Le Shell

Un programme C non intégré au noyau

- Interpréteur de commandes
- Langage de programmation
- Permet d'étendre les commandes du système d'exploitation

Nombreuses versions

- sh shell Bourne (disponible sur toutes les machines)
- csh Berkeley (Bill Joy)
- rsh exécution à distance
- tcsh, ksh, bash, vsh,…

Caractéristiques de l'interprète

- Analyse et interprète des lignes de commandes
- Création et lancement des processus associés aux commandes (contrôle de processus)
- Distinction entre processus bloquant et tâche de fond
- Redirection des E/S
- Mécanisme de substitution de chaînes de caractères
- Accès aux ressources de la machine : le processeur (implicitement), le(s) disque(s), imprimantes,…

  via des **commandes.**
- Affiche un prompt pour signifier qu'il est prêt à interpréter une commande
- Permet d'étendre les commandes du système l'exploitation
- Propose un véritable langage de programmation
- Nombreuses versions (sh, csh, ksh, bash,…)

Structure de la boucle d'interprétation

- Le shell est un programme dont l'algo pourrait être :

Tant que vrai

  - Lire une ligne
  - Identifier la commande (vérifier la syntaxe et sémantique)
  - Création d'un processus exécutant le fichier associé à la commande
  - Attente de la fin d'exécution de ce processus

Fin Tant que

Langage shell : généralités

- Variables utilisateurs (chaînes de caractères)
  - Contenu obtenu en préfixant $ à la variable
  - Affectation par l'opérateur = (autrement selon le shell)

  Exemple : v=bonjour

  echo $v

  Afficher bonjour
- Variables prédéfinies
  - Affectées par le shell : # ? $! –
  - Affectées avant l'exécution : HOME PATH MAIL TERM PS1 PS2 IFS (variables de configuration)
  - Macroprocesseur de substitution
  - Commandes internes break continue cd eval exit export login newgrp read shift set wait unmask trap
- Structure de contrôles : if-then-else for-do-done while-do-done case-in-esac

Variables d'environnement prédéfinies

- **HOME** chemin d'accès au répertoire initial de l'utilisateur
- **PATH** suite de chemins d'accès aux répertoires des exécutables
- **PS1** invite principale du shell en mode interpréteur
- **PS2** invite secondaire du shell en mode programmation
- **IFS** séparateurs de champ des arguments
- **MAIL** chemin d'accès à la boite aux lettres utilisateur
- **MAILCHECK** intervalle en sec au bout duquel le mail est contrôlé
- **DISPLAY** nom de l'écran d'affichage
- **TERM** nom du type de terminal

Quotage

Rôle

le quotage est utilisé pour supprimer les fonctionnalités de certains caractères spéciaux du shell, dont les métacaractères.

Méta-caractères: \ $ * ` " '

Caractère d'échappement: \

le caractère \ préserve la valeur littérale du caractère qui le suit

Simples quotes '...'

les caractères inclus entre 2 simples quotes ne sont pas évalués

Doubles quotes "..."

les caractères inclus entre 2 doubles quotes conservent leur valeur à l'exception de $ ` et \.

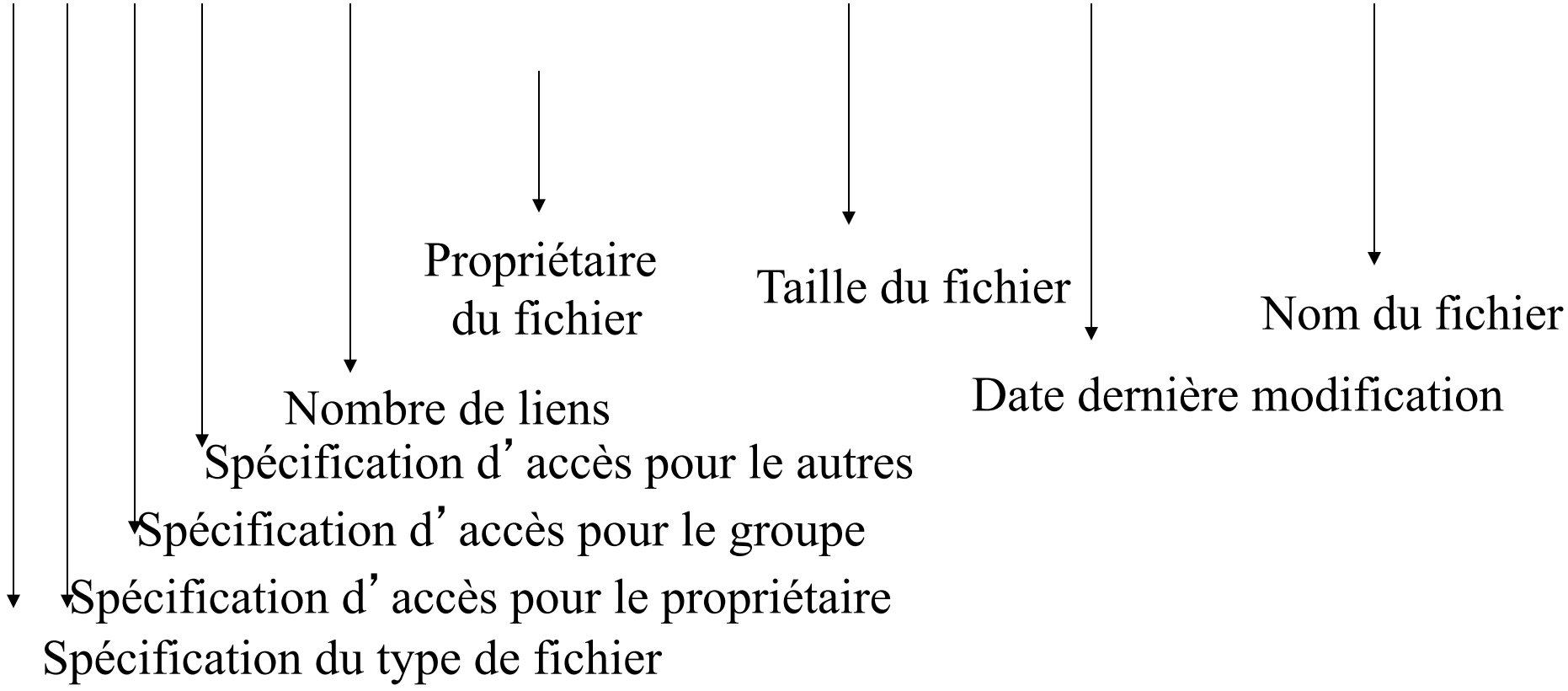
Exemple

```
TOTO='bonjour'
echo '$TOTO et ' "$TOTO"
$TOTO et bonjour
```

Lister le contenu d'un répertoire

- La commande ls –l liste les fichiers en donnant des informations.
- Exemple : Si on est dans gesinfo, ls –l donne :

```
total 2
-rwx r-x r-x   1   etud1   51   Oct 1 12:00   planing
drwx r-x r-x   2   etud1   512  Oct 1 12:00   src
```



Commandes de manipultion de fichier

- basename permet d'extraire le nom le plus relatif d'un fichier : $ basename <nom de fichier> [suffixe]
- Exemples :
  - $ basename /users/ens/moncompte → moncompte
  - $ basename moncompte/src/pgme.c .c → pgme
  - $ basename moncompte/src/pgme.c e.c → pgm

Pas de vérification de l'existence des fichiers

La commande dirname

- Extrait le chemin du fichier
- Exemple :
  - dirname /etc/passwd → /etc

Commandes de surveillance

- who, w, ps
- nice, renice, kill
- vmstat : statistiques sur la mémoire virtuelle
- swapon, swapoff gestion espace pour la pagination
- du espace sur les répertoires
- df espace sur le système de fichiers

Commandes et flux

- Tubes nommés
- Redidections
- Arguments de commandes et entrées de commandes

- Toute commande Unix, dès son lancement, possède 3 fichiers ouverts :
  - L'entrée standard (stdin) ou 0
  - La sortie standard (stdout) ou 1
  - La sortie erreur (stderr) ou 2
- Dans le shell sh, tout fichier ouvert par un programme est représenté par &X, où X est un numéro qui s'incrémente à chaque ouverture de fichier.
  - stdin, stdout et stderr sont respectivement pour toute commande lancée dans le shell &0, &1 et &2.

Exemples :

- Rediriger la sortie :
  - $ ls –l 1>listeFich.txt
- Rediriger les deux flux dans un même fichier (en tapant une commande erronée)
  - $ ls –l >>erreur 2>&1

- Exemple 3 : compter le nombre d'utilisateurs connectés
  - $ who | wc –l

    6
  - $ who | wc –l >temp

    $ cat temp

    6

- $ ls ; who >temp

  bin

  users

  tmp

- $ cat temp

  u1 tty05 Oct 8 08:35

  u2 tty05 Oct 8 09:55

Seule la sortie de la dernière commande est redirigée vers le fichier temp

- $(ls;who)>temp; cat temp

  bin

  users

  tmp

  u1 tty05 Oct 8 08:35

  u2 tty05 Oct 8 09:55

Exécution des commandes

Différents modes d'exécution des commandes

1) Interactif (foreground)
2) Asynchrone (arrière plan, background)
3) Différé
4) cyclique

1) Interactif (foreground)

- Mode dans lequel le système rend de façon interactive le résultat de la commande (vue précédement)

Asynchrone (arrière plan, background)

- <commande> &
- nohup <commande> &
- les touches ctrl (contrôle) et z (simultanément), puis suivies de la commande bg permet de lancer une commande en background
- nohup évite que les processus lancés en backgroud ne soit arrêtés lors du logout. Envoi résultat par mail à l'utilisateur.

- Lancement de plusieurs commandes sans attendre que chacune d'entre elles se termine
- cat essai1 & cat essai2

… numéro du processus lancé par le Shell

- $ cc pgme.c &

124569

On récupère le promt shell avant que la compilation ne soit terminée

- $ (echo « Bonjour »;sleep 15; echo « Bye ») &

Différé

- at <un temps> <commande>

Démon at

Commandes atq et atrm

Fichiers de configuration at.allow et at.deny (voir plus loin fonctionnement similaire à ceux de cron)

Fichers : /var/spool/cron/atspool

/var/spool/cron/at.jobs

Exemple : at 8:15 May 24 < fichCmde

Exécution cyclique

Démon cron

Commande crontab

Pour un utilisateur quelconque :

/var/spool/cron/crontabs/<nom utilisateur>

Pour le super utilisateur :

/var/spool/cron/crontabs/root

Format d'une ligne de ce fichier :

Minute heure jours_mois mois jours-semaine commande

Où :

- Minute : 1 à 60
- heure : 1 à 60
- jours_mois : 1 à 31
- mois : 1 à 12
- jours-semaine : 0 à 6 (0=dimanche)
- Commande : commande à exécuter de façon cyclique.

Contrôle du lancement de commandes cyclique

Fichiers de configuration associés :

- /etc/cron.d/cron.allow
- /etc/cron.d /cron.deny

- Accès autorisé à un utilisateur si :

  /etc/cron.d/cron.allow existe et le user n'y est pas

  Ou bien :

  /etc/cron/cron.allow n'existe pas et si le nom user n'apparait pas dans cron.deny
- Acces refuse à un user si :

  /etc/cron.d/cron.allow existe et le user n'y est pas

  Ou bien

  /etc/cron/cron.deny existe et l'utilisateur est dedans et cron.allow n'existe pas
- Si aucun des deux fichiers n'existe, seuls les utilisateurs ayant les autorisation solaris.jobs.users peuvent soumetre des jobs

- Pour root, les règles s'appliquent si les 2 fichiers /etc/cron/cron.allow et /etc/cron/cron. deny existent

Quelques spécificités syntaxiques des différents shell

sh

- Caractères spéciaux: * ? [akl] [a-g] [!A-Z]
- # $ & ; < << > >> | ‘ « ` / () {}
- Passage d’arguments :

$0 $1… $9

$# nombre arguments (sauf $0)

$$ pid du shell qui exécute

$* chaine des arguments (sauf $0)

$@ liste des arguments (sauf $0)

$? Code retour de la dernière commande

$! Pid du dernier processus asynchrone

sh (suite)

- Variables nomDeVariable=valeur

  Echo $ nomDeVariable

- Variables d'environnement
  - Créées au login (HOME, USER…)
  - Générales (MAIL, CLASSPATH, PATH,…°
- Le problème d'exportation de variables

```
echo $TOTO
TOTO='bonjour je suis sh'
echo $TOTO
bonjour je suis sh
sh
   echo $TOTO          (TOTO n'a pas été exporte)

  Ctrl+D
export TOTO
sh
   echo $TOTO          (TOTO a été exporté)
   bonjour je suis sh
```

Structure de contrôle

- Condition : c'est toujours le code retour d'une commande (0 pour vrai, faux sinon)

Test expression où expression vaut :

- -f nom
- -d nom
- -r (xw) nom
- -s nom (taille)
- -z chaine (chaine vide)
- chaine1 = chaine2 chaine1 != chaine2 (égalité entre chaines)
- Nombre1 –eq nombre2 (-ne –gt –ge –lt –le)

– Connecteurs ! –a -o

expr

- Permet l'arithmétique entière sur les variables.

+ - * / %

  A = `expr $A + 1`

- Permet la comparaison de chaînes de caractères (le résultat est le nombre de caractères reconnus au début)

 expr « 1234trfhj » : '[0-9]*' donne 4

- Permet l'extraction de chaînes

  expr « toto.c » : « \(.*\).c donne toto

- Conditionnelle

```
if cond1
then
Cmde…
elif cond2
 then
Cmde…
…
 else
Cmde…
 fi
```

- Conditionnelle multiple

```
case nomVariable in
    chaine1) cmde…;;
    chaine2) cmde…;;
    …
    esac
```

- Boucles

```
for nomVar in liste
do
Cmde…
Done

while cond
do
Cmde…
done

until cond
do
Cmde…
done
```

Sortie exceptionnelle par break(n) continue exit(n)

- Liste de commande :

  {commande…} && {commande} si code retour est 0

  {commande…} || {commande} si code retour n'est pas 0

- Fonction :

nomFonction() {commande…}

Peut renvoyer un code retour avec return(n)

Entrées/sorties

- read listeNomVariables

Lit sur l'entrée standard. Les chaînes sont séparées par des espaces ou des tab

- echo chaine

\c pas de passage à la ligne

\b backspace

\n newline,…

Compléments sur les variables

- ${nomVariable}chaine
- ${nomVariable:-chaine} la substitution se fait même si la variable est nulle ou non initialisée
- ${nomVariable:+chaine} la substitution se fait si la variable est initialisée, sinon substitution de la chaine vide

csh

- Ce qui change :

  Fichier de configuration ~/.cshrc

  Définition des variables :

  set nomVariable = valeur (unset)

  setenv nomVariable valeur (unsetenv)

  Structures de contrôle

  if (expression) then

  commande…

  elsif (expression)

  commande…

  else

  commande…

  endif

csh

- switch (chaine)
  case motif:
    commande…
    [breaksw]
    …
    default:
      commande…
  endsw

csh (suite)

- foreach nomVariable (listeValeurs)
  Commande…
  end
  while (expression)
  commande
  end

  repeat nbFois commande

  goto etiquette
  etiquette:

csh (suite)

- Les paramètres positionnels ($0 $1… $9)
- argv[] accepté
- Les entrées : le read est remplacé par :

  set nomVariable = $<
- Expression dans if while … :

  + - * / % ++ -- <= >= < > == != =~ !~
- & ^ | (opération bit à bit)
- ! && ||
- if {commande} permet de tester le code retour

Définition d'une fonction

- function <nom>(<liste de paramètres>)

  {

  [return [<expression>]]

  <instructions>

  }

<liste de paramètres> =séquence de variables séparées par des virgules

Les sorties

- print
- print <expression1>, <expression2>,…
- print <expression1>, <expression2>,…**>** <fichier>
- print <expression1>, <expression2>,…**>>** <fichier>
- print <expression1>, <expression2>,…| <commande>
- printf (<format>,<expression1>, <expression2>,…)> <nomfichier>
- printf (<format>,< expression1>, <expression2**>,…)>>** <fichier>
- printf (<format>,< expression1>, <expression2>,…) | <commande>
- close(<fichier>), cloture de la connexion entre printf et <fichier
- close(<commande>) fermeture des pipes
- system (<commande>) exécute une commande et retourne la valeur retournée par la commande <commande>

La fonction getline, pour lire des enregistrement en entrée

- getline lit sur l'entrée standard et remplit les variables $0, $1, $2, $3,…, NR, NF et FNR
- getline <nom de variable> lit sur l'entrée standard et met le met dans une variable
- getline <nom de fichier> lit l'enregistrement suivant dans le fichier et le met dans $0 en mettant à jour NR
- getline <nom de variable> < <nom de fichier> lit le prochain enregistrement du fichier et le met dans une variable
- <commande> | getline lit la sortie de commande et le place dans $0 en mettant à jour NF
- <commande> | getline <nom de variable> lit la sortie de commande et le place dans une variable