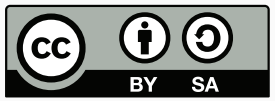
$ SHELL

Jean-Philippe Eisenbarth

Janvier 2021

Université de Lorraine - Telecom Nancy



À propos

Remerciements :

- Victorien Elvinger pour son cours Shell (2017)
- Benjamin Ségault pour le poly du cours

Commandes

- Chaque paramètre/option est séparé par une espace

```
$ program -option1 -option2 param1 param2
```

- Ex. -l et ∽/Desktop sont une option et un paramètre 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[1]

- Désigne un ensemble de fichiers

  e.g. tous les fichiers avec l'extension .jpg
- Composés de caractères spéciaux (wildcards en anglais)
  - ? représente n'importe quel caractère
  - * représente 0,1 ou plusieurs caractères

  e.g. *.jpg, cat?.png (glob pattern en anglais)

[1] voir Wikipedia (glob programming)

Résolutions de motifs de fichiers

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

```
$ ls -l directory/*.jpg
```

```
$ ls -l directory/file1.jpg directory/file2.jpg
```

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 termine par 2018 ?
- Quel motif désigne les fichiers qui contiennent au moins une espace ?

Exercices : 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 termine par 2018 ?

- Quel motif désigne les fichiers qui contiennent au moins une espace ?

Exercices : Motifs de fichiers

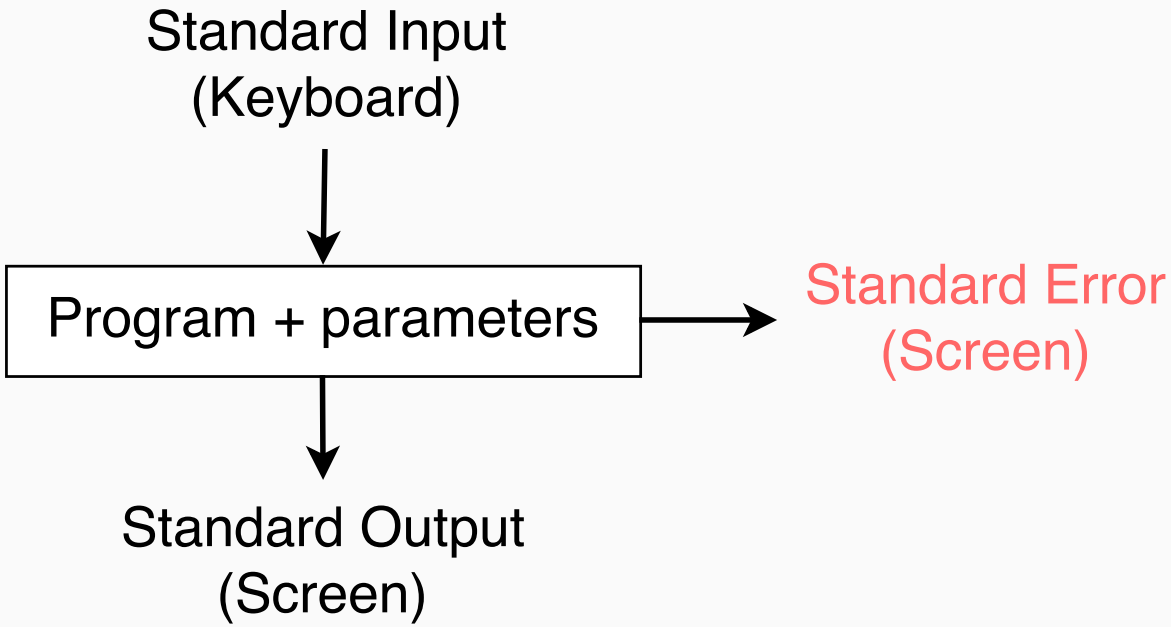
- Quel motif désigne les fichiers dont le nom débute par td ?
  - td*
- Quel motif désigne les photos d'extension jpg dont le nom termine par 2018 ?
  - *2018.jpg
- Quel motif désigne les fichiers qui contiennent au moins une espace ?

Exercices : 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 termine par 2018 ?
  - *2018.jpg
- Quel motif désigne les fichiers qui contiennent au moins une espace ?
  - *' '*

Entrée / Sortie Standards

- Un programme peut
  - recevoir des données sur l'entrée standard
  - renvoyer des résultats sur la sortie standard
- Ne pas confondre paramètres et entrée standard



Entrée / Sortie Standards - Exemple

1. Le programme cowsay -f stegosaurus 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

```
$ command1 > file.txt
```

et

```
$ command1 >> file.txt
```

→ Redirige la sortie standard vers un fichier

- `>` écrase le fichier s'il existe
- `>>` ajoute le contenu à la fin du fichier

Redirections des E/S

```
$ command1 < 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 stegosaurus < file.txt
```

```
$ echo Hello | cowsay -f stegosaurus |
↪  cowsay -nf stegosaurus
```

```
$ yes | apt install
```

Variables

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

Substitutions de variables

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

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

```
$ 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 doubles permettent de substituer une variable en un seul paramètre
- Une variable devrait toujours être entourée de guillemets doubles (à quelques rares exceptions)

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

↓

```
$ cd photos vacs
```

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

↓

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

Fichier texte exécutable

- Droit d'exécution ?
  - ```
    $ chmod u+x script.sh
    ```
- Écrit pour un interpréteur (python, ruby, node, sh, bash, ...)
- Un shebang #! indique au Shell quel interpréteur utiliser
  - Placé sur la première ligne du fichier
  - Suivi par le chemin absolu de l'interpréteur

    #! /bin/sh

    #! /bin/bash

    #! /usr/bin/python

Bonne pratique : utilisez #! /usr/bin/env bash
    ```

POSIX

- 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 `#! /usr/bin/env bash` au lieu de `#! /usr/bin/env sh`

Exercice : move.sh

- Ecrire un script nommé separer.sh qui déplace les fichiers d'extension jpg du sous-dossier **`photo`** vers un sous-dossier **`photo_jpg`** qui n'existe pas encore
- Que pourrions-nous faire pour le rendre plus générique ?

Correction : move.sh

```
1  #! /usr/bin/env bash
2  set -ex # arrêt de l'exécution lorsqu'une commande
   ↪  rencontre une erreur ; affiche les commandes qui
   ↪  sont exécutées
3
4  mkdir photo_jpg
5  mv photo/*.jpg photo_jpg/
```

Récupération de paramètres

- $0 est le nom du script / programme tel que appelé
- $n est le n-ième paramètre (n entre 1 et 9)
- ${n} pour n > 9
- $# est le nombre de paramètres
- $*, $@ est la liste des paramètres séparés par des espaces

Exercice : move.sh avec paramètres

- Adapter le script move.sh pour qu'il déplace les fichiers d'une extension particulière d'un dossier donné vers un sous-dossier `photo_jpg`
  - e.g. ./move.sh jpg photos
  - e.g. ./move.sh txt photos
- Que pourrions-nous faire pour le rendre plus robuste ?

Correction : move.sh avec paramètres

```
1 #! /usr/bin/env bash
2 set -ex # arrêt de l'exécution lorsqu'une commande
  ↪ rencontre une erreur ; affiche les commandes qui
  ↪ sont exécutés
3
4 mkdir photo_jpg
5 mv "$2"/*."$1" photo_jpg
```

Exit Status

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

Programme test ou [

- 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 ] && [ 2 -gt 1 ]` → exit status: 0
- Ne pas oublier les espaces entre chaque paramètres
- Voir le manuel d'utilisation (man test ou man [)
- en bash on peut utiliser [[ ]], voir poly du cours ou cette BashFAQ

Instruction conditionnelle

```
1  if command; then
2      # command has 0 as exit status
3      ...
4  else
5      ...
6  fi
```

Instruction conditionnelle - exemple

```
1  if [ $# -gt 0 ]; then
2      # There exists at least one parameter
3      echo "$1"
4  else
5      echo "usage: $0 param1"
6      exit 1
7  fi
```

Instruction d'itérations : while

```
1  while command; do
2      # while command has 0 as exit status
3      ...
4  done
```

Exercice : lecture

Écrire un script lecture.sh permettant de boucler sur une lecture au clavier jusqu'à l'obtention de la chaîne de caractères "oui" ou "non"

Aide : La commande read lit une ligne sur l'entrée standard et stocke chaque mot de la ligne dans les paramètres passé à la commande

e.g. read answer lit un mot et le stocke dans la variable answer

Instruction d'itérations : foreach

```
1  for var in sequence; do
2      ...
3  done
```

- La séquence est un ensemble de valeurs séparées par des espaces
  - 1 2 3
  - alice bob charlie david eve
  - 'first element' 'second element'

Séquence "$@"

- "$@" a une règle de substitution particulière
  - "$@" est équivalent à "$1" "$2" ...
- A l'inverse "$*" suit la règle de substitution usuelle
  - "$*" est équivalent à "$1 $2 ..."

Exercice : enlever

- Écrire un script enlever nom liste_de_noms qui affiche sur la sortie standard la liste des noms privée de toutes les occurences de nom

Aide :

- shift décale les paramètres
  $1 prends la valeur de $2, $2 prend la valeur de $3, ...
  Attention : $0 reste inchangé
  $# est décrémenté de un
- L'option -n de echo l'empêche 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 une syntaxe alternative de $(command)
- Exemples
  - cowsay -f "$(choix duck tux)" < file.txt
  - enlever a $(enlever b x y a c b)