Plan du Cours

1. Présentation Générale

- Qu'est-ce qu'un ILC (shell) ?
- Algorithme d'un Shell
- Bourne shell et dérivés
- Scripts de configuration

Qu'est-ce qu'un ILC ?

- Un programme comme un autre
  - Ecrit en C
  - Capable de lire/écrire sur un terminal (ou une fenêtre graphique émulant un terminal)
  - Sait lancer des processus
    - Exécution de commandes « externes »
  - En général le premier processus d'un utilisateur
    - Voir fichier passwd
    - Processus –zsh lors d'un rlogin
  - Fournit un langage de programmation interprété
    - Fichiers de commande

Algorithme d'un Shell

```
Init : interprétation des fichiers de configuration
TANT QUE (entrée != {eof, exit, logout})
FAIRE
        Saisir commande (edition « à la emacs », complétion, …)
        Analyse lexicale et syntaxique
        Expansion des paramètres
        CAS de $0 dans
                fonction, cmde interne) : l'exécuter
                fichier binaire dans PATH) :
                        créer process ayant $O comme prog.
                fichier texte) : lancer l'interpréteur
                        (#!…) ou sinon /bin/sh + texte
        FIN CAS
        SI ('&' n'est pas en fin de chaîne)
        ALORS
                attendre fin d'exécution du processus lancé
        FIN SI
FIN TANT QUE
```


Bourne Shell et Dérivés

- Bourne Shell (/bin/sh)
  - Le plus « classique »
  - Utilisé pour les scripts système
- Korn Shell (/bin/ksh)
  - 1ère évolution
  - nombreuses améliorations, mais encore des défauts
- Bourne Again Shell (/bin/bash)
  - Le shell GNU/FSF (shell « officiel de linux »)
  - Récupère nombreuses améliorations d'autres shell (ksh, tcsh), ajoute ses propres améliorations
- Zsh
  - Le plus complet (idem bash/ksh, plus améliorations propres)
  - Ambitieux : tout ce dont les utilisateurs pourraient avoir besoin doit être disponible dans zsh …

Scripts Personnels de Configuration

- Bourne Shell
  - .profile
- ksh
  - .profile, .kshrc
- bash
  - .profile, .bash_profile, .bashrc, .bash_login
- zsh
  - .zprofile, .zshrc, .zlogin, .zlogout, .zshenv

Scripts Personnels de Configuration Le cas de ZSH

- Les scripts sont recherchés dans $ZDOTDIR
  - $HOME par défaut (ZDOTDIR non définie)
- Plusieurs contextes sont considérés
  - shell non interactif
    - .zhsenv
  - shell interactif
    - .zshenv, .zshrc
  - shell interactif de login
    - .zshenv, .zprofile, .zshrc, .zlogin
    - Lors de la terminaison : .zlogout

Scripts de Configuration Systèmes de Zsh

- Pour chacun des scripts personnels de configuration, il existe un équivalent système
  - `/etc/zshenv, /etc/zprofile, /etc/zshrc, /etc/zlogin`
  - Maintenus par root
  - Chacun des scripts de configuration « système » est appelé juste avant son équivalent « utilisateur »
    - la séquence des exécution pour un shell de login est donc la suivante : /etc/zshenv, .zshenv, /etc/zprofile, .zprofile, /etc/zshrc, .zshrc, /etc/zlogin, .zlogin
    - /etc/zlogout avant .zlogout
    - Le script /etc/zprofile est supposé exécuter le script /etc/profile commun à tous les « Bourne Shell »
    - /etc/profile peut lui même exécuter d'autres scripts, par exemple /etc/profile.d/*.sh sous linux …

Scripts Personnels de Configuration de Zsh

- .zshenv
  - Toujours lu => bon endroit pour les variables d'environnement
- .zprofile
  - Lu en premier (apres zshenv) avec un shell de login : le bon endroit pour initialiser une session
- .zshrc
  - Lu quand le shell est interactif => tout ce qui facilite la vie de l'utilisateur interactif
    - alias (raccourcis), complétion, prompt, …
- .zlogin
  - Lu en dernier, lors du login : le bon endroit pour poser des questions si besoin (config DISPLAY, …)

2. Utilisation *interactive* avancée de zsh

- Mécanismes de redirection
- Mécanismes de substitution
  - Historique
  - Commandes
  - Processus
  - File globbing

Mécanismes de Redirection

- Rappel des principes :
  - Chaque processus possède 10 descripteurs de fichiers, numérotés 0 à 9
  - Chaque descripteur indique un fichier
  - 3 descripteurs actifs par défaut :
    - 0 désigne l'entrée standard du processus
    - 1 désigne la sortie standard
    - 2 désigne la sortie standard d'erreur
    - Avec un shell interactif, les 3descripteurs par défaut sont connectés à la (pseudo-)console (/dev/ttyxx)
  - Une redirection permet d'établir ou modifier une liaison entre un descripteur et un fichier

Types de Redirection : Redirection Elémentaires

- < FIC
  - Ouvre le fichier FIC et le place sur stdin
- <> FIC
  - Ouvre le fichier FIC en lecture/ecriture et le place sur stdin
- > FIC
  - Ouvre le fichier FIC et le connecte à la sortie standard
- >> FIC
  - Ouvre le fichier FIC an mode "APPEND" et le connecte sur la sortie standard

Types de Redirections : Redirection "Here Document"

- <<MOT, <<-MOT
  - Les lignes suivantes sont placées sur l'entrée standard jusqu'à l'apparition de MOT
  - Si MOT contient des caractères "protégés" par ', ", ou \, les lignes suivantes (jusqu'à apparition de MOT) ne subissent pas de substitution
  - forme <<- : les tabulations en début de ligne sont ignorées

Types de Redirections : Duplication et Fermeture de Descripteurs

- <&DIGIT
  - L'entrée standard est dupliquée à partir du descripteur DIGIT
- >&DIGIT
  - La sortie standard est dupliquée vers le descripteur DIGIT
- <&-
  - Fermeture de stdin
- >&-
  - Fermeture de stdout

Types de Redirections : Redirections "exotiques"

- Coprocessus
  - Un coprocessus est un processus "binôme" lancé par le processus courant
  - >&p, <&p
    - permettent de brancher (les descripteurs de) le processus courant sur le coprocessus
- <<<WORD
  - WORD subit une passe de substitution, puis le résultat est envoyé sur l'entrée standard

Opérations sur les Descripteurs

- Redirection d'autres descripteurs que stdin et stdout
  - Il suffit de mettre le numéro du descripteur juste avant la redirection
  - Exemple : date >&3 3>toto
    - la sortie std de date est envoyée sur desc num 3, puis desc num 3 est redirigé vers fichier toto
- Associer un fichier à un descripteur
  - exec DIGIT<FIC, exec DIGIT>FIC
  - exec DIGIT&<DIGIT, exec DIGIT>&DIGIT, …
- Entrées/Sorties vers ou depuis un descripteur
  - read [-u DIGIT] variable
  - echo/print [-u DIGIT] chaine

Exemples de Redirections

- find / 2>&1
  - Les messages d'erreur sont redirigés vers la sortie standard (en plus des messages normaux)
- exec 3> toto
  - Le descripteur 3 est redirigé en écriture sur le fichier toto : tout ce qui sera écrit sur le decripteur 3 ira dans le fichier toto
- ls –al >&3
  - Le résultat de 'ls –al' est envoyé vers le descripteur 3 (fichier toto)
- exec 3>&-
  - Le descripteur numéro 3 (fichier toto) est fermé

Redirections Multiples

- Contrairement aux *pipes* , les redirections peuvent être multiples
  - L'option MULTIOS doit être validée
    - commande : setopt multios
  - La sortie standard peut être redirigée plusieurs fois
    - Equivalent à la commande 'tee fic1 fic2 …'
    - exemple : ls –al > titi > toto
  - La sortie standard peut être redirigée plusieurs fois
    - Equivalent a la commande 'cat fic1 fic2 …'

Mécanismes de substitution (Expansion)

- Plusieurs types de substitution, réalisées dans l'ordre suivant :
  1. History expansion (shell interactif)
  2. Alias expansion
  3. Process substitution + Parameter expansion + Command substitution + Arithmetic expansion + Brace expansion
  4. Suppression des \, ', et "
  5. Filename expansion
  6. Filename génération

Mécanisme d'historique (History substitution)

- Réutilisation de (parties) de commandes précédentes
  - Sauvegarde des commandes dans l'historique (taille = HISTSIZE)
- Substitution d'historique : commence par '!'
  - 3 catégories
    - Désignation d'événement (= commande complète)
    - Désignation de mots
    - Modificateurs
  - Syntaxe générale :

  !evt:word…:mod…

Mécanisme d'historique : Désignation d'évènements

- !! : commande précédente
- !N : Nième commande depuis le début l'historique
- !-N : Nième commande en remontant l'historique depuis maintenant
- !STR : la commande la plus récente qui débute par STR
- !?STR[?] : la commande la plus récente qui contient STR
- !{…} : délimite la commande d'historique

Mécanisme d'Historique: Désignation de mots

- 0 (zéro) : le premier mot (commande)
- N : le Nième mot
- ^ : le premier argument (idem '1')
- $: le dernier argument
- % : le mot qui correspond à la recherche ?str la plus récente
- x-y : une suite formée des mots numéros x à y (-y ⇔ 0-y)
- x* : x-$
- x- : idem sauf que le dernier mot est omis
  - Pratique pour les commande du type cp sources … destination

Mécanisme d'Historique Principaux Modificateurs

- h : supprime la fin d'un chemin, ne garde que le premier élément (head)
- r : supprime suffixe de la forme .xxx
- e : supprime tout sauf le suffixe
- t : supprime le début du chemin, ne garde que la fin (tail)
- & : répète la substitution précédente
- l/u : conversion min/MAJ
- s/L/R/ : remplace L par R

Mécanisme d'Historique : Exemples

- [1] cp toto titi tata dest/
- [2] !!0- dest2/
  - tous les mots de la dernière commande sauf le dernier
  - => [2] cp toto titi tata dest2
- [3] ls !-2:$
  - Le dernier mot de l'avant dernière commande
  - => [3] ls dest
- [4] echo "!?cp?" > exmple.txt
  - La dernière commande qui contient cp
  - => [4] echo "cp toto titi tata dest2" > exemple.txt
- [5] mv !!$!!$:r foo
  - le dernier mot de la dernière commande ; le dernier mot de la dernière commande dans lequel on a supprimé le .txt
  - => [5] mv exemple.txt exemple.foo

Mécanisme de Substitution de Processus (Process substitution)

- 3 formes de substitutions
  - <(CMD)
  - >(CMD)
  - =(CMD)
- Règles de fonctionnement communes aux 3 formes :
  - La commande CMD est exécutée dans un processus séparé
  - Le résultat de CMD est placé dans un fichier temporaire
  - C'est **le nom de ce fichier temporaire** qui est substitué à l'une des 3 formes précédentes

Mécanisme de Substitution de Processus : Différences entre les 3 formes

- **<(CMD) :**
  - la sortie standard de CMD est branchée sur le fichier temporaire (autrement dit, CMD écrit dans le fichier)
  - Le fichier temporaire est de type "tube nommé" (named pipe) => "non seekable"
- **>(CMD)**
  - L'entrée standard de CMD est branchée sur le fichier temp. (CMD lit dans le fichier)
  - Tube nommé, donc "non seekable"
- **=(CMD)**
  - Comme pour <(CMD) : la sortie standard est branchée sur le fichier
  - Le fichier temporaire est un "vrai" fichier, donc "seekable"
  - **Inconvénient :** CMD doit être terminée et sont résultat sauvé en entier dans le fichier temporaire

Mécanisme de Substitution de Processus : Exemple

- `paste <(cut -f1 FILE1) <(cut -f3 FILE3) > >(PROCESS1) > >(PROCESS2)`
- Résultat :
  - Le champ 1 du fichier FILE1 et le champ 3 du fichier FILE3 sont extraits avec la commande cut et collés avec la commande paste.
  - Le résultat est envoyé vers les processus PROCESS1 et PROCESS3

Mécanisme de Substitution de Commande

- De la forme $(CMD) ou `CMD`
  - La forme est substituée par le résultat de CMD
  - Exemples :
    - `echo "Répertoire courant : `pwd`"`
    - `echo "Votre groupe d'utilisateur : $(id -gn )"`
  - Les éventuels sauts de lignes en fin de résultat sont éliminés
  - Lorsque l'option GLOB_SUBST est positionnée, le résultat subit la substitution "filename génération" (cf plus loin)

Quelques effets du ~ (tilde)

- ~ seul : équivalent de $HOME
- ~+ : équivalent de $PWD
- ~- : répertoire avant le dernier cd
- ~login : désigne le homedir de l'utilisateur 'login'

Expansion/Génération de noms de fichiers (globbing)

- Désigner de façon simples les paramètres des commandes de type fichier
  - sans avoir besoin de les désigner tous explicitement
    - Via des expressions régulière
    - Expressions **spécifiques** au globbing

Globbing Elémentaire

- option GLOB validée
  - l'expression désigne un nom de fichier entier (du premier au dernier caractère)
  - méta-caratères
    - * : 0 ou plusieurs caractères quelconques
    - ? : un caractère quelconque
    - [xyzt] : un carctère parmis x, y z ou t
    - [^xyzt] : tout sauf un caractère parmis x,y,z ou t
    - <nb1-nb2> : un nombre compris entre nb1 et nb2
    - <-nb2>, <nb1-> : resp un nombre < nb2 ou un nb > nb1
    - <> : un nombre quelconque
    - (exp1|exp2) : exp1 ou exp2

Globbing Elémentaire : Exemples

- ls f* f?o
- ls f[aeiou]o
- ls (f?o|b?r)
- ls <0-9>*
- ls *<-5>
- ls *<>*

Globbing Etendu

- L'option EXTENDED_GLOB doit être validée
  - setopt EXTENDED_GLOB
  - Deux nouveaux meta-caractères :
    - ^ : tout sauf ce qui suit
    - exp# : 0 ou plusieurs occurrences de exp
    - exp## : 1 ou plusieurs occurrences de exp
  - Exemples :
    - ls ^*.c
    - file *.[^(o|out)]
    - ls (fo)#* commence par 0 ou plusieurs fois 'fo'
    - ls fo##* commence par fo…ooo
    - ls (*TP*/)##* : contenu des (sous-)sous-répertoires contenant 'TP' dans leur nom

Globbing des Fichiers Cachés

- L'option GLOB_DOTS doit être validée
- Le globbing s'applique alors aussi aux fichiers cachés (commençant par .)

Globbing : Qualificatifs sur le Type

- Une expression en fin de motif recherché permet de "spécialiser" le globbing :
  - exp(exp2)
  - qualificatifs de exp2 :
    - @ = lien symbolique
    - / = répertoire
    - r : lisible ; R : lisible other ; A : lisible groupe
    - w, W, I : ecriture pour user, other, groupe
    - x, X, E : exécution pour user, other, groupe
    - u : propriétaire, u[lui] : 'lui' est propriétaire
    - s: setuid, S: setgid
  - Combinaisons ET (), OU (,), NON (^)

Globbing: Exemple d'Utilisation des Qualificatifs

- ls *(@) : lister les liens symboliques seulement
- ls *[aeiou](x) : lister les fichiers qui se terminent par une voyelle qui ont la permission d'exécution
- ls *to(/) : lister les répertoires dont le nom se termine par 'to'
- ls *(UX^@) : lister les fichiers qui m'appartiennent, exécutables pour tous, mais qui ne sont pas des liens
- ls *(x,X) : lister les fichiers qui sont exécutables pour moi ou pour les autres

Autres mécanismes de Globbing

- ** : un chemin quelconque
  - ls **/*.c : rechercher et lister tous les fichiers qui se terminent par .c à partir du répertoire courant
- =bin : nom complet de l'exécutable 'bin'
  - ls =emacs
    /usr/local/bin/emacs
  - Equivalent à : ls `which emacs`
  - Desactivé par l'option "noequals"

3. Eléments pour l'Ecriture de Scripts

- Shell = langage de commandes
- Shell Script = programme écrit en langage de commandes
- Rôles du shell
  - Combiner/relier des commandes existantes avec redirections, |, ||, && et instructions de contrôle
    - Permettre la construction de nouvelles commandes
  - Regrouper dans des fichiers des suites de commandes
    - Exécution de tâches répétitives

Variété des Instructions

- Le shell propose toute une "palette" d'instructions pour :
  - Mémoriser (variables, types, arguments)
  - Compter (expressions arithmétiques et logiques)
  - Tester (test, [[ ]])
  - Bifurquer (if, case, ||, &&)
  - Boucler (while, for, select, break, continue)
  - Interagir par E/S (read, print, <, >)
  - Sous-programmes : funtion, unfunction, return, exec, source, export

Variables - Expressions

- Par défaut : variable = chaîne de caractères
  - Initialement vide
- Variables entières
  - typeset –i var1 var2 : var1 et var2 de type entier
    - typeset –i e1 e2
    - echo $e1 $e2
      => 0 0
  - Expressions arith : délimitées par des doubles parentheses
    - ((i1 = $i2 +1))
    - ⇔ i1=$(expr $i2 + 1)

Variables de type Tableau

- Déclaration implicite lors de l'initialisation
  - tab=(e1 e2)
  - echo $tab[0] => e1
  - echo $tab[1] => e1 aussi !
  - echo $tab[2] => e2
  - echo $tab[-1] => e2
  - echo $#tab => 2 (nb éléments)

Mécanismes de Substitution de Variables

- ${var:-word} : remplacé par le contenu de la variable si elle existe, ou par word sinon
- ${var:=word} : idem mais variable est affectée avec word
- ${var#exp} : retranche à var le plus petite chaîne en partant de la gauche correspondant à exp
- ${var##exp} : idem mais retranche la plus grande chaine
- ${var%exp}, ${var%%exp} : idem mais en partant de la droite

Quelques Instructions de Contrôle

- Boucler sur chaque ligne d'un fichier
  - `cat fich | ( while read ligne`
    `do`
    `… echo $ligne`
    `done )`
  - `( while read ligne`
    `do`
    `… echo $ligne`
    `done ) < fich`

Quelques Instructions de Contrôle

- Définir une fonction
  - ```
    backup() {
        suffixe=$2
        for f in $1.*
        do
            mv $f $f.suffixe
        done }
    ```


Fonctions en Autoload

- Au lieu de définir un grand nombre de fonctions dans le .zshenv, zsh peut les charger en mémoire à la demande
  - Stocker la définition de la fonction dans un fichier de même nom
  - Créer un répertoire pour stocker les fichiers contenant les fonctions
  - Dire au shell ou chercher les fonctions
  - Déclarer les noms des fonctions autoload dans le .zshenv
  - Exemple :
    - mkdir $HOME/ZshFunc
    - mv backup $HOME/ZshFunc
    - chmod a+x ZshFunc
    - Dans .zshenv :
      FPATH=$HOME/ZshFunc
      autoload backup

Exemple de Script

- Appliquer backup à une liste de scripts
- ```
  cat toutsauver
  #! /usr/local/bin/zsh
  le_suffixe=$1
  shift
  typeset -i nb_param=$#
  [[ $nb_param -lt 1 ]] && exit
  typeset -i i=1
  while [[ $i -le $nb_param ]]
  do
      eval backup '$'{$i} $le_suffixe
      i=$i+1
  done
  ```


Exemple de Script : Démo

- ```
  ls
  pourtp1.c tetetp1.h tp1.c tp1.out
  ```
- `toutsauver old tp1 tete pourtp1`
- ```
  ls
  pourtp1.c.old tetetp1.h.old tp1.c.old
  tp1.out.old
  ```