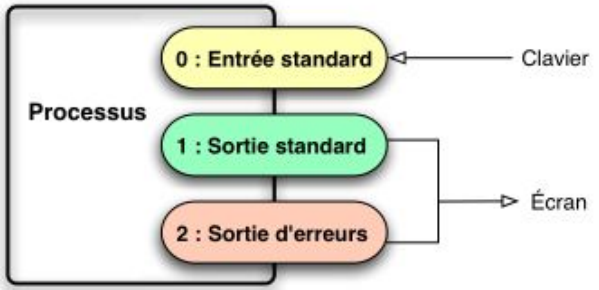
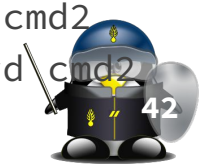
Fonctionnement interactif

semaine 4

Rappels



- Redirection des entrées/sorties (voir cours 1e semestre)
  - >,1>,2>,>>,1>>,2>>,>&2,1>&2,2>&1
  - <,0< (alimenter une commande depuis un fichier)
  - <<, 0<< (saisie de document en ligne)
- De manière générale :
  - n< , n> , et n>> , dans lesquelles n représente un numéro de descripteur de fichier (par défaut, 0 pour < et << , et 1 pour > et >>).
    - redirection des deux sorties : &> et &>>
  - n>&m , où n et m sont deux numéros de descripteurs, la sortie vers n sera redirigée vers m
- Les pipes “tubes” (voir cours 1e semestre)
  - opérateur | redirige sortie standard de cmd1 vers entrée standard cmd2
  - opérateur |& redirige les deux sorties de cmd1 vers entrée standard cmd2



Rappels: Substitution de commande

- Syntaxe : `commande` ou $(commande)
- Exemple :
  - $ echo "date actuelle = `date`"
  - date actuelle = lundi 27 mars 2017, 16:30:38 (UTC+0100)
  - $ echo "date actuelle = $(date)"
  - date actuelle = lundi 27 mars 2017, 16:30:38 (UTC+0100)
  - $ echo "Nombre de fichiers dans le répertoire: `ls | wc -w`"
  - Nombre de fichiers dans ce répertoire: 24

Extraction de sous-chaînes

- L'opérateur ${} est la version généralisée de l'opérateur $ permettant l'accès à la valeur d'une variable
- L'option ":" de l'opérateur ${} permet d'extraire une sous-chaîne d'une variable
  - La syntaxe est: ${variable:debut:longueur}
  - ATTENTION à la portabilité! n'existe pas dans SUS3.
- L'option "#" permet de supprimer le plus court préfixe correspondant au motif qui vient après # :
  - syntaxe : ${variable#motif}

```
$ variable=ABCDEFGHIJKLMNOPQRSTUVWXYZ
$ echo ${variable:5:2}
FG
$ echo ${variable:20}
UVWXYZ
```

```
$ variable=AZERTYUIOPAZERTYUIOP
$ echo ${variable#AZE}
RTYUIOPAZERTYUIOP
$ echo ${variable#*T}
YUIOPAZERTYUIOP
$ echo ${variable#*[MNOP]}
PAZERTYUIOP
```

Extraction de sous-chaînes

- L'expression `${variable##motif}` sert à éliminer le plus long préfixe correspondant au motif transmis.
- Symétriquement, les expressions `${variable%motif}` et `${variable%%motif}` correspondent au contenu de la variable, débarrassée respectivement, du plus court et du plus long suffixe correspondant au motif indiqué.

```
$ variable=AZERTYUIOPAZERTYUIOP

$ echo ${variable##*T}
YUIOP
$ echo ${variable##*[MNOP]}


$ echo ${variable%IOP*}
AZERTYUIOPAZERTYU
$ echo ${variable%%IOP*}
AZERTYU
$ echo ${variable%[X-Z]*}
AZERTYUIOPAZERT
$ echo ${variable%%[X-Z]*}
A
```

Remplacement de sous-chaînes

- ${variable/motif/remplacement} permet de remplacer la première occurrence du motif par la chaîne de remplacement.
- ${variable//motif/remplacement} permet de remplacer toutes les occurrences du motif par la chaîne de remplacement.
- ATTENTION! Cet opérateur est présent dans les shells Bash, Korn 93 et Zsh, mais n’est pas normalisé par SUS3; Aussi on risque d’avoir de légères divergences entre les différentes implémentations

Les variables : longueur du contenu

- ${#variable} permet d'avoir la longueur du contenu de la variable "variable"
- Si le contenu est un nombre, il est traité de la même manière qu'une chaîne de caractères

```
$ variable=AZERTYUIOP
$ echo ${#variable}
10

$ echo $UID
1000
$ echo ${#UID}
4

$ variable=
$ echo ${#variable}
0
$ echo ${#inexistante}
0
```

Les variables : Actions par défaut

Lorsqu'un script fonde son comportement sur des variables qui sont fournies par l'utilisateur, il est important de prévoir une action par défaut si la variable est vide ou n'existe pas.

- L'expression ${variable:-valeur} prend la valeur indiquée à droite du modificateur :- si la variable n'existe pas, ou si elle est vide. Si la variable existe et elle est non vide, l'expression renvoie le contenu de la variable.

```
$ variable=existante
$ echo ${variable:-defaut}
existante
$ variable=
$ echo ${variable:-defaut}
defaut
unset variable
$ echo ${variable:-defaut}
defaut
```

Les variables : Actions par défaut

- L'expression ${variable:=valeur} est similaire à la précédente, mais le contenu de variable sera modifié. Si la variable n'existe pas ou si elle est vide, elle est alors remplie avec valeur. Ensuite, dans tous les cas, le contenu de la variable est retourné.
- L'expression ${variable:?message} : Si la variable est définie et non vide, sa valeur est retournée. Sinon, le shell affiche le message fourni après le point d'interrogation, et abandonne le script ou la fonction en cours.
  - Si le message n'est pas précisé, le shell en affiche un par défaut

```
$ echo $vide
$ echo ${vide:=contenu}
contenu
$ echo $vide
contenu

$ unset vide
$ echo ${vide:?faut la
déclarer}
vide : faut la déclarer

$ echo ${vide:?}
bash: vide : paramètre vide ou
non défini
```

Les variables : Actions par défaut

- ${variable:+valeur} renvoie la valeur fournie à droite du symbole + si la variable est définie et non vide, sinon elle renvoie une chaîne vide
- Les quatre options précédentes de l'opérateur ${} considèrent au même titre les variables indéfinies et les variables contenant une chaîne vide. Il existe quatre modificateurs similaires qui n'agissent que si la variable est vraiment indéfinie ; il s'agit de ${ - } , ${ = } , ${ ? } , et ${ + }

```
$ existante=4
$ echo ${existante:+1}
1
$ echo ${inexistante:+1}

$ var=
$ definie=${var:+oui}
$ : ${definie:=non}
$ echo $definie
non
$ var=1
$ definie=${var:+oui}
$ : ${definie:=non}
$ echo $definie
oui
```

Variables de substitution

- Elles sont définies implicitement et peuvent être utilisées à tout moment dans le script
- Les variables positionnelles et les variables spéciales sont utilisées pour accéder aux informations qui sont fournies sur la ligne de commande lors de l'invocation d'un script (paramètres), mais aussi aux arguments transmis à une fonction. Les arguments sont placés dans des paramètres qui peuvent être consultés avec la syntaxe $1 , $2 , $3 , etc.
- L'affectation est impossible, on ne peut pas imaginer une affectation : 1=2!

Variables positionnelles

- Le premier argument transmis est accessible en lisant $1, le deuxième dans $2 , et ainsi de suite.
- Pour consulter le contenu d'un paramètre qui comporte plus d'un chiffre, il faut l'encadrer par des accolades ${10} (Bash)
- L'argument numéro zéro ($0) contient le nom du script tel qu'il a été invoqué
- Les variables positionnelles peuvent être modifiées :
  - avec la commande set :
    - $ set a b c
    - ATTENTION! l'ensemble des variables positionnelles est affecté!
  - avec la commande shift
    - Si suivie d'un nombre, ce dernier représente le nombre de décalages désirés

Variables positionnelles : exemples

```
$ set a b c
$ echo $1 $2 $3
a b c
$ set d e
$ echo $1 $2 $3
d e
```

```
#!/bin/bash
while [ -n "$1" ] ; do
echo $1
shift
done
```

```
$ set a b c d e
$ echo $0 $1 $2 $3
bash a b c
$ shift
$ echo $0 $1 $2 $3
bash b c d
$ shift
$ echo $0 $1 $2 $3
bash c d e
$ shift
$ echo $0 $1 $2 $3
bash d e
```

Variables spéciales

- $# : nombre d'arguments, sans compter le paramètre $0
- $* : liste de tous les arguments
- $@ : liste de tous les arguments (supporte les guillemets, la plus utilisée)
- Toutes ces variables sont modifiés par l'appel de shift et de set
- $? : code de sortie retourné par la dernière commande
- $$: numéro de processus du shell actuel
- $! : numéro du dernier processus en arrière plan
- $- drapeaux fournis au shell par set (options du shell)

TD/TP 4

1. Écrire un script qui affiche le répertoire de travail en cours, en remplaçant le répertoire personnel par le symbole ~ en vert (préfixe)
2. Écrire un script qui prend comme argument une adresse e-mail et affiche le nom de login correspondant.
3. En utilisant la commande mv, changez l'extension de tous les fichiers d'un répertoire donné en 1e argument, d'une extension donnée en 2e argument à une autre en 3e argument.

TD/TP 4

(suite 1)

4. Vous souhaitez écrire un script qui prend obligatoirement un paramètre d’une longueur de 5 caractères minimum. Écrire une expression qui permet de vérifier la syntaxe. En cas d’erreur, elle doit afficher la syntaxe correcte et quitter avec un code 1.