Exécuter un script: la fonction source()

Fabrice Dessaint
Inra, UMR1347 Agroécologie, Dijon

Décembre 2015

La fonction `source()`

Cette fonction permet d'exécuter un script *R* à l'intérieur d'un autre script. Sa syntaxe est donnée ci-dessous :

```
source(file, local = FALSE, echo = verbose, print.eval = echo, verbose = getOption("verbose"),
    prompt.echo = getOption("prompt"), max.deparse.length = 150, chdir = FALSE,
    encoding = getOption("encoding"), continue.echo = getOption("continue"), skip.echo = 0,
    keep.source = getOption("keep.source"))
```

Elle possède plusieurs arguments qui vont moduler la quantité d'information affichée lors de l'exécution du script.

Pour illustrer l'utilisation des principaux arguments de cette fonction, on va utiliser le script `PetitEx.R`. Le script construit 2 objets : le vecteur `x` et le vecteur `y`, qu'il multiplie ensuite.

Il affiche enfin un message de fin de script. Les premières lignes du script sont des commentaires.

```
# ----------------
# Exemple de script
# ----------------
set.seed(1347)
x <- rnorm(n=10,
           mean=0,
           sd=1)
y <- c(2,3)
x * y
cat("C'est fini !!")
```

Le script `PetitEx.R`

L'argument `file`

C'est le seul argument obligatoire de la fonction ; les autres arguments ont des valeurs par défaut. Il renseigne sur le nom et la localisation du script que l'on souhaite exécuter.

Le nom doit être indiqué sous la forme d'une chaîne de caractères entourée de guillemets simples (`'PetitEx.R'`) ou doubles (`"PetitEx.R"`).

La localisation sur la machine locale (le chemin) peut être absolue ou relative et dans ce cas, c'est le répertoire de travail qui sert de référence. L'utilisation de la contre oblique (ou antislash « \ ») dans l'écriture du chemin est interdite et on doit le remplacer par la barre oblique (ou slash « / », même sous le système d'exploitation `Windows`).

Le script peut être localisé sur une machine distance (internet) ; dans ce cas, on fournit une URL à la fonction

```
source(file="PetitEx.R")

C'est terminé !!

# ou
fichier <- "PetitEx.R"
source(file=fichier)

C'est terminé !!
```

Par défaut, l'utilisation de la fonction ne retourne que l'affichage demandé dans le corps du script, par exemple avec les fonctions `print()` ou `cat()`.

Les objets sont créés dans l'environnement de l'utilisateur [1] :

1. C'est celui par défaut. Il correspond à l'argument `local=FALSE`

```
objects()

[1] "fichier" "x"       "y"
```

Des arguments d'affichage

Plusieurs arguments peuvent être utilisés pour afficher une partie des informations contenues ou produites par le script. Dans l'exemple, on calcule le produit des vecteurs `x` et `y`. L'affichage de toutes les sorties se fait avec l'argument `print.eval`

Sa valeur par défaut est celle de l'argument `echo`

```
source(file="PetitEx.R", print.eval=TRUE)

 [1]  1.0209197  1.7094764 -2.0030693  0.9302666  1.0058464 -1.5146650
 [7] -3.0146174  1.4400773  1.3951502  3.5513835
C'est terminé !!
```

L'affichage du code et des sorties se fait avec l'argument `echo=TRUE`.

Pour n'afficher que le code, on utilise `print.eval=FALSE`

```
source(file="PetitEx.R", echo=TRUE)


> set.seed(1347)

> x <- rnorm(n = 10, mean = 0, sd = 1)

> y <- c(2, 3)

> x * y
 [1]  1.0209197  1.7094764 -2.0030693  0.9302666  1.0058464 -1.5146650
 [7] -3.0146174  1.4400773  1.3951502  3.5513835

> cat("C'est terminé !!")
C'est terminé !!
```

Remarque : Les lignes de commentaire ne sont pas affichées.

L'argument `keep.source` permet un affichage du code qui respecte, dans la mesure du possible, la mise en forme du script. Il prend par défaut la valeur de `getOption("keep.source")` .

Sa valeur par défaut est égale à `FALSE`

```
source(file="PetitEx.R", echo=TRUE, keep.source=TRUE)


> # -----------------
> # Exemple de script
```

```
> # ----------------
> set.seed(1347)

> x <- rnorm(n=10,
+            mean=0,
+            sd=1)

> y <- c(2,3)

> x * y
 [1]  1.0209197  1.7094764 -2.0030693  0.9302666  1.0058464 -1.5146650
 [7] -3.0146174  1.4400773  1.3951502  3.5513835

> cat("C'est terminé !!")
C'est terminé !!
```

L'argument `verbose` permet l'affichage d'informations sur le déroulement du script *R*. Les sorties sont plus importantes (plus verbeuses) que pour les 2 arguments précédents. Par défaut, il prend la valeur retournée par `getOption("verbose")`.

Sa valeur par défaut est égale à `FALSE`

```
source(file="PetitEx.R", verbose=TRUE)

'envir' chosen:<environment: R_GlobalEnv>
encoding = "native.enc" chosen
--> parsed 5 expressions; now eval(.)ing them:

>>>> eval(expression_nr. 1 )
 =================

> set.seed(1347)
curr.fun: symbol set.seed
 .. after 'expression(set.seed(1347))'

>>>> eval(expression_nr. 2 )
 =================

> x <- rnorm(n = 10, mean = 0, sd = 1)
curr.fun: symbol <-
 .. after 'expression(x <- rnorm(n = 10, mean = 0, sd = 1))'

>>>> eval(expression_nr. 3 )
 =================

> y <- c(2, 3)
curr.fun: symbol <-
 .. after 'expression(y <- c(2, 3))'

>>>> eval(expression_nr. 4 )
 =================
```

```
> x * y
curr.fun: symbol *
 [1]  1.0209197  1.7094764 -2.0030693  0.9302666  1.0058464 -1.5146650
 [7] -3.0146174  1.4400773  1.3951502  3.5513835
 .. after 'expression(x * y)'

>>>> eval(expression_nr. 5 )
 =================

> cat("C'est terminé !!")
C'est terminé !!curr.fun: symbol cat
 .. after 'expression(cat("C'est terminé !!"))'
```

Des arguments de mise en forme

L'argument `prompt.echo` permet de modifier le(s) caractère(s) utilisés pour l'invite de ligne. Par défaut, c'est la valeur donnée par `getOption("prompt")` qui est utilisée.

La valeur par défaut de l'option « `prompt` » est : `>`

```
source(file="PetitEx.R", echo=TRUE, keep.source=TRUE, prompt.echo="R> ")


R> # -----------------
R> # Exemple de script
R> # ----------------
R> set.seed(1347)

R> x <- rnorm(n=10,
+             mean=0,
+             sd=1)

R> y <- c(2,3)

R> x * y
 [1]  1.0209197  1.7094764 -2.0030693  0.9302666  1.0058464 -1.5146650
 [7] -3.0146174  1.4400773  1.3951502  3.5513835

R> cat("C'est terminé !!")
C'est terminé !!
```

On peut aussi modifier l'invite secondaire avec `continue.echo`. Par défaut, c'est la valeur donnée par `getOption("continue")` qui est utilisée.

La valeur par défaut de l'option « `continue` » est : `+`

```
source(file="PetitEx.R", echo=TRUE, keep.source=TRUE, print.eval=FALSE,
       prompt.echo="R> ", continue.echo="..> ")


R> # -----------------
R> # Exemple de script
```

```
R> # ----------------
R> set.seed(1347)

R> x <- rnorm(n=10,
..>           mean=0,
..>           sd=1)

R> y <- c(2,3)

R> x * y

R> cat("C'est terminé !!")
C'est terminé !!
```

Avec l'argument `skip.echo`, on peut ignorer les n premières lignes de commentaires du script.

```
source(file="PetitEx.R", echo=TRUE, keep.source=TRUE, print.eval=FALSE,
       skip.echo=2)


> # ----------------
> set.seed(1347)

> x <- rnorm(n=10,
+            mean=0,
+            sd=1)

> y <- c(2,3)

> x * y

> cat("C'est terminé !!")
C'est terminé !!
```

Les autres arguments

L'argument `encoding` marque les chaînes de caractères avec le jeu de caractères indiqué. Les chaînes de caractères ne sont pas recodées mais elles sont marquées et donc reconnues par *R*. Ici, on spécifie un codage qui n'est pas celui d'origine.

```
# Le fichier est codé en UTF8
source(file="PetitEx.R", encoding="latin1")

C'est terminÃ© !!
```

On a un affichage incorrect.

L'argument `chdir` permet de changer le répertoire de travail, de façon temporaire. Avec la valeur `FALSE`, c'est le répertoire de travail du script appelant qui

est utilisé

```
fichier <- "Exemples/Exemple2.R"
source(file=fichier, echo=TRUE, chdir=FALSE)


> dir()
[1] "Exemples"             "logoCubeCaSciSDI.png" "PetitEx.R"
[4] "source.pdf"           "source.Rnw"           "source.tex"
```

alors qu'avec la valeur `TRUE`, c'est celui contenant le script appelé qui est utilisé.

```
fichier <- "Exemples/Exemple2.R"
source(file=fichier, echo=TRUE, chdir=TRUE)


> dir()
[1] "Exemple2.R"
```