# Dépannage et maintenance système

Les scripts et batchs

BASH

THE BOURNE-AGAIN SHELL

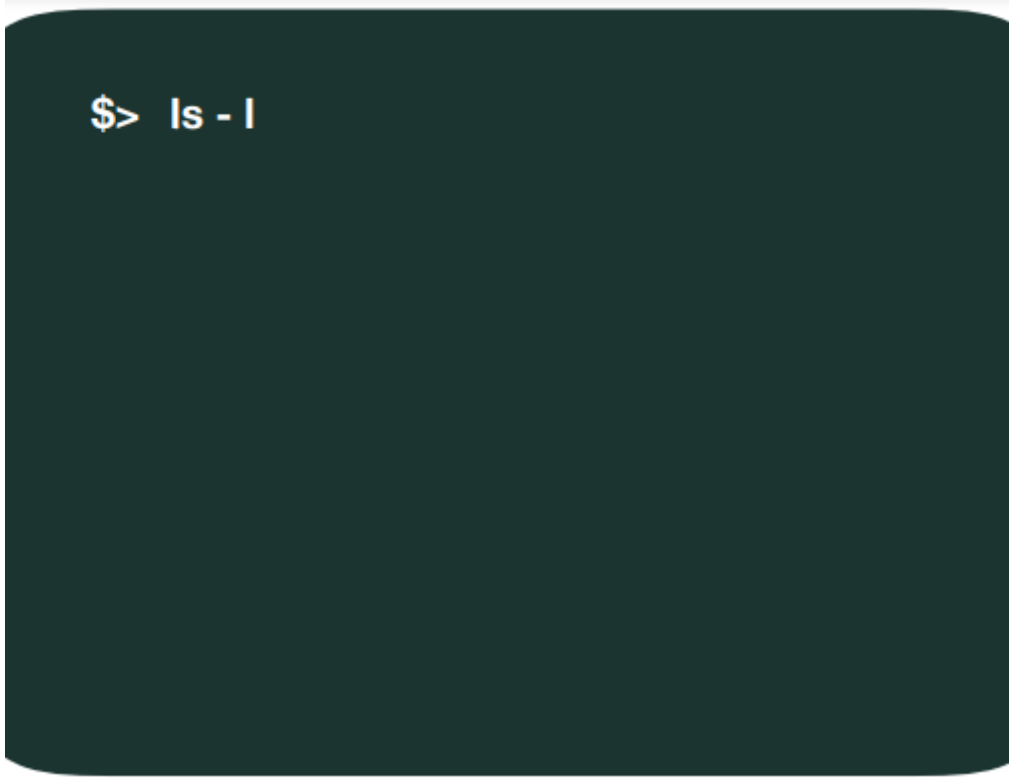
PHILIANCE

FORMATION

# Introduction

- Le système d'exploitation UNIX/LINUX dispose de plusieurs interpréteurs de commandes pour les utilisateurs.

Avant de commencer l'étude des scripts et des algorithmes utilisés (boucles, tests, sauts, etc.), il est recommandé de connaître et de pratiquer la ligne de commande



# Rappel des commandes de base

# Afficher le contenu d'un fichier: cat

cat

- Utilisation: **cat** [***OPTIONS***] [***FICHIER***]
- Cette commande permet **d'afficher** sur la sortie standard du terminal le **contenu** d'un **fichier** sur la sortie standard, exemple:
  - '`echo "Bonjour" > file.txt`' -> Ici on **écrit** "Bonjour" dans **file.txt**
  - Puis on exécute '`cat file.txt`' qui nous **affichera** "Bonjour" sur la sortie standard

cd

# Se déplacer dans des dossiers: **cd**

- Utilisation: **cd** [***DOSSIER***, ***CHEMIN***]
- Comme constaté, vous utiliserez un **terminal** et pas une **interface graphique** pour vous **déplacer** dans vos différents dossiers.
- Exemple: '`cd Rendu/ex01`' ouvrira le dossier ex01 **contenu** dans **Rendu**
- Petite précision: le dossier parent de votre dossier actuel sera nommé comme '**..**' (Tapez `ls -a` pour les liens à ces dossiers).
- Pour remonter dans vos dossiers, tapez donc '`cd ..`'

# chmod

# Modifier les permissions d'un fichier: chmod

- Utilisation: **chmod** [*OCTET_PROPRIETAIRE*][*O_GROUPE*][*O_UTILISATEURS*]
- Cette commande permet de modifier les **permissions** de **lecture**, **écriture** et **exécution** d'un **fichier** pour le propriétaire, le groupe propriétaire du fichier et les autres utilisateurs. Chaque type de permission à son propre numéro:
  - **4** pour les droits de **lecture**
  - **2** pour les droits **d'écriture**
  - **1** pour les droits **d'exécution**
- On peut donc **additionner ces chiffres** pour attribuer les permissions aux **différents utilisateurs**.
- Par exemple: `Chmod 754 file.txt`:
  - Le premier chiffre, **7** (4+2+1) donne les droits d'écriture, lecture et exécution au propriétaire sur file.txt
  - Le second chiffre, **5** (4+1) donne les droits de lecture et d'exécution au groupe sur file.txt
  - Le troisième chiffre, **4** donne seulement le droit de lecture aux autres utilisateurs sur file.txt

echo

# Afficher une ligne de texte: echo

- Utilisation: **echo** [***OPTIONS***] [***TEXTE***]
- Elle permet tout simplement **d'afficher** une **ligne** de **texte** sur la sortie standard. Voici quelques-unes de ses options:
  - `-e` : affiche les **caractères spéciaux** ('\n', '\t', etc...)
  - `-n` : **n'affiche pas** la **nouvelle ligne** après l'affichage
- Cette commande permet surtout d'écrire dans des fichiers sans avoir à ouvrir ces derniers, en utilisant les redirections (symboles: <, <<, >, >>).
  - Exemple: '`echo "Bonjour" > file.txt`' va écrire **Bonjour** dans le **fichier file.txt**

# Afficher les occurrences dans un fichier: grep

grep

- Utilisation: **grep** [***OPTIONS***] **PATTERN** [***FILE***...]
- Cette commande, qui comporte de nombreuses options, permet de **retrouver** et **d'afficher** toutes les **occurrences** d'un **motif** dans un fichier.
  - Exemple: '`grep "test" file.txt`' va **chercher** toutes les **occurrences** de **test** dans le **fichier file.txt** et les **afficher** avec la ou les lignes dans lesquelles le motif est **contenu**

# Lister les fichiers: ls

ls

- Utilisation: **ls** [***OPTIONS***] [***DOSSIER***]
- La commande ls vous permet de **lister** tous les **fichiers** et **dossier** présent dans un dossier. Elle a de nombreuses options, voici les plus utilisées:
  - `ls -l`: permet de lister un fichier **par ligne**
  - `ls -a`: permet de lister les **fichiers cachés**. Cela vous permet aussi de voir le **lien** avec le **dossier parent**, '..', comme vu pour la commande cd
  - `ls -R`: permet de lister **récursivement** les dossiers et fichiers d'un dossier, c'est-à-dire tous **les fichiers** et **dossier contenus dans les sous-dossiers** de celui ou vous taperez la commande
- Vous pouvez **spécifier** le dossier à lister, comme avec la commande '`ls -l /home/Rendu`' pour voir le contenu du **dossier Rendu**

man

# Le "manuel" de linux: man

- Utilisation: **man** [***OPTIONS***] [***nom_de_la_commande***]
- Cette commande vous sera très utile car elle permet d'avoir toutes les informations sur une commande linux, comme par exemple:
  - Son **utilisation** et son **fonctionnement**
  - Les **options** qu'elle propose
  - La "valeur retournée" et beaucoup d'autres choses
- Essayez de taper par exemple '`man mkdir`'

mkdir

# Création d'un dossier: mkdir

- Utilisation: **mkdir** [***OPTIONS***] [***DOSSIER***]
- Cette commande permet de **créer** un nouveau **dossier** à l'emplacement actuel:
  - Exemple: '`mkdir test`' va créer le dossier test **dans** votre **dossier de travail**
- Vous pouvez **spécifier** le **chemin** pour la création de votre dossier
  - Ex: '`mkdir /home/Rendu/ex01`' va créer le **dossier** ex01 dans le **dossier Rendu**

# sed

## Manipuler du texte dans des fichiers ou flux: sed

- Utilisation: **sed** [***OPTIONS***] [***SCRIPT***] [***FICHIER***]
- La commande **sed** (**Stream EDitor**) permet d'effectuer des **modifications** dans un **fichier** ou dans un **flux** (avec une redirection, un tube par exemple). Aussi puissante que complexe, voici certaines de ses utilisations:
  - `sed –i 's/test/toast/g' file.txt` **remplace** (avec le s/ pour substitute) tous les "**test**" en "**toast**" (le /g signifiant global) en modifiant directement (avec –i ) le **fichier file.txt**
  - `sed "1,10d" file.txt` **affiche** sur la sortie choisie (ici standard) le **fichier file.txt** en **enlevant** les lignes **1 à 10**

touch

# Création d'un fichier: touch

- Utilisation: **touch** [***NOM_FICHIER***]
- La commande touch vous permettra de **créer** un **fichier** dans le **dossier actuel**
  - Exemple: '`touch file`' va créer un fichier nommé file
- Au même titre que **mkdir**, vous pouvez **spécifier le chemin** dans lequel vous voulez créer ce fichier
  - Exemple: '`touch ~/Rendu/ex01/test`' va créer le fichier test dans /home/Rendu/ex01
- Cette commande n'est pas très utilisée puisque la majorité des éditeurs de textes (comme emacs sur vos machines) vont créer automatiquement le fichier s'il n'est pas déjà existant.

wc

# Compter les mots d'un fichier: wc

- Utilisation: **wc** [***OPTIONS***] [***FICHIER***]
- Cette commande **wc** (**word counter**) permet de **compter mots** (option –w), caractères (option –m), lignes (option –l) ou encore les bytes (option –c) dans un fichier. Voici quelques exemples de son utilisation:
- On prend un **fichier file.txt** contenant la phrase "**Hello Word**", puis on exécute:
  - `'wc -w file.txt'` **affichera** sur la sortie standard **2 pour 2 mots** (l'option –w)
  - `'wc –m file.txt'` **affichera** sur la sortie standard **12 pour 12 caractères**
  - `'wc –l file.txt'` **affichera** sur la sortie standard **1 pour 1 ligne** du **fichier**

# Le shebang

Le shebang, représenté par `#!`, est un en-tête d'un fichier texte qui indique au système d'exploitation que ce fichier n'est pas un fichier binaire mais un script (ensemble de commandes) ; sur la même ligne est précisé l'interpréteur permettant d'exécuter ce script. Pour indiquer au système qu'il s'agit d'un script qui sera interprété par bash on placera le shebang sur la première ligne :

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

- #!/bin/env/bash
- #!/usr/bin/env/perl
- #!/bin/env/sh
- #!/usr/bin/env/python

Pour connaître votre shell actuel, tapez la commande :

**#** echo $SHELL
/bin/bash

```
vhugo@debian:~$ echo $SHELL
/bin/bash
vhugo@debian:~$
```

# Les SCRIPTS

**Un script shell est un programme informatique développé pour fonctionner dans un shell Unix, un interpréteur de commandes.**

**Le scripting va consister à effectuer des petites tâches**

**pour faire gagner pas mal de temps dans l'AdminSys**

# Exemple : afficher "Hello World" dans le terminal

```
#!/bin/bash
# script0.sh
echo "Hello World"
exit
```

# Les variables prépositionnées

- `$0` : nom du script. Plus précisément, il s'agit du paramètre 0 de la ligne de commande, équivalent de `argv[0]`
- `$1`, `$2`, ..., `$9` : respectivement premier, deuxième, ..., neuvième paramètre de la ligne de commande
- `$*` : tous les paramètres vus comme un seul mot

```
#!/bin/bash
# script1.sh
echo "Nom du script $0"
echo "premier paramètre $1"
echo "second paramètre $2"
echo "PID du shell " $$
echo "code de retour $?"
exit
```

# Les variables internes

En début de script, on peut définir la valeur de départ des variables utilisées dans le script. Elles ne sont connues que par le processus associé au lancement du script.

```
VARIABLE="valeur"
```

Elles s'appellent comme ceci dans le script :

```
echo $VARIABLE
```

Il peut être utile de marquer les limites d'une variable avec les accolades.

```
echo ${VARIABLE}
```

Par exemple :

```
#!/bin/bash
# script2.sh
PRENOM="francois"
echo "dossier personnel /home/${PRENOM}"
exit
```

# L'Interaction utilisateur

La commande `echo` pose une question à l'utilisateur.

La commande `read` lit les valeurs entrées au clavier et les stocke dans une variable à réutiliser.

```
echo "question"
read reponse
echo $response
```

On peut aller plus vite avec `read -p` qui sort du texte et attend une valeur en entrée :

```
read -p "question" reponse
echo $reponse
```

# Les fonctions

Une fonction est un bloc d'instructions que l'on peut appeller ailleurs dans le script. Pour déclarer une fonction, on utilise la syntaxe suivante :

```
maFonction()
{
  echo hello world
}
```

Ou de manière plus ancienne :

```
function ma_fonction {
  echo hello world
}
```

La déclaration d'une fonction doit toujours se situer avant son appel. On mettra donc les fonctions en début de script.

# Exemple de fonction

```
#!/bin/bash
# script3.sh
read -p "quel votre prénom ?" prenom
reponse() {
    echo $0
    echo "merci $prenom"
    exit 1
}
reponse
exit
```

# Les Conditions (ou structures conditionnelles)

```
if condition ; then
    commande1
else
    commande2
fi
```

# Exemple de condition IF

```
#!/bin/bash
# script4.sh test si $passwdir existe
passwdir=/etc/passwdd
checkdir() {
    if [ -e $passwdir ]; then
        echo "le fichier $passwdir existe"
    else
        echo "le fichier $passwdir n'existe pas"
    fi
}
checkdir
exit
```

# Les Boucles

Faire la même chose pour tous les éléments d'une liste. En programmation, on est souvent amené à faire la même chose pour tous les éléments d'une liste. Dans un shell script, il est bien évidemment possible de ne pas réécrire dix fois la même chose. On dira que l'on fait une boucle. Dans la boucle "for-do-done", la variable prendra successivement les valeurs dans la liste et les commandes à l'intérieur du "do-done" seront répétées pour chacune de ces valeurs.

```
for variable in liste_de_valeur ; do
    commande
    commande
done
```

# Exemple de boucle FOR 1/2

Supposons que nous souhaitions créer 10 fichiers .tar.gz factices, en une seule ligne :

```
for num in 0 1 2 3 4 5 6 7 8 9 ; do touch fichier$num.tar.gz ; done
```

Mieux :

```
for num in {0..9} ; do touch fichier$num.tar.gz ; done
```

# Exemple de boucle FOR 2/2

Supposons que nous souhaitions renommer tous nos fichiers `*.tar.gz` en `*.tar.gz.old` :

```
#!/bin/bash
# script6.sh boucle
#x prend chacune des valeurs possibles correspondant au motif : *.tar.gz
for x in ./*.tar.gz ; do
    # tous les fichiers $x sont renommés $x.old
    echo "$x -> $x.old"
    mv "$x" "$x.old"
#on finit notre boucle
done
exit
```

# La Boucle WHILE

Faire une même chose tant qu'une certaine condition est remplie. Pour faire une certaine chose tant qu'une condition est remplie, on utilise une boucle de type "while-do-done" et "until-do-done".

```
while condition ; do
    commandes
done
```

`while;do` répète les commandes tant que la condition est vérifiée.

```
until condition ; do
    commandes
done
```

`until ; do ; done` répète les commandes jusqu'à ce que la condition soit vraie, ou alors tant qu'elle est fausse.

# Protection des variables

On peut annuler la signification des caractères spéciaux comme `*`, `?`, `#`, `|`, `[]`, `{}` en utilisant des caractères d'échappement, qui sont également des caractères génériques.

**\ Antislash**

L'antislash `\`, qu'on appelle le caractère d'échappement, annule le sens de tous les caractères génériques, en forçant le shell à les interpréter littéralement.

```
echo \$var
$var
echo "\$var"
$var
```

### " " Guillemets

Les guillemets (doubles) `" "` sont les guillemets faibles mais annulent la plupart des méta-caractères entourés à l'exception du tube ( `|` ), de l'antislash ( `\` ) et des variables ( `$var` ).

```
var=5
echo la valeur de la variable est $var
la valeur de la variable est 5
echo "la valeur de la variable est $var"
la valeur de la variable est 5
```

### ' ' Apostrophes

Les guillemets simples, ou apostrophes ( `' '` ) annulent le sens de tous les caractères génériques sauf l'antislash.

```
echo '\$var'
\$var
```