Environnement de travail

- Variables définies dans l'environnement du shell
  - Liste affichée par la commande **set**
  - Affichage d'une variable : **echo $HOME**
  - Modification : **$HOME=/var/toto**
- Variables principales
  - Répertoire d'acceuil : **HOME**
  - Liste des répertoires explorés : **PATH**
  - Valeur du répertoire courant : **PWD**
  - Prompt : **PS1 PS2** (\h=hostname, \u=utilisateur, \w=répertoire de travail, \W=partie terminale du répertoire de travail, \d=date)
  - Type de terminal : **TERM**
  - Nom de l'utilisateur : **LOGNAME**
- Exportation : **export var** (utilisé par d'autres processus fils)
- Alias : **alias rm='rm -i'**
- Options du shell : **set -o**

Environnement de travail

- Commandes stockées dans .sh_history (ksh) .bash_history (bash)
- Rappel des commandes : mode emacs ou mode vi
- Activation : **set -o vi** ou **set -o emacs**

Commande	Mode vi	Mode emacs
Saisie caractère	Caractère saisi	Caractère saisi
Exécution	RETURN	RETURN
Remonter historique	k (en mode commande)	CTRL p
Descendre historique	j (mode commande)	CTRL n
Déplacement vers la droite	l (mode commande)	CTRL f
Déplacement vers la gauche	h (mode commande)	CTRL b
Supprimer caractère	x (mode commande)	CTRL d
Supprimer ligne	dd (mode commande)	CTRL k
Complétion simple	ESC\ (ksh)	ESC ESC (ksh) TAB (bash)
Complétion liste	ESC= (ksh)	ESC= (ksh) TAB TAB (bash

Environnement de travail

- shell de connexion ou shell ordinaire
- Fichiers de shell de connexion

Shell	Nom du fichier d'environnement
sh ou ksh	.profile
bash	.bash_profile .bash_login .profile

- Contenu des fichiers
  - Définition et exportation de variables transmises à tous les processus
  - Redéfinition de paramètres système
- Ne pas y mettre les options du shell et les définitions d'alias
- .bashrc utilisé pour un shell ordinaire (contient les options et les alias)

Bases de la programmation SHELL

- Interpréteur dès la première ligne du fichier : #!/bin/bash
- Lignes commençant par # ignorées
- Paramètres positionnels : $# $0 $1... $9 ${10} ${11} $* $@
- Décalage : shift [n]
- Code retour : stocké dans $? retourné par exit
- PID du shell : $$
- PID du dernier processus lancé en arrière plan : $!
- Lecture : **read var1 var2**

Bases de la programmation SHELL

- Liste des variables : **set** ou **env**
- Affichage de la valeur d'une variable : **echo $var**
- Modification de la valeur d'une variable : **var=toto**
- Nommer une variable : **var=toto**
  - Premier caractère : [a-zA-Z_]
  - Caractères suivants : [a-zA-Z0-9_]
  - **var= toto** ne marche pas
  - **var=mot1 mot2 mot3** ne prend en compte que mot1
  - **var=”mot1 mot2 mot3”**
- Retirer une définition : **unset var**
- Assignation dans un tableau : **tab[0]=”toto”**
- Assignation globale d'un tableau :

  tab=(10 11 12 13 14) ou **set -A tab 10 11 12 13 14**
- Affichage de tableau : **echo ${tab[0]}** ou **echo ${tab[*]}**

Bases de la programmation SHELL

- Remplacement de paramètre : **${paramètre}**
- Utilisation d'une valeur par défaut : **${paramètre:-mot}**
- Attribution d'une valeur par défaut : **${paramètre:=mot}**
- Extraction de sous-chaine : **${paramètre:début:longueur}**
- Longueur d'un paramètre : **${#paramètre}**
- Suppression d'un fragment :
  - **${paramètre#mot}** retire le plus petit fragment à gauche
  - **${paramètre##mot}** retire le plus grand fragment à gauche
  - **${paramètre%mot}** retire le plus petit fragment à droite
  - **${paramètre%%mot}** retire le plus grand fragment à droite
- Substitution d'un fragment : **${paramètre/motif/chaine}**

Bases de la programmation SHELL

- Substitution de commande :

  echo connection sur `uname -n`
  echo connection $(uname -n)

- Caractères de protection
  - Simples quotes : **echo '*'**
  - Antislash : **echo ***
  - Guillemets : **echo "*"**
- Script shell : fichier texte avec des commandes unix

  bash mon_script

  bash < mon_script

  ./mon_script (si mon_script a des droits d'exécution)

Bases de la programmation SHELL

- Filtrage de caractères

Paramètre	Signification
*	Tout caractère : ls *.c
?	Un seul caractère : ls f?.txt
[...]	Recherche l'un des caractères : ls [0-9] [:class:] avec class = alnum alph ascii blank digit
?(pattern-list)	Cherche zéro ou une occurrence du pattern
*(pattern-list)	Cherche zéro ou plusieurs occurrences du pattern
+(pattern-list)	Cherche une ou plusieurs occurences du pattern
@(pattern-list)	Cherche l'un des patterns donnés
!(pattern-list)	Cherche tout sauf l'un des patterns

Bases de la programmation SHELL

- Commande test : **test** *expression* ou **[** *expression* **]**

Commande	**Description**
-a fichier	Vrai si le fichier existe.
-b fichier	Vrai si le fichier existe et est un fichier spécial bloc.
-c fichier	Vrai si le fichier existe et est un fichier spécial caractère.
-d fichier	Vrai si le fichier existe et est un répertoire
-e fichier	Vrai si le fichier existe.
-f fichier	Vrai si le fichier existe et est un fichier régulier.
-h fichier	Vrai si le fichier existe et est un lien symbolique.
-r fichier	Vrai si le fichier existe et est accessible en lecture.
-s fichier	Vrai si le fichier existe et a une taille non nulle.
-w fichier	Vrai si le fichier existe et est accessible en écriture.
-x fichier	Vrai si le fichier existe et est exécutable.
-s fichier	Vrai si le fichier existe et a une taille non nulle.
-O fichier	Vrai si le fichier existe et appartient à l'ID effectif de l'utilisateur.

Bases de la programmation SHELL

- Tests sur des chaînes

  utilisent des expressions conditionnelles

Commande	**Description**
-z chaîne	Vrai si la longueur de la chaîne est nulle.
-n chaîne chaîne	Vrai si la longueur de la chaîne est non-nulle.
chaîne1 == chaîne2	Vrai si les deux chaînes sont égales.
chaîne1 != chaîne2	Vrai si les deux chaînes sont différentes.
chaîne1 < chaîne2	Vrai si chaîne1 se trouve avant chaîne2 dans l'ordre lexicographique de la localisation en cours.
chaîne1 > chaîne2	Vrai si chaîne1 se trouve après chaîne2 dans l'ordre lexicographique de la localisation en cours.

Bases de la programmation SHELL

- Tests sur les nombres :

Commande	**Code retour**
nb1 -eq nb2	Vrai si **nb1** est égal à **nb2**
nb1 -ne nb2	Vrai si **nb1** est différent de **nb2**
nb1 -lt nb2	Vrai si **nb1** est strictement à **nb2**
nb1 -le nb2	Vrai si **nb1** est inférieur ou égal à **nb2**
nb1 -gt nb2	Vrai si **nb1** est strictement supérieur à **nb2**
nb1 -ge nb2	Vrai si **nb1** est supérieur ou égal à **nb2**

- Opérateurs :

Opérateurs par ordre de priorité	**Signification**
!	négation
-a	ET
-o	OU

- Priorité modifiable : **[-w $f1 -a \(-e $f2 -o -e $f3\)]**

Bases de la programmation SHELL

- Arithmétique : expr nb1 op nb2

Opérateur	Signification
nb1 + nb2	Addition
nb1 - nb2	Soustraction
nb1 * nb2	Multiplication
nb1 / nb2	Division
nb1 % nb2	Modulo
nb1 \> nb2	Vrai si nb1 est strictement supérieur à nb2
nb1 \>= nb2	Vrai si nb1 est supérieur ou égal à nb2
nb1 \< nb2	Vrai si nb1 est strictement inférieur à nb2
nb1 \<= nb2	Vrai si nb1 est inférieur ou égal à nb2
nb1 = nb2	Vrai si nb1 est égal à nb2
nb1 != nb2	Vrai si nb1 est différent de nb2
-nb1	Opposé de nb1
\& |	Opérateurs logiques

Commandes composées

- Commande [[*expression*]]

  Utilisation d'expressions conditionnelles
  espaces obligatoires

Opérateurs par ordre de priorité	**Signification**
(*expression*)	Négation
! *expression*	VRAI si expression est FAUX
expression1 && expression2	VRAI si expression1 et expression2 sont VRAI
expression1 || expression2	VRAI si expression1 ou expression2 sont VRAI
chaîne = modele	VRAI si la chaine correspond au modèle
chaîne != modele	VRAI si chaine ne correspond pas au modèle
chaine1 < chaine2	VRAI si chaine1 est lexicographiquement avant chaine2
chaine1 > chaine2	VRAI si chaine1 est lexicographiquement après chaine2

Commandes composées

- Commande [[*expression*]]

Caractères spéciaux	Signification
*	0 à n caractères
?	1 caractère quelconque
[abc]	1 caractère parmi ceux cités en entre les crochets
![abc]	1 caractère ne faisant par partie des caractères cités en entre les crochets
Option extglob à activer sous Bourne Shell	
?(expression)	0 ou 1 fois l'expression
*(expression)	0 ou n fois l'expression
+(expression)	1 à n fois l'expression
@(expression)	1 fois l'expression
!(expression)	0 fois l'expression

Commandes composées

- Commande **((** *expression* **))**

  Évaluation arithmétique
  Les arguments n'ont pas besoin d'être séparés par des espaces
  Les variables n'ont pas besoin d'être préfixées par $
  Les caractères spéciaux du shell n'ont pas besoin d'être protégés
  opérateurs :

  - id++ id--
  - --id ++id
  - - +
  - ! ~
  - **
  - * / %
  - << >>
  - <= >= <>
  - == !=
  - &
  - ^
  - |
  - &&
  - ||
  - expr?expr:expr
  - = *= /= %= += -= <<= >>= &= ^= -=
  - expr1, expr2

  Exemple :
  $ a=1
  $ ((a+=2))

  Équivaut à **let** *expression*

Commandes composées

- **for** *nom* [**in** *mot*]; **do** *list* ; **done**
- **for** ((*expr1* ; *expr2*; *expr3*)); **do** *list* ; **done**
- **select** *name* [**in** *word*] **; do** *list* **; done**
- **if** *list*; **then** *list*; [**elif** *list*; **then;**] **…** [**else** *list*;] **fi**
- **while** *list*; **do** *list*; **done**
- **until** *list*; **do** *list*; **done**
- Fonctions shell : [function] nom() commandes [redirection]
  - mafct() { commande1; commande2; }
  - function mafct() { commande1; commande2; }
  - Variables locales dans la fonction : typeset variable=valeur
  - Arguments : $1 $2 $3 $4 ...

SED (Stream Editor)

- Editeur non interactif :
  - **sed 'action' fichier**
  - **sed -e 'action' fichier**
  - **sed -f fichier_actions fichier**
- Syntaxe d'une action : [adresse[,adresse]]commande[arguments]
  L'adresse peut être une ligne ou une expression régulière
- Commandes principales :
  - **sed ”s/toto/TOTO/” fichier**
  - **sed ”2,4d” fichier**
  - **sed ”/toto/p” fichier**
  - **sed ”/toto/l” fichier**
  - **sed ”/toto=” fichier**
  - **sed ”/^toto/w resultat” fichier**
- Autres commandes
  - Ajout de ligne : a\
  - Insertion de ligne : i\
  - Remplacement de ligne : c\
  - Négation de commande : !commande

AWK

- Syntaxe :

  awk [-F] [-v var=valeur] 'action' [fic1 … ficn]

  awk [-F] [-v var=valeur] -f fichier_config [fic1 … ficn]

- Variables spéciales :
  - Champs : $1 $2 $3 …
  - Nombre total de champs : NF
  - Nombre de champs lus : NR
  - Fichier courant : FILENAME
  - Séparateur de champ en entrée : FS
  - Séparateur des enregistrements en entrée : RS
  - Séparateur de champ en sortie : OFS
  - Séparateur des enregistrements en sortie : ORS
  - Description d'une erreur : ERRNO
  - Séparateur d'indiçage : SUBSEP
- Exemple : **df -h | awk {print NR ” device ”$1,”\toccupation ” $5}**

AWK

- Critères de sélection
  - Syntaxe : **awk [-F] 'critère {action}' [fic1 fic2 … ficn]**
  - Expressions régulières : **awk '/^Sys/ {print $1, $5}'**
  - Tests logiques : **awk '!/^Sys/ && !/^none/ {print $1, $5}'**
  - Intervalles de lignes : **awk 'NR==2,NR==$NF {print $1, $5}'**
- Tableaux
  - Unidimensionnels : tab[i]
  - Associatifs : couleur[”raisin”]
  - Multidimensionnels en définissant le séparateur d'indiçage : SUBSEP=”:”

AWK

- Structures d'un script AWK
  - BEGIN { } : exécutée avant tout traitement
  - Sections intermédiaires : exécutées sur chaque enregistrement
  - END { } : exécutée après tous les traitements
- Les sections régulières peuvent comprendre des tests
  - Exemple : **$1 !~ /^Sys/ { print $1}**
- Fonction printf : printf(”format,expr1,expr2,...,exprn)
  - Exemple : **printf(”%2s %3d %10.2f”,ex1,ex2,ex3);**
- Lecture de la ligne suivante : **next**

AWK

- Structures de contrôle

  if (condition) { instruction } else { instruction }

  for (initialisation ; condition ; incrementation) { instruction }

  for (; condition ;) { instruction incrementation }

  for (cle in tableau) { instruction }

  while (condition) { instruction }

  do { instruction } while (condition)

- Mots clés
  - Interrumption de boucle : **break**
  - Exécution de la suite de la boucle : **continue**

AWK

Opérateurs arithmétiques

+	addition
-	soustraction
*	multiplication
/	division
%	modulo
^	exponentiation
++	Incrémentation d'une unité
--	Décrémentation d'une unité
+=	x+=y équivaut à x=x+y
-=	x-=y équivaut à x=x-y
*=	x*=y équivaut à x=x*y
/=	x/=y équivaut à x=x/y
%=	x%=y équivaut à x=x%y
^=	x^=y équivaut à x=x^y

Opérateurs de tests

<	inférieur
>	supérieur
==	égalité
!=	Test d'inégalité
~	Correspondance avec une ER
!~	Non correspondance avec un ER

Opérateurs logiques

!	négation
&&	ET logique
||	OU logique
=	Affectation
e1 ? e2 : e3	vaut e2 si e1 est vrai
e1 e2	Concaténation

AWK

- Fonctions intégrées travaillant sur les chaînes

gsub(er,remp,[ch])	Remplace dans ch les occurences correspondant à er par remp
index(ch1,ch2)	Retourne la position de la chaine ch2 dans ch1
length(ch)	Retourne la longueur de la chaîne ch
match(ch,er)	Retourne la position dans ch de la position de la 1er occurence de l'expression régulière er
split(ch,tab,sep)	Initialise le tableau tab avec les champs de la chaîne ch séparés par sep
sprintf(fmt,e1,..,en)	Retourne une chaine formatée
sub(er,remp,[ch])	Remplace dans ch la 1° occurence de la regex er par remp
substr(ch,pos,lg)	Retourne la sous-chaine de ch commençant à la position pos et de longueur lg
tolower(ch)	Retourne la chaine ”ch” convertie en minuscule
toupper(ch)	Retourne la chaine ”ch” convertie en majuscule