The REBOL Documentation Project

-- FR - Documentation REBOL - Manuels --

Manuels

Manuel de l'utilisateur - Chapitre 5 - Les scripts

Philippe Le Goff

Première publication : 14 septembre 2005, et mis en ligne le mercredi 14 septembre 2005

Résumé :

Ce document est la traduction française du Chapitre 5 du User Guide de REBOL/Core, qui concerne les scripts.

Ce document est la traduction française du Chapitre 5 du User Guide de REBOL/Core, qui concerne les scripts.
Traducteur : Philippe Le Goff

1 Historique de la traduction

Date	Version	Commentaires	Auteur	Email
20 mai 2005 - 21:31	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

2 Présentation générale

Le terme "script" fait référence non seulement à de simples fichiers à évaluer mais aussi à du code incorporé dans d'autres types de fichiers (comme des pages Web), ou à des morceaux de code source sauvegardés en tant que fichiers de données ou transmis comme messages.

2.1 Suffixe des scripts

Typiquement, les scripts REBOL prennent un suffixe **.r** à leur noms de fichiers. Cependant, cette convention n'est pas obligatoire. L'interpréteur lit les fichiers quelque soit leur suffixe et recherche dans leur contenu un en-tête (**header**) valide de script REBOL.

2.2 Structure

La structure d'un script est libre. L'indentation et l'usage d'espaces peuvent être utilisés pour clarifier la structure et le contenu d'un script. De plus, vous êtes encouragés à utiliser les conventions standard de style REBOL pour rendre vos scripts universellement lisibles. Voir le Guide de Style pour plus d'informations, à la fin de ce chapitre.

3 En-têtes

3.1 En-tête

Précédant directement le corps du script, chaque script doit posséder un en-tête qui identifie son propos, ainsi que d'autres attributs du script.

Un en-tête de script (le "header") peut contenir le nom du script, de l'auteur, la date, le numéro de version, le nom du fichier, et des informations supplémentaires.

Les fichiers de données REBOL qui ne sont pas destinés à une évaluation directe ne nécessitent pas un en-tête.

Les en-têtes sont pratiques pour plusieurs raisons.

Ils identifient un script comme étant du texte source valide pour l'interpréteur REBOL. L'interpréteur utilise l'en-tête pour afficher le titre du script, et déterminer quelles ressources et quelles versions sont

nécessaires avant d'évaluer le script. Les en-têtes fournissent un moyen normalisé de communiquer le titre, le sujet, l'auteur et d'autres détails du script.

Vous pouvez souvent déterminer à partir d'un en-tête de script si celui-ci vous intéressera.

Les bibliothèques de scripts et les sites web utilisent les en-têtes pour générer automatiquement les répertoires, les catégories de scripts, et les références croisées.

Certains éditeurs de texte permettent de consulter et de mettre à jour les en-têtes de scripts, afin d'historer les informations comme l'auteur, la date, la version, et l'historique.

La forme générale d'un en-tête de script est :

```
REBOL [block]
```

Pour que l'interpréteur reconnaisse l'en-tête, le bloc doit immédiatement suivre le mot REBOL. Seuls des espaces blancs (espaces, tabulations, et lignes) sont autorisés entre le mot REBOL et le bloc.

Le bloc qui suit le mot REBOL va présenter le script. Il est souhaitable d'avoir au minimum un en-tête du type :

```
REBOL [
    Title:  "Scan Web Sites"
    Date:   2-Feb-2000
    File:   %webscan.r
    Author: "Jane Doer"
    Version: 1.2.3
]
```

Lorsqu'un script est chargé, le bloc d'en-tête est évalué et les valeurs sont attribuées aux mots leur faisant référence. Ces valeurs sont utilisées par l'interpréteur (pour construire un objet REBOL **system/script/header**) et peuvent aussi être utilisées par le script lui-même.

Notez que les mots définis avec des valeurs uniques peuvent aussi être définis avec des valeurs multiples, il suffit de fournir ces valeurs dans un bloc :

```
REBOL [
    Title: "Scan Web Sites"
    Date:   12-Nov-1997
    Author: ["Ema User" "Wasa Writer"]
]
```

Les en-têtes peuvent être plus complexes, avec des informations sur l'auteur, la license, le formatage, les versions requises, l'historique des révisions, et plus encore.

Puisque le bloc est utilisé pour construire l'objet "**header**" (**system/script/header**), il peut aussi être étendu avec de nouvelles informations.

Cela signifie qu'un en-tête de script peut être adapté selon le besoin, mais ceci devrait être fait avec précaution pour éviter les ambiguïtés ou la redondance d'informations.

Un en-tête complet pourrait ressembler à quelque chose comme cela :

```
REBOL [
    Title:   "Full REBOL Header Example"
    Date:    8-Sep-1999
    Name:    'Full-Header  ; Pour le titre de la fenêtre

    Version: 1.1.1
    File:    %headfull.r
    Home:    http://www.rebol.com/rebex/

    Author:  "Carl Sassenrath"
    Owner:   "REBOL Headquarters"
    Rights:  "Copyright (C) Carl Sassenrath 1999"

    Needs:   [2.0 ODBC]
    Tabs:    4

    Purpose: {
        The purpose or general reason for the program
        should go here.
    }

    Note: {
        An important comment or notes about the program
        can go here.
    }

    History: [
        0.1.0 [5-Sep-1999 "Created this example" "Carl"]
        0.1.1 [8-Sep-1999 {Moved the header up, changed
            comment on extending the header, added
            advanced user comment.} "Carl"]
    ]

    Language: 'English
]
```

3.2 Scripts avec préambules

L'écriture des scripts n'impose pas de commencer avec un en-tête. Les scripts peuvent commencer avec n'importe quel texte, ce qui leur permettrait d'être inclus dans des messages emails, des pages web, et d'autres fichiers.

Cependant, l'en-tête marque pour REBOL le début du script, et le texte qui le suit est le corps du script. Le texte qui apparait avant un en-tête est appelé le préambule ("*preface*") et ce texte est ignoré pendant l'évaluation.

Le texte qui apparaît avant les en-têtes est ignoré par REBOL et peut donc être utilisé pour des commentaires, des en-têtes d'email, des balises HTML, etc.

```
un essai de script
REBOL [
  Title:   "Exemple de préambule"
  Date:    8-Jul-1999
]

print "Ce fichier a un préambule avant son en-tête"
```

3.3 Inclusion de scripts

Si un script doit être suivi par du texte sans lien avec le script lui-même, le script doit être placé entre deux crochets [ ] :

```
Un peu de texte avant le script.

[
    REBOL [
        Title:   "Embedded Example"
        Date:    8-Nov-1997
    ]
    print "done"
]

Ici du texte après le script.
```

Seul un espace blanc est compris entre le premier crochet et le mot REBOL.

4 Arguments des scripts

Quand un script est évalué, il peut accéder aux informations le concernant.

Celles-ci se trouvent dans l'objet REBOL **system/script**. Cet objet contient les champs pour le titre du script (**system/script/title**), les informations mises dans l'en-tête (**system/script/header**), et plus encore.

Item	Description
Title	Le libellé décrivant le script
Header	L'en-tête du script, sous la forme d'un objet. Peut être utilisé pour accéder au titre du script, au nom de l'auteur, sa version, la date, et d'autres champs.
Parent	Si le script a été évalué à partir d'un autre script, cet objet donne des informations pour ce script parent.
Path	La localisation (path) du fichier dans l'arborescence des fichiers, ou l'URL à partir de laquelle le script a été évalué.
Args	Les arguments du script. Ils sont passés depuis la ligne de commande, ou à partir de la fonction do qui a été utilisée pour évaluer le script.

Quelques exemples illustrant l'usage de l'objet **system/script** :

```
print system/script/title

print system/script/header/date

do system/script/args

do system/script/path/script.r
```

Le dernier exemple évalue un script appelé script.r, se trouvant dans le même répertoire (system/script/path) que le script en cours d'évaluation.

4.1 Options de programme

Les scripts peuvent aussi accéder aux options fournies à l'interpréteur REBOL quand il a été démarré. Elles peuvent être trouvées dans l'objet **system/option**.

Cet objet contient les champs suivants :

Item	Description
Home	Le chemin vers l'interpréteur, dans l'environnement de votre système d'exploitation. Il s'agit du chemin indiqué dans la variable HOME de votre environnement, ou de la base de registre si votre système l'exploite. C'est le chemin utilisé aussi pour trouver les fichiers rebol.r et user.r.
Script	Le nom du fichier de script intialement fourni quand l'interpréteur a été lancé.
Path	Le chemin vers le répertoire courant.
Args	Les arguments initiaux fournis à l'interpréteur en ligne de commande.
Do-arg	La chaîne fournie en argument à l'option —do en ligne de commande.

L'objet **system/options** peut aussi contenir des options supplémentaires qui ont été fournies en ligne de commande.

Saisissez :

```
probe system/options
```

pour étudier le contenu de l'objet **system/options**.

Exemples :

```
print system/options/script

probe system/options/args

print read system/options/home/user.r
```

5 Exécution des scripts

Il y a deux manières d'exécuter un script : en tant que script initial, quand REBOL est démarré, ou à partir de la fonction **do**.

Pour exécuter un script au lancement de l'interpréteur, donnez le nom du script en ligne de commande juste après le nom de l'interpréteur (rebol ou rebol.exe) :

```
rebol script.r
```

Lors de l'initialisation de l'interpréteur, le script sera évalué.

A la fonction **do**, fournissez le nom de fichier du script en argument. Le fichier sera chargé dans l'interpréteur et évalué :

```
do %script.r
```

```
do http://www.rebol.com/script.r
```

La fonction **do** renvoie le résultat du script quand l'évaluation est terminée. Notez que le script doit inclure en en-tête (header) REBOL valide.

5.1 Chargement des scripts

Les scripts peuvent être chargés comme des données, avec la fonction **load**.

Cette fonction lit le script, et traduit le script en valeurs, mots et blocs, mais sans évaluer le script. Le résultat de la fonction **load** est un bloc, sauf si une seule valeur a été chargée, auquel cas cette valeur est retournée.

L'argument de la fonction **load** est un nom de fichier, une URL, ou une chaîne :

```
load %script.r
load %datafile.txt
load http://www.rebol.org/script.r
load "print now"
```

La fonction **load** réalise les étapes suivantes.

Elle

- lit le texte du fichier, de l'URL, ou de la chaîne de caractères.
- recherche un en-tête de script, s'il est présent.
- traduit les données trouvées après l'en-tête, s'il y en a.
- renvoie un bloc contenant les valeurs traduites.

Par exemple, si un script appelé *buy.r* contient le texte :

```
Buy 100 shares at $20.00 per share
```

il pourra être chargé avec la ligne :

```
data: load %buy.r
```

et il renverra dans un bloc :

```
probe data
[Buy 100 shares at $20.00 per share]
```

Remarquez que l'exemple "Buy" précédent est un dialecte de REBOL et non du code directement exécutable. Voir le chapitre 4 sur les Expressions pour plus d'information.

A noter aussi qu'un fichier ne nécessite pas un en-tête pour être chargé. L'en-tête est nécessaire seulement si le fichier doit être exécuté en tant que script.

La fonction **load** possède quelques raffinements dont voici la description :

Item	Description
/header	Inclue l'en-tête (header) s'il est présent
/next	Charge seulement la valeur suivante, une valeur à la fois. Ceci est pratique pour parser, analyser des scripts REBOL.
/markup	Traite le fichier comme un fichier HTML ou XHTML et renvoie un bloc permettant de manipuler les balises et le texte.

En principe, **load** ne renvoie pas l'en-tête du script. Mais, si le raffinement **/header** est utilisé, l'en-tête est le premier item du bloc renvoyé par **load**.

Le raffinement **/next** charge la valeur suivante et renvoie un bloc contenant deux valeurs. La première est la valeur suivante dans la série. La seconde valeur retournée est la position dans la chaîne suivant immédiatement le dernier item chargé.

Le raffinement **/markup** charge des données XML et HTML sous la forme d'un bloc de balises et de chaînes de caractères. Toutes les balises sont du type **tag !**. Les autres données sont traitées comme des chaînes de caractères.

Si le contenu du fichier suivant est chargé avec **load/markup** :

```
This is an example
```

un bloc sera créé :

```
probe data
[ "This is an example" ]
```

5.2 Sauvegarde des scripts

Des données peuvent être sauvegardées dans un fichier de script selon un format facilement récupérable sous REBOL avec la fonction **load**.

C'est une manière pratique de sauver des données et des blocs de données. Par ce moyen, il est possible de créer une mini base de données.

La fonction **save** attend deux arguments : un nom de fichier et, soit le bloc, soit la valeur à sauvegarder.

```
data: [Buy 100 shares at $20.00 per share]

save %data.r data
```

Les données *data* sont écrites au format texte pour REBOL, ce qui permet de les charger ultérieurement avec :

```
data: load %data.r
```

De simples valeurs peuvent aussi être sauvées et chargées. Par exemple, un horodatage peut être sauvé avec :

```
save %date.r now
```

et plus tard rechargé avec :

```
stamp: load %date.r
```

Dans l'exemple précédent, comme *stamp* est une valeur unique, elle n'est pas mise dans un bloc lorsqu'elle est chargée.

Pour sauvegarder un fichier script avec un en-tête, celui-ci doit être fourni en argument, à load avec le raffinement **/header**, soit sous la forme d'un objet ou sinon d'un bloc :

```
header: [Title: "This is an example"]

save/header %data.r data header
```

5.3 Commentaires dans les scripts

L'usage de commentaires est pratique pour clarifier l'objet de certains paragraphes de scripts. L'en-tête d'un script fournit la description générale du script et les commentaires une description plus courte des fonctions. C'est également une bonne idée de mettre également des commentaires pour d'autres parties de votre code.

Un commentaire mono-ligne est précédé d'un point-virgule (**;**). Tout ce qui suit le point-virgule, et jusqu'à la fin de la ligne, est pris comme un commentaire :

```
zertplex: 10   ; set to the highest quality
```

Vous pouvez aussi utiliser des chaînes de caractères pour les commentaires. Par exemple, vous pouvez créer des commentaires sur plusieurs lignes avec une chaîne de caractères placée entre accolades.

```
{
    This is a long multilined comment.
}
```

Cette façon de commenter fonctionne uniquement si la chaîne n'est pas interprétée comme un argument d'une fonction. Si vous voulez être sûr qu'un commentaire multi-lignes soit reconnu comme tel et non interprété comme du code, faites-le précéder du mot **comment** :

```
comment {
    This is a long multilined comment.
}
```

La fonction **comment** indique à REBOL qu'il faut ignorer le bloc ou la chaîne qui le suit. Notez que les commentaires sous forme de blocs ou de chaînes font actuellement partie du bloc global du script. Faites attention de ne pas mettre ces commentaires dans des blocs de données, car ils pourraient être évalués comme des parties de données.

6 Guide de Style

Les scripts REBOL n'ont pas de formalisme particulier. Vous pouvez écrire un script en utilisant l'indentation, la longueur de ligne ou les marqueurs de fin de ligne que vous voulez. Vous pouvez mettre chaque mot sur une ligne séparée ou les joindre ensemble sur une seule grande ligne.

Bien que le formatage de votre script n'affecte pas l'interpréteur, la présentation affecte, elle, la lisibilité. A cause de cela, REBOL Technologies suggère de suivre un certain style pour écrire vos scripts, qui va être décrit dans cette section.

Bien sûr, vous n'êtes pas obligés de suivre l'une ou l'autre de ses suggestions. Pourtant, le style du code est plus important qu'il ne paraît à première vue. La lisibilité et la réutilisation des scripts peuvent en être facilitées. Les utilisateurs peuvent juger la qualité de vos scripts par la clarté de votre style. Un script alambiqué va souvent de pair avec un code alambiqué. Les codeurs expérimentés trouvent d'habitude qu'un style clair et cohérent rend leur code plus facile à produire, à maintenir et à contrôler.

6.1 Formatage

Utilisez les indications de style suivantes pour clarifier la présentation de vos scripts :

6.1.1 Indentez le contenu pour le clarifier

Le contenu d'un bloc est indenté, mais pas les crochets **[ ]** encadrant le bloc. C'est parce que les crochets ont un niveau de priorité plus important pour la syntaxe, et parce qu'ils définissent le bloc mais ne sont pas le contenu du bloc. De plus, il est plus facile de se focaliser sur les ruptures entre des blocs adjacents quand les crochets sont repérables.

Lorsque cela est possible, un crochet ouvrant **[** demeure sur la ligne avec l'expression qui lui est associée. Le crochet fermant **]** peut être suivi de plusieurs expressions au même niveau. Des régles identiques s'appliquent pour les parenthèses **( )** et les accolades .

```
if check [do this and that]

if check [
    do this and do that
    do another thing
    do a few more things
]

either check [do something short][
    do something else]

either check [
    when an expression extends
    past the end of a block...
][
    this helps keep things
    straight
]

while [
    do a longer expression
    to see if it's true
][
    the end of the last block
    and start of the new one
    are at the WHILE level
]

adder: func [
    "This is an example function"
    arg1 "this is the first arg"
    arg2 "this is the second arg"
][
    arg1 + arg2
]
```

Une exception est faite pour les expressions qui appartiennent normalement à une ligne simple, mais se prolongent sur plusieurs lignes :

```
if (this is a long conditional expression that
    breaks over a line and is indented
)[
    so this looks a bit odd
]
```

Ceci s'applique aussi à des valeurs qui sont normalement groupées ensemble, mais qui doivent être adaptées à la ligne.

```
[
```

```
    "Hitachi Precision Focus" $1000 10-Jul-1999
        "Computers Are Us"

    "Nuform Natural Keyboard" $70 20-Jul-1999
        "The Keyboard Store"
]
```

6.1.2 Taille standard pour les tabulations

La taille standard de tabulation pour REBOL est de quatre espaces. Comme les programmeurs utilisent différents éditeurs de texte pour les scripts, il est suggéré d'employer les espaces plutôt que les tabulations.

6.1.3 Détabuler avant de publier

Dans beaucoup de navigateurs, ou shells, ou lecteurs, le caractère de tabulation (ASCII 9) ne correspond pas à quatre espaces, donc utilisez votre éditeur ou REBOL pour ôter les tabulations d'un script avant de le publier sur le Net. La fonction suivante transforme chaque tabulation d'un fichier en un bloc de quatre espaces.

```
detab-file: func [file-name [file!]] [
    write file-name detab read file-name
]
detab-file %script.r
```

La fonction suivante convertit des tabulations à huit espaces en tabulations à quatre espaces :

```
detab-file: func [file-name [file!]] [
    write file-name detab entab/size read file-name 8
]
```

6.1.4 Limitez les longueurs de ligne à 80 caractères

Pour faciliter la lecture et la portabilité entre éditeurs de texte et clients email, limitez les lignes à 80 caractères. Les longues lignes qui sont mal formatées dans les clients email sont difficiles à lire et posent des porblèmes de chargement.

6.2 Libellés des mots

Les mots de votre code sont en premier lieu ce qui est exposé à un utilisateur, de sorte qu'il vous faut les choisir soigneusement. Un script doit être clair et concis. Lorsque c'est possible, les mots doivent être parlants. Voici les convention de nommage pour REBOL.

6.2.5 Utilisez les mots les plus courts ayant le sens le plus fort

Quand c'est possible, les mots directs donnent plus de sens :

```
size  time  send  wait  make  quit
```

Un mot à portée locale peut souvent être réduit à un mot simple. D'un autre côté, des mots plus descriptifs sont préférables pour les mots à portée globale.

6.2.6 Employez des mots entiers si possible

Ce que vous conservez en abrégeant un mot est rarement le meilleur.

Saisissez *date* et non pas *dt*, ou *image-file* et non pas *imgfl*.

6.2.7 Mettez un trait d'union aux mots ayant des libellés complexes

Le style standard est d'utiliser un trait d'union, et non la casse des caractères, pour distinguer des mots.

```
group-name image-file  clear-screen  bake-cake
```

6.2.8 Débutez les noms de fonction par un verbe

Les noms de fonction commencent avec un verbe et sont suivis par un nom, un adverbe, et un adjectif. Certains noms peuvent aussi être utilisés comme verbes.

```
make  print  scan  find  show  hide  take
rake-coals  find-age  clear-screen
```

Eviter autant que possible les mots inutiles. Par exemple, **quit** est aussi clair que **quit-system**.

Quand un nom est utilisé comme verbe, utilisez des caractères spéciaux comme le caractère (**?**) là où cela est possible. Par exemple, la fonction donnant la longueur d'une série est **length ?**.

Voici d'autres fonctions REBOL utilisant cette convention de nommage :

```
size?  dir?  time?  modified?
```

6.2.9 Commencez les variables avec des noms

Les mots représentant des objets ou des variables qui manipulent des données devraient commencer par un nom. Ils peuvent aussi comprendre un adjectif si nécessaire :

```
image  sound  big-file  image-files  start-time
```

6.2.10 Utilisez des mots standards

Il y a des noms standards en REBOL qui devraient être utilisés pour des types d'opérations similaires. Par exemple :

```
make-blub          ; créer quelque chose de nouveau (make)
free-blub          ; libérer les ressources (free)
copy-blub          ; copier un contenu (copy)
to-blub            ; transformer en quelque chose (to-...)
insert-blub        ; insérer quelque chose (insert)
remove-blub        ; enlever (remove)
clear-blub         ; remettre à zéro
```

6.3 En-têtes de script

L'intérêt des en-têtes est clair. Les en-têtes donnent aux utilisateurs un résumé du script et permettent à d'autres scripts de traiter cette information (comme pour cataloguer de script). Un en-tête minimum de script fournit le titre, la date, le nom du fichier, et le sujet du script. D'autres champs peuvent aussi être fournis comme les auteurs, des notes, l'usage et les pré-requis.

```
REBOL [
    Title: "Local Area Defringer"
    Date:  1-Jun-1957
    File:  %defringe.r
    Purpose: {
        Stabilize the wide area ignition transcriber
        using a double ganged defringing algorithm.
    }
]
```

6.4 En-tête de fonctions

Il est commode d'avoir une description dans le bloc de spécification d'une fonction. Limitez ce texte à une ligne de 70 caractères ou moins.

Au sein de cette description, mentionnez quel type de valeur est attendu normalement par la fonction.

```
defringe: func [
    "Return the defringed localization radius."
    area "Topo area to defringe"
    time "Time allotted for operation"
    /cost num "Maximum cost permitted"
    /compound "Compound the calculation"
][
    ...code...
]
```

6.5 Nom des fichiers de script

La meilleure manière de nommer un fichier est de penser à la façon dont vous pourriez le retrouver dans quelques mois. Les noms courts et clairs sont assez souvent les meilleurs. Les noms complexes devraient être évités, à moins d'être signicatifs.

De plus, en nommant le script, pensez à la manière dont le nom ressortira dans le listing d' un répertoire. Par exemple, conservez les fichiers ayant un lien entre eux en faisant débuter leur nom par un mot commun.

```
%net-start.r
%net-stop.r
%net-run.r
```

6.6 Insertion d'exemples

Le cas échéant, fournissez des exemples dans votre script pour présenter son fonctionnement et pour permettre aux utilisateurs de vérifier rapidement s'il fonctionne correctement sur leur système.

6.7 Déboggage incorporé

Il est souvent utile de construire des fonctions de déboggage intégrées au script. C'est particulièrement vrai pour les scripts utilisant le réseau et manipulant des fichiers, et pour lesquels où il n'est pas souhaitable d'émettre et d'écrire des fichiers tant qu'on est en mode de test. De tels tests peuvent être gérés avec une variable de contrôle au début du script.

```
verbose: on
check-data: off
```

6.8 Minimiser le nombre de variables globales

Dans les longs scripts, et autant que possible, évitez d'utiliser des variables globales qui portent leur

état interne d'une partie ou d'une fonction à l'autre.

Pour les petits scripts, ce n'est pas toujours pratique. Mais reconnaissez que ces scripts courts peuvent devenir au fil du temps de plus en plus grands.

Si vous avez un ensemble de variables globales fortement reliées entre elles, pensez à utiliser un objet pour les regrouper :

```
user: make object! [
    name:  "Fred Dref"
    age:   94
    phone: 707-555-1234
    email: dref@fred.dom
]
```

7 Clarification du format d'un script

Voici un court script qui peut être utilisé pour nettoyer l'indentation d'un script. Il fonctionne en analysant la syntaxe REBOL et en reconstruisant chaque ligne du script. Cet exemple peut être trouvé dans la bibliothèque de scripts sur www.REBOL.com.

```
out: none ; output text
spaced: off ; add extra bracket spacing
indent: "" ; holds indentation tabs

emit-line: func [] [append out newline]

emit-space: func [pos] [
    append out either newline = last out [indent] [
        pick [#" " ""] found? any [
            spaced
            not any [find "[(" last out
                     find ")]" first pos]
        ]
    ]
]

emit: func [from to] [
    emit-space from append out copy/part from to
]

clean-script: func [
    "Returns new script text with standard spacing."
    script "Original Script text"
    /spacey "Optional spaces near brackets/parens"
    /local str new
] [
    spaced: found? spacey
```

```
    out: append clear copy script newline
    parse script blk-rule: [
        some [
            str:
            newline (emit-line) |
            #";" [thru newline | to end] new:
                (emit str new) |
            [#"[" | #"("]
                (emit str 1 append indent tab)
                blk-rule |
            [#"]" | #")"]
                (remove indent emit str 1) |
            skip (set [value new]
                load/next str emit str new) :new
        ]
    ]
    remove out ; remove first char
]

script: clean-script read %script.r

write %new-script.r script
```