

Git

un peu plus en détail

git status

A tout moment dans le terminal l'instruction `git status` nous retournera l'état actuel du dépôt. Il s'agit de la version "ligne de commande" de ce que représente le panneau git de Rstudio.

Pour certain cas complexes, seule cette instruction vous donnera les informations nécessaires à la résolution de votre souci. Ou vous permettra de copier coller dans google les messages d'erreurs que vous risquez de rencontrer.

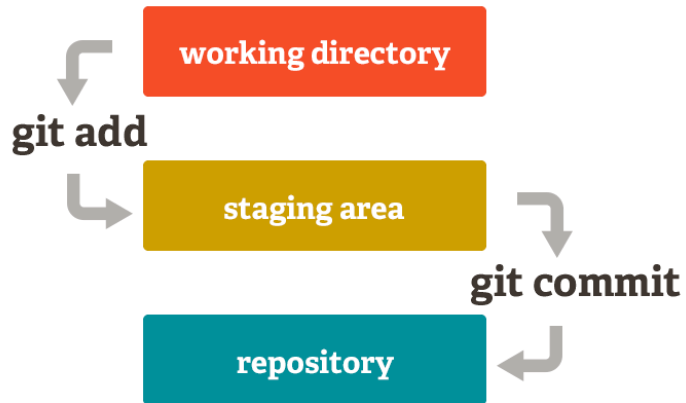
```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    mon premier script.R.orig

nothing added to commit but untracked files present (use "git add" to track)
```

Les trois niveaux de travail en local



- le working directory est votre dossier de travail "classique"
- la zone de staging contient les fichiers que vous avez coché dans l'interface git de Rstudio
- les **commits** sont stockés dans le **repository**.

A chaque action dans l'interface git de Rstudio correspond une ligne de commande, voici un workflow classique :

```
git add . # on coche tout les fichiers
git commit -m "message de commit" # on fait un commit
git pull # on récupère les modifications distantes
git push # on envoie les modifications sur le repository
git checkout -b autre_branche # on construit une branche
```

Le quatrième niveaux 'bonus' de travail en local

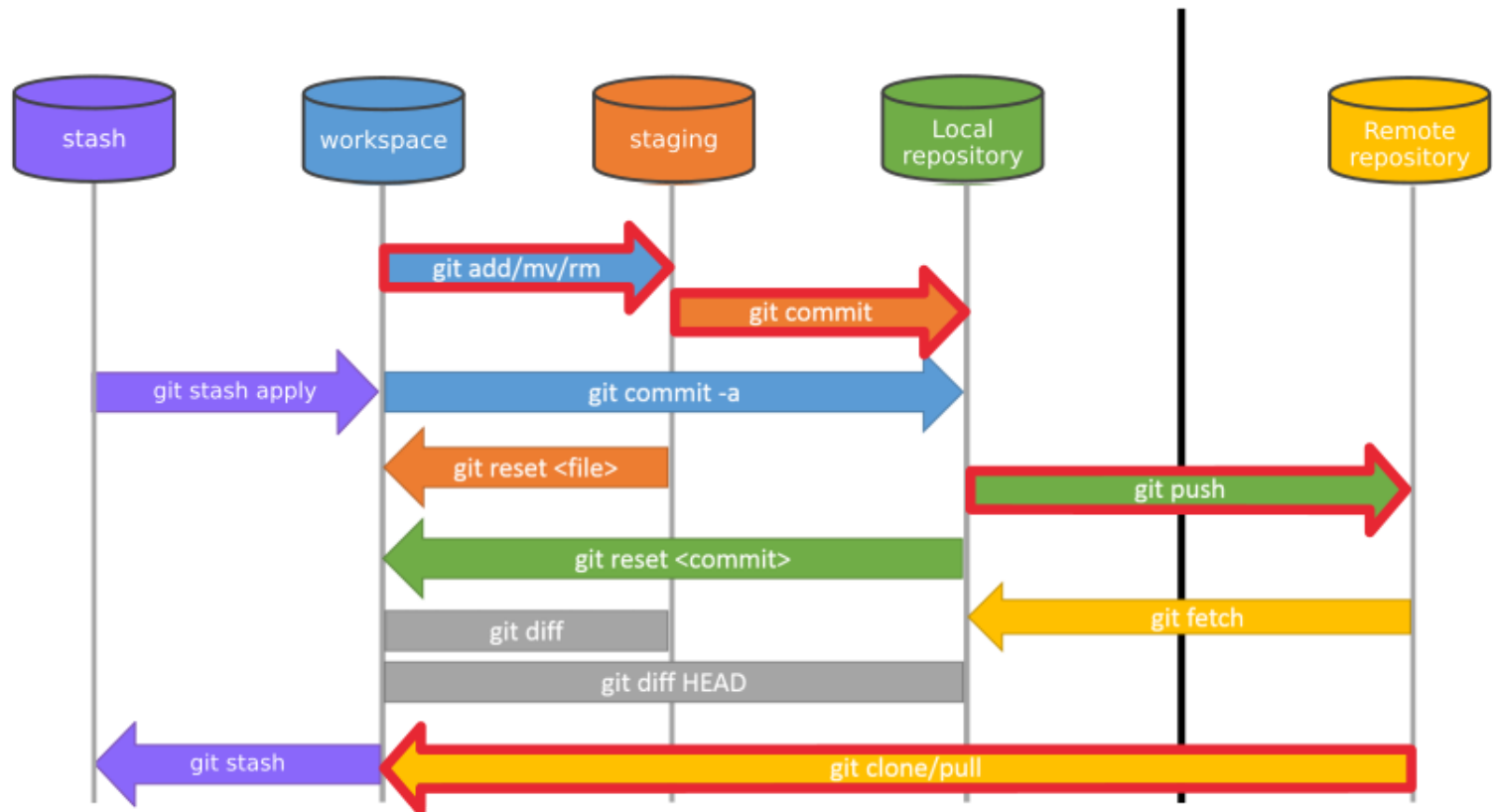
Il existe une 4ème zone de travail appelée **stash**, elle permet d'y remiser des modifications faites dans le working directory, sans devoir les passer en staging (c'est-à-dire les cocher ou les passer dans un commit). Cette zone peut-être vue comme une zone tampon. On l'utilise souvent avant un **pull**, un **merge** ou pour mettre de coté les modifications en cours le temps d'une opération "périlleuse".

Depuis le terminal

```
git stash # envoie le contenu des modifications non suivies dans la  
git stash apply # renvoie la remise vers le working directory
```

Vous pouvez avoir plusieurs remises et les visualiser avec `git stash list`

Vue d'ensemble des commandes de bases



Autour de git

gitlab et github sont bien plus que "git" il s'agit de véritables outils dédiés à la gestion du développement d'un projet. Beaucoup de fonctionnalités annexes y sont implémentés.

On notera tout particulièrement :

- les issues
- le kanban
- l'intégration continue

Autour de git


Les issues


Les issues sont des de taches et/ou de bug relatifs au projets. On peut aisément :

- discuter un point
- définir une échéance
- assigner une personne

ThinkR > supports > Issues

Open 43 Closed 74 All 117

   [Edit issues](#) [New issue](#)

 Search or filter results...

Last updated ▾ 

Dans le cours sur dplyr, on parle deux fois de "data.frame", remplace par "tibble"
#117 · opened 5 days ago by Sébastien

  0
updated 5 days ago

insérer qu'on peut distinct() plusieurs colonnes
#116 · opened 2 weeks ago by Colin

 0
updated 2 weeks ago

Améliorer shiny 3 of 5 tasks completed
#115 · opened 2 months ago by cervan


  2
updated 3 weeks ago

Autour de git

Les issues

Il est possible, et même recommandé d'associer ses **commits** à une issue. Pour cela il faut faire référence à `#numero_issue` dans le message de commit. Le commit viendra alors s'insérer dans la discussion. Des notations spécifiques comme `fix #123` ou `close #123` permettent de fermer une issue via un commit.

golem::get_sysreqs("tidyverse") should return libcurl* #197

 Open VincentGuyader opened this issue 7 days ago · 0 comments



VincentGuyader commented 7 days ago • edited ▾

Member + 😊 ...

the `golem::get_sysreqs("tidyverse")` miss the system requirements to `libcurl14-openssl-dev` ...



VincentGuyader self-assigned this 7 days ago



VincentGuyader added a commit that referenced this issue 7 days ago ⓘ



`fix #197`

c777d05

Assignees



VincentC

Labels

None yet

Projects

None yet

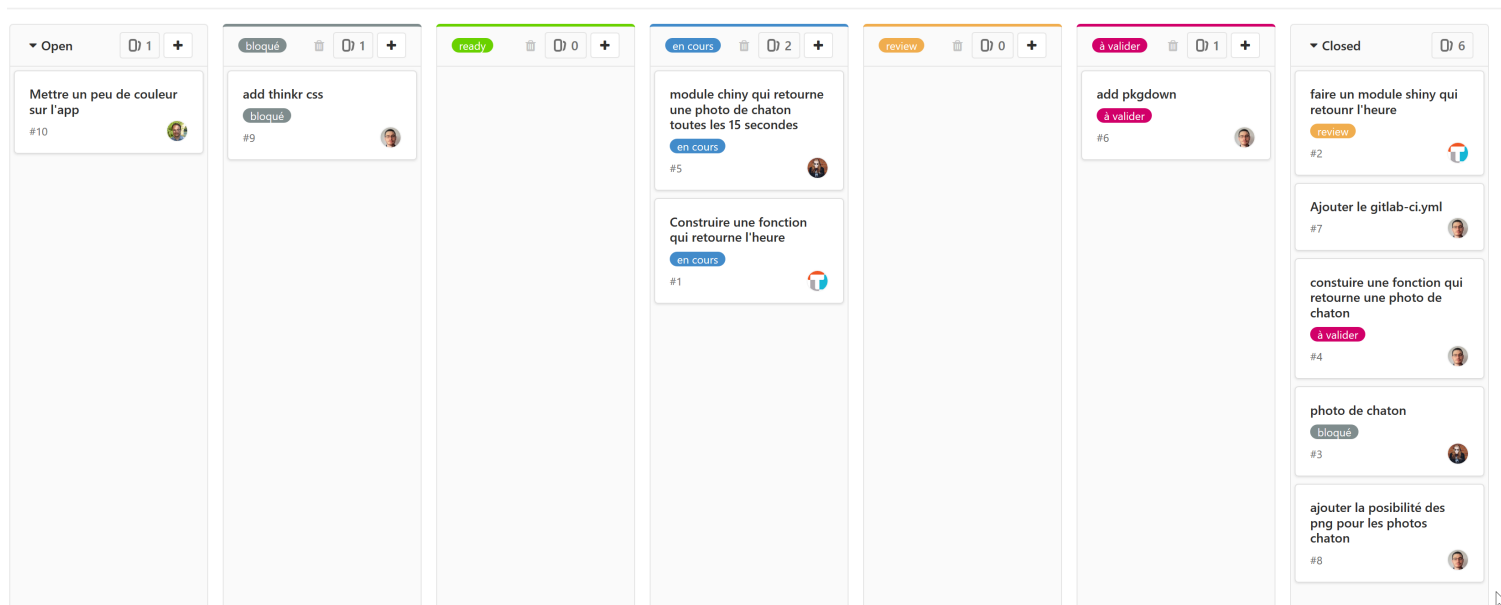
Reactions

Autour de git

Les kanban

Les issues peuvent être associées à des labels en fonction de leurs états. On peut alors voir l'avancement de chacun des points dans le temps et la charge de chacun des contributeurs.

L'idée est de déplacer une issue de la gauche (personne n'a pris en charge le soucis) vers la droite (l'issue est terminée et validée)



Autour de git

L'intégration continue.

Avant de faire un **push** il convient de vérifier que le code envoyé est correct (par exemple dans le cadre de la création d'un package R on lancera un **check**) mais pour plus de prudence on peut demander à gitlab de lui meme lancer les différents test sur le projet. Ces tests peuvent être divers et être lancé dans des environnements différents (ancienne version de R, actuelle et future)

Autour de git

L'intégration continue.

Ces outils se basent sur docker et sont facile a mettre en oeuvre. En quelque clic l'interface est prête, et il ne reste plus qu'à rajouter un fichier `gitlab-ci.yml` à la racine du projet

```
image: rocker/tidyverse

stages:
  - build
  - test
  - deploy

building:
  stage: build
  script:
    - R -e "remotes::install_deps(dependencies = TRUE)"
    - R -e 'devtools::check()'

testing:
  stage: test
  allow_failure: true
  when: on_success
  only:
    - master
  script:
    - Rscript -e 'install.packages("DT")'
```

Git flow

Travailler à plusieurs nécessite de mettre en place une méthode de travail. Elle est à adapter à chaque type de projet et en fonction des équipes. Comment et quand créer une branche, quelle notation, quelle procédure de contrôle... ces notions correspondent à `git flow`



Ressources et crédits

- <https://git-scm.com/book/fr/>
- https://danielkummer.github.io/git-flow-cheatsheet/index.fr_FR.html