

Git/Gitlab

Christophe Dufour

Git

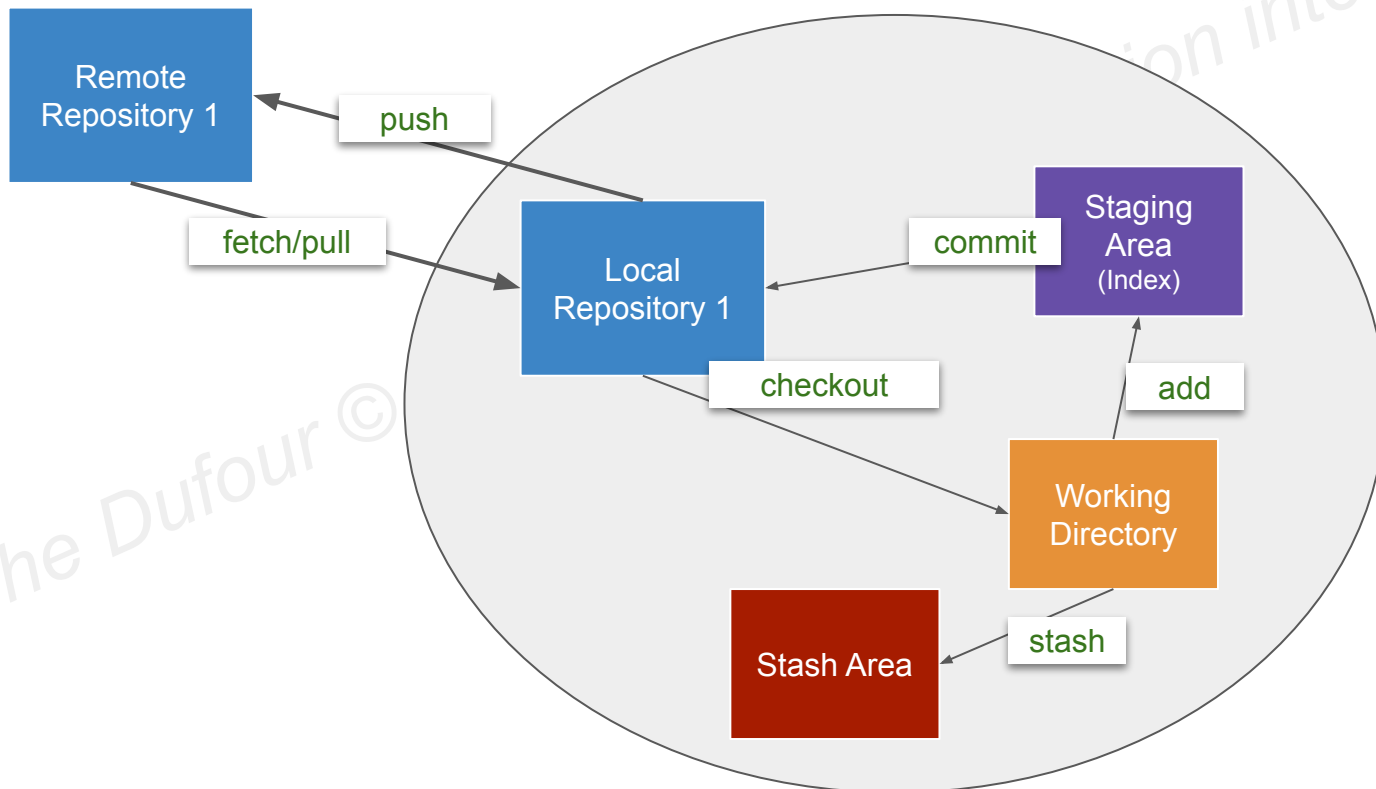
Qu'est-ce que Git ?

- Système de contrôle de version (Version Control System) distribué
 - Permet de mémoriser et retrouver différentes versions d'un projet/fichier
 - Facilite le travail collaboratif
- Créé par T. Linus pour le développement de Linux
- Première version en 2005
- Open Source
- Disponible sur toutes les plateformes

Version Control System (VCS)

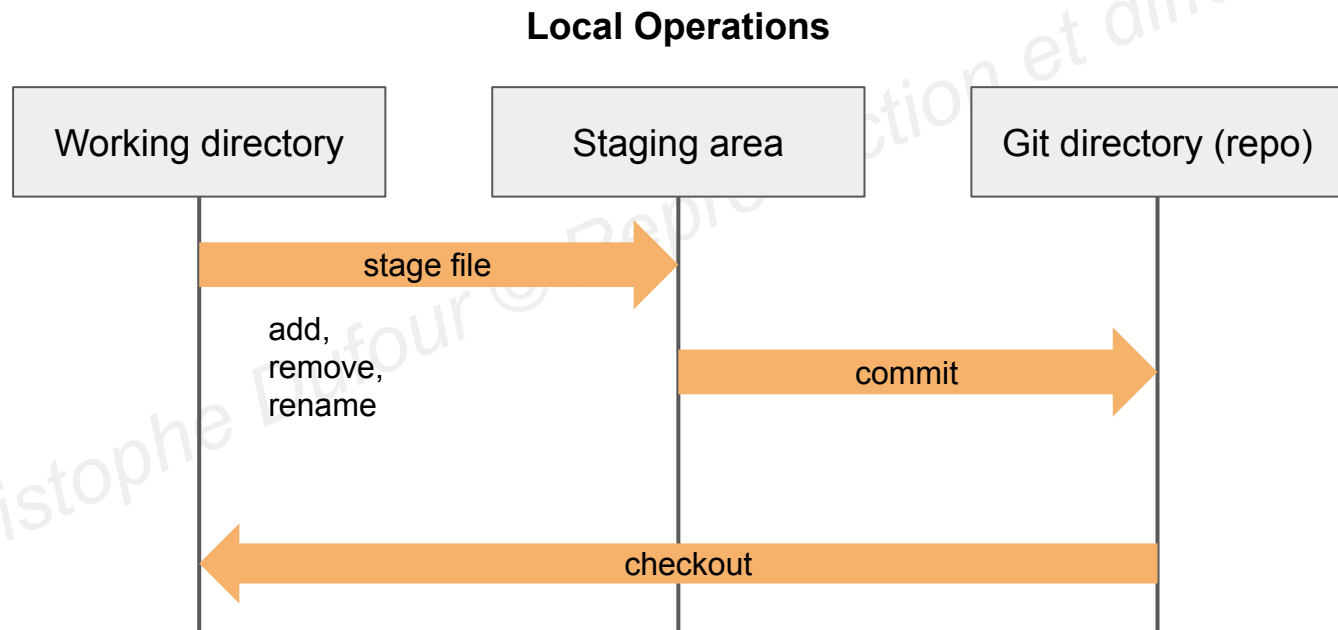
- Définitions simples
 - **Version**: contenu du projet à un moment de son cycle de vie
 - **Dépôt (repository)**: historique du projet, contient toutes ses versions
 - **Branche (branch)**: variante d'un projet
- Pendant la phase de développement d'un projet, il faut:
 - Enregistrer ses versions dans un dépôt afin de pouvoir retrouver une version spécifique (non "buggée" par exemple)
 - Travailler sur différentes variantes (branches) en parallèle, de fusionner ces variantes
 - Partager les modifications faites par différents développeurs

Schéma général



3 zones locales

GIT dispose de 3 zones: le répertoire de travail, la zone d'index, le répertoire git



Configuration

Paramètre le nom qui sera attaché aux commits et aux tags

```
$ git config --global user.name "Your Name"
```

Paramètre l'adresse email qui sera attachée aux commits et aux tags

```
$ git config --global user.email "you@example.com"
```

Démarrer un projet

```
# Génère un nouveau dépôt local. Si [project name] est fourni, Git crée un nouveau dossier et  
# initialise un dépôt dedans ; sinon, un nouveau dépôt est initialisé dans le dossier courant
```

```
$ git init [project name]
```

```
# Télécharge un projet avec son historique complet depuis un dépôt distant
```

```
$ git clone [project url]
```


Tâches courantes 1/2

Affiche l'état du **working directory**

\$ git status

Ajoute un fichier au **staging area**

\$ git add [file]

Affiche les changements entre le **working directory** et **staging area**

\$ git diff [file]

Affiche les changements entre le **staging area** et le **repository**

\$ git diff --staged [file]

Supprime les changements dans le **working directory**. Opération **irréversible**

\$ git checkout [file]

Rétablit le **repository** à un état précédent

\$ git reset [file]

Tâches courantes 2/2

Retire le fichier du **working directory** et du **staging area**

\$ git rm [file]

Crée un nouveau commit à partir des changements ajoutés au **staging area**. Le commit doit être accompagné d'un message

\$ git commit -m "descriptive message"

Stocke les changements du **working directory** dans le **stash** pour un usage futur

\$ git stash

Applique dans le **working directory** les changements stockés puis efface le **stash**

\$ git stash pop

Supprime le stash

\$ git stash drop [stash]

Travail avec les branches

Liste toutes les branches du **repository**. L'option **-a** inclut également les branches distantes
\$ git branch [-a]

Crée une nouvelle branche à partir de la position courante de **HEAD**
\$ git branch [branch_name]

Fait basculer le **working directory** vers le branche spécifiée. L'option **-b** crée la nouvelle
branche spécifiée si elle n'existe pas
\$ git checkout [-b] [branch_name]

Fusionne la branche spécifiée dans la branche courante
\$ git merge [from_branch_name]

Supprime la branche spécifiée
\$ git branch -d [branch_name]

Voir son travail

interdite

Liste l'historique des commits de la branche courante. **-n count** limite la liste aux **count** derniers commits

\$ git log [-n count]

Vue d'ensemble des commits locaux et distants avec labels et historique graphe. Un commit per ligne

\$ git log --oneline --graph --decorate --all

Liste les commits présents dans la branche courante et non fusionnés dans **ref** (ref peut être un nom de
branche ou de tag)

\$ git log ref..

Liste les commits présents dans **ref** et non fusionnés dans la branche courante

\$ git log ..ref

Liste les opérations (checkouts, commits, etc.) effectuées dans le repository local

\$ git reflog

Tags

Liste tous les tags

```
$ git tag
```

Crée un tag référencé **[name]** au niveau du commit courant. **[commit sha]** permet de spécifier le commit
servant de base au tag

```
$ git tag [name] [commit sha]
```

Supprime le tag du repository local

```
$ git tag -d [name]
```

Revenir en arrière (annuler des changements)

Fait basculer la branche courante vers la **[target_reference]**. Tout changement devient "uncommitted change". L'option **--hard** supprime tous les changements

```
$ git reset [--hard] [target_reference]
```

Supprime uniquement les changements apportés par le commit spécifié [commit sha] (sans toucher aux # commits suivants dans l'historique) et crée un nouveau commit

```
$ git revert [commit sha]
```

Travailler avec un repo distant

Télécharge les changements depuis le **remote** mais ne les applique pas localement

\$ git fetch [remote]

Télécharge (fetch) les changements depuis le **remote** et les fusionne dans la branche courante

\$ git pull [remote]

Envoie les changements locaux sur le **remote**. **--tags** pour envoyer les tags également

\$ git push [--tags] [remote]

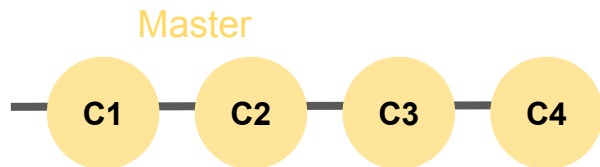
Envoie le branche locale sur le **remote**

\$ git push -u [remote] [branch]

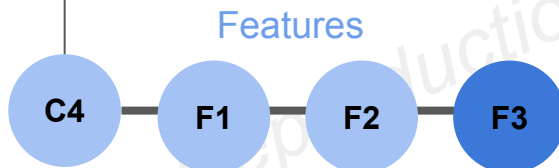
Étapes habituelles pour l'exécution d'un job (tâche)

- Délégation du job à un Gitlab runner
- Téléchargement et démarrage d'une image Docker
- Clonage du dépôt
- Installation des dépendances
- Exécution des étapes (.gitlab-ci.yml)
- Enregistrement du résultat (si nécessaire)

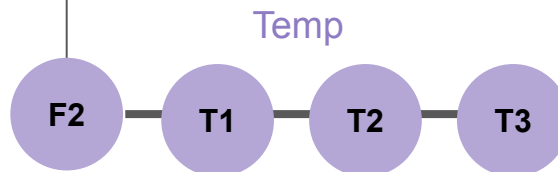
Branches



La branche **Master** se termine au commit C4
La branche **Features** débute au commit C4

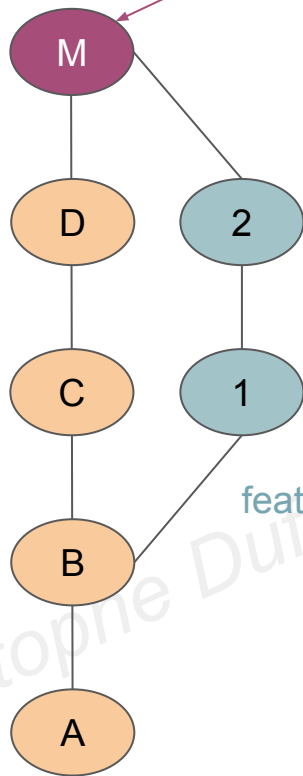


La branche **Temp** débute au commit F2



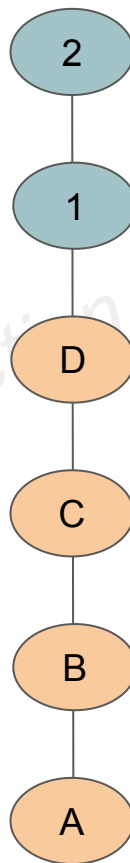
MERGE

Commit de merge



master

REBASE



`git checkout master`
`git rebase features`
`git rebase --continue`

Le merge produit un commit de merge après la résolution des conflits

Le rebase produit un historique linéaire après résolution des conflits

Git stash

- Cette commande permet de mettre de côté des changements qu'on n'est pas sûr de vouloir appliquer immédiatement
- Permet de travailler sur une autre branche, faire des commit...
- ... et d'appliquer les changements mis en attente ("stashed") ultérieurement
- Commandes:
 - **git stash**: met dans la zone de stash les changements
 - **git stash list**: affiche la liste des changements "stashed"
 - **git stash apply**: applique les changements "stashed" dans la branche courante
 - **git stash drop**: supprimer les changements "stashed"

Liens utiles

- <https://learngitbranching.js.org/>
- <https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf>
- <https://books.goalkicker.com/GitBook/>
- <https://www.grafikart.fr/formations/git>

Gitlab

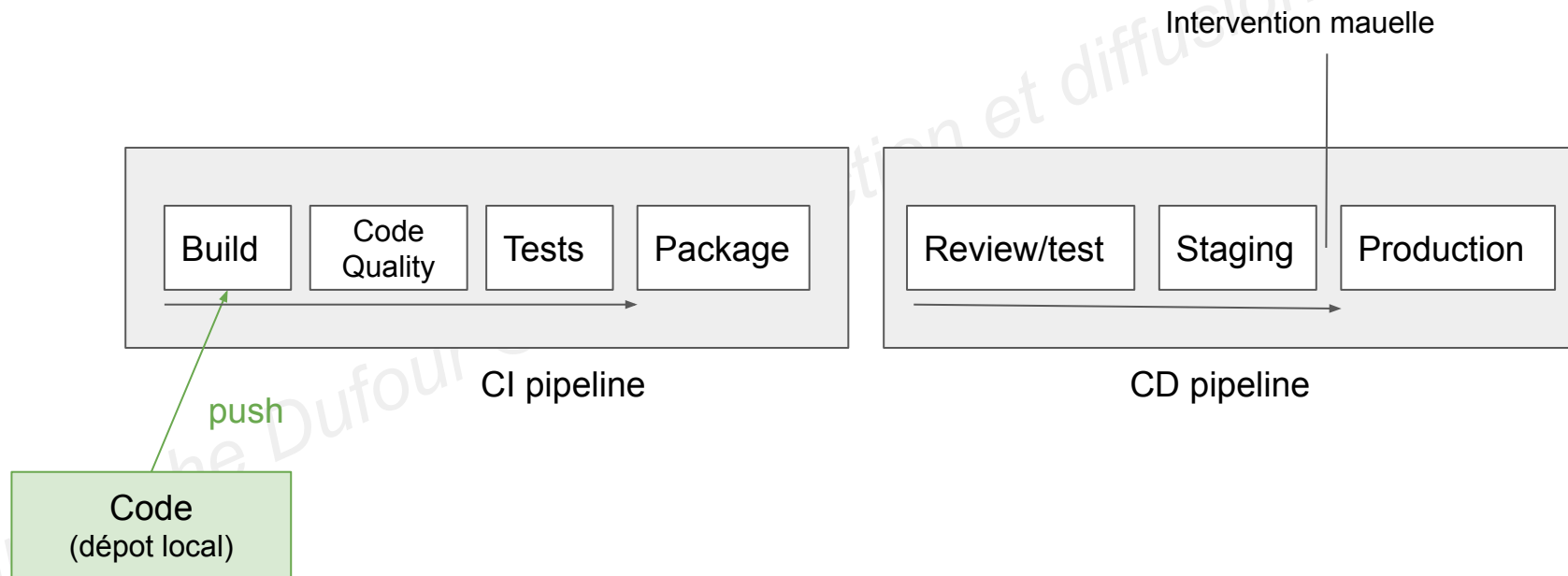
Qu'est-ce que Gitlab ?

- Hôte distant pour dépôts git (comme github)
- Nombreuses fonctionnalités en plus du stockage
 - Wiki
 - Monitoring
 - **CI/CD (fichier .gitlab-ci.yml)**
 - Pages statiques
 - Suivi de bugs
- Première version en 2011
- Open Source

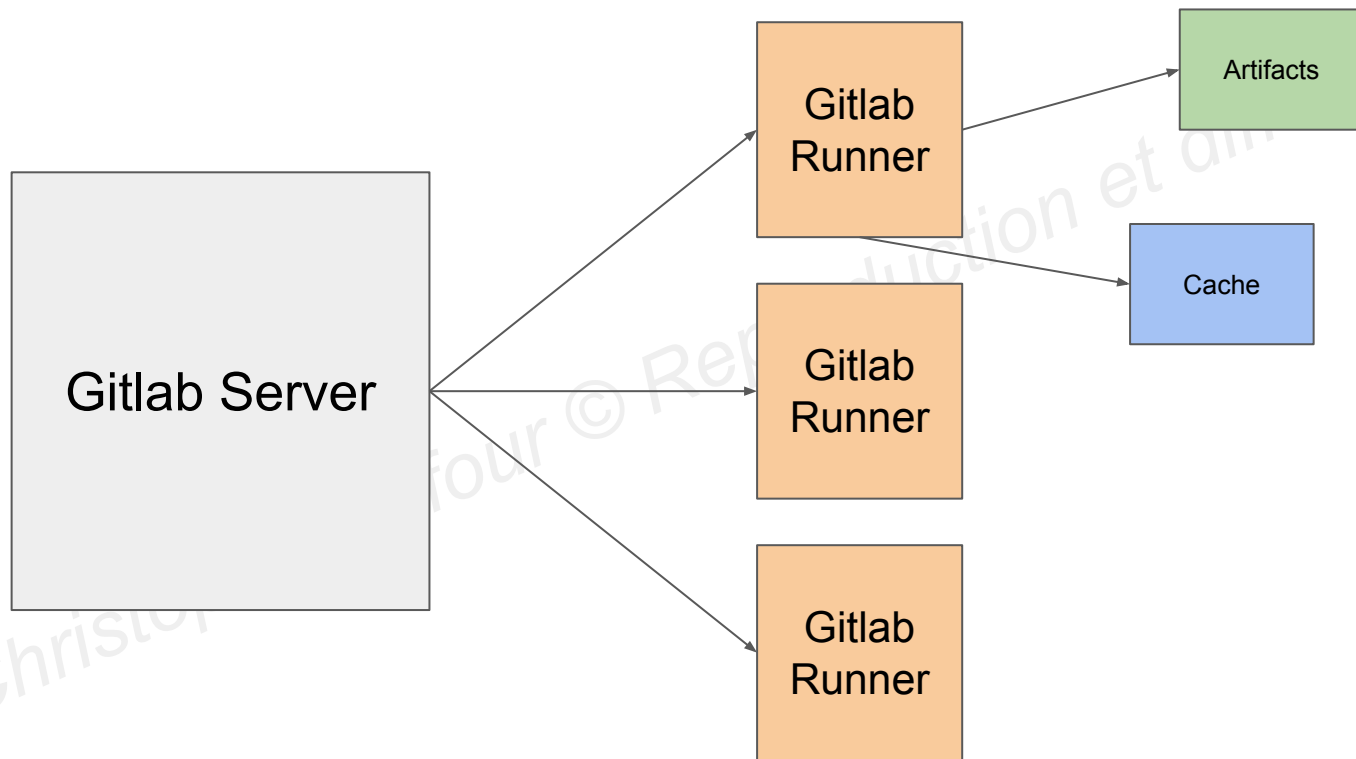
Avantages du CI

- Erreurs détectées tôt dans le processus de développement
- Réduit les problèmes d'intégration
- Permet aux développeurs de travailler plus rapidement
- Assure que chaque changement est "releasable" (peut être déployé)
- Réduit le risque d'un nouveau déploiement
- Fournit de la valeur plus rapidement

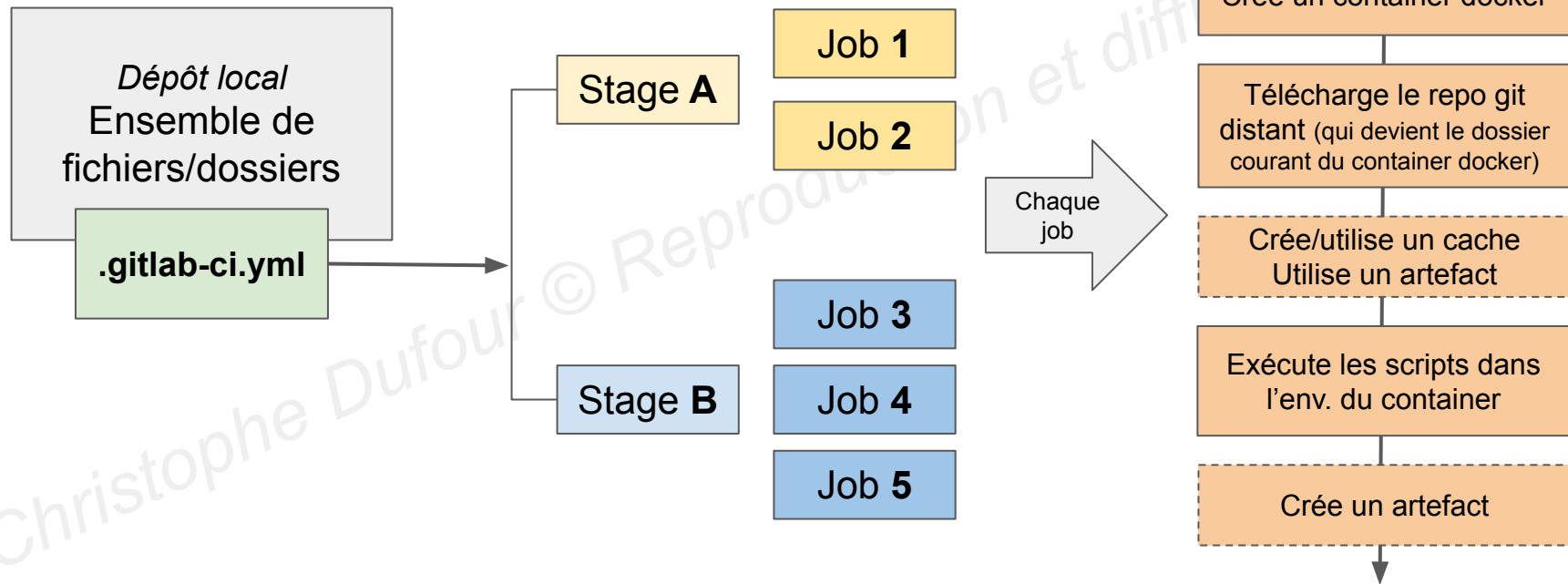
Pipeline classique



Architecture Gitlab



Fichier .gitlab-ci.yml et workflow



Exemple

```
.gitlab-ci.yml X
gitlab-presentation > .gitlab-ci.yml
1  stages:
2    - stageA
3    - stageB
4
5  job1:
6    stage: stageA
7    script:
8      - echo "StageA -> job1"
9  job2:
10   stage: stageA
11   script:
12     - echo "StageA -> job2"
13  job3:
14   stage: stageB
15   script:
16     - echo "StageB -> job3"
17  job4:
18   stage: stageB
19   script:
20     - echo "StageB -> job4"
21  job5:
22   stage: stageB
23   script:
24     - echo "StageB -> job5"
```

commit/push
ayant déclenché la
pipeline

Première étape
de 2 jobs en
parallèle

Deuxième étape
de 3 jobs en
parallèle

The screenshot displays the GitLab CI/CD interface. On the left, a sidebar menu includes 'Project overview', 'Repository', 'Issues', 'Merge Requests', 'CI / CD', 'Jobs', 'Schedules', 'Operations', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', 'Members', and 'Settings'. The 'CI / CD' section is active, showing 'Pipelines' and 'Jobs'. The main area shows a pipeline run for 'Pipeline #194494937' triggered 15 seconds ago. The pipeline status is 'passed'. Below this, a summary shows '5 jobs for master (queued for 1 second)'. The pipeline details section shows a DAG (Directed Acyclic Graph) with two stages: 'Stagea' and 'Stageb'. 'Stagea' contains jobs 'job1' and 'job2', both with a green checkmark. 'Stageb' contains jobs 'job3', 'job4', and 'job5', all with a green checkmark. Arrows from the text annotations point to the 'Repository' menu item, the 'CI / CD' section, and the 'Stagea' and 'Stageb' sections of the DAG.

add gitlab file

Pipeline #194494937 triggered 15 seconds ago by

5 jobs for master (queued for 1 second)

latest

1fecdd9a

No related merge requests found.

Pipeline DAG Jobs 5 Tests 0

Stagea Stageb

job1 job2 job3 job4 job5

Détail d'un job

Opusidea > gitlab-presentation > Jobs > #757551468

✓ passed

Job #757551468 triggered 3 minutes ago by  Opusidea

```
1 Running with gitlab-runner 13.4.0-rc1 (fd488787)
2   on docker-auto-scale z3WU8uu-
3   ✓ Preparing the "docker+machine" executor 00:36
4   Using Docker executor with image ruby:2.5 ...
5   Pulling docker image ruby:2.5 ...
6   Using docker image sha256:70e883b46fffc6e867fa782b3024c9c95366c0aaeffc19c100931b04fe64972c
7   9 for ruby:2.5 with digest ruby@sha256:9aaa579478fe079ad88999a1a551831162a5ac3f916414a4ab
8   a929c85caa2d66 ...
9   ✓ Preparing environment 00:05
10  Running on runner-z3wu8uu--project-21362417-concurrent-0 via runner-z3wu8uu--srm-16010359
11  13-1975e43e...
12  ✓ Getting source from Git repository 00:01
13  $ eval "$CI_PRE_CLONE_SCRIPT"
14  Fetching changes with git depth set to 50...
15  Initialized empty Git repository in /builds/cdufour1/gitlab-presentation/.git/
16  Created fresh repository.
17  Checking out 1fecdd9a as master...
18  Skipping Git submodules setup
19  ✓ Executing "step_script" stage of the job script 00:01
20  $ echo "StageA -> job1"
21  StageA -> job1
22  Job succeeded
```

job1

Retry

Duration: 43 seconds

Timeout: 1h (from project) ?

Runner: shared-runners-manager-7
(#2568000)

Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Commit 1fecdd9a 

add gitlab file

✓ Pipeline #194494937 for master

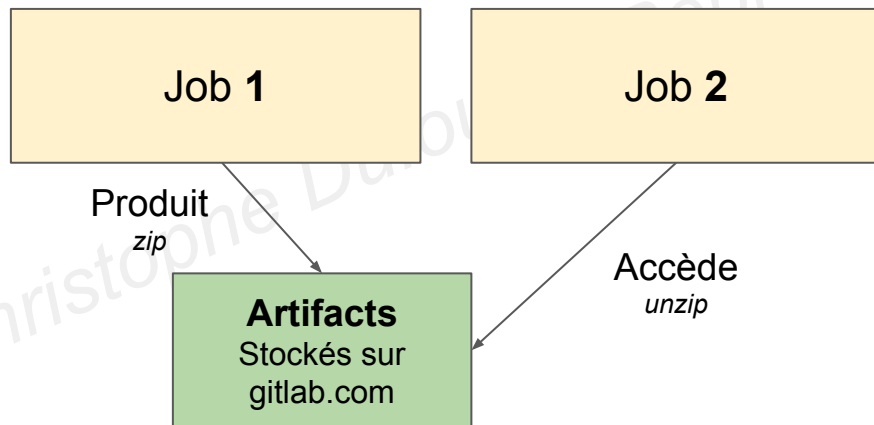
stageA

→ ✓ job1

✓ job2

Les artefacts (artifacts)

- Fichier ou ensemble de fichiers produits par un job (Gitlab runner)
 - Clé **artifacts.paths**: ["path1", "path2", ..."] dans le fichier .gitlab-ci.yml
- Consultables et téléchargeables sur gitlab.com
- Accessibles par les jobs suivants (exemple: tester, déployer le build)
- Moyen de passer le produit (build) d'un job au job suivant



Job produisant un artefact 1/2

```
🔥 .gitlab-ci.yml X
gitlab-presentation > 🔥 .gitlab-ci.yml
1  stages:
2  | ... - one
3
4  # variables accessibles à tous les jobs (référéncées par un "$" initial)
5  variables:
6  | ... ARTIFACT_NAME: poems
7
8  build.artifact:
9  | ... image: alpine # image plus légère que l'image par défaut
10 | ... stage: one
11 | ... script:
12 | ... | ... - mkdir $ARTIFACT_NAME # crée un dossier
13 | ... | ... - cd $ARTIFACT_NAME # entre dans le dossier
14 | ... | ... - echo "Les souvenirs sont cors de chasse" > poem1.txt # file 1
15 | ... | ... - echo "Dont meurt le bruit parmi le vent" > poem2.txt # file 2
16 | ... artifacts:
17 | ... | ... paths:
18 | ... | ... | ... - $ARTIFACT_NAME # artifact = dossier contenant poem1.txt et poem2.txt
```

Job produisant un artefact 2/2

Opusidea > gitlab-presentation > Jobs > #757672756

passed Job #757672756 triggered 1 minute ago by Opusidea

```
1 Running with gitlab-runner 13.4.0-rc1 (fd488787)
2 on docker-auto-scale 72989761
3 Preparing the "docker+machine" executor 00:13
4 Using Docker executor with image alpine ...
5 Pulling docker image alpine ...
6 Using docker image sha256:a24bb4013296f61e89ba57005a7b3e52274d8edd3ae2077d04395f806b63d83
  e for alpine with digest alpine@sha256:185518070891758909c9f839cf4ca393ee977ac378609f700f
  60a771a2dfe321 ...
7
8 Preparing environment 00:01
9 Running on runner-72989761-project-21362417-concurrent-0 via runner-72989761-srm-16010393
  59-96025alb...
10
11 Getting source from Git repository 00:02
12 $ eval "$CI_PRE_CLONE_SCRIPT"
13 Fetching changes with git depth set to 50...
14 Initialized empty Git repository in /builds/cdufour1/gitlab-presentation/.git/
15 Created fresh repository.
16 Checking out 03d26251 as master...
17 Skipping Git submodules setup
18
19 Executing "step_script" stage of the job script 00:00
20 $ mkdir $ARTIFACT_NAME
21 $ cd $ARTIFACT_NAME
22 $ echo "Les souvenirs sont cors de chasse" > poem1.txt
23 $ echo "Dont meurt le bruit parmi le vent" > poem2.txt
24
25 Uploading artifacts for successful job 00:03
26 Uploading artifacts...
27 poems: found 3 matching files and directories
28 Uploading artifacts as "archive" to coordinator... ok id=757672756 responseStatus=201 Cr
  eated token=fPB-CU9_
29
30 Job succeeded
```

build artifact

Retry

Duration: 19 seconds

Timeout: 1h (from project)

Runner: shared-runners-manager-4.gitlab.com (#44949)

Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Keep

Download

Browse

Commit 03d26251

artifact demo

Pipeline #194521712 for master

one

→ build artifact

Opusidea > gitlab-presentation > Jobs > #757672756 > Artifacts

passed

Job #757672756 in pipeline #194521712 for 03d26251

Artifacts / poems

Name

..

poem1.txt

poem2.txt

interdites sans autorisation.

Artefact et jobs 1/2

gitlab-ci.yml X

gitlab-presentation > gitlab-ci.yml

```
1  stages:
2    - one
3    - two
4
5  image: alpine # remplace l'image par défaut pour tous les jobs
6
7  # variables accessibles à tous les jobs (référéncées par un "$" initial)
8  variables:
9    ARTIFACT_NAME: poems
10
11 build_artifact:
12   stage: one
13   script:
14     - mkdir $ARTIFACT_NAME # crée un dossier
15     - cd $ARTIFACT_NAME # entre dans le dossier
16     - echo "Les souvenirs sont cors de chasse" > poem1.txt # file 1
17     - echo "Dont meurt le bruit parmi le vent" > poem2.txt # file 2
18   artifacts:
19     paths:
20       - $ARTIFACT_NAME # artifact = dossier contenant poem1.txt et poem2.txt
21
22 test_artifact:
23   stage: two
24   script:
25     - # exécution d'un script shell placé à la racine le repo git
26     - # ce script prend le nom de l'artefact en entrée
27     - ./test.sh $ARTIFACT_NAME
```

test.sh X

gitlab-presentation > test.sh

```
1  #!/bin/sh
2  echo "*** TEST SCRIPT BY CHRIS (OPUSIDEA) ***"
3
4  DIR="$1"
5  FILES="$DIR/*"
6
7  if [ -d "$DIR" ]; then
8    echo "[+] $DIR folder exists"
9    num_files=$(ls $DIR | wc -l)
10   echo "[+] Contains $num_files files"
11
12   for f in $FILES; do
13     do
14       echo "\t => $f"
15     done
16
17   else
18     echo "[-] $DIR folder do not exist"
19   fi
```

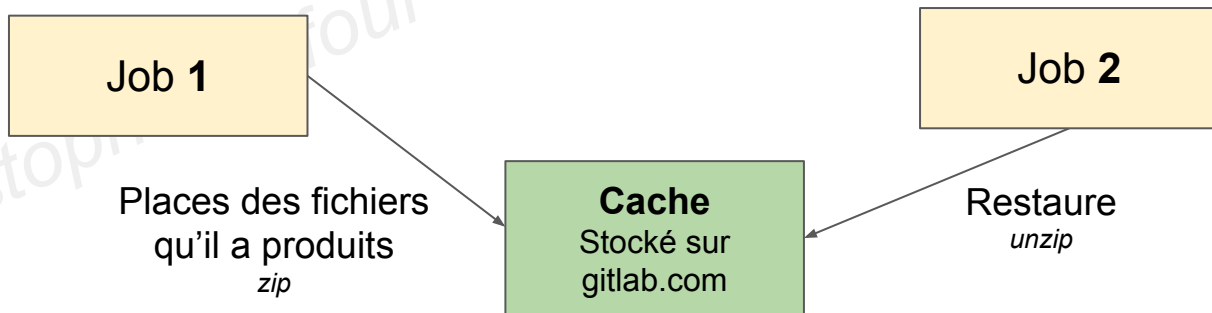

Artefact et jobs 2/2

1 Running with gitlab-runner 13.4.0-rc1 (fd488787)
2 on docker-auto-scale 72989761

> 3 **Preparing the "docker+machine" executor**
> 7 **Preparing environment** 00:01
> 9 **Getting source from Git repository** 00:01
✓ 16 **Downloading artifacts** 00:01
17 **Downloading artifacts for build artifact (757796741)...**
18 Downloading artifacts from coordinator... ok id=757796741 responseStatus=200 OK to ken=LX4KEeTE
✓ 19 **Executing "step_script" stage of the job script** 00:01
20 **\$./test.sh \$ARTIFACT_NAME**
21 ***** TEST SCRIPT BY CHRIS (OPUSIDEA) *****
22 **[+] poems folder exists**
23 **[+] Contains 2 files**
24 **\t => poems/poem1.txt**
25 **\t => poems/poem2.txt**
26 **Job succeeded**

Le cache

- Dans certains cas, optimise le temps d'exécution des runners
- Peut être global ou local (job)
- Si défini globalement, tous les jobs tenteront de le récupérer
- Un job n'ayant pas besoin de récupérer le cache, peut le désactiver (cache: {})
- Ne pas confondre avec les artifacts
- Use case: gestion des dépendances
- Peut être effacé (pipelines/clear cache runners)



Le cache, exemple 1/3

🔥 .gitlab-ci.yml X

gitlab-presentation > 🔥 .gitlab-ci.yml

```
1  stages:
2    ...-step1
3    ...-step2
4
5  image: node:alpine # image incluant les programmes node et npm
6
7  cache:
8    ...key: myKey # nom de la clé au choix
9    ...paths:
10     ...# mise en cache du dossier node_modules créé par "npm install" (step1)
11     ...-node_modules
12
13  install_deps:
14    ...stage: step1
15    ...script:
16     ...-npm install # installe les dépendances (express) listés dans package.json
17
18  # Le job "run server" n'a pas besoin d'installer la dépendance express pour
19  # démarrer le serveur web, il la trouvera dans le cache
20  run_server:
21    ...stage: step2
22    ...script:
23     ...-npm start & # run server in background mode
```

rdites sans autorisation.

Le cache, exemple 2/3

```
1 Running with gitlab-runner 13.4.0-rc1 (fd488787)
2   on docker-auto-scale 72989761
> 3 Preparing the "docker+machine" executor 00:09
> 8 Preparing environment 00:03
> 11 Getting source from Git repository 00:01
✓ 19 Restoring cache 00:01
20 Checking cache for myKey...
21 FATAL: file does not exist
22 Failed to extract cache
✓ 24 Executing "step_script" stage of the job script 00:03
25 $ npm install
26 npm WARN gitlab-presentation@1.0.0 No description
27 added 50 packages from 37 contributors and audited 50 packages in 1.531s
28 found 0 vulnerabilities
✓ 30 Saving cache 00:01
31 Creating cache myKey...
32 node_modules: found 398 matching files and directories
33 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/21
362417/myKey
34 Created cache
36 Job succeeded
```

4.gitlab.com (#44949)

Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Commit [e3934cae](#) 

add cache-demo branch

✓ Pipeline #194583067 for [cache-demo](#)

step1

→ ✓ install deps

Le cache, exemple 3/3

```
1 Running with gitlab-runner 13.4.0-rc1 (fd488787)
2   on docker-auto-scale z3WU8uu-
> 3 Preparing the "docker+machine" executor 00:15
> 8 Preparing environment 00:02
> 11 Getting source from Git repository 00:02
✓ 19 Restoring cache 00:01
20 Checking cache for myKey...
21 Downloading cache.zip from https://storage.googleapis.com/gitlab-com-runners-cache/project/21362417/myKey
22 Successfully extracted cache
✓ 24 Executing "step_script" stage of the job script 00:00
25 $ npm start &
✓ 27 Saving cache 00:01
28 Creating cache myKey...
29 node_modules: found 398 matching files and directories
30 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/21362417/myKey
31 Created cache
33 Job succeeded
```

(#2568000)

Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Commit [11d5de9e](#) 

[add comments](#)

✓ Pipeline #194586409 for [cache-demo](#)

step2

→ ✓ run server

Le cache, erreur en case de désactivation

```
2 on docker-auto-scale fa6cab46
> 3 Preparing the "docker+machine" executor 00:16
> 8 Preparing environment 00:02
> 11 Getting source from Git repository 00:02
> 19 Executing "step_script" stage of the job script 00:01
20 $ npm start
21 > gitlab-presentation@1.0.0 start /builds/cdufour1/gitlab-presentation
22 > node server.js
23 internal/modules/cjs/loader.js:896
24   throw err;
25   ^
26 Error: Cannot find module 'express'
27 Require stack:
28   - /builds/cdufour1/gitlab-presentation/server.js
29     at Function.Module._resolveFilename (internal/modules/cjs/loader.js:893:15)
30     at Function.Module._load (internal/modules/cjs/loader.js:743:27)
31     at Module.require (internal/modules/cjs/loader.js:965:19)
32     at require (internal/modules/cjs/helpers.js:88:18)
33     at Object.<anonymous> (/builds/cdufour1/gitlab-presentation/server.js:1:17)
34     at Module._compile (internal/modules/cjs/loader.js:1076:30)
35     at Object.Module._extensions..js (internal/modules/cjs/loader.js:1097:10)
36     at Module.load (internal/modules/cjs/loader.js:941:32)
37     at Function.Module._load (internal/modules/cjs/loader.js:782:14)
38     at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:72:12) {
39   code: 'MODULE_NOT_FOUND',
40   requireStack: [ '/builds/cdufour1/gitlab-presentation/server.js' ]
41 }
42 npm ERR! code ELIFECYCLE
43 npm ERR! errno 1
44 npm ERR! gitlab-presentation@1.0.0 start: `node server.js`
45 npm ERR! Exit status 1
46 npm ERR!
47 npm ERR! Failed at the gitlab-presentation@1.0.0 start script.
48 npm ERR! This is probably not a problem with npm. There is likely additional logging output above.
49 npm WARN Local package.json exists, but node_modules missing, did you mean to install?
50 npm ERR! A complete log of this run can be found in:
51 npm ERR! /root/.npm/_logs/2020-09-25T16_48_39_112Z-debug.log
52 ERROR: Job failed: exit code 1
```

New issue

Duration: 21 seconds
Timeout: 1h (from project)
Runner: shared-runners-manage-3.gitlab.com (#44028)

Job artifacts
These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Commit d285b2af
disable cache for step2

Pipeline #194622440 for cache-demo

step2

→ run server

disable cache for step2

parent 2c883551 P cache-demo

No related merge requests found

Pipeline #194622440 failed with stages in 1 minute and 26 seconds

Changes 1 Pipelines 1

Showing 1 changed file with 1 addition and 1 deletion

.gitlab-ci.yml

...	...	@@ -20,5 +20,5 @@ install deps:
20	20	run server:
21	21	stage: step2
22	22	script:
23	-	- npm start & # run server in background mode
23	+	- npm start # run server
24	24	cache: {}