signifie
- Eu un aperçu des possibilités offertes par Github et Gitlab
- Découvert Gitlab.
- Découvert Gitlab.com.
- Vu les différentes installations de Gitlab.
- Vu comment installer son propre Gitlab.
- Vu comment administrer Gitlab.
- Manipulé l'administration de Gitlab.
- Vu comment gérer un dépôt git à travers Gitlab.

CHAPITRE 4

GITLAB CI/CD²

GITLAB CI/CD²

- **Automatiser si c'est possible** nous donnera des pistes pour vérifier qu'on peut vraiment automatiser
- **Intégration continue** nous présentera l'intégration continue
- **Intégration continue - TP** nous permettra de mettre en place des premiers jobs
- **Docker dans Gitlab** nous montrera comment on peut combiner Docker et Gitlab
- **Chaîne CD²** nous présentera la livraison et le déploiement continu
- **Chaîne CD² - TP** nous permettra de tester la livraison et le déploiement continus

GITLAB CI/CD²

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Dev

- Le déploiement continu n'est possible que si on a confiance...
- ... On n'a confiance que si les tests existent.
- Les tests n'existent que si le code est testable...
- ... Ce n'est testable que si le dev est de qualité
- Sans un dev de qualité, on n'a pas de CI/CD² valable !

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Principes de codage

- Objectifs
 - Répondre au besoin
 - S'adapter aux évolutions du besoin
 - S'adapter à d'autres besoins
- Qu'est-ce qu'un bon code ?
 - Facile à appréhender (lisibilité)
 - Facile à corriger (maintenabilité)
 - Facile à faire évoluer (évolutibilité)
 - Facile à réutiliser (factorisabilité)
 - Facile à tester (testabilité)

Automatiser si c'est possible

- Intégration continue
- Intégration continue - TP
- Docker dans Gitlab
- Chaîne CD²
- Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

C'est quoi du code testable ?

- Les tests sont pensés avant le code
 - Permet de garantir que le travail est conforme
- Si possible, les tests sont rédigés avant le code (TDD)
- Deux principes importants
 - Niveaux d'abstractions uniques
 - Principe de responsabilité unique

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Niveau d'abstraction unique

- Garder un même niveau d'abstraction à l'intérieur d'une méthode
 - Lisibilité
- Définition de couches cohérentes
 - Factorisabilité
 - Testabilité
- Modifier une couche n'impacte pas les autres
 - Maintenabilité
 - Evolutibilité

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Principe de responsabilité unique

- Une classe = une responsabilité
- Peut être étendu aux méthodes ou aux variables
- Découpage clair des fonctionnalités
 - Lisibilité
 - Factorisabilité
 - Testabilité
- En cas de modification d'une fonctionnalité, les autres fonctionnalités ne sont pas affectées
 - Maintenabilité
 - Évolutibilité
- Aide :
 - Documenter le code (e.g. Doxygen)
 - Définir des noms significatifs

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests

- Exemples de types de tests qu'on peut inclure en CI/CD

- Tests unitaires
- Tests de bout en bout
- Tests d'intégration
- Tests fonctionnels
- Tests de charge
- Tests de qualité
- Tests de sécurité

Automatiser si c'est possible

- Intégration continue
- Intégration continue - TP
- Docker dans Gitlab
- Chaîne CD²
- Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests unitaires

- Dépend de la technologie
 - gtest
 - jUnit
 - etc.
- A faire au fur et à mesure des développements
 - Si possible avant (TDD)
- Utiles pour
 - Documenter
 - Construire une fonction sans rien oublier
 - La non régression
 - etc.

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests de bout en bout

- En anglais, End To End : E2E
- Permet de tester les comportements globaux
- En général utilise le DOM (Document Object Model)
- Utiles pour :
 - Non regression (s'ils sont automatisés !)
 - Tests avec des contextes différents (navigateur, migrations, etc.)
- Exemple : Cypress

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests fonctionnels

- Vérifier que les règles métiers sont bien implémentées
 - Exemple : est-ce que ce calcul insère la bonne valeur en base de données ?
- Différentes méthodes suivant l'interface à tester
 - Interface graphique : Cypress
 - Temps de réponse : K6
 - API : Hopscotch

Automatiser si c'est possible

- Intégration continue
- Intégration continue - TP
- Docker dans Gitlab
- Chaîne CD²
- Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests d'intégration

- On vérifie :
 - Que les différents développements fonctionnent bien ensemble
 - Que les différents composants sont bien opérationnels
 - Exemple : est-ce que la connexion à la base de données se fait correctement ?
- Différentes méthodes suivant ce qu'il faut tester
 - Utiliser l'outil de tests u (jeu de tests différents)
 - Utiliser l'outil de tests e2e

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests de charge

- On vérifie si le logiciel tient la charge
 - Nombre de connexions
 - Nombre de requêtes
 - Quantité de données téléchargées
 - Performances sur la durée
 - Consommation des ressources
- On peut utiliser des logiciels comme :
 - K6
 - Gatling

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests de qualité

- Est-ce que le code répond à toutes les normes de qualité ?
- Analyse statique en fonction du langage
 - Exemple : PHPStan, Sonar Qube
- Analyse dynamique au moment de charger la page
 - Exemple : LightHouse

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Tests de sécurité

- SAST : tests de sécurité statiques, avant la compilation / le build
 - Boîte blanche
 - Inclus dans l'IDE
- DAST : tests de sécurité dynamiques
 - Exemples de soucis détectés : cross-script scripting
 - Exemples d'outils : OWASP Zap, liste ici :
https://owasp.org/www-community/Vulnerability_Scanning_Tools

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Intégration à l'IDE

- Environnement de Développement Intégré

- Editeur de texte
 - Auto-complétion
 - Formatage automatique
 - etc.
- Analyse du code
- Moteur de recherche
- Compilateur et/ou Editeur de liens
- Débogueur en ligne
- Etc.

Automatiser si c'est possible

- Intégration continue
- Intégration continue - TP
- Docker dans Gitlab
- Chaîne CD²
- Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Intégration des tests aussi

- Simple
- Consiste à utiliser les bibliothèques de test / de mock dans l'IDE
- Build, run, c'est testé
- Arrivée d'outils puissants pour faire ça, par exemple GitPod !

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

AUTOMATISER SI C'EST POSSIBLE

Automatiser tout ça dans la CI !

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

- Pourquoi ?

- Savoir que tous les cas sont couverts
 - > Spécification très détaillée
- Assurer la non-régression

- Quels risques ?

- Code supplémentaire à maintenir
- Plus simple à refaire qu'à maintenir en cas :
 - De redéfinition du besoin
 - De refonte d'un code mal pensé
- Dépendance vers des éléments extérieurs

AUTOMATISER SI C'EST POSSIBLE

Comment automatiser ?

- On s'appuie sur le système de branches (git branching modèle)
- On utilise différents environnements
- On utilise un orchestrateur
 - Jenkins ?
 - Github
 - Gitlab !

Automatiser si c'est possible

Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

GITLAB CI/CD²

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

INTÉGRATION CONTINUE

.gitlab-ci.yml

- A la **racine** du projet
- AutoDevOps
- Syntaxe **YML**
- Successions de **stages**
- Jobs **parallélisables**

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

```
before_script:  
  - apt install rubygems ruby-dev -y  
  
run-test:  
  script:  
    - ruby --version
```

INTÉGRATION CONTINUE

Les Runners (1/2)

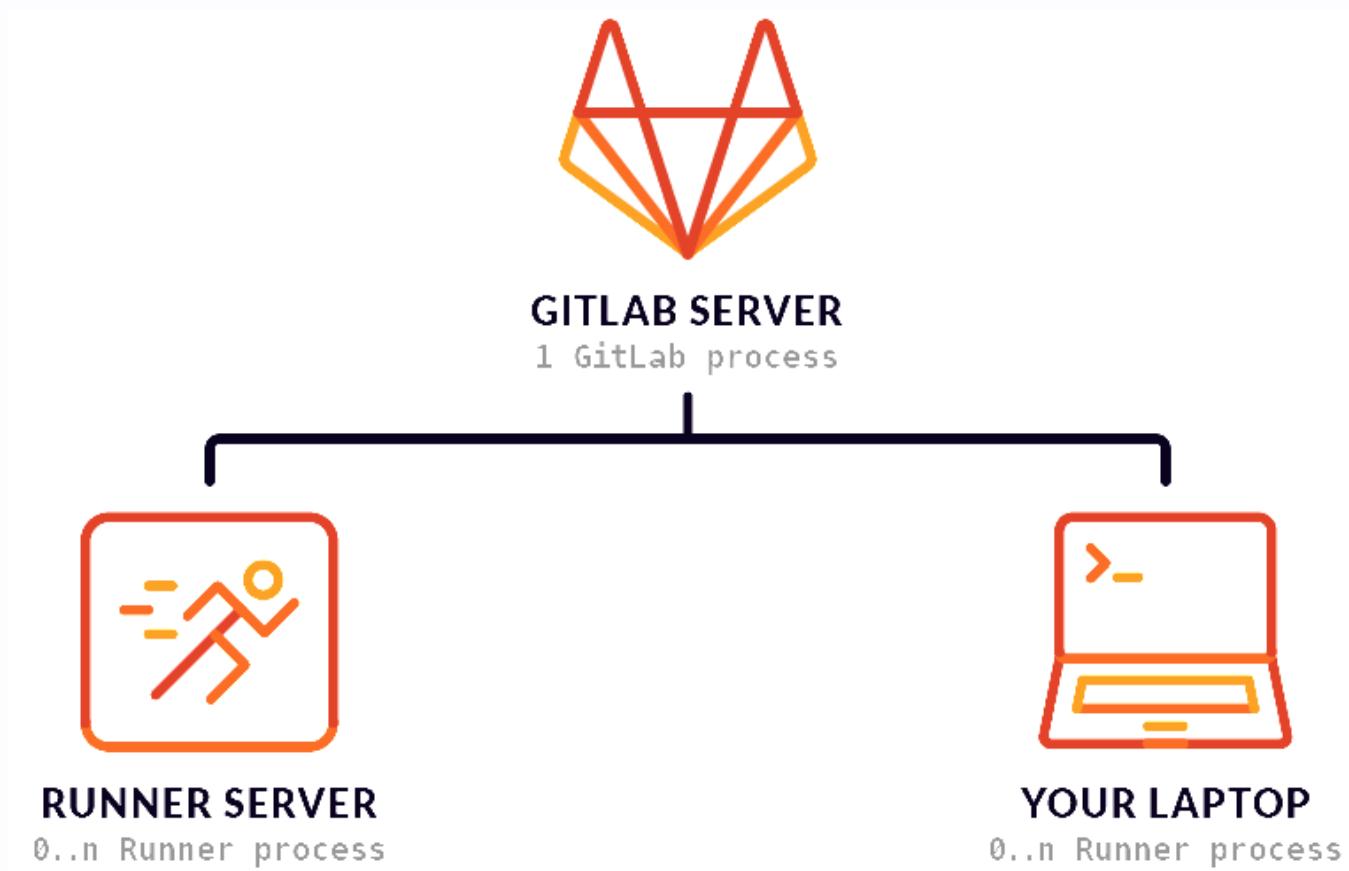
- MultiOS (GNU/Linux, MacOS, BSD, Windows)
- Communication via HTTP
- Possibilité d'utiliser des variables protégées
- Paramétrables par config.toml
- Différents executors
 - Shell
 - Docker
 - VirtualBox
 - Kubernetes
 - ...

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

INTÉGRATION CONTINUE

Les Runners (2/2)

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP



INTÉGRATION CONTINUE

Les types de Runners

Les runners peuvent être associés à différents niveaux :

- Shared (au niveau de l'instance)
- Specific (au niveau des projets)
- Group (au niveau des... groupes)

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP



INTÉGRATION CONTINUE

Des minutes de Runners offertes !

- Gitlab.com met à disposition des Shared Runners
- Suite à des abus ils ont ajouté des limites
 - Offre gratuite : 400 minutes par mois

Automatiser si c'est possible

[Intégration continue](#)

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

INTÉGRATION CONTINUE

Utilisation des Runners

- Installation
 - Ajout des dépôts GitLab
 - Installation via gestionnaire de paquets
- Enregistrement du Runner
 - Via l'interface web
 - En ligne de commande
- Sélection de l'exécuteur
 - Shell
 - Docker
 - ...
- Sélection des tags
 - Choix du runner dans .gitlab-ci.yml

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

INTÉGRATION CONTINUE

Exemple (Debian)

Exemple (Debian)

```
$ curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash  
$ export GITLAB_RUNNER_DISABLE_SKEL=true; sudo -E apt install gitlab-runner  
$ sudo gitlab-runner register
```

Automatiser si c'est possible

[Intégration continue](#)

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD²-TP

INTÉGRATION CONTINUE

Automatiser si c'est possible
Intégration continue
 Intégration continue - TP
 Docker dans Gitlab
 Chaîne CD²
 Chaîne CD² - TP

```

image: python:latest # Image pour l'executor Docker
variables:
  # Définition de variable
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"
stages:
  # Définition des stages
  - build
cache:
  # Objets à garder en cache (sur le runner)
  paths:
    - .cache/pip
    - venv/
before_script:      # A pré-exécuter
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
run:
  # Nom du job
  stage: build      # Place le job dans un "stage"
  tags:
    # Choix du runner
    - builders
script:
  # Les commandes à exécuter
  - python setup.py bdist_wheel
artifacts:
  # Objets exporté utilisables dans d'autres jobs
  paths:
    - dist/*.whl
only:
  # Exécuter job sous certaines conditions (/!\ déprécié)
  - master
when: manual        # Faire une exécution manuel
  
```

INTÉGRATION CONTINUE

Automatiser si c'est possible
Intégration continue
 Intégration continue - TP
 Docker dans Gitlab
 Chaîne CD²
 Chaîne CD²- TP

```

stages:
  - generate
  - publish

build📦:
  stage: generate
  image: klakegg/hugo
  tags:
    - docker
  script:
    - hugo version
    - hugo config
    - hugo -d public
  artifacts:
    name: "$CI_JOB_NAME-$CI_COMMIT_REF_NAME"
    paths:
      - public
    expire_in: 30 minutes

publish📢:
  stage: publish
  image: garland/aws-cli-docker
  tags:
    - docker
  variables:
    S3_BUCKET_NAME: "static-blog-tge"
    DISTRO_ID: "E10YH3P3C9ELCV"
  script:
    - aws configure set preview.cloudfront true
    - aws s3 sync public s3://$S3_BUCKET_NAME --delete;
    - aws cloudfront create-invalidation --distribution-id $DISTRO_ID --paths "/*";
  dependencies:
    - build📦
  only:
    - master

```

INTÉGRATION CONTINUE

Debug d'un Runner / Pipeline

- Vous pouvez voir les logs des jobs sur l'interface en cliquant sur le job
- Un Linter est disponible dans la partie CI/CD du projet
- Dans le panneau de gestion des runners, cliquez sur l'ID de votre runner, vous pouvez limiter son déclenchement

Automatiser si c'est possible

[Intégration continue](#)

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

INTÉGRATION CONTINUE

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²Chaîne CD²-TP

The screenshot shows the GitLab web interface. On the left is a sidebar with various project management options like 'Project information', 'Issues', 'Merge requests', 'CI/CD', 'Pipelines', 'Editor', 'Jobs', 'Schedules', 'Security & Compliance', 'Deployments', 'Packages & Registries', 'Infrastructure', 'Monitor', 'Analytics', 'Wiki', 'Snippets', and 'Settings'. The 'Jobs' option is currently selected.

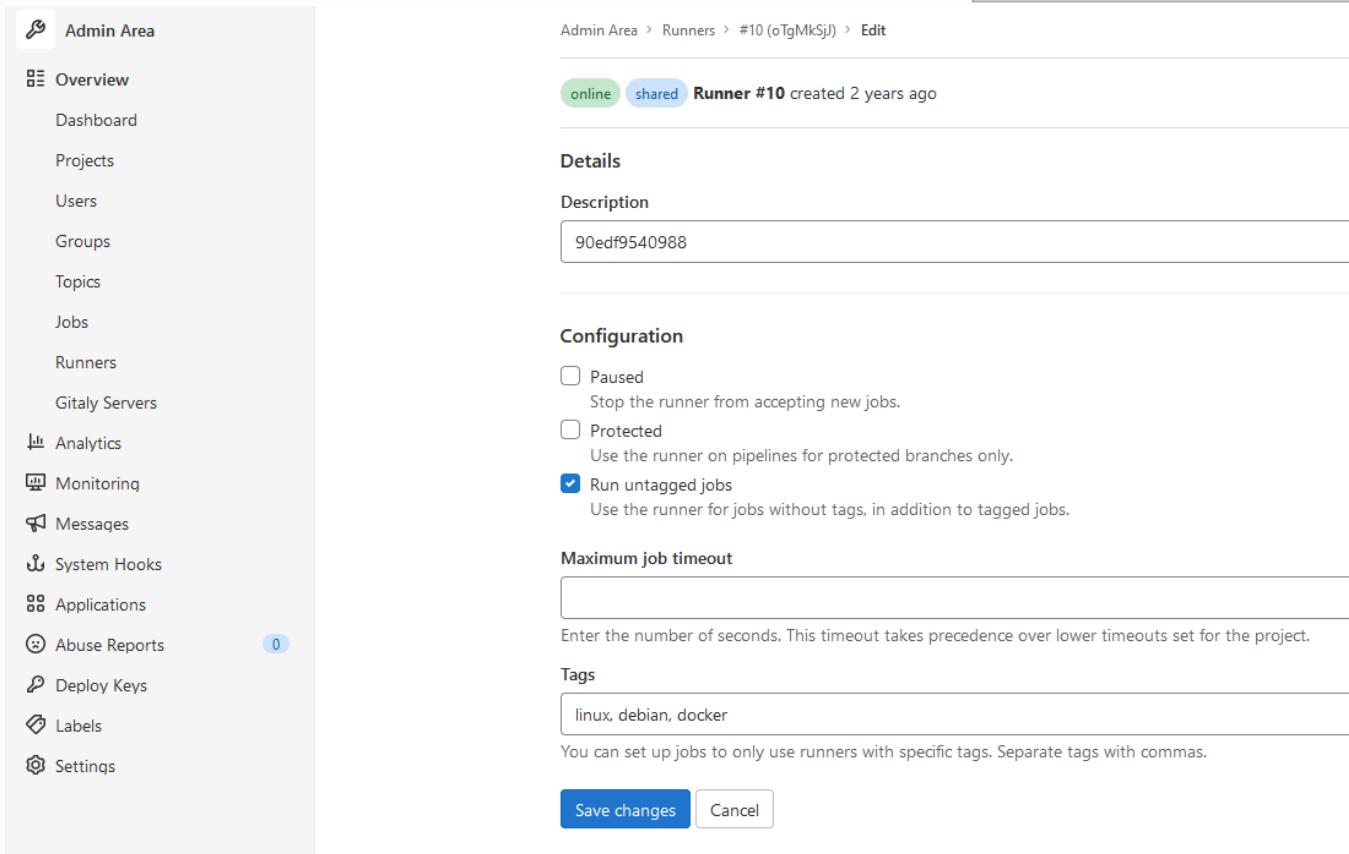
The main area displays a job log for a failed job named 'Job markdownlint' triggered 1 day ago by Thomas Saquet. The log shows the following steps:

- Running with gitlab-runner 15.3.0 (bbcb5aba)
- on 90edf9540988 otgMksJ
- Preparing the "docker" executor**
- Using Docker executor with image registry.gitlab.com/00kellyjac/docker_markdownlint-cli:latest ...
- Pulling docker image registry.gitlab.com/00kellyjac/docker_markdownlint-cli:latest ...
- Using docker image sha256:db52d6a279e23f2ab21b3e17869061f15068dc038d59a75c2c2f2477ea2c93 for registry.gitlab.com/00kellyjac/docker_markdownlint-cli:lates with digest registry.gitlab.com/00kellyjac/docker_markdownlint-cli@sha256:96a92deec3490c09a29b517c36605e31ea788512c3b481df346db8d88c24ca34 ...
- Preparing environment**
- Running on runner-otgmksj-project-46-concurrent-0 via 5547c49c39a3...
- Getting source from Git repository**
- Fetching changes with git depth set to 20...
- Reinitialized existing Git repository in /builds/otgMksJ/0/formations/formations_marp/.git/
- Checking out c5621cc5 as main ...
- Skipping Git submodules setup**
- Executing "step_script" stage of the job script
- Using docker image sha256:db52d6a279e23f2ab21b3e17869061f15068dc038d59a75c2c2f2477ea2c93 for registry.gitlab.com/00kellyjac/docker_markdownlint-cli:latest with digest registry.gitlab.com/00kellyjac/docker_markdownlint-cli@sha256:96a92deec3490c09a29b517c36605e31ea788512c3b481df346db8d88c24ca34 ...
- \$ markdownlint '**/*.md'
- preludes/orsys_remote.md:11:10 M0041/first-line-heading/first-line-h1 First line in a file should be a top-level heading [Context: "## Organisation du cours"]
- preludes/orsys_remote.md:11:10 M0026/no-trailing-punctuation Trailing punctuation in heading [Punctuation: ' ']
- README.md:19:52 M0009/no-trailing-spaces Trailing spaces [Expected: 0 or 2; Actual: 1]
- README.md:48:33 M0009/no-trailing-spaces Trailing spaces [Expected: 0 or 2; Actual: 1]
- README.md:54:11 M0009/no-trailing-spaces Trailing spaces [Expected: 0 or 2; Actual: 1]
- Cleaning up project directory and file based variables**
- ERROR: Job failed: exit code 1**

On the right side, there's a summary for the 'markdownlint' pipeline, showing it finished 1 day ago with a duration of 9 seconds. It also lists the commit (c5621cc5), runner (10), and the message 'everything is launched from golang program | README updated.'

INTÉGRATION CONTINUE

Automatiser si c'est possible
Intégration continue
 Intégration continue - TP
 Docker dans Gitlab
 Chaîne CD²
 Chaîne CD² - TP



The screenshot shows the GitLab Admin Area with the sidebar open. The sidebar includes links for Admin Area, Overview, Dashboard, Projects, Users, Groups, Topics, Jobs, Runners, Gitaly Servers, Analytics, Monitoring, Messages, System Hooks, Applications, Abuse Reports (with a notification count of 0), Deploy Keys, Labels, and Settings.

The main content area is titled "Admin Area > Runners > #10 (oTgMkSjI) > Edit". It shows a runner named "Runner #10" created 2 years ago, which is "online" and "shared".

Details: Description: 90edf9540988

Configuration:

- Paused: Stop the runner from accepting new jobs.
- Protected: Use the runner on pipelines for protected branches only.
- Run untagged jobs: Use the runner for jobs without tags, in addition to tagged jobs.

Maximum job timeout: (Input field is empty)

Enter the number of seconds. This timeout takes precedence over lower timeouts set for the project.

Tags: linux, debian, docker

You can set up jobs to only use runners with specific tags. Separate tags with commas.

Action Buttons: Save changes | Cancel

GITLAB CI/CD²

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

INTÉGRATION CONTINUE - TP

Objectifs

- Nous allons :

- Mettre en place nos propres runner pour exécuter les différentes actions décrites dans le fichier [.gitlab-ci.yml](#)
- Apprendre à configurer les runners avec différents "Exécuteurs", Shell et Docker
- Apprendre à gérer la répartition des jobs entre les runners à l'aide des tags

Automatiser si c'est possible
Intégration continue
[Intégration continue - TP](#)
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

INTÉGRATION CONTINUE - TP

Runner Shell - Mise en place

Automatiser si c'est possible
Intégration continue
[Intégration continue - TP](#)
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

- Installez un premier runner utilisant l'executor **Shell**
- Vérifiez le fichier de configuration du runner qui a été créé pendant la phase d'enregistrement de ce dernier
- Donnez un tag **cli** au runner
- Créez un fichier **.gitlab-ci.yml** à la racine de votre dépôt. Il va permettre d'exécuter :
 - echo "Mon premier job !" dans un premier job
 - echo "Mon second job !" dans un second job
- Faites en sorte que le second job attende la fin de l'exécution du premier

INTÉGRATION CONTINUE - TP

Runner Docker

- Créez un second runner utilisant l'executor **Docker**
 - Attention à ne pas mélanger son fichier de configuration avec le runner précédent !
 - Lui donner le tag **conteneur**
- Ajoutez dans le **.gitlab-ci.yml** :
 - Un troisième job qui est dépendant des deux premiers et qui s'exécute uniquement sur **main**
 - Qui écrit `echo "Mon job final!"`

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

GITLAB CI/CD²

Automatiser si c'est possible

Intégration continue

Intégration continue - TP

Docker dans Gitlab

Chaîne CD²

Chaîne CD² - TP

DOCKER DANS GITLAB

Objectifs

- Dans ce TP on souhaite utiliser les capacités de Gitlab à gérer les Dockerfile et les images Docker, nous allons :
 - Créer notre propre image à partir d'un Dockerfile
 - La ranger dans la Registry Docker intégrée à Gitlab
 - Discuter au passage les points auxquels il faut être attentif quand on héberge son propre Gitlab

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

DOCKER DANS GITLAB

Consignes

- Créez un **Dockerfile** dont la commande est un echo "Hello world", construire l'image & la lancer
 - L'objectif est d'utiliser ce fichier comme entrée de notre CI
- Push votre **Dockerfile** dans un nouveau projet gitlab
- Push votre image Docker dans le registry de votre projet Gitlab

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

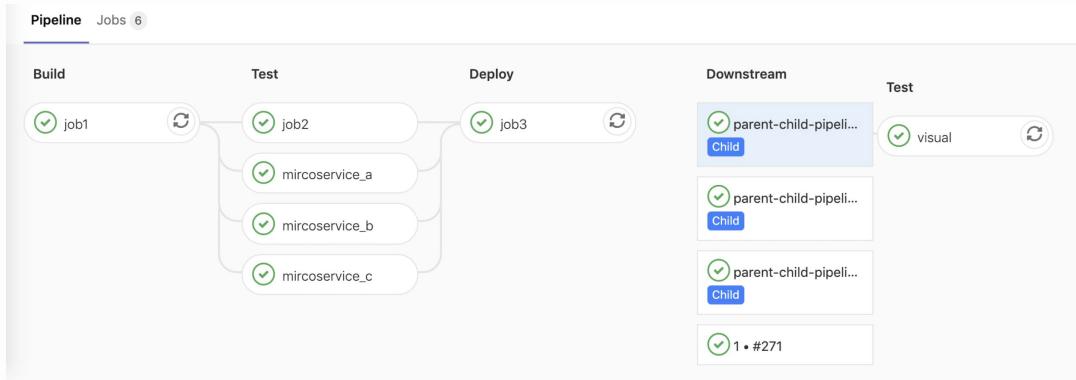
GITLAB CI/CD²

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

CHAÎNE CD²

Exemple

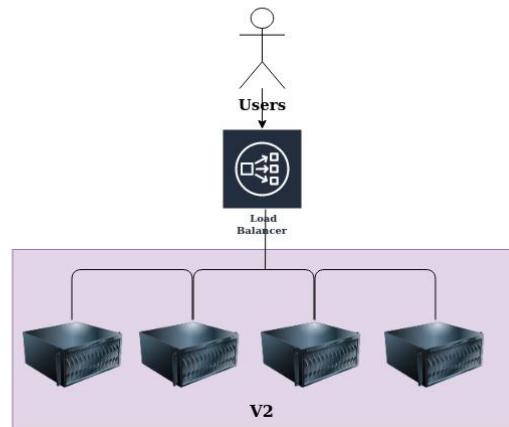
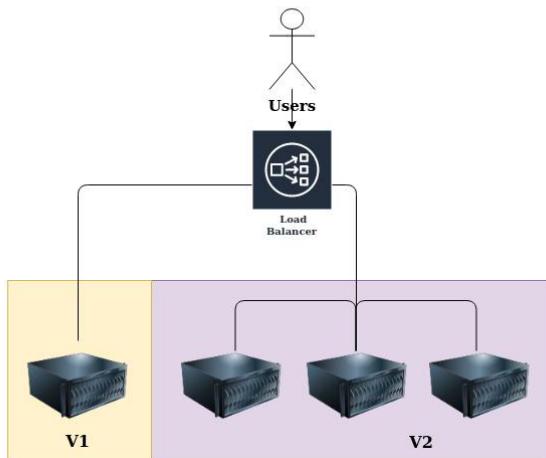
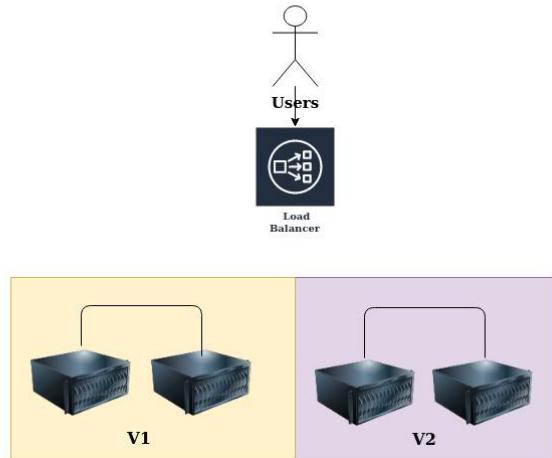
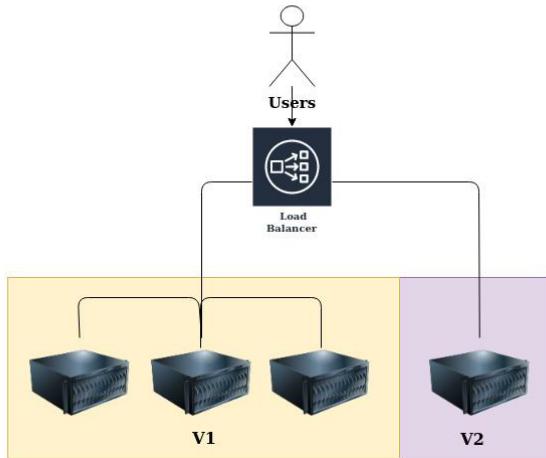
Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP



CHAÎNE CD²

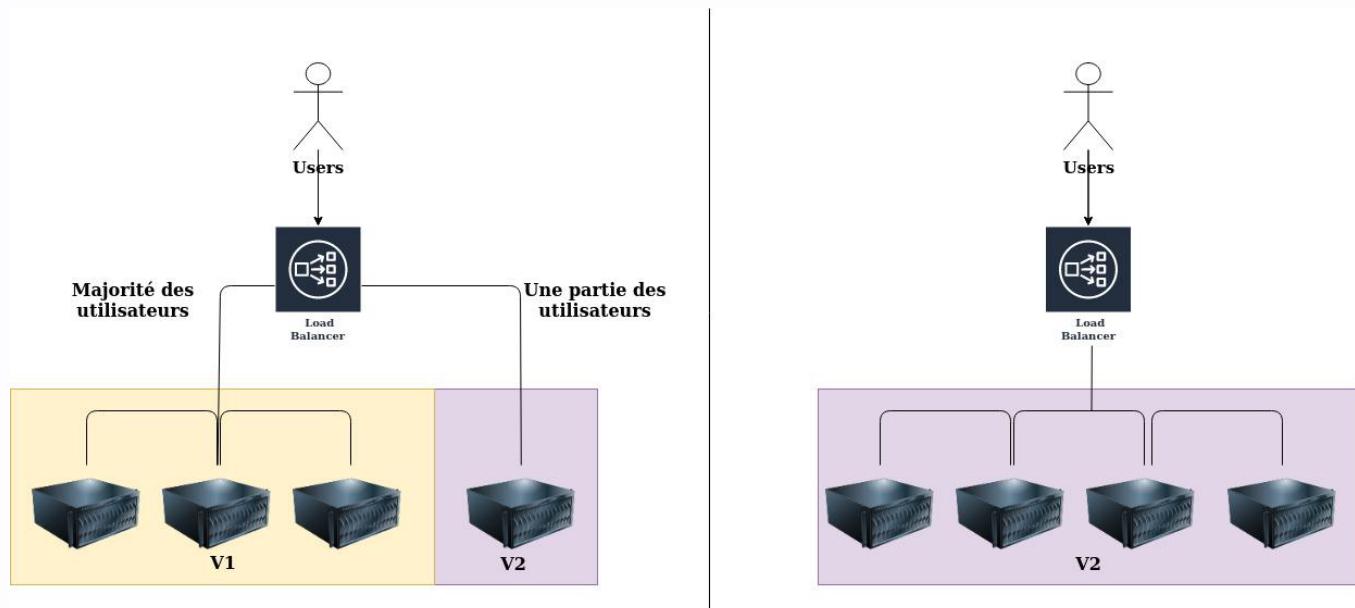
Rolling deployment

Automatiser si c'est possible
 Intégration continue
 Intégration continue - TP
 Docker dans Gitlab
Chaîne CD²
 Chaîne CD² - TP



CHAÎNE CD²

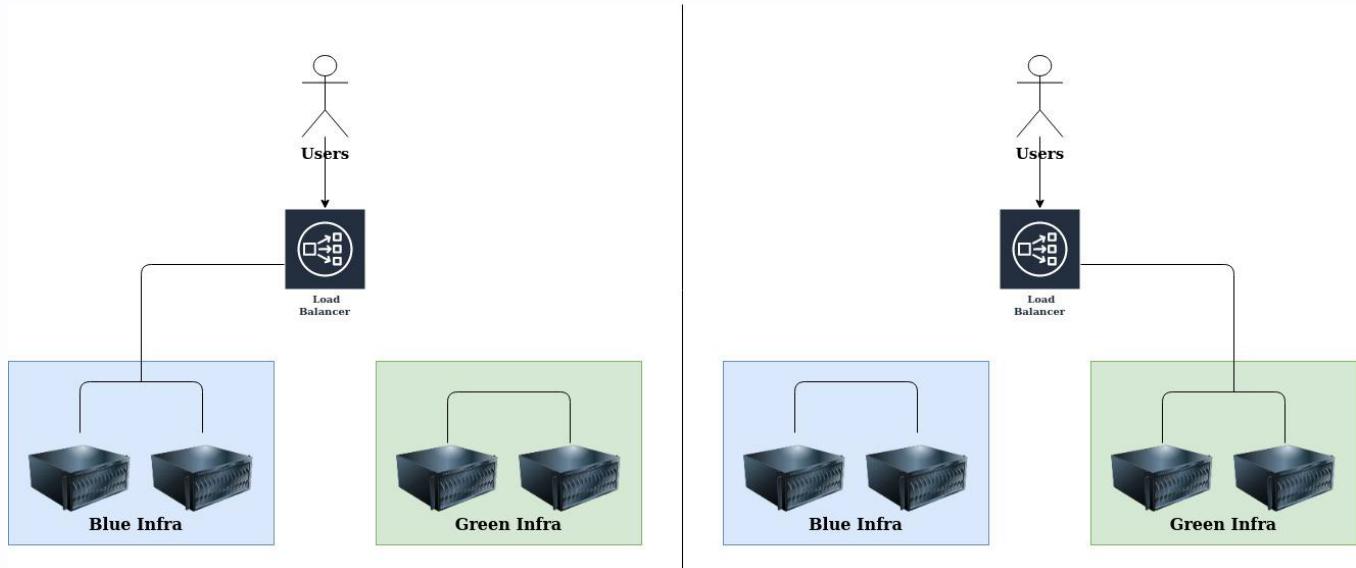
Canary



CHAÎNE CD²

Blue / Green

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP



CHAÎNE CD²

Les registres disponibles dans Gitlab

- CD : livraison continue... D'un paquet / d'une image
- CD : déploiement continu... De ce même paquet / de cette même image
- IaC : déploiement à partir de modules

Gitlab fournit des outils pour stocker les paquets, les images, les modules, etc.

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

CHAÎNE CD²

Package

- Package Registry compatible avec

- Composer (PHP)
- Conan (C/C++)
- Generic (ce que vous voulez !)
- Maven (Java)
- npm (Node)
- NuGet (.Net)
- PyPI (Python)
- RubyGems (Ruby)

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

CHAÎNE CD²

Container

- Container Registry compatible avec
 - Docker
- Permet de stocker des images Docker produites pendant le processus de livraison continue
- Permet d'utiliser les images en question pour les déployer

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

CHAÎNE CD²

Infrastructure

- Infrastructure Registry compatible avec
 - Modules Terraform
- On peut les construire, les stocker et les réutiliser

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

CHAÎNE CD²

Mais aussi

- Harbor
 - Pour stocker les artefacts liés à Docker et Kubernetes
 - A la place de la Container Registry intégrée
- Dependency Proxy
 - Pour stocker les images Docker localement et ne pas les télécharger systématiquement depuis le Hub

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

GITLAB CI/CD²

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD² - TP

CHAÎNE CD² - TP

La première chaîne CD² !

- Créez un nouveau projet
- Créez un conteneur Docker basé sur l'image **nginx** qui affiche une page **index.html** contenant le message de votre choix
- Associez un runner utilisant l'executeur "Docker" au projet (depuis votre machine via un conteneur Docker par exemple)
- Créez un pipeline Gitlab (i.e. un `.gitlab-ci.yml`) en 3 stages qui :
 - dans un stage tests, vérifiez la présence du fichier `index.html`
 - dans un stage build, construisez l'image Docker et la push dans la registry du repository
 - dans un stage deploy, déployez cette image (uniquement sur la branche main)

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD²-TP

CHAÎNE CD² - TP

Les environnements

Dans ce TP, nous allons explorer la notion d'environnement dans Gitlab.

- Ajoutez un nouveau job dans le stage tests dont l'objectif est de :
 - Utilisez des environnements nommés dynamiquement en fonction du nom des branches (il faut trouver la variable prédefinie qui vous permettra de faire ça !)
 - Affichez le contenu d'une variable (définie au niveau du projet) qui aura une valeur spécifique pour les branches **main** et **develop** et une valeur par défaut pour n'importe quelle autre branche.

Automatiser si c'est possible
Intégration continue
Intégration continue - TP
Docker dans Gitlab
Chaîne CD²
Chaîne CD²-TP

GITLAB CI/CD²

Dans ce chapitre nous avons :

- Vu comment créer nos premiers pipelines complets.
- Vu comment créer nos premiers pipelines.
- Vu comment utiliser le fichier .gitlab-ci.yml pour créer des jobs et de stages.
- Vu comment Docker s'intègre dans Gitlab.
- Vu comment créer nos premiers pipelines complets.
- Créé nos premiers pipelines complets.

CHAPITRE 5

GITLAB ET AU-DELA

GITLAB ET AU-DELA

- **Shared runner Windows** nous présentera rapidement les Shared Runner Windows
- **Terraform, IaC** nous permettra de découvrir Terraform
- **Auto DevOps** nous présente les fonctionnalités de configuration automatique de Gitlab
- **Gitlab DAST** nous présentera les tests dynamiques

GITLAB ET AU-DELA

Shared runner Windows

Terraform, IaC

Auto DevOps

Gitlab DAST

SHARED RUNNER WINDOWS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Shared Runner SaaS

- Gitlab 16.1 : Toujours en beta
- Ne devraient pas être utilisés en production
- Sont lancés sur GCP
- Avancement disponible ici :
<https://gitlab.com/groups/gitlab-org/-/epics/2162>

SHARED RUNNER WINDOWS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Limitations

- Long à démarrer
- Peut passer en maintenance sans avertissement
- Incompatible avec Docker
- Gitlab est susceptible d'introduire des "Breaking Changes" dans le futur

SHARED RUNNER WINDOWS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Créer ses propres runners windows !

- Même processus que pour les autres runners
- Utilise le PowerShell pour exécuter la partie "script"
- Pas de limite particulière

GITLAB ET AU-DELA

Shared runner Windows

Terraform, IaC

Auto DevOps

Gitlab DAST

TERRAFORM, IAC

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Qu'est ce que c'est ?

- Open Source
- IaC : Infrastructure As Code
- Créé par HashiCorp (Vault, Consul, Vagrant, etc.)
- Outil de provisionnement d'infrastructure
- Configuration stockable sous forme de code
- Compatible avec la plupart des fournisseurs Cloud

TERRAFORM, IAC

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Quels sont les avantages ?

- Versionnable
- Langage facile à comprendre
- Facilement (la plupart du temps) portable
- Permet à tout le monde de comprendre l'infra

TERRAFORM, IAC

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Les concepts clés des fichiers

- Provider : Le plugin qui permet d'intéragir avec le fournisseur de Cloud
- Module : C'est un dossier avec des modèles Terraform où toutes les configurations sont définies
- Région : Permet de définir dans quelles régions l'infrastructure doit être déployée
- Ressources : Permet de définir quel types de ressources sont utilisées pour l'infrastructure

TERRAFORM, IAC

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Les commandes

- Valeurs de sortie : On peut définir quels sont les éléments renvoyés par terraform, ça permet de récupérer des informations pour la suite du déploiement
- Plan : C'est une étape dans laquelle Terraform détermine toutes les opérations à faire pour passer de l'état actuel à l'état souhaité de l'infrastructure à l'état souhaité.
- Apply : C'est l'étape qui permet d'appliquer les changements déterminés lors du **plan**

TERRAFORM, IAC

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Le state

- Terraform a besoin de stocker un "état"
- Ca lui permet de faire le lien entre la configuration et le monde réel
- Il utilise le **state** pour faire le **plan** et le **apply**
- Il est verrouillé pendant une opération
- Il peut être géré par Gitlab

TERRAFORM, IAC

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Les modules

- Ce sont des ensembles de fichiers qui sont utilisés ensemble
- On peut développer ses propres modules
- C'est comme ça qu'on package et ainsi qu'on peut réutiliser des ressources Terraform

GITLAB ET AU-DELA

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

AUTO DEVOPS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Pipeline par défaut

- Détection de votre langage
- Utilisation du template qui correspond
- Choix de la stratégie de déploiement



AUTO DEVOPS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Configuration

- Cluster Kubernetes / Compte chez un fournisseur Cloud supporté
- Pas de fichier, juste l'appli
- Définition des credentials dans les variables Gitlab
 - EC2
 - ECS
 - Kubernetes
- Configuration dans le fichier .gitlab-ci.yml
- C'est tout !

AUTO DEVOPS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Review App

- C'est un environnement de preview
- Automatisable dans le cadre de Auto DevOps
- Peut être mis en place pour chaque branche
- Peut bloquer un merge automatique vers la branche de mise en prod
- Peut être désactivé / éteint automatiquement après un certain temps



AUTO DEVOPS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Les différentes étapes

- Auto Build
 - Nécessite Docker in Docker
 - Utilise la Container Registry
- Auto Test
 - Detecte le langage et le framework
 - Utilise les tests présents
- Auto Code Quality
 - Utilise CodeClimate
 - Met un rapport à disposition
 - Publie les différences entre les MR
- Auto SAST
 - Met un rapport à disposition
- Auto Secret Detection
 - Vérifie si on n'a pas laissé des secrets en dur
 - Met un rapport à disposition
- Auto Review Apps
 - Si un cluster Kubernetes est dispo
 - Crée les environnements temporaires pour tester l'appli
- Auto Deploy
 - Si un cluster Kubernetes est dispo
 - Tout est personnalisable
 - Basé sur Helm

AUTO DEVOPS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Les différentes étapes (Ultimate)

- Auto Dependency Scanning
 - Scanne les dépendances
 - Vérifie les problèmes de sécurité potentiels
- Auto License Compliance
 - Vérifie les licences des dépendances
 - Les licences sont publiées dans la MR
- Auto Container Scanning
 - Utilisation de Trivy
 - Recherche des problèmes de sécurité dans les images Docker
- Auto DAST
 - Utilisation automatique de l'outil OWASP ZAPProxy
 - Scan des Review App
- Auto Browser Performance Testing
 - Utilise Sitespeed.io pour tester la performance d'une appli Web
 - Le résultat est dans la MR
- Auto Load Performance Testing
 - Utilisation de K6
 - Il faut écrire le test K6 soi-même
 - Le résultat est dans la MR



AUTO DEVOPS

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Personnalisation

- On peut faire ses propres BuildPacks
- On peut utiliser ses propres Dockerfiles
- On peut passer des arguments pour préciser les versions qu'on souhaite utiliser
- On peut utiliser ses propres images Docker
- On peut personnaliser les Helm Charts
- On peut inclure le template dans son **.gitlab-ci.yml** et le modifier au sein du projet

GITLAB ET AU-DELA

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

GITLAB DAST

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Fonctionnalités

- DAST = Dynamic Application Security Testing
- Disponible uniquement pour les utilisateurs **Ultimate**
- Orienté Web !
- Analyse passive (Zap Baseline Scan)
- Analyse active
 - à configurer
 - à combiner avec les "Review App"

GITLAB DAST

Mise en place

- Utiliser un runner avec l'exécuteur docker sur Linux/amd64
- Avoir l'application cible déployée
- Ajouter le stage **dast** dans le fichier **.gitlab-ci.yml** après le stage de déploiement

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

GITLAB DAST

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Déclenchement

- Automatique
 - Déclenché par MR
 - Variables fournies dans le fichier `.gitlab-ci.yml`
- Manuel
 - Une UI permet d'entrer les variables
 - Toutes les variables disponibles pour Dast ne sont pas accessibles

GITLAB DAST

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Template

- Il suffit de l'inclure dans le fichier **.gitlab-ci.yml** et de personnaliser les variables qu'il contient

```
include:  
  - template: <template_file.yml>  
  
variables:  
  DAST_WEBSITE: https://example.com
```



GITLAB DAST

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Résultat

- DAST report artifact
- Disponible
 - Au niveau projet : Security & Compliance.
 - Au niveau de la merge request : Security
 - Au niveau du pipeline : Security
 - Dans le GitLab Security Dashboards and Security Center



GITLAB DAST

Shared runner Windows
Terraform, IaC
Auto DevOps
Gitlab DAST

Gratuitement ?

- Possible de tout faire à la main !
- Utilisation du même outil : <https://www.zaproxy.org/>
- Pas de dashboard, pas d'intégration native dans gitlab, il faut faire les choses à la main
- Il faut un runner configuré pour ça

GITLAB ET AU-DELA

Dans ce chapitre nous avons :

- Découvert les Shared Runner Windows.
- Découvert Terraform.
- Découvert Auto DevOps
- Découvert le côté DAST de Gitlab.

Fin

Merci pour votre attention !