02 - Scripts shell

Gaétan Richard
gaetan.richard@info.unicaen.fr

L2 – S4 2011-2012

1. Commandes et exécution

Une commande

Regardons la commande suivante :

```
$ tr ”[a-z]” ”[A-Z]”
Bonjour
BONJOUR
echo
ECHO
$
```

Une commande

Regardons la commande suivante :

```
$ tr ”[a-z]” ”[A-Z]”
Bonjour
BONJOUR
echo
ECHO
$
```

On a :

- des arguments ;
- des entrées ;
- des sorties.

file descriptor

Pour interagir, le processus possède une liste de descripteur de fichiers (*file descriptor*) qui contient les fichiers (ou assimilés ouverts).

Les premiers ont une signification particulière et contiennent une valeur par défaut :

- **0** est l'entrée standard (**stdin**) : clavier
- **1** est la sortie standard (**stdout**) : écran
- **2** est la sortie d'erreur (**stderr**) : écran

Ouverture / fermeture / utilisation

Il est possible d'ajouter une fichier (ou assimilé) à la liste des descripteur de fichier à l'aide de commande d'ouverture (**open**) et de le retirer par un fermeture (**close**).

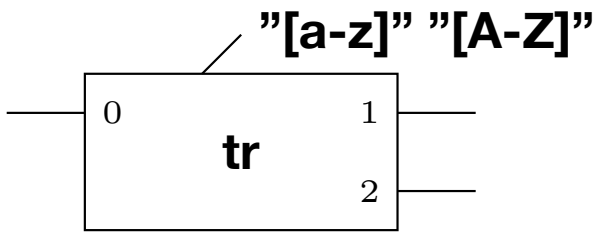
Nous verrons plus en détails cet aspect lors du cours sur les systèmes de fichiers.

Le modèle de la boîte

Pour représenter un processus et ses interactions, nous utiliserons un modèle symbolique de boîte qui met en valeurs :

- le programme ;
- ses arguments ;
- ses entrés sorties.

Nous utiliserons la représentation ci-dessous :



Redirections

Il est courant de vouloir rediriger les sorties standard ou d'erreur dans des fichiers ou inversement de rediriger un fichier vers l'entrée standard. On parle alors de **redirection**.

Pour cela, il suffit de remplacer la valeur situé à la position 0, 1 ou 2 de la liste des descripteurs de fichiers par une nouvelle valeur.

Cette opération se fait par l'intermédiaire de l'appel système

int dup2(int fildes, int fildes2)

qui recopie le descripteur à l'indice fildes à l'indice fildes2.

Redirections vers un fichier

Le shell possède un mécanisme pour rediriger en entrée ou en sortie vers un fichier au travers des symboles **<** et **>**.

- **< file** redirige le fichier `file` vers l'entrée standard ;
- **> file** redirige la sortie standard vers le fichier `file` ;
- **>> file** redirige la sortie standard vers la fin du fichier `file` ;
- **<< delimiter text ... delimiter** redirige le texte entre les délimiteurs (qui peuvent être choisis comme bon vous semble) vers l'entrée standard ;
- Toutes ces commandes peuvent être précédés d'un entier indiquant quel descripteur de fichier est concerné (par exemple **2> file** redirige le descripteur 2 (sortie d'erreur) vers le fichier `file`.

Redirections en shell (2)

Il est également possible de dupliquer ou fermer les descripteurs de fichiers à l'aide des opérateurs suivants :

- ***n1>&n2*** copie le descripteur d'indice $n2$ vers celui d'indice $n1$. (si $n1$ n'est pas précisé, il s'agit de la sortie standard ;
- ***n1<&n2*** similaire mais pour l'entrée ;
- **>&-** et **<&-** ferment respectivement la sortie standard, l'entrée standard.

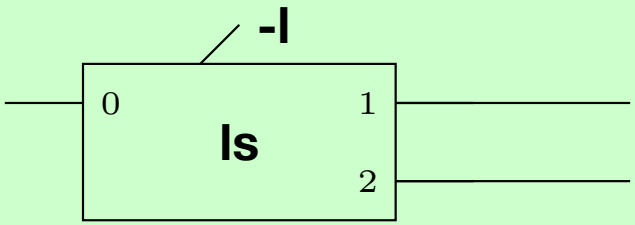
Note : seul le premier est couramment utilisé.

Exemples et représentations

Remarque : Attention à l'ordre des redirections.

Exemple :

```
ls -l 2>&1 > sav
```

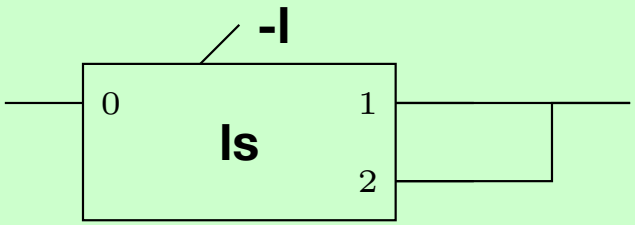


Exemples et représentations

Remarque : Attention à l'ordre des redirections.

Exemple :

```
ls -l 2>&1 > sav
```

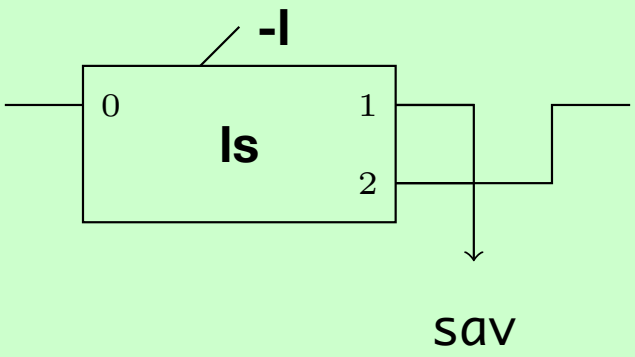


Exemples et représentations

Remarque : Attention à l'ordre des redirections.

Exemple :

```
ls -l 2>&1 > sav
```



Redirections en python

En python, on utilise directement la fonction :

os.dup2(fd, fd2) ;

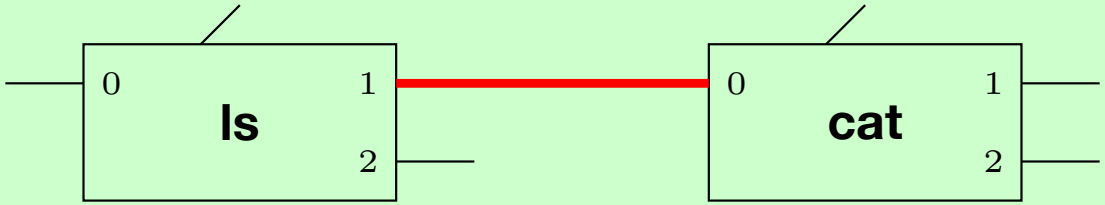
Description : Duplicate file descriptor fd to fd2, closing the latter first if necessary. Availability : Unix, Windows.

Pipe

Un certain nombre de commandes transforme l'entrée standard et revoie le résultat sur la sortie standard. On parle alors de filtre.
En shell, il est possible d'enchaîner des commandes en utilisant le pipe (tube en français) qui "branche" la sortie standard d'une commande sur l'entrée standard d'un autre.

Exemple :

```
$ ls | cat
...
```

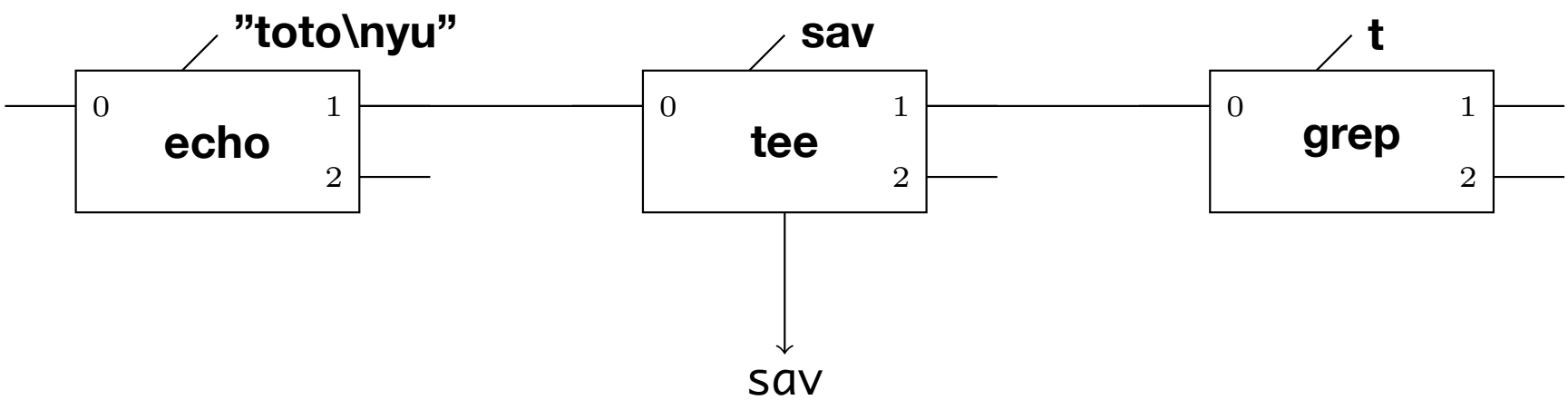


tee

Présentation : La commande tee *file* recopie l'entrée standard sur la sortie et en fait une copie dans le fichier `file`.

Exemple :

```
$ echo ”toto\nyu” | tee sav | grep t
toto
$
```

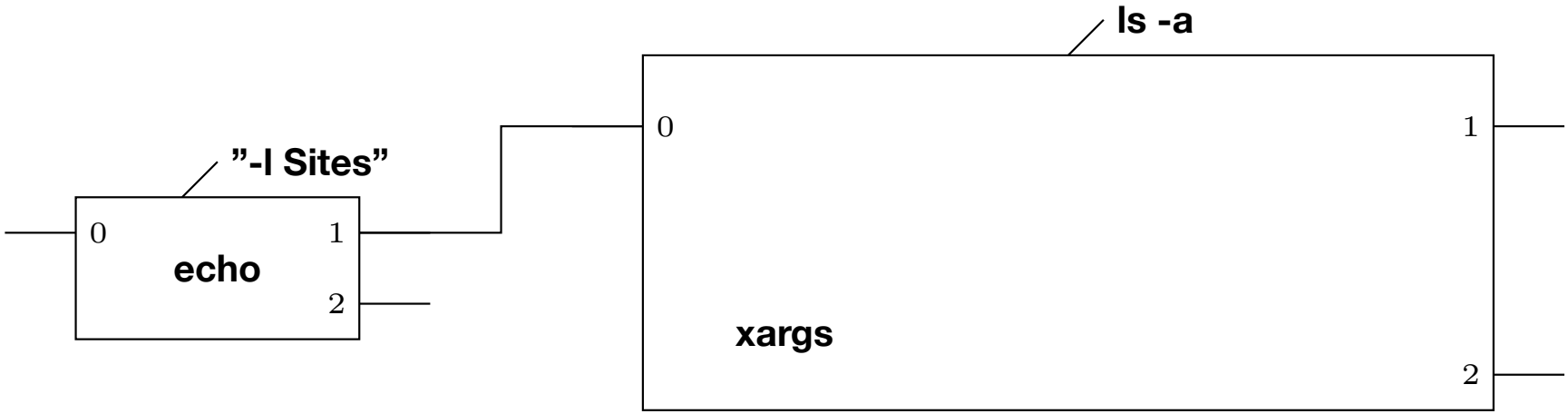


xargs

Présentation : La commande **xargs** permet de transférer l'entrée standard sur les arguments d'une fonction.

Exemple :

```
$ echo ”-l Sites” | xargs ls -a
total 8
drwxr-xr-x+  5 grichard  140   170 16 sep 17 :47 .
...
```

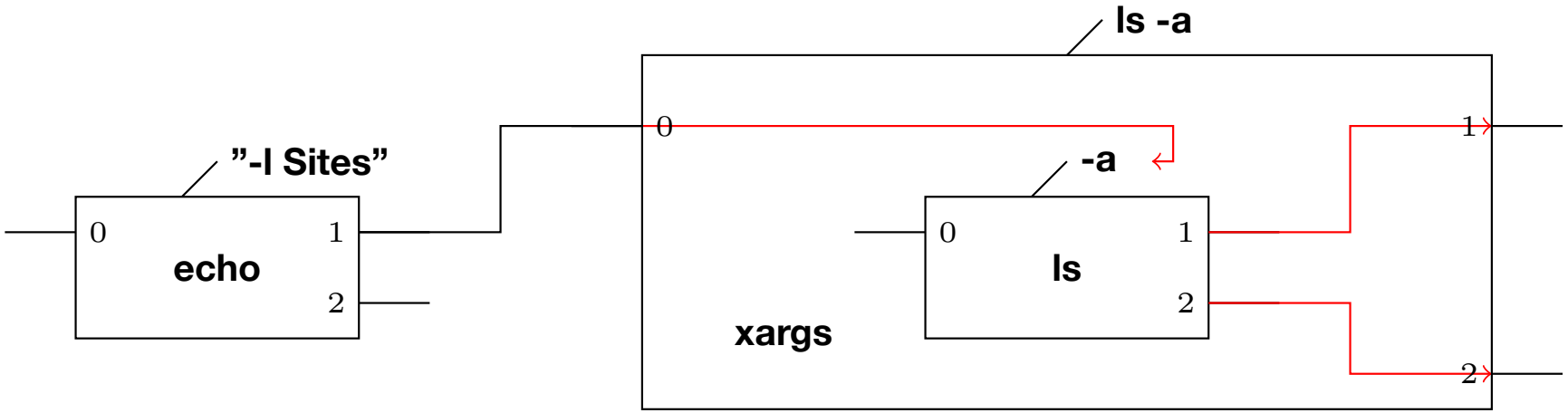


xargs

Présentation : La commande **xargs** permet de transférer l'entrée standard sur les arguments d'une fonction.

Exemple :

```
$ echo "-l Sites" | xargs ls -a
total 8
drwxr-xr-x+  5 grichard  140   170 16 sep 17 :47 .
...
```



2. Scripts

Principe du script

Principe :

Il existe des programmes qui prennent une suite d'instructions sur l'entrée standard et les exécute (ex : shell, python, perl, ...).

Il est possible d'écrire ces instructions dans un fichier et de demander de lancer le programme voulu en lui mettant le fichier en entrée standard.

Exemple :

```
$ python < script
...
$
```

Fonctionnement du script

Méthode :

- On ajoute une ligne spécifique au début du ficher commençant par ***#! chemin_du_programme*** ;
- On rend le fichier exécutable (**chmod u+x *fichier***).

Exemples :

```
#! /bin/python
print ”Ceci est un script python”
```

```
#! /bin/sh
echo ”Ceci est un script shell”
```

Environnement

Tout n'est pas situé dans les arguments / entrées.

Exemple :

```
$ date
Jeu 21 jan 2010 13 :51 :21 CET
$
```

Pourquoi la commande répond en français ?

Il existe deux autres sources d'information pour un processus :

- Les fichiers de configurations ;
- Les variables d'environnement.

Variables d'environnement

Les variables d'environnement sont des variables contenant des valeurs qui sont passées aux programmes lors de leur démarrage.

Usuellement, ces variables sont écrites intégralement en majuscule.

Quelques variables courantes

- **PATH** : emplacement où chercher les programmes ;
- **HOME** : emplacement du “home” ;
- **PWD** : répertoire courant ;
- **LANG** : locale utilisée ;
- **USER** : nom de l’utilisateur ;
- **PROMPT** : format du prompt ;
- **EDITOR** : éditeur de texte par défaut,
- …

Variables d'environnement en shell

Accès : Il suffit de préfixer la variable par un signe $.

```
$ echo $HOME
/Users/grichard
$
```

Modification : Avec une affectation (=) et en utilisant la commande **export**.

```
$ export LANG=C
$ date
Sun Jan 24 13 :18 :06 CET 2010
$ export PATH=$PATH :/sbin
$
```

Note : sur certains shell (csh, tcsh), la modification des variables d'environnement se fait à l'aide de la commande **setenv**

```
% setenv LANG C
%
```

Variables et substitutions

Il existe plusieurs variables utiles quand on utilise le shell. En particulier :

- la variable **?** contient la valeur de retour de la commande précédente ;
- la variable **$** contient le PID du shell courant (permet de distinguer les shells).

Il est également possible d'effectuer des substitutions dans les variables.

- **${*var*}** : même chose que sans les accolades ;
- **${#*var*}** : longueur du contenu de la variable *var* ;
- **${*var*%*motif*}** : valeur de la variable *var* auquel au retire le plus petit suffixe *motif* ;
- **${*var*#*motif*}** : valeur de la variable *var* auquel au retire le plus petit préfixe *motif* ;
- **${*var*%%*motif*}** et **${*var*##*motif*}** : même chose que précédemment avec le plus grand.

Exemples

```
bash-3.2$ echo $HOME
/Users/grichard
bash-3.2$ echo ${HOME}
/Users/grichard
bash-3.2$ echo ${#HOME}
15
bash-3.2$ echo ${HOME%rd}
/Users/gricha
bash-3.2$ echo ${HOME#rd}
/Users/grichard
bash-3.2$ echo ${HOME#/Users/}
grichard
```

Variables d'environnement en python

Les variables d'environnement se trouvent dans le dictionnaire

os.environ

Exemple :

```
>>> import os
>>> os.environ["HOME"]
'/Users/grichard'
>>> os.environ["HOME"]='/chez/moi'
>>> os.environ["HOME"]
'/chez/moi'
>>>
```

Note : il existe également les fonctions **putenv** et **getenv**.

Lecture / écriture

Si un descripteur de fichier est ouvert en lecture (resp. écriture), il est alors possible de lire (resp. d'écrire) dessus.

En général, les fonctions de lecture (resp. d'écriture) d'un langage se font implicitement sur l'entrée standard (resp. la sortie standard) mais il est possible d'utiliser n'importe quel descripteur de fichier avec la syntaxe adaptée.

Il existe aussi souvent une fonction permettant d'afficher directement une chaîne de caractère sur la sortie d'erreur.

En shell

Méthodes :

Écriture : **echo *chaîne*** / lecture : **read *var1 var2* ...**

Principe : read affecte chaque mot à un variable, sauf la dernière qui contient le reste de l'entrée.
Pour utiliser d'autre descripteurs, on utilise des redirections.

Exemple :

```
bash-3.2$ read a rt
le chien vert
bash-3.2$ echo $a
le
bash-3.2$ echo $rt 1>&2
chien vert
bash-3.2$
```

En python

Méthodes :

Écriture : **sys.stdout.write(*chaîne*)**

Lecture : **sys.stdin.readline() sys.stdin.read()**

Principe : On peut remplacer stdin par n'importe quel descripteur de fichier.

On peut également utiliser **print » sys.stderr, "chaine"**.

```
>>> import sys
>>> a=sys.stdin.readline()
toto
>>> print >> sys.stderr, a
toto

>>>
```

Et les fichiers ?

Les fichiers se traitent de la même manière, il suffit juste de les ouvrir pour obtenir un descripteur de fichier.

On verra cela par la suite.

Arguments

Les arguments sont séparés par des espaces (cf la variables *IFS*).
L'utilisation des arguments suit souvent un certain nombre de conventions :

- il existe des options avec une seule lettre qui sont introduits par un unique tiret - (options courtes ex : `-l`) ;
- ces options peuvent être combinés (ex : `-ls` est équivalent à `-l -s`) ;
- les options à plusieurs lettres sont introduites par `--` (ex : `--no-auth`) ;
- les arguments "principaux" sont donnés après les options sans tiret ;
- dans le cas où l'un des argument précédent commencerait par un tiret, il est possible d'utiliser l'option `--` seule pour indiquer la fin des options.

Arguments (suite)

Il existe un certain nombre d'arguments classiques :

- `-h`, `--help` : affiche une aide concise ;
- `-v`, `--verbose` : mode "verbose" affiche plus d'information ;
- `-v`, `--version` : affiche la version.

Si il y a une erreur dans les options, le programme retourne en général une erreur ainsi qu'un court message indiquant comment l'utiliser (usage).

```
$ cat -eeret
cat : illegal option -- r
usage : cat [-benstuv] [file ...]
$
```

En shell

Les arguments sont récupérables au travers des variables suivantes :

- **#** contient le nombre d'arguments ;
- **@** contient la liste des arguments ;
- **0** contient le nom du programme ;
- **1** …**9** contiennent les arguments 1 à 9.

S'il y a plus de 9 arguments, il est nécessaire d'utiliser la commande **shift** qui décale les arguments.

Exemple

script :

```
$ cat tutu.sh
#! /bin/sh

echo ”Il y a $# arguments”
echo ”je m’apelle $0 et le premier argument est $1”
shift
echo ”et là c’est $1 le premier”
```

Déroulement :

```
$ ./tutu.sh a b c
Il y a 3 arguments
je m’apelle ./tutu.sh et le premier argument est a
et là c’est b le premier
$
```

Expansion (shell)

Avant de lancer le programme, le shell modifie les arguments en procédant à une **expansion** :

- il les sépare en regardant la variable `IFS` ;
- il remplace les variables par leur valeur ;
- il exécute les commandes entre accent grave (`) ;
- il remplace `~` par la valeur de la variable `HOME` ;
- il remplace les patterns par toutes la valeurs qui correspondent.

Patterns :

- **?** désigne un unique caractère ;
- ***** désigne un ensemble (potentiellement vide) de caractères
- **[** et **]** servent a désigner un caractère parmi l'ensemble de ceux situés entre les crochets.

Entre crochet, il est possible de mettre : des lettres (ex : `[aei]`), des intervalles (`[a-z]`), une négation (`[ !a]`).

Protection (shell)

Protection :

- les guillemets doubles " servent à grouper un ensemble de mots et empêchent l'expansion des patterns ;
- les guillemets simples ' servent à complètement empêcher l'expansion ;
- les accents graves remplacent le contenu par son exécution.

Il est également possible de protéger un symbole en utilisant un backslash (\).

Note : Dans un script, les expansions / protections peuvent devenir extrêmement complexes.

En python

L'ensemble des arguments est disponible par l'intermédiaire du tableau :

sys.argv

Exemple :

```
$ cat tutu
#! /usr/bin/python

import sys
for i in sys.argv :
    print i
$ tutu r t
tutu
r
f
$
```

Analyse des options

Il existe des bibliothèques qui permettent d'effectuer l'analyse des options de façon simple.

- Pour le shell, vous pouvez vous reporter à la commande **getopt**.
- Pour python, ces fonctionnalités sont présentes dans le module **getopt**.

3. Entiers, chaînes de caractères, etc

En python

Voir votre cours de programmation ou un manuel python.

Les entiers

Principe :

- Stockés dans des variables ;
- Pas de type.

Exemples :

```
a=1
b=3
c=4
```

Opérations sur les entiers

Les opérations sur les entiers se font en utilisant les symboles :

$((et **))**

Il est également possible d'utiliser la commande **expr**.

Exemple :

```
$ a=2;b=5;c=6
$ d=$(($a**$b+$c))
$ echo $(($d-1))
37
$
```

Chaîne de caractères

Les chaînes de caractères sont stockés dans des variables. On les délimite par des doubles guillemets (").

Il est possible d'obtenir des caractères de contrôle de la façon suivante :

- `\n` est un passage à la ligne ;
- `\r` est un retour chariot ;
- `\t` est une tabulation ;
- `\num` écrit le caractère ascii dont l'indice est *num*.

Tests

Principe :

Il est possible d'effectuer des tests sur les entiers ou les chaînes de caractères à l'aide de la commande **test** ou, de façon équivalent en utilisant **[** et **]**. Dans ce dernier cas, il ne faut pas oublier les espaces.

Exemples : `[ $A = 1 ]`, `[ $i -le 4 ]`, `test $i -le 4`, ...

Pour voir l'ensemble des comparaisons possibles, reportez-vous à la page de manuel de **test**.

4. Structures de contrôle

Programmation

Pour programmer, il faut disposer de boucles et de structures de contrôles.

Vous connaissez déjà ces opérations en python (if, for, while, …).

On présentera ici les aspects syntaxiques des structures de contrôle en shell.

Enchaînement de commandes

Il est possible d'enchaîner des commandes :

- En utilisant un pipe ;
- En les séparant par un point-virgule ;
- En utilisant **&&** (et) et **||** (ou).

Les deux derniers opérateurs lance la commande suivante si et seulement si la commande précédente à réussi (pour **&&**) ou échoué (pour **||**).

Il est possible de grouper les commandes à l'aide d'accolades **{**, **}**. Il est également possible de le faire avec des parenthèses, nous verrons plus tard la subtilité introduite par ces dernières.

Contrôle de flot en shell

Principes :

- L'indentation n'est pas significative ;
- Tous les mots-clés possèdent une version ouvrante et fermante ;
- les conditions sont juste des commandes dont on regarde la valeur de retour.

If

Exemple :

```
#! /bin/sh

if [ $# -ne 2 ]; then
    echo "Erreur" >&2
    exit -1;
fi;

if [ $1 = $2 ]; then
    echo "égal"
elif [ $1 -le $2 ]; then
    echo "plus petit"
else
    echo "plus grand"
fi
```

For

Principe : Le même que celui du *for in* en python. Pour effectuer un *for* classique, on utilise habituellement le commande **seq** (ou **jot** sous BSD).

Exemple :

```
#! /bin/sh

for i in $@; do
    echo $i
done

for i in `seq 1 10`; do
    echo $i
done
```

While

Exemple :

```
#! /bin/sh

i=0
while [ $i -le $1 ]; do
    echo "$i"
    i=$(($i+1))
done
```

Note : certains autres shells (bash par exemple) fournissent également le mot-clé **until**.

Break et continue

Dans une boucle, il est possible de sortir en utilisant le mot clef **break** et de passer directement à l'itération suivante à l'aide du mot clef **continue**.

En leur ajoutant un entier, on peut parler de la *n*-ième boucle (à éviter).

Case

Exemple :

```
#! /bin/sh

case $1 in
    1) echo ”bien”;;
    toi) echo ”toi”;;
    lui|elle) echo ”alors”;;
esac
```

Et pour finir

Pour tout problème ou pour obtenir des informations supplémentaires, vous pouvez vous référer à la page de manuel de votre shell.

Vous pouvez également consulter, dans tous les cas, la page de man du shell **dash** qui présente l'avantage d'être complète, concise et bien écrite. Elle est disponible en ligne et peut être trouvée par exemple à l'adresse `http://linux.die.net/man/1/dash`.