Créer son premier package R

Ajouter des vignettes et un site de documentation dans votre package

Juliette ENGELAERE-LEFEBVRE - Maël THEULIERE

Objectif de cet atelier

Après cet atelier vous saurez :

- Ajouter un fichier readme à votre package
- Ce qu'est une vignette de package
- Installer un package en incluant les vignettes
- Ajouter une vignette dans votre package
- Rassembler la documentation du package sous forme de site
- Avoir des notions sur le déploiement de votre site grâce à l'intégration continue

`

Mais qu'est ce qu'une vignette ?

Pourquoi une vignette ?

La vignette est une page html ou pdf, incorporée dans le package qui présente des exemples d'usages des fonctions, leur syntaxe et leurs résultats, entremêlés de commentaires. Elle est produite par un script Rmarkdown lui aussi accessible dans le package.
Elle complète, de manière facultative, la documentation conventionnelle obligatoire de chacune des fonctions.

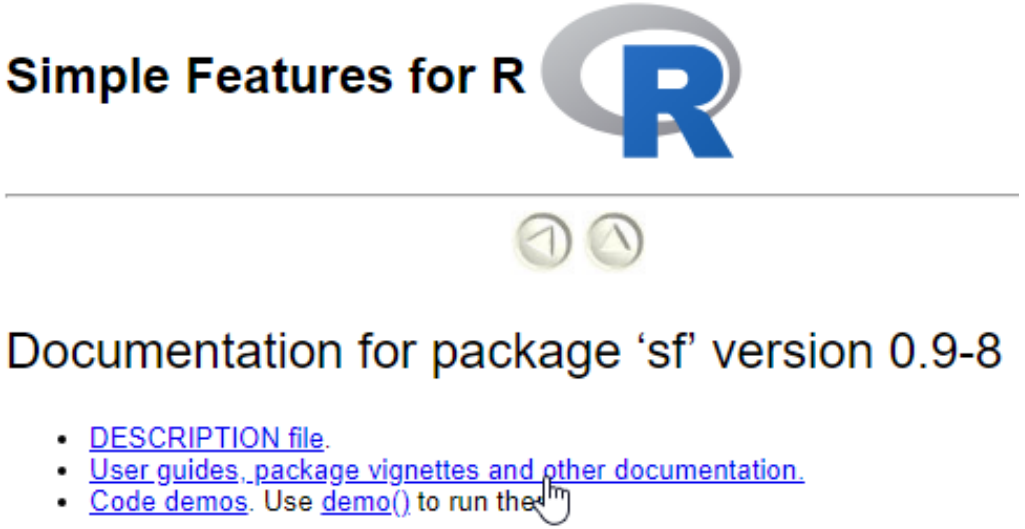
Leur utilité est multiple :

- présenter à l'utilisateur :
    - à quels usages le package sert et pour quels besoins il a été conçu,
    - comment il fonctionne, généralement, en commençant par comment il s'installe,
    - et sa logique d'ensemble, en présentant successivement plusieurs fonctions du package ;
- proposer un tutoriel ;
- vérifier en plus de tests que chaque fonction produit les résultats attendus ;
- ou encore, dialoguer avec le commanditaire d'une publication propre, pour lui présenter le résultat de l'exécution de chaque brique développée, grâce à l'intégration continue.

`

Exemple du package `{sf}` : accès

> Depuis le panneau "Packages" (en bas à droite),
>
> sélectionner `{sf}`
>
> et cliquer sur *User guides, package vignettes and other documentation*
>
> pour avoir accès aux 7 vignettes proposées.



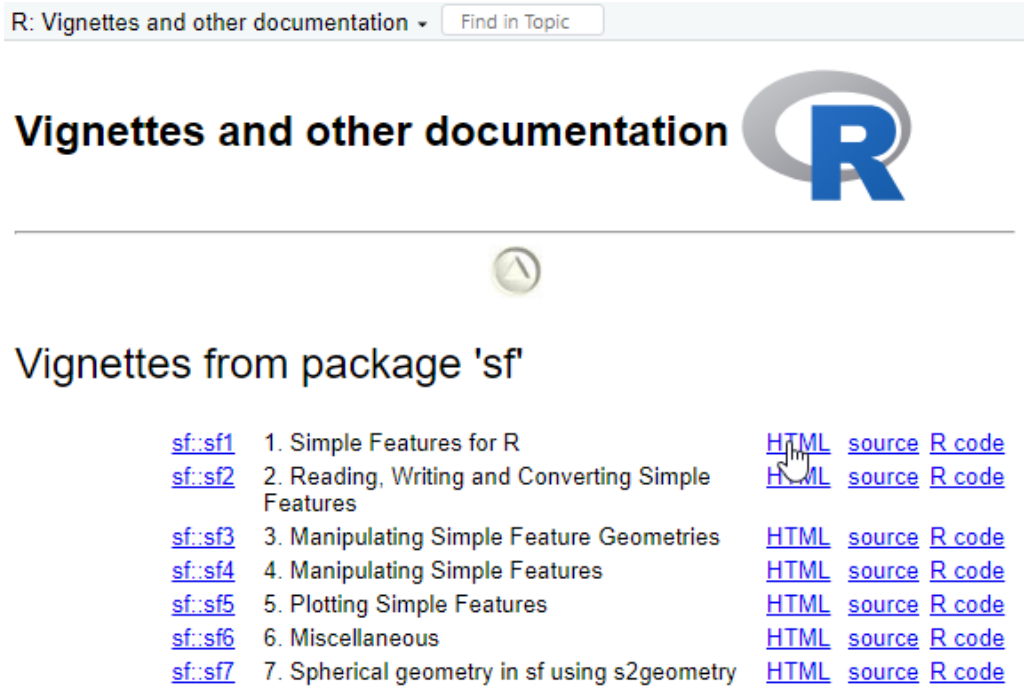
`

Exemple du package `{sf}` : les vignettes disponibles

On a accès :

- à la vignette compilée : `HTML`
  (certains packages proposent plutôt `PDF`),
- au document Rmd source : `source`,
- aux seules portions de code : `R code`.

L'aide de R est inadaptée si on ne connaît pas le nom de la fonction dont on a besoin. Parcourir une vignette permet de découvrir de nouvelles fonctions.



`

Exemple du package `{sf}` : vignette 1

Le lien https://r-spatial.github.io/sf/articles/ invite à consulter la vignette sur le web, produite par l'intégration continue du repo sf github.

Accéder aux vignettes disponibles sur mon poste

La fonction `vignette()` retourne une liste des vignettes disponibles. Cette liste peut être globale et concerner l'ensemble des packages installés, ou partielle en se limitant aux packages en cours d'utilisation avec l'argument `all=FALSE`, ou encore se restreindre à un seul package avec l'argument `package = "DBI"` par exemple, pour lister les seules vignettes du package `{DBI}`.

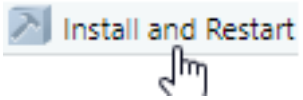
```
vign <- vignette(all=FALSE)
vign$results[, c("Package", "Title")]
```

```
##       Package
##  [1,] "sf"
##  [2,] "sf"
##  [3,] "sf"
##  [4,] "sf"
##  [5,] "sf"
##  [6,] "sf"
##  [7,] "sf"
##  [8,] "showtext"
##  [9,] "xaringanthemer"
## [10,] "xaringanthemer"
## [11,] "xaringanthemer"
## [12,] "stats"
## [13,] "utils"
```

Installer un package en incluant les vignettes

Les vignettes ne sont pas systématiquement installées avec le packages

C'est même le comportement par défaut lorsqu'un package est installé via `remotes::install_github` ou

`remotes::install_gitlab` ou le bouton 

du panneau 'Build'.

Les raisons ?

- cela est consommateur de temps : la vignette est compilée sur votre poste de bout en bout,
- la compilation des vignettes peut échouer si les dépendances ne sont pas les mêmes que celles du package,
- généralement les vignettes sont déjà accessibles sur gitlab ou github pages.

`

Installer localement les vignettes

Et pourquoi ça ?

- naviguer dans l'ensemble de l'aide depuis RStudio,
- vérifier que le package fonctionne correctement sur mon poste.

`

```
remotes::install_gitlab(repo = "dreal-datalab/
                        build_vignettes = TRUE

remotes::install_github("MaelTheuliere/COGiter
                        build_vignettes = TRUE
```

Ajouter une vignette dans notre package

`{usethis}` à la rescousse !

- L'instruction `usethis::use_vignette()` simplifie considérablement l'ajout d'une vignette dans un package.
- La fonction `use_vignette()` prend deux arguments :
  - `name` : le nom du fichier vignette,
  - et `title` : le titre de la vignette.

Placez dans le dev_history.R :

```
usethis::use_vignette(name = "a_mon_premier_pa
                      title = "Mon premier pac
```

et exécutez cette instruction.

> 💡
> En cas de vignettes multiples, si on souhaite ordonner les vignettes pour proposer une progression logique, penser à classer par ordre alphabétique les noms de fichiers .

`

Comportement de `usethis::use_vignette()`

- Cette instruction va :
  - compléter DESCRIPTION avec les dépendances nécessaires à la compilation de la vignette à l'installation du package,
  - compléter le .gitignore,
  - créer le répertoire des vignettes,
  - créer le fichier vignette "nom_fichier.Rmd", initier son contenu et l'ouvrir.

```
usethis::use_vignette(name = "a_mon_premier_pa
√ Setting active project to 'C:/Users/juliette
√ Adding 'knitr' to Suggests field in DESCRIPT
√ Setting VignetteBuilder field in DESCRIPTION
√ Adding 'inst/doc' to '.gitignore'
√ Creating 'vignettes/'
√ Adding '.html', '.R' to 'vignettes/.gitignor
√ Adding 'rmarkdown' to Suggests field in DESC
√ Writing 'vignettes/a_mon_premier_package.Rmd
> Modify 'vignettes/a_mon_premier_package.Rmd'
```

`

Description du fichier ouvert

- une en-tête YAML ;
- un premier chunk de paramétrage des options de compilation du document,
  adaptées aux vignettes
  et invisible dans le document final (compilé) ;
- un second chunk 'setup' qui marque le début de l'histoire à raconter.

Compléter la vignette

On utilise la syntaxe légère de mise en forme markdown pour le texte.

On met les exemples de code dans des chunks, càd entre 2 séries de 3 "'" avec {r nom_du_chunk, options} au début :

```
```{r, eval = FALSE}
mon_resultat <- ma_fonction
mon_resultat
```
```

Si l'option `eval` est fixée à `TRUE`, le résultat sera affiché sous l'instruction dans le document compilé.

On compile le document avec le bouton knit 

pour visualiser le rendu.

Pour en savoir plus sur l'utilisation de Rmarkdown :

- une initiation en français ;
- une feuille de triche ;
- le guide complet.

`

Compléter la vignette

Indiquer par exemple :

- comment s'installe le package
- que le package contient une fonction et un dataset
- et appliquer la fonction au dataset

Et *kniter* pour visualiser le rendu.


````
Le package R `{monpremierpackage}` s'installe

```{r install, eval=FALSE}
remotes::install_gitlab(repo = "jengelaere/mon
```

```{r setup}
library(monpremierpackage)
```

`{monpremierpackage}` contient une fonction `m
On utilise le paramètre `n_head` pour indiquer

Il contient également un jeu de données d'exem


```{r exemple}
mon_resultat <- ma_fonction(data = co2_emissio
mon_resultat
```
````

Ajouter un fichier README

Pourquoi un fichier README ?

Le fichier `README` est une présentation introductive de votre package.

Ce sera souvent le premier contact de vos utilisateurs avec votre package, par exemple ce sera la page d'accueil de votre site pkgdown que nous verrons ensuite.

⚠ Il est donc important de bien le soigner.

Usuellement, on présente dans le `README` l'objet du package, comment l'installer et un premier cas d'usage.

`

Comment ajouter un fichier README ?

🦸 {usethis} bien sûr !

`usethis::use_readme_rmd()` vous permettra de rajouter un fichier readme à votre package, également sous forme de document Rmarkdown.

`

Compléter le README

Indiquer par exemple :

- l'objet du package
- comment s'installe le package
- comment activer le package

Et *kniter* pour compiler le rendu en un fichier `README.md`.


```
L'objectif du package R `{monpremierpackage}`

Il s'installe depuis gitlab avec :

```{r install, eval=FALSE}
remotes::install_gitlab(repo = "jengelaere/mon
```
Il s'active comme cela

```{r setup}
library(monpremierpackage)
```
```

`

Et si, avec cette documentation rédigée, on faisait un site web de présentation du package ?

{pkgdown} est là pour ça

`

Il rassemble la documentation de notre package sous la forme de site web,
avec un **minimum d'efforts**.

{usethis} encore et toujours 🦸

Dans le *dev.history.R*,

1. saisir `usethis::use_pkgdown()` et l'exécuter pour initier la création du site.
    - Des répertoires et fichiers sont ajoutés à la liste des choses à ignorer pour la compilation et le suivi par git.
    - Le fichier de paramétrages complémentaires du site '_pkgdown.yml' est créé et ouvert.
      Il est facultatif !
2. saisir `pkgdown::build_site()` dans le *dev_history.R*, pour terminer.

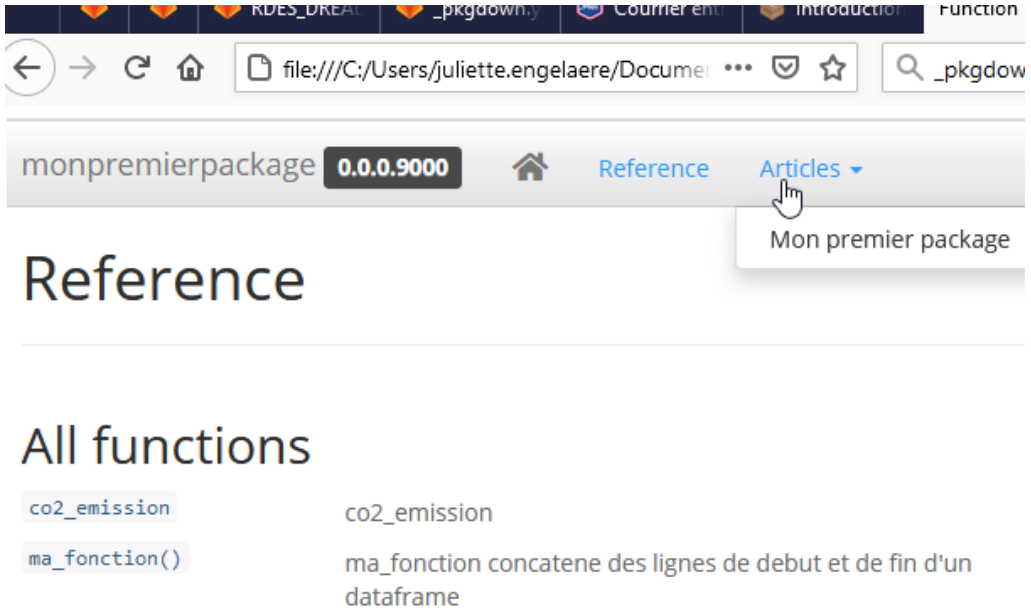
```
usethis::use_pkgdown()
# Setting active project to 'C:/Users/juliette
# Adding '^_pkgdown\\.yml$', '^docs$' to '.Rbu
# Adding '^pkgdown$' to '.Rbuildignore'
# Adding 'docs' to '.gitignore'
# Record your site's url in the pkgdown config
# Modify '_pkgdown.yml'
pkgdown::build_site()
```

Plus d'info sur le paramétrage de notre site sur la vignette de {pkgdown}, voir un exemple basique ici.

`

Produire un site pour notre package

```
== Building pkgdown site ========================
Reading from: 'C:/Users/juliette.engelaere/Doc
Writing to:   'C:/Users/juliette.engelaere/Doc
-- Initialising site ------------------------
Copying '../../../../R/win-library/4.0/pkgdown
Copying '../../../../R/win-library/4.0/pkgdown
Copying '../../../../R/win-library/4.0/pkgdown
Copying '../../../../R/win-library/4.0/pkgdown
Copying '../../../../R/win-library/4.0/pkgdown
Copying '../../../../R/win-library/4.0/pkgdown
Copying '../../../../R/win-library/4.0/pkgdown
Writing '404.html'
-- Building home ----------------------------
Writing 'authors.html'
Reading 'LICENSE.md'
Writing 'LICENSE.html'
-- Building function reference --------------
Writing 'reference/index.html'
Reading 'man/co2_emission.Rd'
Writing 'reference/co2_emission.html'
Reading 'man/ma_fonction.Rd'
Writing 'reference/ma_fonction.html'
-- Building articles ------------------------
Writing 'articles/index.html'
Reading 'vignettes/a_mon_premier_package.Rmd'
Writing 'articles/a_mon_premier_package.html'
== DONE ==========================================
```



Notre site est prêt ! 🎉

`

Configuration

La configuration de votre site pkgdown se fait dans le fichier `_pkgdown.yml`

Vous pouvez notamment :

- ordonner la page des références de vos fonctions et datasets
- organiser les articles correspondants à vos vignettes
- choisir un thème de votre choix parmis ceux disponibles sur bootswatch.com

`

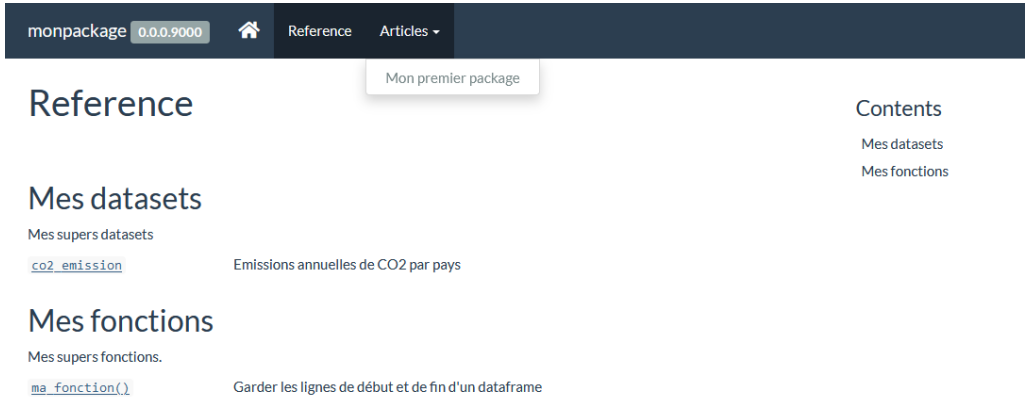
Configuration

Exemple :

```
template:
  params:
    bootswatch: flatly

navbar:
  components:
    accueil: ~

reference:
- title: "Mes datasets"
  desc: >
    Mes supers datasets
- contents:
  - co2_emission
- title: "Mes fonctions"
  desc: "Mes supers fonctions."
- contents:
  - matches("fonction")
```



Bilan

On a maintenant un magnifique site présentant notre package, mais assez difficile d'accès pour l'utilisateur.

Ce dernier peut y avoir accès, une fois notre package installé, via :

```
monpremierpackage::open_pkgdown()
```

Faut le savoir !

`

Et maintenant ?

Déployons notre site grâce à l'intégration continue

La quoi ?

- CI = continuous integration
- un ensemble d'opérations programmées,
- exécutées par gitlab ou github à chaque push,
- qui peuvent :
    - tester notre package,
    - publier ou mettre à jour notre site,
    - publier ou mettre à jour notre application shiny...

Les instructions sont définies dans un fichier YAML, par exemple `.gitlab-ci.yml` pour gitlab, voire dans un dossier de plusieurs fichiers YAML pour Github `.github`.

`

Exemples

Gitlab

`usethis::use_gitlab_ci()`

> √ Writing '.gitlab-ci.yml'
> √ Adding '^\.gitlab-ci\.yml$' to '.Rbuildignore'

Fichier `.gitlab-ci.yml` à compléter, notamment au niveau de l'étape *pages*

Un exemple Gitlab pour un package

`

Github

{usethis} propose plus de fonctions pour Github : une dizaine dédiées à l'intégration continue Github.

`usethis::use_github_actions` -> la plus générique

`usethis::use_github_pages()` -> publier du HTML

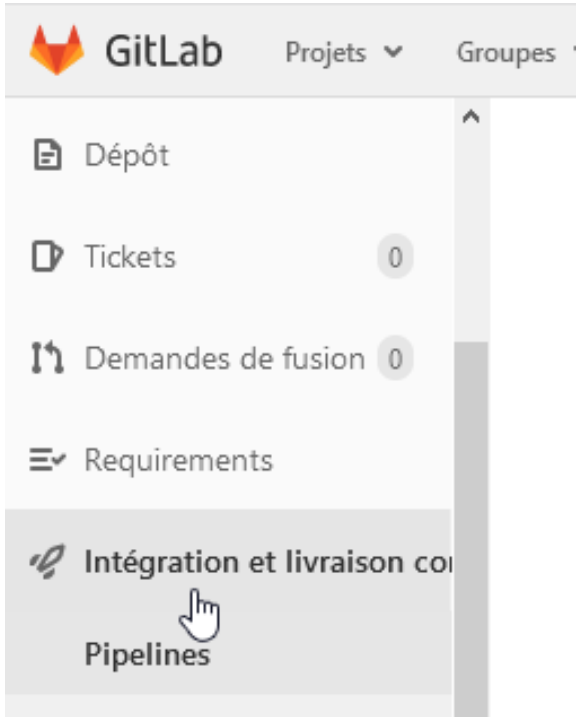
`usethis::use_pkgdown_github_pages()` -> publier un site pkgdown

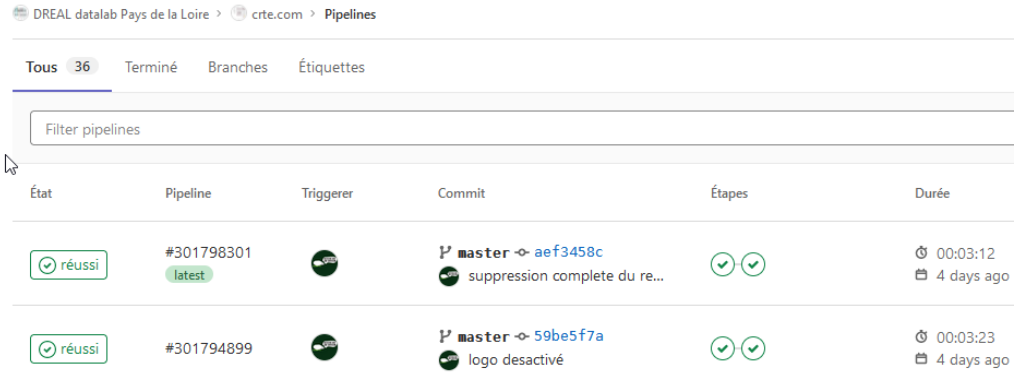
Un exemple concernant `{COGiter}`

En savoir plus sur l'intégration continue sur Github

Accéder aux résultats de la CI sur gitlab

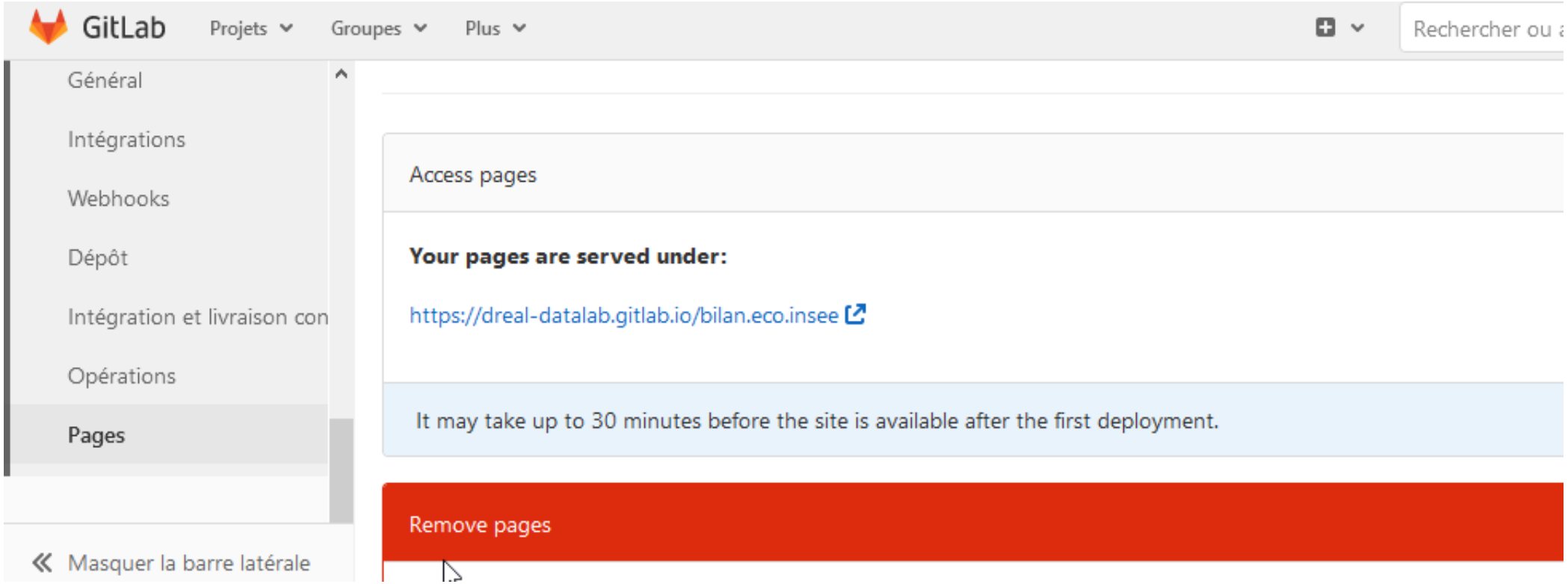
- En cas d'échec un mail est envoyé
- Le menu pipelines et CI de gitlab [`–/pipelines`] :





Accéder aux résultats de la CI sur gitlab

- Pour retrouver l'URL du site publié : aller au dernier menu en bas à gauche : settings/pages [`/pages`]



`

Accéder aux résultats de la CI sur gitlab

- Pour le lien vers le rapport de couverture du code et les badges : menu settings/continuous integration [`/-/settings/ci_cd`]

Merci de votre attention 🙏