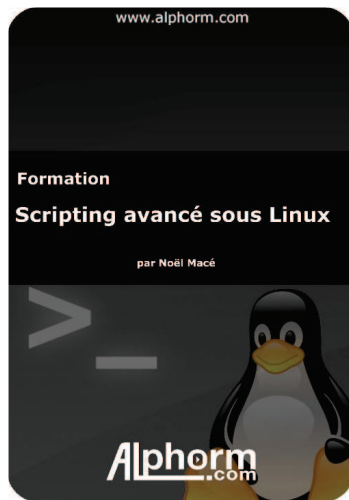




Introduction

Présentation de la formation

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS
Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site [alphorm.com](http://www.alphorm.com)



Plan

- Présentation du formateur
- Qu'est ce que Linux ?
- Qu'est ce que Bash ?
- Pré-requis
- Le plan de formation
- Outils nécessaires
- Comment travailler

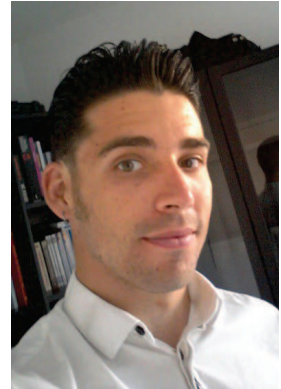




Présentation du formateur

- Noël Macé
- alphorm@noelmace.com
- Formateur consultant expert Unix et FOSS

- Mes références :
 - Mon profil Viadeo : <http://fr.viadeo.com/fr/profile/noel.mace>
 - Mon parcours : <http://vizualize.me/noelmace>
 - Mon site : <http://noelmace.com>
 - Contacts
 - Blogs
 - Base de connaissance
 - CV
 - Etc ...



Qu'est ce que Linux ?

- Un kernel
 - Développé par Linux Torvalds à partir de Minix
 - En 1991
- Un système d'exploitation
 - Libre et open source
 - Issu du projet GNU (1983)
 - Leader sur :
 - les serveurs web (65%)
 - Les systèmes embarqués
 - Les super-calculateurs



Qu'est ce que Bash ?

- Un shell Unix
 - interpréteur de commande
 - interaction (terminal)
 - ou séquentiel (scripting)
 - permet d'exécuter les différentes commandes de GNU/Linux
 - mais aussi d'autre systèmes Unix (Solaris, BSD, Mac OS X, etc ...)
 - permet de créer des programmes simples
 - grâce à des commandes internes de scripting (if, while, etc ...)
 - automatisation
- Cf cours LPIC1



Pré-requis

- bases du scripting Bash
 - cf cours LPIC1
- bonne connaissance de GNU/Linux
 - cf cours LPIC1 et 2
- algorithmique



Le plan de formation

Bases

- premiers pas
- variables
- état de sortie
- tests
- opérations
- boucles
- substitution de commandes

Scripting avancé

- bonnes pratiques
- commandes
- redirection
- documents intégrés
- sous-shells et shells restreints
- fonctions
- alias
- listes et tableaux
- débogage
- options de bash



Outils nécessaires

- il vous suffit
 - d'un interpréteur Bash
 - sous GNU/Linux : shell par défaut
 - sous Windows : [cygwin](#)
 - sous Mac OS X : tcsh est le shell par défaut
 - pour utiliser Bash ; voir [ici](#)
 - d'un éditeur de texte
 - en mode console : vim, nano ou emacs
 - cf LPIC1
 - en mode graphique : gedit, leafpad, ou autre



Comment travailler

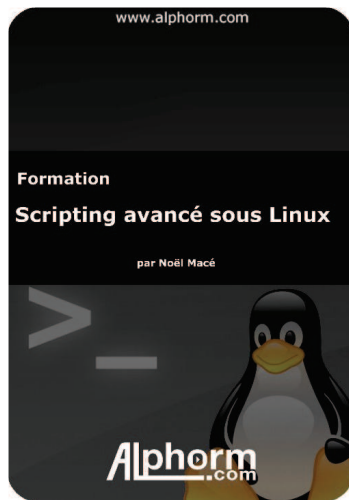
- Chercher un maximum d'exemples
 - init scripts, scripts de configuration des paquets, outils, etc ...
- faire "l'ingénieur fainéant"
 - utiliser le scripting pour toute opération répétitive ou redondante
- Se documenter
 - comme toujours, le man est votre amis
 - ce cours peut être complété par l'excellent guide de tldp.org, porté par Mendel Cooper : [Advanced Bash Scripting Guide](#)
 - traduction en français par traduc.org, disponible [ici](#)



Ce qu'on a couvert

- Introduction de ce cours
 - formateur
 - les outils à utiliser
 - ce que nous verrons





Bases du scripting Bash Premiers pas

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant

Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Executer un script
- Demo





Executer un script

- **directement : nécessite un shebang**
 - première ligne du script
 - `#!` : nombre magique (cf man magic)
 - indique l'interpréteur (`/bin/bash` dans notre cas)

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

- utiliser `/bin/sh` permettra de rendre le script portable aux machines non-linux
 - perte des fonctionnalités spécifiques à Bash
- nécessite le droit d'exécution sur le script



Executer un script

- **via la commande bash**
 - ne nécessite pas les droits d'exécution sur le fichier
 - ni de shebang

```
$ bash /chemin/vers/monscript.sh
```

- **Nous verrons plus loin les différentes options de bash**



Demo

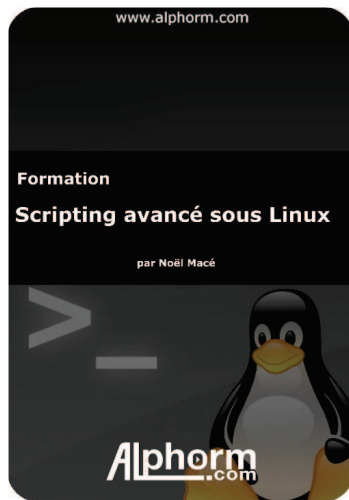
GO !



Ce qu'on a couvert

- **Comment exécuter un script bash**
- **Écrire votre premier script bash.**





Bases du scripting Bash **Variables**

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**

Plan

- Introduction
- Déclarer une variable
- Substitution de variable
- "Supprimer" une variable
- Portée d'une variable
 - Variables d'environnement
- Paramètres positionnels
 - Paramètres positionnels : exemple
- Autres paramètres spéciaux
- Echappements
 - Séquences d'échappement
 - Guillemets et apostrophes
- Extraction et soustraction de chaîne





Introduction

- Aucun typage
 - Toutes les variables sont considérées comme des chaînes de caractères
 - Mais peuvent également être évaluées comme des entiers suivant le contexte
 - Ne nécessite que la présence des chiffres dans sa valeur
 - Permet une plus grande flexibilité
 - Mais aussi certaines erreurs
 - Attention à leur usage
 - Toujours garder trace de leur usage / type
 - Ne pas bâcler votre code



Déclarer une variable

- Sans typage, aucun besoin de déclaration spécifique
- une simple affectation suffit
 - signe égal
 - sans espace
- Exemple

```
$ mavariable="Bonjour"
```

- Attention ! le nom d'une variable est sensible à la casse
 - ie. "mavariable" n'est pas la même variable que "MaVariable"



Substitution de variable

- nécessite le caractère \$ devant le nom
 - Référence à la valeur de la variable
- Exemple :

```
$ echo $mavARIABLE  
Bonjour !
```

- Syntaxe simplifiée de la suivante :

```
$ echo ${mavARIABLE}  
Bonjour !
```



"Supprimer" une variable

- Exemple

```
$ unset MaVariable
```

- La valeur de la variable est alors nulle
 - à ne pas confondre avec une variable contenant une chaîne vide comme nous le verrons avec les tests



Portée d'une variable

- Par défaut : locales
 - n'est définie et disponible qu'à l'intérieur d'un bloc de code ou d'une fonction
- Variables d'environnement
 - Accessible pour l'ensemble du processus et de ses fils
 - Un script ne peut donc exporter une variable vers l'environnement du shell dont il est issu



Variables d'environnement

- utiliser la commande export

```
$ MESSAGE="Bonjour !"
$ export MESSAGE
```

- Par convention, la référence est en majuscules
- Pour "dé-exporter" une variable d'environnement
 - ie : la repasser en local

```
$ export -n MESSAGE
```

- Attention ! l'argument de la commande export est bien la référence (le nom) de la variable
 - pas sa valeur
 - ne pas utiliser le caractère \$



Paramètres positionnels

- variables locales
- \$1, \$2, \$3 ... enregistrent les arguments passés à une commande
 - Doivent être indiquées avec les accolades au delà de 9
- \$0 enregistre la commande ayant permis de lancer le programme
- \$* et @\$ représentent tous les paramètres positionnels
- \$# enregistre le nombre d'arguments
- etc ...



Variables de paramètres: exemple

```
$ /bin/echo -e "Bonjour !"
```

- \$0 est égale à `"/bin/echo"`
- \$1 à `"-e"`
- \$2 à `"Bonjour !"`
- \$* à `"-e 'Bonjour !'"`
- \$# à `2`

Autres paramètres spéciaux

- `$-` : options passées au script via la commande `set`
- `$!` : PID du dernier job exécuté en tâche de fond
- `$?` : code de sortie de la dernière commande exécutée
- `$_` : dernier argument de la dernière commande effectuée
- `$$` : PID du script lui même

Echappements

- Dire au shell d'interpréter le caractère suivant littéralement
 - Grâce au caractère `\`
 - Ex : `cd mon chemin` renvoi une erreur
alors que `cd mon\ chemin` affichera “mon chemin”
- Peut permettre également l'effet inverse
 - Donner un seul particulier à un caractère littéral
 - Par exemple pour `echo` (option `-e`) ou `sed`

Séquences d'échappement

<code>\a</code>	alerte (avertisseur sonore)
<code>\b</code>	retour arrière (backspace)
<code>\c</code>	suppression du retour-chariot final
<code>\f</code>	saut de page
<code>\n</code>	nouvelle ligne
<code>\r</code>	retour-chariot
<code>\t</code>	tabulation horizontale
<code>\v</code>	tabulation verticale
<code>\\</code>	backslash
<code>\nnn</code>	le caractère dont le code ASCII octal vaut nnn (un à trois chiffres)
<code>\xnnn</code>	le caractère dont le code ASCII hexadécimal vaut nnn (un à trois chiffres)

Guillemets et apostrophes

- Protéger (échapper) les caractères spéciaux
 - Empêche leur interprétation pour le shell
 - Permet de les “passer” à une fonction ou au autre programme
- Conseil : placer les substitutions de variables entre guillemets
 - Pour éviter l'interprétation des caractères spéciaux
- Exemples : espaces, *, =, etc ...
- La simple quote est plus stricte
 - Tout caractère y est interprété littéralement
 - Même \$ et \



Extraction et soustraction de chaîne

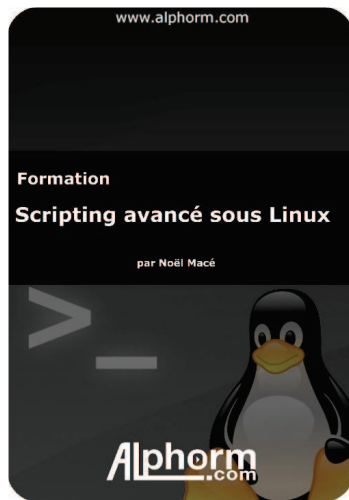
- Il est possible de directement manipuler les chaînes de caractère qui contiennent les variables
 - A noter : le premier caractère d'une chaîne a pour indice 0
- `${#var}` : longueur de la variable (en caractères)
- Extraction
 - `${var:1}` : sous-chaîne à partir du second caractère
 - `${var:2:4}` : sous-chaîne de 4 caractères à partir du 3ème
- Soustraction
 - `${var#expr}` : supprime la plus petite sous-chaîne correspondant à `expr` à partir du début de cette chaîne
 - `${var##expr}` : supprime la plus grande sous-chaîne correspondant à `expr` à partir du début de cette chaîne



Ce qu'on a couvert

- Comment gérer les variables dans un script bash
 - Déclaration
 - Portée
 - Substitution
- Gestion des chaînes de caractère contenues dans les variables
 - Échappement
 - Extraction et soustraction de chaînes





Bases du scripting Bash

États de sortie

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant

Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Etats de sortie
- Tester l'état de sortie d'un programme
- Sortie d'un script





États de sortie

- chaque programme ou fonction renvoi une valeur numérique
 - entre 0 et 255
- 0 si il s'est bien déroulé (sauf cas particuliers)
 - un chiffre différent de 0 indique donc un code d'erreur



Tester l'état de sortie d'un programme

- Nous pouvons évaluer l'état de sortie grâce à plusieurs méthodes de test
 - if/else
 - enchaînements conditionnels
 - case
 - cf cours suivant
- \$? permet également de récupérer la valeur de sortie du dernier programme exécuté



Sortie d'un script

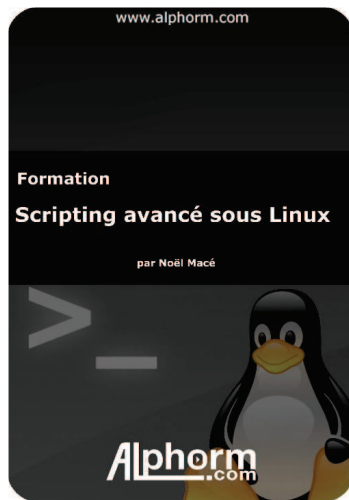
- Commande **exit nnn**
 - où nnn est l'état de sortie à renvoyer
- Sans cette commande, c'est la valeur de sortie de la dernière opération exécutée qui est renvoyée
 - équivalent à **exit** ou encore **exit \$?**



Ce qu'on a couvert

- Valeur de retour des scripts, programmes et fonctions
- Comment sortir d'un script en renvoyant une valeur.





Bases du scripting Bash Tests

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant

Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Test
- Composition
- Alternatives : si - sinon
- Alternatives : Selon





Test

```
test expression
```

- ou

```
[ expression ]
```

- Exemples

```
[ 2 -lt 3 ]
```

```
[ "$var1" != "$var2" ]
```

```
[ -x /bin/bash ]
```



Composition

- séquentielle : `cmd1 ; cmd2`
- parallèle : `cmd1 & cmd2`
- sur erreur (ou) : `cmd1 || cmd2`
- sur succès (et) : `cmd1 && cmd2`



Alternatives : si - sinon

```
if test
then
    instructions
elif test
then
    instructions
    ...
else
    instructions

fi
```



Alternatives : Selon

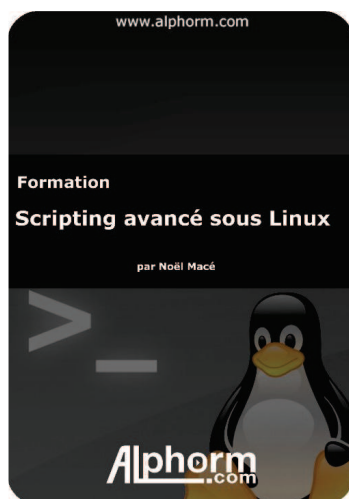
```
case variable in
    expr)
        instructions
        ;;
    ...
esac
```

- Attention : case n'utilise pas d'expression régulière, mais du pattern matching



Ce qu'on a couvert

- Comment réaliser des tests en scripting Bash
 - la commande test
 - composition
 - if ; then ; else
 - case



Alphorm.com

Bases du scripting Bash Opérations

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com

Plan

- Opérateurs arithmétiques
- Affectation arithmétique
- Opérateurs binaires
- Opérateurs logiques
- Changement de base
- Evaluation arithmétique



Opérateurs arithmétiques

Opérateur	Description
+	addition
-	soustraction
/	division
*	multiplication
**	puissance (Bash 2.02 et supérieurs)
%	modulo (reste de division entière)

Remarque : Bash ne comprend pas l'arithmétique à virgule flottante, et interprète donc les nombres décimaux comme des chaînes de caractère. Toute opération sera donc toujours entière.

Pour des calculs complexes, utilisez le langage [bc](#).

Affectation arithmétique

Associe affectation et opérateur arithmétique pour **modifier la valeur d'une variable** par le résultat d'une opération entre sa valeur précédente et une **constante**.

Opérateur	Description
<code>+=</code>	incrémentement
<code>-=</code>	décrémentement
<code>/=</code>	affectation par division
<code>*=</code>	affectation par multiplication
<code>%=</code>	affectation du reste de la division entière

Opérateurs binaires

Opérateur	Description
<code><<</code>	décalage d'un bit à gauche (multiplication par 2)
<code>>></code>	décalage d'un bit à droite (division par 2)
<code>&</code>	ET binaire
<code> </code>	OU (inclusif) binaire
<code>~</code>	NON binaire
<code>^</code>	XOR (ou exclusif) binaire

Tout ces opérateurs peuvent également être associés à l'affectation comme | opérateurs arithmétiques.

Exemple : `<<=`

Opérateurs logiques

- la composition sur erreur (||) et la composition du succès (&&) peuvent également être utilisés comme des OU et ET logiques.
- la commande ! permet d'inverser le retour d'une commande
 - et donc d'agir comme un NON logique
 - exemple :

```
! which /bin/bash
```

Changement de base

- base 10 par défaut
- si un nombre est précédé de 0 : base 8 (octal)
- si un nombre est précédé de 0x : base 16 (hexadécimal)
- Pour utiliser d'autres bases, précéder le nombre de BASE#
 - maximum : 64
 - notation : 10 chiffres + 26 caractères minuscules + 26 caractères majuscules + @ + _

Evaluation arithmétique

- commande `let`

```
let 'var = 2 + 3'
```

- double parenthèse
 - apporte les mêmes fonctionnalités que `let`
 - permet la manipulation de variables à la manière du C
 - Exemple

```
(( var++ ))
```

- permet une “substitution” d'évaluation arithmétique

```
echo $(( 2 ** 3 ))
```

Ce qu'on a couvert

- Comment effectuer des opérations en Bash
 - opérateurs arithmétiques, binaires et logiques
 - changement de base
 - évaluation





Bases du scripting Bash

Boucles

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Pour
- Tant que
- Contrôle de boucles





Pour

```
for var in liste
do
    ...
done
```

- la variable `var` prendra successivement la valeur de chacun des éléments de la liste (séparés par des espaces)
- Exemple :

```
for couleur in rouge orange vert
do
    echo $couleur
done
```



Pour

- Astuce : il est possible d'écrire un `for` à la manière du C grâce à `(( ... ))`
- Exemple :

```
for (( i=1 ; i<=5 ; i++ ))
do
    echo $i
done
```

- Remarque : `(( ... ))` n'est pas ici considéré comme une évaluation arithmétique mais comme un argument particulier de `for`



Tant que

- tant que la commande `cmd` renverra 0

```
while cmd
do
    ...
done
```

- tant que la commande `cmd` ne renverra pas 0

```
while cmd
do
    ...
done
```

- `cmd` est généralement un test



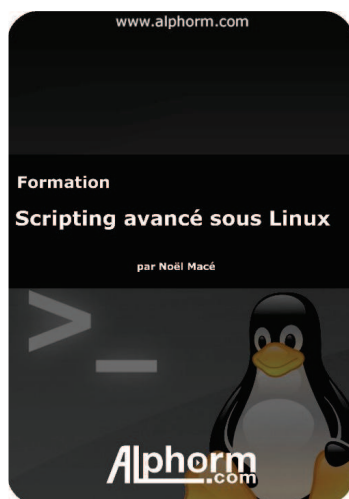
Contrôle de boucles

- `break` : termine (sort de) la boucle
 - `break N` : sortir de N niveaux de boucles (1 par défaut)
- `continue` : provoque un saut jusqu'à la prochaine itération de la boucle
 - `continue N` : provoque un saut jusqu'à la prochaine itération de la boucle de N niveaux au dessus
 - sans effectuer les itérations suivantes des boucles de niveau 1 à N-1



Ce qu'on a couvert

- Comment effectuer des boucles en Bash
 - Pour (for)
 - Tant que (while et until)
- Comment contrôler les itérations de ces boucles
 - grâce à break et continue



Alphorm.com

Bases du scripting Bash

Substitution de commande

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com



Plan

- Qu'est ce que c'est ?
- Syntaxe
- Exemples d'utilisation



Qu'est ce que c'est ?

- la substitution de commande permet de “brancher” la sortie d'une commande dans un autre contexte
- Elle permet :
 - l'affectation de cette sortie à une variable
 - l'utilisation de cette sortie comme argument d'une autre commande
 - etc ...



Syntaxe

- On encadre pour cela la commande d'anti-quotes
 - altgr + 7 sur les claviers Azerty

```
`ma commande`
```

- Nouvelle syntaxe : `$( ... )`
 - à ne pas confondre avec l'évaluation arithmétique
 - permet l'imbrication
 - meilleure lisibilité



Exemples d'utilisation

- Comme argument
 - d'un test

```
[ -x `which bash` ]
```

- de for

```
for i in `seq 1 100` ; do
```

- Pour affectation
 - du contenu d'un fichier

```
variable=`<fichier`
```

```
variable=`cat fichier`
```

- la deuxième solution étant moins performante, puisque nécessitant un nouveau processus



Ce qu'on a couvert

- Ce qu'est la substitution de commande.
- Comment la réaliser.
- Quelques exemples d'utilisation courante.



Alphorm.com

Scripting avancé

Documents
intégrés

Site : <http://www.alphorm.com>

Blog : <http://www.alphorm.com/blog>

Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com



Plan

- Qu'est ce que c'est
- Syntaxe
- Exemples : édition de texte
- Exemples : multi-lignes
- Pour aller plus loin
- Chaînes intégrées



Qu'est ce que c'est

- une solution simple et rapide pour envoyer une suite de commandes à un programme interactif.
- permet donc de réaliser des “scripts” pour beaucoup de programmes
 - dont certains non prévus à cet usage
 - cat, ex, echo, wall, etc ...
- permet également une meilleure lisibilité des entrées multi-lignes



Syntaxe

```
commande <<délimiteur  
entrées ...  
délimiteur
```

- le délimiteur indique la fin des entrées
 - il peut être n'importe quelle suite de caractères
 - par convention, on utilisera généralement EOF



Exemples : édition de texte

- avec vim

```
vim monfichier <<xxxLIMITExxx  
i  
ligne 1  
ligne 2  
^[  
:wq !  
xxxLIMITExxx
```

- avec ex

```
ex $mot <<EOF  
:%s/$ORIGINAL/$REPLACEMENT/g  
:wq  
EOF
```

Exemples : multi-lignes

```
cat <<-EOF
-----
Ceci est la ligne 1 du message.
Ceci est la ligne 2 du message.
Ceci est la ligne 3 du message.
Ceci est la ligne 4 du message.
Ceci est la dernière ligne du message.
-----
EOF
```

- Astuce : vous pouvez supprimer les tabulations en début de ligne d'un document intégré en précédant le délimiteur du signe '-'

Pour aller plus loin

- Les documents intégrés supportent la substitution de variable
- Il est possible de rediriger la sortie de la commande appelée

```
commande > fichier <<délimiteur
```

- On peut réaliser un bloc de commentaire Bash en effectuant un document intégré sur la commande ':'



Chaînes intégrées

- forme minimale de document intégré
 - permet de n'envoyer qu'un chaîne simple
- syntaxe

```
commande <<< chaîne
```

- très utile pour tester votre script

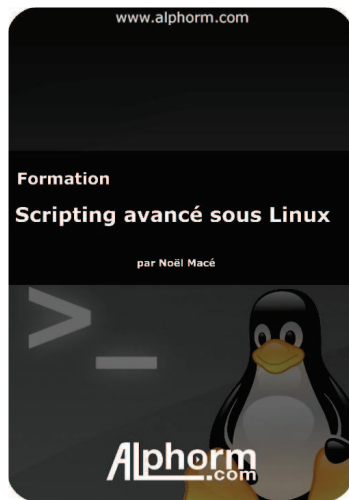
```
read -p "entrez un nombre" nbr <<< "22"
```



Ce qu'on a couvert

- Comment "scripter" n'importe quel programme grâce aux documents intégrés.
 - syntaxe
 - usages
 - fonctions avancées
- Comment envoyer une valeur à n'importe quel programme grâce aux chaînes intégrées.





Scripting avancé

Sous-shells et shells restreints

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant

Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Sous-shell
 - Syntaxe
 - A savoir
- Shell restreint
 - Actions essentielles désactivées





Sous-shell

- permet de lancer un processus fils de notre script
 - qui exécutera son propre "flot" de commandes
 - dont les variables ne sont pas visibles du shell père
- chaque commande externe appelé dans notre shell lance son propre processus
 - ce n'est pas le cas des commandes internes (qui demandent donc moins de ressources)



Syntaxe

- encadrer une série de commandes par des parenthèses

```
(  
cmd1  
cmd2  
cmd3  
)
```



A savoir

- La commande `exit` ne termine que le processus courant
 - si elle est réalisée dans un sous-shell, elle ne mettra donc fin qu'à celui-ci, et non au script `bash` lui-même
- Une commande `cd` effectuée au sein d'un sous-shell n'a d'impacte que sur celui-ci
- Il est possible d'exécuter deux sous-shells en parallèle en les séparant par un `'&'` (sans retour à la ligne)



Shell restreint

- mode spécial
- désactive certaines commandes
 - minimise les risques
 - en limitant les droits
- Pour passer en mode restreint :

```
set -r
```

ou

```
set --restricted
```

Actions essentielles désactivées

- changement de répertoire de travail (cd)
- changement de valeur des variables d'environnement suivantes : \$PATH, \$SHELL, \$BASH_ENV, \$ENV.
- lecture et remplacement d'options d'environnement de shell \$SHELLOPTS.
- redirection de sortie.
- appel à des commandes contenant un ou plusieurs /.
- substituer un processus différent de celui du shell (exec)
- sortir du mode restreint au sein du script

Ce qu'on a couvert

- Comment lancer une série d'opération au sein d'un nouveau processus (sous shell).
 - les principales utilités de cette démarche
- Comment limiter les fonctionnalités d'un script grâce au shell restreint.





Scripting avancé Fonctions

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Fonctions
 - Syntaxe
 - A savoir
- Aliases





Fonctions

- semblables aux fonctions des autres langages
 - bien que très limitées
 - sous-routine implémentant un certain nombre d'opérations dans un but précis
- A utiliser pour remplacer toute suite d'opération répétées plusieurs fois au cours d'un script



Syntaxe

- classique bash

```
function nom_fonction {  
...  
}
```

- "C-like"

```
nom_fonction()  
{  
...  
}
```



A savoir

- Une fonction ne peut être vide
- Elles peuvent prendre des arguments de la même manière qu'un script
 - pas entre les parenthèses !
 - accessibles via \$1 \$2 etc ... au sein de la fonction
- Comme tout programme, elle renvoie également un code de sortie



Aliases

- simple raccourci pour une longue séquence de commandes
 - déclaré grâce à la commande alias
 - exemple

```
alias la="ls -la"
la
```

- pour supprimer un alias : commande unalias
- particulièrement utile en mode interactif
 - cd bashrc
- très peu en scripting, car très limité



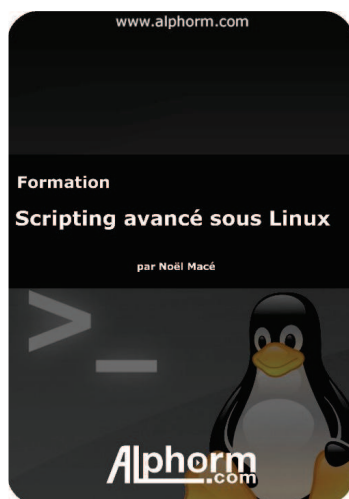
Ce qu'on a couvert

- Comment écrire des fonctions en Bash.
- Comment mettre en place un alias.



Scripting Bash

tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Alphorm.com

Scripting avancé

Listes et tableaux

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash

tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Affectation et substitution
- A savoir



Affectation et substitution

- Affectation
 - de l'ensemble d'un tableau

```
tableau=(val1 val2 val3)
```

- d'un élément d'un tableau

```
tableau[xx]="valeur"
```

- Substitution d'un élément d'un tableau

```
tableau[xx]="valeur"
```

- toutes les opérations de chaîne déjà connues sur les variables sont également réalisables
- exemple

```
${#tableau[@]}      # "longueur" (nombre d'éléments) d'un tableau
```



A savoir

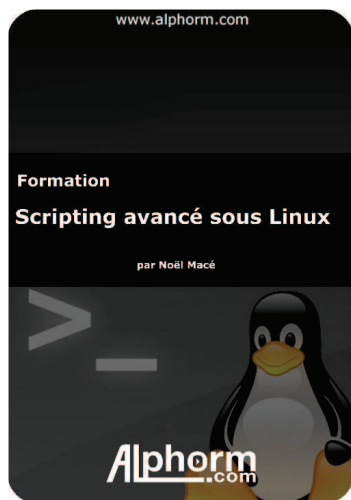
- le premier indice d'un tableau est 0
- `${tableau}` référence le premier élément d'un tableau, non le tableau en entier
- un tableau bash n'a pas de taille fixe. Il s'apparente donc plus à une liste



Ce qu'on a couvert

- Comment déclarer et manipuler des tableaux (ou listes) en scripting bash.





Alphorm.com

Scripting avancé Débogage

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS
Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Introduction
- Suivre l'exécution de votre script
- Quelques recommandations



Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Introduction

- Pas de débogueur intégré
- Des messages d'erreurs souvent obscures
- Nous allons donc voir ici quelques astuces pour déboguer votre script.



Suivre l'exécution de votre script

- Effectuer des affichages (echo) aux points clés
 - Encore mieux : ne le faire qu'en mode debug

```
### debecho (debug-echo) par Stefano Falsetto ###
### Affichera les paramètres seulement si DEBUG est configuré. ###
debecho () {
if [ ! -z "$DEBUG" ]; then
echo "$1" >&2
# ^^ vers stderr
fi
}
```

- Filter les sorties des commandes clés grâce à tee
- Utiliser la variable LINENO
 - prend automatiquement la valeur de la ligne du script où elle est appelée

Quelques recommandations

- Tester chaque variables et conditions aux points critiques
 - On peu pour cela **créer** une fonction assert, inspirée du C
 - Qui sortira du script avec un message d'erreur spécifique en cas de non respect d'un condition donnée
 - Exemple : tester si une variable est un nombre

```
assert "$var1" =~ '^[0-9]+$'
```

- ici, le premier argument de la fonction assert sera bien entendu testé via la commande test pour un if, qui exécutera un exit en cas de non respect de cette condition

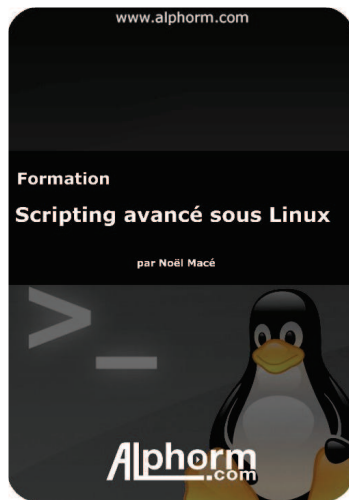
- Intercepter le signal EXIT avec trap afin d'afficher un message personnalisé en cas d'arrêt du script
 - comme par exemple pour afficher les valeurs de différentes variable

```
trap 'echo variables : var1 = $var1 / var2 = $var2' EXIT
```

Ce qu'on a couvert

- Comment déboguer un script bash.





Scripting avancé

Options de Bash

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS

Contact : alphorm@noelmace.com

Scripting Bash tous droits réservés **noelmace.com** autorisation d'exploitation exceptionnellement délivrée pour le site **alphorm.com**



Plan

- Introduction
- Options utiles au débogage
- Autres options utiles
- Pour aller plus loin



Introduction

- Permettent de modifier le comportement du shell
- Activées soit
 - au lancement du script, via l'appel de l'interpréteur
 - shebang ou ligne de commande
 - en cours d'exécution, via la commande set
 - activer une option : set -o nom ou set -abréviation

```
set -o verbose
```

```
set -v
```

- désactiver une option : set +o option ou set +raccourci

Options utiles au débogage

Abréviation	Nom	Description
-n	noexec	Teste les erreurs de syntaxe du script sans l'exécuter.
-v	verbose	Affiche chaque commande avant de l'exécuter.
-x	xtrace	Développe et affiche chaque commande avant de l'exécuter.

Autres options utiles

Abréviation	Nom	Description
-C	noclobber	Empêche tout écrasement de fichier pré-existant avec les opérateurs >, >& et <>. Surpassable par l'usage de >
-a	allexport	Exporter toutes les variables et fonctions modifiées ou créés.
-f	noglob	Désactive le globbing (développement des chemins).
-r	restricted	Mode restreint.
-u	nounset	Empêche l'usage de variable non définie (erreur et sortie du script).

Pour aller plus loin

- Bien d'autres comportements facultatifs sont (dés)activables dans vos scripts

- grâce à la commande shopt

```
shopt [-pqsu] [-o] [nom_opt ...]
```

- Options

- -s var : activer
 - -d var : désactiver

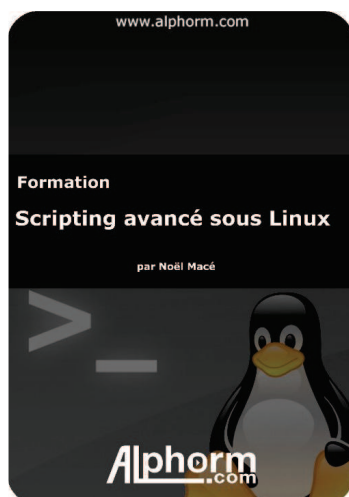
- Exemple : rendre case insensible à la casse

```
shopt -s nocasematch
```



Ce qu'on a couvert

- Comment modifier le comportement de notre script
 - en activant / désactivant certaines fonctionnalités du shell
 - grâce aux options de la commande bash, à set et à shopt



Alphorm.com

Conclusion

Conclusion

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Noël Macé

Formateur et Consultant indépendant
Expert Unix et FOSS
Contact : alphorm@noelmace.com



Ce qu'on a couvert

- Merci à tous et à bientôt.

