Note Technique 2065

La commande “do shell script” dans AppleScript

Cette note technique décrit les techniques de base et répond aux questions les plus fréquentes sur la commande *Do Shell Script*, commande apparue avec AppleScript 1.8.

Utilisation de la commande

Q : Ma ligne de commande fonctionne bien dans l’application “Terminal”, mais lorsque j’essaie de l’utiliser dans un script avec la commande *Do Shell Script*, j’obtiens une erreur comme quoi “Commande non trouvée.”

R : Il y a deux possibilités : la première, la commande *Do Shell Script* utilise toujours le shell `sh` et non votre shell par défaut qui lui est utilisé par l’application “Terminal”. Le shell par défaut pour les nouveaux utilisateurs est `tcsh`, aussi tant que vous n’optez pas pour un autre shell, c’est celui-ci qui est utilisé. Tandis que certaines commandes ne sont pas sensibles au shell utilisé, d’autres le sont et vous risquez d’utiliser l’une d’entre elles. Si vous testez votre ligne de commande d’abord dans l’application “Terminal”, il est recommandé d’opter pour le shell `sh`. Pour basculer sur le shell `sh`, il vous suffit de taper dans l’application “Terminal”, avant vos essais, `/bin/sh` ; pour revenir à votre shell normal, vous taperez juste `exit`.

La seconde, lorsque vous utilisez juste le nom de la commande, le shell utilise une liste de répertoires (appelé `PATH`) pour essayer de trouver le chemin complet de la commande. Pour des raisons de sécurité et de portabilité, la commande *Do Shell Script* utilise sa propre liste, laquelle peut ne pas correspondre à celle utilisée par votre shell par défaut. Utilisez le chemin complet de la commande, par exemple, `/sbin/ifconfig` au lieu de juste `ifconfig`. Pour trouver le chemin complet dans l’application “Terminal”, tapez “`which command_name`”, par exemple, “`which ifconfig`” ; pour voir la liste des emplacements visités par la commande *Do Shell Script*, tapez “`do shell script "echo $PATH"`”.

Q : Pourquoi *Do Shell Script* ne fonctionne pas exactement comme l’application “Terminal” ?

R : Pour deux raisons : la première, cela garantit que les scripts tourneront sur différents systèmes sans modification. Si la commande *Do Shell Script* utilisait votre shell ou `PATH` par défaut, votre script serait certainement inutilisable si vous le transmettiez à quelqu’un d’autre. La seconde, elle correspond aux mécanismes d’échappement de shell d’autres langages, comme le langage Perl.

Q : Comment puis-je utiliser ma commande avec un autre shell que `sh` ?

R : Indiquez explicitement dans votre commande le shell que vous souhaitez utiliser. Il existe différentes

manières de transmettre des commandes à un shell bien particulier. Vous pourriez écrire la commande dans un fichier, puis exécuter le fichier comme ceci :

```
do shell script "/bin/tcsh my_command-file-path"
```

Certains shell accepteront un script comme paramètre, comme ceci :

```
do shell script "/bin/tcsh -c 'my_command'"
```

Et la plupart accepteront un script depuis l'entrée standard, comme ceci :

```
do shell script "echo my_command | /bin/tcsh"
```

En cas de doutes, lire la documentation de votre shell préféré. Lorsque vous mettrez la commande dans la chaîne de caractères de *Do Shell Script*, vous devrez probablement la quoter, ou `sh` interprétera les caractères spéciaux de celle-ci.

Q : Comment puis-je utiliser plusieurs commandes dans une seule instruction *Do Shell Script* ? Par exemple, je voudrais me positionner sur un répertoire bien particulier avec la commande `cd`, puis y faire certaines manipulations, mais malheureusement *Do Shell Script* ne se souvient pas du répertoire choisi d'un appel à l'autre.

R : Chaque invocation de la commande *Do Shell Script* utilise à chaque fois un nouveau processus de shell, aussi le répertoire de travail choisi lors d'un précédent appel n'est pas gardé d'une fois sur l'autre. Pour exécuter plusieurs commandes lors d'un unique appel, séparez les commandes avec des points virgules, comme ceci :

```
do shell script "cd ~/Documents ; ls"
- résultat : "Welcome.txt"
```

À cause d'un bug dans le fonctionnement de l'invocation, cela ne fonctionnera pas correctement avec `with administrator privileges` - voir ci-dessous.

Q : Comment obtenir les privilèges administrateur pour une commande ?

R : Utilisez les paramètres `administrator privileges` et `password` comme ceci :

```
do shell script "command" password "mypassword" with administrator privileges
```

Si vous omettez le paramètre `password`, *Do Shell Script* demandera le mot de passe lors de son foctionnement.

Gardez à l'esprit que les privilèges administrateurs permettent de modifier n'importe quel fichier n'importe où dans le système. Vous pouvez rendre votre système non-bootable ou même supprimer de façon définitive le contenu du disque avec quelques commandes bien placées, par conséquent, prenez vos précautions. Même plus, n'utilisez pas les privilèges administrateurs tant que vous n'en avez pas expressément besoin. Si vous

faites du développement système de haut niveau, vous ne devriez pas avoir besoin de modifier quoi que ce soit dans le répertoire `/System` - modifier le répertoire `/Library` devrait suffire.

À cause d'un bug, les privilèges administrateurs ne fonctionneront pas correctement avec plusieurs commandes. Vous devrez écrire votre commande dans une seule invocation de `sh`, comme ceci :

```
set normal_command to "command; command2"
do shell script "sh -c " & quoted form of normal_command with administrator privileges
```

Obtenir une réponse

Q : Comment faire pour obtenir le résultat de la commande *Do Shell Script* ?

R : Les commandes shell peuvent écrire leurs résultats dans un des deux canaux de sortie : la sortie standard et la sortie erreur. La sortie standard est la sortie normale, tandis que la sortie erreur est faite pour les messages et les diagnostics d'erreur. Supposons que votre script aboutit avec succès - s'il n'y arrive pas, voir la question suivante - le résultat est un texte imprimé dans la sortie standard, peut-être avec quelques modifications.

Par défaut, *Do Shell Script* transforme toutes les retours à la ligne du résultat avec des retours-chariots façon Mac ("\r" ou *ASCII character 13*), et supprime la fin des lignes, si elle existe. Cela signifie, par exemple, que le résultat de "`do shell script "echo foo"`" sera simplement `"foo"` et non `"foo\n"` qui est normalement renvoyé par `echo`. Vous pouvez supprimer toutes ces modifications en ajoutant le paramètre `without altering line endings parameter` à la commande *Do Shell Script*. Pour la gestion des données non-ASCII, voir "Traiter le texte".

Q : Comment la commande *Do Shell Script* rapporte-t-elle les erreurs ?

R : Toutes les commandes shell retournent un nombre entier lorsqu'elles ont fini : zéro signifie la réussite ; n'importe quoi d'autre signifie un problème. Si le script sort avec un nombre autre que zéro, *Do Shell Script* retournera une erreur AppleScript mentionnant le chiffre obtenu. La page man d'une commande vous indiquera normalement quel code elle peut retourner. La plupart des commande utilisent généralement 1 pour toutes les erreurs. Si le script a imprimé quelque chose dans le canal de la sortie erreur, ce texte devient le message d'erreur dans AppleScript. S'il n'y a pas de texte d'erreur, la sortie standard (s'il y a) est utilisée comme message d'erreur.

Q : Lorsque je lance ma commande dans l'application "Terminal", j'obtiens un groupe en sortie, mais lorsque j'utilise *Do Shell Script*, certains morceaux sont manquants.

R : Lorsqu'une commande est lancée dans l'application "Terminal", la sortie standard et la sortie erreur sont toutes les deux envoyées à la même place, aussi il est difficile de les séparer. La commande *Do Shell*

Script garde séparés ces deux flux. Si vous voulez les combiner, pistez le résultat de la commande avec `2>&1` comme ceci :

```
do shell script "command 2>&1"
```

Pour plus de détails, voir les pages man de `sh` sur les redirections.

Traiter le texte

Q : Ma commande ne fonctionne pas lorsqu'un paramètre contient des espaces ou certaines ponctuations - parenthèses, $, *, etc...

R : Comme le shell sépare les paramètres avec des espaces, et que certaines ponctuations ont des significations bien précises, vous devez traiter votre chaîne de caractères afin que le shell l'interprète comme un unique paramètre contenant des espaces, parenthèses, etc... Cela s'appelle le "quoting", et il y a plusieurs manières de le faire, mais la plus simple et la plus efficace est d'utiliser la propriété `quoted form` des chaînes de caractères.

Par exemple, considérez ce gestionnaire, lequel prend une chaîne et l'ajoute à un fichier nommé "stuff" situé dans votre répertoire home :

```
to appendMessage(s)
   do shell script "echo " & s & " >> ~/stuff"
end appendMessage
```

Ce script fonctionnera avec la plupart des chaînes de caractères, mais si vous l'appelez avec une chaîne comme `"$100"`, la chaîne qui en sortira sera `"00"` car le shell pense que `$1` est une variable dont la valeur est une chaîne vide (les variables dans `sh` commencent avec un signe dollar). Pour fixer ce problème, modifiez le script comme ceci :

```
do shell script "echo " & quoted form of s & " >> ~/stuff"
```

La propriété `quoted form` présente la chaîne de caractères de façon à ce qu'elle soit bien interprétée par le shell, mais elle ne modifie pas son contenu. Pour plus de détails sur le "quoting", voir les pages man de `sh` sur "Quoting".

Q : J'ai besoin de mettre des guillemets (") et des anti-slashs (\)dans ma commande shell, mais AppleScript m'alerte alors sur une erreur de syntaxe lorsque j'essaie.

R : Les chaînes de caractères dans AppleScript sont délimitées par des guillemets, un au début et un à la fin. Pour qu'AppleScript considère vos guillemets supplémentaires comme des caractères normaux à ne pas interpréter, vous devez les échapper avec un caractère anti-slash, comme ceci :

```
"a \"quote\" mark"
```

Le caractère anti-slash signifie “traiter le caractère suivant de manière spéciale”, donc pour obtenir un caractère anti-slash, il suffira de l'échapper ou de saisir deux caractères anti-slash l'un à la suite de l'autre, comme ceci :

```
"a back\\slash"
```

Ces deux règles de syntaxe AppleScript peuvent être combinées dans votre commande shell afin de pouvoir utiliser les guillemets et les anti-slashs, comme par exemple :

```
set s to "ceci est mon texte."
do shell script "echo " & quoted form of s & " | perl -n -e 'print \"\\U$_\"'"
- résultat : "CECI EST MON TEXTE."
```

La chaîne de caractères réellement transmises à l'option -e de perl est

```
print "\U$_"
```

Q : Chaque fois que mon script shell retourne des guillemets ou des caractères anti-slash dans son résultat, ceux-ci apparaissent avec un caractère anti-slash devant.

R : La fenêtre résultat vous montre le résultat au format source, dans un format respectant la syntaxe AppleScript. Cela signifie que les chaînes de caractères obtenues sont quotées, et que les caractères spéciaux, comme les guillemets et les anti-slashs, sont échappés comme l'a décrit la question précédente. Les anti-slashs supplémentaires ne font pas réellement partie de la chaîne de caractères. Si vous transmettez cette chaîne de caractères à une commande `display dialog` ou que vous l'écrivez dans un fichier, elle s'affichera sans les anti-slashs supplémentaires.

Q : Que fait *Do Shell Script* avec les textes non-ASCII (caractères accentués, japonais, etc...) ?

R : Comme AppleScript version 1.8.3, *Do Shell Script* gère toutes ces entrées et sorties comme de l'UTF-8. Cela fonctionne correctement avec les noms de fichiers et autant que possible avec les commandes elles-mêmes.

Il n'y a pas de support pour les encodages autres que l'UTF-8, et si une commande produit des caractères non-ASCII et non valables avec l'UTF-8 (par exemple, utiliser la commande `cat` avec un fichier texte enregistré avec le format MacRoman), *Do Shell Script* retournera une erreur comme quoi elle n'arrive pas à rendre les données dans le type attendu. Les solutions de rechange sont d'écrire les données en sortie dans un fichier, puis de les lire en utilisant la commande `read` d'AppleScript ou de les rediriger avec le caractère pipe (|) vers `vis`.

N'oubliez pas que la plupart des commandes shell ignorent complètement l'Unicode et l'UTF-8. UTF-8 ressemble à l'ASCII pour les caractères ASCII - par exemple, “A” est l'octet 0x41 à la fois dans ASCII et UTF-8 - mais n'importe quel caractère non-ASCII est représenté par une séquence d'octets. En ce qui concerne les commandes shell, toutefois, un octet égal un caractère, et elles n'essaieront pas d'interpréter

n'importe quoi en dehors des caractères ASCII. Cela signifie qu'elles préserveront les séquences UTF-8 et feront des correspondances octet pour octet : par exemple, `echo "©"` produira le symbole copyright, et `grep "α"` trouvera chaque ligne avec un caractère alpha. Toutefois, elles ne peuvent pas intelligemment trier, modifier ou comparer des séquences UTF-8 : par exemple, les commandes comme `tr` ou `[]` dans `sed` porteront sur chaque octet de la séquence de manière individuelle, `sort` triera les caractères accentués sans ordre précis. Perl est une exception à ce désordre, mais vous aurez probablement besoin d'ajouter `use utf8` à vos scripts perl. Voir les pages man `perlunicode` pour plus de détails.

Q : Quelles sont les règles pour les fins de ligne ?

R : Il existe deux conventions différentes pour les fins de ligne dans Mac OS X : façon Mac (les lignes finissent avec un retour-chariot : "\r" ou le caractère ASCII 13) et façon Unix (les lignes finissent avec des sauts de ligne : "\n" ou le caractère ASCII 10). Les commandes shell généralement gèrent uniquement les fins de ligne façon Unix, aussi leur fournir des textes façon Mac produira des résultats moins qu'utiles. Par exemple, `grep` considérera que le contenu entier de l'entrée n'a qu'une seule ligne, aussi vous n'obtiendrez tout au plus qu'une seule correspondance.

Si vos données viennent d'AppleScript, vous pouvez transformer les fins de ligne dedans ou générer des sauts de ligne tout de suite - "\n" ou caractère ASCII 10, les deux produisant des sauts de ligne. Si vos données viennent d'un fichier, vous pouvez utiliser un shell script pour transformer les fins de ligne en utilisant `tr`. Par exemple, l'instruction suivante trouvera les lignes qui contiennent "something" dans n'importe quel fichier texte au format brut. L'idiome `quoted form of POSIX path of f`" est discuté dans "Traiter les fichiers".

```
set f to choose file
do shell script "tr '\\r' '\\n' < " & quoted form of POSIX path of f & " | grep
something"
```

AppleScript lui-même ignore les fins de ligne - l'élément `paragraph` des objets `string` et `Unicode text` considère les fins de ligne Mac, Unix et Windows équivalentes. Il n'y a généralement pas besoin d'utiliser les `text item delimiters` pour obtenir les lignes d'un texte façon Unix ; `paragraph n` ou `every paragraph` fonctionneront comme il faut. Toutefois, si vous vouliez prendre en compte que les fins de ligne façon Unix, `text item delimiters` sera la solution appropriée. Aussi, avant la version 1.9.1 d'AppleScript, les objets `Unicode text` ont considéré comme fin de paragraphe uniquement les retour-chariots et le caractère Unicode séparateur de paragraphe.

Traiter les fichiers

Q : J'ai un objet AppleScript `file` ou `alias` ; comment le passer à une commande shell ?

R : Le shell spécifie les fichiers en utilisant des chemins POSIX, des chaînes de caractères avec des slashs séparant les différents composants du chemin (par exemple, "`/dossier1/dossier2/fichier`"). Pour obtenir le chemin POSIX d'un objet `file` ou `alias`, utilisez la propriété `POSIX path`. Toutefois, voir la question suivante. Par exemple :

```
POSIX path of file "HD:Users:me:Documents:Welcome.txt"
-- résultat : "/Users/me/Documents/Welcome.txt"
```

Pour obtenir l'effet inverse - les commandes shell retournent un chemin POSIX comme résultat - utilisez l'objet `POSIX file`. `POSIX file` transformera un chemin POSIX en chemin normal que vous pourrez alors transmettre à des commandes AppleScript. Par exemple :

```
set p to do shell script "echo ~"
POSIX file p
-- résultat : file "HD:Users:me:"
```

Q : POSIX path ne fonctionne pas correctement si le nom du fichier contient certains caractères, comme des espaces, des parenthèses, $, *, etc...

R : C'est un cas spécial de "quoting" : vous devez quoter le chemin afin que le shell interprète la ponctuation de façon littéraire. Pour faire cela, utilisez sur le chemin la propriété `quoted form`. Par exemple, le script suivant fonctionnera avec n'importe quel fichier, peu importe les caractères composant le chemin :

```
choose file
do shell script "ls -l " & quoted form of the POSIX path of the result
-- résultat : "-rw-r--r-- 1 me unknown 1 oct 25 17:48 Look! a file!"
```

Q : Pourquoi POSIX path ne quote pas à ma place ?

R : Pour deux raisons : la première, certaines utilisations des chemins POSIX n'en ont pas besoin avec les paramètres shell, et quoter le chemin serait une erreur dans certains cas. La seconde, `quoted form` peut être utile avec autres choses que des chemins. Par conséquent, il y a deux opérations au lieu d'une seule combinant les deux.

Autres inquiétudes

Q : Comment puis-je contrôler un outil interactif comme `ftp` ou `telnet` avec *Do Shell Script* ?

R : La réponse courte est que vous ne pouvez pas. *Do Shell Script* a été conçue pour lancer une commande et puis la laisser tourner sans aucune interaction jusqu'à ce qu'elle ait fini, comme l'opérateur `&` dans la plupart des shells Unix ou l'appel `system` dans `awk` et `perl`.

Toutefois, deux solutions permettent de contourner ce problème. Vous pouvez, tout d'abord, scripter l'application "Terminal" et envoyer une série de commandes à la même fenêtre (cela ne fonctionnera qu'avec la version 10.2 et supérieure de Mac OS X), ou vous pourriez utiliser un package Unix prévu pour scripter les outils interactifs, comme `expect`. De plus, certaines commandes interactives ont un ou plusieurs équivalents non-interactifs. Par exemple, `curl` peut se substituer à `ftp` dans de nombreux cas.

Q : Mon script produira pendant un long moment des sorties. Comment puis-je lire les résultats au fur et à mesure qu'ils sortent ?

R : De nouveau, vous ne pouvez pas - *Do Shell Script* ne retournera rien tant que la commande n'aura pas fini. Ce que vous pouvez faire, toutefois, est de mettre la commande en tâche de fond (voir la question suivante), d'envoyer ses sorties dans un fichier, puis de lire ce fichier alors qu'il se remplit.

Q : Je voudrais lancer un process de serveur en tâche de fond ; comment puis-je faire pour que la commande *Do Shell Script* n'attende pas la fin de la commande ?

R : Utilisez "`do shell script "command > file_path 2>&1 &"`". *Do Shell Script* retournera immédiatement aucun résultat et votre script AppleScript tournera en parallèle avec votre script shell. Les sorties du script shell iront dans `file_path` ; si vous n'avez pas besoin des sorties, utilisez "`/dev/null`". Il n'existe pas de support direct pour obtenir ou manipuler les process en tâche de fond depuis AppleScript. Toutefois, si vous arrivez à récupérer le numéro du process avec `top`, vous pouvez le "tuer" avec la commande `kill numero_process`.

Q : J'essaye d'utiliser la commande `top`, mais j'obtiens un message d'erreur disant que je ne peux pas obtenir les attributs du terminal ou erreur lors de l'ouverture du terminal.

R : `top` dans son mode par défaut fait toutes sortes de choses ingénieuses afin de créer un affichage des process courants, mis à jour de façon dynamique. Mais `top`, par défaut, a besoin de la présence d'une fenêtre de terminal pour fonctionner correctement, ce qui n'est pas le cas avec *Do Shell Script*. Toutefois, cette commande a une option qui permet de la lancer sans la présence obligatoire d'une fenêtre de terminal. Pour obtenir les process courants avec la commande `top`, il vous suffit de spécifier "`top -l1`" et vous obtiendrez la liste de ces process, mais uniquement ceux présents au lancement de `top`, il n'y aura pas de mise à jour dynamique. D'autres options sont disponibles pour `top`, voir les pages man de cette commande.

Ce même problème apparaîtra avec toute commande qui demande la présence d'une fenêtre de terminal. Heureusement, la plupart possèdent des options permettant de se passer de cette fenêtre.

Q : Quel est le répertoire de travail par défaut de la commande *Do Shell Script* ?

R : *Do Shell Script* hérite par défaut du répertoire de travail de son process parent. Pour la plupart des applications, comme l'application "Éditeur de Scripts", il s'agit du répertoire de travail de son process parent, le Finder, lequel est "/". Pour `osascript`, il s'agit du répertoire de travail du shell au moment du lancement de `osascript`. Vous ne devrez pas compter sur le répertoire de travail par défaut, celui-ci pouvant être n'importe quoi. Si vous avez besoin de viser un répertoire de travail bien précis, spécifiez-le vous-même.