Définition

- SHELL: programme en mode texte assurant l'interface entre l'utilisateur et le système unix.
- S'utilise :
  - En interactif depuis une fenêtre terminal (xterm, connexion distante texte, ...) : interpréteur de commande
  - Pour réaliser des scripts (fichiers de commandes) : langage de programmation

Présentation

- Pascal PETIT
- petit@shayol.org
- tel: 01 69 47 XX XX

Shell: où que je clique ?

- On ne clique pas : ça s'utilise avec une souris à 105 touches et sans boule : un clavier :-)
- L'accès est moins immédiat que celui d'une interface graphique
- Plus de liberté/possibilités qu'avec une interface graphique
- Langage de programmation: possibilité d'exprimer des requètes complexes
- Utilisation interactive ou pour écrire des fichiers de commandes (scripts)

Unité libre: langages de scripts

- 15h00: 6 séances de cours/TD de 3h00
- La dernière séance est une séance évaluée
- Séance1: historique, bases, SGF
- Séance2: processus, enchaînement de commandes
- Séance3: variables
- Séance4:
- Séance ....:
- Séance: évaluation

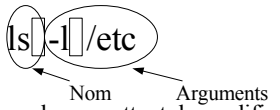
Historique

- Les deux shells des origines sont à l'origine de deux familles de shells aux syntaxes incompatibles :
  - Le shell le plus ancien : sh ou Bourne shell écrit dans les années 70 par Steve Bourne. Tout système système unix a un shell /bin/sh qui est un bourne shell (ou un shell compatible);
  - Le csh: écrit à la même époque par Bill Joy incompatible avec le bourne shell mais offrant quelques fonctionnalités supplémentaires (historique des commandes, aliases, contrôle de tâches, ...

Séance No 1

- Définition, Historique
- Boucle d'interprétation, commandes simples: forme générale
- Erreurs classiques et aide en ligne
- Systèmes de gestion de fichiers (SGF), fichiers, dossiers, arborescence
- droit d'accès, commandes sur les fichiers/dossiers
- processus
- redirections, enchaînements

commandes simples: forme générale



- arguments:
  - paramètres optionnels permettant de modifier le comportement de la commande
  - liste des entités auxquelles doit s'appliquer la commande (nom de fichier, processus, utilisateur, ...)
- Exemples:
  - mozilla
  - mozilla -P toto www.univ-evry.fr
  - ls -lrt /etc
  - find ~ -name *.avi -exec rm -f {} \;

quelques commandes simples

- who: liste des utilisateurs ayant une session en cours sur l'ordinateur
- w: idem mais indique aussi ce qu'ils font
- date: date courante
- echo: affiche ses arguments séparés par une espace

Exemples

- Le shell va servir à lancer des commandes internes ou externes
- Exemple de session :

```
#un commentaire commence par #
# lister les fichiers présents
# dans le dossier /etc
ls -l /etc
# liste des utilisateurs connectés
# sur l'ordinateur
who
```

Historique (2)

- Ksh: korn shell (David Korn, 1983) sur la base du bourne sh. Le ksh 88 (ou +) est livré avec tous les unix commerciaux. Base de la norme IEEE Posix 1003.2;
- Tcsh: un shell évolué de la famille csh utilisé dans les années 90 comme shell interactif;
- Bash: Bourne Again sh, le shell de la FSF. Compatible posix 1003.2. Le shell de base des distribution linux.
- Zsh: un shell riche en fonctionnalités. Probablement le meilleur choix actuel en interactif.

Historique (3)

- POSIX:
- SUS: Single Unix Specification: spécification suivie par les unix commerciaux (et de nombreux non commerciaux) modernes. Proche de la norme POSIX.
- se limiter à SUSv3/POSIX garantit une compatibilité maximale avec les unix utilisés de nos jours
- SUS: http://www.unix.org/what_is_unix/single_unix_specification.html
- De nos jour, il est conseillé d'utiliser un shell compatible posix/sus: ksh, bash et zsh.

Boucle d'interprétation

- Le shell est un programme qui réalise la boucle suivante :
  - Boucle :
    - Lire la ligne de commande
    - Décoder la ligne de commande
    - Exécution de la ligne de commande en créant un processus dans le cas de commandes externes
    - Attendre la fin de l'exécution du processus
    - Retourner en début de boucle

Systèmes de gestion de fichiers (SGF)

- SGF: mode d'organisation et de stockage des données sur disque;
- Exemples: FAT32, NTFS, ext2fs, ext3fs, reiserfs, UFS, ...
- Les SGF ont des propriétés et fournissent des services variés
- Exemple:
  - les SGF Unix (ext2fs, UFS, ...) : droits sur les fichiers.
  - FAT32: pas de droits d'accès aux fichiers

Erreurs

- 5 causes classiques d'erreurs
  1) syntaxe ou chemin incorrect (commande inconnue, ...)
  2) paramètres incorrects (fichier inconnu, ..)
  3) droits d'accès : permission refusée (accès à un fichier, ...)
  4) options invalides (syntaxe des options de la commande)
  5) erreur de conception : le comportement n'est pas celui attendu

SGF (suite)

- Les SGF unix fournissent un sous-ensemble commun de fonctionnalités: celui dont nous parlerons.
- Chaque SGF peut fournir plus que ce sous-ensemble
- Fichier unix: fichier disque mais aussi ressource du système (pilote de périphérique, terminal, mémoire, ...)
  - /dev/hda1 : partition 1 du disque 1 (Linux)
  - /dev/kmem: mémoire virtuelle du noyau

•A l'aide

- le manuel
- option « --help » de certaines commandes
- la documentation de votre système d'exploitation ou du programme posant problème
  - souvent /usr/share/doc, /usr/local/share/doc
- recherche sur le WeB: quelqu'un d'autre a forcément déjà eu ce problème

Droits d'accès aux fichiers

- 3 types d'accès: lecture (R), écriture (W) et exécution (X)

Objet/Droit	R (lecture)	W (écriture)	X (exécuter)
fichier régulier	lire le contenu	modifier le fichier	exécuter le fichier
dossier	lister le contenu du dossier	modifier le contenu du dossier (y compris destruction de fichier)	utiliser le dossier dans un chemin ou s'y positionner

- 3 classes d'utilisateurs: le propriétaire du fichier, le Groupe du propriétaire du fichier, les Autres utilisateurs.

type fichier	Propriétaire			Groupe du proprio			Autres utilisateurs		
-	R	W	X	R	-	X	R	-	X

- informations dans l'inode, affichage avec « ls », changement avec chmod, chgrp et chown

•Le manuel

- dans une version ultérieure de ce document
- les sections du manuel: ràf
- exemples d'utilisation: ràf + exemple dans 2 sections

commande de base: umask

- définit les droits d'accès par défaut d'un fichier
- les droits sont le complément du paramètre d'umask: on laisse tout sauf les droits précisés
- Exemple:
  - umask 002 : mode par défaut: RWXRWXR-X (tout sauf 002)
  - umask 026: mode par défaut: RWXR-X--X (tout sauf 026)
  - umask a=rx,gu+w: mode par défaut: RWXRWXR-X
  - umask -S : affiche le l'état courant sous forme symbolique : u=rwx,g=rwx,o=rw dans notre exemple.

Droits d'accès (2): suid, sgid, sticky bit

- 3 autres « droits » spéciaux:
  - bit SUID: le programme s'exécute avec les droits de son propriétaire (au lieu de ceux de l'utilisateur qui le lance)
  - bit SGID: le programme s'exécute avec les droits du groupe propriétaires du fichier
  - sticky bit :
    - sur un fichier exécutable : (obsolète) maintient le fichier en mémoire après l'exécution pour diminuer le temps de la prochaine invocation
    - sur un dossier: seul le propriétaire du fichier a le droit de le supprimer. Exemple: /tmp/

Commandes de base: chown, chgrp

- chown -R [-H | -L | -P] proprio[:groupe] fichier
- chgrp -R [-H | -L | -P] groupe fichier ...

Commandes de base: chmod

- chmod [-R] mode fichier ...
- -R: fichier est un dossier, chmod agit récursivement sur fichier et sur son contenu
- mode:
  - forme numérique: 644
    - pour u: 400 (r), 200 (w) et 100 (x)
    - pour g: 40 (r), 20 (w) et 10 (x)
    - pour o: 4 (r), 2 (w) et 1 (x)
  - forme symbolique: [ugo][+-=][rwxXstguo]

chown/chgrp: exemples

chmod: exemples

- chmod ràf

Exemples

- Stat fichier (noter ctime, mtime et atime)
- Cat fichier
- Stat fichier (atime a changé)
- Chmod fichier
- Stat fichier (ctime a changé)
- Modif fichier
- Stat fichier (mtime a changé)

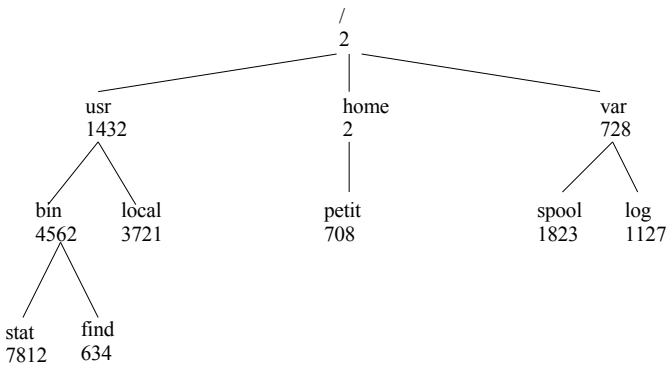
Commandes de base: ls

Arborescence

- Sous unix, on a une arborescence unique (donc pas de C:\, D:\, ...comme sous windows)
- Le disque système contient la racine absolue /
- toute l'arborescence est sous cette racine absolue
- Les systèmes de fichiers des autres partitions s'intègrent dans l'arborescence en prenant la place d'un dossier existant
- la racine d'un système de fichier a 2 comme numéro d'inode

Commandes de base: cat

arborescence



Commande de base: stat

Commandes de base: cp

- copie des données d'un fichier (source) dans un autre (cible) :
  - la cible n'existe pas : création d'un nouvel inode et recopie des données du fichier
  - la cible existe: inode destination inchangée, recopie des données du fichier dans la cible

Chemin

- /usr/bin/stat: chemin absolu du fichier stat
- chemin absolu: chemin depuis la racine absolue
- notion de dossier courant
- chemin relatif: chemin depuis le dossier courant

cp (2)

- gnu cp: en standard sous Linux, installable facilement ailleurs
- fournit des options non standard mais pratiques
- ràf

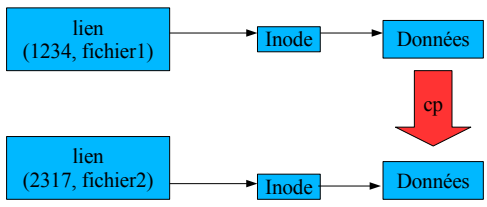
Commandes de base:

- pwd : indique le dossier courant
- cd : changer de dossier courant
- mkdir: pour créer un dossier
- rmdir: détruit les dossiers vides

Commandes de base: rm

- suppression d'un lien d'un fichier ou plusieurs fichiers: `rm [-fiRr] fichier1 ...`
- options:
  - -i: demande de confirmation pour tout fichier à supprimer (aff sur stderr et lecture sur stdin)
  - -f: supprime les messages d'erreur lorsqu'un fichier n'existe pas et supprime la demande d'acquittement si l'utilisateur de rm n'a pas les droits d'écriture sur le fichier à supprimer
  - -R ou -r: supprime récursivement le contenu d'un dossier avant d'appliquer rmdir au dossier.

commande de base: cp



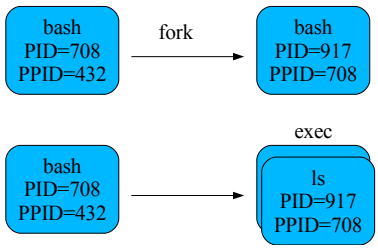
cp fichier1 fichier2

processus

- un programme: un fichier sur disque
- un processus: un programme en cours d'exécution
  - le code exécutable du programme
  - les données de l'instance en train de s'exécuter
- programme réentrant:
  - deux instances du même programme partagent le même code exécutable
  - elles ont par contre chacune leurs données
- processus système (daemon)/utilisateur

hiérarchie de processus, recouvrement

- un processus (processus fils) est toujours créé par un autre processus (processus père):
  - fork: création d'une copie du processus père
  - exec: recouvrement par le processus fils



Hiérarchie de processus

- tout processus a un processus parent sauf le processus initial
- processus initial : init (pid 1)
- arrêter la machine: demander à init d'arrêter tous ses processus fils

•rm :exemples

Commandes de base: mv

- mv [-fi] source destination
  - renomme un lien. Source et fichier sont des fichiers réguliers
- mv [-fi] source1 ... destination
  - renomme les sources en les déplaçant **dans** le dossier destination
- la commande fonctionne aussi si sources et destinations sont dans des systèmes de fichiers différents.
- la seconde forme est utilisée si la destination est un dossier existant

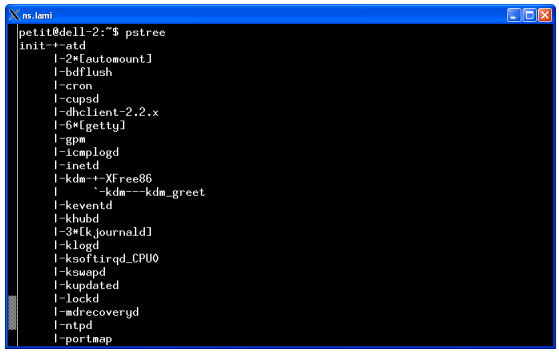
mv: exemples

- mv [-fi] source destination
  - renomme un lien. Source et fichier sont des fichiers réguliers
- mv [-fi] source1 ... destination
  - renomme les sources en les déplaçant **dans** le dossier destination
- la commande fonctionne aussi si sources et destinations sont dans des systèmes de fichiers différents.
- la seconde forme est utilisée si la destination est un dossier existant

caractéristiques des processus

- statiques
  - PID
  - PPID
  - propriétaire
  - terminal d'attache pour les entreés-sorties
- dynamique
  - priorité
  - nice
  - consommation cpu/mémoire
  - dossier de travail

pstree



commande ps

- 2 syntaxes:
  - syntaxe Systeme V: option précédées de -
  - syntaxe BSD: options NON précédées de -
  - quelques options SysV:
    - -e ou -A: tous les processus
    - -a: tous les processus associés à un terminal
    - -H: représentation hiérarchique (forêt)
    - -f: format complet;-l: format long (encore plus détaillé)
    - -o: pour modifier le format de sortie (cf manuel)
    - -g, -p, -t, -u: n'affiche que les processus des groupe (-g), processus (-p), terminaux (-t) ou utilisateurs (-u) listés.

commandes internes/externes

- commande externe: fichier exécutable:
  - création d'un nouveau processus chargé d'exécuter la commande
- commande interne:
  - exécutée par le shell (pas de création de nouveau processus)

commande ps: exemple

type

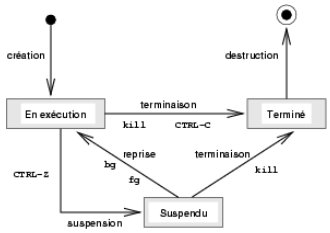
- type indique si une commande est interne
- options non standard:
  - -a: indique toutes les implémentations
  - -p : indique le chemin de la commande (rien si interne)
- Exemples: testez type sur les commandes suivantes :
  - cd
  - ls
  - pwd
  - file

avant plan/arrière plan/détachement

- dans une version future de ce document (ràf)

Etat d'un processus

- R: exécution
- Z: zombi: il est mort mais son père ne le sait pas
- D: le processus ne peut être interrompu
- S: suspendu
- T: terminé



priorité des processus

- l'exécution des divers processus est gérée par un ordonnanceur (scheduler)
- une priorité est définie dynamiquement
- but: que chaque processus puisse avancer son exécution tout en respectant des priorités
- nice: permet d'influer sur la priorité des processus
  - de 0 à 19 pour un utilisateur
  - de -20 à 19 pour root
  - plus le chiffre est élevé, moins le processus est prioritaire

gestion de processus

- &
- bg
- fg
- jobs
- Ctrl-C
- Ctrl-Z

code de retour

- valeur à laquelle le processus père peut accèder
- 0: terminaison normale
- autre valeur: situation anormale
- commande1 && commande2:la commande2 est exécutée si la commande 1 réussit
- commande1 || commande2: la commande2 est exécutée si la commande 1 échoue
- exemple: commande test
- exemple: construction if/then/else/fi

Signaux

- permettent au système de communiquer avec les processus
- signaux utiles
  - STOP: suspendre
  - CONT: reprendre
  - HUP (1): souvent: relecture configuration
  - KILL(9): tuer sans possibilité de traitement
  - INT(2): équivalent à Ctrl-C: interruption gérable. permet au processus de gérer son interruption
- kill -signal PID

redirection de la sortie d'erreur standard

- commande 2> fichier
- commande 2>> fichier
- pratique pour isoler messages d'erreur et sortie
- /dev/null: le trou noir: pour éliminer les messages d'erreur
- du -sk /var/* 2> /dev/null

Entrées-sorties

- Entrées-sorties
  - entrée standard: 0
  - sortie standard: 1
  - sortie d'erreur standard: 2

redirection simultanées

- on peut rediriger plusieurs descripteurs sur une même ligne de commande
- ls > /tmp/f1 2> f2
- les redirections sont traîtées de gauche à droite
- du -sk /var/* > /tmp/resultat 2> /tmp/erreur
- en cas de redirection simultanée avec cette syntase: impérativement vers des fichiers différents

Héritage

- les descripteurs d'un processus enfant sont initialement les mêmes que ceux du processus père.
- si on ne les modifie pas, en sortie, les affichages s'entrelacent.
- il est possible de changer la valeur de entrée standard et des sorties standard et en erreur
  - pour les rediriger depuis/vers un fichier : <, >, >>, 2>, 2>>
  - pour les rediger depuis/vers un processus : |

redirection en entrée

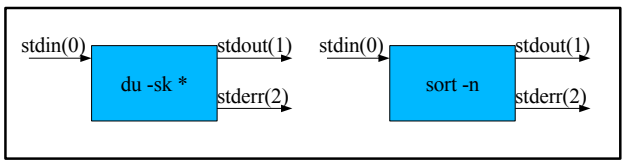
- commande < fichier
- exemples:
  - mail petit < texte
  - wc -l < /etc/passwd

redirection des sorties

- >: le contenu du fichier est remplacé par la sortie de la commande
- >>: la sortie s'ajoute à la fin du fichier
- exemples:
  - ls /etc > /tmp/foo.txt
  - cat ls
  - ls /usr/bin >> /tmp/foo.txt
  - du -sk /var/* > /tmp/bar.txt
  - date > /tmp/bar.txt (noter que le No d'inode est inchangé)

enchaînement de commandes

du -sk * |sort -n



L'ensemble forme une nouvelle commande

- 1
- 12
- 3
- 3456
- 56
- 789
- 8
- 9

Filtres

- commande lisant leurs données sur l'entrée standard et envoyant leur sortie sur la sortie standard
- pratique pour les enchaînements de commandes
- philosophie unix: des commandes simples que l'on combine entre elles

redirections avancées

- &1: valeur du descripteur de fichier 1
- &2: valeur du descripteur de fichier 2
- 1>&2: l'entrée standard est redirigée vers le même fichier que la sortie standard
- sert pour des redirection simultanées vers un même fichier
- Exemples:
  - ls > /tmp/test 2>&1 #OK
  - ls 2>&1 >/tmp/test #pas OK: stderr est toujours lié au terminal

<<

- dans une version ultérieure de ce document

fermeture d'un descripteur

- dans une version ultérieure de ce document

Sort: options courantes:

- -t car_sep: permet de préciser le caractère qui sépare les champs du fichier
- -k : précise les champs sur lesquels portent le tri
- -o: indique un fichier de sortie (par défaut: sortie standard)
- -d: supprime les doublons
- -c : vérifie si un fichier est trié. Le résultat est indiqué uniquement par le code de retour: 0 si trié, 1 sinon.
- -m : fusionne des fichiers supposés déjà triés

Bilan

- A la fin de cette première séance, vous devez :
  - connaître les notions de systeme de gestion de fichiers, fichiers, dossier, chemin
  - connaître la forme générale d'une commande
  - savoir utiliser le manuel unix
  - savoir vous déplacer dans une arborescence,
  - créer/déplacer/copier des fichiers, des dossiers
  - connaître la notion de processus, d'arborescence de processus, les caractéristiques d'un processus, entrée et sortie standard
  - comprendre la notion d'enchaînement de commandes

uniq

- supprime les doublons d'une liste triée
- exemple: cat /tmp/test.txt |sort|uniq
- voir manuel pour les autres options
-

Séance No 2

- rappels sur les entrées/sorties
- quelques filtres classiques

tail/head

- queue/tête d'un fichier

sort

- selon SUSv3, sort a trois fonctions sur son entrée standard ou des fichiers textes constituées de lignes contenant un ou plusieurs champs :
  - trier les données (par défaut)
  - fusionner des fichiers triées en une sortie globale triée (option -m)
  - vérifier que les données sont triées (option -c)

tr

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

wc

- compte le nombre de lignes, de mots et de caractères
- wc -l : nombre de lignes
- wc -w : nombre de mots
- wc -c : nombres de caractères
- ls | wc -l : donne le nombre de fichier du dossier courant

more/less

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:

find

cut

- sélectionner certaines colonnes

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]

commandes qui ne lisent pas leur entrée standard

- ls, who, find
- chmod, cp, mv, rm, ln, mkdir
- date
- kill
- file, type
- echo

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

Bilan

- A la fin de cette deuxième séance, vous devez :
  - savoir ce qu'il un PID, PPID
  - comprendre et savoir utiliser sur des exemples de base les redirections et les enchaînements de commandes
  - comprendre ce qu'est un filtre
  - connaître quelques filtres de base

Scripts shell

- fichier texte contenant des commandes
- nouvelle commande

Séance No 3

- résumé séance 2 (processus, redirection et enchaînement de commandes, quelques filtres classiques)
- premiers scripts shell
- variables

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 »

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

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
```

•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 ?

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

Exemple (suite)

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

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

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 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

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
```

Bilan séance No 3

- A la fin de cette séance No 3, 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

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