PRESENTATION UNIX UTILISATEUR

Objectif de ce document :

Il s'adresse à une personne désirant **travailler sous UNIX avec un interface shell** sur des commandes simples.
Il traite de manière superficielle le système d'exploitation mais en explique certaines contraintes.

Jusqu'au paragraphe "commandes de base, ce document relate les caractéristiques d'UNIX de manière schématique.
Ensuite sont développées les commandes principales avec des exemples.
Puis quelques exercices pour les plus décidé(e)s.
Enfin des annexes et fiches récapitulatives sont fournies.

Versions :

V0 : Claude Conty ("support de cours unix") 1999
V1 : Eric Hamon / Franck Mendras / Christian Floirat / Alexandre Pezzoli 02/2000
V2 : Relecture 02/2000

Remarques préliminaires :

Dans la suite de ce document, parfois des touches du clavier sont présentées entre < > .
<CTRL> correspond à la touche "Contrôle" marquée Ctrl normalement.
***<CTRL>+<D>** signifie qu'il faut appuyer sur les deux touches simultanément.*

La touche <ESC> correspond à la touche <Echap> ou "Escape" du clavier.

Si vous êtes bloqués, essayez les touches suivantes :

<CTRL>+<D> = Sortie propre
<CTRL>+<Z> = Quitter le mode saisi
<ESC> = Idem sous vi

<CTRL>+<C> = Interruption

** Lorsque vous écrirez sous UNIX, les caractères majuscules et minuscules ont une signification.*

** Oubliez les caractères accentués afin d'éviter des déconvenues.*
- Dans les noms de fichier (portabilité dans d'autres environnements)
- Dans le contenu des fichiers (risques de pertes en cas de transfert)

Table des matières

I Introduction à UNIX..... 4

I.a Généralités sur les systèmes d'exploitation..... 5

Grands principes d'unix..... 6

II Concepts de base UNIX..... 8

II.a Le Login 8
II.b Gestion du mot de passe (commande passwd) 9

II.c Aide en ligne (commande man <instruction>)..... 10

III La notion de shell..... 11

III.a Lancer une commande par l'interpréteur du shell..... 11

III.b Variables d'environnement (.profile)..... 13

III.c Raccourcis de commandes en KSH : les alias..... 14

III.d Redirection (> ou >>)..... 15

III.e Mécanisme de pipe..... 16

III.f Lancement de plusieurs sessions (sur HP seul) : tsm..... 17

III.g Avoir une gestion des commandes à la doskey (korn shell seul)..... 17

IV Manipulation de chaines de caractères..... 18

V Manipulations de fichiers..... 20

V.1 Identification logique d'un fichier..... 20

V.2 Liste du contenu d'un répertoire (ls)..... 21

V.3 Droits sur les fichiers et répertoires..... 22

Changements de permissions (chown, chgrp) 23
Changer les droits d'un fichier ou d'un répertoire (chmod) 24
Initialisation des droits sur les nouveaux fichier (umask) 24

V.4 Gestion des fichiers..... 25

Liens physiques de fichiers 25
Liens symboliques de fichiers 25
Types de fichiers (file) 26
Visualiser un fichier texte (cat , pg, tail, head) 26
Création de fichiers (touch, cat, echo, vi) 26
Editeur vi 27
Déplacement et changement de nom de fichiers (mv) 28
Copie de fichiers (cp) 28
Destruction de fichiers (rm) 28
Recherche de fichiers (find) 29
Comparaison de fichiers (cmp, diff) 30
Comptage d'un fichier (wc) 30
Recherche d'une chaine de caractère (grep) 30
Tri de chaines de caractères (sort) 30

VI Notions de filesystems ou « systèmes de fichier »..... 31

Commandes df (AIX) ou bdf (HP) 32

VII Manipulations sur les répertoires..... 33

Création d'un répertoire (mkdir) 33
Destruction de répertoires (rmdir) 33
Changement de répertoire (cd) 33
Affichage du répertoire courant (pwd) 33
Occupation des répertoires (du) 33

VIII Gestion des archives / compressions..... 34

Programme tar 34
Programme cpio 34
Copie de fichiers avec conversion (dd) 35
Compression des données (compress, uncompress, gzip) 35

IX Communication 36

IX.1 Les protocoles de communication (IP) 36
La commande ping (tester une adresse IP) 37
Le fichier /etc/hosts (nom supplémentaire à l'adresse IP) 37
La notion de Domain Name Server (DNS) 38

IX.2 Les conversations netre utilisateurs/trices 39
Qui est donc connecté ? (who) 39
w (who amélioré) 39
mail (conversation asynchrone) 39
Talk (conversation synchrone) 40
write user (message synchrone) 41
wall (message synchrone à tous les users connectés) 41

X Les processus 42

X.1 Gestion des processus 42
Etat des tâches (ps) 43
Affectation d'une priorité (nice / renice) 43
Destruction d'un process (kill) 44

X.2 Etude de la charge système (top sur HP, sar) 45
Commande top sous HP 45
Activité système par la commande sar 46

XI Les imprimantes 47

XII Fonctions avancées 48
Exécution programmée permanente de commandes (crontab) 48
Exécution différée exceptionnelle de commandes (at) 49

Transfert de fichiers FTP 49

XIII Langage de programmation awk 51

XIV Le script shell 59

Variables et Affectation du résultat d'une commande 63
Exemple de script commenté 66

Exercices sur les scripts 68

Solutions 69

Annexe 1 Mémo de l'éditeur vi 70

Annexe 2 Mémo commandes UNIX 71

Annexe 3 Mémo commande awk 72

I Introduction à UNIX

Le système d'exploitation UNIX a été créé dans les années 70.

Son nom est l'opposé du système d'exploitation qu'il devait remplacer « MULTICS ».

Le langage de prédilection d'UNIX est le **langage C**, très proche du langage machine (plus performant mais moins accessible au non initié).

Développé dans les services d'ATT, ce système d'exploitation a connu un fort développement dû à son don gratuit aux universités américaines.

Plusieurs versions ont vu le jour depuis et des standards différents nécessitent de distinguer deux tendances :

a) UNIX SYSTEM V (ATT) :

b) BERKELEY BSD 4.2 (du nom de l'université américaine) :

Les UNIX tendance BSD sont plus souples (noms de fichiers longs, gestion en réseau, ...) mais chaque amélioration est reprise dans la version suivante de l'autre tendance.

Ces deux branches se retrouvent petit à petit et un comité de standardisation tente d'harmoniser les évolutions.

Le système AIX d'IBM est du type BSD alors que HP-UX de Hewlett Packard est hybride.

Ce qu'il faut retenir est qu'il y a plusieurs UNIX et que selon le constructeur de la machine où vous travaillez, les commandes / interfaces sont différents.

I.a Généralités sur les systèmes d'exploitation

La complexité d'une machine physique, comme un micro-ordinateur, la rend inexploitable telle quelle par toute personne ne l'ayant pas conçue !

Il est donc nécessaire de créer, au dessus de cette machine, une machine "symbolique" avec laquelle on va pouvoir converser plus facilement. C'est le but du système d'exploitation.

On appelle ainsi l'ensemble des programmes résidants en mémoire permettant l'accès aux ressources physiques de la machine par l'intermédiaire d'une machine "virtuelle". On peut dire qu'il s'agit de **l'ensemble des procédures de gestion automatique des ressources physiques** (processeurs, mémoire...) **à la disposition des utilisateurs**.

D'autre part, le système alloue ces ressources d'une manière "optimisée", au sens d'un certain critère, par ceux qui l'ont écrit. Ces critères varient fortement en fonction de l'utilisation qui sera faite de la machine : par exemple la gestion mémoire et le système de protection de fichiers, fortement développé sur mini-systèmes de gestion, sont quasi-inexistants sur certains micros - ordinateurs.

Les fonctions du système d'exploitation sont les intermédiaires entre l'utilisateur et le matériel informatique

Elles permettent la gestion des ressources système :

- Unités centrales (processeur et son dispositif)
- Mémoires principales et secondaires (zone de travail du processeur)
- Entées/sorties (communication vers imprimantes, clavier, écran, réseau, ...)
- Fichiers (zones de stockage des données ou des programmes)

Elles gèrent l'interface entre l'utilisateur et les applications

- Compilateurs (constructeur de programme)
- Éditeurs (permet de saisir du texte)
- Utilitaires systèmes (gestion de fichier, ...)

Elles fournissent la première interface avec l'utilisateur (interpréteur de commande)

- Partie visible du système d'exploitation
- Accepte les commandes de l'utilisateur et les interprète

Un peu de franglais

Kernel : En français "noyau". C'est la partie la plus profonde d'un système d'exploitation. Il faut un minimum de codes pour que celui-ci soit fonctionnel. Le nombre et la définition des modules qui sont inclus dans le "kernel" varient de personne à personne et de système à système.

Multitasking : En français multi-tâche. Un système multi-tâche est un système d'exploitation qui peut traiter des tâches de façons intercalées.

Shell : Nom de l'interpréteur de commande d'unix

Grands principes d'unix

Système de fichiers hiérarchique (par arborescence de répertoire)

- Fichiers ordinaires (texte, exécutables, ...).
- Répertoires (comme un fichier mais leur rôle est de stocker des entrées de fichier pour les hiérarchiser)
- Protection en lecture, écriture et droits d' exécution selon le type d'utilisateur..

Lecture		Propriétaire
Écriture	⟷	Groupe
Exécution		Autre

- Utilisateur particulier (super utilisateur) = **root** ayant tous les droits.

Processus

Définition : Le(s) programme(s) lié(s) à la réalisation d'une fonction.
Chaque tâche donnée au système UNIX est un processus.

- Multi-tâche préemptif
  ==> Ceci signifie que tous les process sont gérés par un sheduler (qui supervise : déclenche les process, les fige pour donner la main à un autre, gère les priorités, …) à l'inverse d'un multitâche coopératif où chaque process lancé doit rendre la main pour permettre à un autre process de travailler (cas de Windows par exemple).
  Ce type de multi-tâche est plus lourd en gestion mais bien plus stable.

- Un processus est créé par copie du processus créateur
  (Le processus d'initialisation de la machine est le « père » de tous les autres)

- Un processus père et un processus fils peuvent se synchroniser et communiquer

Cette notion de processus est importante à connaître pour se sensibiliser aux interactions entre tâches systèmes ou utilisateurs.

Interprèteur de commandes (shell)

- Exécution de commandes en provenance d'un terminal ou d'un fichier.

- Redirection des entrées/sorties
  `>` sortie
  `<` entrée
  `>>` sortie en fin de fichier (extension)
  `<<` définition d'un caractère fin de fichier (procédures shell)
- Gestion des tâches de fond, des exécutables séquentielles et parallèles.
- Concaténation des commandes (tubes)

Plusieurs shell avec des syntaxes / possibilités différentes.

Entrées/Sorties

- Unicité d'interface (exemple, le signe > de redirection)
  Une redirection est écrite de la même manière quelque soit la sortie
  `cat texte > /dev/lp` texte vers imprimante
  `cat texte > copie` texte vers fichiers

Environnements de développement

De nombreux environnements et outils sont à disposition des utilisateurs.
Ces contextes de travail sont une interface supplémentaire pour agir avec la machine.

- Compilateurs c, apl, f77, basic, pascal....
- Débuggers (adb)
- Éditeurs de texte (ed, sed, vi, emacs)
- Traitements de texte (troff, nroff, tbl, egn)
- Correcteur orthographique (spell)
- Courrier électronique (mail, write, wall)
- Gestion de projets (make, sccs)
- Communications inter-cpu (cu, uucp)
- Analyseur lexical (awk, lex)
- Manuel en ligne (man)
- Sauvegardes/archivage (tar, dd, dump)
- Logiciels d'applications

II Concepts de base UNIX

Ah ! enfin du concret !!

Les principes suivants sont très importants :

Comment fonctionne le login ?
Comment se gère un utilisateur ainsi que son mot de passe ?
Comment se connecter et se déconnecter ?
Comment obtenir l'aide en ligne ?

Maitrisez les et surtout pensez à cette **aide en ligne précieuse** (la commande **man**) car les UNIX sont différents et un paramètre peut s'écrire différemment d'une machine à l'autre.

Rappelez vous : **man** comme manuel, manitou, maman, m'enfin !

II.a Le Login

Pour démarrer une session, la connexion passe par une identification (le **"login"**) où vous donnez votre identifiant user et votre mot de passe.

Pour plus de précisions sur cette notion de "session", vous pourrez consulter le paragraphe sur la communication.

Puis, après acceptation, le « shell » propose à l'utilisateur un "**prompt** " (message d'attente ou d'invite) destiné à marquer la zone de saisie des commandes unix à venir.

Chronologie d'une session

a) Connexion à la machine (service telnet ou ftp ….)

b) Saisie du login et du pass associé

c) Vérification du nom et du mot de passe par le système
(fichier /etc/passwd)

d) Si le login et le pass sont bons :

Signale la présence de **courrier** éventuel

Initialisation du shell
(l'interpréteur de commande prend la main)

Exécution du fichier **.profile**
fichier présent dans le répertoire d'accueil de l'utilisateur
(initialisation de variables personnelles)

e) Gestion des commandes par le shell
Le prompt SHELL est activé (par défaut " **>** ")
(L'utilisateur saisi ses instructions au prompt)

f) Fin du shell ou Retour au login par <CTRL>+<D>
ou taper la commande « exit » ou « logout »

Le test d'identification s'effectue grâce au fichier /etc/passwd où sont stockés tous les éléments distinguant les utilisateurs possibles.

Extrait de fichier /etc/passwd

```
root:!:0:0::/:/bin/ksh
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
maurice:!:3:3::/usr/sys:
moi:!:202:207::/users/mqm:/usr/bin/ksh
unautre:!:203:204::/users/aep:/usr/bin/ksh
```

Plusieurs utilisateurs sont présents (root, daemon, bin,...)

Les champs du fichier /etc/passwd sont séparés par le sigle :

Le **nom utilisateur** (celui du loggin),
le **mot de passe crypté ou « ! »,**
le **n° UID** (numéro unique pour gérer les utilisateurs par un nombre plutôt qu'une séquence de caractères),
le **n° de groupe,**
un **commentaire,**
le **répertoire d'accueil**
et enfin le **type de shell à lancer** sont autant d'éléments distinguant un utilisateur.

Un second fichier recense les groupes d'utilisateur.

Extrait de fichier /etc/group

```
system:!:0:nuucp,root
bin:!:2:root,bin
sys:!:3:root,bin,sys
adm:!:4:bin,adm
usr:!:100:guest
informix:!:200:informix
```

L'administrateur de la machine a en charge ces deux fichiers.
Vous ne pouvez que changer le mot de passe.

Toutes autres modifications sont à demander auprès de votre administrateur préféré(e).

II.b Gestion du mot de passe (commande passwd)

Commande **passwd**

* Eviter de mettre un mot trouvable dans le dictionnaire par sécurité (la méthode la plus simple pour trouver un mot de passe pour un intrus, est de récupérer le /etc/pawwsd et faire des tests automatiques avec un dictionnaire en ligne).

* L'emploi de caractères comme « (ou & est un plus.
* Prendre un mot composé : fleur!noire par exemple …
* Attention aux combinaisons majuscules/minuscules, celles-ci sont mémorisées :
Goupi"goupa commence par un "G" (un "g" serait faux)
* Mélanger chiffres et lettres sur une longueur de 8 caractères ou plus pour assurer une meilleure sécurité.

Connaître la date : commande **date**

Connaître le type de machine : commande **uname -a**

Connaître le nom de la machine : commande **hostname**

II.c Aide en ligne (commande man <instruction>)

Par la suite, de nombreuses commandes seront disponibles.

Leurs options varient d'un système UNIX à l'autre (HP, AIX, SUN, ...) et rendent difficiles leurs mémorisations par leur codage spécifique.

Une manière de connaître les arguments d'une commande et son rôle est d'utiliser l'aide en ligne sous forme de manuel.

Pour l'obtenir, tapez **man nom_de_la_commande**

Si le manuel n'est pas installé sur cette machine, vous obtiendrez un message d'erreur (voir avec l'administrateur pour l'installation de l'aide en ligne)..
Sinon, un descriptif de la commande s'affiche.

Le nombre d'options varie suivant la commande utilisée. Pour avoir plus d'informations sur les options disponibles, lancer la commande man

exemple : **man ls** **<aide en ligne de la commande ls>**

```
ls(1)                                                    ls(1)

 NAME
      ls, l, ll, lsf, lsr, lsx - list contents of directories

 SYNOPSIS
      ls [-abcdefgilmnopqrstuxACFLR1] [names]

      l [ls options] [names]
      ll [ls options] [names]
      lsf [ls options] [names]
      lsr [ls options] [names]
      lsx [ls options] [names]

 DESCRIPTION
      For each directory argument, the ls command lists the contents of the
      directory.  For each file argument, ls repeats its name and any other
      information requested.  The output is sorted in ascending collation
      order by default (see Environment Variables below).  When no argument
      is given, the current directory is listed.  When several arguments are
      given, the arguments are first sorted appropriately, but file
```

La touche <ESPACE> vous permet de passer à la page suivante
<RETURN> passe à la ligne suivante
< q > pour QUITTER

Astuce :

Si vous sortez le contenu de l'aide en ligne dans un fichier texte, des caractères de contrôle ainsi que des n° de pages apparaîtront dans votre fichier texte.
Pour éviter ces caractères, tapez l'instruction man comme suit :

man commande | col –b > texte.txt

III La notion de shell

En premier lieu, il s'agit de l'interpréteur de commande.
C'est cet outil qui vous permet de dialoguer avec le système d'exploitation.

Il est l'élément à connaître dans vos pérégrinations !

Le shell standard est le bourne-shell, du nom de son concepteur.
On rencontre également le bash-shell (bash), le tc-shell (tsh), c-shell (csh) dont la syntaxe est proche du c, plus puissant mais plus difficile de compréhension, le visual-shell (vsh) à base de menus, le restricted-shell limité à des instructions sommaires (pas de commande cd par exemple) et le korn-shell (plus complet).

Les différences entre shells sont importantes en terme d'usage.

Si vous travaillez en korn shell, certaines commandes que vous pouvez utiliser (rappeler les dernières commandes saisies par exemple) ne sont pas connus en Bourne shell.

Pour connaître le shell sur lequel vous travaillez, tapez au prompt :

echo $SHELL

/bin/**sh** est le Bourne shell /bin/**ksh** est le Korn shell, /bin/**csh** est le C-shell

Le shell est à la fois un puissant interpréteur de commandes et un langage.

Lorsqu'un utilisateur se connecte (login), le noyau UNIX lance le programme décrit dans le dernier champ de la ligne associée à l'utilisateur dans le fichier /etc/passwd qui correspond généralement au shell à lancer (sh pour bourne shell, ksh pour korn shell).

Vu du système, ce shell est un processus utilisateur comme les autres.
Il est continuellement en attente d'une entrée au clavier.

Il meurt quand la touche **^D** (<CTRL> + <d>) est frappée ou que vous quittez le process courant (c'est à dire le shell qui attend vos instructions au prompt) par la commande **logout** ou **exit**.

Le caractère d'attente du shell (Il s'agit du message d'invite à gauche du curseur) est $ pour un utilisateur quelconque et # pour le super utilisateur.
Ce caractère est modifiable (variable d'environnement PS1 vue plus loin).

III.a Lancer une commande par l'interpréteur du shell

Le format général d'un saisie sous shell est

commande argument1 .....argument n **<RC>**

Chaque argument est séparé par un ou plusieurs espaces.

La commande peut être un fichier binaire exécutable ou un fichier texte contenant d'autres commandes shell.

Les arguments sont des chaînes de caractères contigus

Exemple : **cp texte ../jean/lettre**
cp est la commande
texte est le premier argument
../jean/lettre est le deuxième argument

Il est possible d'exécuter des commandes en séquence. Ces commandes sont séparées par un ";"

Exemple : **cc -c io.c ; mv a.out prg.exe**
Compilation du programme en C io.c, puis renommer le fichier résultat a.out en prg.exe

Exécution d'un programme sur disque

L'exécution de /chemin/proc peut se faire de deux façons :

/chemin/proc
si la procédure possède l'attribut x
(exécutable ; droit d'exécution OK offert par la commande chmod +x proc
==> *voir chapitre sur les droits des fichiers*)

sh /chemin/proc
si la procédure ne possède pas l'attribut x
Dans ce cas, un nouveau shell (ici sh) englobe le programme. Un autre shell (ksh – korn shell par exemple) peut être utilisé.

Regroupement de commandes

Deux méthodes

accolades
{ cmd1 ; cmd2 }
Les deux commandes sont exécutées simplement

parenthèses
(cmd1 ; cmd2)
Les deux commandes sont exécutées par un nouveau processus.

Chaînage de commandes

cmd1 && cmd2
cmd2 n'est exécutée que si cmd1 est terminée avec succès

cmd1 || cmd2
cmd2 n'est exécutée que si cmd1 a échoué

Exécution même après déconnexion (nohup)

Dès que le shell père s'arrête, tous les process associés sont stoppés.
Dans la mesure où votre shell de connexion est le papinou de tous les process que vous déclenchez, votre déconnexion scelle leur destin.

Si vous souhaitez voir votre commande rester en mémoire et s'arrêter normalement malgré votre départ , faites précéder votre commande par **nohup**

Exemple : **nohup commande1**
Lance le process avec l'instruction 'commande1' qui restera en mémoire indépendamment du départ de l'utilisateur.

Un fichier ***nohup.out*** d'informations est créé dans le répertoire où est lancé la commande.

Exécution en arrière plan (&)

Le shell rend la main dès le lancement du processus sans attendre la fin de celui-ci (le pid du process lancé est affiché en retour).

Exemple : **commande1 &**
Lance le process avec l'instruction 'commande1' en arrière plan et donne immédiatement la main pour continuer.

==> Vous pouvez combiner les deux systèmes nohup et &
nohup commande1 &

III.b Variables d'environnement (.profile)

==> Cette notion est reprise dans le paragraphe sur les scripts.

Afin d'éviter de ressaisir des valeurs constantes et d'assurer une meilleure portabilité des applications, des variables d'environnement sont positionnées.

Par la suite, dans les scripts ou les commandes passées à l'interpréteur de commande, ces variables seront utilisées facilitant les traitements automatisés.

Une variable est défini par une chaîne de caractère et peut être alimenté par n'importe quel type de valeur (chaîne de caractère, nombre, résultat de commande, …).

Son contenu est interprété en préfixant le nom de variable par **$**

Par exemple, tapez les instructions suivantes :

PROMPT UNIX> TEST=`pwd`

Met le chemin du répertoire courant dans la variable TEST
(Ne pas mettre d'espace(s) autour du " = " car cela sera interprété
==> Ici le nom de la variable est "TEST" et non pas "TEST ")

PROMPT UNIX> echo $TEST

Affiche le contenu de la variable TEST (/grosloulou/repertoire_travail/jeux par exemple).

N'oubliez pas que vous êtes dans un shell (celui de votre session).

Les processus (éditeurs, autres shells, programmes, …) que vous déclencherez à partir de ce shell n'hériteront pas de vos variables.
Si vous voulez voir vos variables prises en compte dans tous les sous-shell, … songez à les "exporter" par la commande **export**

export TEST
ou **export TEST=5** pour exporter en donnant une valeur numérique à 5.
ou **export CHEMIN=/users/applix**
pour exporter en donnant un nom de répertoire.

A l'invite, plusieurs variables d'environnement ont des valeurs définies.

HOME ***Répertoire d'accueil de l'utilisateur***

désigné aussi par le caractère "Tilde" **~**
cd $HOME/archive vous ramène à votre répertoire de départ dans la sous arborescence 'archive'.
Ceci est équivalent à **cd ~/archive**

PATH ***Chemin de recherche des commandes exécutables***

Lorsque vous tapez une commande, l'interpréteur SHELL la recherche dans ses commandes internes puis ensuite sur le disque.
Cette recherche sur le disque s'effectue en parcourant les répertoires contenus dans $PATH.
Si l'interpréteur ne la trouve pas, il vous renvoie un message
" sh: Prout: not found.".
Il ne trouve pas la commande "Prout",
(La commande est le 1er mot de la ligne saisie au prompt)

Soit vous avez commis une faute de frappe (c'était "Pet"), soit le $PATH ne contient pas le chemin d'accès au programme voulu (Prout est dans /users/applix/80q).

Pour rajouter le chemin /users/applix/80q à la recherche du shell :
export PATH=$PATH:/users/applix/80q

PS1 ***Chaîne de caractères du PROMPT*** ($ par défaut)

Le prompt, c'est le bittonio qui sert de message d'invite à gauche du curseur.

Vous pouvez y placer tout plein de truc : le répertoire courant, la date et l'heure, votre nom de famille, la couleur de votre vin préféré.

Vous désirez voir le nom de la machine (instruction uname), votre login (variable LOGNAME) et le répertoire courant (instruction pwd) ?

export PS1=`uname -n`:$LOGNAME:'$PWD'\>

==> Cette formule est décortiquée dans le paragraphe sur les scripts.

TERM ***Type de terminal***

Ca cafouille avec votre clavier ?
Si vous êtes sur un micro PC, essayez **export TERM=vt220**

Pour connaître les variables d'environnement du shell en cours, taper **env**

Pour connaître toutes les variables d'environnement connues, taper **set**

Le fichier situé dans votre répertoire d'accueil ($HOME)de nom **.profile** est lu à l'initialisation du shell.
Mettez y vos variables préférées !
Faites gaffe aux erreurs de syntaxe et … **sauvegardez le avant !!**

Caractères spéciaux pour les noms de variables shell

Certains caractères spéciaux, représentant des variables shell spéciales, sont automatiquement définis par le shell. Comme toutes les variables, ils sont précédés d'un $.

$? Statut retourné par la dernière commande

echo $? ==> donne le code retour de la dernière commande

$# Nombre d'arguments passés au programme
$$ Numéro de processus du programme en cours
$! Numéro du dernier processus fils
$n Valeur de l'argument à la position n du programme
$0 nom du programme courant

III.c Raccourcis de commandes en KSH : les alias

Certains commandes sont souvent écrites.
Pour les taper rapidement avec un "pseudo", vous pouvez utilisez un alias.

Ceci n'est possible qu'en **korn shell (ksh).**

Définir l'alias

PROMPT UNIX > alias ll='ls -la'
PROMPT UNIX > alias ora='ps -ef | grep ora'
==> utiliser la quote ' (sous le <4>)
==> Ne pas mettre d'espace autour du =

Utiliser l'alias défini

PROMPT UNIX > **ora**

Donne le résultat de ps –ef | grep ora

Le fichier situé dans votre répertoire d'accueil ($HOME)de nom **.profile** est lu à l'initialisation du shell.
Mettez y vos alias préférées !
Faites gaffe aux erreurs de syntaxe et **sauvegardez le avant !!**

Supprimer l'alias défini auparavant

PROMPT UNIX > **unalias ora**

Retire l'alias "ora"

Voir la liste des alias définis

PROMPT UNIX > **alias**

Liste les alias avec leur définition

Attention : Un alias n'est pas exporté !
Il n'est donc pas utilisable directement dans un sous-shell, il faut le redéfinir dans ce cas.

III.d Redirection (> ou >>)

Au login de l'utilisateur, le système UNIX associe automatiquement l'ouverture de 2 fichiers spéciaux "clavier" (entrée) et "écran" (sorties)

clavier	0	(stdin)	Entrée standard
écran	1	(stdout)	Sortie standard
écran	2	(stderr)	Sortie erreur standard

Il est possible de rediriger la "sortie des messages d'erreur" (stderr) dans un fichier

Exemple : cat /etc/passwd 2 > /tmp/errfich

ou sur la sortie standard (ce qui est fait par défaut)

Exemple cat /etc/passwd 2> &1

> Redirige la sortie (qui peut provenir d'une commande)

>> rajoute la sortie (qui peut provenir d'une commande)

L'opération la plus courante consiste à diriger la sortie d'une instruction vers un fichier texte :

Commande > fichier.txt met le résultat de "commande" dans fichier.txt

Commande >> fichier.txt rajoute le résultat de "commande" dans fichier.txt
Si ce fichier n'existe pas, il sera créé.

ls –la > liste.txt
Liste les fichiers du répertoire courant et met le résultat dans liste.txt

ATTENTION :

Le caractère > écrase le fichier existant en sortie. Prenez donc garde !

Une erreur de manipulation facile avec le copier/coller de Windows consiste à attraper malgré soi le caractère > mis dans le prompt et à l'insérer dans la commande.

Pour les plus valeureuses, valeureux, un casse tête :

Vous pouvez utiliser les redirections pour gérer une séquence de commande donnée à un programme :

```
sqlplus -s utilisateur/password 2>$Ftest @<<!
    set heading off
    set echo off
    set feedback off
    spool $Ftest
    select decode( count(*),0,'Aucune Periode a Archiver','-') from t80qmvt
    where daf < add_months( sysdate, -18)
      and archive = 'N' ;
    spool off
    exit ;
    /
!
```

Un ensemble de séquence sql a été passé à l'outil sqlplus.
Les commandes pour sqlplus démarrent par un @

<< suivi d'un sigle (ici !) indique le début de la séquence en entrée pour la commande.
Le << provoque la lecture de ce qui suit par la commande.
Le ! à la fin signifie l'arrêt des commandes passées à sqlplus.

Le résultat de la requête sql est mis dans le fichier $Ftest (instruction sql "spool").
Vous remarquerez que Ftest est aussi destinataire des messages d'erreur (renvoie de 2>$Ftest.

III.e Mécanisme de pipe

Les caractères issus d'une commande écrivant normalement au terminal peuvent servir de flot d'entrée à une commande lisant normalement le clavier.
On établit ainsi **un tuyau de la première commande vers la seconde**. Ce tuyau est symbolisé par le caractère **|**

Exemple : **cat /etc/passwd | wc -l**

Donne le nombre de lignes du fichier /etc/passwd

La première instruction 'cat /etc/passwd' est exécutée
Elle donne le contenu du fichier /etc/passwd
Ce résultat est fourni, via le "pipe", à la commande suivante 'wc -l' qui compte le nombre de lignes
Le résultat de ces commandes associées est un chiffre (le nombre de lignes du fichier)

Une suite de commandes liées par des "pipe" est appelée "pipeline"

Exemple : cmd1 | cmd2 | cmd3 ....

III.f Lancement de plusieurs sessions (sur HP seul) : tsm

tsm terminal Share Management : taper **tsm**

indique ensuite les touches qui sont généralement :
CTRL W Passer d'une session à une autre
CTRL T Ouvrir le menu pour quitter ou autre..

III.g Avoir une gestion des commandes à la doskey (korn shell seul)

Si votre shell est ksh (korn shell), placez les commandes suivantes (dans le .profile par exemple)
Ceci est très utile avec un **clavier PC**.

```
alias __A=`echo "\020"`   # up arrow = ^p = back a command
alias __B=`echo "\016"`   # down arrow = ^n = down a command
alias __C=`echo "\006"`   # right arrow = ^f = forward a character
alias __D=`echo "\002"`   # left arrow = ^b = back a character
alias __H=`echo "\001"`   # home = ^a = start of line
set -o gmacs
```

Vous pourrez ainsi rappeler vos commandes passées et les retravailler avec les flèches.

Chaque ligne donne un alias à une touche du clavier PC.
Cet alias donne une séquence ASCCI (en valeur octale). Par exemple 020 pour effectuer le retour arrière.

D'autre part, ^u efface la ligne, ^e fin de la ligne.

Exercice : Retrouvez les valeurs octales de ces deux valeurs (je les indiquerai dans une future version de ce document ou lorsque je serai réincarné en salade de tomates).

IV Manipulation de chaînes de caractères

Par la suite, vous manipulerez des fichiers, des chaînes de caractères, etc …

Pour pouvoir effectuer des opérations globales, sur un ensemble de données et ne sélectionner que celles qui correspondent à votre action, vous indiquerez par des caractères particuliers les éléments à considérer.

Les méta caractères suivants sont utilisés pour les fichiers et aussi dans la manipulation de chaînes de caractères (manipulation de résultats de commandes ou de contenu de fichier).

***** désigne
un champ quelconque
ou la répétition du bloc précédent entre [ ]

? un caractère quelconque

^ Début de chaîne

$ Fin de chaîne

***** \ suivi d'un caractère permet d'utiliser ce caractère sans interprétation (ici le caractère * sera pris en compte tel quel.

[a-c] Les crochets donnent les valeurs sélectionnées ("-" donne une plage)
(ici, les lettres a, b ou c)

[a-c]* L' * rajoutée indique une répétition possible du bloc

! Est la négation

| OU Logique (obtenu avec <Alt Gr> + <6>)

& ET Logique

Exemples (Expliquez la sélection opérée par ces instructions) :
Note : pour les fichiers, il s'agit du nom du fichier et non pas le contenu qui est testé

a) `Poir*.c`

b) `Fl[OkL1-4]?h`

c) `*.[a-zA-Z]`

d) `[!f]*`

e) `^..$`

f) `^[A-Za-z]$|^[A-Za-z][0-9]`

Explications :
Note : pour les fichiers, il s'agit du nom du fichier et non pas le contenu qui est testé

a) `Poir*.c`

Tous les fichiers ou chaines commencant par "Poir" et se terminant par ".c"

b) `Fl[OkL1-4]?h`

Tous les fichiers ou chaînes commencant par "Fl", le 3ème caractère peut être "O" ou "k" ou un chiffre entre 1 et 4, le 4ème caractère est quelconque, le 5ème caractère est "h".

c) `*.[a-zA-Z]`

Tous les fichiers ou chaines dont l'extension (après le ".") est une lettre majuscule ou minuscule.

d) `[!f]*`

Tous les fichiers ou chaînes ne commençant pas par "f"

e) `^..$`

Tous les fichiers ou chaînes dont la longueur est de 2 caractères (^ et $ indique le début et la fin du texte)

f) `^[A-Za-z]$|^[A-Za-z][0-9]`

Tous les fichiers ou chaînes avec une seule lettre (^ et $ indique le début et la fin de la chaîne de caractère) *ou* une lettre au début suivi de chiffres

g) `J'ai#malà_'{la+tête[]`

Prenez une aspirine :-)

V Manipulations de fichiers

Un fichier est un ensemble de données stockés ensemble dans la même enveloppe sur le disque. Il s'agit du regroupement le plus simple et le plus petit (un peu comme une feuille volante où vous avez écrit des informations).
Les données sont d'un format défini (texte, binaire, …) car utilisées avec des outils associés (comme si vous écriviez des feuilles de texte ou des feuilles de chiffres).

Le répertoire est comme un classeur où vous rangez vos feuilles volantes. Il stockera un ensemble de fichiers et n'aura pour rôle que de gérer hiérarchiquement les fichiers.
Il est défini comme un fichier mais son contenu stocke les points d'entrée aux fichiers placés dans ce répertoire.

FAITES TRES ATTENTION AUX MODIFICATIONS APPORTEES A UN FICHIER.

Il n'y a pas de corbeille comme sous windows ou de retour arrière après destruction ou déplacements.

Recopiez toujours un fichier que vous allez traiter en lui donnant une extension .ancien ou .avant_manip_perso_de_la_mort pour garder une image des données avant votre passage.

V.1 Identification logique d'un fichier

Nom d'un fichier

- **Majuscules et minuscules ont une signification**
- Tous caractères autorisés sauf /
- Éviter -, ?, *, [,], ", %, \, qui ont une signification particulière
- Un nom commençant par '.' est invisible de la commande de listage des fichiers 'ls' (pseudo caché)

Masques de saisie

Vous souhaiterez sûrement effectuer une opération (copie, destruction, …) sur de nombreux fichiers.

Soit vous écrivez tous les noms de fichiers soit vous pouvez les décrire par un masque de saisie (voir paragraphe précédent sur les manipulations de chaîne).

Par exemple, **lot*** désigne tous les fichiers dans l'arborescence dont le nom commence par lot

FAITES TRES ATTENTION AVANT D'EXECUTER DE TELLES ACTIONS .
Vérifiez d'abord que vous avez bien la bonne liste de fichier (en utilisant ce masque avec une commande d'affichage des fichiers) avant d'exécuter leur destruction ou leur déplacement ou toute autre modification.

Il n'y a pas de "retour arrière", d'annulation d'ordre passé ou de corbeille où vous retrouverez vos fichiers détruits par la suite !

Si vous utilisez le masque .* , tous les fichiers et répertoires dont le nom commence par un '.' sont concernés, en particulier le répertoire actuel et aussi … le répertoire parent (désigné par '..')
Si votre commande est une suppression, vous risquez de faire des dégâts sérieux.

Si vous listez d'abord (commande ls) les fichiers et répertoires concernés, vous éviterez pas mal de déconvenues.

DE MANIERE GENERALE, N'UTILISEZ JAMAIS * ou *.* (à moins d'être joueur)

V.2 Liste du contenu d'un répertoire (ls)

ls (options) (arg 1, arg2, ...)

Permet l'accès aux informations contenues dans un répertoire

Sans option, « ls » donne les noms de fichier.

```
>ls
.Xauthority  .sw          cop          lost+found   ps           tmp_mnt
.dt          SD_CDROM     dev          mbox         rapports     users
.dtprofile   TT_DB        etc          migration    sbin         usr
.profile     bin          ficinf       net          site         var
.rhosts      cdrom        home         opt          stand
.sh_history  cdromnfs     lib          prog         tmp
```

Principales options

-l affiche un maximum d'informations

```
>ls -l
total 228
-r--r--r--   1 bin        bin           1071 Dec 12 12:03 .profile
-rw-r--r--   1 root       sys              0 Mar  5 08:42 .rhosts
-rw-------   1 root       sys           3400 Mar 30 15:26 .sh_history
drwxr-xr-x   3 root       root          1024 Jun 10  1996 .sw
drwxr-xr-x   4 root       root          1024 Dec 12 16:24 home
```

-t trie les informations par date de modification plus récente

-rt trie les informations par date de modification plus ancienne

```
>ls -lt
total 228
dr-xr-xr-x   2 root       root             2 Mar 30 15:29 net
-rw-------   1 root       sys           3432 Mar 30 15:29 .sh_history
drwxr-xr-x   8 root       sys           1024 Mar 30 15:16 users
-rw-r--r--   1 root       sys          62181 Mar 26 09:10 mbox
dr-xr-xr-x  15 bin        bin           3072 Mar 19 16:16 dev
drwxr-xr-x   2 root       sys             24 Mar 16 17:47 cdromnfs
-rw-r--r--   1 root       sys              0 Mar  5 08:42 .rhosts
drwxr-xr-x   6 cft        cft           1024 Mar  3 12:05 prog
dr-xr-xr-x  12 bin        bin           2048 Feb 25 10:53 sbin
```

Le résultat retourné par la commande ls marque les fichiers ordinaires d'un '-' dans la première colonne de chaque ligne

Ce '-' peut être remplacé par

d : pour un répertoire

l : pour un lien symbolique

c : un fichier spécial mode caractère

b : un fichier spécial mode bloc

```
lrwxr-xr-x   1 root       sys              6 Mar  3 12:05 cft -> cft220
drwxr-xr-x  23 cft        cft           1024 Mar  5 16:26 cft220
brw-r-----   1 root       sys          28 0x002000 Jun 10  1996 c0t2d0
crw-r-----   1 root       sys         177 0x002000 Jun 10  1996 c0t2d0
```

Exemples :

ls -ldA .* ne voir que les fichiers commençant par .*

ls -rt voir les fichiers triés par dates plus anciennes

ls -ld * | grep ^d

ne voir que les répertoires (^d indique les lignes avec premier caractère égal au caractère "d", le sigle ^étant utilisé pour désigner le début de chaîne de caractères)

V.3 Droits sur les fichiers et répertoires

Dans un premier temps, nous allons voir que les fichiers sont accessibles suivant des critères d'autorisation.
Un répertoire peut ne pas être visible ou un fichier peut vous être interdit en écriture …

Fichiers ordinaires

Chaque fichier possède un masque de protection composé de 3 blocs, chaque bloc pouvant contenir 3 permissions. Ce masque est visible par la commande 'ls -l'

Les 3 permissions possibles de chaque bloc peuvent être, de gauche à droite :

- Accès [autorisé ou refusé] en lecture : "r"
- Accès [autorisé ou refusé] en écriture : "w"
- Accès [autorisé ou refusé] en exécution : "x"
- Acquisition [autorisée ou refusée] de l'identification du propriétaire (ou du groupe de propriétaire) lors de l'exécution : "s" (set uid, gid) . Cette permission se situe à la place du "x" lorsqu'elle est concédée.

Les 3 blocs définissant les destinataires des privilèges sont de gauche à droite

- Le propriétaire du fichier
- Les utilisateurs de son groupe sauf lui-même
  (Cette notion de groupe permet de partager des données pour une équipe déterminée)
- Les autres utilisateurs sauf ceux de son groupe (n'importe qui)

Exemple (extrait de /prog/informix/etc)

```
-rw-r--r--   1 informix   informix       983 Oct  2  1997 sm_versions.std
-rwsr-sr-x   1 informix   informix      2127 Oct  2  1997 smi_6to7
-rwxr-xr--   1 informix   informix      3004 Oct  2  1997 smi_load
-rwxr-xr--   1 informix   informix      3036 Oct  2  1997 smi_unld
-rw-r--r--   1 informix   informix       978 Oct  2  1997 sqlhosts
```

fichier smi_load : Le propriétaire (informix du groupe informix) a tous les droits. Les utilisateurs de son groupe peuvent lire et exécuter le fichier. Les autres ne peuvent que le lire.

fichier sqlhosts : Tout utilisateur peut lire le fichier, mais seul le propriétaire est autorisé à le modifier.

Fichier smi_6to7 : Le propriétaire peut lire et écrire dans le fichier. Tout utilisateur exécutant le programme se voit conférer les droits du propriétaire pour la durée de cette exécution.
Les utilisateurs de son groupe peuvent lire le fichier. Tout utilisateur exécutant le fichier se voit conférer les privilèges du groupe.
Les autres utilisateurs peuvent lire et exécuter le fichier.

Identification dans l'arborescence des répertoires

- Chemin absolu : /prog/informix/etc
- Chemin relatif : informix/etc si le répertoire courant est /prog
- Caractères spéciaux : '.' désigne le répertoire courant
  '..' désigne le répertoire père

exemple :
ls /prog/informix/etc (liste d'un répertoire désigné par son chemin absolu)

Droits sur répertoires

Un répertoire peut être lu, écrit ou exploré ("r", "w", "x")

Lire un répertoire signifie "pouvoir connaître les noms des fichiers qu'il contient (ls)

Ecrire dans un répertoire signifie "pouvoir créer ou détruire un fichier ou sous-répertoire de ce répertoire.

L'écriture dans un fichier est possible s'il a l'autorisation "w" sur le fichier, même s'il n'a pas l'autorisation sur le répertoire.

Explorer un répertoire signifie "pouvoir accéder par la commande cd et pouvoir accéder à un fichier dont le chemin d'accès passe par ce répertoire.

Changements de permissions (chown, chgrp)

A sa création un fichier est déclaré appartenir à un utilisateur faisant partie d'un groupe

```
Prompt shell> ls -l
total 2
-rw-rw-rw-   2 test1      formation      128 Mar 31 09:34 courrier
```

Le fichier courrier appartient à l'utilisateur test1 qui lui-même appartient au groupe formation.

Il est possible de changer ces paramètres et attribuer un fichier à un autre utilisateur du même groupe ou à un utilisateur d'un groupe différent.

- Changement d'utilisateur (commande **chown**)

Exemple : **chown test2 courrier**
(le fichier courrier appartient désormais à l'utilisateur test2)

- Changement d'utilisateur et de groupe (commande **chgrp** et/ou **chown**)

Méthode 1 : **chgrp users courrier**
(le fichier courrier appartient au groupe users)
chown applix courrier
(le fichiers appartient à l'utilisateur applix)

Méthode 2 : **chown users:applix courrier**
(changement d'utilisateur et de groupe en une seule commande)

Remarque : L'option -R permet d'effectuer le changement sur les sous arborescences. Cette option est dangereuse car elle risque d'effectuer des mises à jour non voulues (voir remarque sur les masques de fichier).

Changer les droits d'un fichier ou d'un répertoire (chmod)

Il est possible de changer ces droits si vous avez bien sûr le droit d'écriture sur cet élément.

Les droits sont gérés de deux manières :
Voir les paragraphes précédant pour rappel des droits sur un élément.

Soit avec des chiffres, soit avec des lettres

a) Gestion par chiffres

Chaque bloc (utilisateur / groupe / autres) est défini comme un octet :
r-x qui donne le droit en écriture et exécution correspond à '101' soit 5
rwx qui donne tous les droits correspond à '111' soit 7

Exemples :
Chmod 755 fichier met les droits sur fichier à rwxr-xr-x
Soit lecture/exécution pour tous, tous droits pour le propriétaire

b) gestion par lettres

u => utilisateur, g => Groupe, o => others (autres utilisateurs),
a => all (les 3)
\+ => rajoute le droit, - => le retire
r => droit en lecture, w => droit en écriture, x => droit en exécution

Exemples :
chmod o-w fichier retire le droit en écriture pour les autres utilisateurs sur "fichier"
chmod a+x fichier donne les droits d'exécution pour tous sur "fichier"

Remarque : L'option -R permet d'effectuer le changement sur les sous arborescences.
Cette option est dangereuse car elle risque d'effectuer des mises à jour non voulues (voir remarque sur les masques de fichier).

Initialisation des droits sur les nouveaux fichier (umask)

Chaque fichier nouvellement créé doit posséder des droits.
Cette valeur par défaut des droits est donnée par la variable d'environnement umask.

Pour la connaître, il suffit de taper la commande **umask**

La valeur de umask est inverse de celle que vous attendiez.
En effet, pour avoir des droits à rwxr-xr-x (c'est à dire 755), la valeur de umask devra être de --- -w- -w- (soit 022).

Cette valeur est obtenue en retranchant à 777 la valeur umask (en base OCTAL, c'est à dire que les chiffres de la valeur umask vont de 0 à 7).

Cette variable peut être modifiée par l'utilisateur en tapant **umask 033** pour avoir des droits à rwxr—r—(soit 744) par exemple .

La valeur umask est initialisée dans le fichier .profile de l'utilisateur et vaut 022 en standard.

V.4 Gestion des fichiers

Liens physiques de fichiers

Un fichier créé dans un répertoire peut être accédé depuis d'autres répertoires, sous un nom identique ou différent par la commande 'ln' afin **d'en faciliter le partage**.

Arbo niveau 1	Arbo niveau 2	Arbo niveau 3
/users	Jean	Note divers repertoire_perso
	Andre	lettre autre_fichier.doc

ln /users/Jean/Note /users/Andre/lettre

Dans le répertoire "Jean", **le fichier "Note" a été créé et physiquement alloué.**
L'utilisateur "Andre" a seulement créé une **entrée logique** dans son répertoire de nom "lettre", et qui désigne en permanence le fichier physique "/users/Jean/Note".

Pour créer un lien taper la commande ln suivi du chemin du document d'origine et du nom du lien à créer.

Exemple : ln /usr/Jean/courrier /usr/Andre/lettre

La commande ls permet de visualiser ce lien

```
/users/test2>ls -l
total 2
-rw-rw-rw-   2 test1      formation      128 Mar 31 09:34 lettre
```

Notez le « 2 » qui désigne le nombre de liens du fichier. Sa totale suppression du disque nécessite de détruire les deux entrées de fichier « lettre » et « courrier » dans leurs répertoires respectifs.

- Nécessite la permission d'exploration sur le répertoire natif
- Un lien ne peut franchir un système de fichier
- Un lien peut pointer sur un répertoire ou un fichier

Liens symboliques de fichiers

Le lien physique est limité au même filesystem (espace disque).

Afin de traverser le filesystem de départ, un lien symbolique est bâti sur le même principe que le lien physique.
Toutefois, la destruction physique du fichier de départ ne détruit pas le lien symbolique qui ne pointe plus sur rien dans ce cas.

La syntaxe de création d'un lien symbolique est la même que pour le lien physique avec l'option « -s ».

```
ln -s <Fichier_existant>  <Nom_lien_symbolique>
```

ln -s /rep/fic /rep2/fic2
/rep2/fic2 s'utilise désormais comme /rep/fic

Types de fichiers (file)

La commande file scrute les premières lignes d'un fichier et retourne une information sur le type de ce fichier (Texte, Binaire , …) .
La probabilité d'erreur n'est pas nulle (il peut indiquer un fichier texte parce que le début est au format texte).

Exemple : **file /usr/bin/cpio**

Visualiser un fichier texte (cat , pg, tail, head)

ATTENTION : Bien s'assurer que vous travaillez avec un fichier texte.
Sinon, vous listerez sur la sortie standard (l'écran généralement) un scribouilli binaire incompréhensible qui peut bloquer votre connexion.

cat <nom fichier> : affiche le contenu du fichier à l'écran

more <nom fichier> : affiche le contenu du fichier à l'écran avec un défilement page/page

pg <nom fichier> : affiche le contenu du fichier à l'écran avec un défilement page/page
possibilité d'effectuer des recherches dans le texte

tail <nom de fichier> : affiche les 10 dernières lignes du fichier
tail –n <nom de fichier > : Affiche les n dernières lignes

head <nom fichier> : affiche les 10 premières lignes du fichier
head –n <nom de fichier > : Affiche les n premières lignes

Création de fichiers (touch, cat, echo, vi)

Il est possible de créer un fichier vide (enregistrement de son nom uniquement sans données) à l'aide de la commande **touch**

Exemple : **touch newfic**

Il est possible de créer un fichier contenant quelques lignes de texte à l'aide de la commandes **cat** et en utilisant la redirection (**>**)

Exemple :
```
cat > newfic
ceci est saisi a partir du clavier  (saisir un texte au clavier)
sur plusieurs lignes
^D            (finir la saisie avec ^D soit  <CTRL> + <D> )
```

De même il est possible d'ajouter ce texte à un fichier déjà existant à l'aide la redirection **>>**

Ajouter le contenu d'un fichier à un fichier déjà existant est également réalisé à l'aide de la commande **cat**

Exemple : **cat oldfic >> newfic**

Il est possible d'ajouter une ligne à un fichier à l'aide de la commande echo. Cette commande est généralement utilisée pour afficher un message à l'écran (exemple : echo bonjour). En utilisant la redirection (**>** ou **>>**) on crée un fichier contenant le texte (**>**) ou on ajoute ce texte à un fichier (**>>**)

Exemples :
```
echo ligne ajoutee >> oldfic
echo ligne cree > newfic
```

Editeur vi

==> Un mémento est fourni en annexe.

La méthode la plus utilisée sous UNIX pour créer un fichier texte est l'utilisation du programme **vi** .

Ceci n'est pas une commande mais un utilitaire installé sur tous les systèmes Unix. Il est possible évidemment d'utiliser d'autres programmes comme Emacs par exemple.

Attention : vi est un éditeur TEXTE. Ne modifier que des fichiers texte avec.

Exemple : **vi fichierAediter**

vi fonctionne en deux modes (commande et insertion).

Le programme démarre toujours en **mode commande.**

* Pour entrer en **mode insertion** (saisie du texte proprement dite), vous utiliserez la **touche "i".**

* Pour être en **mode commande** (actions générales : enregistrer, quitter, copier/coller des lignes, …), utilisez la touche **<ESC>**

Si vous ne savez plus où vous en êtes, tapez <ESC>. Vous êtes en mode commande.

quelques exemples de commandes (en mode commande) :

o	insertion d'une ligne au-dessous de la ligne courante
O	insertion d'une ligne au-dessus de la ligne courante
dd	effacement de la ligne courante
6dd	effacement de la ligne courante et des 5 suivantes
7yy	copie de la ligne courante et des 6 suivantes dans le buffer
p	écriture du contenu du buffer en dessous de la ligne suivante
:q	quitte vi
:q!	quitte vi sans sauvegarder
:w	sauve sous le nom de fichier courant
:wq	sauve et quitte

s substitution de chaîne

Exemple :10,20 s/courrier/lettre/g

(remplace la chaîne courrier par la chaîne lettre partout ou elle est présente dans les lignes 1à à 20 du fichier)

Si vous avez de nombreux fichiers à modifier, vous pouvez effectuer une modification en cascade :

*vi *.config*	*Edition des fichiers se terminant par '.config'*
<modif >	*Modification (vérifier le nom du fichier en bas)*
:w	*Ecriture*
:n	*Passer au fichier suivant (revient en <modif>)*
:q	*Quitter l'éditeur vi à la fin*

Ps : Un éditeur pleine page sous Windows avec émulateur FTP tel que **ULTRA EDIT** est une manière efficace de travailler avec de tels fichiers textes UNIX.
N'oubliez pas cependant que connaître vi peut vous dépanner dans des cas particuliers où vous ne pouvez pas passer par un un tel éditeur.

Déplacement et changement de nom de fichiers (mv)

Exemples : **mv fich1 fich2** Renomme le fichier fich1 en fich2. Si fich2 existe déjà il est détruit (sauf privilèges l'interdisant)

mv fich1 rep1 Déplace le fichier fich1 du répertoire courant dans le répertoire rep1

mv rep1 rep2 Renomme le répertoire rep1 en rep2 sans toucher les fichiers contenus dans ce répertoire

Copie de fichiers (cp)

- Méthode 1
  Copier un fichier dans un répertoire en gardant son nom d'origine. Si le fichier destinataire existe déjà son contenu est perdu.

  Exemple : **cp /site/test1 /users/test1**

- Méthode 2
  Copier un fichier dans un répertoire en changeant son nom

  Exemple : **cp /site/test1 /users/test2**

Remarque :

Les fichiers copiés sont déposés dans le dernier paramètre, par exemple
cp toto titi tutu /repertoire

Si vous désirez recopier un ensemble de fichiers et avoir en sortie un ensemble de fichiers, la commande cp ne l'acceptera pas, considérant le dernier paramètre (des fichiers) comme des éléments à copier.
Ainsi, cp titi titi*.doc ne sera pas accepté car la commande cp considère titi*.doc comme des éléments à copier alors que le but recherché est de donner une nouvelle extension aux fichiers titi**

Vous devez passer par une séquence de commande (script) dont voici le contenu :

```
PROMPT UNIX>for i in `ls titi*`
> do
> cp $i $i.doc
> done
```

Destruction de fichiers (rm)

Pour détruire un fichier utiliser la commande rm suivie du nom du fichier à supprimer. Ne peuvent être détruits que les fichiers ayant l'autorisation "w".

Exemple : **rm /users/test1/liste**

L'option -i demande une confirmation avant l'exécution de la commande

L'option -r permet de détruire récursivement le contenu d'un répertoire (voir le chapitre destruction de répertoires)

Remarque :

Faire très attention à cette commande sur de nombreux fichiers.
Testez d'abord les fichiers à supprimer avec la commande ls pour éviter des destructions irréparables.

```
ls titi*
< Vérification que les fichiers à supprimer sont ceux là >
rm titi*
```

*==> EVITEZ ABSOLUMENT rm * OU rm *.* NON MAITRISABLES !! ("." - répertoire courant - et ".." - répertoire parent - peuvent être impactés)*

Recherche de fichiers (find)

La commande find permet de rechercher à partir d'un chemin défini le ou les fichiers correspondants à un ou plusieurs critères de sélection.

Les critères de sélection les plus courants sont :

-print imprime le chemin d'accès aux fichiers trouvés
A METTRE ABSOLUMENT SI VOUS VOULEZ VOIR LE RESULTAT !
(En effet, sans ce paramètre, la commande find n'est utile que pour étudier le code retour : à zéro, il a trouvé un résultat, sinon il n'a rien trouvé).

-name recherche d'un fichier suivant son nom
-links n recherche les fichiers possédants n liens
-size +x recherche les fichiers de taille supérieure à x blocs
-size +100 ==> Fichiers de + de 50 K0 si bloc = 512 octets
-mtime n recherche les fichiers sur leur date de modification
avec n= +7 ==> inchangés depuis 7 jours au moins
avec n= -2 ==> modifiés les 2 jours derniers
-atime n recherche les fichiers sur leur date d'accès
-depth liste tous les fichiers de l'arborescence
-prune liste tous les fichiers du répertoire hors sous répertoires
-type c type d'élément cherché c vaut : f "fichiers", d "répertoires"
! négation

Attention : Mettez l'option -print en dernier car sinon, les autres paramètres seront ignorés !

Exemples : **find /usr/bin -name test -print** (recherche le fichier ou le répertoire test à partir de /usr/bin et affiche le chemin absolu si trouve)

find / -size +100000 -print (recherche à partir de la racine tous les fichiers dont la taille est supérieure à 100000 blocs)

find . -type f -mtime +3 -print
(recherche à partir du répertoire courant tous les fichiers ordinaires (type "f") dont la date de modification indique que le fichier est inchangé depuis 3 jours au moins)

find . ! -atime -2 -print
(recherche à partir du répertoire courant tous les fichiers ou répertoires dont la date d'accès indique une utilisation depuis plus de 2 jours au moins.
"! atime -2" est équivalent à "atime +2")

D'autres options vous sont offertes avec la commande find (voir man) :
* Usage de tests logiques pour combiner plusieurs tests (OU, ET)
* Action à mener sur les fichiers trouvés – Clause exec (à utiliser avec précaution)

Remarque :
La commande find en récupérant des noms de fichier peut être associée à d'autres commandes (cpio, …) grâce à un mécanisme de pipe (caractère |)
Il est préférable d'effectuer la recherche et la poser dans un premier fichier.
Ensuite ce fichier sera lu (commande cat) pour être pris en entrée de la commande cpio ou autre.
Ceci vous évite en plus certaines désynchronisations qui peuvent perturber l'ensemble (la commande cpio risque de ne traiter que les premiers fichiers fournis par la commande find de manière discontinu alors que la commande cat indique les fichiers à prendre en compte de manière continue).

Au lieu d'écrire **find … | cpio ……**
Ecrivez plutôt :
find … > liste.txt
cat liste.txt | cpio ……

Comparaison de fichiers (cmp, diff)

La commande cmp affiche les octets (en octal) et les numéros de lignes où les fichiers diffèrent

Exemple : **cmp /prog/conf.x25 /prog/conf.1298**

La commande diff affiche les lignes où les fichiers diffèrent.
L'option **-b** permet d'ignorer les blancs en fin de ligne.

Exemple : **diff /prog/conf.x25 /prog/conf.1298**

Comptage d'un fichier (wc)

La fonction wc associée à certaines options permet de compter :
-l le nombre de lignes
-w le nombre de mots (séparés par des espaces ou des tabulations)
-c le nombre de caractères

Par défaut les 3 options sont actives

Exemple : **wc -l /etc/hosts**
Compte le nombre de lignes du fichier /etc/hosts
(donne le nombre de logins possibles)

Recherche d'une chaîne de caractère (grep)

La commande grep recherche une chaîne de caractères dans le texte fourni en paramètre et affiche le résultat à l'écran.
L'option -v permet d'afficher les lignes ne comportant pas la chaîne recherchée.

Exemples :

grep cpio *.ksh
(recherche la chaîne "cpio" dans tous les fichiers se terminant par ksh)

grep -v "LV NAME" /tmp/lspv.rpt > /tmp/lspv2.rpt
(recherche les lignes sans la chaîne "LV NAME" dans le fichier /tmp/lspv.rpt et met le résultat dans /tmp/lspv2.rpt*)*

Le texte fourni en paramètre est généralement le nom d'un fichier pour analyser son contenu. Il est possible de filtrer d'autres textes tels que le résultat d'une commande.

Tri de chaînes de caractères (sort)

La commande sort trie le texte fourni en paramètre et affiche le résultat à l'écran.
L'option -r permet d'inverser le tri.

VI Notions de filesystems ou « systèmes de fichier »

Les données sont stockées sur disque.

La gestion des données s'effectue grâce à un **découpage du disque en partitions** appelées "filesystem"

Seul le super utilisateur (« root ») peut créer ou détruire des filesystems.

Les autres utilisateurs peuvent utiliser les "filesystems" qui sont accessibles via l'arborescence de répertoires.

Ainsi, à chaque endroit de l'arborescence où vous êtes, vous vous trouvez en fait sur un filesystem donné.

Si un message vous indique que l'espace disque est saturé, cela veut dire que le "filesystem" est plein.
Sur un autre répertoire associé à un autre filesystem, vous pouvez disposer encore d'espace.

- La racine (le répertoire « / ») est le point de montage du filesystem de départ où se trouvent les composants principaux du système d'exploitation.

- **A partir de ce point, d'autres répertoires sont créés et un filesystem est rattaché** (le mot utilisé est « monté ») au répertoire.

  Ainsi, l'utilisateur qui travaille dans la sous-arborescence /repertoire1/repertoire2 stocke en fait ses données dans un filesystem différent de / ou de /repertoire1 selon l'organisation voulue par l'utilisateur « root ».

Ceci permet une gestion très souple de l'espace disque alloué aux utilisateurs.

L'utilisateur U1 qui a un droit d'accès sur l'arborescence FS1 pourra y stocker ses données dans la limite de l'espace du filesystem associé sans pouvoir écrire dans un autre filesystem où il n'a pas d'accès.

Le transfert d'un filesystem à un autre point de l'arborescence est facile et l'augmentation dynamique de la taille d'un filesystem est un autre atout de cette organisation.

Par exemple :

/
est le répertoire racine associé à un filesystem F1 de taille 300 Mo
/REP
est un répertoire monté sur / associé à un filesystem F2 de 850 Mo
/REP/AUTRE
est un répertoire monté sur / associé à un filesystem F3 de 400 Mo

Ainsi, si vous travaillez sur /REP ou sur /REP/REP2, vous travaillez sur le filesystem F2 (le point de montage le plus élevé est /REP).

Sur /REP/AUTRE ou toute autre chemin commençant par /REP/AUTRE, vous travaillez sur F3.

Sur tout chemin différent de /REP (ou /REP/AUTRE), vous travaillez sur le point de montage le plus élevé, c'est à dire "/"

Mais comment savoir quel est le point de montage le plus élevé ?
Les commandes suivantes vous l'indiquent.

Commandes df (AIX) ou bdf (HP)

Ces commandes permettent de visualiser les volumes logiques montés, leur taille et le point de montage

« df -k » sur système AIX à la même fonction que « bdf » sur HP (l'option -k traduit les tailles en Ko).

Exemple :

```
/ # df -k
Syst. fichiers     Blocs 1024      Libre %Util    Iutil %Iutil MontÚ sur
/dev/hd4                8192       5296   36%      818    20% /
/dev/hd2              442368     346924   22%     5566     6% /usr
/dev/hd9var             4096       3496   15%      111    11% /var
/dev/hd3               36864      35368    5%       51     1% /tmp
/dev/hd1                4096       3920    5%       20     2% /home
/dev/d80q_rA          188416      49088   74%       24     1% /users/applix
/dev/prog             614400     195964   69%     7803     6% /prog
/dev/site             106496      39176   64%      400     2% /site
/dev/images           106496      92100   14%       88     1% /images
/dev/users            319488      32824   90%      591     1% /users
```

Ici, le « / » fait 8 192 Ko et il reste 5 296 Ko de libre.
Tous les répertoires mentionnés à droite sont en fait des points de montage de filesystem.

Chaque arborescence à partir du point de montage est lié au filesystem attaché.
Les données stockées dans /tmp et sous répertoires ne sont pas dans le même filesystem que /prog ou /site.

Pour les deux points de montage /users et /users/applix, il faut comprendre que toute l'arborescence commencant par /users/applix est sur un filesystem (taille 188 416 Ko).
Tous les fichiers sur /users et toutes les fichiers et répertoires dont le nom commence par /users sauf /users/applix sont dans un autre filesystem (taille 319 488 Ko).

Le résultat de la commande « bdf » sur HP est similaire :

```
/ # bdf
Filesystem          kbytes    used   avail %used Mounted on
/dev/vg00/lvol1      91669   63482   19020   77% /
/dev/vg00/lv_home     3925    2052    1480   58% /home
/dev/vg00/lvol7     159509   73904   69654   51% /var
/dev/vg00/lvol4     199381  120744   58698   67% /opt
/dev/vg00/lvol6     343061  220252   88502   71% /usr
```

Ici, l'espace de départ et libre est indiqué ainsi que l'espace occupé.

Grâce à de ces deux commandes, vous pouvez vérifier l'espace disque disponible sur le filesystem où vous travaillez.

Un message « **filesystem is full** » ou approchant indique une saturation de l'espace disque.

En cas de saturation, il faudra nettoyer les fichiers obsolètes ou les déplacer dans un autre filesystem (dans la limite de vos droits d'utilisateur) ou sur un autre support (transfert sur K7 magnétique à demander à l'utilisateur « root ») ou les compresser (action présentée plus loin).

VII Manipulations sur les répertoires

- **le répertoire courant est référencé par le caractère '.'**
- **le répertoire juste en amont du répertoire courant est référencé par '..'**
- **Si l'argument répertoire cible est absent, le contenu de la variable "HOME" est utilisé**

Création d'un répertoire (mkdir)

Pour créer un répertoire taper la commande mkdir suivi du nom du répertoire à créer. Le chemin peut être exprimé en absolu ou en relatif.

Exemples : **mkdir exemples**
(le répertoire est créé dans le répertoire courant)
mkdir /site/exemples
(le répertoire est créé dans le répertoire /site à la condition que celui-ci existe)

mkdir -p /site/test/exemples
(le répertoire est créé dans une arborescence /site/test. Si /site ou /site/test sont inexistants, ils sont automatiquement créés)

Les droits concédés au répertoire créé sont définis par la valeur courante de la variable umask, si celle-ci a été déclarée dans le fichier .profile

Destruction de répertoires (rmdir)

Pour détruire un répertoire taper la commande rmdir suivie du nom du répertoire à détruire. Ce répertoire doit être vide.

Exemple : **rmdir /site/test/exemples**
(les répertoires /site/test et /site ne sont pas touchés)

Il est aussi possible de détruire un répertoire avec la commande
rm –R <nom_répertoire>

La commande rm -fr permet de détruire un répertoire et tout ce qu'il contient. Cette commande est rapide, efficace mais ***dangereuse*** !!! A n'utiliser qu'avec beaucoup de précautions.

Changement de répertoire (cd)

Pour changer de répertoire utiliser la commande cd suivie du nom de répertoire cible

Exemples : **cd /usr/Jean**
cd ../Andre

Affichage du répertoire courant (pwd)

La commande pwd permet d'afficher le nom du répertoire courant.

Exemple : **pwd**
#/usr/Jean

Occupation des répertoires (du)

du permet de connaître la place occupée (en nombre de blocs) par les fichiers d'un répertoire. L'option -s permet de n'obtenir que le total. L'option -k permet d'obtenir le résultat en ko.

Exemple : **du /prog/informix**

La dernière ligne donne le nombre total de blocs.

VIII Gestion des archives / compressions

Vous pouvez stocker l'ensemble d'une arborescence dans un fichier archive ou sur K7.
Ce fichier archive pourra ensuite être comprimé (si vous l'avez mis sur disque).

Dans le cas d'une copie sur K7, **NOTEZ L'OUTIL UTILISE SUR LA K7** (tar, cpio, …).
==> **Notez bien les options que vous inscrivez avec l'outil** cpio ou tar pour bien préparer le désarchivage (prendre les mêmes options de taille de bloc , … que pour votre archivage).

Programme tar

options :

c	écriture
x	extraction
t	liste
v	verbeux (sinon, pas d'affichage écran)
f	selectionne le device en sortie
r	rajoute le fichier à la fin
u	rajoute le fichier à la fin sauf si celui-ci existe dans l'archive avec une date plus récente.

Exemple : **tar cvf /dev/rmt0 fich1 fich2**
Ici, le device /dev/rmt0 désigne le lecteur de K7 (voir l'administrateur de votre machine pour votre serveur) et les fichier fich1 et fich2 y seront copiés.

tar cvf archive.tar repert1 fich2 repert3
créé un fichier "archive.tar" où seront stockés les deux arborescences repert1 et repert3 ainsi que le fichier fich2

tar xvf archive.tar /repertoire/arrivee
extrait le contenu de "archive.tar" dans le répertoire "/repertoire/arrivee"

N'oubliez pas de mettre l'extension .tar à votre fichier !

Programme cpio

exemples d'options :

-i	lecture
-o	ecriture
-c	entête en ascii
-a	met à jour la date d'accès du fichier
-t	liste
-v	verbeux (donne la liste des fichiers)
-B	par bloc (taille du bloc dépend du système)
-u	"unconditionnel" ==> Ecrase la cible si existe
-d	"directory" avec sous répertoires

Exemple 1: **cpio -icvBdu < /dev/rmt/0m**

Dans le répertoire courant (par défaut), inscrire le contenu de la K7 (device /dev/rmt/0m sur cette machine).

Exemple 2: **find . -print > liste**
cat liste | cpio -oacvB > /tmp/archive.cpio

Récupérer la liste des fichiers du répertoire courant et des sous répertoires (commande find), puis effectuer leur stockage dans un fichier /tmp/archive.cpio

Exemple 3: **cpio -icvtB < /dev/rmt/0m**

Lire le contenu de la K7 (device /dev/rmt/0m sur cette machine).

N'oubliez pas de mettre l'extension .cpio à votre fichier !

Copie de fichiers avec conversion (dd)

Moins utilisée, cette commande sert à copier des flux de données.

La commande dd permet de lire des données et de le recopier après conversion selon des paramètres spécifiés.

Les options utilisées sont :

- ibs=blocksize taille des blocs lus en entrée
- obs=blocksize taille des blocs lus en sortie
- bs=blocksize taille des blocs lus en entée et des blocs écrits en sortie (ibs et obs inhibés)
- if=filename nom du fichier en entée
- of=filename nom du fichier en sortie
- count=n nombre de blocs lus et écrits
- conv=type type de conversion (ASCII, EBCDIC, lcase, ucase ...)

Exemple :

```
dd if=f1 of=/dev/rmt/0mn bs=2k
dd if=f2 of=/dev/rmt/0m  bs=10k
```

La procédure copie le fichier f1 sur une cartouche avec une taille de blocs de 2k, puis sur la même cartouche et à la suite (/dev/rmt/0mn) le fichier f2 avec une taille de blocs de 10K.

Compression des données (compress, uncompress, gzip)

Il peut être nécessaire de compresser un fichier pour le sauvegarder ou tout simplement pour limiter la place occupée sur disque si ce fichier est peu utilisé.

- Programme compress (extension de fichier .Z)

  Exemple : **compress courrier** (crée un fichier courrier.Z qui remplace le fichier courrier)

  Les fichiers ainsi compressés sont décompressés par la commande **uncompress**.

- Programme gzip (extension de fichier .gz)

  L'option -r permet de compresser une arborescence, -l donne la liste des composants archivés avec leur taux de compression, -t permet un test d'intégrité.

  Exemple : **gzip courrier** (crée un fichier courrier.gz qui remplace le fichier courrier)

  Les fichiers ainsi compressés sont décompressés par la commande **gzip -d** ou **gunzip**.

IX Communication

IX.1 Les protocoles de communication (IP)

Vous travaillez sur plusieurs machines mais comment arrivent-elles à vous suivre ?

Si vous changez de micro-ordinateur, vous pouvez continuer à travailler sous UNIX.

Il faut bien pourtant qu'il y ait un écho à vos requêtes sinon vous n'auriez rien qui s'affiche sur votre écran en retour, non ?
Vous ne vous posez pas ce genre de questions ? Bon … je continue quand même.

Peu de machines composaient les premiers réseaux.

> Toutes les machines étaient reliées entre elles par câble (pas de serveur centralisateur). Chaque nouvelle machine devait être rajoutée dans le paramétrage réseau sur toutes les autres machines (lourd, non ?).
> Si vous montez un petit réseau local chez vous avec Windows, vous serez contraint de définir un nom pour chaque ordinateur de la même manière.

Pour simplifier la gestion du réseau, plusieurs protocoles ont été élaborés dont le fameux Internet Protocol (IP).

Celui-ci est associé à Transfert Control Protocol (TCP) sous le sigle TCP/IP.

> *Ce protocole est basé sur des principes d'acheminement de l'information par découpage (on parle de "trames IP" pour désigner les morceaux d'informations qui sont regroupés chez le destinataire).*
> *C'est assez efficace et surtout IP est un protocole très SIMPLE à manipuler, à déployer, à auditer, à utiliser, à sécuriser et il est devenu une norme internationale de fait.*
>
> *De plus, il y a tout plein de services qui vivent sur ce protocole (dont le WEB, la toile d'araignée de l'Internet où les grosses firmes commerciales feront mouche).*

le service unix que vous emploierez le plus sur IP est **telnet** qui vous permet d'effectuer une session comme si vous étiez connecté à une console sur la machine.

Ce que vous avez à retenir de IP (ou TCP/IP selon les sources), c'est que chaque machine possède une adresse IP.

La forme de cette adresse est une suite de 4 chiffres de 1 à 255 du style
192.243.203.78

En gros (vraiment très en gros), les 3 premiers chiffres sont généralement l'adresse principale du réseau (ici 192.243.203) où vous et vos voisins sont rattachés.
Le dernier chiffre permet de distinguer l'ordinateur sur votre réseau local IP.

Cette adresse IP vous suit pendant vos opérations (ceci est transparent bien sûr).

Comment vous connecter via telnet à une machine ?

Tapez simplement **telnet 192.243.203.78**
pour demander une session sur cette machine.

Si cette machine est un serveur UNIX, le message d'invite à saisir votre login s'affiche.
Vous pouvez démarrer une session.

La commande ping (tester une adresse IP)

L'adresse IP que vous avez donné peut être erronée ou inactive (la machine est arrêtée).

Dans ce cas, votre commande échoue (tapez <CTRL>+<C> pour sortir si vous êtes bloqué).

Pour savoir si la machine est joignable par IP, utilisez la commande "ping"

==> tapez <CTRL>+<C> pour sortir

```
PROMPT UNIX >  ping 193.253.223.46
       PING 193.253.223.46: 64 byte packets
       64 bytes from 193.253.223.46: icmp_seq=0. time=11. ms
       64 bytes from 193.253.223.46: icmp_seq=1. time=0. ms

                     ==> tapez <CTRL>+<C> pour arrêter le défilement

       ----193.253.223.46  PING Statistics----
              11 packets transmitted, 11 packets received, 0% packet loss
```

Ici, la machine à l'adresse IP 193.253.223.46 est joignable en un temps optimal (inférieur à 20 ms).

Il se peut que la communication échoue :

```
PROMPT UNIX >  ping 193.253.223.46
       PING 193.253.223.46: 64 byte packets

              Pas de défilement ==> tapez <CTRL>+<C> pour débloquer

       ----193.253.223.46 PING Statistics----
              5 packets transmitted, 0 packets received, 100% packet loss
```

Tous les paquets d'informations envoyés ont été perdus (comprendre "n'ont pas eu d'écho").

Dans ce cas, la machine 193.253.223.46 n'est pas accessible via IP actuellement.
Ceci peut être dû à ce que votre propre machine n'est pas connectée au réseau : essayez ping sur d'autres adresses IP valables. En cas d'échec, contactez les secours !

Le fichier /etc/hosts (nom supplémentaire à l'adresse IP)

A terme, l'usage des adresses IP est fastidieux.
Il faut se souvenir des adresses IP peu mnémoniques et si l'adresse IP change pour des raisons de gestion du réseau indépendantes de votre activité, vous devez suivre ces modifications qui devraient être transparentes.

Vous communiquez avec la machine de développement UNIX, pas avec 139.12.54.186 après tout.

Pour faciliter les communications entre machines, un fichier géré par l'administrateur/trice fait la correspondance entre l'adresse IP et un ou plusieurs noms supplémentaires plus commode d'emploi : /etc/hosts
Ce fichier recense ainsi les machines importantes sur le réseau IP.

Son contenu est du type **<adresse Ip> <nom1> <nom2>** …

Dans la même ligne, les noms placés à droite de l'adresse IP pourront donc être réutilisés.

Attention, *il se peut que plusieurs lignes concernent la même adresse IP (après tout c'est un simple fichier texte et votre admistrateur/trice a peut être le retour chariot facile).*
Si ceci ne peut pas être corrigé (il y a trop de synonymes et une seule ligne ne suffit pas), certains programmes peuvent ne considérer que la première ligne d'où risque potentiel d'erreur à surveiller selon le programme employé.

Exemple de fichier /etc/hosts

```
#########################################################
# @(#)/etc/hosts
#########################################################
#
127.0.0.1       localhost       loopback
#
181.25.213.49    uiv6r049        uiv6r049.francetelecom.fr
181.25.213.20  uiv6r020        bac.francetelecom.fr
```

Le # permet de signaler le début d'un commentaire
(Le reste de la ligne est ignoré)
L'adresse IP 181.25.213.49 a pour synonyme uiv6r049 et uiv6r049.francetelecom.fr

Ainsi, sur la machine UNIX où se trouve ce fichier /etc/hosts

```
telnet 181.25.213.49
telnet uiv6r049
telnet uiv6r049.francetelecom.fr
```

sont des commandes identiques.

La notion de Domain Name Server (DNS)

L'informatique évoluant sans cesse, voilà encore un truc qu'il vous faudra maîtriser à fond pour briller devant votre hiérarchie à la pause café (pas trop noir le cirage … euh, le café).

Nous avons donc un fichier /etc/hosts paisible qui nous permet de nommer avec des mots à nous, et pas des bêtes chiffres, une machine déterminée.

Premier inconvénient :
Sur la machine de développement, la machine avec laquelle mon application communique s'appelle "courgette" parce que l'administrateur local est végétarien.
Sur la machine d'exploitation, elle est désignée, toujours dans /etc/hosts, sous le vocable "petite_puce" parce que l'administratrice de cette autre machine s'est grattée la tête pour trouver un nom valable.
Cet exemple peut paraître stupide mais le manque de normes et de concertation provoque ce genre de décalage.

Second inconvénient :
L'adresse IP de la machine X est changée.
Pour être propre, il faudrait que tous les fichiers /etc/hosts qui mentionnent cette machine X avec l'ancienne adresse soient mis à jour.
Or, ce fichier est la propriété de l'administrateur/trice pour que des petits malins, comme vous, ne taquinent pas sa patience.
Pas question de le modifier avec facilité de l'extérieur, sécurité oblige.

C'est pourquoi une couche supplémentaire (eh oui, on rajoute toujours une pile jusqu'à ce que …) appelée DNS va permettre de gérer les adresses mnémoniques de manière centralisée.

Une machine sur le réseau fait office de serveur DNS et traduit toutes les nom DNS en adresse IP. (pour information, *il y a aussi d'autres copies du serveur DNS en secours*)

Le principe est : Nous gérons un fichier /etc/hosts central.

La norme et le rafraîchissement de ce fichier dépendent ensuite de l'entreprise/service où vous êtes.

Ainsi, vous verrez des adresses vers des machines sans adresse IP et sans relation avec votre fichier /etc/hosts local qui fonctionnent.

En fait, s'il y a un seul truc à retenir, c'est qu'il y a plusieurs méthodes pour désigner un partenaire en IP.

IX.2 Les conversations entre utilisateurs/trices

Gardez le contact virtuel, les petits loups !

Qui est donc connecté ? (who)

Bonjour les espions en herbe …… Tachez de ne pas vous précipiter pour enquêter sur ses petits voisins …

La commande who vous indique qui est connecté.
L'option -u vous donne aussi soit l'adresse IP de départ, soit la session (des tx…. signifie une session X-Windows multi process)

who -u *Liste des utilisateurs avec adresse*

```
root       ttypb        mars 24 09:25  1:53  25810  tx011:0.0
root       network      mars 24 09:56  1:22  26987  tx011:0
root       pty/ttyqc    mars 24 09:56 19:47  27126
root       ttyqe        mars 24 09:56  1:22  27140  tx011:0.0
root       ttyr0        mars 24 09:56  0:50  27143  tx011:0.0
maurice    ttyr4        mars 24 10:11   .    27526  193.253.223.143
```

w (who amélioré)

```
1:54pm  up 43 days,  1:52,  5 users,  load average: 2.10, 2.12, 2.14      ①
User     tty           login@  idle   JCPU   PCPU  what
 ②        ③             ④      ⑤      ⑥     ⑦     ⑧
cmm      ttyp4        10:06am    11                -ksh
cmm      ttyp6         1:05pm                      w
root     ttyp7         1:19pm    16                -sh
```

① Indications générales (résultats de la commande uptime: dernier boot)
② Login
③ Terminal (généralement ttyp+numéro) qui permet de distinguer la session
④ Instant où la connexion a débuté
⑤ Temps total passé en connexion
⑥ et ⑦ Temps CPU consommé en total
⑧ Dernière commande

mail (conversation asynchrone)

A votre connexion, un message peut vous indiquer la présence de mail à votre attention sur la machine unix.

tapez : PROMPT UNIX> **mail**

Les messages avec leur origine et contenu défileront.
Vous pouvez les supprimer avec **'d'**, voir le suivant avec **'n'**, revenir au précédant avec **'-'**, sortir avec **'q'**, taper une commande shell en la faisant précéder de **'!'** *(pour répondre, faites !mail nom_user).*

Pour envoyer un mail à un utilisateur, tapez
PROMPT UNIX> **mail <adresse correspondant>**

Vous rentrez en mode saisie, tapez votre texte puis terminer par
<CTRL> + <D> pour terminer et envoyer ou
<CTRL> + <C> pour sortir sans envoyer.

Par exemple,

```
PROMPT UNIX> mail GrosBill
      Hello, …
      pov' blaireau
<CTRL> + <D>
```

Le message sera transmis à l'utilisateur GrosBill à vos risques et périls.

A moins que vous vous soyez connecté sur un login d'un collègue, n'oubliez pas que vos messages risquent d'être lus.

Pour envoyer un mail à un correspondant sur une autre machine, l'adresse du contact contient le nom de la machine :

```
PROMPT UNIX>    mail alex@machine.fr
```

==> Fonctionne sur certaines configurations (un système DNS doit être actif – voir l'administrateur).

Talk (conversation synchrone)

Sur la même machine, vous **pouvez discuter en direct avec une personne connectée en même temps.**

Pour quitter ce type de conversation, tapez sur <CTRL> + <C>

```
PROMPT UNIX> who -u
kbh        ttyp2        Nov 16 10:28  1:16  19912  192.140.27.144
root       ttyp4        Nov 16 10:41   .    20059  193.253.223.206
pbt        ttyp5        Nov 16 10:43  0:56  20082  192.140.27.143
alex       ttyp6        Nov 16 10:48  1:03  20141  192.140.27.144
pbt        ttypb        Nov 16 12:04  0:27  21017  192.140.27.143
pbt        ttyp8        Nov 16 10:54  0:26  20229  192.140.27.143
eeh        ttyp9        Nov 16 11:54  0:44  20823  192.145.167.15
```

relever le terminal en deuxième colonne

```
PROMPT UNIX>  talk root ttyp4
```

==> fait un message sur le poste de votre interlocuteur :

```
Message from Talk_Daemon@uiv6r009 at 16:17 ...
talk: connection requested by root@uiv6r009.
talk: respond with:  talk alex@uiv6r009
```

Tant que le correspondant n'a pas répondu, vous êtes bloqué en attente.
Pour arrêter, tapez <CTRL> + <C>

==> Si quelqu'un veut communiquer en direct, répondre pour rentrer en conversation directe, la requête s'affiche sur votre session et vous indique la commande à taper:

```
talk alex@uiv6r009
```

Discuter avec une personne sur une autre machine (uiv6r020)

```
talk root@uiv6r020
```

Fonctionne sur certaines configurations (Voir l'administrateur/trice).

write user (message synchrone)

Sans vouloir être bloqué dans la conversation, vous souhaitez émettre un message à un autre user.

```
PROMPT UNIX> who -u
kbh         ttyp2        Nov 16 10:28  1:16  19912  192.140.27.144
root        ttyp4        Nov 16 10:41   .    20059  193.253.223.206
pbt         ttyp5        Nov 16 10:43  0:56  20082  192.140.27.143
```

relever le terminal en deuxième colonne

```
PROMPT UNIX> write root ttyp4
      bla bla, et bli bli, pata coussin et patati et patata  ……
```

Terminer par **<CTRL> + <D>** pour envoyer ou **<CTRL> + <C>** pour arrêter sans envoyer.

Conseil : Eviter de trop taquiner le root, il peut sévir.

wall (message synchrone à tous les users connectés)

Une petite annonce à passer ? Un coup de blues ? Une âme sœur prévue sur les ondes numériques dans votre horoscope ?
Envoyez votre message à tous les gens connectés, effets garantis !

```
PROMPT UNIX>  wall

      bla bla, et bli bli, pata coussin et patati et patata  ……
```

Terminer par **<CTRL> + <D>** pour envoyer ou **<CTRL> + <C>** pour arrêter sans envoyer.

X Les processus

Généralités

Un processus est en fait une tâche administrée par le système.
Chaque action que vous menez peut générer autant de processus que nécessaire gérés par le système (impression d'un fichier, listage des fichiers d'un répertoire, recherche d'un fichier, …).

Connaître ces tâches en mémoire vous permettra principalement de savoir ce qui est actif ("lancé") de ce qui ne l'est pas.

Techniquement, on appelle processus l'ensemble :

- d'une zone de code exécutable partageable
- d'une zone de données non partageable
- de tables locales aux processus
- de buffers d'entrée / sortie contrôlés par le système UNIX

Chaque processus est caractérisé par un numéro unique, son 'pid'.

Il est également le fils d'un processus dont le numéro est noté 'ppid'.
Tous les process sont des descendants du processus unit de pid=1

X.1 Gestion des processus

Un nombre limité de processus peut occuper la mémoire. Les autres attendent leur tour sur une partie du disque (swap)

Tous les processus présents en mémoire ne peuvent s'exécuter simultanément.

Un seul des processus prêt à s'exécuter et n'étant pas en attente d'événements externes est sélectionné.

Cette sélection s'effectue selon le temps machine disponible la priorité de ce processus.

Cette priorité du processus est identique à tous les utilisateurs sur une configuration UNIX standard.

Elle peut être modifiée par la commande nice (voir plus loin) bien que cette action soit limitée et qu'elle doit être traitée avec précaution (un programme qui dure trop longtemps est peut être planté. Lui donner une plus grande priorité qui lui permettra de consommer davantage de temps machine risque de vous pénaliser).

Etat des tâches (ps)

La commande ps permet de lister les tâches en cours.

Options : -e afficher tous les travaux
-f afficher plus d'informations

Exemple : **ps –ef | grep CFT**

```
UID   PID  PPID  C    STIME TTY       TIME COMMAND
cft  4361  4352  0  Dec 29  ?         1:26 CFTTCOM
cft  4367  4364  0  Dec 29  ?         2:04 CFTTCPS
cft  4352     1  0  Dec 29  ?         0:00 CFTMAIN
cft  4358  4352  0  Dec 29  ?         0:00 CFTLOG
cft  4364  4352  0  Dec 29  ?         0:00 CFTTPRO
user1 14670 13126  1 14:32:02 ttyp9     0:00 grep CFT
```

Ces process permettent de voir que l'outil CFT est actif

On remarque sur la dernière ligne le process "grep CFT" créé qui est une tâche qui suit la commande 'ps' (donc présente dans l'extraction de la commande 'ps').

* l'**UID** désigne le propriétaire du process. En fait l'UID est un nombre mais il est retraduit par la commande 'ps' pour une meilleure lisibilité, le nom de l'utilisateur étant plus parlant.

* Le **PID** est le n° du process

* Le **PPID** est le n° de process parent (CFTMAIN est le parent de CFTTCOM, CFTLOG et CFTTPRO)

* **STIME** est le moment de début du process (ces process ont été lancés le 29 décembre) = START TIME.

* **TTY** est le n° de "session" (en réalité, on parlera plutôt de "terminal")

* **TIME** est la durée en temps machine consommée.

* **COMAND** est le texte de l'instruction qui a fait naître ce process.

Affectation d'une priorité (nice / renice)

La commande nice permet d'affecter une priorité à une commande lancée.

Les priorités sont codifiées de 0 (la plus haute) à 39 (la plus basse), la valeur par défaut étant de 20.

Exemple : **nice +10 cc** (diminue la priorité du programme cc : 30)
nice –10 cc (augmente la priorité du programme cc : 10)

Donc, vous indiquez la priorité au lancement de la commande.

Cette commande est très limitée et avait un rôle important sur les machines anciennes moins puissantes où l'utilisateur devait agir sur le système.
D'ailleurs, dans certaines versions d'Unix, seul l'administrateur peut augmenter la priorité.

Vous pouvez vous servir de nice pour diminuer la priorité d'un process peu important.

Dans certains UNIX, la commande **renice** permet de gérer la priorité d'un process en cours d'exécution (Faites "man renice" pour vérifier son existence et son paramétrage).

Destruction d'un process (kill)

La commande kill permet d'arrêter une tâche en mémoire (qui vous appartient, bien sûr !).

A moins d'avoir des prédispositions au meurtre gratuit, n'effectuer cette commande qu'en cas de véritable pépin.

Demandez conseil à votre administrateur/trice avant de rendre zombie vos processus (le mot zombie existe réellement dans le monde unix).

En ayant mal arrêté un programme, vous risquez de rendre instable vos applications et la machine.

Le process est déterminé par son PID (voir instruction ps).

Muni de ce nombre, vous allez pouvoir jouer les Atilla dévastateurs de process qui bouclent, tarpouillent ou cafouillent.

Exemple : **kill 4264**
(demande gentiment au process n° 4264 d'évacuer les lieux)

Si le récalcitrant ne veut pas partir, vous pouvez l'atomiser.

ATTENTION : Avec le paramètre -9, vous risquez de perturber le fonctionnement de vos applications.

A n'utiliser que si c'est très important et que votre administrateur/trice est sympa ou est en manque de chocolats.

kill -9 4264

(appelle la police pour embarquer l'indésirable process.
Le signal 9 n'a rien à voir avec le n° de téléphone d'un service d'urgence - *sauf, paraît-il, en Angleterre où 999 est l'appel à la police, Merci Patrick de gâcher ce document avec tes remarques déplacées !* -, ni avec le nombre des cavaliers de l'apocalypse)

X.2 Etude de la charge système (top sur HP, sar)

La machine peut vous sembler ralenti.
Le temps machine est en effet compté. des process gourmands ou zombis peuvent réduire les capacités de la machine.

Pour connaître le fonctionnement actuel de la machine, quelques commandes sont à votre disposition.

Vous pouvez contacter votre administrateur/trice préféré(e) si vous avez diagnostiqué des dysfonctionnements (soyez clairs et n'allez pas chatouiller vos correspondants pour un rien, vous risquez de saturer la capacité de fonctionnement de l'administrateur/trice autant que celle de la machine).

Pour plus de précisions sur les sorties des commandes, utilisez l'aide en ligne :
man commande

Commande top sous HP

Pour quitter le panneau de top, tapez **<q>** ou **<CTRL> + <C>**

```
machine_HP>  top

System: <Nom machine>                                   <date et heure>
Load averages: 0.01, 0.02, 0.03
125 processes: 124 sleeping, 1 running
Cpu states:
CPU   LOAD   USER   NICE    SYS   IDLE  BLOCK  SWAIT   INTR   SSYS
 0    0.02   0.6%   0.0%   1.0%  98.4%   0.0%   0.0%   0.0%   0.0%
 1    0.00   0.0%   0.0%   0.4%  99.6%   0.0%   0.0%   0.0%   0.0%
---   ----  -----  -----  -----  -----  -----  -----  -----  -----
avg   0.01   0.4%   0.0%   0.6%  99.0%   0.0%   0.0%   0.0%   0.0%

Memory: 13244K (7180K) real, 24000K (14836K) virtual, 38884K free  Page# 1/12

CPU TTY    PID USERNAME PRI NI   SIZE    RES STATE    TIME %WCPU  %CPU COMMAND
 0 pty/ttyp1 25125 cmm      158 20   224K   188K sleep    0:00  1.13  0.64 ksh
 0 pty/ttyp1 25147 cmm      178 20   632K   336K run      0:00  1.98  0.51 top
 1 rroot      3 root     128 20     0K     0K sleep   66:53  0.17  0.17 statdaemon
 1 rroot    375 root     154 20    32K    32K sleep   72:40  0.16  0.16 syncer
 0 rroot     56 root     100 20     0K     0K sleep   88:55  0.14  0.14 netisr
 1 pty/ttyp1 25124 root     154 20    88K   160K sleep    0:00  0.15  0.09 telnetd
 1 rroot 23581 oracle   156 20  9884K   340K sleep    2:16  0.07  0.07 ora_dbwr_d80q
 1 rroot     58 root     100 20     0K     0K sleep    0:03  0.05  0.05 nvsisr
 1 rroot    870 root     127 20   124K   132K sleep   19:40  0.05  0.05 netfmt
 0 rroot     14 root     138 20     0K     0K sleep    1:10  0.05  0.05 vx_iflush_thread
```

Plusieurs informations sont données et rafraîchies dans ce panneau.
Pour le quitter, tapez **<q> ou <CTRL> + <C>** .

Les premières lignes donnent un condensé clair de l'activité globale de la machine.
les valeurs cpu désigne le temps octroyé par le processeur de la machine aux différents processus. Le tri s'effectue sur la colonne %cpu qui lorsqu'il atteint + de 90% désigne un process ultra gourmand ou zombie non maîtrisé.

Vérifiez donc particulièrement les deux colonnes %cpu et COMMAND qui identifient le process et son activité.
La première colonne donne le nom du propriétaire.

Activité système par la commande sar

La commande **sar** vous donne différentes indications sur les flux entre composants systèmes.
Deux chiffres donnent la période d'étude en secondes : sar 2 5 étudie sur 5 périodes de 2 secondes l'activité cpu.

```
PROMPT  UNIX > sar 3 5

HP-UX uiv6r020 B.10.20 U 9000/879    02/04/00

14:44:53    %usr    %sys    %wio   %idle
14:44:56       0       0       1      98
14:44:59       1       1       1      98
14:45:02       1       0       1      98
14:45:05       0       0       0     100
14:45:08       0       1       0      99

Average        0       0       1      98
```

Ici le temps cpu n'est pas trop utilisé (1% entre le système et les utilisateurs) et les entrées/sorties %wio (écriture/lecture sur disque notamment) ne sont pas très gourmands en ressource.

La colonne %idle indique la ressource disponible (100% moins les autres colonnes).

Si %wio est très important, les écritures sur disque sont trop longues.

Une consommation prolongée élevée de %usr (> 20 %) indique des process trop gourmands.
Quant à %sys, c'est l'administrateur/trice qui peut intervenir.

-b vous donne des précisions sur les buffers
-d formule l'activité par device (disque, …)

XI Les imprimantes

Lister les imprimantes :

Les imprimantes actives sur la machine sont listées par la commande
lpstat -t ou **lpstat -a**

```
device for HP01A: /dev/null
remote to: HP5SI_1A on ivry02bur
uiv6r009: HP01A: ready and waiting
```

Ici l'imprimante est définie comme HP01A (elle est associée à l'imprimante HP5SI_1A sur l'autre réseau 'ivry02bur')

Elle est "ready and waiting" donc opérationnelle mais si son état est "down", cela ne veut pas dire qu'elle "donne" encore mais qu'elle qu'elle est "tombée" (du lit ?)
==> contactez votre administrateur/trice favori(e).

Imprimer un fichier : lp ou lpr

lp -d<nom_imprimante> fichier
Exemple :
lp -dHP01A fic.txt
imprime le fichier fic.txt sur l'imprimante HP01A

Imprimante par défaut :

Pour les utilisateurs qui veulent l'imprimante **mon_local** par défaut, mettre dans le .profile du compte unix:

LPDEST=mon_local ; export LPDEST

XII Fonctions avancées

Exécution programmée permanente de commandes (crontab)

Vous voulez effectuer des actions à des moments déterminés suivant une fréquence calendaire. Pour y parvenir, UNIX vous fournit un moyen simple : Le "cron"

Il vous faut avant tout avoir le droit d'utiliser ce service (voir votre administrateur/trice préféré(e)).

crontab -l

Affiche la liste des travaux planifiés pour le compte utilisateur

crontab < fichier_de_commande.txt

Mise à jour des travaux planifiés pour le compte utilisateur avec le fichier "fichier_de_commande.txt" qui respecte la syntaxe suivante de 5 champs séparés par des espaces :

```
⓪ ① ② ③ ④  ⑤
01  20  *  *  1-5  /users/applix/80q/programme.ksh
```

⓪ Minutes (0 à 59)

① Heure (0 à 23)

② jour (1 à 31)

③ Mois (1 à 12)

④ Jour de la semaine (0 à 6) 0 indique Dimanche, 1 Lundi, ...

=> Une * dans ces champs indique tous, la virgule permet d'indiquer plusieurs valeurs, le tiret indique une plage de valeurs (3-5 est équivalent à 3,4,5)

Si plusieurs jours sont indiqués (au travers des champs ②, ③, ④), tous sont considérés.

⑤ Instruction à lancer

Chaque ligne (sauf celles en commentaires commencant par #) décrit une action à mener à un moment donné.

```
#              Exemple de contenu d'un fichier  cron
# Epuration quotidienne des fichiers temporaires
01 20 * * 1-5 /users/applix/80q/sh/raz.sh > /dev/null
# Arret instance Oracle
00 20 * * 1-5 /users/applix/80q/sh/dbclose.sh 1> /dev/null 2>&1
# Lancement instance Oracle
00 07 * * 1-5 /users/applix/80q/sh/dbstart.sh  1> /dev/null 2>&1
```

A 20h01 (raz.sh), 20h00 (dbclose.sh) puis 07h00 (dbstart), tous les jours du lundi au samedi (valeurs 1-5), 3 scripts sont effectués dont les résultats sont jetés dans la poubelle (/dev/null) ainsi que les sorties erreurs (redirection de 1> et 2>).

Attention à ne pas lancer un script avec des variables ou un adressage relatif qui ferait échouer l'interprétation à partir du cron *(celui-ci ne lance pas le .profile du login lors de l'exécution).*

Si, par exemple, vous utilisez la variable $TITI dans votre script et que cette variable est positionnée dans votre .profile, elle ne sera pas actualisée par le cron et l'interpréteur de commande remplacera $TITI par rien du tout, ce qui peut être gênant.

Exécution différée exceptionnelle de commandes (at)

Comment effectuer une action à une heure donnée ?

Exemples : **at 2300 prog** Le process prog sera exécuté à 23h00
at now + 2 hours prog
Le process prog sera exécuté dans 2 heures
at -l Liste des travaux planifiés
at -r prog Le process prog planifié est supprimé

Il se peut que vous n'ayez pas le droit d'utiliser la commande at.
Offrez des chocolats à votre administrateur/trice préféré(e) pour avoir les droits sur cette commande.

Transfert de fichiers FTP

Si vous n'avez pas d'émulateur FTP sous Windows (pratique avec l'usage de la souris), vous devrez passer par le clavier (méthode décrite ici).
Des explications générales sur ce service de transfert vous sont présentées et peuvent servir même avec un émulateur graphique.

Le service ftp est l'interface pour le **transfert de fichiers.**

Ce service fournit une connexion à un ordinateur distant ainsi que des fonctions d'envoi et de réception de fichiers .

Connexion et déconnexion

Il y a donc deux machines : la machine distante et celle de départ ('locale').

*En cas de transfert exceptionnel entre deux machines, privilégiez ce service plutôt que **CFT** si le transfert est en IP (IP est un protocole de communication entre machines, demandez à votre voisin si vous travaillez en IP. S'il sait, ce sera sûrement un bon partenaire pour le tennis, sinon, ignorez le à la cantine).*

La connexion est établie automatiquement en mode commande.
Exemple : **ftp 198.143.223.12**
ou **ftp machine2**

A la connexion une identification et un mot de passe est demandé sur la machine distante (gestion de droits).

==> Cette connexion peut être refusée pour différents motifs (tester la communication IP par ping, service ftp non actif sur machine distante ou machine de départ l'administrateur vous a interdit l'usage de FTP,…)
Vous ne savez pas ce que ping veut dire ?
Relisez le chapitre sur la communication et arrêter d'interroger votre voisin … Il va peut être s'imaginer que vous jouez au ping … pong avec des questions étranges et il risque des répandre des rumeurs sur votre compte.

Mode de transfert

Le mode de transfert par défaut est **ascii**. La commande ascii définit ce type de transfert.
Plus rapide, le mode ascii n'est pas adapté au transferts entre systèmes UNIX et systèmes non-UNIX.

La commande **binary** définit un type de transfert binaire adapté aux exécutables et est plus sûr sur un fichier dont vous ne connaissez pas la teneur.

Pour passer de l'un à l'autre, tapez soit **ascii**, soit **bin** ou **binary**.

Privilégiez le mode binary.

Principales commandes de FTP

help ou ?	liste les commandes possibles
ftp (site) ou ftp n° IP	connexion
Login + pass	autorisation
binary ou ascii	Mode de transfert
dir	liste les fichiers du répertoire courant
cd /tmp/toto	va au répertoire indiqué (machine distante)
get/put fic	récupère ou envoie le fichier (répertoires courants des deux sites)
mget / mput *fic?	idem avec de nombreux fichiers
prompt off	Ne plus poser de questions (accélère les mget / mput)
open <adresse IP>	Nouvelle connexion
close	Fin de connexion
bye	quitter FTP
lcd /rep	change le chemin du répertoire d'origine (machine de départ)

Transferts pilotés par un fichier

Il est possible d'automatiser les commandes transmises au service ftp en mettant celles-ci dans un script interprété par ftp.

Exemple : ftp -i -v < /tmp/ftp.in

-i = mode interactif inhibé
-v = "verbose" mode verbeux (la commande renvoie des explications)

contenu du fichier ftp.in :

open 198.143.223.12	*adresse ip de la machine distante*
jean	*Utilisateur sur la l'hôte distant*
alfred2	*Mot de passe sur l'hôte distant*
cd essais	*Répertoire sur l'hôte distant*
bin	*mode binary*
*mget **	*récupération de tous les fichiers*
bye	*quitter ftp*

XIII Langage de programmation awk

Pour les plus motivé(e)s, voici un outil précieux (un mémo est fourni en annexe).

awk est un outil très pratique pour effectuer des opérations sur des fichiers textes.

Il travaille par lignes et vous permet l'emploi de tests, actions, … qui permettent une programmation rapide.

La présentation qui suit est volontairement succincte car beaucoup d'options rendent cet outil très puissant.

En gros, vous avez un **fichier texte en entrée découpé en champs**.

La sortie du programme awk est une extraction/modification de ce fichier.

Par défaut, cette sortie est l'écran (standard) mais vous pouvez la rediriger vers un nouveau fichier.

Les champs du fichier étudié peuvent être numériques, des opérations de calcul sont possibles.

Introduction

L'entrée traitée par awk est une **suite de lignes au format texte découpés en champ**.

C'est à dire qu'un séparateur (généralement l'espace et la tabulation) sépare des informations dans des lignes
Ces lignes ou enregistrements sont elles-mêmes séparées par un retour chariot.

Exemple d'entrée pour awk : Fichier toto.txt

```
maurice  : caramel mou : veste  :120  : 35
valerie      : orange : parapluie:60:40
corinne : croissant : oreiller:180:18
Jean Jacques        : croissant :  oreiller:  78: 65
```

Chaque champ donne
* le nom de la personne,
* son met préféré (eh, oui de la bouffe … on est foutu, on mange trop),
* un vêtement important dans la vie de cette personne
* deux valeurs numériques quelconques (appelées première et seconde par la suite).

Ca vous change de l'éternel exercice sur les sales airs des gens ou leurs notes à l'école.

Ici, chaque ligne du fichier est composé de cinq champs séparés par le caractère :
Des espaces ont été placés un peu partout pour montrer qu'ils n'ont pas de caractères de séparation systématique.

Si le séparateur était l'espace, des termes en plusieurs mots (caramel mou) ne seraient pas considérés comme dans le même champ et chaque espace signifierait "nouveau champ".

Il faudra changer donc le *séparateur de champ par défaut qui est l'espace* mis dans le paramètre **FS** *(file separator) en stipulant* **-F ":"**
(pour indiquer le caractère tabulation, prendre **"\t"** *)*

Pour désigner le *nombre de champs de la ligne en cours*, on utilise **NF** *(Number of Fields)*

Pour désigner le *nombre de lignes lues*, on utilise **NR** *(Number of Records)*

Pour désigner la *ligne en cours*, on utilise **$0**

Pour désigner les *champs*, on utilise les variables **$1 $2 $3 …..**
$1 désigne le premier champ (le prénom), $2 le second champ (le met préféré), …..

Note : Il est possible d'utiliser des variables au lieu des chiffres.
Par exemple $NF désigne le dernier champ (NF est le nombre de champ)

La sortie s'effectue par la commande print avec comme séparateur l'espace.
Ce *séparateur en sortie* peut aussi être modifié par le paramètre **OFS** *(output file separator)*

Le nom du fichier en entrée est mis dans la variable **FILENAME**

Il reste à préciser l'opération menée sur chaque ligne lue sous la forme

<test> <action>

Remarque importante :
==> Si aucun test n'est mentionné, l'action s'applique à chaque ligne lue.
==> Si aucune action n'est mentionnée, chaque ligne avec un test concluant est sortie.
==> Si vous ne mettez ni action, ni tests, laisser tomber awk et partez en vacances.

La structure de la commande est

awk paramètres ' test {action} ' fichier en entrée

Le résultat s'affiche sur la sortie standard (il est possible de le rediriger vers un fichier en rajoutant **> fichier_resultat** à la fin)

Il est aussi possible de mettre toute la séquence d'instructions à awk dans un fichier paramètre avec le paramètre **-f**.

awk -fFICHIER_PARAM **fichier en entrée**

Dans FICHIER_PARAM se trouve les différentes instructions sous la forme

test {action}.

Ceci vous permet de conserver/modifier/diffuser facilement vos paramètres awk.

L'exemple suivant vous montre rapidement une opération simple sur un fichier texte :

Recherchez toutes les lignes avec le mot croissant et donner en sortie le nom avec le met préféré puis multiplier les deux valeurs numériques..

```
awk -F":" '/croissant/ {print $1, " adore  ", $2, $4 * $5}' toto.txt
```

Résultat :

```
corinne   adore    croissant  3240
Jean Jacques          adore    croissant  5070
```

* Le paramètre -F":" indique que le séparateur est le :
* Le test /croissant/ est effectué sur chaque ligne.
Dans le paragraphe suivant, les différents tests possibles sont décrits
* Pour chaque ligne où le test réussit, on imprime le premier et le deuxième champ par (print $1, " aime ", $2)
* Une chaîne de caractère " adore " a été rajoutée pour montrer la concaténation simple des caractères.
* Les champs 4 et 5 sont multipliés.
(Si la multiplication n'aboutit pas - champ non numérique -, la sortie est zéro -pas de plantage !-).
* Le fichier lu en entrée est toto.txt

En une seule commande relativement simple , vous avez pu effectuer une extraction avec des calculs sur un fichier de départ brut.

Les tests

Opérateurs :

==	Egalité (2 signes = se suivant)		
>	supérieur	**>=**	Supérieur ou égal
<	inférieur	**<=**	inférieur ou égal
&&	ET logique	**||**	OU Logique
!	Négation	**!=**	Non égal

/chaine/ Teste si "chaine" est présent

~ /masque/ Teste avec une équivalence de méta caractères (voir le chapitre sur les manipulations de chaînes)

par exemple **$1 ~ /^[a-zA-Z]*$/**

teste que le 1er champ ne contient que des caractères alphabétiques.

Exemple :

! ($5 >= 40) && /moutarde/

La valeur du 5ème champ ne doit pas être supérieur ou égal à 40 et la chaîne de caractère "moutarde" doit être présente dans la ligne étudiée.

Dans l'exemple du paragraphe précédent, /croissant/ signifie "si le mot 'croissant' est dans la ligne en cours"

On pourrait écrire ' $2 == "croissant" {action}' dans la mesure où la recherche ne se porte en fait que sur le second champ, mais il faut trouver exactement "croissant" sans espaces avant ou après dans le second champ or il y a des espaces avant les séparateurs de champ dans l'exemple : Ces phrases auraient été ignorées.

Les actions

Action 1 : La sortie (print ou printf)

Vous pouvez modifier la forme de la sortie qui renvoie le contenu de la ligne en entrée par défaut.
Cette action est placée (comme les autres actions) dans le bloc entre { }.

Pour cela, la première commande simple est print.

print chaine1, chaine2, ….

Les chaînes de caractères sont concaténées.
Il peut s'agir de champs en entrée ($i) , de textes entre ", d'opérations entre champs, ….
La sortie est formatée selon les chaînes.

Remarque :

print " "	imprime une ligne vide (saut de ligne),
print	sans arguments imprime la ligne courante.

Une autre sortie peut permettre de gérer la structure affichée de manière plus rigoureuse:

printf ("Format de la ligne en sortie", valeur1 valeur2, ….)

"Format de la ligne en sortie" contient le texte à afficher tel quel ainsi que des indicateurs (précédés d'un %) pour afficher les valeurs selon un format défini.

Par exemple :

printf("%s adore %s et la multiplication donne %.2f \n", $1, $2, $4 * $5)

ainsi dans la zone format, les 3 valeurs en sortie sont marquées :

$1 ==> format %s chaine de caractère
$2 ==> format %s chaine de caractère
$4 * $5 ==> format %.2f nombre avec deux chiffres après la virgule

Le \n indique un saut de ligne

Attention : **avec printf, vous devez gérer vous même les sauts de lignes en sortie,** printf écrivant tout sans rajouter d'autres caractères que ceux voulus dans le format choisi.

Les formats les plus courants sont :

c Conversion en Ascii de l'entrée
f Nombre
d entier
s String (Chain de caractère)

Vous pouvez rajouter des tailles pour l'affichage :
%7.2f indique une sortie d'un nombre sur 9 caractères dont 2 derrière la virgule. L'alignement de vos chiffres sera plus clair.
%10s sortira la chaîne sur 10 caractères cadré à droite.
%-10s sortira la chaîne sur 10 caractères cadré à gauche.

Pour afficher le caractère % , mettez **%%**

Exemple :

```
{ printf "%-20s a %2.2f %% du total ", $1, $3/5 }
```

%-20s est joint à $1 qui sera affiché sur 20 caractères cadrés à gauche
%2.2f est associé à $3/5 qui sera affiché comme nombre sur 4 chiffres dont 2 après la virgule
%% est placé pour signifier la sortie du caractère "pourcentage".

Des caractères particuliers permettent de gérer les

sauts de lignes	\n	
sauts de page	\f	
tabulations	\t	
caractères particulier #	\#	utile pour \ / * % …

Action 2 : les clauses BEGIN et END

Les actions BEGIN { …… } et END { …… } permettent de procéder à des actions avant et après la lecture du fichier.

```
BEGIN { print "DEBUT DE LECTURE DU FICHIER ……" ; print " "}
{ print }
END { print "FIN DE LECTURE DU FICHIER" }
```

La phrase DEBUT DE LECTURE DU FICHIER …… sera affichée puis une ligne blanche.
Chaque ligne en entrée sera ensuite affichée.
La phrase FIN DE LECTURE DU FICHIER sera affichée après que le fichier en entrée aura été lu.

Action 3 : les actions IF et ELSE

Vous pouvez procéder à des tests directement dans le pavé action.

```
if (expression de test)
  instruction1
else
  instruction2
```

Action 4 : les actions plus sophistiquées

Dans les actions, vous avez pu constater que l'usage de l'affichage est simple à mettre en œuvre (commande print ou printf plus élaboré).
Les clauses BEGIN et END vous permettent de gérer des entêtes ou des fins de texte.

Toutes les actions sont mises entre { }
Plusieurs actions peuvent être effectuées
(séparées par des ; ou des "fins de ligne" ou <RETURN>)

Dans ces actions, vous pouvez utiliser des champs ou des valeurs awk : NR ou NF
Vous pouvez aussi employer des variables perso avec des noms à vous (chouette alors !)

Les opérateurs sont :

= affectation (mise d'une valeur)
+ addition **++** incrémentation
- Soustraction **--** décrémentation
***** multiplication **/** Division
^ exposant (x^y veut dire x exposant y)
% Modulo (x%y est le reste de x / y)
int(x) Partie entière de x tronquée à la valeur inférieure

Remarque :

Il faut bien sûr pouvoir compliquer la lecture pour dérouter le non initié. Ainsi, certain(e)s pourront écrire plus concis : $2 /= 1000 signifie $2 = $2 / 1000 ou val += 3 signifie val=val+3
Si vous placer l'opérateur avant le signe d'affectation =, alors awk comprendra que vous effectué l'opération sur la variable affectée.

C'est peu lisible et trompeur. ==> A éviter.

Exemple de petit programme :

Combien de gens aiment les croissants ?

Le paramétrage awk est mis dans un fichier texte param

Le contenu du fichier param étant :

```
BEGIN { FS = ":"  # Le separateur de champ est le  :
        Nb = 0    # Initialisation
                }
  /croissant/ {nb = nb + 1}
END { print nb, " personne(s) aime(nt) les croissants, ce qui fait ", nb/NR * 100 ,
" % du total des personnes" }
```

Une variable nb est initialisée pour compter le nombre de lignes où "croissant" se trouve. Elle est affichée à la fin.

Vous remarquerez dans la clause BEGIN deux actions séparées par un saut de ligne (sur la même ligne, il aurait fallu mettre un ;) ainsi que des commentaires.

Résultat

```
PROMPT unix >    awk -fparam toto.txt
2  personne(s) aime(nt) les croissants, ce qui fait  50  % du total des personnes
```

Note : Vous n'êtes pas obligés d'initialiser les variables.
Par défaut, awk les initialise à 0 ou espace.

Ainsi, vous pouvez employer des variables pour calculer des moyennes, sommes, etc …

Pour les **chaînes de caractères**, des fonctions sont acceptées (!) :

length(s) Longueur de la chaîne s (= entier)
index(s,t) Position de la chaîne t dans la chaîne s (= entier)
substr(s,p,n)
Retourne la partie de la chaîne s qui débute en position p sur n caractères (= chaîne de caractère). Si n n'est pas mentionné, de p jusqu'à la fin de la chaîne s
gsub(r,s,t)
remplace globalement la chaîne r par la chaîne s dans la chaîne t et renvoie le nombre de substitution (= entier). Si t n'est pas mentionné, cela concerne $0 (la ligne en cours)

Vous pouvez aussi employer des clauses plus perfectionnées telles que des fonctions (**rand()** qui renvoie une valeur aléatoire entre 0 et 1) des boucles (**WHILE, FOR**) et des **tableaux** (!) .

Nous n'étudierons que la boucle FOR car elle vous permet de traiter l'ensemble des champs si vous devez effectuer une opération sur tous les champs présents dans une ligne.

vous écrirez la boucle **for** comme suit :

```
for(initialisation; test; avant itération suivante)
  {action1; action2
   action3}
```

Si vous avez une seule action, les { } ne sont pas nécessaires.

a) initialisation n'a lieu qu'en début de boucle
b) si le test est OK, on passe à l'itération suivante (ligne lue)
c) si le test n'est pas OK, on effectue l'itération suivante en effectuant l'opération en 3ème paramètre

exemple :

```
{   for (i=1; i <= $3; i = i + 1 )
      print $i^2
```

==> Ce que fait cette boucle est en fait
pour chaque ligne du fichier en entrée, le 3ème champ est pris en compte.
Si sa valeur N est numérique, tous les carrés de 1² à N² seront affichés

entrée

```
              Martin second_champ 3
              Autre_ligne chaine2  2

1             <== N vaut 3, 3 lignes alors 1² puis 2² et 3²
4
9
1             <== N vaut 2
4
```

Bon, cet exemple est plutôt nul…
Un autre alors ?

```
{     somme = 0
      for (i=1; i <= $NF; i = i + 1 ) somme = somme +$i
      print somme
}
```

Imprime les sommes des champs de chaque ligne.

Comme vous pouvez vous en apercevoir, awk est un langage de programmation assez étendu que vous pouvez approfondir si le cœur vous en dit …

Astuces :

a) Pour effectuer des **comparaisons sur des éléments au format différent**, vous pouvez effectuer des conversions en rajoutant un opérateur :

```
$1 + 0     permet de forcer l'usage de $1 comme valeur numérique (rajoute zéro)
$1 " "     permet de forcer l'usage de $1 comme valeur alphabétique (rajoute un espace)
```

b) Pour aller plus vite, passer par des **commandes UNIX.**

Au lieu de demander à awk de lire toutes les lignes d'un **fichier volumineux** alors que vous ne souhaitez travailler que sur les lignes où le mot "croissant" est présent, utilisez la commande 'grep' d'unix beaucoup plus rapide que awk.

```
grep croissant toto.txt | awk –fparam
```

Vous pouvez utiliser des commandes tel que sort (tri) ou des redirections vers des fichiers pour dispatcher des données vers des fichiers différents

```
(print " … " > fic.txt)
```

Quelques exercices :

Le fichier en entrée est exemple.txt.
Le séparateur de champs est le caractère espace
Le séparateur de lignes est le <retour chariot>

La commande sera du type awk –fFICHIER_PARAM exemple.txt
Seul le contenu de FICHIER_PARAM nous intéresse.

Essayez d'écrire les paramètres à mettre en œuvre pour résoudre les demandes.

a) Compter les lignes, mots et caractères (considérant que chaque champ est un mot).

b) imprimer la dixième ligne

c) imprimer le dernier champ de chaque ligne

d) imprimer le dernier champ de la dernière ligne

e) Faire un traitement de contrôle : Message d'anomalie avec affichage ligne incorrecte
Si le 1er champ n'est pas de longueur 5
Si le 2ème champ dépasse 5 fois le 3ème
Comptez les anomalies

f) Imprimer chaque ligne après avoir remplacé chaque champ par sa valeur absolue

Propositions de solution :

a) Compter les lignes, mots et caractères (considérant que chaque champ est un mot).

```
{ Nb_mots = Nb_mots + NF
  Nb_caracteres = Nb_caracteres + length($0)
}

END { print "Il y a ", NR, " lignes et ", Nb_mots, " mots et ", Nb_caracteres,
" caracteres dans le fichier en entree" }
```

b) imprimer la dixième ligne

```
 { NR == 10 }
```

c) imprimer le dernier champ de chaque ligne

```
{ print $NF }
```

d) imprimer le dernier champ de la dernière ligne

```
{ champ = $NF }
END { print champ }
```

e) Faire un traitement de contrôle : Message d'anomalie avec affichage ligne incorrecte
Si le 1er champ n'est pas de longueur 5
Si le 2ème champ dépasse 5 fois le 3ème
Comptez les anomalies

```
BEGIN { Nb_ano = 0 }

lenght($1) != 5  { print "Anomalie sur ligne : ", NR, " 1er champ de taille
incorrecte, contenu : ", $0
Nb_ano = Nb_ano + 1
 }

$2 > 5 * $3
{ print "Anomalie sur ligne : ", NR, " 2eme champ non conforme, contenu : ", $0
Nb_ano = Nb_ano + 1
 }


END { print "Fin du traitement, " , Nb_ano, " anomalies" }
```

f) Imprimer chaque ligne après avoir remplacé chaque champ par sa valeur absolue

```
{ for( i=1 ; i <= $NF ; i = i + 1)  if ($i < 0) $i = - $i
  print
}
```

==> Pour chaque ligne lue, i parcourt les champs qui, s'ils sont des valeurs négatives, sont changés.
==> print est équivalent à print $0

XIV Le script shell

Plusieurs commandes du shell vous permettent d'automatiser des actions ou de déclencher des opérations sur contrôle préalable.

La commande test

Cette commande retourne un code vrai ou faux interprété par exemple par la commande if

commande	condition vraie si
test -f fich	le fichier fich existe
test -s fich	le fichier fich est vide
test -d fich	fich est un répertoire
test -b fich	fich est fichier spécial (mode bloc)
test s1 = s2	si chaînes s1 et s2 identiques
test s1 != s2	si chaîne s1 différente de s2
test n1 -eq n2	si entier n1 égale entier n2
test n1 -ne n2	si entier n1 différent entier n2
test n1 -gt n2	si entier n1 plus grand que n2
test n1 -lt n2	si entier n1 plus petit que n2

test expression peut s'écrire **[expression]**

Exemple:
[$nbligne -lt 12] retourne vrai si la variable nbligne est inférieure à 12

L'instruction if

Format :
```
if liste-commandes1
then liste-commandes 2
( else liste-commandes 3 )
fi
```

Si la dernière commande liste-commandes 1 a un status de sortie égal à 0, liste-commandes 2 est exécuté, sinon (si précisé) c'est liste-commandes 3 qui est exécuté.

Exemple :
```
if test -f  /tmp/exemple
then pg /tmp/exemple
else echo "Fichier /tmp/exemple introuvable"
fi
```

Il est possible d'imbriquer plusieurs conditions :

Format :
```
if liste-commandes 1
then liste-commandes 2
elif liste-commandes 3
then liste-commandes 4
(else liste-commandes 5)
fi
```

Exemple :
```
if test -f  /tmp/exemple
then pg /tmp/exemple
elif test -f /tmp/exemple2
then pg /tmp/exemple2
else echo "Fichiers /tmp/exemple et /tmp/exemple2 introuvables"
fi
```

L'instruction echo

echo imprime ses arguments avec un retour chariot après le dernier. Une chaîne de caractères disposée ente " peut contenir des caractères non imprimables :

```
\b     backspace
\t     tabulation
\n     retour à la ligne
\\     backslash
\c     pas de retour chariot
\n     caractère 8 bits de code ascii sur 1 à 3 digits commençant par 0.
```

Exemple :

```
echo "\033[2J Nom du fichier : \c"     Effacement de l'écran, affichage du message "Nom du fichier : "
```

et le curseur attend après :

L'instruction case

```
Format :     case $string in
             choix pattern1 )  liste-commande 1 ;;
             choix pattern2 | pattern3 ) liste-commande 2 ;;
             choix patternx )  liste-commande 3 ;;
             *) liste-commande 4 ;;
             esac
```

case compare la variable string aux différents pattern et exécute la liste de commandes associées en cas d'identité.

Le ;; sert à sortir du case une fois un cas reconnu. Le cas par défaut est représenté si nécessaire à la fin par *).

Quand on désire associer une même action à plusieurs patterns, on peut séparer ceux-ci par le caractère |.

```
Exemple :    case $choix in
             o | O ) mkdir -p /tmp/essai
                          echo "Repertoire /tmp/essai cree";;
             n | N ) echo "Repertoire non cree";;
             *) echo "Erreur de saisie;;
             esac
```

L'instruction exit

Cette instruction permet de quitter le process en cours sans exécuter les commandes qui suivent l'exit.
Elle permet de définir le statut de fin de programme.

```
Exemple :    if test -f /tmp/essai
             then
                    exit 1
             else
                    exit 0
             fi
```

Les instructions de boucle

L'instruction while

Format :
```
while   liste-commandes 1
do
        liste-commandes 2
done
```

A chaque passage liste-commandes 1 est exécuté. Si la dernière commande de cette liste renvoie un statut 0, liste-commandes2 est exécutée.

Exemple :
```
while test -f /tmp/flag
do
        echo "Fichier /tmp/flag présent"
done
```

Il est possible d'effectuer le test à l'intérieur de la boucle do...done. Dans ce cas on défini l'instruction while indéfinie à l'aide des variables true (toujours vrai) ou false (toujours faux).

On peut quitter à tout moment une boucle à l'aide de l'instruction break

Exemple :
```
while true
do
        if test -f /tmp/flag
        then
                echo "Fichier /tmp/flag présent"
        else
                break
        fi
done
```

L'instruction for

Format :
```
for var in liste-valeurs
do
        liste-commandes
done
```

La variable var balaye la liste de valeurs et à chaque passage exécute la liste de commandes. La liste est une suite de chaînes de caractères séparées par des espaces.

Exemple :
```
for rep in /tmp/x /tmp/y /tmp/z
do
        mkdir -p $rep
done
```

Ce traitement peut être saisi aussi en direct au PROMPT du shell :

```
PROMPT UNIX>for i in `ls /tmp/ti*`
> do
> cp $i $i.doc
> done
```

Pour chaque fichier du répertoire /tmp qui commence par 'ti', recopier ce fichier en lui donnant une extension .doc (la commande cp /tmp/ti* /tmp/ti*.doc ne fonctionnant pas)

L'instruction until

```
Format :    until liste-commandes 1
            do
                  liste-commandes 2
            done
```

La liste de commandes 2 est réalisée tant que la dernière commande de la liste 1 renvoie un statut différent de 0.

```
Exemple :   until test -f /tmp/fich
            do
                  echo "Fichier /tmp/fich inexistant"
            done
```

L'instruction break

L'instruction break provoque la sortie de la boucle la plus proche. La rupture de n niveaux de boucles imbriquées est obtenue par break n.

L'instruction sleep

suspend l'exécution de la procédure pendant n secondes.

```
Exemple :   echo "Montez la cartouche sauvegarde
            sleep 10
            cp /tmp/*   /dev/rmt/0m
```

L'instruction read

Lit une ligne au clavier et affecte le 1er mot à var1, le 2eme à var2 etc....

```
Exemple :   read choix
```

L'instruction shift

Cette instruction décale de une position sur la gauche la liste de paramètres d'une procédure shell

```
Exemple :   while test $1
            do
                  echo $1
                  shift
            done
```

Variables et Affectation du résultat d'une commande

Une variable est définie par un nom et est traduite par le shell lorsqu'elle est précédée d'un $

```
Varia=8            Mettre '8' dans la variable  'varia'
echo varia         affiche      varia
echo $varia        affiche      8
```

La valeur d'une variable est alphanumérique.
Il suffit de placer le signe = pour effectuer l'affectation.

!! ==> Ne pas mettre d'espaces autour du signe = , ces espaces sont interprétés par le shell.

Généralement, le nom d'une variable est construit avec des lettres et des chiffres courants et le caractère _

Si vous êtes joueur ou joueuse de nature, vous pouvez y glisser des caractères accentués ou spéciaux _ @ ^ { ~ et tout plein de trucs pour perdre du temps à déplanter.

**Pour utiliser un caractère spécial, faites le précéder d'un **

Par exemple, vous voulez mettre la chaîne de caractère "TOTO>" dans une variable mais le > est un signe de redirection et risque de provoquer un malentendu.
Tapez :

```
Variable=TOTO\>
```

et le shell mettra TOTO> dans votre variable.

Le résultat d'une commande peut être affecté à une variable en la backquotant

(caractère ` obtenu avec le bitonio <ALT GR> + <7> et un espace)

Ceci vous permet d'élaborer des outils simples, voire des applications rapidement sur votre environnement.

Exemples :

```
PROMPT UNIX>  d=`pwd`
PROMPT UNIX>  echo $d
/users/applix
```

La variable d prend le résultat de la commande pwd (donne le répertoire courant).

```
PROMPT UNIX>  uname -a
HP-UX uiv6r003 B.10.20 A 9000/806 914307222 two-user licence
PROMPT UNIX>  OS=`uname -s`
PROMPT UNIX>  echo $OS
HP-UX
```

L'imbroglio des pattes de mouches :

UNIX est très ouvert et ses subtilités ont permis à plus d'un vilain d'initier des formules secrètes et pratiquement illisibles pour le commun dans des rites scribouillards et parfois fumeux.

Lorsque les ' ,les `, les " et autres sigles espiègles jouent à cache-cache, se lit dans les yeux du créateur, une extase grandissante inversement proportionnelle au moral de celui ou celle qui va maintenir ce charabia.

Voyons ces siglons qui épinglent la curiosité du profane.

{Texte} permet de désigner clairement le nom de la variable (ici TEXTE)

Si vous écrivez **$TEXTE1** le shell cherchera la variable **TEXTE1** pour la valoriser. Si vous mettez **${TEXTE}1** , la variable cherchée sera **TEXTE** et la valeur aura le caractère "1" rajouté à la fin.
Ceci est intéressant si vous avez un chemin à écrire :
$variable/repertoire sera interprété comme la variable variable/repertoire alors que *${variable}/repertoire* distinguera $variable de /repertoire

`Texte` est le résultat rendu par la commande Texte

'Texte' est le délimiteur d'une chaîne de caractères constante

"Texte" est le délimiteur d'une chaîne de caractères interprétée

Mais c'est quoi, la différence exactement ?
Avec des " **Texte** sera interprété par le shell.
Toutes les variables (chaînes commencant par un $) ou les commandes seront valorisées à l'intérieur des " à la différence de '

Par exemple :

echo " Le repertoire courant est `pwd`"
donne **Le repertoire courant est /repert/rep2**
La commande pwd a été exécutée lors de l'interprétation par le shell

echo ' Le repertoire courant est `pwd`'
donne **Le repertoire courant est `pwd`**
La commande pwd n'a pas été exécutée lors de l'interprétation par le shell

Si la valorisation par le shell traite une variable non alimentée, le résultat est une chaîne vide.

```
if [ "$var" = "" ]; then
        echo "la variable var est non alimentee"
fi
```

Remarque :
Que se passe t'il lorsqu'une variable est vide ?

if [$var = "vrai"] then
est interprété et transformé en **if [= "vrai"] then**
En effet, $var ne donne rien, ce qui provoque une erreur de syntaxe !

Songez à
* mettre une valeur par défaut à l'initialisation (plus propre)
* tester en rajoutant les " que l'interpréteur de commande laissera.

if ["$var" = "vrai"] then
est interprété et transformé en **if [""= "vrai"] then**

Certain(e)s rajoutent un caractère concaténé (une lettre quelconque) au début ou à la fin de chaque critère pour éviter les cas de nullité mais c'est plus lourd et moins lisible.

Autre remarque :

Comment concaténer des chaînes de caractères dans différentes variables ?

Il existe un caractère de concaténation & mais l'usage de la commande est une solution assez pratiquée car passe partout :

Var2=`echo "debut ${Var1}"`

Var2 recevra bien le contenu de la chaîne "debut " (avec un espace à la fin) concaténée au contenu de $Var1.

Ainsi, vous ne risquez pas de vous tromper dans les caractères de concaténation, les espaces en trop, et faites appel à une fonction simple …

Lors de vos avancées, vous découvrirez des subtilités parfois déroutantes.

Faites bien attention aux caractéristiques de votre shell !

Comme pour la syntaxe des formules mathématiques, le shell interprète la phrase selon un ordre bien précis.
Interprétation des variables $ (sauf celles entre ') puis des commandes (les `) puis interprétation des méta-caractères (les * [^ …).

Evitez de faire des formules tarabiscotées, **utilisez des variables intermédiaires** (qui vous permettent d'ailleurs de tracer vos pérégrinations sur le chemin des élus d'UNIX) pour plus de lisibilité.

Testez vos scribouillis directement sur le shell pour voir les différences.

Exemples :

Afficher un format de date particulier dans une variable :

DATE_HEURE=`date '+%Y-%m-%d %H:%M:%S'`

Le format de date sera au format : 2000-01-27 12:00:57

Le rajout de ' pour les paramètres du programme date est nécessaire car celui-ci ne traite que le premier paramètre passé et l'espace entre le d et le % retirerait l'affichage de l'heure.

Afficher un prompt fonctionnel :
nom machine:nom user:répertoire courant

export PS1=`uname -n`:$LOGNAME:'$PWD'\>

Export : La variable PS1 sera connue de tous les sous-shells
uname -a donne le nom de la machine.
Il est entre ` pour récupérer le résultat de la commande
$LOGNAME contient votre nom d'utilisateur
$PWD contient le répertoire courant

Vous pouvez faire des essais pour voir les différences d'interprétation.

Si vous mettez pwd entre ` que se passe t-il ?

L'interpréteur transforme la variable $PWD en nom du répertoire courant (par exemple /repert/toto) puis tente d'exécuter ce résultat et normalement doit vous remettre un message d'erreur sur l'exécution de /repert/toto qui n'est pas un programme.

La commande expr

Cette commande évalue ses arguments et imprime le résultat. Les termes de l'expression évaluée doivent être séparés par des blancs. Les caractères spéciaux du shell doivent être précédés de \.

```
Exemples :   a=`expr $a + 1 `          incrément a de 1
             a=`expr $b / 1024 `
                    Le résultat de b/1024 est affecté à la variable a
             a=`expr $b \* 2 `
                    Le résultat de b*2 est affecté à la variable a
```

Exemple de script commenté

```
#Version S0F1:"@(#)essai.ksh"
#-----------------------------------------------
#Description    : analyse et sauvegardes
#
#Conditions     : cft non actif
#Utilisateur    : root
#Creation       : xxxxxx le 19/10/1998
#Modification   : Contrôle de l'arrêt de cft - 07/11/1998
#-----------------------------------------------
```

Les lignes commençant par un # sont des lignes de commentaire, et par conséquent non interprétées par le shell. Ces lignes son facultatives mais permettent aux personnes visualisant le script de connaître son objectif, le nom de l'auteur, la ou les modifications apportées au script depuis sa création, et les conditions d'exécution (compte utilisateur etc...)

```
ANAREP='/tmp/essai'
ANATEMP=$ANAREP'/temp'
TMP1=$ANATEMP'/essai.tmp1'
TMP2=$ANATEMP'/essai.tmp2'
```

Définition de variables utilisées par la suite dans le script

ANAREP correspond au répertoire /tmp/essai
ANATEMP correspond au répertoire /tmp/essai/temp
TMP1 correspond au fichier /tmp/essai/temp/essai.tmp1
TMP2 correspond au répertoire /tmp/essai/essai/tmp2

```
OS=`uname -s`
```

On récupère dans la variable OS le type de système d'exploitation

```
echo "\033[2J\tTraitement en cours....\n"
```

On efface l'écran et après une tabulation on affiche le message "Traitement en cours..." suivi d'un retour ligne

```
NbProcess=`ps -ef | grep CFT | grep -v "grep CFT" | wc -l`
```

On regarde combien de process cft sont actifs et on affecte le résultat à la variable NbProcess

ps -ef liste tous les process actifs
grep CFT sélectionne les process cft
grep -v "grep CFT" on extrait la ligne correspondant au process grep
wc -l compte le nombre de process

```
if [ $NbProcess -gt 0 ] ; then
  echo "CFT est actif - Veuillez le stopper"
  exit
fi
```

Si le nombre de process est supérieur à 0 c'est qu'au moins 1 process cft est actif. Dans ce cas on affiche un message et on abandonne la procédure.

```
echo "\nLe repertoire /tmp va etre vide"
```

Affichage d'un message précédent d'un retour ligne pour être certain que le message s'affiche à gauche

```
while true
do
  echo "\nConfirmez par O pour Ok ou Q pour quitter"
  read choix
  case $choix in
  o|O)
       rm -fr /tmp/*
       break
       ;;
  q|Q)
       exit
       ;;
  *)
    echo "Erreur de saisie"
    ;;
  esac
done
```

Boucle attendant une réponse par o, 0,n ou N.
si la réponse est différente, le message "Erreur de saisie est affiché", puis le message de confirmation affiché à nouveau
Si o ou O est tapé, le répertoire est vidé et on quitte la boucle.
Si q ou Q est tapé, la procédure est abandonnée.

```
#
# CREATION DU REPERTOIRE DE TRAVAIL
#
mkdir -p $ANATEMP
```

On crée le répertoire /tmp/essai/temp

```
#
# APPEL DEPENDANT DE LA CONFIGURATION
#
echo "Analyse de la configuration\n"
if [ $OS = "AIX" ] ; then
  $ANAREP/aix_config.ksh 2>&1
else
  $ANAREP/hpux_config.ksh 2>&1
fi
```

Si la variable OS est égale à "AIX", la procédure /tmp/essai/aix_config.ksh est exécutée sinon on exécute la procédure /tmp/essai/hpux_config.ksh.
Les messages d'erreur sont redirigés sur l'écran

```
# Fin de la procedure essai.ksh
```

Exercices sur les scripts

exercice 1

Écrire une procédure ex1 pour tester si le fichier /tmp/param existe. S'il existe afficher son contenu à l'écran, si c'est un répertoire afficher un message à l'écran, sinon le créer à vide

exercice 2

Écrire une procédure ex3 testant si le fichier /tmp/param existe. S'il existe le renommer /tmp/param2 et sortir avec un statut égal à 0 sinon sortir avec un statut égal à 1.

exercice 3

Écrire une procédure ex2 explorant une arborescence et affichant pour chaque répertoire analysé, le nombre de fichiers et de sous-répertoires contenus

exercice 4

Remplacer la chaîne de caractères AEPCFG par la chaîne CFTCFG dans tous les fichiers d'un répertoire dont le nom est passé en paramètre au script

Solutions

Solution 1

```
file='/tmp/param'
if  [ -f  $file ] ; then
      pg $file
elif [ -d $file ] ; then
      echo "$file est un repertoire"
else
      touch $file
fi
```

Solution 2

```
file='/tmp/param'
if [ ! -f  $file ] && exit 1
mv $file $file"2"
exit 0
```

Solution 3

```
c=0
f=0
for fich in *
do
      if test -d $fich
      then c=`expr $c + 1 `; (cd $fich;/tmp/ex2)
      else f=`expr $f + 1 `
      fi
done
a=`pwd`
echo "\n >>> repertoire : $a"
echo "\t nb. de fichiers : $f"
echo "\t nb. de repertoires : $c"
```

solution 4

```
if [ $# -ne 1 ] ; then
             echo "Nom du repertoire absent"
             exit 1
fi
rep=$1
for fich in `ls $rep`
do
             grep AEPCFG $rep/$fich > /tmp/resultat
             if [ -s /tmp/resultat ] ; then
                    sed -e "1,$ s/AEPCFG/CFTCFG/g" $rep/$fich > /tmp/result2
                    cp /tmp/result2 $rep/$fich
             fi
done
exit 0
```

Annexe 1 Mémo de l'éditeur vi

Touches (de commande)	*Fonction*
	Déplacements
k *ou* ↑	déplacer le curseur à la **ligne précédente**
j *ou* ↓	déplacer le curseur à la **ligne suivante**
l *ou* →	déplacer le curseur d'une **position à droite** sur la ligne de commande affichée
h *ou* ←	déplacer le curseur d'une **position à gauche** sur la ligne de commande affichée
G *ou* **:$**	déplacer le curseur en **fin de fichier** (go to)
1G *ou* **:1**	déplacer le curseur en **1ère ligne**
n **G** *ou* **:n**	déplacer le curseur sur la **'n' ième ligne**
0	déplacer le curseur en **début de ligne**
$	déplacer le curseur en **fin de ligne**
<CTRL>f	déplacer le curseur à la **page suivante** (forward)
<CTRL>b	déplacer le curseur à la **page précédente** (backward)
	Effacement, remplacement de texte
x	**supprimer** le **caractère** affiché sous le curseur
D	**supprimer** la **fin de ligne** courante à partir du curseur (delete)
dd	**supprimer** la **ligne** courante
n **dd**	**supprimer 'n' lignes** à partir de la ligne courante
d1G	**supprimer** toutes les **lignes avant** la ligne courante
dG	**supprimer** toutes les **lignes après** la ligne courante
:n,md	**supprimer les lignes n à m**
:n,m s/*old***/***new***/g**	**remplace** old par new dans les lignes n à m (g force le remplacement de toutes les occurrences de old sur une même ligne, si non précisé seule la première occurrence trouvée sur chaque ligne est remplacée)
	Mode insertion (on en sort par <ESC>)
I	**insérer** des caractères **en début de ligne** (mode insertion)
i	**insérer** des caractères **avant le curseur** (mode insertion)
A	**ajouter** des caractères en **fin de ligne** (mode insertion, ajout)
a	**ajouter** des caractères **après le curseur** (mode insertion, ajout)
O	ouvrir une **nouvelle ligne avant** le curseur (mode insertion)
o	ouvrir une **nouvelle ligne après** le curseur (mode insertion)
R	**remplace** les caractères (mode insertion, refrappe)
r	**remplace** le caractère **courant** (mode insertion, refrappe)
cw	**remplace le mot** courant (mode insertion, change word)
	Copie et insertion de texte
Y	fonction **copier** : mémorise la **ligne courante** (dans le buffer principal)
n **Y**	fonction **copier** : mémorise les **'n' lignes** à partir de la ligne courante (dans le buffer principal)
p	fonction **coller** (paste) : copie le contenu du buffer principal
	Sauvegarde de fichiers et sortie
:w	**sauvegarder** le fichier en cours de modificatrion (write)
:w *fichier*	**sauvegarder** le fichier en cours de modification sous un autre nom
:w! *(fichier)*	**forcer la sauvegarde** (en cas de problème d'écriture disque)
:q	**sortir** de l'éditeur (quit)
:q!	**forcer la sortie** de l'éditeur (pas de sauvegarde !)
:x *ou* **:wq**	**sortir de l'éditeur en sauvegardant le fichier en cours de modification**
:n	**charger le prochain fichier** (relatif aux arguments de la ligne de commande " vi …. ")
:e *fichier*	**éditer un autre fichier** que celui en visualisation (p.ex. " :e ../config/config.dat ")
	Divers
u	**annuler** la dernière modification
.	**refaire** la dernière modification
/*chaîne*	**recherche** la chaîne de caractères (n pour afficher successivement chaque occurrence suivante)
:set nu	afficher les **numéros de ligne**
:set nonu	ne pas afficher les numéros de ligne
:set list	**afficher les caractères spéciaux** (notamment les caractères de fin de ligne, les tabulations)

Annexe 2 Mémo commandes UNIX

Touches (de commande)	*Fonction* / **niveau d'expertise obtenu à l'usage**
	Contrôleur de Touches
<CTRL>+<D>	Quitter proprement le process en cours
<CTRL>+<C>	Quitter brutalement le process en cours (break)
<CTRL>+<Z>	Fin de saisie de texte généralement (comme <CTRL>+<D>)
	Mémorisateur / Secoureur 1er degré
man commande	aide en ligne sur "commande"
	Démarreur (Pour la bleusaille)
passwd	changer le mot de passe
exit, quit, bye, logout	quitter (selon le programme en cours)
date	date et heure
uname -a	type de machine
hostname // pwd	nom de la machine (hostname) - Répertoire en cours (pwd)
	Gestionneur de son environnement
nohup commande	déclenche "commande" qui restera actif après déconnexion
& commande	déclenche "commande" en arrière plan (rend la main)
alias inst='commande -p'	remplace la saisie de *commande -p* par *inst*
	Niveau 1 Manipulateur de fichiers
type fichier1	Où se trouve fichier1 (recherche dans $PATH)
file fichier1	De quel type est fichier1
rm fichier1	détruire fichier1
find . -name "*.txt" -print	Recherche tous les fichiers se terminant par .txt
find . -mtime +10 -size +100 -print	Recherche tous les fichiers inchangés depuis 10 jours de taille 100 x "taille du bloc" (généralement 512 octets).
mv // cp fichier1 fichier2	Renomme (mv) / copie (cp) fichier1 en fichier2
cmp // diff fich1 fich2	Compare les fichiers
grep chaine fichier1	Recherche les lignes avec "chaine" dans fichier1
chmod 774 fic1 rep1	donne les droits 774 à fic1 et rep1
chown user2:group2 fic1 rep1	change le propriétaire et le groupe de fic1 et rep1
chgrp group2 fic1 rep1	change le groupe de fic1 et rep1
ln -s fic1 fic2	créé un lien (une doublure) de fic1 qui s'appelle fic2
	Niveau 2 Manipulateur de fichiers texte
vi fichier1	Editer fichier1 avec l'outil vi (voir autre annexe)
touch fichier1	créer fichier1 si inexistant ou actualiser sa date
pg fichier1 /cat fichier1	visualiser le contenu de fichier1
tail // head -10 fich1	10 dernières (tail) ou premières (head) lignes de fich1
	Niveau 3 Manipulateur de répertoires
cd .. cd repert1	Se déplacer dans un répertoire (.. est le répertoire père)
rmdir // mkdir repert1	détruire (rmdir), créer (mkdir) un répertoire
bdf // df -k	bdf (sous HP) et df (sur IBM) informent sur les FILESYSTEM
du repertoire1	donne la taille physique de l'arborescence
	Archiveur / Compresseur / anti-dépresseur (et ta soeur ?)
find . -print > liste **cat liste | cpio -oacvB > dest**	stocke l'arborescence (liste obtenu par commande find) dans dest (fichier ou K7)
cpio -icvtB < orig	écrit le contenu de orig qui doit être au format "cpio"
tar cvf archive.tar repert1 fich2	stocke l'arborescence et le fichier dans un format "tar"
tar xvf archive.tar	Extrait le contenu du fichier de format "tar"
compress // gzip fich1	compresse le fichier (.Z pour compress, .gzip pour gzip)
uncompress // gzip -d fich1	décompresse le fichier (.Z pour compress, .gzip pour gzip)
	Communicateur / Imprimeur / des primes l'heure !
w // who -u	Qui est connecté ?
mail user // talk user	ecrire un mail // discuter en direct
write user // wall	envoyer un message à un user (write) // à tous (wall)
lpstat -a	informations sur les imprimantes actives
lp -dIMPRIM fichier1	Imprime fichier1 sur IMPRIM
	Observateur système / Enerveur d'administrateur
ps -ef | grep chaine	visualise les process en cours où "chaine" apparaît
kill 152	arrête le process de PID (identifiant) 152
top // sar 1 5	informations systèmes (top seulement sur HP)
	Ecriveur / scripteur / Sheller (humaine)
echo $Toto // export Toto=5	Affiche le contenu de Toto // Met "5" dans Toto
${Var1}	Ecriture rigoureuse de la variable Var1 avec des { }
echo `commande`	les ` font exécuter la commande par le shell
echo "Texte" **echo 'Texte'**	Texte est affiché. Avec des ", s'il y a des commandes ou des variables, elles sont interprétées.
Var2=`echo "debut ${Var1}"`	exemple de concaténation de chaînes en utilisant echo

Annexe 3 Mémo commande awk

Fichier en entrée : Lignes de texte comportant des champs séparés par des espaces par défaut

==> Utilisation d'un fichier paramètre param.

awk -fPARAM fichier_entree > fichier_sortie

Pour aller plus vite sur un gros fichier, utilisez la commande grep
Vous ne désirez travailler que sur les lignes où "valide" apparait

grep valide fichier_entree | awk *paramètres*

Touches (de commande)	*Fonction*
	Variables définies par awk
$0 $1 $2 $var	$0 Ligne lue complète, $i est le ième champ
NF $NF	Nombre de champs, $NR dernier champ
NR	Nombre de lignes lues
-FS ":"	Séparateur en entrée est le : (par défaut espace)
-OFS ":"	Séparateur en sortie est le : (par défaut espace)
FILENAME	Nom du fichier lu
BEGIN {} END {}	Actions en début ou en fin de fichier lu
	Les tests
! ($5 >= 45) && /moutarde/	Le 5ème champ est >= 45 **et** la ligne contient "moutarde"
valeur == 78 || $4 != "carotte"	La variable est égale (2 signes = qui se suivent) à 78 **ou** le 4ème champ est différent de "carotte"
$1 ~ /^[a-zA-Z]*$/	Le 1er champ ne contient que des lettres
	Action 1 L'impression en sortie
print ""; print $4*8	Imprime une ligne blanche, puis le 4ème champ * 4
printf (" format %X ", variable)	%X est associé à variable pour la sortie
	%f Nombre %d entier décimal
	%7.2f nombre avec 2 décimales
Caractères spéciaux	\n saut de ligne, \f saut de page, \t tabulation
	\\ pour obtenir "\", \& pour obtenir "&", …
	Action 2 hors impression
if … else …	action selon test
int(x) x%y	valeur entière de x x%y (modulo) est le reste de x/y
length(s)	longueur de la chaine s
rand()	renvoie une valeur aléatoire entre 0 et 1
{ for(i=1 ; i <= $NF ; i = i + 1) if ($i < 0) $i = - $i print }	Boucle for : Imprimer chaque ligne après avoir remplacé chaque champ par sa valeur absolue

Exemple : *Le fichier en entrée a deux champs (séparateur tabulation).*
Le premier est le pays, le second, la superficie. Seules les lignes avec deux champs sont considérées. Un total de la superficie est souhaité en fin.

Commande : awk -fparam essai

Contenu du fichier param

```
BEGIN { FS=":"      # le caractere : devient le separateur en entree
        somme = 0   #  initialise somme a zero
        print "Traitement du fichier en entree de nom : ", FILENAME
      }
NF == 2  {  # teste si le nombre de champs est egal a  2
     somme = somme + $2   # $2 est rajoute a somme
        printf("Pays : %-20s Superficie : %10.2f  \n", $1, $2)
      }      # impression de la ligne avec un format
END { printf ("\n %10s     %6d\n", "TOTAL", somme) }
```

Entrée (fichier essai)	Sortie obtenue
ipays: 78:: trop: de champs Papouasie :100 Pomeranie_occidentale: 50 n'importe: quoi bidule Stroumphland: 50	Traitement du fichier en entree de nom : essai Pays : Papouasie Superficie : 100,00 Pays : Pomeranie_occidentale Superficie : 50,00 Pays : n'importe Superficie : 0,00 Pays : Stroumphland Superficie : 50,00 TOTAL 200