Séance No 2

- résumé séance 1 (processus, redirection et enchaînement de commandes, quelques filtres classiques)
- premiers scripts shell
- variables

grep, egrep & Co

- grep chaine: sélectionne les lignes qui contiennent la chaine
- grep petit /etc/passwd: sélectionne les lignes de /etc/passwd contenant la chaîne petit
- caractères spéciaux de la commande egrep:
  - ^: début de ligne
  - $: fin de ligne
- grep '^petit:' /etc/passwd: sélectionne les lignes commençant par petit:

cut

- sélectionner certaines colonnes

tr

- permet de remplacer des caractères par d'autres
- tr 'abcdefghijklmnopqrstuvwxyz' 'nopqrstuvwxyzabcdefghijklm'

Expressions rationnelles

- utilisées par des nombreux outils pour décrire des chaînes de caractères
- Il existe deux familles d'expressions rationnelles :
  - expression régulières de base utilisées par
    - vi
    - grep
    - expr
    - sed
  - expressions régulière étendues utilisées par :
    - grep -E et egrep
    - awk

•expressions rationnelles: éléments communs

- les caractères suivants sont communs aux expressions rationnelles de base et étendues:
  - ^: début de ligne
  - $: fin de ligne
  - .: un caractère quelconque
  - [liste]: un caractère de la liste (idem [] du shell, ^: négation)
  - *: 0 à n fois le caractère/expression précédente
  - \<, \>: chaîne vide de début et de fin de mot
  - \c: protège le caractère c de tout interprétation. ex.: \$: caractère $

•Exemples:

- petit: la chaîne petit
- ^petit: chaîne démarrant par petit
- petit$: chaîne finissant par petit
- DSCN[0-9]*\.JPG: DSCN suivi de 0 à N chiffres suivi de .JPG (noter la protection du caractère .)
- ^$: chaîne vide (utile pour détecter les lignes vides)
- ^[^:]*:[^:]*:[^:]*:1000 une chaîne contenant des champs séparés par le caractère : et dont le 4e champ vaut 1000. Appliqué sur un fichier /etc/passwd, sélectionne les lignes dont le groupe est 1000.

caractères de protection

- \: protège le caractère suivant (sauf si c'est un saut de ligne)
- \ en fin de ligne:
- 'chaine': protège tous les caractères de la chaîne
- "chaine": protège tous les caractères sauf $, ` et \ dans le cas où \ est devant $, ", \ et en fin de ligne.
- caractères à protéger :

  | & ; < > ( ) $ ` \ " ' <espace> <tab>

  <saut de ligne> * ? [# ~ = %

- caractères de protection: exemples

```
$ echo 'un test $#\c\$'
un test $#\c\$
$ echo "un test $#\c\$"
un test 0\c$
$ echo un test \$
un test $
$ echo un test \$ \\
un test $ \
```

commande test

- réalise des tests simple, le code de retour indique que le test est positif ou négatif
- test -d /var/tmp : teste si /var/tmp est un dossier
- test -x /bin/ls: teste si /bin/ls est un exécutable
- test 1 = 2: teste l'égalité de deux chaînes
- forme alternative :
  - test -d /var/tmp
  - [-d /var/tmp]

structure de contrôle if

- syntaxe:

```
if commande1
then
    commande2
[elif commande3
then commande4]
...
[else
    commande5]
fi
```

- si commande1 retourne 0, on exécute commande2 sinon, si commande3 retourne 0, on exécute commande4 ... sinon commande5

Scripts shell

- fichier texte contenant des commandes
- nouvelle commande

lancement d'un script shell

- méthode nécessitant une accès en lecture
  - sh nomScript
  - sh < nomScript (peu utilisée)
- Méthodes nécessitant un accès en exécution et en lecture
  - nomScript ou chemin/nomScript ou ./nomScript
  - la première ligne du script qui précise le choix de l'interpréteur est de la forme:

    #! /bin/bash

•Exemple:

- considérons le script /tmp/foo.sh:

```
#! /bin/bash
pwd
cd /etc
pwd
echo coucou
```

- Utiliser les trois méthodes proposées pour le lancer
- exécuter la commande pwd après chaque lancement
- ces 3 méthodes exécutent le script dans un processus shell enfant. Comment le vérifier ?

Exemple (suite)

- ràf: une animation qui montre l'exécution du script et la création du processus
- dans une version ultérieure de ce document

Exécution par le shell courant

- on utilise la commande interne « . »
- . foo.sh
- conséquences :
  - le script est exécuté par le shell courant
  - il peut modifier les données de ce shell courant (variables, dossier de travail, ...)
- Exemple: exécuter « . foo.sh »

Erreurs classiques

- le dossier courant n'est pas dans le PATH. Indiquer le chemin absolu ou relatif. Par ex.: ./foo.sh

```
cd /tmp;foo.sh
commande non trouvée
```

- le fichier n'est pas exécutable. Utilisez chmod pour lui ajouter le droit x:

```
cd /tmp;./foo.sh
permission non accordé
```

Code de retour

- exit: termine le script en retournant un code de retour
- Rappel:
  - 0 : exécution normale
  - autre valeur: situation anormale
- sans utilisation de exit: le code de retour est celui de la dernière commande du shell

Variables

- variables d'environnement
  - exportée ou non exportées
- variables utilisateurs
- variables réservées du shell

Variables d'environnement

- variables au shell et/ou au commandes lancées à partir de lui
- set : pour obtenir la liste des variables d'environnement
- obtenir la valeur d'une variable: $nomVariable ou ${nomVariable}
- changement de valeur: nomVariable=valeur
- variables exportées:
  - variables communiquées aux commandes lancées
  - env: pour obtenir la liste des variables exportées
  - export nomVariable

Exemples de variables d'environnement

Nom	rôle
PATH	liste des dossiers où le shell cherche les exécutables
HOME	dossier personnel de l'utilisateur
PWD	dossier courant
TERM	type de terminal
SHELL	nom du shell en cours d'exécution
USER	
LANG	lié aux locale et à la gestion des langues
DISPLAY	« display » X Window (clavier/écran/souris)

Quelques variables d'environnement définies sous bash

variables utilisateur

- définie par l'utilisateur pour ses besoins
- nom des variables
  - premier caractère [a-zA-Z]
  - caractères suivants [a-zA-Z0-9_]
- définition: nom=mot
- valeur: $nom ou ${nom}

variables utilisateur (2)

- variables indéfinie:
  - une variable jamais initialisée est vide
  - unset permet d'annuler la définition d'une variable qui devient alors indéfinie

```
$test=hello
$echo $test
hello
$unset hello
$echo $hello
```

variables utilisateurs (3)

- espace dans les valeurs: " ou \
  - test1="m1 m2 m3"
  - test1=m1\ m2\ m3
- Attention à bien isoler le nom d'une variable:
  - f= toto;g=titi;h=$titi_$toto # $titi_ est vide
  - f= toto;g=titi;h=${titi}_$toto

Substitution de commandes

- remplacer une commande par l'affichage de son résultat d'exécution
- ancienne syntaxe à éviter : `commande`
- syntaxe moderne: $(commande)
- Exemples:

```
echo je suis $(whoami)
je suis petit
nom=$(whoami)
echo je suis $nom
```

caractères de protection

- apostrophe : protège tous les caractères spéciaux sauf elle
  - echo '$HOME' affiche '$HOME'
  - echo $HOME affiche la valeur de la variable HOME
- \: protège le caractère qui suit
- guillemets : protège tout sauf lui-même, $, $() et \

variables réservées du shell

- paramètres positionnels
  - $#: nombre d'arguments
  - $0: nom du script
  - $1: valeur du premier argument
  - $2: valeur du deuxième argument
  - ...
  - $9: valeur du neuvième argument
  - ${10}: avec des shells modernes
  - $* et $@: liste de tous les arguments

variables réservées du shell (2)

- shift [n]: décale de n positions la liste des arguments vers la gauche
  - par défaut, n vaut 1
  - shift: remplace $1 par $2, $2 par $3, ...
- $?: code de retour de la dernière commande exécutée
- $$: pid du shell
- $!: pid du dernier processus lancé en arrière plan

Bilan séance No 2

- A la fin de cette séance No 2, vous devez :
  - être à l'aise dans l'utilisation des filtres de la séance No 2 (cf sujet de TD No 3)
  - savoir définir une variable et utiliser sa valeur
  - savoir écrire de petits scripts shell
  - faire la différence entre variables exportées et non exportées