AIDE MEMOIRE UNIX

LES PRINCIPALES COMMANDES UNIX

QUELQUES COMMANDES SIMPLES

CONSULTER LE MANUEL

```
man [n] commande
```

Visualisation à l'écran des informations concernant la commande spécifiée. L'affichage est réalisé par un *more*.
Le manuel est divisé en huit sections:

- 1 : Les commandes utilisateurs
- 2 : Les appels systèmes
- 3 : La librairie des sous-routines
- 4 : Les formats de fichiers
- 5 : Les fichiers spéciaux
- 7 : Les possibilités diverses
- 8 ou 1m : Les commandes d'administrations système
- 9 : glossaire

On peut spécifier la section dans laquelle on veut effectuer la recherche (grâce au paramètre n).

IDENTIFIER LES UTILISATEURS DU SYSTEME

who Fournit des informations sur l'ensemble des utilisateurs qui sont actuellement connectés sur la station.

who am i Renvoi uniquement les informations relatives à l'utilisateur courant.

whoami Renvoi l'identificateur de l'utilisateur courant.

id Renvoie l'UID (user identifier), le GID (Groupe identifier) de l'utilisateur courant.

Il ne faut pas confondre who am i (cas particulier de la commande who) et whoami. Le premier donne des informations sur l'utilisateur connecté et le second l'identificateur de l'utilisateur courant.

CHANGER DE MOT DE PASSE

Pour se connecter, il faut :

- un login (identificateur de l'utilisateur) assigné par votre administrateur système
- un password (mot de passe) propre à chaque utilisateur

passwd Permet de définir et de contrôler son mot de passe.
A l'appel de cette commande, vous devez saisir l'ancien mot de passe, puis vous devez saisir deux fois votre nouveau mot de passe.

AFFICHER UNE CHAINE DE CARACTERES

echo chaine Affiche la chaine passée en paramètre. (Vous pouvez aussi afficher des variables: echo $PATH, pour visualiser la variable PATH)

banner chaine Affiche la chaine passée en paramètre avec des grosses lettres

COMMANDES ET MANIPULATION DES REPERTOIRES

VISUALISER LE CONTENU D'UN REPERTOIRE

ls [-FaRl] Liste le contenu d'un répertoire quelques options :

-F : Positionne à la fin des noms un / pour les répertoires et un * pour les fichiers exécutables

-a : Affiche tous les fichiers, y compris les fichiers cachés (ceux qui commencent par .)

-R : Affichage récursif

-l : Description complète du contenu d'un répertoire (une ligne par fichier). Le premier caractère de la ligne indique le type du fichier :

- : standard
- d : répertoire

-d : Evite de lister le contenu d'un répertoire : si rep est un répertoire, *ls -l rep* listera le contenu du répertoire rep, alors que *ls -ld rep* listera la description du répertoire

CHANGER DE REPERTOIRE

cd chemin Change le répertoire courant pour celui spécifié par le chemin.

cd Change le répertoire courant pour le home directory.

cd - Change le répertoire courant pour le répertoire précédent

pwd (print working directory) affiche le chemin du répertoire courant

CREER ET DETRUIRE UN REPERTOIRE

mkdir répertoire Création d'un répertoire contenant les deux fichiers . et ..

rmdir répertoire Supprime un répertoire vide (pour supprimer un répertoire non vide, il faut utiliser la commande *rm*)

COMMANDES ET MANIPULATION DES FICHIERS

VISUALISER LE CONTENU D'UN FICHIER

cat fich1 fich2 Concatène et affiche (sur la sortie standard) le contenu des fichiers

paste fich1 fich2 Concatène horizontalement les fichiers fich1 et fich2 et affiche (sur la sortie standard) le résultat.

more fich Visualise le contenu du ou des fichiers par page.
Pour un fichier contenant plus d'une page :

q ou **Q**	pour terminer la visualisation
RETURN	pour visualiser une ligne supplémentaire
ESPACE	pour visualiser la page suivante
H	pour obtenir de l'aide

IMPRIMER UN FICHIER

lp [-**d**imp] fichiers Imprime le ou les fichiers spécifiés sur l'imprimante par défaut ou sur celle spécifiée par l'option **-d** (attention, pas de blanc entre l'option et le nom de l'imprimante).

lpq Visualise la file d'impression courante

l**pstat** [-t] Renvoie des informations sur l'état de l'imprimante par défaut et de sa queue d'impression. (L'option **-t** permet de visualiser toutes les imprimantes)

cancel num_impression Détruit l'impression désignée par num_impression(vous récupérez ce numéro par la commande lpq ou lpstat)

CHERCHER UNE CHAINE DE CARACTERES DANS UN FICHIER

grep [-iv] expression fichiers

sans option	recherche dans les fichiers les lignes contenant l'expression
-i	pour ne pas tenir compte des majuscules/minuscules
-v	pour afficher les lignes ne contenant pas l'expression spécifiée.

FAIRE UNE COPIE D'UN FICHIER

cp source destination Copie le fichier source dans le fichier destination.
Si le fichier destination n'existe pas, il est créé. Sinon son contenu est écrasé sans avertissement.
Si la destination est un répertoire, alors la source peut être une liste de fichiers.

DEPLACER OU CHANGER LE NOM D'UN FICHIER

mv source destination Renomme ou déplace le fichier source en destination. Si la destination est un répertoire, alors la source peut être une liste de fichiers.

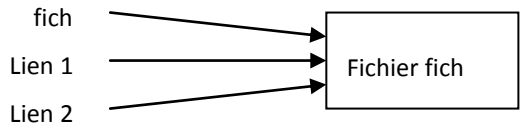
IDENTIFIER LE TYPE D'UN FICHIER

file fichiers Détermine le type du ou des fichiers spécifiés. (Attention, le résultat est parfois erroné)

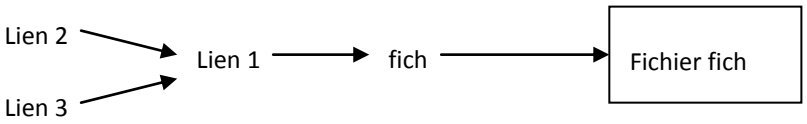
CREER UN LIEN SUR UN FICHIER

ln [**-s**] source destination Création un lien sur un fichier ou un répertoire. Un lien est un moyen d'accéder à un même fichier ou un répertoire sous plusieurs noms ou à partir de plusieurs répertoires. Attention un lien n'est pas une copie: si vous modifiez le fichier alors tous les liens sur ce fichier seront modifiés.
Il existe deux sortes de liens: le lien physique et le lien symbolique (avec l'option **-s**). Le lien physique ne peut adresser que des fichiers, alors que le lien symbolique peut aussi lier des répertoires.

ln fich lien1
ln fich lien2 ou ln lien1 lien2



ln –s fich lien1
ln -s lien1 lien2
ln lien1 lien3



Dans le cas de lien physique, pour effacer le fichier, vous devez effacer tous les liens qui pointent sur ce fichier. Par contre pour des liens symboliques, vous pouvez effacer le fichier sans effacer les liens, mais alors ceux-ci seront invalides.

DETRUIRE UN FICHIER

rm [**-irf**] fichiers Efface les fichiers (attention, on ne peut pas récupérer un fichier qui a été effacé)

- **-i** interactif, demande une confirmation sur chaque fichier
- **-f** force la suppression du fichier
- **-r** récursivité. permet d'effacer un répertoire et son contenu

RECHERCHER UN FICHIER

find rep -name nom **-print** Recherche le(s) fichier(s) caractérisé par *name* (vous pouvez utiliser une expression régulière), à partir du répertoire *rep* et affiche le résultat.
Vous pouvez décrire le fichier à rechercher par une expression régulière, ou indiquer le type de fichiers à chercher ou encore le propriétaire ...
vous pouvez aussi exécuter d'autres actions, comme effacer le fichier... (Pour plus de détails, voir le man)

PERMISSION SUR LES FICHIERS

Pour chaque fichier, il y a trois classes d'utilisateurs

Utilisateur	le propriétaire du fichier
Groupe	le groupe auquel appartient le fichier
autre	tous les autres

Les permissions accordées à ces trois classes sont :

r lecture
w écriture
x exécution (pour un fichier, peut être executé, pour un répertoire, peut devenir répertoire courant)

chmod mode fichiers Change les permissions du ou des fichiers/répertoires.
exemple mode désiré : rwxr-xr—

user	*group*	*other*	
rwx	r-x	r--	
111	101	100	en binaire
7	5	4	

D'ou la commande *chmod 754 fichier*

chown propriétaire fichiers Change le propriétaire du fichier. Le nouveau propriétaire doit être connu du système.

chgrp groupe fichiers Change le groupe du fichier. Le nouveau groupe doit être connu du système.

DIVERS

SE CONNECTER SUR UN AUTRE COMPTE

su [-] utilisateur Change l'utilisateur courant. Vous devez saisir le mot de passe du nouvel utilisateur (sauf si le compte d'origine est le root).
Si vous indiquez - alors les fichiers de login (.cshrc ou autre suivant le shell) sont exécutés et vous vous retrouvez dans le home directory du nouveau compte.

SE CONNECTER SUR UNE AUTRE STATION

rlogin station vous êtes connecte sue une autre station mais avec le même login. Vous devez saisir le mot de passe sur la nouvelle station (sauf si vous positionner le fichier .rhosts). Si vous ne saisissez pas le mot de passe, vous pouvez alors changer de compte.

SURVEILLER LES PROGRAMMES EN COURS D'EXECUTION

ps Affiche la liste des processus actifs. Attention les options de cette commande change suivant le système que vous avez, (vérifiez les par le man)

TUER UN PROCESSUS

kill num_process Supprime le processus spécifié (vous récupérez le numéro du processus par un ps).
Si malgré la commande, le processus n'est pas détruit, essayez *kill -9 num_process*

REDIRECTION DES ENTREES-SORTIES

< l'entrée standard est lue à partir d'un fichier

> La sortie standard est redirigée dans un fichier (RAZ du fichier)

>> La sortie standard est redirigée dans un fichier (concaténation du fichier)

La redirection des erreurs dépend du shell dans lequel vous êtes, soit sh , soit ksh ou encore csh

QUELQUES COMMANDES VI

L'éditeur vi est constitué de deux modes, un mode insertion et un mode commande. Lorsque vous lancez vi, vous êtes dans le mode commande.

Voici les commandes que j'utilise couramment.

DEPLACEMENT EN MODE COMMANDE

En général, suivant la configuration de votre terminal, vous pouvez utiliser les flèches pour vous déplacer. Si ce n'est pas le cas, vous pouvez utiliser les touches h j k l (qui elles fonctionnent toujours).

^ pour aller en début de ligne

$ pour aller en fin de ligne

:n pour aller à la ligne n (le : ouvre une ligne de commande en bas de l'écran et vous impose un ENTER pour exécuter la commande)

:$ pour aller à la dernière ligne

POUR PASSER EN MODE INSERTION

i insertion avant le curseur

I insertion en début de ligne

a insertion après le curseur

A insertion en fin de ligne

o insertion sur la ligne suivant

O insertion sur la ligne précédente

POUR REVENIR EN MODE COMMANDE

Une seule chose à utiliser : la touche ESC.

DIVERS

x effacer le caractère courant et le copie dans le buffer (attention, suivant votre configuration, la touche DEL peut avoir des actions surprenantes, à éviter)

r remplace le caractère courant (on reste en mode commande)

R remplace plusieurs caractères (On passe en mode édition mais en mode remplacement jusqu'à ce que l'on tape sur ESC)

dd efface la ligne courante et la copie dans le buffer

yy copie la ligne courante dans le buffer

p copie le buffer après le curseur ou la ligne courante (si le buffer contient des lignes obtenu par exemple avec la commande dd).

J concatène la ligne courante avec la ligne suivante

:u annule la dernière commande ou la dernière édition

:q quitte

:w sauve le fichier

:q ! force la fermeture du fichier sans sauvegarde

:n dans le cas d'édition de plusieurs fichiers, passe au fichier suivant

:/*chaine* recherche la chaine. Touche *n* pour les occurrences suivantes.

:s/*chaine1*/*chaine2*/ remplace la *chaine1* par la *chaine2* (voir la commande sed pour avoir plus d'options)

COMBINAISON DE COMMANDES LES PLUS UTILISEES

:wq sauve puis quitte

Déplacement d'une ligne : se positionner sur celle-ci, faire *dd*, puis se déplacer et faire *p* au bon endroit. Attention, une fausse manip est très vite arriver et on perd le contenu du buffer.

Copier une ligne : se positionner sur celle-ci, faire *yy*, puis se déplacer et faire *p* au bon endroit. Attention, une fausse manip est très vite arriver et on perd le contenu du buffer.

LES SHELLS UNIX : LE BOURNE SHELL

PROCESSUS SEQUENTIELS

```
proc1

proc2

proc3

proc1 ; proc2 ; proc3
```

PROCESSUS EN PARALLELES

```
proc1 & proc2 & proc3 &
```

REDIRECTION DES ENTREES-SORTIES

<	l'entrée standard est lu à partir d'un fichier
>	La sortie standard est redirigée dans un fichier (RAZ du fichier)
>>	La sortie standard est redirigée dans un fichier (concaténation du fichier)
2>	les erreurs sont redirigées dans un fichier
2>&1	les erreurs sont redirigées dans le même fichier que la sortie standard

LES PIPES

```
proc1  |  proc2
```

Équivaut à :

proc1 > fich
proc2 < fich

GENERATION DES NOMS DE FICHIERS

*****	n'importe quelle chaîne de caractères
?	n'importe quel caractère
[...]	n'importe quel caractère décrit entre les crochets

LE LOGIN

Exécution du fichier .login pour initialiser l'environnement

LES VARIABLES D'ENVIRONNEMENT DU SHELL

HOME	le home directory (répertoire de login)
PATH	chemin de recherche pour l'exécution des commandes
CDPATH	chemin de recherche pour la commande cd
MAIL	chemin indiquant le répertoire du courrier
PS1	primary system prompt
PS2	secondary system prompt
IFS	internal field separator
SHELL	indique le shell de login

L'EXECUTION D'UN SCRIPT EN BOURNE SHELL

sh nom_fichier ou rendre le fichier exécutable (chmod u+x nom_fichier) puis taper le nom du fichier
Pour forcer l'exécution du fichier en Bourne Shell, le fichier doit commencer par #!/bin/sh

sh -n nom_fichier	interprète les commandes sans les exécuter
sh -v nom_fichier	imprime les lignes comme elles sont lues
sh -x nom_fichier	imprime les lignes comme elles sont interprétées

LES VARIABLES

variable=valeur	affectation (Attention, ne pas mettre d'espace autour de =)
$variable	valeur de la variable
${variable**}**	valeur de la variable (permet d'éviter certaines ambigüités: si a="var",${a}b renvoie varb alors que $ab est invalide)

RECUPERATION DES PARAMETRES DANS LA LIGNE DE COMMANDE

$0	nom de la commande
$ n	nième paramètre
$#	nombre de paramètres
$*	liste de tous les paramètres

Pour décaler les paramètres, on peut utiliser la commande *shift*

QUELQUES VARIABLES SPECIALES

$$	le numéro de processus de la dernière commande

$?	Status de la dernière commande

LES CARACTERES SPECIAUX

****	banalise le caractère suivant
" ... "	banalise les caractères sauf \ , $ et `
'...'	banalise tous les caractères
`...`	substitution de commande

LES INSTRUCTIONS

Les commandes *if, while* , *until* teste le status de la commande. (Voir le man pour déterminer le status renvoyé par une commande particulière, en général si la commande s'exécute correctement, le status est vrai)
Attention mettre une négation devant la commande (if ! cmde) ne permet pas de tester l'échec de la commande car la négation s'applique au résultat de la commande et non pas à son status.
Dans cette optique, le test est considéré comme une commande, on peut d'ailleurs l'utiliser sur une ligne de commande.

IF

SYNTAXE

```
if cmde
        then
                liste_commandes
        [elif liste_commandes
                then
                        liste_commandes] ...
        [else liste_commandes]
fi
```

EXEMPLES

```
if test -f $1
then
        file $1
else
        echo " le fichier n'existe pas "
fi


if grep jean personnel
then
        echo jean >> disponible
elif grep pierre personnel
        then
                echo pierre >>  disponible
        else
```

```
                echo vide >>  disponible
fi
```

WHILE

SYNTAXE

```
while commande
do
        liste_commandes
done
```

EXEMPLE

```
while [ -r " $1 " ]
do
        cat $1 >> concat
        shift
done
```

UNTIL

SYNTAXE

```
until commande
do
        liste_commandes
done
```

EXEMPLE

```
until [ ! -r " $1 " ]
do
        cat $1 >>  concat
        shift
done
```

CASE

SYNTAXE

```
case para in
        choix1[|choix2] ... ) liste_commandes ;;
esac
```

EXEMPLE

```
case $1 in
        -d | -r ) rmdir $dir
                  echo "option -d ou -r ";;
```

```
        -o )        echo "option -o ";;
        * )         echo "réponse incorrecte ";;
esac
```

FOR

SYNTAXE

```
for para [in liste]
do
        liste_commandes
done
```

La variable *para* prend successivement les valeurs de la liste
si la liste est omise, *para* prend alors les valeurs passées en paramètres du script

EXEMPLES

```
for i in `ls`
do
        cp $i /dir/$i
        echo "$i copie "
done
```

N'oublier pas les ` qui force l'exécution du ls.

```
for dir in /dev /usr /users/bin /lib
do
        num=`ls $dir|wc -w`
        echo "$num fichiers dans $dir "
done
```

```
for i
do
        echo $i
done
```

LES TESTS

N'oubliez pas que le test est une commande qui peut être exécutée directement sur la ligne de commande
test expr ou [expr] (Attention il faut un espace après [et avant])
ou expr vaut :

-r fichier	vrai si le fichier existe et est accessible en lecture (R)
-w fichier	vrai si le fichier existe et est accessible en écriture (W)
-x fichier	vrai si le fichier existe et est exécutable (X)
-f fichier	vrai si le fichier existe et est un fichier régulier
-d fichier	vrai si le fichier existe et est un répertoire
-s fichier	vrai si le fichier existe et a une taille non nulle
-H fichier	vrai si le fichier existe et est un répertoire caché
-h fichier	vrai si le fichier existe et est un lien symbolique

s1 **=** s2 vrai si les deux expressions sont égales

s1 **!=** s2 vrai si les deux expressions sont différentes
s1 vrai si s1 n'est pas la chaîne nulle
e1 **-eq** e2 vrai si les deux entiers e1 et e2 sont algébriquement égaux
(autres comparaisons : -ne , -gt , -ge , -lt , -le)
! négation unaire
-a opération binaire ET
-o opération binaire OU

DIVERS

#	commentaires, mais #!/bin/sh en début de fichier force l'exécution en Bourne Shell
(cmde **)**	exécute la commande dans un sous-shell
read a	lecture d'une entrée pendant l'exécution d'un script
exit num	renvoie le status de la commande (en général 0 la commande s'est bien exécutée)
return num	code d'erreur
. script	exécution du script dans le shell courant
eval arg	interprète arg avant de l'exécuter
cmd1 **&&** cmd2	séparateur conditionnel (cmd2 sera exécuté si cmd1 s'est exécuté correctement)
cmd1 **||** cmd2	séparateur conditionnel (cmd2 sera exécuté si cmd1 ne s'est pas exécuté correctement)
nom_fonction () **{** liste_commandes **;}**	définition d'une fonction
exec arg	exécute la commande dans un nouveau shell
set var	initialisation d'une variable liste de tous les paramètres du système positionne les paramètres $i (set a b c positionne $1 à a, $2 à b et $3 à c)
unset var	raz d'une variable
type cmde	indique la localisation d'une commande
readonly var	empêche la modification d'une variable

LES EXPRESSIONS

expr exp *op* exp	exécute des opérations arithmétiques (op vaut : + , - , * , / , % , = , \> , \< , \> = , \< = , !=)
expr exp1 **|** exp2	renvoie exp1 si l'expression n'est pas nulle, sinon exp2
expr exp1 **\&** exp2	renvoie exp1 si l'expression n'est pas nulle, sinon exp2
expr exp1 **:** exp2	comparaison des deux arguments (renvoie le nombre de caractères en commun)

expr length exp	retourne le nombre de caractères de exp
expr substr exp n1 n2	retourne une sous chaîne de exp commençant a la place n1 et de n2 caractères
expr index exp car	retourne la position du caractère car dans la chaîne exp

L'expression est une commande donc pour affecter une opération à une variable, il faut forcer son exécution :

```
a=`expr $b + $c`
```

LES SHELLS UNIX : LE C SHELL

PROCESSUS SEQUENTIELS

```
proc1
proc2
proc3

proc1 ; proc2 ; proc3
```

PROCESSUS EN BATCH

proc **&**

REDIRECTION DES ENTREES-SORTIE

< l'entrée standard est lu à partir d'un fichier

> La sortie standard est redirigée dans un fichier (RAZ du fichier)

>> La sortie standard est redirigée dans un fichier (concaténation du fichier)

>& La sortie standard et les erreurs sont redirigées dans un fichier(RAZ du fichier)

>>& La sortie standard et les erreurs sont redirigées dans un fichier (concaténation du fichier)

LES PIPES

```
proc1 | proc2
```

Équivaut à

```
proc1 > fich
proc2< fich
```

proc1 **|&** proc2 : proc2 reçoit en entrée les sorties standards et les erreurs de proc1

GENERATION DES NOMS DE FICHIERS

***** n'importe quelle chaîne de caractères

? n'importe quel caractère

[...] l'intervalle décrit entre les crochets

{ ... } un des caractère décrit entre les accolades

~ le home directory

~user le home directory du compte user

LE LOGIN

Exécution du fichier .login et .cshrc pour initialiser l'environnement
.login est exécuté une seule fois lors du login
.cshrc est exécuté à chaque ouverture d'une session C Shell (ex: ouverture d'une fenêtre)
.logout est exécuté lors du logout

LES PARAMETRES DU SHELL

LES VARIABLES SHELL

variables locales au shell courant (en général en minuscule)
définie par la commande **set**

argv	les arguments de la ligne de commande (numérotées à partir de 1)
autologout	temps de déconnexion automatique d'un shell (en minutes)
cwd	le chemin du répertoire courant
home	le home directory
ignoreeof	si elle n'est pas initialisée, on ne peut pas se déconnecter par ^D
cdpath	chemin de recherche pour les commandes cd, pushd, chdir
notify	si elle est positionnée, l'utilisateur est immédiatement avertie à la terminaison d'un processus en tache de fond. Sinon il est avertie lors d'un shell prompt
path	chemin de recherche pour l'exécution des commandes
prompt	positionne la valeur du prompt
shell	indique le shell courant
status	retourne le status de la dernière commande exécutée (0 si elle s'est bien exécutée)

LES VARIABLES D'ENVIRONNEMENT

variables globales exportées aux sous shells (en général en majuscule)
définie par la commande **setenv**
trois variables sont automatiquement importées et exportées

PATH	path
TERM	term
USER	user

L'HISTORIQUE DES COMMANDES

set history = 20	créer un buffer pour enregistrer les 20 derniers commandes
set savehist = 15	sauvegarde les 15 derniers commandes lors du logout et les restaure au login
set prompt = "[\!]%"	affiche le numéro de la commande dans le prompt

REEXECUTER LES COMMANDES

!!	commande précédente

!n	nième commande
!-n	emplacement relatif (n commandes ont été exécutées depuis)
!car	la dernière commande commençant par car

REUTILISER LES ARGUMENTS

!n:i	le ième argument de la commande n (la commande est numérotée 0)
!n:$	le dernier argument de la commande n
!n:^	le deuxième argument de la commande n (le premier paramètre de la commande)
!n:i-$	référence les arguments du ième au dernier de la commande n
!n:*	tous les arguments de la commande n

MODIFIER LES COMMANDES PRECEDENTES

commande	paramètres à modifier : modification
s/old/new	remplace la première occurrence de old par new
g	substitution globale (à combiner avec une autre commande)
h	garde uniquement la partie répertoire du nom de fichier
p	imprime la commande sans l'exécuter
q	empêche une modification future
r	enlève l'extension du nom du fichier
t	garde le dernier élément du nom du fichier
&	recommence la dernière substitution

LES ALIAS

Pour personnaliser les commandes

alias	pour obtenir la liste des alias
alias ch cmd d'history	positionne l'alias ex alias rm rm -i alias cd 'cd \!* ; ls' !* correspond à une commande
unalias ch	détruit l'alias

L'EXECUTION D'UN SCRIPT EN C SHELL

csh nom_fichier ou rendre le fichier exécutable (chmod u+x nom_fichier) puis taper le nom du fichier
Pour forcer l'exécution du fichier en C Shell, le fichier doit commencer par *#!/bin/csh*

csh -n nom_fichier	interprète les commandes sans les exécuter
csh -v nom_fichier	imprime les lignes comme elles sont lues
csh -x nom_fichier	imprime les lignes comme elles sont interprétées

LES VARIABLES

set variable valeur	affectation
$variable	valeur de la variable
${variable**}**	valeur de la variable
$?variable	renvoie 0 si la variable n'est pas initialisée, 1 sinon

RECUPERATION DES PARAMETRES DANS LA LIGNE DE COMMANDE

$#	indique le nombre de composants (séparés par des blancs) dont est composée la variable
$n	équivaut à $argv[n] mais ne sortira jamais d'erreur out of range
$*	liste des paramètres (équivaut à $argv)

QUELQUES VARIABLES SPECIALES

$$	le numéro de processus de la dernière commande
$<	remplacé par la prochaine entrée courante (pour les scripts interactifs)

var=$< équivaut à *read var* en Bourne Shell

LES CARACTERES SPECIAUX

****	banalise le caractère suivant
"..."	banalise les caractères sauf \ , $ et `
'...'	banalise tous les caractères
`...`	substitution de commande
% num	préfixe pour les numéros de job (le numéro de job est indiqué lors du lancement d'une commande en background) Pour visualiser la liste des jobs : **jobs**

LES INSTRUCTIONS

IF

SYNTAXE

```
if  (exp) then
        liste_commandes
        [else if  ( exp ) then
                liste_commandes] ...
        [else
                liste_commandes]
endif


if  (exp) commande


if  (exp)  \
        commande
```

WHILE

SYNTAXE

```
while (exp)
        liste_commandes
end
```

FOREACH

SYNTAXE

```
foreach index_var (liste)
        liste_commandes
end
```

break : arrête la boucle
continue: arrête l'itération courante de la boucle et commence la prochaine itération

EXEMPLE

```
foreach i ($argv)
        if ($i != *.c) then
                echo " $i n'est pas un programme C"
                continue
        else
                echo " $i est un programme C"
        endif
        cc $i
end
```

SWITCH

SYNTAXE

```
switch ( para )
        case choix1 :
                liste_commandes
                breaksw
        default :
                liste_commandes
                breaksw
endsw
```

GOTO

SYNTAXE

```
boucle :

        liste_commandes

        goto boucle
```

LES TESTS

-e fichier	vrai si le fichier existe
-r fichier	vrai si le fichier existe et est accessible en lecture (R)
-w fichier	vrai si le fichier existe et est accessible en écriture (W)
-x fichier	vrai si le fichier existe et est exécutable (X)
-f fichier	vrai si le fichier existe et est un fichier régulier
-d fichier	vrai si le fichier existe et est un répertoire
-z fichier	vrai si le fichier existe et a une taille non nulle
-o fichier	vrai si le fichier existe et nous appartient

DIVERS

#	commentaire
(cmde**)**	exécute la commande dans un sous-shell
{cmde**}**	permet de savoir si une commande s'est bien passée (1) ou non (0)
cmd1 **&&** cmd2	séparateur conditionnel (cmd2 sera exécuté si cmd1 s'est exécuté correctement)
cmd1 **||** cmd2	séparateur conditionnel (cmd2 sera exécuté si cmd1 ne s'est pas exécuté correctement)
source script	exécution du script dans le shell courant
rehash	reconstruit la hash table de localisation des commandes
repeat n cmd	répète la commande n fois
setenv/unsetenv	positionne une variable d'environnement
time cmde	détermine le temps d'exécution de la commande

LES OPERATEURS ARITHMETIQUES

@var = exp	assigne une valeur à une variable numérique
() , + , - , * , /	opérations arithmétiques classiques
%	reste de la division
^	ou exclusif bit à bit
~	complément unaire

LES OPERATEURS BOOLEENS

==	comparaison
!=	diffèrent de
!	négation
> ,< ,>= ,<=	comparaison
>>	décalage à droite
<<	décalage à gauche
&	Et bit à bit
|	Ou bit à bit
&&	Et logique
||	Ou logique

LES OPERATEURS D'AFFECTATION

=	affectation
+=	x+=y équivaut à x = x + y
-=	x-=y équivaut à x = x - y
***=**	x*=y équivaut à x = x * y
/=	x/=y équivaut à x = x / y
%=	x%=y équivaut à x = x % y
^=	x^=y équivaut à x = x ^ y
++	x++ équivaut à x = x + 1
--	x-- équivaut à x = x - 1

Attention, les opération suivantes ne marchent pas : &=, |=,<<=,>>=

LES SHELLS UNIX : LE KORN SHELL

PROCESSUS SEQUENTIELS

```
proc1

proc2

proc3

proc1 ; proc2 ; proc3
```

PROCESSUS EN PARALLELES

```
proc1 & proc2 & proc3 &
```

REDIRECTION DES ENTREES-SORTIES

<	l'entrée standard est lue à partir d'un fichier
>	La sortie standard est redirigée dans un fichier (RAZ du fichier)
>>	La sortie standard est redirigée dans un fichier (concaténation du fichier)
2>	les erreurs sont redirigées dans un fichier
2>&1	les erreurs sont redirigées dans le même fichier que la sortie standard

HERE DOCUMENT

Création de fichier dans un script

```
cmde<<[-] Délimiteur
ligne1
ligne2
ligne3
...
Délimiteur
```

Si **-** est ajouté, les tabulations de début de ligne sont supprimées du document.
exemple

```
cat > fichier << EOF
abc
def
EOF
```

LES PIPES

```
proc1  |  proc2
```

Équivaut à :

```
proc1 > fich
proc2 < fich
```

GENERATION DES NOMS DE FICHIERS

*****	n'importe quelle chaîne de caractères
?	n'importe quel caractère
[...]	n'importe quel caractère décrit entre les crochets

LE LOGIN

Exécution du fichier .login pour initialiser l'environnement
Exécution du fichier .profile pour initialiser l'environnement
Si on positionne la variable ENV, alors le fichier spécifié (en général $HOME/.kshrc) est exécuté à chaque ouverture de session shell (pour les alias)

LES PARAMETRES DU SHELL

HOME	le home directory (répertoire de login)
PATH	chemin de recherche pou l'exécution des commandes
CDPATH	chemin de recherche pour la commande cd
MAIL	chemin indiquant le répertoire du courrier
PS1	primary system prompt
PS2	secondary system prompt
IFS	internal field separator
SHELL	indique le shell de login

L'EXECUTION D'UN SCRIPT EN KORN SHELL

ksh nom_fichier ou rendre le fichier exécutable (chmod u+x nom_fichier) puis taper le nom du fichier
Pour forcer l'exécution du fichier en Korn Shell, le fichier doit commencer par *#!/bin/ksh*

ksh -n nom_fichier	interprète les commandes sans les exécuter
ksh -v nom_fichier	imprime les lignes comme elles sont lues
ksh -x nom_fichier	imprime les lignes comme elles sont interprétées

LES VARIABLES

variable **=** valeur affectation (Attention, ne pas mettre d'espace autour de =)

$variable valeur de la variable

**${**variable**}** valeur de la variable (permet d'éviter certaines ambigüités: si a="var",${a}b renvoie varb alors que $ab est invalide)

RECUPERATION DES PARAMETRES DANS LA LIGNE DE COMMANDE

$0 nom de la commande

$ n nième paramètre

$# nombre de paramètres

$* $@ liste de tous les paramètres La signification de $* et de $@ est identique. Cependant, lorsque "$*" est utilisé comme argument de commande, il est équivalent à "$1 $2 " alors que "$@" est équivalent à "$1" "$2"

Pour décaler les paramètres, on peut utiliser la commande *shift*

QUELQUES VARIABLES SPECIALES

$$ le numéro de processus de la dernière commande

$? Status de la dernière commande

LES CARACTERES SPECIAUX

**** banalise le caractère suivant

" ... " banalise les caractères sauf \ , $ et `

' ... ' banalise tous les caractères

` ... ` substitution de commande

LES INSTRUCTIONS

Les commandes *if, while* , *until* teste le status de la commande. (Voir le man pour déterminer le status renvoyé par une commande particulière, en général si la commande s'exécute correctement, le status est vrai)
Attention mettre une négation devant la commande (if ! cmde) ne permet pas de tester l'échec de la commande car la négation s'applique au résultat de la commande et non pas à son status.
Dans cette optique, le test est considéré comme une commande, on peut d'ailleurs l'utiliser sur une ligne de commande.

IF

SYNTAXE

```
if cmde
        then
```

```
        liste_commandes
        [elif liste_commandes
                then
                        liste_commandes] ...
        [else liste_commandes]
fi
```

EXEMPLES

```
if test -f $1
then
        file $1
else
        echo " le fichier n'existe pas "
fi


if grep jean personnel
then
        echo jean >> disponible
elif grep pierre personnel
        then
                echo pierre >> disponible
        else
                echo vide >> disponible
fi
```

WHILE

SYNTAXE

```
while commande
do
        liste_commandes
done
```

EXEMPLE

```
while [ -r " $1 " ]
do
        cat $1 >> concat
        shift
done
```

UNTIL

SYNTAXE

```
until commande
do
```

```
        liste_commandes
done
```

EXEMPLE

```
until [ ! -r " $1 " ]
do
        cat $1 >> concat
        shift
done
```

CASE

SYNTAXE

```
case para in
        choix1[|choix2] ... ) liste_commandes ;;
esac
```

EXEMPLE

```
case $1 in
        -d | -r ) rmdir $dir
                  echo "option -d ou -r ";;
        -o )      echo "option -o ";;
        * )       echo "réponse incorrecte ";;
esac
```

FOR

SYNTAXE

```
for para [in liste]
do
        liste_commandes
done
```

La variable *para* prend successivement les valeurs de la liste
si la liste est omise, *para* prend alors les valeurs passées en paramètres du script

EXEMPLES

```
for i in `ls`
do
        cp $i /dir/$i
        echo "$i copie "
done
```

N'oublier pas les ` qui force l'exécution du ls.

```
for dir in /dev /usr /users/bin /lib
do
        num=`ls $dir|wc -w`
        echo "$num fichiers dans $dir "
done


for i
do
        echo $i
done
```

LES TESTS

N'oubliez pas que le test est une commande qui peut être exécutée directement sur la ligne de commande
test expr ou [expr] ou [[exp]] (Attention il faut un espace après [et avant])
ou expr vaut :

-r fichier	vrai si le fichier existe et est accessible en lecture (R)
-w fichier	vrai si le fichier existe et est accessible en écriture (W)
-x fichier	vrai si le fichier existe et est exécutable (X)
-f fichier	vrai si le fichier existe et est un fichier régulier
-d fichier	vrai si le fichier existe et est un répertoire
-s fichier	vrai si le fichier existe et a une taille non nulle
-L fichier	vrai si le fichier existe et est un lien symbolique
s1 **=** s2	vrai si les deux expressions sont égales
s1 **!=** s2	vrai si les deux expressions sont différentes
s1	vrai si s1 n'est pas la chaîne nulle
e1 **-eq** e2	vrai si les deux entiers e1 et e2 sont algébriquement égaux
(autres comparaisons : -ne, -gt, -ge, -lt, -le)	
!	négation unaire
-a	opération binaire ET
-o	opération binaire OU
(Expression**)**	Vrai si Expression est vraie. Permet de regrouper des expressions.
Expression1 **&&** Expression2	Vrai si Expression1 et Expression2 sont vraies.
Expression1 **||** Expression2	Vrai si Expression1 ou Expression2 est vraie.

DIVERS

#	commentaires, mais #!/bin/sh en début de fichier force l'exécution en Bourne Shell
(cmde **)**	exécute la commande dans un sous-shell
read a	lecture d'une entrée pendant l'exécution d'un script
exit num	renvoie le status de la commande (en général 0 la commande s'est bien exécutée)
return num	code d'erreur
. script	exécution du script dans le shell courant
eval arg	interprète arg avant de l'exécuter

cmd1 **&&** cmd2 séparateur conditionnel (cmd2 sera exécuté si cmd1 s'est exécuté correctement)

cmd1 **||** cmd2 séparateur conditionnel (cmd2 sera exécuté si cmd1 ne s'est pas exécuté correctement)

nom_fonction ()
{ liste_commandes **;}** définition d'une fonction

exec arg exécute la commande dans un nouveau shell

set var initialisation d'une variable
liste de tous les paramètres du système
positionne les paramètres $i (set a b c positionne $1 à a, $2 à b et $3 à c)

unset var raz d'une variable

type cmde indique la localisation d'une commande

readonly var empêche la modification d'une variable

LES EXPRESSIONS

let exp ou ((exp))
let x=x+2
((x > 10))
Attention le test de l'égalité est ==

LES FONCTIONS

function Identificateur {Liste ;}
La liste est composée de deux partie : la déclaration de variables et les commandes
On peut faire des fonctions récursives

DECLARATION DES VARIABLES

typeset var déclaration d'une chaîne de caractères

integer var
typeset -i var déclaration d'un entier

typeset -r var=valeur
readonly var=valeur définition d'une constante

On peut utiliser des tableaux qui ne sont déclarés que lors de leur assignation: tab[100]=toto

LES LIBRAIRIES

On créé un répertoire dans lequel on stocke les fichiers contenant les fonctions (le nom du fichier doit être le même que le nom de la fonction)
Pour utiliser cette librairie, positionner la variable FPATH, puis importer les fonctions (par autoload ou typeset -fu)

```
FPATH=$HOME/lib/rep1:$HOME/lib/rep2
autoload init
tyteset -fu ecr
```

On peut définir des fonctions dans le fichier définie par ENV mais elles ne seront visibles que du shell interactif, pour les rendre toujours visible il faut les exporter par **typeset -fx** fonc

RECUPERATION DES RESULTATS DE LA FONCTION

On positionne le status de la fonction par return (récupéré par $?)
On renvoie une valeur par l'intermédiaire de la commande echo (récupéré par : a=`fonction` ou a=$(fonction)

LES ALIAS

alias donne la liste des alias

alias nom=valeur initialisation d'un alias

Les alias suivants sont définis par défaut par le shell

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

GETOPTS

Cette commande permet de récupérer facilement les options passées en paramètre du script.

SYNTAXE GENERALE

getopts Chaîne_options Nom [Argument ...]
Nom prend successivement comme valeur celles passées en paramètre du script.
Les seules valeurs valides sont données par chaine_option (voir ci dessous pour plus de détails)
Par exemple

```
while getopts vy argument
do
        case $argument in
                v) ...;;
                y) ... ;;
        esac
done
```

TRAITEMENT DES ERREURS

Pour rajouter un traitement d'erreur automatique, il suffit de rajouter **:** devant la liste des options valides
Lorsque l'utilisateur utilise une option non valide, **getopts** renvoie **\?**

```
while getopts :vy argument
do
        case $argument in
                v) ...;;
                y) ... ;;
                \?) ... ; exit ;;
        esac
done
```

OPTION AVEC ARGUMENTS (ET GESTION DES ERREURS)

Pour permettre de saisir un paramètre apres une option, il suffit de rajouter **:** après l'option concernée
La variable **OPTARG** contient le paramètre Si l'utilisateur omet le paramètre, **getopts** renvoie **:** et **OPTARG** l'option concernée.

```
while getopts :y: argument
do
        case $argument in
                y) ... ; arg=$OPTARG;;
                :) echo " l'option -$OPTARG a besoin d'un argument " ; exit
;;
                \?) ... ; exit ;;
        esac
done
```

LA GESTION DES MENUS

```
select Identificateur [in liste_choix]
do
        commande
        ...
done
```

La commande **select** écrit sur la sortie d'erreur standard la liste des choix, chacun étant précédé d'un numéro.
Si **in liste_choix** n'est pas spécifié, les paramètres positionnels sont utilisés.
Le contenu de la variable **PS3** s'affiche et l'entrée standard est lu. Si le numéro d'un des mots listés est saisie, le paramètre **Identificateur** prend la valeur de ce mot.
Si la ligne est vide, la liste s'affiche de nouveau. Sinon, la valeur du paramètre Identificateur est "". Le contenu de la ligne lue à partir de l'entrée standard est sauvegardé dans le Paramètre **REPLY**. La liste est exécutée pour chaque sélection jusqu'à un caractère d'interruption ou de fin de fichier.

```
PS3=" votre choix "
select menu_list in "choix 1"  "choix 2"  "fin"
do
        case $menu_list in
                "choix 1").... ;;
                "choix 2")... ;;
                "fin") exit 0 ;;
                "") echo "$REPLY est une option invalide "
```

```
        esac
done
```

POSITIONNEMENT DES OPTIONS

set : Les options de cette commande sont :

-n Lit les commandes en recherchant les erreurs de syntaxe, sans les exécuter. Cette option est ignorée par les shells interactifs.

-t Génère une sortie après la lecture et l'exécution d'une commande.

-v Ecrit les lignes d'entrée du shell à mesure qu'elles sont lues.

-x Ecrit les commandes et leurs arguments à mesure qu'ils sont exécutés.

- Désactive les options -x et -v et interrompt la vérification des arguments.

Remplacer le signe - par + désactive l'option. Celles-ci peuvent aussi être utilisées sur appel du shell.
Les options en cours peuvent être visualisées dans $-. Sauf si -A est spécifiée, les arguments restants sont des paramètres positionnels et sont affectés, dans l'ordre, à $1, $2, etc.
Si aucun argument n'est spécifié, les noms et les valeurs de tous les paramètres sont écrits sur la sortie standard. Si + est le seul argument spécifié, les noms de tous les paramètres sont écrits.

LES EXPRESSIONS REGULIERES

Les expressions régulières caractérisent uniquement des chaines de caractères et pas des noms de fichiers.
Elles sont utilisées notamment avec les commandes *ed*, *vi*,*ex*, *sed*, *awk*.
Pour les exemples, j'utiliserai la commande de substitution de vi ou sed :
s/RE/chaine de remplacement/ (ou RE caractérise une expression régulière)
Lorsqu'on recherche une chaine de caractères à l'aide d'une expression régulière, la chaine renvoyée est la chaine la plus grande correspondant avec l'expression.

Pour banaliser un caractère, il faut utiliser \ avant ce caractère.

.

Caractérise n'importe quel caractère.
Rappel : pour pouvoir caractériser le **.**, il faut le banaliser : **\.**

chaine de départ	asdeuy...dur	
commande	s/.../---/	s/\.\.\./---/
résultat	---euy...dur	asdeuy---dur

[ ]

Un des caractères entre les crochets ou
Si le premier caractère entre crochet est **^**, alors cela caractérise un caractère qui ne correspond pas avec les caractères entre crochets.

[abc]	a, b ou c
[a-z]	une lettre minuscule
[0-57]	0, 1, 2, 3, 4, 5 ou 7
[a-d5-8X-Z]	a, b, c, d, 5, 6, 7, 8, X, Y ou Z
[0-5-]	0, 1, 2, 3, 4, 5 ou -
[^0-9]	pas un chiffre
[^a-zA-Z]	pas une lettre
[012^]	0, 1, 2 ou ^

^, $

^ : caractérise le début de ligne (Attention ne pas confondre ^ et [^]).
$: Caractérise la fin de ligne.

/^abc/	ligne commençant par abc
/abc$/	ligne finissant par abc
/^$/	ligne vide

*

Attention, l'utilisation de * est un peu particulière dans les expressions régulières :
Elle signifie de 0 à n fois le caractère la précédant

a*	0 ou n fois *a*
aa*	au moins un *a*
.*	n'importe quelle chaine de caractères

(y compris la chaine vide)

chaine de départ	aabbabbaab		
commande	s/[ab]*/x/	s/a.*b/y/	s/a.*bb/z/
résultat	x	y	zaab

/^[0-9][0-9]*$/ ligne qui ne contient que des chiffres et non vide

\(\)

Pour isoler des sous chaines. On peut réutiliser les sous chaines ainsi trouvées grâce à \1, \2 ..

chaine de départ	ejkf fed 158e fd
commande	s/.*\([0-9][0-9]*\).*/ resultat = \1/
résultat	resultat = 158

LA COMMANDE SED

sed est un éditeur non interactif.
Cette commande permet d'appliquer un certain nombre de commandes sur un fichier puis d'en afficher le résultat (sans modification du fichier de départ) sur la sortie standard.

SYNTAXE

sed [-n] [-e commande] [-f fichier de commandes] [fichier]

-n écrit seulement les lignes spécifiées (par l'option /p) sur la sortie standard

-e cmde permet de spécifier les commandes à appliquer sur le fichier. Cette option est utile lorsque vous appliquez plusieurs commandes. Afin d'éviter que le shell interprète certains caractères, il faut mieux encadrer la commande avec des ' ou des " .

-f fich les commandes sont lu à partir d'un fichier.

PRINCIPE DE FONCTIONNEMENT

Pour chaque ligne, on applique la commande (si cela est possible) puis on affiche sur la sortie standard la ligne modifiée ou non.

La syntaxe générale des commandes est de la forme *caracterisation_des_adresses commandes* avec caractérisation_des_adresses de la forme :

	toutes les lignes
num	la ligne *num* (la dernière ligne est référencée par *$)*
num1,num2	les lignes entre les lignes *num1* et *num2*
RE	les lignes correspondant à l'expression régulière RE
RE1,RE2	les lignes entre la première ligne correspondant à l'expression régulière*RE1* et la première ligne correspondant à l'expression régulière *RE2*

LES COMMANDES SED

LA COMMANDE DE SUBSTITUTION S

syntaxe : ad1,ad2s/RE/remplacement/flags
Remplace les expressions régulières RE par la chaine de remplacement entre les lignes ad1 à ad2 en tenant compte du flag :

g global, c'est à dire toutes les occurrences de la chaine RE (par défaut seule la première occurrence est remplacée)
p imprime la ligne si la substitution a réussie
w fichier écrit la ligne dans le fichier spécifié en plus de la sortie standard si la substitution a réussie.(voir exemple ci-dessous)

exemple :
`sed "s/[Cc]omputer/COMPUTER/g" fichier`
`sed -e "s/\([0-9][0-9]*\)/**\1**/" fichier` : encadre le premier nombre de la ligne avec des **

LA NEGATION !

syntaxe : ad1,ad2 !fonction argument
La fonction est appliquée à toutes les lignes qui ne correspondent pas à la caractérisation .

LA COMMANDE DE SUPRESSION D

efface les lignes (au niveau de la sortie, le fichier d'origine n'est pas modifié)
exemple:
`sed "1,10d" fichier :` sortie du fichier à partir de la onzième ligne
`sed "/^From/!d" fichier :` On n'efface tout sauf les lignes commençant par *From* , donc on imprime les lignes commençant par *From*.
Remarque : Il est parfois plus facile de caractériser la négation de ce que l'on veut (voir exemple précédent) . Par exemple plutôt de récupérer ce que je veux, j'efface ce qui ne m'intéresse pas

LES COMMANDE D'INSERTIONS A,I

SYNTAXE

```
a\
texte
```
: écrit le texte après la ligne

```
i\
texte
```
: écrit le texte avant la ligne

EXEMPLE :

fichier de commandes :

```
1i\
\
------------------------------------------------------------------------
---------\
LOGIN USER\
------------------------------------------------------------------------
---------
s/:/!/
s/:/-/
s/:/-/
s/:/!/
s/!.*!/ /
s/:.*//
```

```
sed -f fich_commandes /etc/passwd
```

resultat :

```
------------------------------------------------------------------------------
---------
LOGIN                           USER
------------------------------------------------------------------------------
---------
root                             Operator
jd                                Jean Dupond
vm                               Vincent Martin
```

LES AUTRES COMMANDES: Q, = ET W

SYNTAXE

Q	quitte
=	écrit les numéros de ligne
w fichier	écrit dans un fichier

EXEMPLE :

fichier d'entrée :

```
Un,
deux.
Trois,
quatre.
```

```
sed -e "q" fichier
```

Résultat : Un,

```
sed -e "/\./=/" -e "/[A-Z]/w capitale" fichier
```

résultat à l'écran :

```
Un,
2
deux.
Trois,
4
quatre.
```

Résultat dans le fichier capitale :

```
Un,
Trois,
```

LA COMMANDE AWK

Cette commande permet d'appliquer un certain nombre d'actions sur un fichier. La syntaxe est inspirée du C

SYNTAXE

awk [-Fs] [-v variable] [-f fichier de commandes] 'program' fichier

-F	Spécifie les séparateurs de champs
-v	Définie une variable utilisée à l'intérieur du programme.
-f	Les commandes sont lu à partir d'un fichier.

PRINCIPE DE FONCTIONNEMENT

Le programme awk est une suite d'action de la forme : motif { action } , le motif permet de déterminer sur quels enregistrements est appliquée l'action.

Un enregistrement est : Une chaine de caractères séparée par un retour chariot, en général une ligne.
Un champ est : Une chaine de caractères séparée par un espace (ou par le caractère spécifié par l'option -F), en générale un mot.

On accède à chaque champs de l'enregistrement courant par la variable *$1*, *$2*, ... *$NF*. *$0* correspond à l'enregistrement complet. La variable *NF* contient le nombre de champs de l'enregistrement courant, la variable $NF correspond donc au dernier champ.

EXEMPLES

`awk -F ":" '{ $2 = "" ; print $0 }' /etc/passwd`	imprime chaque ligne du fichier /etc/passwd après avoir effacé le deuxième champ
`awk 'END {print NR}' fichier`	imprime le nombre total de lignes du fichier
`awk '{print $NF}' fichier`	imprime le dernier champ de chaque ligne
`who | awk '{print $1,$5}'`	imprime le login et le temps de connexion.
`awk 'length($0)>75 {print}' fichier`	imprime les lignes de plus de 75 caractères. (print équivaut à print $0)

LES VARIABLES PREDEFINIES

ARGC	Nombre d'arguments de la ligne de commande
ARGV	tableau des arguments de la ligne de commande
FILENAME	nom du fichier sur lequel on applique les commandes

FNR	Nombre d'enregistrements du fichier	
FS	séparateur de champs en entrée	(valeur par défaut « . »)
NF	nombre de champs de l'enregistrement courant	
NR	nombre d'enregistrements deja lu	
OFMT	format de sortie des nombres	(valeur par défaut . « %.6g »)
OFS	séparateur de champs pour la sortie	(valeur par défaut . « . »)
ORS	séparateur d'enregistrement pour la sortie	(valeur par défaut . « \n»)
RLENGTH	longueur de la chaine trouvée	
RS	séparateur d'enregistrement en entrée	(valeur par défaut . « \n»)
RSTART	début de la chaine trouvée	
SUBSEP	séparateur de subscript	(valeur par défaut . « \034»)

SYNTAXE DU MOTIF

Si le motif existe dans l'enregistrement, l'action sera appliquée à la ligne.
Le motif peut être :

un expression régulière /expression régulière/
$0 ~ /expression régulière/
expression ~ /expression régulière/
expression !~ /expression régulière/

une expression BEGIN ou END

une expression de comparaison: <, <=, == , !=, >=, >

une combinaison des trois (à l'aide des opérateurs booléens || *ou*, && *et*, ! *négation*)

une caractérisation des lignes motif1,motif2 : chaque ligne entre la première ligne correspondant au motif1 et la première ligne correspondant au motif2

EXAMPLES

```
 awk 'BEGIN { print "Verification des UID et GID dans le fichier
/etc/passwd";
              FS=":"}
      $3 !~ /^[0-9][0-9]*$/ {print "UID  erreur ligne "NR" :\n"$0 }
      $4 !~ /^[0-9][0-9]*$/ {print "GID  erreur ligne "NR" :\n"$0 }
      END   { print "Fin" }
' /etc/passwd
```

Résultat :
Vérification des UID et GID dans le fichier /etc/passwd
UID erreur ligne 14 :
clown:*:aaa:b:utilisateur en erreur:/home/clown:/bin:sh
GID erreur ligne 14 :
clown:*:aaa:b:utilisateur en erreur:/home/clown:/bin/sh
Fin

```
 awk 'BEGIN { print "Verification du fichier /etc/passwd pour ...";
              print "- les utilisateurs avec UID = 0 " ;
              print "- les utilisateurs avec UID >= 60000" ;
```

```
                FS=":"}
      $3 == 0 { print "UID 0 ligne "NR" :\n"$0 }
      $3 >= 60000  { print "UID >= 60000 ligne "NR" :\n"$0 }
      END   { print "Fin" }
' /etc/passwd
```

Résultat :
Vérification du fichier /etc/passwd pour ...
- les utilisateurs avec UID = 0
- les utilisateurs avec UID >= 60000
UID 0 ligne 5 :
root:*:0:b:administrateur:/:/bin/sh
UID >= 60000 ligne 14 :
clown:*:61000:b:utilisateur en erreur:/home/clown:/bin/sh
Fin

```
 awk 'BEGIN { print "Verification du fichier /etc/group";
              print "le groupe 20 s'appelle t-il bien users ? " ;
              FS=":"}
      $1 == "users" && $3 ==20 { print "groupe "$1" a le GID "$3" !" }
      END   { print "Fin" }
' /etc/group
```

Résultat :
Vérification du fichier /etc/group
le groupe 20 s'appelle t-il bien users ?
groupe users a le GID 20 !
Fin

```
 awk 'NR == 5 , NR == 10 {print NR" : " $0 }' fichier
```

Imprime de la ligne 5 à la ligne 10 , chaque ligne précédée par son numéro

SYNTAXE DE L'ACTION

Une action transforme ou manipule des données. Par défaut *print*
type des actions

- fonctions prédéfinies, numérique ou chaine de caractères
- contrôle de flots
- affectation
- impression

FONCTIONS NUMERIQUES

atan2(y,x)	arc tangente de x/y en radians dans l'intervalle -pi pi
cos(x)	cosinus (en radians)
exp(x)	exponentielle e à la puissance x
int(x)	valeur entière
log(x)	logarithme naturel

rand()	nombre aléatoire entre 0 et 1
sin(x)	sinus (en radians)
sqrt(x)	racine carrée
srand(x)	réinitialiser le générateur de nombre aléatoire

LES FONCTIONS SUR LES CHAINES DE CARACTERES

Dans le tableau suivant :
s et t représente des chaines de caractères
r une expression régulière
i et n des entiers

gsub(r,s,t)	sur la chaine t, remplace toutes les occurrences de r par s
index(s,t)	retourne la position la plus à gauche de la chaine t dans la chaine s
length(s)	retourne la longueur de la chaine s
match(s,r)	retourne l'index ou s correspond à r et positionne RSTART et RLENTH
split(s,a,fs)	split s dans le tableau a sur fs, retourne le nombre de champs
sprintf(fmt,liste expressions)	retourne la liste des expressions formatée suivant fmt
sub(r,s,t)	comme gsub, mais remplace uniquement la première occurrence
substr(s,i,n)	retourne la sous chaine de s commençant en i et de taille n

LES VARIABLES ET EXPRESSIONS

LES OPERATIONS ET AFFECTATIONS ARITHMETIQUES

Les opérateurs arithmétiques sont les opérations usuelles : + - * / % (reste division entière) et ^ (puissance).
Tous les calculs sont effectués en virgule flottante.

La syntaxe de l'affectation : var = expression
Vous pouvez aussi utiliser les operateurs +=, -=, *=, /=, %= et ^= (x+=y équivaut à x=x+y)

LES VARIABLES DE CHAMPS

Rappel : Les champs de la ligne courant sont : $1, $2, ..., $NF
La ligne entière est $0
Ces variables ont les mêmes propriétés que Les autres variables. Elles peuvent être réaffectées. Quand $0 est modifiées, les variables $1,$2 ... sont aussi modifiées ainsi que NF.Inversement si une des variables $i est modifiées, $0 est mise à jour.
Les champs peuvent être spécifiés par des expressions, comme $(NF-1) pour l'avant dernier champs.

EXAMPLE

```
awk 'BEGIN { FS=":" ;
             OFS=":"}
     $NF != "/bin/ksh" { print $0 }
     $3 == "/bin/ksh" && NF == 7  { $7 = "/bin/posix/sh" ;
                                    print $0 } '
     /etc/passwd > /etc/passwd.new
```

Résultat :
On crée un nouveau fichier de mot de passe /etc/passwd.new en remplaçant le shell /bin/ksh par /bin/posix/sh

CONCATENATION DE CHAINES DE CARACTERES

Il n'y a pas d'opérateur de concaténation, il faut simplement lister les chaines à concaténer.

EXAMPLES:

```
awk '{ print NR " : " $0 }' fichier
```

Résultat :
On numérote les lignes du fichier

```
awk 'BEGIN { FS=":" ;
             OFS=":" ;
             print " Run Level 2 : Liste des actions "}
      $2 ~ /2/  { print "Keyword <<"$3">>, \n Tache <<"$4">>" }
      $2 == ""  { print "Keyword <<"$3">>, \n Tache <<"$4">>" }
' /etc/inittab > /etc/passwd.new
```

Résultat :
Affiche les actions exécutées lors du passage à l'état 2

WHILE

SYNTAXE

while (condition) action

FOR

SYNTAXE

for (expression ; expression ; expression) action

for (var in array) action (voir ci-dessous utilisation des tableaux)

BREAK, CONTINUE

break sortie de boucle
continue commence une nouvelle itération de la boucle.

IF , ELSE

SYNTAXE

if (expression) action
else action

COMMENTAIRE ET ACTION VIDE

Le commentaire est précédé par **#**. Tout ce qui est entre # et la fin de la ligne est ignoré par awk
Une action vide est représenté par ;

NEXT, EXIT

Next passe à l'enregistrement suivant. On reprend le script awk à son début
Exit ignore le reste de l'entrée et exécute les actions définie par END

AFFICHAGE

print exp, exp ou print (exp , exp) affiche les expressions
print equivaut à *print $0*
printf format , exp, exp ou printf (format,exp , exp) identique à print mais en utilisant un format (voir printf en C)
Remarque: La sortie d'un print ou d'un printf peut être redirigée dans un fichier ou sur un pipe
Les noms de fichiers doivent être entre guillemets sinon ils sont considérés comme des variables

EXEMPLE:

awk ' { print NR " :" , $0 > "fich.numerote" } ' fichier	le fichier *fich.numerote* contient le fichier *fichier* avec les lignes numérotées
awk ' { printf "%3d : %s " , NR , $0 > "fich.numerote" } ' fichier	le fichier *fich.numerote* contient le fichier *fichier* avec les lignes numérotées sur 3 caractères

TABLEAU

On peut utiliser des tableaux de chaines de caractères et de nombres à une dimension
Il n'est pas nécessaire de les déclarer. La valeur par défaut est "" ou 0.
Les indices sont des chaines de caractères.

```
awk 'BEGIN { print "Mémorisation de votre fichier " FILENAME }
                {memfile [NR] = $0 }
          END   { for ( i = NR ; i >= 1 ; i-- ) {
                    print i ":" memfile[i]
                    }
                  print "Fin"
                } ' fichier
```

Résultat :
Affiche le fichier en commençant par la dernière ligne

```
awk ' NF > 0 {
                    for (i=1;i<=NF;i++) {
                        if ( $i ~ /^[0-9a-zA-Z][0-9a-zA-Z]*$/ ) {
                            index[$i] = index[$i] ";" NR "," i " " ;
                            index["total"]++ ;
```

```
                        }
                    }
                 }
        END     { x="total" ;
                  printf("%s mots detectés = %d\n",x,index[x]);
                  } ' fichier
```

Résultat :
Construction d un index de cross référence

FOR ET LES TABLEAUX

Comme les indices des tableaux sont des chaines de caractères, on ne peut pas déterminer la taille d'un tableau
On doit donc utiliser la construction :

```
 for (var in tableau)
action
 awk ' NF > 0 {
                    for (i=1;i<=NF;i++) {
                        if ( $i ~ /^[0-9a-zA-Z][0-9a-zA-Z]*$/ ) {
                            index[$i] = index[$i] ";" NR "," i " " ;
                            index["total"]++ ;
                            }
                        }
                     }
        END     {
                for ( x in index ) {
                 if ( x != "total" )
                   printf("%-20s\t%s\n",x,index[x]) | "sort -f "
                  }
                  x="total";
                  printf("%s mots detectés = %d\n",x,index[x]);
                  } ' fichier
```

Résultat :
Construction d un index de cross référence

SIMULATIONS DES TABLEAU MULTIDIMENSIONS

On ne peut pas utiliser des tableaux multidimentionnels.
On peut les simuler en concaténant des composants des sous chaines avec le séparateur SUBSEP

```
 awk 'BEGIN { print "Mémorisation de votre fichier " FILENAME
                  SUBSEP=":"
                }
                {  for ( i=1 ; i <=NF ; i++ ) {
                   memfields[ NR , i ] = $i
                   }
                }
           END  { for ( i in memfields ) {
                   print i ":" memfields[i] | "sort -n -t: "
                   }
```

```
                print "Fin"
               } ' fichier
```

Résultat :
Affiche le fichier en commençant par la dernière ligne