



## Une Introduction au Scripting pour les Admins Apple

**Si vous êtes un admin Apple et que le scripting vous fait peur, rassurez-vous : ce guide est là pour vous aider !**

Optimisez la vitesse, la précision et la satisfaction de vos utilisateurs finaux grâce à quelques scripts utiles.

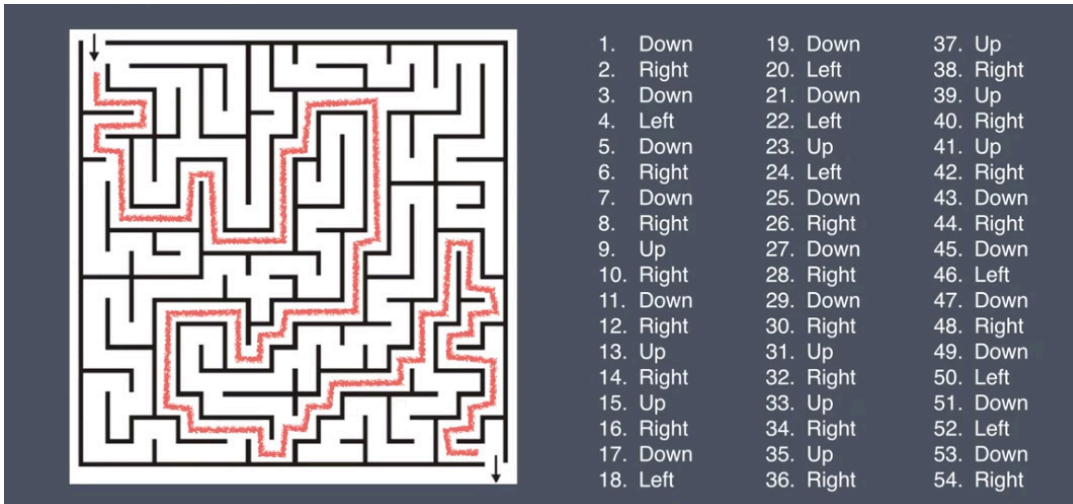
Si vous avez voulu vous essayer au scripting sans parvenir à maîtriser le concept, c'est peut-être simplement parce que la plupart des didacticiels vous proposent des exemples de scripts à mémoriser sans vous donner d'explications claires sur ce que sont les scripts et la manière dont ils peuvent vous être utiles.

**Avant de se lancer, il est important de comprendre comment fonctionnent les scripts et comment vous en servir, plutôt que d'apprendre des lignes de commande sans contexte.**

# Pourquoi utiliser des scripts ?

## Le scripting simplifie les processus.

Le scripting permet de transformer une tâche complexe, par exemple parcourir un labyrinthe, en un ensemble d'instructions simples. Seules quatre commandes sont nécessaires : haut, bas, droite, gauche. Il s'agit de diviser le problème en plusieurs étapes, comme dans cet exemple :



Il est possible d'accomplir des tâches qui paraissent complexes avec quatre commandes seulement.

## Le scripting permet l'automatisation.

Pour ouvrir un fichier sur votre ordinateur, vous n'avez qu'à double-cliquer sur son icône. Pour ouvrir un fichier sur votre ordinateur et sur un autre ordinateur, vous devez double-cliquer à deux reprises. Et pour ouvrir 24 fichiers sur 24 ordinateurs ?

Avec une solution MDM comme Jamf Pro, vous pouvez automatiser cette tâche pour l'exécuter sur tous les ordinateurs avec une simple commande d'une ligne : open (ouvrir), puis l'emplacement du fichier.

## Le scripting réduit les probabilités d'erreur.

Les tâches réalisées manuellement sont plus susceptibles de comporter des erreurs. Il est donc parfois nécessaire de recommencer une tâche, ou bien d'appeler le centre d'assistance. Avec le scripting, les tâches systématiques sont toujours accomplies correctement (et rapidement).

# Comment le terminal fonctionne-t-il ?

## Interpréteurs

Vous ne savez pas de quoi il s'agit ? En réalité, si !

Vous utilisez des interpréteurs depuis des années. Les interpréteurs les plus courants sont l'Explorateur sur Windows et le Finder sur Mac. Leur rôle est d'interpréter les clics sur votre souris et les commandes tapées au clavier pour les transposer en actions sur votre ordinateur. Lorsque vous cliquez sur un fichier, l'Explorateur sur Windows et le Finder sur Mac interprètent ce clic comme votre volonté de sélectionner un fichier. Si vous double-cliquez, ils comprendront que vous souhaitez ouvrir le fichier.

Cependant, tous les interpréteurs ne comprendront pas nécessairement vos actions de la même façon. Par exemple, si vous sélectionnez un fichier dans l'Explorateur sur Windows et appuyez sur la touche ENTRÉE, celui-ci s'ouvrira. En revanche, sur Mac, vous serez invité à modifier le nom du fichier.

Le terminal n'est pas un interpréteur. Il s'agit d'une fenêtre ouverte sur un interpréteur appelé bash. Votre Mac comporte plusieurs interpréteurs, mais dans ce guide, nous allons uniquement parler de bash.

Lorsque vous ouvrez le terminal depuis le dossier des utilitaires des applications, vous accédez à un interpréteur appelé bash. Bash signifie « Bourne-AgainSHell ». Il s'agit d'une version modernisée du shell (sh), interpréteur créé par Steven Bourne il y a 40 ans.



**Le terminal est l'endroit où les utilisateurs Apple exécutent leurs scripts. Comprendre le fonctionnement du terminal vous aidera à mieux visualiser le concept de scripting.**



# Comment les commandes fonctionnent-elles ?

Pour faire simple, les commandes disent à l'ordinateur ce qu'il doit faire. Vous pouvez effectuer toutes actions possibles à partir du Finder. Vous pouvez par exemple ouvrir un fichier en double-cliquant sur son icône. Vous pouvez également ouvrir le terminal et utiliser la commande qui suit, en remplaçant le nom du document par le nom d'un document présent sur votre bureau :

```
open /Users/jamfwebinar/Desktop/scripting101.docx
```

Pourquoi pas une page web plutôt qu'un fichier ? La commande « open » permet aussi d'effectuer cette action. Donnez-lui simplement l'URL et le navigateur avec lequel vous l'ouvrez :

```
open https://www.jamf.com -a Safari
```

Ce guide se veut interactif : nous vous conseillons d'ouvrir le terminal et de suivre les instructions et exemples donnés.

*Applications > Utilitaires > Terminal*

## Commençons par les bases.

Le premier programme que les gens apprennent est « **Hello, World** » dans le terminal.

Cet exercice peut paraître simple, mais il vous permettra d'utiliser la commande echo de plusieurs façons au fur et à mesure que vous progressez dans votre script. Essayez !

Dans le terminal, saisissez :

```
echo 'Hello World'
```

Et appuyez sur la touche Retour.

Comme vous pouvez le constater, la commande echo signifie : **répète ce que j'ai dit**.

Pour étoffer une commande, vous devez y ajouter un argument : le texte compris entre les apostrophes. L'argument explique quoi faire à la commande.

Pour découvrir davantage de scripts simples qui pourront vous servir dans vos processus quotidiens, commencez par vous occuper des tâches basiques réalisées sur ordinateur. Pour vous entraîner, vous pouvez par exemple effectuer des actions simples tout au long de la journée en vous servant du Finder, afin de vous familiariser avec le fonctionnement du scripting.

Vous savez par exemple déjà effectuer les actions suivantes dans le Finder sur Mac :

- Créer un dossier nommé « MyStuff ».
- Créer un fichier avec du contenu.
- Mettre le fichier dans le dossier.

Ces opérations peuvent aussi être réalisées depuis le terminal, avec les commandes suivantes :

```
mkdir = make directory (créer un dossier : cette commande crée un nouveau dossier)
echo = répète ce que j'ai dit
mv = move (déplacer ou renommer, mais nous utiliserons déplacer dans cet exercice)
```

Et vous pouvez utiliser les commandes différemment.

Pour créer un dossier nommé « MyStuff » sur votre bureau, allez dans le terminal et tapez la ligne de commande :

```
mkdir Desktop/MyStuff
```

Et appuyez sur la touche Retour.

Pour créer un fichier avec du contenu, procéder à l'envers : créez le contenu, puis pointez (>) vers l'emplacement où vous souhaitez l'enregistrer en précisant le format souhaité. Utilisez le format d'emplacement de fichier habituel :

```
echo "The quick brown fox jumped over the lazy dogs." > Desktop/Quick-Brown-Fox.txt
```

Et appuyez sur la touche Retour.

Mettez ensuite le fichier dans le dossier. Dans l'outil de ligne de commande, tapez :

```
mv Desktop/Quick-Brown-Fox.txt Desktop/MyStuff
```

Et appuyez sur la touche Retour.

Lorsque vous donnez une instruction par le biais d'une ligne de commande dans le terminal, celui-ci effectue directement l'action associée. Il n'affiche pas de message de confirmation, à moins que vous ne le demandiez explicitement.

Utilisez donc le Finder pour vérifier que le nouveau dossier MyStuff a été créé, et qu'il contient le document Quick-Brown-Fox.txt.

Vous pouvez aussi ouvrir le document depuis le terminal !

Saisissez :

```
open Desktop/MyStuff/Quick-Brown-Fox.txt
```

Et appuyez sur la touche Retour.

Le terminal dispose aussi d'une commande permettant de supprimer des documents et des dossiers : `rm`.

`rm` = remove (supprimer)

`rm -r` = remove recursively (supprimer de manière récursive)

Pour supprimer un document, utilisez simplement `rm`. Essayez la ligne de commande suivante :

```
rm Desktop/MyStuff/Quick-Brown-Fox.txt
```

Ouvrez le dossier depuis le Finder et assurez-vous que le document Quick-Brown-Fox.txt a bien été supprimé.

La même commande permet aussi d'effacer le dossier :

```
rm Desktop/MyStuff
```

Lorsque le terminal ne peut ou ne veut pas accomplir la tâche demandée, il envoie un message d'erreur, ici :

```
rm: Desktop/MyStuff: is a directory
```

Pour confirmer au terminal que vous souhaitez VRAIMENT supprimer le dossier/répertoire et tout son contenu, vous devez utiliser la commande de suppression récursive, ou `rm -r`.

Utilisez cette commande dans le terminal, puis allez vérifier que le dossier MyStuff a bien été supprimé.

```
rm -r Desktop/MyStuff
```

Remarque : pour mettre des espaces dans le nom d'un dossier ou d'un document, encadrez le nom avec des apostrophes. Par exemple, "My Stuff" ou "Quick Brown Fox.txt".

## Écrire des scripts avec des commandes du terminal

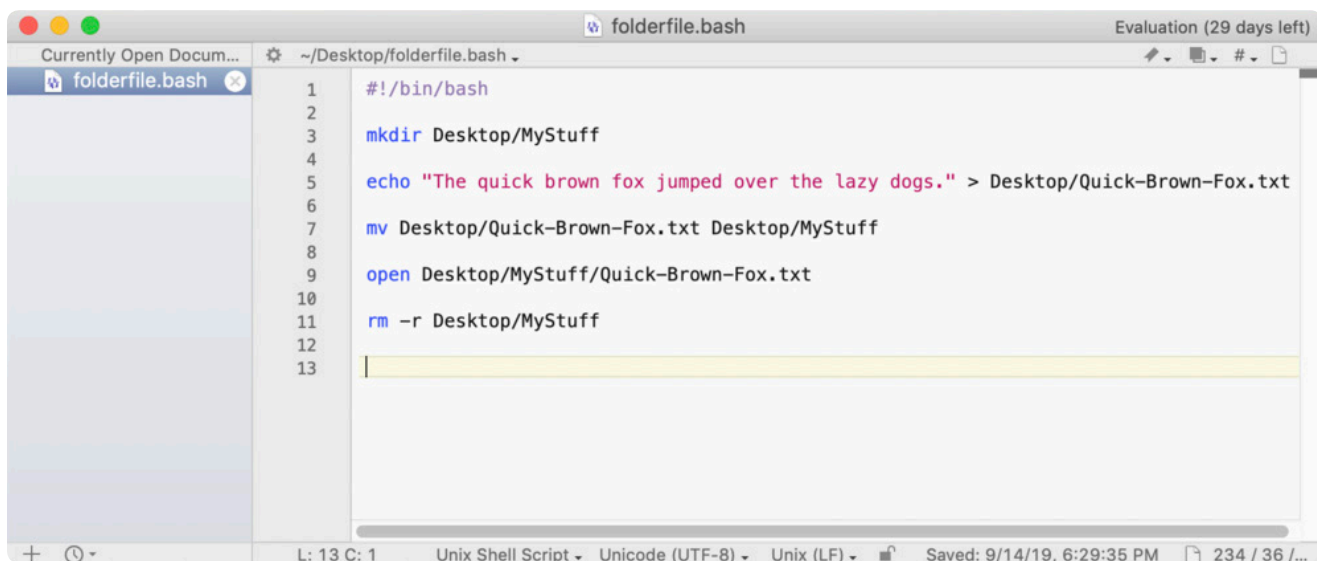
Nous allons maintenant créer un script à partir de ce que nous venons de faire dans le terminal !

Commencez par télécharger et ouvrir un éditeur de script. Nous aimons travailler avec BBEdit, mais vous trouverez des liens vers d'autres programmes à la fin de ce guide. Vous pouvez utiliser un éditeur de texte, mais les éditeurs de scripts gratuits offrent un avantage non négligeable : la mise en forme syntaxique. Les couleurs utilisées changent selon les éléments spécifiques du script pour les rendre plus visibles, et trouver et corriger plus rapidement les erreurs. Cette fonction est particulièrement utile pour les débutants en écriture de script, qui peuvent ainsi mieux se familiariser avec la syntaxe, et pour le dépannage. Ici, [les commandes sont en bleu](#) et [les arguments sont en rose](#).

Ajoutez toujours un élément que l'on appelle un « shebang » (hash + bang #!) en en-tête de votre script. Faites-le suivre de /bin/bash, ainsi, lorsque vous utiliserez vos scripts, le terminal saura qu'il devra utiliser bash en tant qu'interpréteur.

```
#!/bin/bash
```

Copiez-collez chacune des lignes de commande que vous avez utilisées dans l'éditeur de script. Sauter des lignes vous aidera à y voir plus clair.



```
#!/bin/bash
mkdir Desktop/MyStuff
echo "The quick brown fox jumped over the lazy dogs." > Desktop/Quick-Brown-Fox.txt
mv Desktop/Quick-Brown-Fox.txt Desktop/MyStuff
open Desktop/MyStuff/Quick-Brown-Fox.txt
rm -r Desktop/MyStuff
```

Enregistrez le script sur votre bureau sous le nom « **folderfile.bash** ».

Faites maintenant glisser ce fichier dans le terminal, et appuyez sur la touche Entrée.

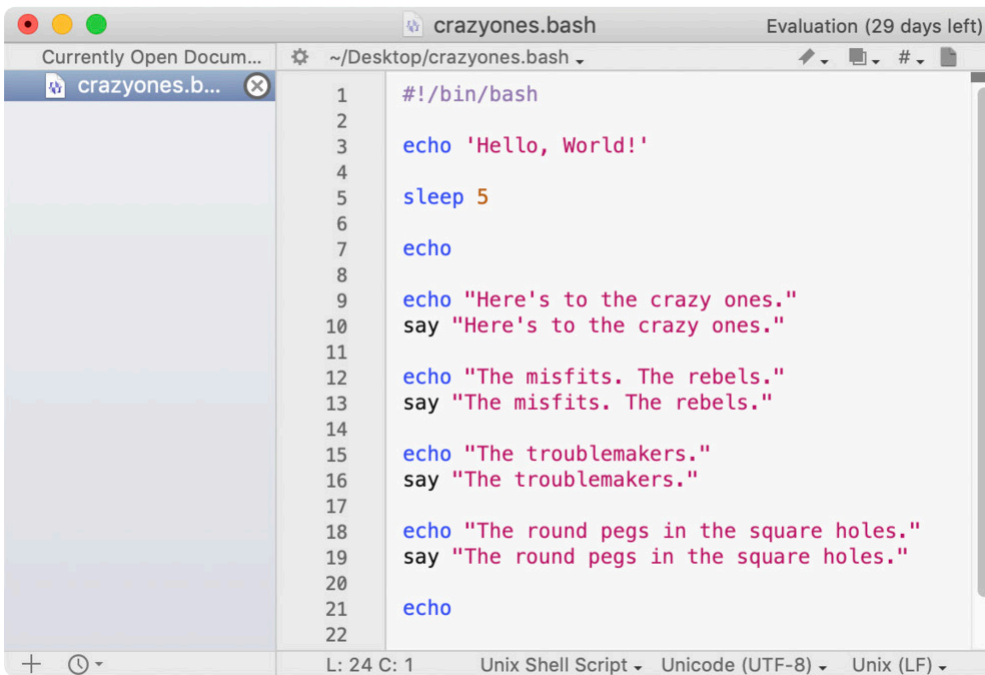
Pas le temps de cligner des yeux, vous pourriez tout louper ! Les scripts ont un autre avantage non négligeable : leur rapidité d'exécution.

Amusez-vous à tester le script suivant dans l'éditeur de script.

Ouvrez votre éditeur de script et saisissez :

```
#!/bin/bash
echo 'Hello, World!'
sleep 5
echo
echo "Here's to the crazy ones."
say "Here's to the crazy ones."
echo "The misfits. The rebels."
say "The misfits. The rebels."
echo "The troublemakers."
say "The troublemakers."
echo "The round pegs in the square holes."
say "The round pegs in the square holes."
echo
exit
```

Vous allez obtenir le résultat suivant :



```
1  #!/bin/bash
2
3  echo 'Hello, World!'
4
5  sleep 5
6
7  echo
8
9  echo "Here's to the crazy ones."
10 say "Here's to the crazy ones."
11
12 echo "The misfits. The rebels."
13 say "The misfits. The rebels."
14
15 echo "The troublemakers."
16 say "The troublemakers."
17
18 echo "The round pegs in the square holes."
19 say "The round pegs in the square holes."
20
21 echo
22
```

Comme vous pouvez le voir, la troisième ligne est une commande echo avec « Hello, World! » echo est en bleu et les variables sont en rose.



La cinquième ligne est `sleep 5`.

`sleep` équivaut à une pause, et « 5 » signifie qu'elle durera 5 secondes.

`say` est une autre commande : vous allez voir de quoi il s'agit dans une minute.

Une commande `echo` vide sert simplement à insérer un espace vide : « `echo [blankspace]`. »

Mises à la suite, toutes ces commandes forment un script.

1. Enregistrez ce script de la même manière que le précédent, en ajoutant `.bash` comme suffixe.
2. Sélectionnez le fichier et faites-le glisser dans le terminal.
3. Assurez-vous d'avoir allumé le son.
4. Appuyez sur la touche Retour pour exécuter le script.

Que venons-nous de faire ? Maintenant, vous savez à quoi sert la commande `say`.

Voici un récapitulatif des commandes que vous avez apprises jusqu'à présent :

`echo` = répéter le texte saisi

`sleep` = pause (à utiliser avec un argument indiquant la durée de la pause en secondes)

`say` = lire à voix haute

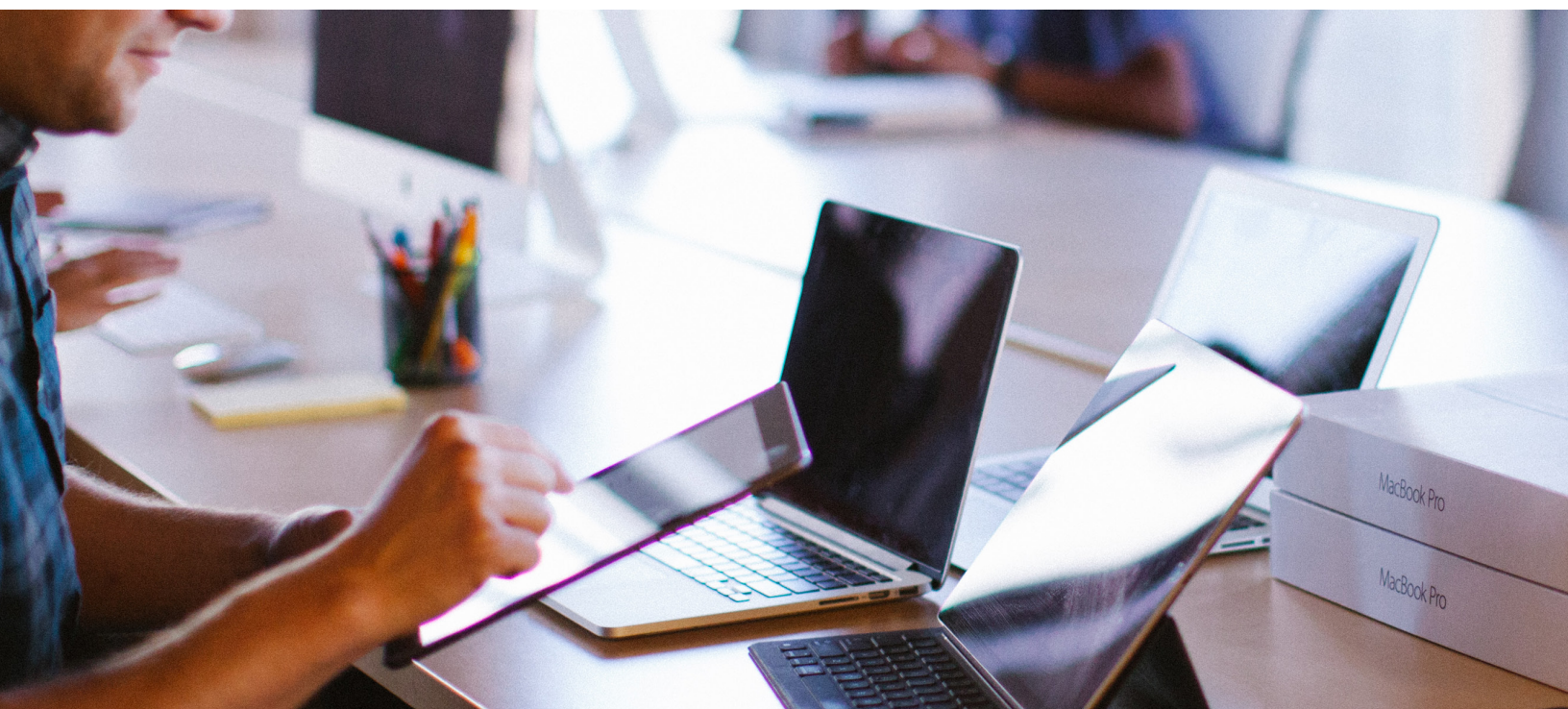
`open` = ouvrir un fichier

`mkdir` = créer un dossier/répertoire

`mv` = déplacer ou renommer un fichier

`rm` = supprimer un fichier

`rm -r` = supprimer de manière récursive (à utiliser pour supprimer un dossier/répertoire)



# Le fonctionnement du script et l'optimisation apportée

## Variables : des emplacements réservés pour les éléments à ajouter

Les variables sont des espaces vierges à compléter, des emplacements réservés pouvant accueillir les éléments désirés. Elles peuvent accueillir toutes sortes de valeurs : nombre, texte, nom de fichier, appareil ou autres données. Une variable n'est pas un élément à proprement dit. Elle REDIRIGE vers les véritables données, qui seront ajoutées ultérieurement.

Les variables sont des outils puissants. Utilisées correctement, elles permettent d'exécuter un script à plusieurs reprises sans avoir à le réécrire, tout en pouvant modifier les données qui y sont associées.

Étudions leur fonctionnement en utilisant la forme de variable la plus simple : un nom. Pour le moment, nous allons travailler en sens inverse, comme lorsque nous avons créé un fichier comportant du contenu.

Tout d'abord, nous allons préciser le type de données utilisé à l'ordinateur. Dans notre situation, nous allons saisir les éléments suivants dans le terminal :

```
myName=Martin
```

Puis nous appuyons sur Entrée. Utilisez toujours des noms de variables clairs, simples et faciles à mémoriser.

Nous allons ensuite dire à l'ordinateur comment il doit utiliser les données. Lorsque vous utiliserez une variable, vous devrez indiquer qu'il s'agit d'une variable avec le symbole \$. Vous allez donc écrire ceci dans le terminal :

```
echo Hello, my name is $myName.
```

Appuyez ensuite sur Entrée.

Vous devriez voir ceci s'afficher dans le terminal :

```
Hello, my name is Martin.
```

Cette méthode consistant à définir une variable puis à l'utiliser dans un script aura de nombreuses implications pratiques au sein de vos processus.

## Commandes sur une ligne

Voici une autre commande pouvant être modifiée par une variable. Dans cet exemple, nous imaginons que vous voulez savoir si le système d'exploitation (OS) d'un Mac doit être mis à niveau.

Nous allons à nouveau travailler en sens inverse.

Nous allons commencer par choisir la variable à utiliser, puis laisser un résultat de commande la trouver pour nous.

Cette opération va nécessiter deux nouvelles commandes (une commande, et une version plus précise de cette même commande) :

```
sw_vers = software version (version du logiciel)
sw_vers -productVersion = uniquement la version de produit du logiciel
```

La commande `sw_vers` nous donne des informations sur la version de macOS : le nom de l'OS, la version de l'OS exécutée et le numéro de version. Si vous voulez uniquement connaître la version, vous pouvez ajouter au script l'élément `-productVersion` après la commande, et vous obtiendrez l'information voulue.

Essayez de saisir cette commande dans le terminal.

Vous devriez obtenir les résultats suivants avec la première version :

```
ProductName:  Mac OS X
ProductVersion:  10.14.4
BuildVersion:  18E226
```

Avec la deuxième version, vous aurez simplement l'information suivante :

```
10.14.4
```

En utilisant des commandes existantes, vous n'aurez pas besoin de définir la variable au préalable. Vous devrez cependant signaler que vous utilisez une variable. Voici comment écrire une variable :

```
$( sw_vers -productVersion )
```

Le symbole `$` est toujours présent, mais c'est la commande de version logicielle qui est utilisée comme variable, et qui est mise entre parenthèses. Comme en algèbre, les éléments mis entre parenthèses sont calculés en premier. Les autres éléments sont pris en compte par la suite.

Ajoutez les éléments suivants à votre ligne de commande :

```
echo My operating system is $( sw_vers -productVersion ).
```

Nous allons ensuite améliorer cette commande avec des éléments appelés « conditionnelles ».

## Question de logique : les commandes if/then, des conditionnelles qui rendent le script intelligent

Les conditionnelles impliquent les éléments « si.. alors » (if/then). Les instructions if/then permettent de prendre des décisions. On leur donne des conditions à respecter, puis on leur demande de déterminer si celles-ci sont applicables ou non.

Par exemple :

SI j'ai des citrons, ALORS je ferai de la citronnade.

Vous pouvez combiner des variables et des conditionnelles pour des résultats plus pertinents.

Écrivez le script suivant dans un éditeur de script :

```
#!/bin/bash
osVersion=$( sw_vers -productVersion )
if [ $osVersion = "10.14.5" ]; then
    echo No upgrade needed.
fi
```

En commençant son script par une variable, un administrateur peut demander quelle version d'OS est installée sur le Mac.

En y ajoutant une conditionnelle, commençant par `if` et suivie d'`echo`, le terminal va retourner la version d'OS installée sur le Mac, et s'il s'agit de la version souhaitée par l'administrateur, il terminera par la mention « No upgrade needed » (Aucune mise à niveau requise).

Les conditionnelles se terminent par `fi`, ou « if » à l'envers. Pratique, n'est-ce pas ?

Essayez par vous-même en enregistrant ce script et en le faisant glisser dans le terminal.

Que se passe-t-il si la condition `if` est fausse ? Ajoutez simplement une instruction `else` et précisez ce qui doit se passer dans ce cas. « Else » signifie « autrement ».

Ajoutez-le à votre script. Vous obtiendrez :

```
#!/bin/bash
osVersion=$( sw_vers -productVersion )
if [ $osVersion = "10.14.5" ]; then
    echo No upgrade needed.
else
    echo Update required.
fi
```

À présent, exécutez ce code dans le terminal.

Est-il possible d'exécuter un script qui détecte non seulement la version d'OS installée sur un Mac, mais installe aussi automatiquement la version la plus récente, le cas échéant ?

C'est tout à fait possible avec les règles de Jamf.

## Configurer des règles

Dans cet exemple, nous allons utiliser l'outil binary de Jamf Helper, qui dit au Mac d'exécuter un événement : la mise à niveau de l'OS.

Jamf Pro installe son propre outil de ligne de commande, binary, sur tous les Mac enrôlés. La commande `jamf` réalise des opérations spécifiques à la gestion Jamf, comme l'exécution de règles. Vous pouvez attribuer un nom unique, ou déclencheur personnalisé (Custom Trigger), à une règle. Celui-ci vous permettra de l'appeler à partir de l'outil de ligne de commande `jamf` pour l'exécuter.

Ici, le script que nous allons utiliser va appeler la règle `runUpgrade`, qui permet d'exécuter une mise à niveau.

Pour effectuer cette opération sur plusieurs ordinateurs à la fois, vous devrez utiliser un service de MDM tel que Jamf Pro.

Pour ajouter ce script à Jamf Pro, allez dans **Réglages > Gestion des ordinateurs > Scripts** et appuyez sur le bouton « Nouveau ».

C'est à cet endroit que vous pourrez lui donner un nom plus précis, par exemple `CheckOSVersionandRunUpgrade`, ajouter des notes, déplacer le script dans l'onglet « Script », attribuer des options et des restrictions, et sauvegarder.

Ajoutez les lignes suivantes à l'onglet « Script » :

```
#!/bin/bash
osVersion=$( sw_vers -productVersion )
if [ $osVersion = "10.14.5" ]; then
    echo No upgrade needed.
else
    jamf policy -event runUpgrade
fi
```

La vue de Jamf Pro permet d'identifier facilement les différents types de scripts, comme les éditeurs de scripts. Sélectionnez « shell » dans le menu déroulant pour afficher par défaut l'éditeur offrant la meilleure lisibilité.

Vous pouvez ajouter des commandes `echo`, qui seront alors répercutées dans vos journaux de règles. Elles pourront vous aider lors de dépannages, et identifier les appareils n'ayant pas été mis à jour.

Pour créer cette règle, créez une nouvelle règle et attribuez-lui le périmètre d'un groupe intelligent. Faites-en un événement récurrent, que vous associerez au code que vous avez créé et nommé dans les étapes précédentes. (Pour en savoir plus sur la création de règles, veuillez consulter le document suivant : [https://docs.jamf.com/10.1.0/jamf-pro/quickstart-computers/Create\\_a\\_Policy\\_to\\_Run\\_Software\\_Update.html](https://docs.jamf.com/10.1.0/jamf-pro/quickstart-computers/Create_a_Policy_to_Run_Software_Update.html)).

## While : la conditionnelle qui attend son heure

L'instruction « while » est aussi une conditionnelle. Au lieu de déterminer la véracité d'une déclaration, elle patiente jusqu'à ce que cette déclaration soit vraie.

```
#!/bin/bash
while [ $( pgrep TextEdit ) ]
do
    echo Waiting for TextEdit to quit.
    sleep 2
done
echo TextEdit has quit.
```

La commande `pgrep` détermine si un processus est en cours d'exécution. Il s'agit ici de l'application TextEdit. Les deux instructions suivantes nous informent sur ce qu'il DEVRAIT se passer lors de l'exécution de TextEdit :

1. indiquer avec « echo » que nous attendons la fin du processus.
2. attendre deux secondes avec la commande « sleep »
3. évaluer à nouveau la condition « while ».
4. lorsque TextEdit sera fermé et que la condition « while » sera fausse, le reste du script pourra être exécuté.

Ouvrez TextEdit, exécutez ce script dans le Terminal, puis quittez TextEdit pour observer le fonctionnement de ce script.

## Boucles for : réaliser des opérations sur des fichiers ou des processus

Ajoutons maintenant une boucle `for`. Nous allons traiter une liste de fichiers dans une boucle `for`, qui va renommer chaque fichier jusqu'à avoir traité l'intégralité de la liste.

Dans cet exemple, plusieurs commandes seront utilisées :

```
cd = change directory (changer de répertoire)
~ est un raccourci vers le dossier principal, que l'on appelle tilde.
ls = list (crée une liste d'éléments dans le répertoire sélectionné)
aFile = nom de variable signifiant « any file » ou n'importe quel fichier. Il peut s'agir de la phrase de votre choix.
« for », dans notre situation, définit une variable et la lit, simultanément. Cette commande dit : Effectue les actions suivantes pour chaque fichier de la liste. Signale avec « echo » que le fichier actuel va être renommé, puis utilise la commande « move » pour réaliser l'action.
```

Dans cet exemple, nous allons ajouter le préfixe « webinar » au nom de tous les fichiers du dossier MyStuff.

Voici la démarche à suivre :

```
#!/bin/bash

cd ~/Desktop/MyStuff

filelist=$( ls )

for aFile in $filelist
do
    echo Renaming file $aFile
    mv $aFile webinar-$aFile
done
```

Tous les fichiers de votre dossier MyStuff doivent maintenant porter le préfixe « webinar ».

## Aller plus loin

Ne vous inquiétez pas. Il est normal que vous n'ayez pas encore retenu toutes les commandes que nous avons évoquées. Cependant, vous allez voir que vous les mémoriserez plus rapidement que vous ne le croyez.

Exemple : inciter les utilisateurs à passer à Mojave

Imaginons qu'un administrateur veuille inciter les utilisateurs à passer à Mojave, puis mettre automatiquement à niveau les Mac sur lesquels Mojave n'a pas été installé au bout d'une certaine période.

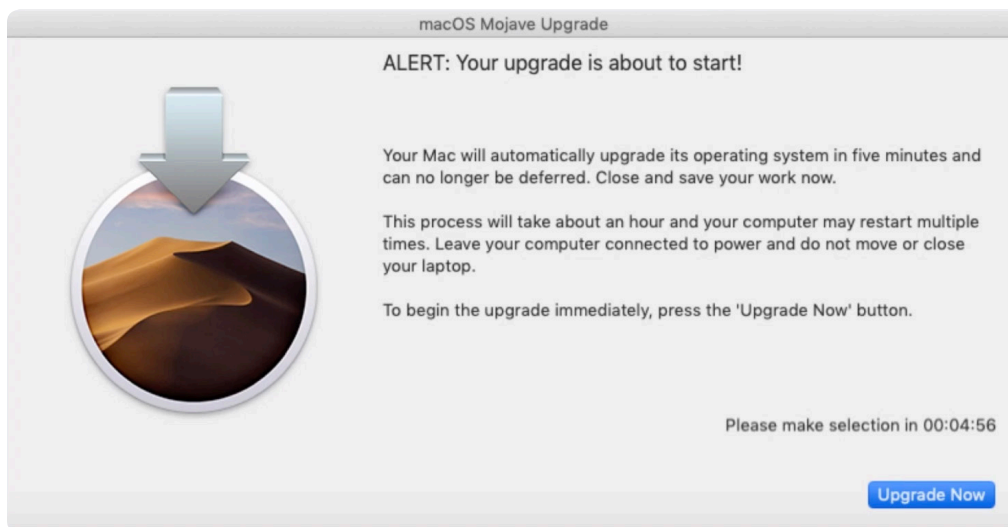
Voici un exemple de ce à quoi peut ressembler le script une fois ce délai passé.

```
#!/bin/bash

"/Library/Application Support/JAMF/bin/jamfHelper.app/Contents/MacOS/jamfHelper" \
-windowType utility \
-lockHUD \
-title "macOS Mojave Upgrade" \
-heading "ALERT: Your OS upgrade is about to start!" \
-description "Your Mac will automatically upgrade its operating system in five
minutes and can no longer be deferred. Close and save your work now. The process
will take about an hour and your computer may restart multiple times. Leave your
computer connected to power and do not move or close your laptop. To begin the
upgrade immediately, select the 'Upgrade Now' button." \
-icon "/Applications/Install macOS Mojave.app/Contents/Resources/InstallAssistant.
icns" \
-iconSize 256
-button1 "Upgrade Now" \
-defaultButton 1 \
-countdown \
-timeout 300 \
-alignCountdown right

exit 0
```

Une fois exécuté, le script aura l'apparence suivante :



Comme vous pouvez le voir, il ne prend pas tout l'écran (-iconSize 256). L'utilisateur peut ainsi désélectionner la fenêtre pour enregistrer son travail en cours. Le compte à rebours apparaît sur le côté droit, et le bouton de sélection affiche le message « Upgrade Now » (Mettre à niveau).

Appuyer sur le bouton ou arriver au bout du compte à rebours lancera le processus suivant :

```
"/Applications/Install macOS Mojave.app/Contents/Resources/startosinstall"  
--agreetolicense &
```

Cet script d'une ligne est intégré à la plupart des programmes d'installation. Il suppose que Mojave est déjà présent dans le dossier et demande un redémarrage de l'ordinateur avec l'esperluette.

Cet aperçu de code plus complexe n'a pas pour but de vous intimider, mais de vous montrer qu'avec quelques connaissances, vous pourrez mettre au point des scripts plus avancés qui vous aideront à automatiser et améliorer vos processus de travail et l'expérience de vos utilisateurs.

Cela vous montre également que vous pouvez parfaitement utiliser des scripts écrits par d'autres personnes. Vous voulez effectuer une tâche par le biais d'un script ? Lancez une recherche Google. Quelqu'un l'a peut-être déjà fait !

Ce qui nous amène à la section consacrée aux ressources.



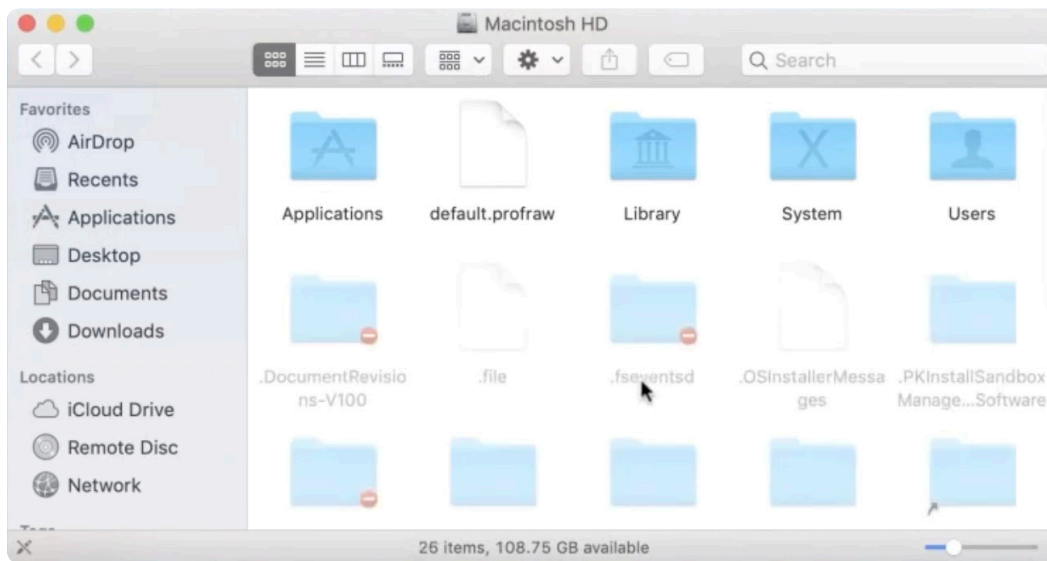
# Étapes suivantes : Ressources

## Votre Mac

Si vous avez envie de découvrir de nouvelles commandes, vous avez déjà le bon outil en main : votre ordinateur sera votre meilleure ressource.

Suivez ces étapes :

Ouvrez une fenêtre du Finder, sélectionnez majuscule ⌘, commande ⌘, point . pour afficher les dossiers et les fichiers cachés. Les commandes du terminal sont toutes enregistrées. Votre ordinateur conserve une trace des instructions relatives à chaque commande dans un fichier séparé.



Commencez par inspecter le dossier bin. « bin » est un diminutif de « binary ». Ouvrez-le pour voir de quoi il s'agit. Ce dossier contient environ 35 éléments : echo, bash, mkdir... certains nous sont déjà familiers, d'autres sont nouveaux.

Il n'est pas nécessaire de connaître toutes les commandes. Pensez plutôt, lorsque vous écrivez des scripts, aux opérations que vous devriez effectuer dans le Finder, puis recherchez des commandes qui correspondent. Google est votre ami ! Souvenez-vous qu'il existe une ligne de commande pour chaque action que vous voulez réaliser dans le Finder.

Les autres dossiers à examiner sont sbin et uxr (qui signifie Unix system resources). Jetez-y un œil !

En tout, vous avez plus de 1 000 commandes à portée de main.

## Vidéos explicatives

Ce guide est une version simplifiée du webinaire « Scripting for Beginners » (webinaire en anglais présentant le scripting pour les débutants). Si vous avez une meilleure mémoire auditive, vous pouvez regarder le cours en vidéo. Vous trouverez également d'autres exemples ici : « [Scripting 101 for Apple Admins](#) » ([vidéo en anglais sur les bases du scripting pour les admin Apple](#))

Après avoir regardé la vidéo, lisez la publication de blog qui la suit : elle contient des suggestions qui vous aideront à poursuivre votre apprentissage et une liste de ressources plus complète : [Just 10 Commands, Then Keep Learning](#) ([blog en anglais sur les 10 commandes pour bien commencer](#))

## Trainingcatalog.jamf.com

Nous proposons 15 heures de courtes vidéos explicatives sur le thème du scripting, des niveaux débutant à avancé. Consultez notre série consacrée au scripting, réservée aux clients abonnés.

## La communauté open source

La communauté open source sur [github.com/jamf](https://github.com/jamf) met à disposition de nombreux scripts open source et gratuits. Profitez de ce qui existe déjà ! Faites des recherches pour voir s'il existe déjà un script correspondant à la tâche que vous souhaitez réaliser. Vous ferez également connaissance avec la fantastique communauté MacAdmin de Jamf Nation, un forum indépendant destiné aux administrateurs Apple et hébergé par Jamf.

<https://www.jamf.com/jamf-nation/>

## Un bon éditeur de script

Nous espérons vous avoir donné envie d'utiliser un éditeur de script, qui vous aidera à mieux visualiser leur syntaxe. En voici quelques-uns. La plupart sont gratuits.

bbedit : [barebones.com](https://barebones.com) (version de base gratuite)

Atom : [atom.io](https://atom.io) (version de base gratuite)

Coderunner : [coderunnerapp.com](https://coderunnerapp.com) 14,99 \$ au moment de la publication de ce document

**Nous espérons que ce guide vous a donné envie d'explorer les capacités des scripts, de les tester, de les utiliser pour améliorer vos processus et d'en apprendre plus sur le sujet.**

Utilisé seul, le scripting peut vous aider dans vos tâches. Cependant, combiné à une solution MDM efficace, il permet l'automatisation de nombreuses opérations et peut vous faire gagner un temps précieux. Nous vous recommandons Jamf Pro.



Demander un essai



[www.jamf.com/fr](https://www.jamf.com/fr)

Ou contactez votre revendeur d'appareils Apple agréé habituel pour tester Jamf Pro