R : Automatisation / production

Datastorm - B. Thieurmel

1. Commencer un nouveau projet **RStudio** et rédiger un script **R** amené à être appeler en ligne de commande :
   - il prendra un seul argument : le chemin vers un fichier de configuration yaml (.yml)
   - le yaml contiendra les trois champs suivants (à adapter...!):

```
path_report : report.txt
path_file : /path/to/data.csv
n_rows : 5
```

- **path_report** : chemin du rapport à écrire
- **path_file** : chemin du fichier à importer
- **n_nrows** : nombre de lignes à afficher

Contenu du script :

- récupération, contrôle (pas d'argument, fichier inexistant, champs présents, ...) et lecture du fichier de configuration (`commandArgs(trailingOnly = TRUE)`, `yaml::yaml.load_file`)
- Importation et contrôle (fichier inexistant, erreur lors de la lecture) du fichier **path_file**
- Ecriture d'un rapport affichant les **n_nrows** premières lignes du fichier importé. Vous pouvez vous inspirez des lignes ci-dessous :

```
sink(conf$path_report) # ouverture de la connexion
cat("Données \n")
cat(format(Sys.time(), "%a, %d %b %Y %H:%M:%S"), "\n\n")
print(head(data, n = conf$n_rows))
cat("\n\n")
sink() # fermeture de la redirection
```

2. Tester votre script en le lançant en ligne de commande depuis un terminal. Il doit générer le rapport.

```
cd /path/to/script
Rscript --vanilla name_script.R /path/to/conf.yml
```

3. Rajouter un fichier de log. Vous pouvez vous inspirez des lignes ci-dessous :

```
require(futile.logger)
flog.appender(appender.file("log.txt"), name="td.io")
# set layout
flog.layout(layout.format('[~t] [~l] ~m'), name="td.io")
# and threshold
flog.threshold("INFO", name = "td.io")

withCallingHandlers({
  ... # code R
}, simpleError  = function(e){
  futile.logger::flog.fatal(
    gsub("^(Error in withCallingHandlers[[:punct:]]{3}[[:space:]]*)|(\n)*$",
         "", e), name="td.io")
}, warning = function(w){
  futile.logger::flog.warn(gsub("(\n)*$", "", w$message), name = "td.io")
}, message = function(m){
  futile.logger::flog.info(gsub("(\n)*$", "", m$message), name = "td.io")
})
```

4. Tester cette nouvelle version. Vérifier que vous avez bien une erreur dans votre fichier de log si l'appel au script n'est pas bon. (Mauvais fichier de configuration par exemple)
5. Relier **packrat** à votre projet :

Initialisation :

```
# initialisation
# préférable de répartir d'une session vide
packrat::init() # si vous etes dans le bon répertoire
packrat::init("/path/to/project")
```

Un dossier *packrat* a été créé. Que contient-il ?

Vérification du status :

```
# status
packrat::status()
```

activation/désactivation :

```
packrat::off()
packrat::on()
```

ajout / suppression de packages :

```
# pas utile dans dans ce TD (à titre indicatif, mais vous pouvez essayer)
# comme dans R, mais packrat le prendra en compte
install.packages(c("plyr", "ibr"))
remove.packages("plyr")

# apres des modifications : aller voir le status et faire un snapshot
packrat::status()
packrat::snapshot()
```

activation de packrat pour votre script :

```
# rajout de packrat::on au début
packrat::on() # si on est sûr d'être dans le bon répertorie
packrat::on("/path/to/project") # peut-être plus sûr...!
```

Tester votre script en le lançant en ligne de commande depuis un terminal.

Création et utilisation d'un bundle :

Pour finir, créer un **bundle** de votre projet et essayer de le déployer dans un autre dossier :

```
# creation d'un bundle
packrat::bundle()

# unbundle
packrat::unbundle(bundle = "/path/project/packrat/bundles/name-2017-12-01.tar.gz",
                  where = "/path/to/new_project")
```

Plus d'informations : https://rstudio.github.io/packrat/