

\$ SHELL

Victorien Elvinger

Octobre 2017

 Université de Lorraine - Telecom Nancy

Commandes

- Chaque paramètre est séparé par un espace

```
$ program param1 param2
```

- Ex. `-l` et `~/Desktop` sont deux paramètres passés à `ls`

```
$ ls -l ~/Desktop
```

- Les guillemets simples permettent de passer des paramètres qui incluent des espaces ou des caractères spéciaux (\$, *, ...)

```
$ ls -l '~/my folder'
```

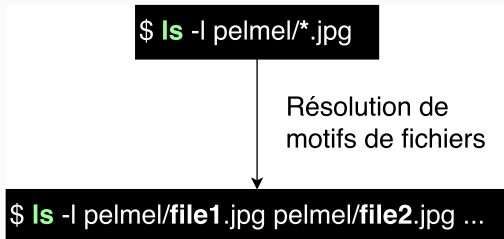
Motifs de fichiers

- Désigne un ensemble de fichiers
ex. tous les fichiers avec l'extension *.jpg*
- Composé de caractères spéciaux
 - ? représente n'importe quel caractère
 - * représente n'importe quel nombre (incluant aucun) de un ou plusieurs caractères
 - ... voir Wikipedia ([glob programming](#))

ex. ***.jpg**

Résolutions de motifs de fichiers

- Le shell résout les motifs de fichiers avant l'exécution de la commande



Exercices : Motifs de fichiers

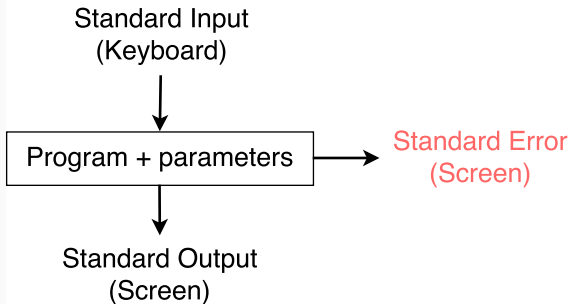
- Quel motif désigne les fichiers dont le nom débute par **td** ?
- Quel motif désigne les photos d'extension **jpg** dont le nom contient **2017** ?
- Quel motif désigne les fichiers qui contiennent au moins un espace ?

Correction : Motifs de fichiers

- Quel motif désigne les fichiers dont le nom débute par **td** ?
 - **td***
- Quel motif désigne les photos d'extension **jpg** dont le nom contient **2017** ?
 - **2017*.jpg**
- Quel motif désigne les fichiers qui contiennent au moins un espace ?
 - ***' '***

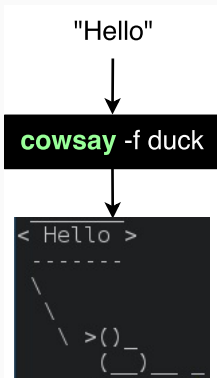
Entrée / Sortie Standards

- Un programme peut
 - recevoir des données sur l'Entrée Standard
 - rendre des résultats sur la Sortie Standard
- Ne pas confondre paramètres et entrée standard



Entrée / Sortie Standards - Exemple

1. Le programme **cowsay -f duck** est lancé
2. L'utilisateur imprime "hello" sur l'entrée standard et valide
3. Le programme imprime le résultat sur la sortie standard



Redirections des E/S

- *command > file.txt*
command >> file.txt
 - Redirige la sortie standard vers un fichier
 - > écrase le fichier si il existe
 - >> ajoute le contenu à la fin du fichier
- *command < file.txt*
 - Utilise un fichier comme entrée standard
- *command1 | command2*
 - Redirige la sortie standard d'un premier programme vers l'entrée standard d'un second programme

Exemples de redirections

- **cowsay -f duck < file.text** utilise le contenu de file.text comme message
- **echo Hello | cowsay -f duck | cowsay -n -f duck**



- **yes | apt install** permet d'imprimer "y" (yes) à chaque fois que le programme attend une entrée

- Préfixées par **\$**
 - **\$HOME** contient le chemin absolu du dossier personnel de l'utilisateur courant
 - **\$PATH** contient les chemins absolus vers les dossiers dans lesquels les programmes sont recherchés
- **a=1** déclare ou modifie la variable **\$a** avec la valeur **1**
Pas d'espace autour de **=**
- **unset a** supprime la variable **\$a**

Substitutions de variables

- Le Shell remplace les variables par leur valeurs avant d'exécuter la commande

```
$ dir='/home'  
$ ls -l $dir
```

Substitution de
variables

```
$ ls -l /home
```

Substitutions de variables et paramètres

- La valeur d'une variable peut contenir des espaces
 - Une substitution peut donc produire plusieurs paramètres
 - Les guillemets double permettent de substituer une variable en un seul paramètre
- Une variable devrait toujours être entourée de guillemets double (à quelques rares exceptions)

```
$ dir='photos vacs'  
$ cd $dir
```

Substitution de
variables

```
$ cd photos vacs
```

```
$ dir='photos vacs'  
$ cd "$dir"
```

Substitution de
variables

```
$ cd 'photos vacs'
```

- Est autorisé à être exécuté
 - **chmod u+x script**
- Ecrit pour un interpréteur (python, node, sh, bash, ...)
- Un shebang **#!** indique au Shell quel interpréteur utiliser
 - Il est placé sur la première ligne du fichier
 - Il est suivi par le chemin absolu de l'interpréteur
 - #! /bin/sh**
 - #! /bin/bash**
 - #! /usr/bin/python**
- Si l'interpréteur n'est pas indiqué, le Shell essaye de l'interpréter

- POSIX offre un standard pour le langage Shell
 - Voir [Shell Command Language](#)
- Bash respecte en grande partie POSIX et offre beaucoup de structures supplémentaires
 - Si vous avez un doute, préférez toujours utiliser `#!/bin/bash` au lieu de `#!/bin/sh`
- Dans ce cours nous apprenons les bases de POSIX
 - Nous utiliserons donc `#!/bin/sh`

Exercice : separer

- Ecrire un script nommé **separer** qui déplace les fichiers d'extension **jpg** du sous-dossier **pelmel** vers un sous-dossier **sep**
- Que pourrions-nous faire pour le rendre plus générique ?

Correction : separer

```
#!/bin/sh  
mkdir sep  
mv pelmel/*.jpg sep/
```

- **\$0** est le nom du script / programme tel que appelé
- **\$n** est le n-ème paramètre
- **\$#** est le nombre de paramètres
- **\$***, **\$@** est la liste des paramètres séparés par des espaces

Exercice : separer avec paramètres

- Adapter le script **separer** pour qu'il déplace les fichiers d'une extension particulière d'un dossier donné vers un sous-dossier **sep**
 - `./separer jpg pelmel`
 - `./separer txt pelmel`
- Que pourrions-nous faire pour le rendre plus robuste ?

Correction : separer avec paramètres

```
#!/bin/sh  
mkdir sep  
mv "$2" /*."$1" sep/
```

- L'exécution d'une commande réussit ou échoue
- Après son exécution, une commande renvoie un status qui est un entier naturel
 - **0** indique un succès
 - Un entier plus grand que 0 indique un échec
- **exit n** retourne le status *n* au Shell appelant

- **test** **EXPR** ou **[EXPR]** permet de tester des conditions
 - **['abc e' = 'abc e']** exit status: 0
 - **["\$var" != 'a']** pour var=a, exit status: 1
 - **[1 -eq 1]** exit status: 0
 - **[1 -lt 2 -a 2 -gt 1]** exit status: 0
- **]** est un paramètre obligatoire de la commande **[**
- Voir le manuel d'utilisation (**man test** ou **man [**)

Instruction Conditionnelles

```
if command
then
    # command has 0 as exit status
    command1
else
    command2
fi
```

Itérations : tant-que

```
while command
do
    # command has still 0 as exit status
    command1
done
```


Exercice : choix

- Écrire un programme qui boucle sur l'entrée standard jusqu'à obtenir la chaîne de caractères "yes" ou "no" et l'imprimer sur la sortie standard
- **read answer** lit une ligne sur l'entrée standard et l'écrit dans la variable **\$answer**
- Une amélioration possible est de personnaliser le choix
 - **choix a b** attends **a** ou **b**

Itérations : for-in

for var **in** sequence

do

command1

done

unset var

- La séquence est un ensemble de valeurs séparées par des espaces
 - **\$***
 - **1 2 3**
 - **alice karim tristan hadjer**

Séquence "\$@"

- "\$@" a une règle de substitution particulière
 - "\$@" est équivalent à "\$1" "\$2" ...
 - "\$@" est donc la séquence des paramètres
- A l'inverse "\$*" suit la règle de substitution usuelle
 - "\$*" est équivalent à "\$1 \$2 ..."
 - "\$*" est donc une séquence avec un seul élément

Exercice : enlever

- Écrire un programme qui enlève un nom d'une séquence de noms
 - **enlever tristan alice karim tristan hadjer**
imprime sur la sortie standard :
alice karim hadjer
- **shift** décale les paramètres
\$j devient \$i avec $j = i + 1$
Attention : \$0 est inchangé
\$# est décrémenté de un
- L'option **-n** de **echo** empêche **echo** d'imprimer un retour à la ligne

Substitution de commandes

- **\$(command)** est substitué par ce qui est imprimé sur la sortie standard par **command**
- **`command`** (backquotes) est l'ancienne syntaxe de **\$(command)**
- Exemples
 - `cowsay -f "$(choix duck tux)" < file.txt`

CC-BY 4.0 - Attribution 4.0 International