Plan du cours

- Fonctions d’un système d’exploitation
- Partage des ressources et virtualisation
- IHM et ligne de commande
- **Langages de commande**

Pourquoi utiliser des scripts shell : avantages ?

- Dans de nombreux contextes, on ne dispose pas d'interface graphique
- Travail en ligne de commande souvent plus efficace qu'à travers une interface graphique
- Automatisation de tâches répétitives

- Meilleure compréhension du système UNIX (fichiers de configuration...)

Pourquoi ne pas utiliser des scripts shell : inconvénients ?

- Documentation difficile d'accès pour le débutant
- Syntaxe cohérente mais parfois obscure (concision vs. clarté)
- Messages d'erreurs parfois difficilement exploitables
- Relative lenteur

Les différents shells

- Shell de Stephen R. Bourne :
  - Bourne shell : **sh**
  - Bourne-Again shell : **bash**
- Shell de David Korn :
  - Korn shell : **ksh**
- C shell : **csh**
- Tenex C shell (version moderne du csh) : **tcsh**
- Shell de Kenneth Almquist prenant peu de place sur le disque :
  - Almquist shell : **ash**
  - Debian Almquist shell : **dash**
- Z Shell (**zsh**), intégrant les fcts les plus pratiques de bash, ksh et tcsh

Ecriture de scripts bash

- Un script bash est un simple fichier texte exécutable (droit x) dont la première ligne doit obligatoirement être **#!/bin/bash**

- Dans un éditeur de texte, écrivons le script suivant :

```
#!/bin/bash
#
# Shell-script affichant "bonjour" sur la sortie standard
#
message="bonjour"
echo $message
```

- Enregistrons ce script sous le nom **bonjour.sh**

Exécution de scripts bash

- Dans un terminal, en ligne de commande, rendons le script exécutable :

```
chmod u+x bonjour.sh
```

- Exécutons le script (plusieurs solutions) :

```
bonjour.sh
```

ou

```
./bonjour.sh
```

ou

```
bash bonjour.sh
```

ou

```
exec bonjour.sh
```

Paramètres de scripts bash

- Il est possible d'exécuter un script en lui passant un certain nombre de paramètres (ou arguments), comme pour n'importe quelle autre commande :

```
mon_script.sh arg1 arg2 ... argN
```

- En bash, les arguments de la ligne de commande sont stockées dans des variables spéciales :

Variable	Description
$#	Nombre d'arguments
$*	Liste des arguments
$0	Nom de la commande
$1	Valeur du premier paramètre
$i	Valeur du ième paramètre si i compris entre 1 et 9
$9	Valeur du neuvième paramètre

Scripts shell en bash : les concepts de base

- **Caractères spéciaux**
- **Variables**
  - variables d'environnement
  - variables de l'utilisateur
- **Opérateurs**
- **Structures de contrôle**
  - exécution conditionnelle
  - choix multiple
  - boucle for
  - boucles while et until
- **Expressions régulières**

Caractères spéciaux (ou métacaractères)

Caractère	Description
*	Métacaractère qui remplace n'importe quelle chaîne de caractères (même vide)
?	Métacaractère qui remplace un caractère quelconque
;	Permet de séparer plusieurs commandes écrites sur une même ligne
( )	Regroupe un ensemble de commandes et les exécute dans le "shell courant"
{ }	Regroupe un ensemble de commandes et les exécute dans un "shell fils"
[ ]	un caractère quelconque ∈ à la liste donnée entre crochets, définie par énumération ou par un intervalle
[!]	un caractère quelconque ∉ à la liste donnée entre crochets, définie par énumération ou par un intervalle
&	Permet le lancement d'un processus en arrière plan
|	Permet la communication par tube entre deux commandes
#	Introduit un commentaire. Tout ce qui suit dans une ligne est ignoré par le shell
\	Déspécialise le caractère qui suit
'...'	Définit une chaîne de caractères qui ne sera pas évaluée par le shell
"..."	Définit une chaîne de caractères dont les variables seront évaluées par le shell
`...`	Définit une chaîne de caractères qui sera interprétée comme une commande et remplacée par la chaîne qui serait renvoyée à l'exécution de la dite commande

Variables d'environnement (1)

Variable	*Description*
PWD	Stocke le chemin et le nom du répertoire courant
HOSTNAME	Nom du serveur
HOSTSIZE	Taille de l'historique des dernières commandes passées au shell
LANGUAGE	Suffixe de la langue du système
PS1	Chaîne apparaissant à l'invite du Shell
USER	Nom de l'utilisateur
DISPLAY	Adresse du terminal d'affichage
SHELL	Chemin et nom du programme Shell
HOME	Chemin du répertoire de connexion
PATH	Liste des répertoires où chercher les exécutables des commandes externes
TERM	Type de terminal ASCII

Variables d'environnement (2)

- Les variables d'environnement sont manipulées via les commandes :
  - **`printenv`** : affiche la liste des variables d'environnement
  - **`export VARIABLE=VALEUR`** : donne une valeur à une variable
  - **`echo $VARIABLE`** : affiche la valeur de la variable
- Exemples :

```
printenv
PWD=/home/Olivier
LANG=fr_FR.UTF-8
SHELL=/bin/bash

printenv LANG
fr_FR.UTF-8
```

Variables de l'utilisateur

- L'utilisateur peut déclarer facilement de nouvelles variables par l'affectation directe d'une valeur (numérique, chaîne de caractères) :

```
ma_variable=valeur
```

- Exemples :

```
EMAIL=audrey.queudet@univ-nantes.fr
moi=audrey
vous=stagiaires
phrase1="Bonjour $vous, moi c\'est $moi"
phrase2='Bonjour $vous, moi c\'est $moi'
echo $phrase1
Bonjour stagiaires, moi c'est audrey
echo $phrase2
Bonjour $vous, moi c\'est $moi

rep=`pwd`
echo $rep
/home/queudet/data
```

Opérateurs sur les fichiers

Opérateur	Description
`-e filename`	Vrai si *filename* existe
`-s filename`	Vrai si *filename* est vide
`-d filename`	Vrai si *filename* est un répertoire
`-f filename`	Vrai si *filename* est un fichier ordinaire
`-L filename`	Vrai si *filename* est un lien symbolique
`-r filename`	Vrai si *filename* est lisible (r)
`-w filename`	Vrai si *filename* est modifiable (w)
`-x filename`	Vrai si *filename* est exécutable (x)
`file1 -nt file2`	Vrai si file1 plus récent que file2
`file1 -ot file2`	Vrai si file1 plus ancien que file2

Opérateurs sur les chaînes

Opérateur	Description
`-z chaîne`	Vrai si la *chaîne* est vide
`-n chaîne`	Vrai si la *chaîne* est non vide
`chaîne1 = chaîne2`	Vrai si les deux *chaînes* sont égales
`chaîne1 != chaîne2`	Vrai si les deux *chaînes* sont différentes

Opérateurs arithmétiques

Opérateur	Description
+	addition
-	soustraction
*	multiplication
/	division
**	puissance
%	modulo

- Expressions arithmétiques :

```
..=$(( ... ))

n=1
m=$((5*n+1))
echo $m
```

ou

```
let "..=..."

n=1
let "m=5*$n+1"
echo $m
```

Opérateurs de comparaison numérique

Opérateur	*Description*
`num1 -eq num2`	égalité
`num1 -ne num2`	inégalité
`num1 -lt num2`	inférieur (<)
`num1 -le num2`	inférieur ou égal (≤)
`num1 -gt num2`	supérieur (>)
`num1 -ge num2`	supérieur ou égal (≥)

Opérateurs booléens

Opérateur	*Description*
-a	**ET logique**
-o	**OU logique**
!	**NON logique**

Structures de contrôle : exécution conditionnelle

- L'instruction **if** permet d'exécuter des instructions si une condition est vraie

- Le bloc **if/then**

```
if [ condition ]
then
    actions
fi
```

- Le bloc **if/then/else**

```
if [ condition ]
then
    action1
else
    action2
fi
```

- Enchaînement de plusieurs conditions

```
if [ condition1 ]
then
    action1
elif [ condition2 ]
then
    action2
elif [ condition 3 ]
then
    action3
else
    action4
fi
```

Structures de contrôle : choix multiple

- L'instruction **case** permet de choisir une suite d'instructions suivant la valeur d'une expression

```
case "$x" in
    case1)
            actions1
            ;;
    case2)
            actions2
            ;;
    ...
    caseN)
            actionsN
            ;;
esac
```

Structures de contrôle : boucle for

- L'instruction **for** permet une exécution répétitive d'une suite d'instructions

- Schéma classique

```
for VAR in LISTE
do
    actions
done
```

- Schéma alternatif

```
for ((initialisation de VAR; contrôle de VAR; modification de VAR))
do
    actions
done
```

Structures de contrôle : boucles while et until

- L'instruction **while** permet une exécution répétitive d'une suite d'instructions tant qu'une condition est vraie

```
while [ condition ]
do
    actions
done
```

Condition de continuation de la boucle

- L'instruction **until** permet une exécution répétitive d'une suite d'instructions jusqu'à ce qu'une condition soit vraie

```
until [ condition ]
do
    actions
done
```

Condition d'arrêt de la boucle