Shell et scripts

Licence Pro DevOps

Module Administration Linux

Redirection

- 3 entrées sorties sont associées à un processus lorsqu'il est lancé du terminal:
  - STDIN: entrée standard (clavier par défaut)
  - STDOUT: sortie standard (affichage sur le terminal par défaut)
  - STDERR: sortie erreur (affichage dans le terminal par défaut)
- Les programmes sont censés écrire les erreurs sur la sortie erreur:
  - std::cerr << « Error blablabla » << std::endl;
  - fprintf(stderr, »Error blablabla \n»);
- Possibilité de rediriger ces entrées sorties.

Redirection (2)

- Redirection de l'entrée standard: <
  - read X < file.txt
  - cat < file.txt
- Redirection de la sortie standard: > (trunc) >> (ajout)
  - echo « Toto » > file.txt (le fichier file.txt contiendra Toto)
  - echo « Toto » >> file.txt (Toto sera ajouté à la fin du fichier file.txt)
- Redirection de la sortie erreur: 2> (trunc) 2>> (ajout)
- Redirection de la sortie erreur sur la sortie standard: 2>&1
  - Et inversement 1>&2
- Redirection des sorties erreur et standard: &> &>>
- De manière générale: M>&N
  - Redirection du descripteur M (1 si pas indiqué) sur le descripteur N

Quelques questions

- Le fichier toto existe mais pas titi. Donnez le résultat des commandes suivantes ainsi que les contenus des fichiers.
- ls toto.txt > file.txt
- ls titi.txt 2> file2.txt
- ls titi.txt toto.txt > file.txt
- ls titi.txt toto.txt 2> file.txt
- ls titi.txt toto.txt >> file.txt
- grep devops toto.txt > file.txt
- grep devops toto.txt 2> file.txt

Redirection (3)

- Pour éviter un affichage on peut rediriger sur le fichier « null »: /dev/null
  - echo « Toto » > /dev/null
- Redirection plus poussé
  - Commande exec:
    - execute une commande au sein du shell (sans fork mais avec l'exec)
    - si aucune commande n'est indiquée, la redirection s'applique au shell
    - exec > file.txt (la sortie standard du shell est redirigé vers file.txt)
  - n<> file : ouvre (r/w) un fichier « file » et lui associe le numéro n

Quelques questions

- Donnez le résultat des commandes suivantes. Le fichier toto existe mais pas titi.

```
___$ exec ls

___$ exec 3<> file.txt
___$ cat <&3
___$ echo « Toto » >&3

___$ exec > file.txt
___$ ls
___$ cat toto.txt
___$ ls titi.txt
___$ cat file.txt 1>&2
```

Redirection: le pipe |

- Redirection de la sortie standard sur l'entrée standard
  - Exemple: cat file.txt | grep toto
- Une redirection de la sortie erreur peut être faite au préalable
  - ls eeee 2>&1 | grep eeee

Caractère spéciaux

- & background
- && (ET)
  - commande1 && commande2
  - Exécute commande 2 que si commande1 a réussit
- || (OU)
  - commande1 || commande2
  - Exécute commande 2 que si commande1 a échoué
- | : pipe

Caractères spéciaux (2)

- #: commentaires
- \: banalise un caractère spécial
- Espace et tabluation: séparateur
  - Donc à gérer dans les noms de fichiers (mieux vaut éviter).
- $$: pid du processus
- $1, $2, etc.: arguments d'un script (celles du bash sont vides)
- $#: nombre d'arguments (vaut 0 dans le bash)
- $?: résultat de la dernière commande
- $(): exécute la commande entre parenthèse et la substitue
  - Exemple sans intérêt: echo $(echo toto)

Quelques questions

- Donnez le résultat des commandes suivantes
  - Le fichier toto.txt existe mais pas titi.txt
- ls toto.txt && date
- ls titi.txt && date
- ls toto.txt || date
- ls titi.txt ||date

RegEx dans le shell

- Expression régulière interprétée par le shell
- Remplacé par des noms de fichiers:
  - *: caractère ou suite de caractères quelconques
  - [1-9]: remplace un nombre
  - [a-d]: remplace une lettre de a à d
- Exemples:
  - ls *.txt : tous les fichiers finissant par .text
  - cat *[a-d]*: tous les fichiers contenant a, b, c, ou d.

Quelques questions

- Le contenu du dossier courant est le suivant

fichier devops.txt
ascii.txt
file0.txt
file2.txt
retour.txt
data0
data1

- Donnez le résultat des commandes suivantes:
- ls *
- ls *a*
- ls *[0-9]
- ls fichier devops.txt
- ls *[a-i]*

Les « ‘ `

- ‘ ‘ banalise tous les caractères à l’intérieur.
- « « banalise les caractères sauf $ et `
- ` ` ne banalise rien – exécute la commande à l’intérieur

Quelques questions

- Donnez le résultat des commandes suivantes

____$toto=titi
____$echo « echo $toto »
____$echo ‘echo $toto’
____$echo `echo $toto`

Interprétation d'une ligne de commande

- De gauche à droite
  - Avant le lancement des commandes tous les caractères spéciaux sont remplacés par leurs valeurs:
    - Variables ($...)
    - Expression régulière du shell -> avec les noms de fichiers
  - Exécution des commandes prioritaires (``, $( ))
  - Fork et redirection (dépend des |, &&, etc.)
  - Lancement des commandes (exec)
  - Note: sudo s'applique qu'à la première commande

Regex des commandes

- Pour les commandes les expressions régulières ont un sens différent
  - Test et grep principalement
- Elles doivent être banalisées pour le shell
  - « « , ‘’
- grep ou ~= (test)
  - *: n'importe quel caractère unique
  - .: répétition du caractère précédent
  - [0-9]: caractère entre 0 et 9
  - ^: début de ligne
  - $: fin de ligne
  - |: ou

Variables d'environnement du shell

- Variables d'environnement
  - Ensemble de variables décrivant l'environnement
    - Pour les applications par exemple:
      - Dossiers contenant les fichiers de configurations, la base de donnée, etc.
  - Propre à chaque processus
  - Hérité au moment du fork (création du processus)
- Export
  - Passe une variable du shell en variable d'environnement

Structure de contrôle

- If / while / for
- Fonction

```
if test
then
   commande1
   commande2
fi
```

```
if test
then
   commande1
else
   commande2
fi
```

```
if test
then
   commande1
elif
then
   commande2
else
   commande3
fi
```

```
while test
do
   commande1
   commande2
done
```

```
do
   commande1
   commande2
while test
```

```
for var in ….
do
   commande1
   commande2
done
```

Les tests

- Les structures de contrôles if, while, etc. évalue le résultat d'une commande
- La commande test.
  - test, [ ], [[ ]] permettent d'effectuer des tests
    - [[ ]] est le plus récent et inclut les fonctionnalités des précédents
    - Test et [ ] sont équivalents
  - Retourne 0 quand vrai et 1 sinon (l'inverse d'un booléen habituel)
  - =, != comparent des chaînes des caractères
  - -z chaine (resp. –n) test si la chain est vide (resp. non vide)
  - -eq, -gt, -lt, -neq, etc. comparent des entiers
  - -a, -o: et et ou
    - [toto = toto –a 4 –leq 2]
  - ! est la négation (attention aux espaces): [! 4 –leq 2]

Les test (suite)

- Test sur les fichiers:
  - -f: fichier existe (fichier ordinaire)
  - -e: existe (répertoire, fichier ou autre)
  - -d: répertoire
  - -r, -w, -x: existe avec les droits en lecture, écriture, ou exécution
  - fichier1 –nt fichier2: fichier1 est « newer than » fichier2 (ot marche aussi)
- Beaucoup d'autres existent!

Quelques questions

- Donnez le résultat des commandes suivantes

Attention aux espace!

_____$ test [toto = titi]
_____$ echo $?
_____$ test [toto = toto]
_____$ echo $?
_____$ [[4 –gt 2]]
_____$echo $?

Les fonctions

- Doit être déclaré avant son appel
- Appelé comme une commande au sein du script.
- Gère les arguments comme pour le script ($1, $2, etc.).
  - Ceux du scripts sont donc écrasés
- return renvoi le code de retour (pas obligatoire).
  - Même principe qu'une commande.
- Les variables sont globales par défaut
  - Mais possibilité de déclarer des variables locales à la fonction
  - Ecrase une variable globale dans ce cas (dans la fonction)
- Quand une commande est tapé, le shell regarde d'abord si il s'agit d'une fonction
  - Permet d'écraser les commandes connues (au sein du script)
- N'est pas exécuté dans un sous shell (sauf les commandes bien sûr).

```
function functionName
{
   local myVariable
   commande1
   commande2
   return 0
}

functionName toto
```

```
functionName ()
{
   local myVariable
   commande1
   commande2
   return 0
}

functionName toto
```

Quelques questions

- Donnez la sortie du script ci-dessous:

```
ls

function ls
{
    local var=toto
    echo $var
    return 2
}

var=titi
echo $var
ls
echo $?
echo $var
```

Les calculs

- Pas fait pour (à éviter).
- Toutes les variables du shell sont traitées comme des chaînes de caractères
- Commandes permettant d'effectuer des calculs:
  - expr
  - bc

Commandes usuelles

- Fichiers
  - cat, touch, tail, head
  - ls, mkdir, rm, rm –r, cp, mv
- Filtres
  - cut, grep, etc.
- Trouver un fichier:
  - locate, find
- Commandes systèmes:
  - Réseau: ip, iptable, arp
  - Système: fdisk, mount