Manuel du gem `clir`

```
require 'clir'
```

> NOTE : CE DOCUMENT EST TRÈS LOIN DE PRÉSENTER TOUTES LES MÉTHODES

L'application

Rejouer une commande (mode `Replayer`)

Pour rejouer une commande — pas seulement la commmande mais aussi toutes les entrées qui ont été faites — on joue juste `app _`.

> Attention : il faut vraiment qu'il n'y ait que '_' après la commande et rien d'autre, même pas une option.

C'est une commande particulièrement utile quand on teste l'application car elle évite d'avoir à retaper chaque fois les commandes et les options.

File extensions

`File#tail`

Retourne les x dernières du fichier sans avoir à le lire en intégralité (parfait pour les fichiers énormes.

```
@syntax
  File.tail(path, nombre_lignes)

# Par exemple

File.tail("monfichier.txt", 2000)
# => Retourne les 2000 dernières lignes du fichier (ou moins s'il y en a moins)
# Note : les lignes sont dans le même ordre.
```

CLI Tests

inputs

Simuler l'interruption forcée du programme

Dans l'input, utiliser `EXIT` ou `CTRL-C` (et ses variantes avec tiret plat ou espace) ou `^C`.

Pour `Q.yes?`

`with_inputs` peut contenir :

- "o", "y", 1, "true", "yes", "oui" pour retourner `true`
- toute autre valeur retournera `false`

Pour `Q.select`

`with_inputs` peut contenir :

- la valeur explicite à retourner

  ```
  with_inputs ['valeur explicite']
  ```

- la valeur par le nom du menu (non sensible à la casse)

  ```
  with_inputs [{name:"le nom du menu"}]
  ```

- la valeur par une expression régulière (non sensible à la casse)

  ```
  with_inputs [{rname: "men"}] # => le premier menu contenant "men"
  ```

- la valeur par un index (1-start) de menu

```
with_inputs [{index:2}] # => la valeur du deuxième menu
```

Pour `Q.multiselect`

`with_inputs` peut contenir :

- les valeurs explicites à retourner

```
with_inputs [
  ['valeurs','explicites']
]
```

- les valeur par le nom des menus (non sensible à la casse)

```
with_inputs [{names:["premier", "deuxième"]}]
```

- les valeurs par une expression régulière (non sensible à la casse)

```
with_inputs [{rname: "men"}] # => Tous les menus contenant "men"
```

- les valeurs par plusieurs expressions régulières

```
with_inputs [{rnames: ["men", "nue"]}] # => Tous les menus contenant "men" ou "nue"
```

- les valeurs par liste d'index (1-start) de menu

```
with_inputs [{index:[2,4]}] # => la valeur du 2e et 4e menu
```

Pour `Q.multiline`

On peut envoyer le texte explicite qu'aurait tapé l'utilisateur :

```
with_inputs = ["Le\nTexte\nExplicite"]
```

Pour simuler les touches ^D (control et D), on peut utiliser au choix : `CTRL-D`, `CTRL D`, `CTRL_D` ou `^D`.

> Note : mais cela revient simplement à mettre un texte vide.

Mode interactif/entrées

Parfois, pendant les tests, on peut vouloir taper les entrées à la main (ou par osascript). Dans ce cas, il faut changer TTY-Prompt de mode. Cela se fait à l'aide de :

```
TTY::MyPrompt.set_mode_interactive
# => Q passe en mode interactif, avec l'affichage à l'écran du prompt
```

```
TTY::MyPrompt.unset_mode_interactive
# => Pour revenir en mode par inputs
```

Pour vérifier le mode courant, on peut faire appel à :

```
Q.mode
# => :interactive ou :inputs
```

CLI.options

Pour définir ses propres options en ligne de commande :

```
CLI.set_options_table({
  s: :simulation,
  ...
})
```

> Noter que les deux éléments doivent être des `Symbol`s.