IUP GMI D'AVIGNON

www.centraliup.fr.st

Les Systèmes d'Exploitation des Ordinateurs

>2/3

Pascal Nocera

Objectifs

L'objectif est double : connaître les mécanismes internes des **Systèmes d'Exploitation** en général, et appliquer ces connaissances au système **Unix** en particulier, et plus concrètement, étudier l'interpréteur de commandes, la programmation Shell, les filtres, et les processus systèmes. Tout ce qui concerne la théorie des Systèmes d'Exploitation a été étudié dans la partie 1.

Les commandes Unix

Connexion

Pour se connecter au système Unix, il faut entrer son nom, ou login, ainsi que son mot de passe ou password. Ces deux composantes étant nécessaires à la connexion. La commande *logout* ou la combinaison de touches CTRL-D permet de sortir. Pour modifier son mot de passe en cours de session, on peut avoir recours à la commande *passwd*.

Pour la suite des évènements, on travaillera toujours dans une unique **console**, sans se préoccuper outre mesure de la souris ou des possibilités multi-fenêtres d'Unix. Toute ouverture de session se fera donc au sein de la fenêtre console en cours.

Informations générales

Voici quelques commandes d'ordre général :

`cal` ***mois année***	Liste le calendrier.
`date`	Renvoie la date en cours.
`hostname`	Renvoie le nom de la machine.
`banner` ***chaîne***	Affiche le contenu de *chaine* en grosses lettres.
`whereis` ***cmd***	Localise la commande *cmd*.
`who`	Liste le nom des utilisateurs connectés sur le système.
`whoami`	Liste le nom et les informations relatives à l'utilisateur.
`echo` ***arg***	Affiche *arg* à l'écran.
`man` ***cmd***	Affiche des informations concernant *cmd*.

Informations générales

Un chemin d'accès peut être de deux types :

- Chemin **absolu** : On part de la racine (/) jusqu'au répertoire ou au fichier. Exemple : /IUP/IUP2/public/SE.
- Chemin **relatif** : Il est défini par rapport à la position actuelle, c'est-à-dire le répertoire courant. Le répertoire courant est désigné par un point (.), le répertoire père par deux points (..).

Les commandes suivantes agissent ou informent sur le répertoire courant :

`pwd`	Donne le chemin absolu du répertoire courant.
`cd`	Affecte le répertoire courant à celui de l'utilisateur.
`cd` ***path***	Affecte le répertoire courant à *path*.
`cd ..`	Remonte d'une branche dans l'arborescence des répertoires.

Répertoires et fichiers

La commande *ls* permet de lister le contenu d'un répertoire. Elle possède de très nombreuses options permettant d'afficher plus ou moins d'informations et à un certain degré.

et fichiers

`ls -a` Affiche les fichiers précédés d'un point (fichiers cachés) en plus.

`lsf` Utilise les chemins absolus des fichiers (utile en programmation).

`ls -l` Affiche en plus des noms des fichiers leurs caractéristiques.

Un fichier est caractérisé par son **type** (fichier ou répertoire) ; ses autorisations de lecture, écriture et exécution pour l'utilisateur, le groupe et les autres ; le nombre de liens du fichier (eh oui : un même fichier peut avoir plusieurs noms !), les noms du propriétaire et du groupe, sa taille, la date de dernier accès, et bien sûr le nom du fichier.

`cat` ***fich*** Liste le contenu du fichier *fich* d'un seul coup.

`more` ***fich*** Liste *fich* de manière plus diplomatique : page par page.

`lp` ***fich*** Imprime le contenu de *fich*.

Manipulations

Voici quelques commandes de manipulation des fichiers :

`cp` ***fich1 fich2*** Copie le fichier *fich1* dans *fich2*. Si *fich2* existe, il est remplacé par le contenu de *fich1*. Si *fich2* est un répertoire, *fich1* est copié dans le répertoire *fich2* et portera le même nom.

`ln` ***fich1 fich2*** Crée un nom supplémentaire pour *fich1*.

`mv` ***fich1 fich2*** Change le nom de *fich1* en *fich2*. Si *fich1* et *fich2* désignent des répertoires différents, le fichier est déplacé.

`rm` ***fic*** Supprime le fichier *fic*. Attention, pas de corbeille, pas de *undelete*, la suppression est irrécupérable…

`rm –i` ***fic*** Impose en plus une confirmation lors de la suppression.

`rm –r` ***fic*** La suppression est récursive, c'est-à-dire que les répertoires sont supprimés, ainsi que leurs sous-répertoires, etc.

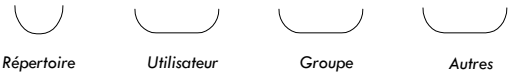
Les manipulations des répertoires mettent en jeu des commandes particulières :

`mkdir` ***rep*** Crée le répertoire *rep*.

`rmdir` ***rep*** Supprime le répertoire *rep*. Il doit être vide.

`mv` ***rep1 rep2*** Renomme le répertoire *rep1* en *rep2*.

Les **droits d'accès** sont décomposés en quatre groupes de bits. Le premier bit indique si le fichier est un répertoire ou non, le groupe suivant de 3 bits mémorise les permissions concernant l'utilisateur (le propriétaire du fichier), puis pour le groupe, et enfin pour tous les autres connectés.



Droits :	Lecture : **r**	Ecriture : **w**	Exécution : **x**
Exemple de commande pour un fichier :	*cat*	*vi*	*com*
Exemple de commande pour un répertoire :	*ls*	*rmdir*	*cd*

Les modifications des droits d'accès des fichiers et des répertoires font l'objet d'une fonction particulière :

`chmod` ***xyz arg*** : Modifie les droits d'accès du fichier *arg* en fonction des indications données : *x*,*y* et *z* sont des octets calculés à partir des 3 bits autorisant l'accès (*ex : chmod 764 donne rwx rw- r--*).

`chmod` ***expr fic*** : Au lieu d'exprimer les bits d'accès directement, on peut entrer une expression, formée de 3 lettres. La 1e indique quel groupe modifier (u,g,o) la 2e définit l'opération à effectuer (+ ajout, - retrait, = affectation) et le 3e indique quel attribut (r,w,x).

Entrées/sorties, pipelines

Unix travaille avec trois fichiers d'entrées/sorties particuliers qui sont constamment ouverts :

- Sortie : *stdout* (écran)
- Entrée : *stdin* (clavier)
- Sortie erreur : *stderr* (écran)

On peut rediriger manuellement l'entrée ou la sortie de la plupart des commandes en utilisant les flots. Par exemple, si on utilise le caractère > suivie d'un nom de fichier, la sortie standard sera envoyée dans le fichier. Si ce fichier existe déjà, son contenu sera écrasé. Pour ajouter plutôt en fin de fichier, il faut utiliser les >>. De la même manière, l'entrée est redirigée par <, et la sortie erreur par 2> et 2>>.

Les fonctionnalités de **pipeline** servent à **chaîner des commandes**. Le caractère | utilisé entre deux commandes permet de rediriger la sortie standard de la première commande sur l'entrée standard de la seconde. Le système assure la gestion de la synchronisation. On peut également ajouter des filtres, c'est-à-dire des outils qui lisent l'entrée et écrivent sur la sortie. Ils sont alors insérés entre les deux commandes, et séparés par deux | (*pipe*). Par exemple :

`ls | wc -l` : Redirige la sortie de *ls* (qui liste le répertoire courant) vers la commande *wc*, qui avec le paramètre *-l*, lit le nombre de lignes. On obtient donc le nombre de fichiers du répertoire.

com1 | ***F*** | ***com2*** : La sortie de *com1* passe dans le filtre *F* avant d'être redirigé vers l'entrée standard de *com2*.

Les interpréteurs de commandes

Un **interpréteur de commande** est un processus lancé au login de l'utilisateur. C'est un langage de programmation à part entière. Il propose trois types de commandes : les commandes définies par des fonctions shell, les commandes internes, et les commandes externes.

Il possède deux types de variables : les variables **internes**, qui ne sont connues que par le shell actuel ; et les variables d'**environnement** qui sont globale au shell en court et à ses fils (mais pas à son père !).

Le Bourne Shell

En début de session, les fichiers */etc/profile* (commun à tous les utilisateurs) et *.profile* (propre à l'utilisateur) sont lancés pour initialiser le **Bourne Shell**. Pour finir une session, une simple commande *exit* suffit.

Les variables internes sont déclarées et utilisées et détruites de la façon suivante :

var=valeur	Crée et initialise une variable interne.
`echo $`***var***	Ecrit le contenu de la variable *var* (attention à ne pas oublier le symbole $).
`set`	Liste toutes les variables actuellement utilisées.
`unset` ***var***	Supprime la variable

Les variables d'environnement ont droit à un traitement légèrement différent :

`set` ***var=valeur*** `export` ***var***	Crée et initialise une variable d'environnement. La commande *export* est nécessaire.
`env`	Liste toutes les variables d'environnement chargées.

Certaines variables sont prédéfinies, dès le lancement du Shell :

`HOME`	Répertoire d'accueil.
`PATH`	Liste des répertoires où se trouvent les commandes.
`CDPATH`	Liste des répertoires pour la commande *cd*.
`PS1`	Valeur du prompt principal ($).
`PS2`	Valeur du second prompt (>).
`USER`	Le login de l'utilisateur actuel.

Les fonctions du Bourne Shell sont déclarées en utilisant la syntaxe suivante :

nom_fonction `()` `{`***commandes ;*** `}`

Toutes les fonctions ainsi crées sont détruites à la fin de la session.

Exemple de fonction pratique à réaliser :

```
dir(){
ls -l $*|more ;}
```

Crée une fonction appelée *dir* qui liste le répertoire ou les fichiers donnés en argument, page par page (grâce à *more*).

Les **commandes internes** sont implantées dans le Shell, et peuvent être masquées par des fonctions utilisateurs portant le même nom :

`cd` ***[rep]***	`pwd`
`echo` ***texte***	`read` ***nom***
`exec` ***cmd***	`times`
`exit` ***[n]***	`type` ***nom***
`newgrp` ***groupe***	

Pour exécuter un script, il faut utiliser la commande suivante :

`sh` ***options script arg1 arg2 ... argN***	On peut utiliser l'option *-x* qui force l'exécution pas à pas.
script arg1 arg2 ... argN	Crée un shell et exécute le script à l'intérieur. Pour cela, il faut que le script soit exécutable (*cf* droits d'accès).
.script arg1 arg2 ... argN	Lance directement le script dans le shell en cours, sans en créer un autre.

Le C-Shell

En début de session, les fichiers *.login* et *.cshrc* sont lancés pour initialiser le **C-Shell**. Le fichier *.cshrc* est exécuté à chaque fois qu'une console est créée. Pour finir une session, on peut taper la commande *exit*, *logout*, ou le raccourci CTRL-D.

Les variables internes sont déclarées et utilisées et détruites de la façon suivante :

`set` ***var***`[=`***valeur***`]`	Crée et initialise une variable interne.
`echo $`***var***	Ecrit le contenu de la variable *var*.
`set`	Liste toutes les variables actuellement utilisées.
`unset` ***var***	Supprime la variable

Les variables d'environnement ont droit à un traitement légèrement différent :

Les variables suivantes sont prédéfinies : *home*, *path*, *cdpath*, *prompt*, *cwd*, et *shell*.

On peut créer un alias en utilisant la syntaxe suivante :

`alias` ***nom_alias*** `'`***commandes***`'`

La chaîne *!** désigne la liste de paramètres d'un alias. Un alias peut être inclus dans un autre alias. Les alias sont valables uniquement dans la session en cours.

Les **commandes internes** sont les mêmes que pour Bourne Shell. On peut ajouter la commande *history* qui donne la liste des dernières commandes tapées. Elle offre les fonctionnalités suivantes :

`!!`	Rappelle la dernière commande.
`!10`	Rappelle la commande n°10.
`!ls`	Rappelle la dernière commande commençant par *ls*.
`alias prev "\!-1 | more"`	Réaffiche la dernière commande, mais page par page…

Le C-Shell offre également la possibilité de compléter automatiquement le nom d'une commande ou d'un fichier en utilisant la touche ESC. Pour cela, il suffit de positionner la variable *filec*.

Le caractère ~ (tilde) représente le contenu de la variable d'environnement *home*, c'est-à-dire le répertoire d'accueil de l'utilisateur. S'il est suivi immédiatement d'un nom d'utilisateur, il désignera alors le répertoire d'accueil de cet utilisateur. Pratique.

Programmation Bourne Shell

Le Bourne Shell offre de véritables possibilités d'un langage de programmation. Il gère deux séparateurs conditionnels, dont le fonctionnement est analogue à ceux du C++, c'est-à-dire qu'ils suivent le principe de la *short-evaluation* (si le résultat de l'opération est effectué dès le calcul de la première opérande, la deuxième n'est même pas évaluée). Ces opérateurs sont le ET logique && et le OU logique ||.

Voici deux exemples d'utilisation de ces opérateurs :

`cd projet && rm *`	Entre dans le répertoire projet et en efface le contenu. Si la commande *cd* échoue, par exemple si le répertoire n'existe pas, il
`(cd projet) || mkdir projet`	Ne crée le répertoire projet que s'il n'existe pas.

Comme tout langage de programmation, le Bourne Shell gère un certain nombre de caractères génériques, c'est-à-dire des caractères qui ont un sens particulier pour l'interpréteur. En voici une liste non exhaustive :

'***chaine***'	Indique que *chaine* ne doit pas être interprétée, mais utilisée littéralement, par exemple dans une commande *echo*.
"***chaine***"	Même utilisation que précédemment, sauf que les caractères $, `, et \ sont interprétés. Mais aucune autre commande !
# ***rem***	Place en commentaire la chaîne *rem*.

Voici un petit exemple d'utilisation des *backquotes* :

`echo date`	Affiche le texte « date », littéralement.
`echo` `` `date` ``	Affiche *la* date…

L'opérateur *shift* permet de parcourir les paramètres d'une commande. Exécuter *shift* revient à faire défiler de droite à gauche touts les arguments d'une occurrence. Le premier argument sera perdu, et le neuvième prendra la valeur du dixième. Cette commande a donc toute son utilité quand on travaille avec plus de neuf arguments.

Ouvrons une parenthèse sur les conventions utilisées pour différencier un script à destination de Bourne Shell ou de C-Shell. Lorsque à la première ligne du fichier, il n'y a pas de commentaire, il s'agit d'un script Bourne Shell. Par contre s'il y a un commentaire, le shell utilisé sera le même que celui dans lequel le script a été lancé. Bien sûr, on peut spécifier le shell à utiliser en entrant à la première ligne la commande cachée *#!* puis indiquer le nom du programme à exécuter. N'oublions pas de refermer cette parenthèse.

Comme tout langage de programmation qui se respecte, le Bourne Shell intègre des structures de contrôle qui tiennent d'une architecture évoluée.

Pour **tester une expression**, on peut avoir recours à la commande *test* :

`test` ***expression*** ou simplement [***expression***]

L'expression en question peut prendre les syntaxes suivantes, suivant le test à effectuer :

`-d` ***nom***	Vrai si le répertoire *nom* existe.
`-f` ***nom***	Vrai si le fichier *nom* existe.
`-s` ***nom***	Vrai si le fichier *nom* existe et n'est pas vide.
`-r` ***nom***	Vrai si le fichier *nom* existe et est accessible en lecture.
`-w` ***nom***	Vrai si le fichier *nom* existe et est accessible en écriture.
`-x` ***nom***	Vrai si le fichier *nom* existe et est exécutable.
`-z` ***chaine***	Vrai si la chaîne *chaine* existe est vide.
`-n` ***chaine***	Vrai si la chaîne chaine n'est pas vide.
c1 **=** ***c2***	Vrai si les chaînes c*1* et c2 sont égales.
c1 **!** **=** ***c2***	Vrai si les chaînes c*1* et c2 sont différentes.
c1 **-** ***operateur*** ***c2***	Vrai si l'expression est vérifiée. L'option *operateur* peut prendre les valeurs suivantes : *eq* (égal), *ne* (différent), *lt* (inférieur strict), *gt* (supérieur strict), *le* (inférieur ou égal), *ge* (supérieur ou égal), etc.

La syntaxe de la structure de test est la suivante :

```
if [expression]
then liste_commandes
else liste_commandes
fi
```

Par exemple :

```
if test –f $1
then echo $1 existe
else echo $1 existe pas
fi
```

L'instruction case propose une série de tests d'égalités consécutifs. Sa syntaxe est la suivante :

```
case chaine in
motif) liste_commandes
motif) liste_commandes
*) liste_commandes
esac
```

Les **boucles itératives** reprennent la même structure que celle des langages dits évolués :

```
while liste
do
    liste_commandes
done
```

```
until liste
do
    liste_commandes
done
```

Et bien sûr, on retrouve notre boucle itérative bornée préférée :

```
for var in chaine1 chaine2 …
do
    liste_commandes
done
```

Le Bourne Shell offre quand même quelques opérations arithmétiques, opérations relativement rudimentaires, car ne travaillant que sur des valeurs entières.

Exemple :

```
x=3
y=2
```

Définition des variables de façon classique.

```
expr $x \* $y
```

Affiche à l'écran le résultat : 6.

```
z=`expr $x \* $y`
echo $z
```

Stocke dans la variable z le résultat de l'opération, puis l'affiche à l'écran.

Les filtres

Le système Unix propose un certain nombre de commandes qui modifient leur entrée en l'envoyant sur leur sortie. Ce sont les **filtres**. Ils peuvent être connectés les uns aux autres par le caractère | (pipe).

Début et fin d'un fichier

Pour obtenir le début et la fin d'un fichier, on peut avoir recourt aux commandes suivantes :

`head –`***n fich*** Récupère les *n* premières lignes du fichier *fich*.

`tail –`***nb fich*** Renvoie les *nb* dernières lignes du fichier *fich*.

`tail +`***nb fich*** Renvoie les dernières lignes du fichier *fich* à partir de la ligne *nb* incluse. On peut ajouter l'option *-r* pour inverser l'ordre.

`cat` ***fich*** `| head –` ***n*** `| tail -1` Voici un exemple de combinaison de deux filtres. Ici, on renvoie la ligne *n* du fichier *fich*.

Commande uniq

La commande *uniq* recherche les lignes uniques ou dupliquées, à noter toutefois qu'elle ne détecte que si de telles lignes sont adjacentes (en gros, il vaut mieux trier le fichier avant de l'envoyer à cette commande !)…

Sans option, la commande *uniq* renvoie le fichier passé en argument en ayant enlevé les répétitions de lignes. Voici les options disponibles :

`-u` N'affiche que les lignes uniques.

`-d` N'affiche que ce qui n'est pas unique.

`-c` Compte les lignes en double.

`+` ***nb*** Saute les *nb* premiers caractères de chaque ligne.

Commandes tr et find

La commande *tr* permet de transformer des caractères spécifiques d'un fichier. L'ajout de l'option *–d* a pour conséquence d'effacer les caractères spécifiés au lieu de les remplacer. Voici quelques exemples de l'utilisation de ce filtre :

`tr "[A-Z]" "[a-z]" < fich` Met en minuscule le fichier *fich*.

`cat fich | tr –d "[0-9]"` Supprime tous les chiffres du fichier.

La commande *find* recherche un fichier selon différentes caractéristiques (taille : *-size*, nom : *-name*), et permet d'exécuter une commande sur chacun des fichiers trouvés, avec l'option *-exec*. On peut ainsi les renommer, les supprimer. Les caractères {} représentent alors le fichier trouvé.

Manipulation de données

Certains filtres permettent de trier et de manipuler des données. Les données sont traitées sous forme de champs. Un champ est défini par un début, qui est soit le début de la ligne, soit la fin du champ précédent ; et par une fin, qui peut être soit la fin de la ligne, soit le premier caractère délimiteur de champ, généralement un espace ou une tabulation.

La syntaxe de la commande de tri est la suivante :

`sort –` ***option*** `[[+`***d***`[-`***f***`]] [`***liste_de_fichiers***`]`

Cette commande prend en compte les principales options suivantes :

`-n`	Tri numérique.
`-r`	Trie dans l'ordre inverse.
`-t`***x***	Choix du séparateur de champ : *x*.
`-f`	Ignore la différence entre minuscules et majuscules.
`+`***d***	Exclut du tri les champs 1 à *d*.
`-`***f***	Exclut les champs suivants *f*.

Extraction de données

La commande *cut* permet l'extraction de données dans un fichier. Elle dispose des options suivantes :

`-d`***x***	Le caractère *x* est considéré comme le séparateur de champ.
`-s`	Supprime les lignes qui n'ont pas de séparateur.
`-c`	Désigne une suite de caractères.
`-f`	Désigne une suite de champs (ex : 1,3 : champs 1 et 3 ; 1-3 : champs 1 à 3 ; -3 : trois premiers champs ; 3- : champs après le n°3 compris).

Fusion et jointure

La commande *paste* fusionne ligne à ligne deux ou plusieurs fichiers. Voici un exemple d'utilisation :

```
cut –f4 fich > fich1
cut –f1,2 fich > fich2
paste fich1 fich2 > fich3
```

Extrait les champs 1,2 et 4 du fichier *fich*, puis les fusionne dans *fich3*, en prenant les fichiers *fich1* et *fich2* comme intermédiaires.

Pour effectuer une jointure plus précise, on peut utiliser la commande *join*, qui met en relation deux fichiers triés selon des champs de jointure. Voici les options disponibles :

`-j` ***m***	Jointure sur les champs *m* des deux fichiers.
`-j1` ***m***	Jointure sur le champ *m* du premier fichier.
`-j2` ***m***	Jointure sur le champ *m* du second fichier.
`-t`***x***	Spécifie que le caractère séparateur de champ est *x*.
`-o` ***n.m***	Liste des champs à conserver dans le nouveau fichier, *n* est le numéro du fichier, et *m* le numéro du champ.

Quelques exemples valent mieux qu'une longue explication…

```
join –j1 3 –j2 2 –o 1.1 1.2 2.1 1.3 fich1 fich2 > fich
```

```
join –j1 4 –j2 2 –o 2.1 employes cdpost | sort | uniq -c
```

Recherche

La commande *grep* met en jeu une fonction de recherche avancée dans un fichier, employant des expressions régulières. Ces expressions sont caractérisées par des caractères spéciaux :

^	Symbolise le début de ligne.
$	Symbolise la fin d'une ligne.
.	Désigne un caractère quelconque.
[*ccc*]	Classe de caractère, c'est-à-dire que ce peut être un caractère parmi ceux proposés entre les crochets.
[^*ccc*]	Complémentaire de la classe de caractère : tous les caractères possible sauf ceux compris entre les crochets.
{***liste_de_nombres***}	Nombre de répétitions de l'expression qui est juste avant. On peut aussi mettre le caractère * pour désigner aucun, + pour désigner 1, et ? pour désigner 0 ou 1.

Voici un petit exemple concret d'utilisation de cette commande impressionnante :

```
grep -c "^[^\:]*\:[\:]*\:[^\:]*\:570\:" /etc/passwd
```

La commande *grep* admet les options suivantes :

-c	Affiche le nombre de lignes.
-i	Ne différencie pas les majuscules des minuscules.
-n	Affiche les numéros de lignes trouvées.
-H	Affiche le nom du fichier.
-v	Affiche les lignes ne contenant pas l'expression.
-f ***fic***	Les expressions à chercher sont lues dans le fichier *fic*.