The REBOL Documentation Project

-- FR - Documentation REBOL - Manuels --

Manuels

Manuel de l'utilisateur - Chapitre 12 - Fichiers

Philippe Le Goff

Première publication : 19 octobre 2005, et
mis en ligne le mercredi 19 octobre 2005

Résumé :

Ce document est la traduction française du Chapitre 12 du User Guide de REBOL/Core, qui concerne les fichiers.

Traducteur : Philippe Le Goff

Ce document est la traduction française du Chapitre 12 du User Guide de REBOL/Core, qui concerne les fichiers.

Traducteur : Philippe Le Goff

1 Historique de la traduction

Date	Version	Commentaires	Auteur	Email
27 juin 2005 19:33	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

2 Présentation

Une caractéristique importante de la puissance de REBOL est sa capacité à manipuler les fichiers et les répertoires. REBOL fournit plusieurs fonctions sur mesure permettant des opérations allant de la lecture d'un simple fichier jusqu'à l'accès direct aux fichiers et répertoires. Pour plus d'informations sur l'accès direct aux fichiers et répertoires, voir le chapitre sur les Ports.

3 Noms et Paths

REBOL fournit une convention de nommage des chemins et des fichiers qui se veut indépendante du type de machines.

3.1 Noms des fichiers

Dans les scripts, les noms de fichiers et les chemins sont écrits avec un signe pourcentage **(%)** suivi par une suite de caractères :

```
%examples.r
%big-image.jpg
%graphics/amiga.jpg
%/c/plug-in/video.r
%//sound/goldfinger.mp3
```

Le signe pourcentage **(%)** est nécessaire pour éviter que les noms de fichiers soient interprétés comme des mots au sein du langage.

Bien que cela ne soit pas une bonne habitude, des espaces peuvent être inclus dans les noms de fichiers en incluant le nom du fichier entre des guillemets **(" ")**.

Les guillemets évitent que le nom du fichiers soit interprété comme une série de plusieurs mots :

```
%"this file.txt"
%"cool movie clip.mpg"
```

La convention standard Internet d'utiliser le signe pourcentage **(%)** et un code hexadécimal est aussi autorisée pour les noms de fichiers. Quand ceci est fait, les guillemets ne sont pas requis.

Les noms de fichiers précédents peuvent encore être écrits sous la forme :

```
%this%20file.txt
```

```
%cool%20movie%20clip.mpg
```

Notez que le suffixe standard pour les scripts REBOL est "**.r**". Sur les systèmes où cette convention produit un conflit avec un autre type de fichier, un suffixe "**.reb**" peut être utilisé à la place.

3.2 Chemins

Les chemins vers les fichiers sont décrits par un signe pourcentage **(%)** suivi par une séquence de noms de répertoires qui sont chacun séparés par le signe **(/)**.

```
%dir/file.txt
%/file.txt
%dir/
%/dir/
%/dir/subdir/
%../dir/file.txt
```

Le caractère standard pour séparer les répertoires est le caractère slash **(/)**, pas le backslash **(\)**. Si des caractères "backslash" sont trouvés, ils sont convertis en caractères "slash" :

```
probe %\some\cool\movie.mpg
%/some/cool/movie.mpg
```

REBOL fournit une méthode standard, indépendante du système d'exploitation pour spécifier les chemins vers les répertoires. Les chemins peuvent être relatifs au répertoire courant ou absolus à partir du plus haut niveau de l'arborescence du système d'exploitation.

Les chemins vers les fichiers qui ne commencent pas par un symbole slash **(/)** sont des chemins relatifs.

```
%docs/intro.txt
%docs/new/notes.txt
%"new mail/inbox.mbx"
```

La convention standard qui utilise deux points successifs **(..)** pour indiquer un répertoire parent ou un point unique **(.)** pour faire référence au répertoire courant est aussi supportée. Par exemple :

```
%.
%./
%./file.txt
%..
%../
%../script.r
%../../plans/schedule.r
```

Les chemins vers les fichiers utilisent la convention Internet qui fait commencer un chemin absolu avec un slash **(/)**. Le slash indique le point de départ depuis le plus haut niveau de l'arborescence du système. (Généralement, les chemins absolus devraient être évités pour rendre les scripts

indépendants du contexte de la machine.

L'exemple :

```
%/home/file.txt
```

devrait référer à un volume disque ou une partition nommée "*home*".

Voici d'autres exemples :

```
%/ram/temp/test.r
%/cd0/scripts/test/files.r
```

Pour faire référence à un disque C comme cela est souvent le cas pour Windows, la notation est :

```
%/C/docs/file.txt
%"/c/program files/qualcomm/eudora mail/out.mbx"
```

Notez que dans les lignes précédentes, le disque C n'est pas écrit avec :

```
%c:/docs/file.txt
```

L'exemple précédent n'est pas indépendant du format utilisé pour la machine et provoque une erreur.

Si le premier nom de répertoire est absent, et que le chemin commence avec deux slashs consécutifs, **(*//*)**, alors le chemin du fichier est relatif au volume courant :

```
%//docs/notes
```

3.3 Sensibilité à la casse

Par défaut, en REBOL, les noms de fichiers ne sont pas sensibles à la casse des caractères. Malgré tout, lorsque de nouveaux fichiers sont créés via le langage, leurs noms conservent la casse avec lesquels ils sont créés :

```
write %Script-File.r file-data
```

L'exemple précédent crée un nom de fichier avec un S et un F majuscules.

De plus, lorsque des noms de fichiers sont issus de répertoires, leur casse est conservée :

```
print read %/home
```

Pour les systèmes sensibles à la casse, comme UNIX, REBOL trouve la correspondance la plus proche pour le nom de fichier.

Par exemple, si un script demande à lire %test.r, mais trouve seulement %TEST.r, ce fichier %TEST.r sera lu. Ce comportement est nécessaire pour permettre de conserver des scripts indépendants du

contexte machine.

3.4 Fonctions de noms de fichiers

Diverses fonctions sont fournies pour vous aider à créer des noms de fichiers et des chemins (paths). Elles sont listées ci-dessous :

to-file : *Convertit des chaînes de caractères et des blocs en noms de fichiers ou en chemin vers un fichier.*

split-path : *Scinde un chemin en deux parties : celle relative au répertoire et celle relative au nom de fichier .*

clean-path : *Renvoie un chemin absolu qui est équivalent à n'importe quel chemin fourni contenant (..) ou (.).*

what-dir : *Renvoie le chemin absolu du répertoire courant.*

4 Lecture des fichiers

Les fichiers sont lus comme des séries de caractères (mode texte) ou d'octets (mode binaire). La source pour le fichier est soit un fichier local sur votre système, ou un fichier à partir du réseau.

4.1 Lecture de fichiers texte

Pour lire un fichier texte local, utilisez la fonction **read** :

```
text: read %file.txt
```

La fonction **read** renvoie une chaîne qui contient le texte entier du fichier. Dans l'exemple ci-dessus, la variable *text* fait référence à cette chaîne.

Au sein de cette chaîne renvoyée par **read**, les fins de lignes sont convertis en caractères newline, indifféremment du genre de fin de ligne qu'utilise votre système d'exploitation. Ceci permet d'écrire des scripts qui recherche des sauts de lignes, sans se soucier des caractères particuliers qui constitue une fin de ligne.

```
next-line: next find text newline
```

Un fichier peut aussi être lu en séparant les lignes, qui sont stockées dans un bloc :

```
lines: read/lines %file.txt
```

Voir le paragraphe "Conversion de Lignes" pour plus d'informations concernant newline et la lecture

ligne par ligne.

Pour lire un fichier par morceaux, utilisez la fonction **open** qui est décrite dans le chapitre sur les Ports.

Pour voir le contenu d'un fichier texte, vous pouvez le lire en utilisant **read** et l'afficher en utilisant **print** :

```
print read %service.txt
I wanted the gold, and I sought it,I scrabbled and mucked like
a slave.
```

4.2 Lecture de fichiers binaire

Pour lire un fichier binaire comme une image, un programme ou un son, utilisez **read/binary** :

```
data: read/binary %file.bin
```

La fonction read/binary renvoie une série binaire qui comprend le contenu entier du fichier. Dans l'exemple ci-dessus, la variable data fait référence à la série binaire. Aucune conversion d'aucun type n'est réalisée pour ce fichier.

Pour lire un fichier binaire par morceaux, utilisez la fonction open comme décrite dans le chapitre sur les Ports .

4.3 Lecture au travers du réseau

Les fichiers peuvent être lus à partir du réseau. Par exemple, pour voir un fichier texte à partir du réseau en utilisant le protocole HTTP :

```
print read http://www.rebol.com/test.txt
Hellotherenewuser!
```

Le fichier peut être écrit localement en une ligne de code :

```
write %test.txt read http:/www.rebol.com/test.txt
```

Dans le processus d'écriture, le fichier aura ses terminaisons de ligne converties en ce qui est utilisé pour cela sur votre système d'exploitation.

Pour lire et sauver un fichier binaire, comme une image, utilisez la ligne suivante :

```
write %image.jpg read/binary http:/www.rebol.com/image.jpg
```

Voyez le chapitre sur les protocoles Réseau pour plus d'information et d'exemples sur les moyens d'accéder à des fichiers au travers le réseau.

5 Ecriture de fichiers

Vous pouvez écrire un fichier de caractères (texte) ou d'octets (binaire). L'emplacement du fichier peut soit être local sur votre système, soit sur le réseau.

5.1 Ecriture de fichiers texte

Pour écrire un fichier texte localement, utilisez la ligne de code suivante :

```
write %file.txt "sample text here"
```

Celle-ci écrit le texte entier dans le fichier. Si un fichier contient des caractères newline, ils seront convertis en ceux que votre système d'exploitation utilise. Ceci permet de manipuler les fichiers d'une manière homogène, mais de les écrire en utilisant la convention en vigueur sur votre système.

Par exemple, la ligne de code suivante transforme n'importe quel texte ayant un style de terminaison de ligne (UNIX, Macintosh, PC, Amiga) en celui utilisé localement par votre système :

```
write %newfile.txt read %file.txt
```

La ligne précédente lit le fichier en convertissant les fins de ligne vers le standard REBOL, puis à l'écriture du fichier, en convertissant celui-ci au format propre au système d'exploitation local.

Pour ajouter quelque chose à la fin d'un fichier, utilisez le raffinement **/append** :

```
write/append %file.txt "encore du texte"
```

Un fichier peut aussi être écrit à partir de lignes distinctes stockées dans un bloc.

```
write/lines %file.txt lines
```

Pour écrire un fichier texte morceaux par morceaux, utilisez la fonction open décrite dans le chapitre sur les Ports.

5.2 Ecriture de fichiers binaires

Pour écrire des fichiers binaires comme une image, un programme, un son, utilisez **write/binary** :

```
write/binary %file.bin data
```

La fonction **write/binary** crée le fichier si celui-ci n'existe pas ou l'écrase s'il existe. Aucune conversion n'est réalisée pour le fichier.

Pour écrire un fichier binaire morceaux par morceaux, utilisez la fonction **open** décrite dans le chapitre sur les Ports.

5.3 Ecriture de fichiers sur le réseau

Les fichiers peuvent aussi être écrits sur le réseau. Par exemple, pour écrire un fichier texte en utilisant le protocole FTP, utilisez :

```
write ftp://ftp.domain.com/file.txt "save this text"
```

Le fichier peut être lu localement et écrit sur un emplacement en réseau via une ligne comme celle-ci

```
write ftp://ftp.domain.com/file.txt read %file.txt
```

Dans le processus, le fichier a ses fins de lignes convertis au format standard CRLF. Pour écrire un fichier binaire comme une image, via le réseau, utilisez le code suivant :

```
write/binary ftp://ftp.domain.com/file.txt/image.jpg read/binary %image.jpg
```

Voyez le chapitre sur les protocoles Réseau pour plus d'information et d'exemples sur les moyens d'accéder à des fichiers via le réseau.

6 Conversion de Ligne

Quand un fichier est lu en tant que texte, toutes les fins de lignes sont converties en caractères **newline** (line feed). Les caractères LF (utilisés comme caractères de fin de lignes pour Amiga, Linux, et les systèmes UNIX), les retours chariots CR (utilisés sur Macintosh) ou la combinaison CR/LF (PC et Internet) sont tous transformés en leurs équivalents **newline**.

L'usage d'un caractère standard au sein d'un script permet de le faire marcher indépendamment du contexte machine. Par exemple, pour chercher et compter tous les caractères **newline** à l'intérieur d'un fichier texte :

```
text: read %file.txt
count: 0
while [spot: find text newline][
    count: count + 1
    text: next spot
]
```

La conversion de ligne est aussi pratique pour la lecture de fichiers distants :

```
text: read ftp://ftp.rebol.com/test.txt
```

Quand un fichier est écrit, le caractère **newline** est converti dans le type de fin de ligne pour le système d'exploitation cible. Par exemple, le caractère newline est transformé en CRLF pour les PCs,

LF sur UNIX ou AMIGA, ou CR pour un Macintosh. Les fichiers sur le réseau sont écrits avec CRLF.

La fonction suivante transforme n'importe quel texte, quel que soit le style de fin de ligne en celui qu'utilise le système d'exploitation local :

```
convert-lines: func [file] [write file read file]
```

Le fichier est lu et tous les caractères de fin de ligne sont transformés en caractère newline, puis le fichier est écrit et les caractères newline sont convertis dans le type nécessaire pour le système d'exploitation local.

La conversion de ligne peut être désactivée en lisant le fichier texte en mode binaire.

Par exemple, la ligne suivante :

```
write/binary %newfile.txt read/binary %oldfile.txt
```

préserve les fins de lignes du fichier texte original (*%oldfile.txt*).

7 Blocs de Lignes

Les fichiers textes peuvent facilement être atteints et gérés sous forme de lignes individuelles, plutôt que comme une seule série de caractères. Par exemple, pour lire un fichier sous la forme d'un bloc de lignes :

```
lines: read/lines %service.txt
```

L'exemple précédent renvoie un bloc contenant une série (au sens REBOL) de chaînes de caractères (une pour chaque ligne), sans caractères de fin de ligne. Les lignes vierges sont représentées par des chaînes vides.

Pour afficher une ligne spécifique, vous pouvez utiliser le code suivant :

```
print first lines
print last lines
print pick lines 100
print lines/500
```

Pour afficher toutes les lignes d'un fichier, utilisez l'exemple de code suivant.

```
foreach line lines [print line]
I wanted the gold, and I sought it,
I scrabbled and mucked like a slave.
Was it famine or scurvy -- I fought it;
I hurled my youth into a grave.
I wanted the gold, and I got it --
Came out with a fortune last fall, --
Yet somehow life's not what I thought it,
```

```
And somehow the gold isn't all.
```

Pour afficher toutes les lignes qui contiennent la chaîne "*gold*", utilisez la ligne de code suivante :

```
foreach line lines [
   if find line "gold" [print line]
]
I wanted the gold, and I sought it,
I wanted the gold, and I got it --
And somehow the gold isn't all.
```

Vous pouvez écrire un fichier texte ligne par ligne en utilisant la fonction **write** avec le raffinement **/lines** :

```
write/lines %output.txt lines
```

Pour écrire un fichier à partir de lignes spécifiques d'un bloc, utilisez :

```
write/lines %output.txt [
    "line one"
    "line two"
    "line three"
]
```

En fait, les fonctions **read/lines** et **write/lines** peuvent être combinées pour traiter des fichiers ligne par ligne. Par exemple, le code suivant efface tous les commentaires d'un script REBOL :

```
script: read/lines %script.r
foreach line script [
    where: find line ";"
    if where [clear where]
]
write/lines %script1.r script
```

Le bout de script précédent est indiqué à des fins de démonstration. En effet, en plus d'effacer les commentaires, le code effacerait aussi les points virgules valides présents dans les chaînes de caractères entre apostrophes.

Les fichiers peuvent également être lus ligne par ligne depuis le réseau :

```
data: read/lines http://www.rebol.com

print pick (read/lines ftp://ftp.rebol.com/test.txt) 3
```

8 new

Le raffinement **/lines** peut aussi être utilisé avec la fonction **open** pour lire une ligne à la fois à partir d'une saisie à la console. Voir le chapitre sur les Ports pour plus d'information.

De surcroît, **/lines** peut servir, avec le raffinement **/append**, à ajouter des lignes à un fichier, depuis un bloc.

9 Information sur les fichiers et les répertoires

Il existe de nombreuses fonctions permettant d'avoir des informations utiles sur un fichier comme : s'il existe, sa taille en octets, lorsqu'il a été modifié, ou s'il s'agit d'un répertoire.

9.1 Contrôle de Répertoire

Pour déterminer si un nom de fichier est celui d'un répertoire, utilisez la fonction **dir ?**.

```
print dir? %file.txt
false
print dir? %.
true
```

La fonction **dir ?** fonctionne également avec la plupart des protocoles réseau :

```
print dir? ftp://www.rebol.com/pub/
true
```

9.2 Existence de fichier

Pour déterminer si un fichier existe, utilisez la fonction **exists ?** :

```
print exists? %file.txt
```

Pour savoir si un fichier existe avant d'essayer de le lire :

```
if exists? file [text: read file]
```

Pour éviter d'écraser un fichier, vous pouvez contrôler sa présence :

```
if not exists? file [write file data]
```

La fonction **exists ?** fonctionne aussi avec la plupart des protocoles réseau :

```
print exists? ftp://www.rebol.com/file.txt
```

9.3 Taille de fichier

Pour obtenir la taille en octets d'un fichier, utilisez la fonction **size ?** :

```
print size? %file.txt
```

La fonction **size ?** est utilisable avec la plupart des protocoles réseau :

```
print size? ftp://www.rebol.com/file.txt
```

9.4 Date de modification d'un fichier

Pour obtenir la date à laquelle un fichier a été modifié, utilisez la fonction **modified ?** :

```
print modified? %file.txt
30-Jun-2000/14:41:55-7:00
```

Tous les systèmes d'exploitation ne conservent pas la date de création d'un fichier, donc pour gardez les scripts REBOL indépendants du système d'exploitation, utilisez **modified ?** juste lorsque la date de dernière modification est accessible.

La fonction **modified ?** fonctionne aussi avec la plupart des protocoles réseaux :

```
print modified? ftp://www.rebol.com/file.txt
```

9.5 Information relative à un fichier

La fonction **info ?** récupère toutes les informations sur les fichiers et les répertoires en même temps. Ces informations sont retournées sous la forme d'un objet :

```
probe info? %file.txt
make object! [
    size: 306
    date: 30-Jun-2000/14:41:55-7:00
    type: 'file
]
```

Pour afficher des informations concernant tous les fichiers du répertoire courant, utilisez :

```
foreach file read %. [
    info: info? file
    print [file info/size info/date info/type]
]
build-guide.r 22334 30-Jun-2000/14:24:43-7:00 file
code/ 11 11-Oct-1999/18:37:04-7:00 directory
data.r 41 30-Jun-2000/14:41:36-7:00 file
file.txt 306 30-Jun-2000/14:41:55-7:00 file
```

La fonction **info ?** est utilisable avec beaucoup de protocoles réseau :

```
probe info? ftp://www.rebol.com/file.txt
```

10 Répertoires

Il y a plusieurs fonctions prêtes à l'emploi pour lire des répertoires, gérer des sous-répertoires, créer de nouveaux répertoires, renommer et effacer des fichiers.

De plus, il existe les fonctions standards, pour connaître, modifier et lister le répertoire courant.

Pour plus d'informations concernant l'accès aux répertoires, voir le chapitre relatif aux Ports.

10.1 Lire un répertoire

Les répertoires sont lus de la même manière que les fichiers. La fonction **read** renvoie un bloc de noms de fichiers au lieu de données texte ou binaires.

Pour connaître tous les noms de fichiers du répertoire courant, utilisez la ligne suivante de code :

```
read %.
```

L'exemple précédent lit le répertoire entier et renvoie un bloc composé des noms des fichiers.

Pour afficher les noms de tous les fichiers dans un répertoire, utilisez la ligne de code suivante :

```
print read %intro/
CVS/ history.t intro.t overview.t quick.t
```

A l'intérieur du bloc renvoyé, les noms des répertoires sont spécifiés avec un slash final. Pour afficher chaque nom de fichier sur un ligne différente, saisissez :

```
foreach file read %intro/ [print file]
CVS/
history.t
intro.t
overview.t
quick.t
```

Voici une manière facile d'afficher seulement les répertoires qui ont été trouvés :

```
foreach file read %intro/ [
    if #"/" = last file [print file]
]
CVS/
```

Si vous voulez lire un répertoire présent sur le réseau, n'oubliez pas d'inclure le symbole "slash" à la fin de l'URL pour indiquer au protocole que vous faites référence à un répertoire :

```
print read ftp://ftp.rebol.com/
```

10.2 Créer un répertoire

La fonction **make-dir** permet de créer un nouveau répertoire.

Le nom du nouveau répertoire doit être relatif au répertoire courant ou à un chemin absolu.

```
make-dir %new-dir
make-dir %local-dir/
make-dir %/work/docs/old-docs/
```

Le slash final est optionnel pour cette fonction. En interne, la fonction **make-dir** appelle la fonction **open** avec le raffinement **/new**.

La ligne :

```
close open/new %local-dir/
```

crée également un nouveau répertoire. Le slash final est par contre important dans cet exemple, car il indique qu'un répertoire doit être créé plutôt qu'un fichier.

Si vous utilisez la fonction **make-dir** pour créer un répertoire déjà existant, une erreur sera générée. L'erreur peut être capturée avec la fonction **try**. L'existence du répertoire doit être contrôlée auparavant avec la fonction **exists ?**.

10.3 Renommage des répertoires et des fichiers

Pour renommer un fichier, utilisez la fonction **rename** :

```
rename %old-file %new-file
```

L'ancien nom de fichier doit inclure le chemin d'accès complet au fichier, mais ceci n'est pas nécessaire pour le nouveau nom de fichier. En effet, la fonction **rename** n'est pas destinée à déplacer des fichiers entre différents répertoires.(Beaucoup de systèmes d'exploitation ne permettent pas cette fonctionnalité.)

```
rename %../docs/intro.txt %conclusion.txt
```

Si l'ancien nom de fichier est un répertoire (indiqué par un slash final), la fonction **rename** renommera le répertoire :

```
rename %../docs/ %manual/
```

Si le fichier ne peut être renommé, une erreur se produira. L'erreur peut être capturée avec la fonction **try**.

10.4 Effacer des répertoires et des fichiers

Les fichiers peuvent être effacés avec la fonction **delete** :

```
delete %file
```

Le fichier à supprimer doit avoir un chemin d'accès complet :

```
delete %source/docs/file.txt
```

Un bloc de plusieurs fichiers au sein d'un même répertoire peut aussi être supprimé en une fois :

```
delete [%file1 %file2 %file3]
```

Un ensemble de fichiers peut être supprimé en utilisant un caractère joker et le raffinement **/any** :

```
delete/any %file*
delete/any %secret.?
```

Le caractère joker "astérisque" **(*)** est équivalent à "tous les caractères", et le caractère joker "point d'interrogation" **(?)** équivaut à remplacer un unique caractère.

Pour supprimer un répertoire, mettez à son nom le slash final :

```
delete %dir/
delete %../docs/old/
```

Si le fichier ne peut être supprimé, une erreur sera générée. Il est possible de capturer cette erreur avec la fonction **try**.

10.5 Répertoire courant

Utilisez la fonction **what-dir** pour déterminer le répertoire courant :

```
print what-dir
/work/REBOL/
```

La fonction **what-dir** fait référence au répertoire courant relatif au script en oeuvre comme indiqué dans la variable **system/script/path**.

10.6 Modifier le répertoire courant

Pour modifier le répertoire courant, utilisez la fonction **change-dir** :

```
change-dir %new-path/to-dir/
```

Si le slash final n'est pas inclus, la fonction l'ajoute.

10.7 Listing du Répertoire courant

Pour lister le contenu du répertoire courant, utilisez :

```
list-dir
```

Le nombre de colonnes utilisées pour afficher le contenu du répertoire dépend de la taille de fenêtre de la console, et de la longueur maximale des noms de fichiers.