SYSTÈME D'EXPLOITATION UNIX PROGRAMMATION SCRIPTS

Dr. Fatimazahra EL BIACH
Dr. Siham LAMZABI
Dr. Mohammed BELATAR

2024

Plan du semestre

Semaine 1 : Programmation par scripts

Semaine 2 : Programmation par scripts (suite)

Semaine 3 : Mécanismes de base

Semaine 4 : Fonctionnement en interactif

Semaine 5 : Shell-scripts portables

Semaine 6 : Mécanismes complémentaires

Semaine 7 : Mécanismes complémentaires (suite)

Semaine 8 : Evaluation

Semaine 9 : Automatisation des tâches

Semaine 10 : Automatisation des tâches (suite)

Semaine 11 : Automatisation des tâches (suite)

Semaine 12 : Manipulation des fichiers et les données

Semaine 13 : Manipulation des fichiers et les données (suite)

Semaine 14 : Langage AWK

Semaine 15 : Langage AWK (suite)

Semaine 16 : Evaluation

Introduction et Rappels

semaine 1

Shell, c’est quoi exactement?

- Le Shell : Il s’agit d’une interface texte entre l’utilisateur et le système informatique
  - Tout se fait au clavier
  - Pas de clic de souris
- L’utilisateur tape des commandes qui sont exécutées par le système
  - Le shell est donc un « interpréteur de commandes »
  - Chaque commande a une syntaxe particulière
  - Il existe des milliers de commandes différentes
- Les commandes peuvent aussi provenir d’un fichier
  - Le fichier contient les commandes à exécuter
  - L’utilisateur appel le fichier plutôt que de taper toutes les commandes
    - Utile pour les tâches répétitives

Pourquoi le shell?

- Le shell reste le moyen le plus efficace pour contrôler le système. C’est aussi le plus utilisé sous Linux/Unix.
- Le shell est un véritable environnement de programmation
  - Variables, boucles, structures de contrôle « if »
  - Programmes
- Les programmes écrits pour le shell sont interprétés au moment de l’exécution
  - Aucune compilation préalable n’est nécessaire
  - On peut profiter de différents outils développés sous différents langages
  - Les performances n’égalent pas un programme en C
- Contrôler une machine sans y avoir un accès physique (Serveur)

Rappel : langages Compilés vs Interprétés

Leurs principales différences sont les suivantes :

- Après l'écriture d'un fichier script, il est possible de le soumettre directement à l'interpréteur de commandes, tandis qu'un code source écrit en langage compilé doit être traduit en instructions de code machine compréhensibles pour le processeur.
- Le code compilé étant directement compris par le processeur du système, son exécution est très rapide, alors qu'un script doit être interprété dynamiquement, ce qui ralentit sensiblement l'exécution.
- Le fichier exécutable issu d'une compilation est souvent volumineux et n'est utilisable que sur un seul type de processeur et un seul système d'exploitation. À l'inverse, un fichier script est généralement assez réduit et directement portable sur d'autres processeurs ou d'autres systèmes d'exploitation du moment que l'interpréteur de commandes correspondant soit disponible.
- Un fichier compilé est incompréhensible par un lecteur humain. Il n'est pas possible d'en retrouver le code source. Cela peut garantir le secret commercial d'un logiciel. Inversement, un fichier script est directement lisible et modifiable, et peut contenir sa propre documentation sous forme de commentaires.

Les scripts shell

Script: Un script est un fichier contenant une série d'ordres que l'on va soumettre à un programme externe pour qu'il les exécute. Ce programme est appelé interpréteur de commandes. Il existe de nombreux interpréteurs de commandes. Naturellement, le shell en fait partie, tout comme certains outils tels que Sed et Awk que nous verrons ultérieurement, ainsi que d'autres langages tels que Perl, Python, Tcl/Tk, Ruby, etc. Ces langages sont dits interprétés, par opposition aux langages compilés (comme C, C++, Fortran, Ada, etc.)

Rappel des principes de base

- Super utilisateur : root
  - s’occupe de l’administration du système UNIX (installation des logiciels, création des profils utilisateurs, sauvegarde et restauration des données etc...)
- Hôte (serveur) :
  - système centralisé sur lequel peuvent se connecter les utilisateurs
- Console (client) :
  - un écran, le plus souvent noir, destiné à recevoir des commandes Shell via un clavier, branchée directement à la machine Hôte
- Terminal :
  - Environnement d’entrée/sortie de texte permettant l’émulation d’une console

Rappel des principes de base

- Voir Annexe : liste des commandes de base
- Caractères de contrôle :
  - <cr> fin de ligne (retour chariot)
  - <tab> tabulation
  - <bs> ^H, backspace, effacement du caractère précédent
  - <del> ^?, souvent identique à <bs>
  - <^C> interruption d'un processus attaché au terminal (SIGINT/2)
  - <^\> arrêt / quit (SIGQUIT/3)
  - <^Z> suspension d'un processus en premier plan (SIGSTOP/19)
  - <^U> effacement de la ligne complète
  - <^W> effacement du mot qui précède
  - <^D> fin de fichier => si le shell lit, fin d'un shell
  - <^S> suspension de l'affichage écran (Xoff)
  - <^Q> reprise de l'affichage écran (Xon)

```
$ stty -a
```

Rappel des principes de base

- le “prompt” (invite de commandes): variable PS1
  - $ en mode utilisateur
  - # en mode super-utilisateur
- Prompt secondaire : variable PS2
  - > en mode interactif
- Chemin de recherche des noms de commandes : variable PATH
- Structure d’une commande
  - une commande simple est une séquence de mots séparés par un séparateur blanc et de redirections. Le premier mot désigne le nom de la commande à exécuter, les mots suivants sont passés en arguments à la commande.
  - la valeur retournée d'une commande est celle de son exit

Rappel des principes de base

- Les commandes :
  - syntaxe : <commande> <liste d'arguments>
  - sensibilité à la casse (majuscules/minuscules)
  - importance du séparateur “espace”
- le manuel (help) :
  - man <commande Unix>
- Historique :
  - commande “fc”
  - commande “history”
- Nettoyage de de l'écran
  - commande “clear”

Rappel des principes de base

- Enchaînement de commandes:
  - Commande simple en arrière plan
    - $ cmd [arg ...] &
  - Pipeline (tube de communication entre commandes)
    - $ cmd1 [arg ...] | cmd2 [arg ...] | …
  - Exécuter la commande suivante en fonction de l'exit de la première
    - $ cmd1 [arg ...] || cmd2 [arg ...]
    - $ cmd1 [arg ...] && cmd2 [arg ...] ...
  - Séquencement de commandes successives en premier plan
    - $ cmd1 [arg ...]; cmd2 [arg ...]; cmd3 …
  - Groupement des commandes : exécution en sous-shell
    - $ (cmd1 [arg ...]; cmd2 [arg ...]; cmd3 …)
  - Groupement des commandes : exécution dans le shell courant
    - $ { cmd1 [arg ...]; cmd2 [arg ...]; cmd3 …;}

Les scripts shell : exemple

Un ensemble de commandes, d'outils et de programmes rassemblés dans un seul fichier dans une structure particulière permettant d'accomplir une ou plusieurs tâches

- Exemple :

```
# Test de l'existence du fichier
if [ -f $logfile ]
then
  rm $logfile
  echo effacement de ipscan.log effectue
fi

echo scan des ip...
```

Choix du shell

- Plusieurs versions du shell:
  - sh (Bourne Shell) : créé par Stephen R. Bourne en 1977
  - csh (C Shell) : créé par Bill Joy en 1979
  - tcsh (C Shell amélioré) : créé par Ken Greer en 1979
  - ksh (Korn Shell) : créé par David Korn en 1982
  - bash (Bourne again shell) : créé par Brian Fox en 1987
  - zsh (Z Shell) : créé par Paul Falstad en 1990
- Possibilité de choisir
  - 1 exemplaire particulier du Shell par utilisateur
  - Il peut lancer plusieurs Shells sur sa session
- Sous Linux, on peut choisir son shell (commande chsh)
  - Le shell bash domine le marché depuis longtemps

Pourquoi écrire un script shell ?

- Exemples de domaines d'utilisation :
  - lancer quelques commandes disposant d'options complexes, et qu'on désire employer régulièrement sans avoir à se reporter sans cesse à leur documentation.
  - scripts d'initialisation (de boot) du système, permettent d'énumérer les périphériques disponibles, et de les initialiser, de préparer les systèmes de fichiers, les partitions, et de configurer les communications et les services réseau.
  - préparation ou installation de certaines applications spécifiques (logiciels) nécessitent un paramétrage complexe.
  - tâches automatisées pour vérifier l'état du système (mémoire, espace disque, antivirus...) et maintenance périodique.
  - exploiter la sortie de certaines commandes et programmes, et les traces des scripts Shell enregistrées dans des fichiers de journalisation

TD 1

1. Affichez les shells disponibles sur votre système
2. Affichez votre shell actuel
3. Affichez votre shell par défaut
4. Basculez d’un shell à un autre puis fermer chaque shell ouvert
5. Changez votre shell par défaut
6. Vérifiez la disponibilité des éditeurs de texte et familiarisez-vous avec l’un d’eux

TP 1

1. En utilisant les commandes tail et head, écrire un script qui permet de réordonner les lignes d'un fichier "losange" contenant un losange dont les lignes sont inversées.
2. Re-écrivez votre script en une seule ligne.

Programmation shell scripts

semaine 2

Structure d’un script

1. spécifier l'interpréteur
2. vérifier la syntaxe
3. signaler le lancement
4. effectuer la/les tâches
5. nettoyer l’environnement
6. signaler la fin de l'exécution

```
#!/bin/bash
[ $# -ne 1 ] && echo "Usage : `basename $0` int" 1>&2 && exit 1
echo debut du travail
for ((i=0; i<$1; i++)); do
    for ((j=0; j<$1; j++)); do
        echo -n '*'
    done
    echo -en '\n'
done
echo fin

# N.B : les variables spéciales seront étudiées plus tard dans ce cours
# N.B : ces étapes ne sont pas toutes obligatoires
```

Ligne shebang

Pensez à insérer la ligne shebang en début de script pour indiquer le nom de l'interpréteur à employer. En général on appellera :

- #!/bin/sh
- #!/bin/bash

Selon le contexte d'utilisation, ou sur certains systèmes (Solaris par exemple), on fera appel à d'autres shells:

- #!/bin/ksh
- #!/bin/zsh

Commentaires

```
# une ligne de commentaire
echo bonjour # ou après une commande
: pseudo commentaire, toujours vrai
```

Les commentaires sont indispensables pour assurer une lisibilité correcte d'un script shell. La syntaxe du shell est parfois complexe et les lignes de commande incluant des appels à grep , sed ou awk se révèlent souvent de véritables défis pour le lecteur, même expérimenté.

Il est donc important de bien commenter un script, en spécifiant le but/rôle d'un ensemble de lignes.

Cela simplifie nettement la relecture ultérieure et les opérations de maintenance sur votre programme.

Commentaires : entête

Pour faciliter la maintenance d'un script, il est intéressant d'insérer quelques lignes de commentaire formant un en-tête, contenant :

- Le nom du script
- Quelques lignes décrivant le rôle et l'utilisation du script
- Le nom de l'auteur et son adresse e-mail, surtout si le programme est destiné à être diffusé
- Un historique des changements

```
###################################
# monscript.sh
# Permet de faire une synthèse des
# différents fichiers log du serveur Apache

# Auteur : Prénom NOM <prenom.nom@hote.com>

# Historique :
# 18/01/2017 (1.3) :
# correction bug sur longueur des champs
# 14/01/2017 (1.2) :
# ajout option -f
# 09/01/2017 (1.1) :
# inversion de l'option -i
# 07/01/2017 (1.0) :
# écriture initiale
###################################
# VERSION=1.3
```

Indentation

Les programmeurs débutants ont parfois du mal à savoir comment décaler leurs lignes de code de la marge.
Il s'agit pourtant d'une très bonne habitude à prendre, qui devient vite un automatisme et améliore la lisibilité du script.
L'organisation logique du script doit apparaître au premier regard ; le début et la fin de chaque structure de contrôle (boucle, condition, fonction...) étant mis en relief par le décalage du code contenu dans la structure.

Affichage des messages

- Nous avons déjà largement utilisé la commande echo qui permet à un script d’envoyer un message sur la sortie standard. C’est une commande interne du shell, mais il en existe souvent un équivalent sous forme d'un exécutable binaire /bin/echo . Les principales options reconnues par la version Gnu (Linux) de “echo” et de son implémentation interne dans Bash sont :
  - -n : Ne pas effectuer de saut de ligne après l’affichage. Cela permet de juxtaposer des messages lors d’appels successifs, ou d’afficher un symbole invitant l’utilisateur à une saisie.
  - -e : interpréter les caractères spéciaux (\n,\r,\\,...)

Affichage avec “echo” : caractères spéciaux

- \\ : Affichage du caractère \
- \XXX : Affichage du caractère dont le code Ascii est XXX exprimé en octal
- \a : Beep
- \c : Éliminer le saut de ligne final (comme l’option -n)
- \n : Saut de ligne
- \r : Retour chariot en début de ligne
- \b : Retour en arrière d’un caractère
- \t : Tabulation horizontale
- \v : Tabulation verticale

Affichage formaté : printf

Permet d'afficher des données formatées. Cette commande est une sorte de "echo" nettement amélioré proposant des formats pour afficher les nombres réels. Les programmeurs C reconnaîtront une implémentation en ligne de commande de la célèbre fonction de la bibliothèque stdio.h

- $ printf chaine arg1 arg2 arg3…

Exemples de formats (voir tableau) :

%20s	Affichage d'une chaine (string) sur 20 positions avec cadrage à droite
%-20s	Affichage d'une chaine (string) sur 20 positions avec cadrage à gauche
%3d	Affichage d'un entier (décimal) sur 3 positions avec cadrage à droite
%03d	Affichage d'un entier (décimal) sur 3 positions avec cadrage à droite et complété avec des 0 à gauche
%-3d	Affichage d'un entier (décimal) sur 3 positions avec cadrage à gauche
%+3d	Affichage d'un entier (décimal) sur 3 positions avec cadrage à droite et affichage systématique du signe
%10.2f	Affichage d'un nombre flottant sur 10 positions dont 2 décimales
%+010.2f	Affichage d'un nombre flottant sur 10 positions dont 2 décimales, complété par des 0 à gauche, avec cadrage à droite et affichage systématique du signe

Exécution d’un shell script

- Par invocation de l'interpréteur :
  - $ bash essai.sh
  - $. essai.sh
  - $ source essai.sh
- Par appel direct :
  - $ essai
    - Déplacer le script dans l’un des répertoires de PATH
    - Ajouter le répertoir du script à la variable PATH
    - Ajouter . à la variable PATH (déconseillé)
      - $ PATH=$PATH:.
    - Indiquer le chemin du répertoire ou se trouve le script:
      - ./essai
  - Il faut rendre le script exécutable : $ chmod +x essai

Remarque :

le suffixe .sh ajouté à la fin du nom du script **n’a aucune importance**; il ne s’agit que d’une information pour l’utilisateur. Le système Unix ne tient aucun compte des suffixes éventuels des noms de fichiers.

TD/TP 2

1. Quel est le résultat de la commande suivante :
   - $ # echo bonjour
2. Avec une seule commande d'affichage, affichez sur deux lignes le message suivant:
   - Bonne
   - Journée
3. Écrire un script qui affiche la date en permanence, chaque écriture écrasant la précédente.
4. Écrire un script qui affiche sous forme de tableau dix étudiants, et leur classement à l'ordre inverse (en utilisant printf et des espaces).
   - Etudiant1 10
   - Etudiant2 9
   - ….
   - Etudiant10 1

Mécanismes de base

semaine 3

Personnalisation du shell

- Fichier de profile :
  - Emplacement : /etc/profile
  - Standard POSIX
  - Point de passage obligatoire après un login
  - Fournit par le système, modifiable par l'administrateur
  - Permet de configurer le Shell (couleurs, terminal, variables PATH, PS1…)
  - Il concerne tous les utilisateurs
  - Il peut avoir des compléments (ex. dans /etc/profile.d/)
  - Il peut faire appel (sourcing) à d'autres compléments shell spécifiques (ex. /etc/bash*)

Personnalisation du shell

Autres scripts d’initialisation :

- bash:
  - ~/.profile, ~/.bash_profile: initialisations en tant que shell de connexion
  - ~/.bashrc : à l’invocation du shell (sans fonction de connexion)
  - ~/.bash_logout : quand l’utilisateur se déconnecte du système (nettoyage de l’environnement de travail)
- Bourne shell, Korn shell & POSIX: ~/.profile
- C-shell: ~/.login, ~/.cshrc, ~/.logout
- Zsh: ~/.zshrc
- T-csh: ~/.tcshrc
- Modifiables par l’utilisateur : Personnalisation du shell
- Peuvent être utilisés pour:
  - Redéfinir les variables de /etc/profile
  - Définir de nouvelles variables
  - Définir des fonctions
  - Personnaliser l’apparence (PS1…)
  - Définition d’alias pour des commandes complexes

Couleurs de texts (terminal compatible)

- Afficher des message en couleurs
- Redéfinir les variables $PS1 et $PS2
- Coloriage du texte par la séquence :
  \033[XXm
    - XX est le code de la couleur
    - Le code couleur « 0 » ramène à la couleur par défaut
  - ex. `echo -e "\033[31mUn texte en rouge et\033[0m retour a la normale"`
    - Les codes de couleurs peuvent se cumuler :
  - `$ echo -e "\033[31m\033[44mHello world rouge sur fond bleu\033[0m"`
    - On peut cumuler plusieurs couleurs/effets séparés par ;
  - `$ echo -e "\033[31;44;1;4;5;7mLe texte\033[0m"`
- le caractère \033 peut être remplacé par \e
- On peut utiliser n'importe quelle couleur RGB avec
    - `\e[38;2;R;G;Bm` `#RGB foreground color`
    - `\e[48;2;R;G;Bm` `#RGB background color`

Couleur Text		Couleur du fond	
30	Noir	40	Noir
31	Rouge	41	Rouge
32	Vert	42	Vert
33	Orange	43	Orange
34	Bleu	44	Bleu
35	Magenta	45	Magenta
36	Cyan	46	Cyan
37	Blanc	47	Blanc

Nombre	Effet
01	**Gras**
04	Sous-ligné
05	Clignotant
07	Sur-ligné/ Inversé

Personnalisation du prompt

———

- On peut utiliser des caractères spéciaux

```
\d  la date actuelle au format
"Weekday Month Date" ("Tue May 26")
\h  le nom de l'ordinateur
\H  le nom complet de l'ordinateur
\n  saut de ligne
\s  nom du shell ("bash")
\[ démarre une séquence de
caractères non imprimable
(couleurs…),
\] ferme une séquence non
imprimable
```

```
\u  le nom de login de l'utilisateur
courant
\v  la version du bash ("2.00")
\V  la version release du bash, version
+ patchlevel ("2.00.0")
\W  le répertoire de travail actuel
\w  le répertoire de travail actuel
depuis la racine
\!  numéro de commande courant
\@  heure actuelle au format 12h am/pm
\T  heure actuelle au format 12h
HH:MM:SS
\A  heure actuelle au format 24h HH:MM
\t  heure actuelle au format 24h
HH:MM:SS
\$  si l'UID=0 affiche # sinon $
……
```

Les variables

- Types de variables :
  - Variables globales :
    - Le shell en fournit une copie à sa descendance
    - Dans les scripts d'initialisation
      - appelées "Variables d'environnement"
      - commande "env" pour les lister
    - Commande export pour les déclarer (setenv sous csh)
  - Variables locales
    - Créées par l'utilisateur dans le shell actuel
    - Utilisables dans l'environnement actuel
    - Liste de toutes les variables disponibles dans le shell actuel (locales + environnement) : commande "set"
  - Variables de substitution (variables spéciales)
    - Variables positionnelles $1, $2 ...
    - Autres variables spéciales

Les variables

- Déclaration :
  - variable=valeur
  - declare variable=valeur (bash v.2+)
  - typeset variable=valeur (bash, ksh..)
- Contenu d'une variable :
  - $variable
  - ${variable}
- Suppression d'une variable:
  - unset variable

- Exemple :

```
$ nom=ALI
$ adresse="2 bd de France"
$ echo $nom
ALI
$ echo $adresse
2 bd de France
$ unset adresse
$ set
HOME=/home/ali
PATH=/bin:/usr/bin
PS1=$
nom=ALI
```

Les variables : exemples

```
---
$ fleur=rose
$ echo "une $fleur, des $fleurs"
une rose, des
$ echo "une $fleur, des ${fleur}s"
une rose, des roses
$ transport="air mer terre"
$ echo $transport
air mer terre
$ unset fleur transport
$ echo "fleur = $fleur et transport = $transport"

fleur = et transport =
```

Les variables (suite)

- Une variable peut être déclarée en lecture seule :
  - `$ readonly variable=valeur`
- Le contenu d'une variable peut être directement lu depuis l'entrée standard (clavier) :
  - `$ read variable`
- Si vous avez des espaces dans la valeur à affecter :
  - employez des guillemets droits : `$ variable="Bonjour Monsieur"`
  - employez des quotes (apostrophes) : `$ variable='Bonjour Monsieur'`
- Les simples quotes groupent les mots et suppriment toute évaluation, les guillemets permettent l'évaluation :
  - `$ nom="Kamel" ; echo 'Bonjour Monsieur $nom'` ⇒ `Bonjour Monsieur $nom`

Les tableaux (bash)

- tab[i]=valeur : définir un emplacement mémoire pour la i^{eme} case du tableau, et la remplir avec valeur.
- ${tab[i]} : pour consulter la valeur de la i^{eme} case du tableau.
  - Les accolades sont obligatoires pour éviter les ambiguïtés avec ${tableau}[i]
  - ${tableau[0]} est identique à $tableau . N'importe quelle variable peut être considérée comme le rang zéro d'un tableau qui ne contient qu'une case.

- On peut initialiser le tableau entier en une seule expression :
  - `$ tab=( zero un deux trois quatre )`
  - `$ tab=( "fry" "leela" [42]="bender" "flexo" )`
  - `$ tab=( ['un']="one" ['deux']="two" ['trois']="three" )`
- L'expression ${tab[@]} fournit une liste de tous les membres du tableau.
- ${!tab[@]} fournit la liste des clés du tableau
- ${#tab[i]} fournit la longueur du ieme membre du tableau
- ${#tab[@]} , donne le nombre de membres du tableau

Variable vide vs variable non déclarée

- Par défaut, une variable qui n’a jamais été affectée est traitée par défaut comme une chaîne vide :
  - `$ echo '-'$inex'-'`
  - `--`
- Une variable à laquelle on affecte une chaîne vide existe quand même. Ce n’est pas comme si on la supprime avec “unset”
- On peut différencier les variables inexistantes des variables vides en utilisant l’option -u à l'exécution du shell (ou avec la commande set) :
  - `$ set -u`
  - `$ echo $inexistante`
  - `bash: inexistante : variable sans liaison`

TD/TP 3

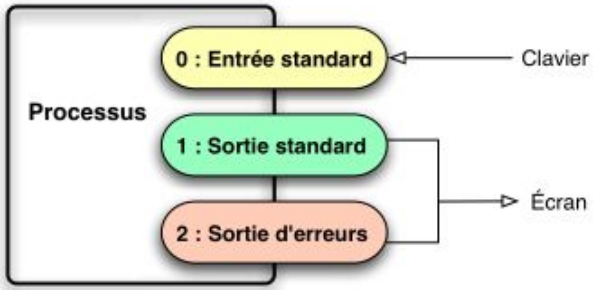
1. Identifiez les fichiers d’initialisation de sessions propres à votre compte.
2. Créez un alias permanent pour remplacer la commande “ls -la” par “ls”
3. Écrivez un script qui compte à rebours de 5 à 0, avec une pause d’une seconde à chaque affichage. (en rouge)
4. Explorez la variable PS1 et redéfinissez de manière permanente votre propre prompte en utilisant des couleurs et en affichant les informations suivantes :

<user>@<host>/<pwd>_<time>_$_

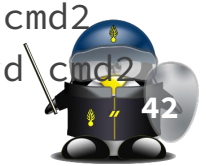
Fonctionnement interactif

semaine 4

Rappels



- Redirection des entrées/sorties (voir cours 1e semestre)
  - >,1>,2>,>>,1>>,2>>,>&2,1>&2,2>&1
  - <,0< (alimenter une commande depuis un fichier)
  - <<, 0<< (saisie de document en ligne)
- De manière générale :
  - n< , n> , et n>> , dans lesquelles n représente un numéro de descripteur de fichier (par défaut, 0 pour < et << , et 1 pour > et >>).
    - redirection des deux sorties : &> et &>>
  - n>&m , où n et m sont deux numéros de descripteurs, la sortie vers n sera redirigée vers m
- Les pipes “tubes” (voir cours 1e semestre)
  - opérateur | redirige sortie standard de cmd1 vers entrée standard cmd2
  - opérateur |& redirige les deux sorties de cmd1 vers entrée standard cmd2



Rappels: Substitution de commande

- Syntaxe : `commande` ou $(commande)
- Exemple :
  - $ echo "date actuelle = `date`"
  - date actuelle = lundi 27 mars 2017, 16:30:38 (UTC+0100)
  - $ echo "date actuelle = $(date)"
  - date actuelle = lundi 27 mars 2017, 16:30:38 (UTC+0100)
  - $ echo "Nombre de fichiers dans le répertoire: `ls | wc -w`"
  - Nombre de fichiers dans ce répertoire: 24

Extraction de sous-chaînes

- L'opérateur ${} est la version généralisée de l'opérateur $ permettant l'accès à la valeur d'une variable
- L'option ":" de l'opérateur ${} permet d'extraire une sous-chaîne d'une variable
  - La syntaxe est: ${variable:debut:longueur}
  - ATTENTION à la portabilité! n'existe pas dans SUS3.
- L'option "#" permet de supprimer le plus court préfixe correspondant au motif qui vient après # :
  - syntaxe : ${variable#motif}

```
$ variable=ABCDEFGHIJKLMNOPQRSTUVWXYZ
$ echo ${variable:5:2}
FG
$ echo ${variable:20}
UVWXYZ
```

```
$ variable=AZERTYUIOPAZERTYUIOP
$ echo ${variable#AZE}
RTYUIOPAZERTYUIOP
$ echo ${variable#*T}
YUIOPAZERTYUIOP
$ echo ${variable#*[MNOP]}
PAZERTYUIOP
```

Extraction de sous-chaînes

- L'expression `${variable##motif}` sert à éliminer le plus long préfixe correspondant au motif transmis.
- Symétriquement, les expressions `${variable%motif}` et `${variable%%motif}` correspondent au contenu de la variable, débarrassée respectivement, du plus court et du plus long suffixe correspondant au motif indiqué.

```
$ variable=AZERTYUIOPAZERTYUIOP

$ echo ${variable##*T}
YUIOP
$ echo ${variable##*[MNOP]}


$ echo ${variable%IOP*}
AZERTYUIOPAZERTYU
$ echo ${variable%%IOP*}
AZERTYU
$ echo ${variable%[X-Z]*}
AZERTYUIOPAZERT
$ echo ${variable%%[X-Z]*}
A
```

Remplacement de sous-chaînes

- ${variable/motif/remplacement} permet de remplacer la première occurrence du motif par la chaîne de remplacement.
- ${variable//motif/remplacement} permet de remplacer toutes les occurrences du motif par la chaîne de remplacement.
- ATTENTION! Cet opérateur est présent dans les shells Bash, Korn 93 et Zsh, mais n’est pas normalisé par SUS3; Aussi on risque d’avoir de légères divergences entre les différentes implémentations

Les variables : longueur du contenu

- ${#variable} permet d'avoir la longueur du contenu de la variable "variable"
- Si le contenu est un nombre, il est traité de la même manière qu'une chaîne de caractères

```
$ variable=AZERTYUIOP
$ echo ${#variable}
10

$ echo $UID
1000
$ echo ${#UID}
4

$ variable=
$ echo ${#variable}
0
$ echo ${#inexistante}
0
```

Les variables : Actions par défaut

Lorsqu'un script fonde son comportement sur des variables qui sont fournies par l'utilisateur, il est important de prévoir une action par défaut si la variable est vide ou n'existe pas.

- L'expression ${variable:-valeur} prend la valeur indiquée à droite du modificateur :- si la variable n'existe pas, ou si elle est vide. Si la variable existe et elle est non vide, l'expression renvoie le contenu de la variable.

```
$ variable=existante
$ echo ${variable:-defaut}
existante
$ variable=
$ echo ${variable:-defaut}
defaut
unset variable
$ echo ${variable:-defaut}
defaut
```

Les variables : Actions par défaut

- L'expression ${variable:=valeur} est similaire à la précédente, mais le contenu de variable sera modifié. Si la variable n'existe pas ou si elle est vide, elle est alors remplie avec valeur. Ensuite, dans tous les cas, le contenu de la variable est retourné.
- L'expression ${variable:?message} : Si la variable est définie et non vide, sa valeur est retournée. Sinon, le shell affiche le message fourni après le point d'interrogation, et abandonne le script ou la fonction en cours.
  - Si le message n'est pas précisé, le shell en affiche un par défaut

```
$ echo $vide
$ echo ${vide:=contenu}
contenu
$ echo $vide
contenu

$ unset vide
$ echo ${vide:?faut la
déclarer}
vide : faut la déclarer

$ echo ${vide:?}
bash: vide : paramètre vide ou
non défini
```

Les variables : Actions par défaut

- ${variable:+valeur} renvoie la valeur fournie à droite du symbole + si la variable est définie et non vide, sinon elle renvoie une chaîne vide
- Les quatre options précédentes de l'opérateur ${} considèrent au même titre les variables indéfinies et les variables contenant une chaîne vide. Il existe quatre modificateurs similaires qui n'agissent que si la variable est vraiment indéfinie ; il s'agit de ${ - } , ${ = } , ${ ? } , et ${ + }

```
$ existante=4
$ echo ${existante:+1}
1
$ echo ${inexistante:+1}

$ var=
$ definie=${var:+oui}
$ : ${definie:=non}
$ echo $definie
non
$ var=1
$ definie=${var:+oui}
$ : ${definie:=non}
$ echo $definie
oui
```

Variables de substitution

- Elles sont définies implicitement et peuvent être utilisées à tout moment dans le script
- Les variables positionnelles et les variables spéciales sont utilisées pour accéder aux informations qui sont fournies sur la ligne de commande lors de l'invocation d'un script (paramètres), mais aussi aux arguments transmis à une fonction. Les arguments sont placés dans des paramètres qui peuvent être consultés avec la syntaxe $1 , $2 , $3 , etc.
- L'affectation est impossible, on ne peut pas imaginer une affectation : 1=2!

Variables positionnelles

- Le premier argument transmis est accessible en lisant $1, le deuxième dans $2 , et ainsi de suite.
- Pour consulter le contenu d'un paramètre qui comporte plus d'un chiffre, il faut l'encadrer par des accolades ${10} (Bash)
- L'argument numéro zéro ($0) contient le nom du script tel qu'il a été invoqué
- Les variables positionnelles peuvent être modifiées :
  - avec la commande set :
    - $ set a b c
    - ATTENTION! l'ensemble des variables positionnelles est affecté!
  - avec la commande shift
    - Si suivie d'un nombre, ce dernier représente le nombre de décalages désirés

Variables positionnelles : exemples

```
$ set a b c
$ echo $1 $2 $3
a b c
$ set d e
$ echo $1 $2 $3
d e
```

```
#!/bin/bash
while [ -n "$1" ] ; do
echo $1
shift
done
```

```
$ set a b c d e
$ echo $0 $1 $2 $3
bash a b c
$ shift
$ echo $0 $1 $2 $3
bash b c d
$ shift
$ echo $0 $1 $2 $3
bash c d e
$ shift
$ echo $0 $1 $2 $3
bash d e
```

Variables spéciales

- $# : nombre d'arguments, sans compter le paramètre $0
- $* : liste de tous les arguments
- $@ : liste de tous les arguments (supporte les guillemets, la plus utilisée)
- Toutes ces variables sont modifiés par l'appel de shift et de set
- $? : code de sortie retourné par la dernière commande
- $$: numéro de processus du shell actuel
- $! : numéro du dernier processus en arrière plan
- $- drapeaux fournis au shell par set (options du shell)

TD/TP 4

1. Écrire un script qui affiche le répertoire de travail en cours, en remplaçant le répertoire personnel par le symbole ~ en vert (préfixe)
2. Écrire un script qui prend comme argument une adresse e-mail et affiche le nom de login correspondant.
3. En utilisant la commande mv, changez l'extension de tous les fichiers d'un répertoire donné en 1e argument, d'une extension donnée en 2e argument à une autre en 3e argument.

TD/TP 4

(suite 1)

4. Vous souhaitez écrire un script qui prend obligatoirement un paramètre d’une longueur de 5 caractères minimum. Écrire une expression qui permet de vérifier la syntaxe. En cas d’erreur, elle doit afficher la syntaxe correcte et quitter avec un code 1.

Portabilité des scripts

Portabilité des scripts

- Une norme de l'IEEE : Posix
- Un standard de l'Open Group : SUS (Single Unix Specification)
- La version 3 de ce standard (abrégé habituellement en SUSv3) a intégré le contenu de la norme POSIX.
- pour plus d'info :
  - http://www.opengroup.org/standards/unix

Portabilité des scripts

- Un grand nombre de shells et de langages de scripts semble converger vers le standard POSIX.
- On peut appeler Bash avec l'option --posix ou insérer un set -o posix au début d'un script pour que Bash se conforme très étroitement à ce standard.
- Une autre alternative consiste à utiliser, dans le script, l'en-tête #!/bin/sh plutôt que #!/bin/bash pour désactiver les fonctionnalités étendues de Bash.

Portabilité des scripts

- La plupart des scripts Bash fonctionnent directement avec ksh, et vice-versa, car Chet Ramey a beaucoup travaillé sur le portage des fonctionnalités du Kourn aux dernières versions de Bash.
- Sur un UNIX commercial, les scripts utilisant les fonctionnalités spécifiques aux commandes standards GNU peuvent ne pas fonctionner. Ceci devient de moins en moins un problème de nos jours car les outils GNU ont petit à petit remplacé les versions propriétaires même sur les UNIX dits “solides” (commerciaux).

Mécanismes complémentaires

Options du shell

- La commande set
  - set -x : Active le mode “trace”
  - set - : désactive le mode “trace”
  - set -e : quitter immédiatement si le retour d’une commande est différent de 0
  - set -C : interdire l’écrasement de fichiers existants par redirection
  - set -o option : activer certaines options du shell
    - exemple: $ set -o posix
- Désactiver l’option :
  - set +<l’option>
  - ex: $ set +x

- Options du shell :
  - $ bash -x monscript # exécution du script en mode trace
    - affiche les étapes de substitution, signalées par la ligne +
  - $ bash -v monscript # exécution en mode “bavard”
    - affiche la ligne telle qu’elle est écrite dans le script, avant que le shell ne l’interprète
  - $ bash -n monscript
    - vérification de la syntaxe
  - $ bash -e monscript
    - provoque la fin d’exécution d’un script si une commande se termine avec un code de retour différent de zéro

Les options d'un script

- On peut lire les options d'un script en tant que liste d'arguments avec les variables de substitution, mais il existe un moyen plus approprié pour supporter tous les formats des options:
- Commande *getopts* :
  - getopts ":opt1:opt2" variable
  - à chaque appel de cette commande on consomme une option dans $variable
  - $OPTARG contient la valeur de l'option

- Exemple :

```
#!/bin/bash
usage() {
  echo "Usage: `basename $0` -n <5-10> -m <chaîne de caractères>" 1>&2;
  exit 1;
}

while getopts ":n:m:" option; do
    case "${option}" in
        n)
            n=${OPTARG}
            ((n >= 5 && n <= 10)) || usage
            ;;
        m)
            m=${OPTARG}
            ;;
        -)
            [ ${OPTARG%%=*} = “help” ] && usage #option longue
            ;;
        :)
            echo "Error: -${OPTARG} nécessite un argument." 1>&2 && usage
            ;;
        *)
            usage
            ;;
    esac
done
shift $((OPTIND-1))
if [ -z "${n}" ] || [ -z "${m}" ]; then
    usage
fi
echo "n = ${n}"
echo "m = ${m}"
```

tee

- Cet utilitaire peut être imaginé comme un « T » de plomberie, d'où son nom, c'est-à-dire un accessoire que l'on insère dans un pipeline pour disposer d'une sortie annexe. tee lit les données sur son entrée et les copie sur sa sortie standard, tout en envoyant une copie supplémentaire dans tous les fichiers dont le nom lui est fourni en argument.
  - `$ ls | grep 'sh' | tee fichier1 | wc -l`
- Il est également possible d'utiliser le nom de fichier spécial /dev/tty pour obtenir un affichage à l'écran des données qui transitent par tee .

xargs

- Elle prend des lignes de texte sur son entrée standard, les regroupe pour les transmettre en argument à une autre commande. C’est le moyen le plus efficace de transformer des données arrivant dans un flux (de type pipeline ou autre) en arguments sur une ligne de commande.
- Exemple :
  - $ ls | xargs -n 1 grep ’bash’
- Exemple :
  - $ xargs -t -n 2 diff <<fin
    fic1 fic2 fic3
    fic4 fic5 fic6
    fin
    diff fic1 fic2
    diff fic3 fic4
    diff fic5 fic6
- Options principales :
  - -t pour avoir l'écho des commandes
  - -n pour découper le flux d'entrée en paquets

TD/TP 5

1. Écrire un script qui nécessite la saisie d’un nom de fichier complet en option -f (avec son chemin d’accès) et affiche successivement le nom seul et le chemin seul.
2. En utilisant xargs dans un script, cherchez le texte donné en premier argument du script dans tous les fichiers du répertoire donné en deuxième argument.

TD/TP 5

(suite)

3. Écrire un script bash qui prend des options -h ou --host et -p ou --port pour télécharger une page web en utilisant curl ou wget

Devoir: Écrire un script qui prend en options l'url d'un site Web et un port pour télécharger une copie du site entier avec pages HTML, CSS, JS, images …

Automatisation des scripts

Les tâches cron

- Cron est un gestionnaire de tâches
- Il permet de lancer l'exécution des tâches de manière régulière et répétitive.
- Chaque utilisateur peut avoir sa propre crontab, lui permettant de configurer les actions à effectuer régulièrement.
- La commande qui permet l’édition des tâches planifiées est : crontab
  - crontab -l : pour lister les tâches planifiées
  - crontab -e : pour éditer la tables des tâches planifiées

Crontab: syntaxe

- Une tâche planifiée dans un fichier de Cron est composée de 3 données différentes :
  - Sa période de répétition définie par 5 données différentes :
    - Les minutes (de 0 à 59)
    - Les heures (de 0 à 23)
    - Les jours dans le mois (de 1 à 31)
    - Les mois (de 1 à 12 ou `jan,feb,mar,apr…`)
    - Les jours de la semaine (de 0 à 7 ou `sun,mon,tue,wed,thu,fri,sat`)
  - L'utilisateur système sous lequel la tâche sera réalisée (cas crontab du root seulement);
  - La commande à réaliser ;
- Exemple :
  - `59 23 * * * /home/backup/backup.sh > /dev/null`

Tâches Cron : périodicité

- Pour chacune des 5 composantes de temps, les syntaxes suivantes sont utilisables :
  - `*` : pour chaque unité de temps ;
  - `5` : pour une unité de temps particulière, ici la cinquième ;
  - `5-10` : pour une intervalle, chaque unités entre 5 et 10 soit `5,6,7,8,9,10` ;
  - `*/5` : représente un cycle défini, ici toutes les toutes les 5 unités soit `0,5,10,15...` ;
  - `5,10` : représente une série définie, 5 et 10 ;
- Pour les mois et les jours de la semaine, on peut aussi utiliser les abréviations :
  - `jan,feb,mar,apr …`
  - `sun,mon,tue,wed,thu,fri,sat`

Exemples

min	heure	jour/mois	mois	jour/semaine	Périodicité
*	*	*	*	*	Toutes les minutes
30	0	1	1,6,12	*	à 00:30 le premier janvier, juin et décembre
0	20	*	10	1-5	à 20:00 chaque jour de la semaine sauf weekend (du lundi au vendredi) d'octobre
0	0	1,10,15	*	*	à minuit les premiers, dixièmes, et quinzième jours de chaque mois
5,10	0	10	*	1	à 00:05 et 00:10 chaque lundi et le 10 de chaque mois

Crontab : raccourcis

Il existe également certains raccourcis :

- @reboot ⇒ Au démarrage du système
- @yearly ⇒ Tous les ans ⇒ 0 0 1 1 *
- @annually ⇒ Tous les ans ⇒ 0 0 1 1 *
- @monthly ⇒ Tous les mois ⇒ 0 0 1 * *
- @weekly ⇒ Toutes les semaines ⇒ 0 0 * * 0
- @daily ⇒ Tous les jours ⇒0 0 * * *
- @midnight ⇒ Tous les jours⇒ 0 0 * * *
- @hourly ⇒ Toutes les heures⇒ 0 * * * *

Cron : notification

- Une fois une tâche cron est exécutée, sa sortie est envoyée par e-mail à l'administrateur du système
  - l'adresse e-mail de l'admin doit être mentionnée dans la variable MAILTO
  - l'utilitaire mailutils doit être installé sur le système
- Pour éviter qu'une tâche envoie un e-mail, on peut rediriger ses sorties vers un trous noir :
  - en ajoutant > /dev/null 2>&1 à la fin de la commande ;
  - en ajoutant &> /dev/null à la fin de la commande ;

TD/TP 6

1. Mettre en place un script qui, toutes les 5 minutes, ajoute la date, l'heure et le pourcentage de l’utilisation actuelle de la mémoire dans un fichier nommé "memoire.log". (utilisez la commande *free* pour récupérer les informations sur la mémoire)

Les services

C’est quoi un service

- Un service est un programme qui s'exécute en arrière-plan, plutôt contrôlé par le système d'exploitation que par l'utilisateur directement.
  - Aussi appelé “daemon”
  - Souvent avec un nom qui se termine par “d” (ex. sshd, mysqld …)
- La plupart des logiciels fonctionnent en mode serveur ont besoin d’installer un service

- **systemd** est le logiciel qui initialise le système d'exploitation au démarrage (dont le fameux processus *init* fait partie) et qui démarre et pilote les différents services de la machine.
- Les services sont gérés par Systemd, qui les démarre et les arrête en fonction du besoin.
- Dans le cas d'un serveur, il arrive que l'on ajoute des services à la machine (par exemple un serveur web, ou un SGBD) et qu'il soit nécessaire de les piloter avec systemctl.

Etats d’un service

- Les services ont des états :
  - **Enabled** : en cours d'exécution. Aucun problème.
  - **Disabled** : non actif, peut être démarré à tout moment.
  - **Masked** : ne fonctionnera que si nous lui enlevons cette propriété.
  - **Static** : ne sera utilisé qu'au cas où un autre service en aurait besoin.

- Pour lister les services (en tant que root):

```
$ systemctl list-units --type service
```

On peut manipuler les services avec les commandes suivantes :

- `systemctl start [service_name]` pour démarrer un service stoppé
- `sudo systemctl stop [service_name]` pour stopper un service
- `systemctl restart [service_name]` pour redémarrer un service, cela revient à faire un stop puis un start
- `systemctl reload [service_name]` pour demander à un service de recharger sa configuration sans s'arrêter
- `systemctl status [service_name]` pour vérifier l’état d’un service
- `systemctl enable [service_name]` pour que le service soit démarré au démarrage du système
- `systemctl disable [service_name]` pour qu’il ne soit pas lancé au démarrage

Configuration d’un service

- Les fichiers de configuration sont stockés dans /lib/systemd/system/
- On peut l’éditer en utilisant la commande suivante :
  - sudo systemctl edit nomduservice.service
- Le fichier du service stocke le nom, description, l’emplacement du fichier de configuration, les commandes à utiliser pour démarrer ou arrêter le service, et bien d’autres paramètres du service.
- Pour créer un nouveau service, il faut créer son fichier /lib/systemd/system/ puis exécuter les commandes suivantes :
  - systemctl daemon-reload
  - systemctl enable monservice.service
  - systemctl start monservice.service

```
[redacted] $ cat /lib/systemd/system/ssh.service
[Unit]
Description=OpenBSD Secure Shell server
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStartPre=/usr/sbin/sshd -t
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/usr/sbin/sshd -t
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=notify
RuntimeDirectory=sshd
RuntimeDirectoryMode=0755

[Install]
WantedBy=multi-user.target
Alias=sshd.service
```

La trace des services

- L’exécution des daemons est enregistrée dans les journaux systèmes.
- Lorsqu’un service ne s’exécute pas correctement, on peut consulter les journaux pour obtenir des informations.
- La commande **journalctl** sert à consulter les journaux (logs) systemd.
- Pour avoir la fin du journal systemd, il faut utiliser la commande journalctl suivante :
  - journalctl -xe

TD/TP 7

1. Vérifier les services existants sur votre système et leurs états
2. Vérifier le fichier de configuration d’un service existant.
3. Vous avez une application NodeJS que vous souhaitez lancer comme daemon. Créez le fichier du service et lancez-le.

Filtrage des données

Définition

- Un ensemble de commandes qui :
  - acceptent des données de l'entrée standard
  - effectuent des transformations sur ces données
  - dirigent ces données vers la sortie standard
  - affichent leurs messages d'erreur vers la sortie d'erreurs

- Filtres connus sous Unix :
  - tr
  - cut
  - sort
  - grep
  - paste
  - diff
  - sed
  - awk
- Tous ces filtres lisent donc leur données sur l'entrée standard et écrivent leurs résultats sur la sortie standard. On peut bien sûr les alimenter avec le contenu d'un fichier/pipe et/ou rediriger leur sortie vers un fichier/pipe

Le filtre tr

- Syntaxe : tr [options] chaine1 chaine2
- tr = translate characters
- fonction: substitution ou suppression des caractères sélectionnés
- Un caractère de chaine1 est remplacé par le caractère de même position dans chaine2
- Options principales:
  - -d : suppression des caractères sélectionnés
  - -s : traiter caractère répétitif comme caractère unique. ex. “aaaaa” traité comme “a”
  - -c : tous les caractères qui ne sont pas spécifiés dans la première chaîne sont filtrés (selon autre options -d ou -s)
  - voir manuel pour plus d’options...
- Abréviations :
  - [a-z] = segment de 26 caractères, allant de ’a’ à ’z’
  - [a*n] = a...a (n fois)
  - \xyz = désigne le caractère de code octal xyz

Le filtre tr (exemples)

- $ echo "coucou" | tr [a-z] [A-Z]
  - COUCOU
- $ echo "aaabbbaaa" | tr -s [a-z] [A-Z]
  - ABA
- $ tr -d '\r' < ms_dos_file.c > inix_file.c
  - suppression du CR : cas transfert fichier MS DOS vers Unix
- $ echo "ariane" | tr "ai" "ou"
  - oruone

Le filtre cut

- La commande cut permet d'afficher des zones spécifiques d'un fichier.
- options :
  - -c spécifie le numéro du caractère à extraire
  - -f spécifie le numéro du champs (field) à extraire
  - -d spécifie le délimiteur des éléments dans une ligne (default TAB)
- ex:
  - $ cut -c1 /etc/passwd # affichera le premier caractère du fichier /etc/passwd.
  - $ cut -d: -f6 /etc/passwd #affichera le 6eme champ du fichier /etc/passwd, dont le séparateur de champs est le caractère “:”
  - $ cut -f1,3 -d: /etc/passwd # extraire les éléments 1 et 3
  - $ cut -f1-3 -d: /etc/passwd # extraire les éléments de 1 à 3
  - $ cut -f4- -d: /etc/passwd # extraire les éléments à partir du 4eme

Le filtre sort

- Le programme sort permet de trier les lignes d'un fichier. Les caractères ‘+’ et ‘-’ permettent de spécifier de quelle colonne à quelle colonne le tri doit s'effectuer (1ere colonne pour 0, 2eme colonne pour 1...)
- exemples :
  - `$ sort -t: +1 -2 fichier.csv # t: utiliser le caractère ‘:’ comme séparateur`
  - `$ sort -t: +5.1 fichier # tri sur un caractère situé à la 2eme position du 6eme champ`
  - `$ sort -t ";" +0d -1 +2dr -3 fichier # tri multicritères`
- On peut combiner et trier plusieurs fichiers à la fois
  - `$ sort ... fichier1 fichier2 ..`

Le filtre sort : quelques options

- -b Saute les colonnes constituées de blancs (Ignore les espaces placés en début de champ).
- -d Trie de type dictionnaire (alphabétique).
- -n Trie par ordre numérique.
- -f Aucune différenciation n'est faite entre minuscules et majuscules.
- -r Trie inverse.
- -M Trie chronologiquement les mois.
- -t: Trie suivants les champs séparés par “:” (on peut spécifier un autre séparateur).

Le filtre paste

- Usage : paste fichier1 fichier2...
- concatène les lignes de même n dans fichier1 et fichier2
- fusion horizontale de deux fichiers ou plus
- possibilité de spécifier un séparateur avec l'option -d (par défaut TAB)

Le filtre grep

- filtre les lignes qui contiennent un texte (ou une expression régulière)
- Exemple : ls -l | grep “fi”
  - -v : afficher les lignes qui ne contiennent pas l'expression rationnelle.
  - -E : expression régulière étendue

Le filtre diff

- usage : diff fichier1 fichier2
- La commande diff donne les modifications à apporter au premier fichier spécifié pour qu'il ait le même contenu que le second. Les différences sont indiquées sous forme de codes. Exemples:
  - 3a4,7 indique que après la troisième ligne du premier fichier doivent être incrustées les lignes 4 à 7 du second.
  - 5,8d4 indique que les lignes 5 à 8 du premier fichier doivent être supprimées, car elles n'existent pas derrière la ligne 4 du second.
  - 10,15c12,17 indique que les lignes 10 à 15 du premier fichier doivent être intégralement changées contre les lignes 12 à 17 du second.
- Dans les trois cas de figure, les lignes précédées du signe < se rapportent au premier fichier, et les lignes précédées du signe > se rapportent au second.
- -b permet de ne pas tenir compte des blancs (espaces) lors de la comparaison des lignes.

TD/TP 8

1. Écrire un script qui affiche le processus qui consomme le plus de mémoire à l'instant, ainsi que la taille de l'espace mémoire utilisé par ce processus. Vous pouvez vous servir des commandes ps et free en plus des filtres de votre choix.

Filtrage étendu
sed

La commande sed

- *sed* est un éditeur ligne par ligne non interactif, il lit les lignes d'un fichier une à une (ou provenant de l'entrée standard) leur applique un certain nombre de commandes d'édition et renvoie les lignes résultantes sur la sortie standard.
- c’est une évolution de l'éditeur *ed* lui même précurseur de *vi*, la syntaxe n'est pas très conviviale, mais il permet de réaliser des commandes complexes sur de gros fichiers.
- Syntaxe :
  - `$ sed -e 'liste_d_instructions' fichier_à_traiter`
  - `$ sed -f fichier_script fichier_à_traiter`

sed : usage et comportement

- Si aucun fichier à traiter n'est indiqué, sed attend les données sur son entrée standard.
- Lorsqu'on fournit directement les commandes sur la ligne, grâce à l'option -e , il est préférable de les inclure entre apostrophes simples, en raison de l'usage fréquent des caractères $, * , ? , etc., susceptibles d'être interprétés par le shell
- Sed est un outil très puissant à la main des administrateurs système puisqu'il permet de traiter de grandes quantités de données en un temps record et avec un minimum de ressources.
- option -n : mode silencieux, envoi vers la sortie seulement si demandé

Expressions rationnelles et métacaractères

- [] permettent de désigner des caractères compris dans un certain intervalle
- Le métacaractère ^ identifie un début de ligne. Par exemple l'expression régulière ^a va identifier les lignes commençant par le caractère a.
- Le métacaractère $ identifie une fin de ligne. Par exemple l'expression régulière a$ va identifier les lignes se terminant par le caractère a.
- Les métacaractères \(\) permettent d’isoler une sous chaîne pour la réutiliser dans la chaîne de substitution
- * signifie la répétition (zéro ou plusieurs fois)
- \+ signifie la répétition (une ou plusieurs fois)
- . remplace n’importe quel caractère

Fonction de substitution s

- Permet de remplacer une chaîne par une autre
- sed 's/regexp/remplacement/options'
  - sed 's/toto/TATA/' fichier #remplacer la première occurrence de la chaîne toto par TATA
  - sed 's/toto/TATA/3' fichier #remplacer la troisième occurrence de la chaîne toto par TATA
  - sed 's/toto/TATA/g' fichier #remplacer toutes les occurrences de la chaîne toto par TATA
  - sed 's/toto/TATA/p' fichier #en cas de remplacement, la ligne concernée est affichée sur la sortie
  - sed 's/toto/TATA/w resultat.txt' fichier #en cas de substitution la ligne en entrée est inscrite dans un fichier résultat

Fonction de substitution s

- Exemples : (suite)
  - sed -e 's/[Cc]haise/CHAISE/g' fichier #substitue toutes les chaînes Chaise ou chaise par CHAISE
  - sed -e 's/^#//' fichier #décommenter tous les commentaires situés en début de ligne
  - sed -e 's/^[a-zA-Z]*$/#&/g' #commenter toute ligne qui contient seulement des lettres et des espaces
  - sed -e 's/\([0-9][0-9]*\)\([a-z]\)/__\1__\2/g' #entourer tous les nombres avec des __ s'ils sont suivis par une lettre
- Note:
  - & permet de faire une référence arrière vers la chaîne filtrée
  - \1, \2 …. permettent de faire une référence arrière à une sous chaîne filtrée, les sous chaînes sont délimitées par des parenthèses

Fonction de suppression d

- La fonction d de sed supprime les lignes correspondantes au critère donné :
  - `sed '20,30d' fichier #supprimer les lignes de 20 à 30 du fichier fichier.`
  - `sed '/toto/d' fichier #supprime les lignes contenant la chaîne toto.`
  - `sed '/toto/!d' fichier #supprime les lignes ne contenant pas la chaîne toto.`
- REMARQUE : les changements apportés par sed n'affectent pas le fichier en entrée. Elles sont juste envoyées vers la sortie, sauf si l'option -i est spécifiée!

Autres fonctions w,p,q,=,

- p (print) : filtre les lignes sélectionnées vers la sortie (même si -n est activée)
- w (write) : écrire les lignes filtrées dans un fichier
- q (quit) : quitter sed dès que l'expression est réalisée (ne pas traiter les lignes suivantes)
- = : afficher le numéro de la ligne

TD/TP 9

1. Supprimer tous les commentaires commençant par // dans un fichier php
2. Mettre en gras le mot Linux ou linux dans un fichier html
3. Sachant qu'on a un fichier qui contient des dates au format 2023-04-20, écrire un script qui permet de les convertir au format 20/04/2023

Filtrage étendu
Le langage AWK

Introduction

- awk est une commande très puissante, c'est un langage de programmation à elle toute seule
- awk rend possible des actions plus complexes encore que celles rendues possibles par sed, y compris des opérations mathématiques, ou des séquences logiques complètes.
- Si la fonction principale de sed est de remplacer des sous-chaînes, awk est le plus souvent utilisée pour extraire des données depuis les fichiers.
- La dénomination AWK vient des initiales de ses trois créateurs, Alfred Aho, Peter Weinberger et Brian Kernighan. Ce dernier est l’un des inventeurs du langage C, d’où une grande similarité dans la syntaxe.

Introduction (suite)

- Les fonctionnalités standards de Awk sont décrites dans la norme, ce qui leur confère une bonne portabilité entre les différentes versions d'Unix.
- Il existe des dérivations de awk améliorées, ou spécifiques à un environnement déterminé comme gawk, nawk, mawk, jawk… et autres
- Awk peut faire des opérations par ligne (considérée comme un ensemble de champs $1, $2… NF avec espace comme séparateur par défaut) mais aussi des opérations sur l'ensemble du fichier (considéré comme un ensemble d'enregistrements)

Fonctionnement

- Le principe général de Awk est le même que celui de Sed :
  - lecture des lignes de texte sur l'entrée standard,
  - traitement en fonction d'instructions regroupées sur la ligne de commande ou dans un fichier externe,
  - écriture des résultats sur la sortie standard.
- Syntaxe : `awk [-F separ] [-v variable] [-f fichier_awk] 'commande' fichier_données`
- Une commande pour awk se présente sous la forme :
  - `motif { action }`
  - où l'action est appliquée à chaque ligne d'entrée qui correspond au motif.
- Un programme awk peut être une seule commande ou plusieurs qu'on met généralement dans un fichier à part (fichier de programme awk)

Enregistrements et champs

- On considère que le fichier d'entrée est constitué d'une suite d'enregistrements, eux-mêmes composés de plusieurs champs. Par défaut, les enregistrements correspondent aux lignes du fichier d'entrée, séparés donc par un caractère de saut de ligne, mais on peut modifier ce séparateur pour lire des fichiers organisés différemment. La variable spéciale RS (Record Separator) décrit la séparation entre les enregistrements:
  - lorsque RS est vide, la séparation se fait sur des lignes blanches
  - lorsque RS contient un seul caractère, celui-ci sert de séparateur d'enregistrements ; par défaut c'est le saut de ligne
  - lorsque RS contient plusieurs caractères, sa valeur est considérée comme une expression rationnelle décrivant les séparations. on peut donc avoir plusieurs séparateurs d'enregistrement pour le même fichier

Les champs (suite)

- Par défaut, les champs sont séparés par des caractères blancs (espaces et tabulations)
- Le séparateur peut être modifié en ligne de commande par l'option -F ou dans le programme awk par la variable FS (Fields Separator)
  - si FS est vide, chaque caractère est considéré comme un champ indépendant
  - si FS contient un seul caractère, il est considéré comme étant un séparateur de champs
  - si FS contient plusieurs caractères, il est interprété comme expression rationnelle

Les champs

- Lorsque l'interpréteur awk reçoit un enregistrement, il le découpe automatiquement en champs. L'enregistrement lui-même est représenté par la variable spéciale $0 . Les champs sont disponibles dans les variables $1, $2, $3 ...
- Le nombre de champs détectés est disponible dans la variable NF
- On peut accéder directement au dernier champ avec l'expression $NF, à l'avant dernier avec $(NF-1) ...
  - `awk -F ';' '{print $NF }' ~/notes.csv`
  - `awk -F ';' '$NF < 10 {print $(NF-1) }' ~/notes.csv`

Syntaxe des motifs

- Les motifs avec lesquels les lignes sont comparées ressemblent un peu aux sélections déjà rencontrées avec Sed, mais offrent plus de possibilités
- Le motif peut être :
  - une expression rationnelle :
    - /expression régulière/ (équivalent à $0 ~ /expression régulière/)
    - expression ~ /expression régulière/
    - expression !~ /expression régulière/
  - une expression de comparaison: <, <=, == , !=, >=, >
  - une combinaison des deux (à l'aide des opérateurs logiques || ou, && et, ! négation)
  - une expression BEGIN ou END

Syntaxe des motifs (suite)

- Le motif peut être aussi :
  - une série de lignes caractérisées par deux motifs
    - motif1,motif2 : chaque ligne entre la première ligne correspondant au motif1 et la première ligne correspondant au motif2
  - Une condition motif1 ? motif2 : motif3
  - Une chaîne vide. L'action sera alore appliquée à toutes les lignes.
- La ligne BEGIN est très utile pour l'initialisation de awk , notamment quand il s'agit d'initialiser des variables requises pour l'exécution du programme.
- La ligne END sert généralement à faire un calcule cumulé sur des variables obtenues lors du traitement des différentes lignes, et pour afficher un message ou un résultat à l'utilisateur

Syntaxe des motifs (exemples)

```
$  awk '/root/ {print $0}' /etc/passwd
```

- `# afficher toutes les lignes qui contiennent le texte “root”`

```
$  awk '$1 ~ /root/' /etc/passwd
```

- `# afficher les lignes qui contiennent “root” dans le premier champs`

```
$  awk 'BEGIN { print "Vérification du fichier /etc/passwd
   pour ...";
   print "- les utilisateurs avec UID = 0 " ;
   print "- les utilisateurs avec UID >= 1000" ;FS=":"}
     $3 == 0 { print "UID 0 ligne "NR" :\n"$1 }
     $3 >= 1000  { print "UID >= 1000 ligne "NR" :\n"$1 }
     END   { print "Fin" }' /etc/passwd
```

- `# affiche les utilisateurs dont le UID = à 0 ou >= à 1000`

Syntaxe des motifs (exemples, suite)

```
$  awk '/sync/ , /news/ {print NR" : " $0 }' /etc/passwd
```

- # affiche la ligne qui contient “sync” et celles qui la suivent jusqu’à celle qui contient “news” (sinon jusqu’à la fin du fichier)

```
$  awk 'NR==5 , NR==10 {print NR" : " $0 }' /etc/passwd
```

- # affiche les lignes entre 5 et 10

```
$  awk -F ';' '$3 !~ /^[0-9]+$/ {print "Le champs 3 de la ligne "NR" n’est pas numérique\n"$0 }'
```

- # affiche les lignes qui contiennent un champs numéro 3 non numérique, en utilisant le séparateur “;”

Les actions

- Les actions sont regroupées entre accolades, et séparées par des points-virgules et/ou des sauts de ligne
- L'action par défaut (si on ne spécifie pas d'action) c'est d'afficher toute la ligne sur la sortie standard.
  - `awk 'NR == 5 , NR == 10' /etc/passwd`
- Dans l'action, on peut aussi faire appel à :
  - fonctions prédéfinies, numériques ou chaine de caractères
  - contrôles (conditions, boucles …)
  - calculs arithmétiques et affectations

Les variables

- Une variable est utilisable dès le moment qu'elle est affectée par une valeur
  - {var=valeur; print var}
- On peut affecter de la même manière les éléments d'un tableau, sans besoin d'allocation de la taille et on peut affecter des valeurs de différents types aux différentes éléments du même tableau.
  - {tab[1]=15; tab[2]="bonjour"; print tab[1] tab[2]}
  - {tab["un"]=15; tab["salut"]="bonjour"; print tab["un"] tab["salut"]}

Fonctions numériques

Nom des fonctions	signification
atan2(y,x)	arctangente de x/y en radians dans l'interval -pi pi
cos(x)	cosinus (en radians)
exp(x)	exponentielle e à la puissance x
int(x)	valeur entière
log(x)	logarithme naturel
rand()	nombre aléatoire entre 0 et 1
sin(x)	sinus (en radians)
sqrt(x)	racine carrée
srand(x)	réinitialiser le générateur de nombre aléatoire

Fonctions sur les chaînes de caractères

Nom des fonctions	signification
gsub(r,s,t)	sur la chaîne t, remplace toutes les occurrences de r par s
index(s,t)	retourne la position la plus à gauche de la chaîne t dans la chaîne s
length(s)	retourne la longueur de la chaîne s
split(s,a,fs)	split s dans le tableau a sur fs, retourne le nombre de champs
sprintf(fmt,liste expressions)	retourne la liste des expressions formatées suivant fmt
sub(r,s,t)	comme gsub, mais remplace uniquement la première occurrence
substr(s,i,n)	retourne la sous chaîne de s commençant en i et de taille n

TP/TD 10

1. En prenant un fichier html en entrée, nous souhaitons afficher seulement les textes inclus entre et (Maj ou min)
2. Ecrire un script awk qui permet de calculer le pourcentage du CPU utilisé par un utilisateur donné (en se basant sur ps aux)

AWK (suite)

AWK: variables prédéfinies

- Variables prédéfinies :
  - FS : séparateur de champ traité
  - RS : séparateur d'enregistrement
  - OFS :séparateur de champ en sortie
  - ORS :séparateur d'enregistrement en sortie
  - NF : nombre de champs dans l'enregistrement courant
  - NR : nombre total d'enregistrements lus dans tous les fichiers de données
  - FNR : nombre d'enregistrements lus dans le fichier de données courant, en comptant l'enregistrement courant
  - FILENAME :nom du fichier en cours de traitement
  - ARGC : nombre d'arguments de la commande
  - ARGV : tableau contenant les arguments de la commande Awk

Structures de contrôle (if)

- Syntaxe : if (condition) instruction1 else instruction2
- Si on a plusieurs instructions on les met entre accolades
- Exemple :

```
cat /etc/passwd | awk ’BEGIN { print "test de l absence de mot de passe";
FS=":"}

NF==7 { #pour toutes les lignes contenant 7 champs

if ($2=="") # si le deuxième champ est vide

{ print $1 " n a pas de mot de passe"}

else { print $1 " a un mot de passe"} }

END { print ("C est fini")} ’
```

Structures de contrôle (while et do..while)

- Syntaxe:

```
while (condition)
{instructions}

do
{
instructions
}
while (condition)
```

- Exemple :
  - awk 'BEGIN { while (count++<50) string=string "x"; print string }'
  - #crée et affiche une chaîne de 50 fois la lettre x

Structures de contrôle (for)

- Syntaxe :

```
for (initialisation; condition; instruction de comptage)
{
     instructions
}

for (clé in Tableau)
{
     instructions
}
```

- Exemple :
  - `for (i=1;i<=NF;i++) print "--"$i"--"`
  - `for (i in ARGV) print "argument " i " : " ARGV[i]`

Les sauts contrôlés

- Les sauts contrôlés permettent d'interrompre un traitement.
- Sauts de boucles :
  - break : permet de sortir de la boucle en cours,
  - continue : permet de passer à l'itération suivante de la boucle (Les instructions après continue ne seront pas prises en compte). Cependant le nouveau passage se fera si et seulement si la condition est remplie.
- Sauts du programme :
  - next : arrête le traitement de l'enregistrement en cours et il passe au suivant.
  - exit : permet l'arrêt complet des traitements. Awk exécute tout de même le bloc END.

TP/TD 10

suite

1. Ecrire un script awk qui permet de numéroter toutes les lignes d’un script shell, sauf si elles sont commentées ou vides
2. Ecrire un script awk qui permet d’inverser les lignes d’un petit fichier
3. Écrire un script awk qui permet de réécrire le fichier notes.csv en remplaçant la notation sur 20 par une notation sur 10.

TP/TD 10

suite

1. Dans un corpus de textes en Français, afficher la première phrase qui contient le mot “il”.
2. Dans un corpus de textes en Français, calculer le nombre d’apparitions du mot “il” peu importe la casse.
3. Dans un corpus de textes en Français, calculer la fréquence de succession de chaque deux lettres (ou caractères).

TP/TD 10

suite

1. En utilisant awk et ls, afficher la taille totale en Kilo-Octets des fichiers du répertoire courant.
2. Écrire un script awk qui supprime tout texte entre parenthèses.
3. Simuler (approximativement) avec un script awk la commande wc, qui sans option supplémentaire donne respectivement le nombre de lignes, de mots et d'octets du fichier qui lui est passé en argument.

TP/TD 10

suite

On dispose de différents fichiers à la structure identique : une première ligne non pertinente qui pourra être ignorée, suivie d'un ensemble de lignes structurées en 13 champs séparés soit par des espaces, soit par des tabulations. Voici un exemple de ligne :

str1 int2 int3 int4 int5 int6 int7 int8 int9 int10 int11 int12 int13

Travail à faire: On souhaiterait faire la somme des colonnes de chacun des fichiers et écrire les résultats obtenus dans un fichier résultat, au format suivant :

file_i tot2 tot3 tot4 tot5 tot6 tot7 tot8 tot9 tot10 tot11 tot12 tot13

Projets

Projet 1

évalué sur la base de :

1. la réalisation des objectifs
2. Utilisation du CPU
3. Utilisation de la RAM

- Écrire un script qui prend en argument un répertoir local de site web et effectue les opérations suivantes :
  - Optimiser les fichiers CSS en supprimant les entrées non utilisées
  - Minifier les fichiers CSS
  - Minifier les fichiers JS
  - Minifier le HTML inclut dans les fichiers ayant les extensions : .htm, .html, .php

Projet 2

évalué sur la base de :

1. la réalisation des objectifs
2. temps d'exécution
3. utilisation du CPU
4. utilisation de la RAM

- Écrire un script qui prend en argument une URL et un fichier texte contenant une liste d'URLs d'exceptions, et permet de générer les différents fichiers sitemaps pour ce site.

Projet 3

Évalué sur la base de :

1. La réalisation des objectifs
2. La qualité de la présentation des résultats.

- Vous êtes enseignant et vous souhaitez réaliser un script pour évaluer les projets/travaux rendus par les étudiants. Ce script doit être capable de :
  - Évaluer le taux de ressemblance entre tous les projets dans un dossier donné en argument,
  - Évaluer les ressemblances même si les noms (variable, classes,fichiers,fonctions...) ou les structures sont différents.
  - Évaluer le plagiat sur internet et afficher les liens sources pour chaque projet.
  - Évaluer le projet et toute sa hiérarchie même s’il est compressé.