Guide shell Pour l'Impatient(e)

"bash - GNU Bourne-Again SHell.", /usr/bin/man 2004.
"Le Guide *bash* Pour l'Impatient(e) est un guide de l'UNIX-bricole, pour l'utilisation d'une interface à dispenser sans se disperser.", nj 2004.

Point de départ : un interpréteur de commandes (*shell*), des commandes, des fichiers.
Objectif : exécuter des programmes, lister des fichiers et/ou des répertoires, conditionner des exécutions, créer des listes, rechercher des expressions régulières et plus si affinité.

Le *bash* reprend des concepts du Bourne Shell *sh*, du Korn Shell *ksh*, du C-Shell *csh* et de *tcsh*. *bash* est l'acronyme de Bourne-Again SHell, développé par Stephen Bourne, également auteur de *sh*. Le fichier **/etc/shells** liste les différents interpréteurs supportés par le système. Votre *shell* par défaut est spécifié dans le fichier **/etc/passwd**. Chaque login implique l'ouverture d'un shell login. Un shell login lit les fichiers */etc/profile* notamment pour définir des variables d'environnement et s'ils existent, *~/.bash_profile* ou *~/.bash_login* ou *~/.profile* et *~/.bash_logout* lors d'une fermeture de session. Chaque ouverture d'un terminal sous X11 implique l'ouverture d'un shell non-login. Un shell non-login lit le fichier *~/.bashrc*. Les commentaires sont précédés de `#`.

Lien utile :
- http ://www.gnu.org/software/bash/bash.html
- man bash

`VAL=`cmd1`` ou `VAL=$(cmd1)` : la variable `VAL` reçoit la chaîne affichée (en sortie standard) par la commande `cmd1`. La variable `VAL` s'affiche avec la commande `echo $VAL`.

1- Opérateurs

Les commandes indiquent le résultat de leur exécution par une valeur de retour : une valeur de retour nulle indique un succès ; une valeur de retour différente de zéro indique un échec. Si une commande se termine par un signal numéro N, la valeur de retour est 128+N. Pour une commande non trouvée, la valeur de retour est 127. Pour un fichier trouvé et non exécutable, la valeur de retour est 126.
- `une nouvelle ligne` : indique une nouvelle commande,
- `cmd1 || cmd2` : `cmd2` est exécutée si `cmd1` retourne une valeur non-nulle (faux),
- `cmd1 && cmd2` : `cmd2` est exécutée si `cmd1` retourne 0 (vrai),
- `cmd1 & cmd2` : `cmd1` et `cmd2` sont exécutées simultanéement,
- `cmd1 ; cmd2` : `cmd1` est exécutée, puis `cmd2` est exécutée,
- `cmd1 | cmd2` : l'entrée de `cmd2` est la sortie standard de `cmd1`,
- `cmd1 > fic` : La sortie standard de `cmd1` est redirigée dans `fic` (attention, `fic` est écrasée),
- `cmd1 >> fic` : la sortie standard de `cmd1` est ajoutée à la fin de `fic`,
- `(` : début de shell fils pour exécuter des commandes,
- `)` : fin de shell fils.

2- Structures de contrôle

- `until exp1 ; do cmd1 ; done` : exécute `cmd1` tant que `exp1` retourne une valeur non-nulle,
- `while exp1 ; do cmd1 ; done` : exécute `cmd1` tant que `exp1` retourne 0,
- `for mot in mot1 mot2 ; do cmd1 ; done` : exécute `cmd1` en substituant à la première itération `mot` par `mot1` puis à la deuxième itération `mot` par `mot2`,
- `for (( exp1 ; exp2 ; exp3 )) ; do cmd1 ; done` : exécute `cmd1` à partir de `exp1`, en vérifiant `exp2` et exécutant `exp3`, à chaque itération,

- `if test1 ; then cmd1 ; elif test2 ; then cmd2 ; else cmd3 ; fi ;` : exécute `cmd1` si `test1` retourne 0, sinon évalue `test2` ; exécute `cmd2` si `test2` retourne 0, sinon exécute `cmd3`,
- `case mot in mot1 | mot2 ) cmd1 ; ; mot3 ) cmd3 ; ; *) cmd4 ; ; esac` : exécute `cmd1` si `mot` est égal à `mot1` ou `mot2` et exécute `cmd3` si `mot` est égal à `mot3` et sinon, en dernier ressort, exécute `cmd4`,
- `select mot in mot1 mot2 ; do cmd1 ; done` : exécute `cmd1` en substituant `mot` par `mot1` puis par `mot2`,
- `let` "exp1" ou `((exp1))` : évalue l'expression arithmétique `exp1` (test si `exp1` non nulle),
- `[[ exp1 ]]` : évalue l'expression `exp1` et retourne 0 ou 1 indépendamment de `exp1`,
- `( list )` : exécute la liste des commandes `list` dans un shell fils,
- `{ list }` : exécute la liste des commandes `list` dans le shell courant,
- `func() { cmd1 ; cmd2 ; }` : associe la fonction `func` à la liste des commandes `cmd1`, `cmd2`. Dans un fichier shell comme dans une fonction, les variables `$0` à `$9` sont les arguments, `$#` le nombre d'arguments et `$*` la liste de tous les arguments. `$$` est le pid du script, `$ !` est le pid du dernier processus lancé en arrière-plan et `$ ?` la dernière valeur de retour.

3- Tests

- `/usr/bin/test -f fic` ou `/bin/[-f fic]` : retourne 0 si un fichier nommé `fic` exite (`-b` pour un fichier en mode binaire, `-c` pour un fichier en mode caractère, `-d` pour un répertoire, `-e` pour un fichier, `-f` pour un fichier normal, `-g` pour un fichier avec un group id, `-h` pour un lien symbolique, `-k` pour un fichier avec sticky-bit, `-r` pour un fichier lisible, `-s` pour un fichier de taille non nulle, `-w` pour un fichier modifiable, `-x` pour un fichier éxecutable, `-O` pour un fichier de même id que l'utilisateur, `-G` pour un fichier de même gid que l'utilisateur, `-L` pour un lien symbolique, `-S` pour une socket).
- `[fic1 -nt fic2]` : retourne 0 si `fic1` est plus récent que `fic2` (`-ot` pour plus ancien),
- `[ch1 = ch2]` : compare les chaînes de caractères `ch1` et `ch2` et retourne 0 pour 2 chaînes identiques (`!=` pour deux chaînes différentes),
- `[-n ch1]` : retourne 0 si `ch1` n'est pas une chaîne vide,
- `[-z ch1]` : retourne 0 si `ch1` est une chaîne vide,
- `[exp1 -eq exp2]` : évalue les deux expressions arithmétiques `exp1` et `exp2` et retourne 0 si `exp1` et `exp2` sont égales (`-ne` pour deux expressions différentes, `-ge` pour `exp1` $\geq$ `exp2`, `-le` pour `exp1` $\leq$ `exp2`, `-gt` pour `exp1` $>$ `exp2`, `-lt` pour `exp1` $<$ `exp2`),
- `[ !exp]` : inverse la valeur retournée par `exp`.

4- Substitutions dans une chaîne de caractères

- `${param :-mot}` : si `param` null ou non défini, retourne `mot`, sinon retourne `param`,
- `${param :=mot}` : si `param` null ou non défini, retourne `mot`, sinon retourne 0,
- `${param : ?mot}` : si `param` null ou non défini, affiche `mot`,
- `${param :+mot}` : si `param` null ou non défini, `param` reçoit `mot`,
- `${#param}` : retourne la longueur de `param`,
- `${param#mot}` : élimine la plus courte chaîne correspondant au motif `mot` dans `param` en partant du début (`${param%mot}` en partant de la fin),
- `${param##mot}` : élimine la plus longue chaîne correspondant au motif `mot` dans `param` en partant du début (`${param%%mot}` en partant de la fin),
- `${param/pattern/chaîne}` : substitue le premier motif `pattern` par `chaîne` dans `param`,
- `${param//pattern/chaîne}` : substitue tous les motifs `pattern` par `chaîne` dans `param`.

5- Lecture ligne par ligne d'un fichier

- `cat fic| while read ligne ; do echo $ligne ;done` : redirige chaque ligne du fichier `fic` vers la variable `ligne` pour éxecuter la commande `echo $ligne`
(exemple : `echo ${DISPLAY/:/$HOSTNAME:}`).