

GK – UX001

Unix, Linux : Les Bases indispensables

2775cbe5ce
Global Knowledge

SOMMAIRE

Table des matières

Chapitre 1 Introduction	7
1. Historique.....	8
2. Unix aujourd'hui	9
.a Principales versions commerciales	9
.b Versions libres	11
3. Composantes d'un système Unix	13
.a Le noyau (mono ou multi-processeurs)	13
.b Le shell (langage de commandes)	13
.c Les commandes	13
.d Les protocoles TCP/IP.....	14
.e Les interfaces de programmation.....	14
.f L'offre logicielle - les domaines d'utilisation	15
4. Connexion et environnement de travail	15
.a Diverses possibilités de connexion à un système Unix	15
.b Notions de compte utilisateur	16
.c Le fichier /etc/passwd	17
.d Le fichier /etc/group	18
5. La documentation.....	19
.a Ressources Internet	19
.b La commande man.....	19
6. Premiers éléments de syntaxe	21
7. Premières commandes utiles.....	23
.a Effacement de l'écran	23
.b Calendrier perpétuel	23
.c Qui est connecté ?	24
.d Commandes d'identification	25
.e Afficher du texte.....	26
.f Modifier son mot de passe	27
.g Afficher l'heure ou la date système.....	27
.h Récapitulatif des commandes à approfondir dans la documentation.....	27
8. TPs 1 et 2 – Ouverture de session et premières commandes	29
.a Connexion / Changement de mot de passe	30
.b Commandes de base	30
.c La documentation en ligne : la commande <code>man</code>	30
.d Envoyer et recevoir du courrier : commande <code>mail</code>	31
.e Communiquer avec les autres utilisateurs.....	31
.f Actions au clavier.....	32
Chapitre 2 Les systèmes de fichiers	33
1. Vision globale de l'organisation des disques.....	34
2. Notion de filesystem	35
.a Filesystems de type journalisé.....	36
.b Montage.....	37
.c Filesystems de type CD-Rom	38
.d Pseudo filesystem /proc.....	38

3. Types et désignations des fichiers.....	39
.a Les conventions à connaître	39
.b Comment désigner les fichiers ?	40
4. Parcours et visualisation de l'arborescence Unix.....	42
.a Commandes de base (pwd, cd, ls).....	42
.b Les principaux répertoires de la racine Unix	45
.c Créer ou supprimer des répertoires (mkdir, rmdir).....	47
5. TP 3 – Nommages des fichiers et répertoires	48
6. Commandes essentielles de manipulation des fichiers.....	50
.a Visualiser le contenu de fichiers de texte (cat, pg, more, less)	50
.b La commande pg	50
.c La commande more.....	51
.d La commande less	52
.e Copier des fichiers (cp).....	53
.f Gérer les liens (ln)	55
.g Liens physiques	55
.h Liens symboliques	56
.i La commande ln	56
.j Renommer ou déplacer des fichiers (mv)	58
.k Supprimer des fichiers (rm)	60
.l D'autres commandes utiles	61
.m La commande head	61
.n La commande tail	62
.o La commande wc	62
7. TP 4 – Manipulations de fichiers.....	64
.a vérifiez votre environnement.....	64
.b manipulations de fichiers	64
.c création et manipulation de répertoires	65
.d suppression de répertoires	66
8. Droits d'accès	67
.a Sémantique des permissions de base.....	67
.b Permissions supplémentaires.....	69
.c Choisir les droits par défaut en création de fichiers (umask)	72
.d Modifier les droits des fichiers existants (chmod).....	74
.e ACLs (Access Control List)	77
9. Récapitulatif des commandes à approfondir dans la documentation	78
10. TP 5 – Droits d'accès des fichiers	79
.a Affichez les attributs des fichiers	79
.b Travaillez avec les droits d'accès des fichiers ordinaires	81
.c Travailler avec les droits d'accès aux répertoires.....	81
Chapitre 3 L'éditeur de texte vi.....	83
1. Les conventions à connaître	84
2. Les commandes essentielles	86
a. Déplacements.....	86
b. Insertions, suppressions, modifications.....	87
c. Recherches d'expressions	88
d. Substitutions répétitives.....	88
e. Duplications et déplacements de lignes (copier-coller et couper-coller)	89
f. Autres commandes utiles	89
g. Sorties et sauvegardes	90
h. Édition de plusieurs fichiers	90

i. Paramétrage de l'éditeur	91
3. L'éditeur vim des distributions Linux.....	93
4. TP 6 – L'éditeur vi.....	94
.a Création d'un fichier	94
.b Déplacement du curseur	96
.c Personnalisation	99
.d Edition de l'historique de la ligne des commandes.....	100
Chapitre 4 Processus et Mécanismes.....	103
1. Quelques définitions	104
2. Mécanisme de base fork+exec.....	105
3. Attributs des processus	106
4. La commande ps	107
5. Signaux, interruption des processus	109
a. Description des signaux Unix	109
b. Comment éliminer un processus ?	110
6. Redirections.....	111
a. Redirection de l'entrée standard	114
b. Redirection de la sortie standard	114
c. Redirection de la sortie en mode écrasement	115
d. Redirection de la sortie en mode ajout	115
e. Élimination de la sortie.....	116
f. Création de fichier.....	116
g. Protection contre l'écrasement accidentel.....	117
h. Redirection de l'erreur standard	117
7. Processus séquentiels.....	119
8. Mécanisme du pipeline	120
.a Principes généraux	120
.b Mémoriser les résultats intermédiaires	122
9. Processus en arrière-plan	123
a. Pouvoir se déconnecter (mode détaché).....	124
b. Contrôle des tâches	125
10. Récapitulatif des commandes à approfondir dans la documentation	127
11. TP 7 - Gestion des process et redirections	128
.a Process courant.....	128
.b Contrôle des process lancés en tâche de fond.	129
.c Terminer un process.....	130
.d Les redirections	130
.e Tubes (pipes) et filtres	130
.f Commandes groupées et annulation de fin de ligne	131
Chapitre 5 Utilisation du shell	133
1. Les différents shells.....	134
2. Variables et environnement.....	135
a. Variables	135
b. Environnement.....	136
c. Quelques variables prédéfinies	137
d. Internationalisation.....	140

3. Caractères spéciaux.....	140
a. Rappel des caractères spéciaux déjà évoqués	141
b. Désignations abrégées de noms de fichiers (jokers, caractères génériques).....	141
c. Substitutions de commandes	143
d. Caractères de protections	144
4. TP 8 – Eléments du Shell, variables et jokers	146
.a Variables	146
.b Jokers	147
.c Substitution de commande	148
.d Environnement d'un process, export de variable.....	149
.e Apostrophes, Guillemets et banalisation explicite (antislash).....	150
5. Fonctionnalités interactives	151
a. Alias	151
b. Historique des commandes	152
c. Rappel simple de commandes	153
d. Rappel et édition de commandes avec l'éditeur intégré	154
e. Rappel et édition de commandes avec la commande interne fc.....	155
6. Fichiers de connexion.....	155
a. Prise en compte des modifications.....	157
7. TP 9 – Personnaliser son environnement.....	158
.a Personnalisation de .kshrc ou .bashrc.....	159
.b Gestion des alias	160
Chapitre 6 Sélection de commandes	161
1. Commandes complémentaires sur les fichiers	162
a. La commande file	162
b. La commande nl	163
c. La commande cmp	163
d. Diverses commandes complémentaires	164
2. La commande find	165
3. Sauvegardes	169
a. Les commandes de compression.....	170
b. Caractéristiques communes aux commandes de sauvegarde Unix	171
c. La commande tar.....	171
d. La commande cpio	175
4. Commandes d'impression.....	178
5. Autres commandes utiles.....	179
a. La commande script	179
b. La commande crypt	180
c. La commande du.....	181
d. La commande su	182
e. La commande cut	184
6. Filtres.....	185
a. Tris avec sort	186
b. Transformations de caractères avec tr	188
c. Recherche d'expressions (grep, egrep, fgrep)	190
d. Édition non interactive de fichiers avec sed.....	192
7. Récapitulatif des commandes à approfondir dans la documentation	194
8. TP 10 - Utilitaires	196
.a La commande find	197

.b	La commande <code>grep</code>	198
.c	La commande <code>sort</code>	198
.d	Les commandes <code>head</code> et <code>tail</code>	199
.e	Les commandes <code>diff</code> et <code>cmp</code>	199
.f	La commande <code>tar</code>	199

Chapitre 7 Commandes réseau, Environnements graphiques 201

1.	Interfaces physiques	202
2.	Résolution des noms	204
a.	Fichier <code>/etc/hosts</code>	204
b.	Aspect client DNS	205
3.	Applications standards	206
a.	Terminal virtuel (<code>telnet</code>)	206
b.	Les "remote commands"	208
c.	La commande <code>rlogin</code>	208
d.	La commande <code>rsh</code>	210
e.	La commande <code>rcp</code>	211
f.	Transferts de Fichiers avec <code>ftp</code> (<code>sftp</code>)	211
g.	L'alternative sécurisée <code>ssh</code>	214
4.	Environnements graphiques	214
a.	Protocole X-Window, schéma fonctionnel et terminologie	214
b.	Paramétrage et lancement de clients	216
c.	La variable <code>DISPLAY</code>	217
d.	Lancement de clients distants depuis un émulateur de terminal	218

Chapitre 8 Bases de la programmation Korn shell 219

.1	Procédures et paramètres	220
.2	Instructions de Contrôle	221
a.	Tests	221
b.	Tests simples	221
c.	Test séquentiels	223
d.	La commande <code>test</code>	225
e.	Boucles	227
f.	La boucle <code>for</code>	227
g.	Les boucles <code>while</code> et <code>until</code>	228
h.	Traitements associés aux boucles	229
i.	Lectures au clavier	229
j.	Expressions arithmétiques : <code>let</code> et <code>expr</code>	230
k.	Instructions de branchement	231
l.	Aiguillage	232
.3	Quelques aspects complémentaires	233
a.	Fonctions	233
b.	Gestion des signaux	234
c.	Autres commandes internes	235
.4	TP 11 – Les bases de la programmation shell	236
a.	Ecrire une procédure script shell	236
b.	Utilisation de <code>for</code> , <code>test</code> et <code>if</code>	237
c.	Utilisation de <code>while</code> et de <code>expr</code>	237

ANNEXE..... 239

Correction des Travaux Pratiques 239

.1	Introduction.....	240
.2	TPs 1 et 2 – Ouverture de session et premières commandes	240
.a	Connexion / Changement de mot de passe	240
.b	Commandes de base	240
.c	La documentation en ligne : la commande <code>man</code>	242
.d	Envoyer et recevoir du courrier : commande <code>mail</code>	242
.e	Communiquer avec les autres utilisateurs.....	243
.f	Actions au clavier.....	243
.3	TP 3 – Nommages des fichiers et répertoires	245
.4	TP 4 – Manipulations des fichiers et répertoires.....	247
.a	vérifiez votre environnement.....	247
.b	manipulations de fichiers	247
.c	création et manipulation de répertoires	248
.d	suppression de répertoires	249
.5	TP 5 – Droits d'accès des fichiers	251
.a	Affichez les attributs des fichiers	251
.b	Travaillez avec les droits d'accès des fichiers ordinaires	252
.c	Travailler avec les droits d'accès aux répertoires.....	253
.6	TP 6 – L'éditeur vi.....	256
.a	Création d'un fichier	256
.b	Déplacement du curseur	258
.c	Personnalisation	261
.d	Edition de l'historique de la ligne des commandes.....	262
.7	TP 7 - Gestion des process et redirections	265
.a	Process courant.....	265
.b	Contrôle des process lancés en tâche de fond.	265
.c	Terminer un process.....	267
.d	Les redirections	267
.e	Tubes (pipes) et filtres	268
.f	Commandes groupées et annulation de fin de ligne	270
.8	TP 8 – Eléments du Shell, variables et jokers	272
.a	Variables.....	272
.b	Jokers	274
.c	Substitution de commande	275
.d	Environnement d'un process, export de variable.....	276
.e	Apostrophes, Guillemets et banalisation explicite (antislash).....	278
.9	TP 9 – Personnaliser son environnement.....	280
.a	Personnalisation de <code>.kshrc</code> ou <code>.bashrc</code>	280
.b	Gestion des alias	281
.10	TP 10 – Utilitaires.....	283
.a	La commande <code>find</code>	283
.b	La commande <code>grep</code>	284
.c	La commande <code>sort</code>	285
.d	Les commandes <code>head</code> et <code>tail</code>	285
.e	Les commandes <code>diff</code> et <code>cmp</code>	286
.f	La commande <code>tar</code>	286
.11	TP 11 – Les bases de la programmation shell.....	288
.a	Ecrire une procédure script shell	288
.b	Utilisation de <code>for</code> , <code>test</code> et <code>if</code>	289
.c	Utilisation de <code>while</code> et de <code>expr</code>	290

Chapitre 1

Introduction

2775cbe5ce
Global Knowledge

1. Historique

Au début des années 1970, au sein des laboratoires Bell AT&T, **Ken Thompson** et **Dennis Ritchie** écrivent les premières versions d'un système qu'ils baptisent en premier lieu **Unics** (UNiplexed Information and Computing System) puis finalement **Unix**.

Unix fonctionne initialement sur des matériels DEC de la gamme PDP. Il est rapidement (1973) réécrit en langage C pour s'ouvrir à d'autres architectures matérielles.

Dans les premières années, le code source d'Unix est distribué assez librement aux universités américaines. Ensuite, AT&T envisage de commercialiser son système. Cela incite l'université californienne de **Berkeley** à démarrer le développement d'un système **Unix BSD** (Berkeley System Development). Ce projet est notamment soutenu par le DARPA (Defense Advanced Research Projects Agency) et débouche sur les versions BSD 4.2 (1983) puis BSD 4.3 (1987). Les protocoles **TCP/IP** sont abondamment développés dans le cadre de ces versions.

De son côté, le système AT&T prend le nom d'**Unix System V** (1983) puis System V.3 (1987). Les autres acteurs du marché commencent aussi à développer leur propre version (**HP-UX** pour HP, **AIX** pour IBM...).

En 1990, la version Unix AT&T **System V.4** est le fruit d'une collaboration entre AT&T et SUN. Elle intègre de nombreux standards et fédère un certain nombre d'autres constructeurs (NCR, SGI, Siemens ...).

Dès 1988, inquiets de l'alliance entre AT&T et SUN, certains gros constructeurs (IBM, DEC, HP) décident de créer le groupement **OSF** (Open Software Foundation) avec un projet de version commune **OSF/1** qui voit le jour en 1991. Cette période correspond aussi au démarrage de nombreux projets liés à l'informatique libre tels que les versions **FreeBSD** ou **Linux**.

Par la suite, les réalités du marché ont finalement poussé les plus gros acteurs à faire cavalier seul et à développer leur propre version. L'adoption d'un certain nombre de normes et de standards de fait les rend finalement assez équivalentes en terme fonctionnel et, par conséquent, fortement concurrentielles.

2. Unix aujourd'hui

Aujourd'hui, il faut considérer **Unix** comme un **nom générique** qui désigne des systèmes multi-utilisateurs et multi-tâches, présents sur de nombreuses plates-formes matérielles, en sachant notamment en exploiter les aspects multi-processeurs. L'impact de ces diverses implémentations se traduit au niveau de l'administration système, activité dans laquelle il convient de traduire des concepts généraux en des mises en œuvre spécifiques.

	Utilisateurs non Informaticiens	Utilisateurs Informaticiens	Développeurs	Exploitants, Administrateurs
Impact de la Version	Nul (contexte logiciel)	Faible	Faible	Important

L'héritage historique des deux familles (Unix AT&T et Unix BSD) se traduit parfois par l'existence de commandes ou de bibliothèques concurrentes au sein d'un même système. L'arrivée ultérieure des constructeurs informatiques sur le marché Unix et la concurrence naturelle qui en a découlé se sont révélées comme des facteurs supplémentaires de diversification des implémentations.

.a Principales versions commerciales

Les versions évoluent naturellement assez vite. Les informations données ici reflètent la situation au moment de la rédaction de ces lignes. Une consultation régulière de la presse informatique et des sites Internet permettra de suivre de près l'actualité et l'évolution des offres. Il est fréquent de trouver sur des sites de production des versions antérieures avec un décalage plus ou moins important.

AIX

AIX, d'origine IBM, constitue depuis plusieurs années l'offre commune des constructeurs IBM et Bull. La dernière version proposée est la version **AIX 5L**. L'architecture matérielle s'appuie principalement sur les processeurs *Power*.

HP-UX

HP-UX constitue l'offre du constructeur Hewlett-Packard. La dernière version proposée est la version **HP-UX 11i**. L'architecture matérielle s'appuie sur les processeurs *PA-RISC* ou sur les processeurs *Itanium*.

Solaris

Solaris constitue l'offre du constructeur Sun. La dernière version proposée est la version **Solaris 9**. L'architecture matérielle s'appuie sur les processeurs *Ultra SPARC*. Solaris est également disponible sur plate-forme x86 mais cette configuration se rencontre moins souvent en production.

Autres versions

Les trois versions précédentes dominent largement le marché. D'autres versions, plus anciennes ou moins répandues, doivent malgré tout être mentionnées :

SCO

Outre sa version historique **OpenServer 5**, SCO propose la version **UnixWare 7** sur des plates-formes x86.

Tru64

Tru64 constituait l'offre du constructeur Compaq. La dernière version proposée est **Tru64 V5**. À l'origine, il s'agissait de *Digital Unix* (plus anciennement encore *Ultrix* puis *OSF/1*). L'architecture matérielle s'appuie sur les processeurs *Alpha*. Depuis la fusion de Compaq avec HP, cette version est destinée à être unifiée avec l'offre HP-UX au sein de nouveaux matériels à base de processeurs *Itanium*.

IRIX

IRIX constitue l'offre du constructeur SGI. La dernière version proposée est **IRIX 6.5**. L'architecture matérielle s'appuie sur les processeurs *MIPS*. SGI s'oriente également vers une offre Linux au sein de matériels à base de processeurs *Itanium*.

.b Versions libres

Les logiciels libres proviennent d'une communauté de programmeurs bénévoles reliés par l'Internet. L'ouverture des sources et leur mise à disposition gratuite débouchent sur un mode de développement radicalement différent et se trouvent à l'origine d'un nouveau modèle économique à la fois pour les constructeurs, les éditeurs et les entreprises. Le terme **Open Source** est aujourd'hui très utilisé pour désigner le logiciel libre et il a, sans doute, le mérite de lever l'ambiguïté concernant la gratuité en insistant plus sur la particularité essentielle, à savoir le libre accès au code source.

Un certain nombre de ces versions libres, proches des systèmes Unix traditionnels, est aujourd'hui largement déployé en entreprise :

FreeBSD, OpenBSD, NetBSD

Ces systèmes sont des descendants de la version historique Unix BSD. Ils sont reconnus comme très stables et très performants. Les trois systèmes sont très proches les uns des autres. NetBSD supporte de très nombreuses plates-formes. OpenBSD est très orienté sécurité. FreeBSD se veut plus simple d'utilisation et se place en concurrent direct de Linux sur les plates-formes x86.

Linux

Conçu sur l'initiative du célèbre étudiant finlandais Linus Torvalds, le noyau Linux a été ensuite développé et amélioré par des centaines de spécialistes dans le monde. Le noyau Linux est concrètement accompagné de nombreux composants pour constituer une **distribution** (éditeurs de texte, shell, pile TCP/IP avec implémentation de tous les protocoles et services majeurs, bases de données, bureaux graphiques, langages de programmation, applications variées...). Une distribution Linux correspond donc bien à un système d'exploitation complet

disponible aujourd'hui sur les mêmes architectures matérielles que les systèmes Unix (x86, Power, Sparc, PA-RISC, Alpha, Mips, Itanium...).

On dénombre plus de 200 distributions de Linux qui se différencient par le détail des composants, la procédure d'installation et le paramétrage par défaut des différents services. Certaines d'entre elles sont proposées par des sociétés commerciales qui ont bâti leur stratégie autour des logiciels libres. D'autres sont le fruit d'associations à but non lucratif. Un excellent descriptif détaillé et actualisé existe notamment sur le site www.distrowatch.com.

Parmi les distributions les plus répandues, on peut citer :

RedHat (rebaptisé **Fedora** pour la version gratuite)

Mandrake

SUSE (racheté récemment par la société **Novell**)

Debian

Slackware

Les constructeurs majeurs (IBM, SUN, HP...) élaborent diverses stratégies pour prendre en compte l'avènement des solutions *Open Source*. La manifestation la plus visible en est l'intégration de Linux sur certaines gammes de matériels en complément, voire en remplacement de leur offre Unix traditionnelle. Le développement prévisible des processeurs 64 bits *Itanium* est un facteur supplémentaire de progression des distributions Linux dans le catalogue des solutions proposées.

Les éditeurs adoptent bien évidemment la même démarche en faisant de Linux une plate-forme importante, voire privilégiée, dans leur offre logicielle.

3. Composantes d'un système Unix

Unix est orienté avant tout comme un **système de développement** avec des mécanismes conçus pour favoriser l'activité de programmation. Les développeurs disposent d'un large éventail de fonctionnalités au travers d'interfaces bien normalisées (en langage C). En utilisation interactive, Unix propose des commandes et des utilitaires relativement simples qui offrent une grande richesse fonctionnelle quand on parvient à les combiner intelligemment au travers du langage de commandes (**shell**). Pour tirer le meilleur parti d'Unix, il convient donc d'adopter une approche *boîte à outils* qui passe, d'une part, par la compréhension des mécanismes et, d'autre part, par la connaissance des commandes et de leur syntaxe. Après l'apprentissage initial, la maîtrise du système s'acquiert naturellement au fur et à mesure de la pratique.

.a Le noyau (mono ou multi-processeurs)

Le noyau constitue le cœur du système et réalise les fonctionnalités essentielles de bas niveau (gestion des processus, de la mémoire, des entrées/sorties, implémentation des systèmes de fichiers, des protocoles réseau ...). Il est de petite taille et est écrit à un très fort pourcentage en langage C. Du point de vue de l'utilisateur, nous pouvons considérer ce noyau comme une boîte noire dont l'efficacité et la fiabilité ne sont plus à démontrer.

.b Le shell (langage de commandes)

Le shell est un **programme extérieur au noyau**. Cette caractéristique est essentielle et son intérêt apparaîtra au fur et à mesure de la découverte du système. Lors de la création d'un compte utilisateur, l'administrateur système choisit le shell de connexion qui réalisera l'interprétation de la ligne de commandes. L'utilisateur dispose alors d'un certain nombre de fonctionnalités interactives intéressantes (fichiers de connexion, alias de commandes, rappel de commandes avec possibilités d'édition ...). D'autre part, le shell intègre un aspect programmation (variables, opérateurs, instructions ...) qui permet de réaliser des procédures utilitaires (**scripts**). De ce fait, le shell se révèle comme un outil de travail fondamental pour l'administrateur. Dans les versions commerciales, le shell standard de fait est aujourd'hui le *Korn shell* (**ksh**). Dans les versions Linux, le shell par défaut est plutôt le programme **bash** (*Bourne-Again shell*). Ce dernier tire son nom du shell historique *Bourne shell* (**sh**). Il existe bien d'autres shells disponibles tels que, par exemple, le **csk** (*C shell* issu des versions Berkeley).

.c Les commandes

Le shell réalise donc l'interprétation de la ligne de commandes et joue ainsi le rôle d'interface entre le noyau et l'utilisateur. Unix s'est doté au fil des années de plusieurs centaines de commandes quasiment identiques sur toutes les versions. Heureusement, seule la maîtrise d'environ trente ou quarante d'entre elles s'avère nécessaire et suffisante pour une utilisation efficace au quotidien.

.d Les protocoles TCP/IP

La famille des protocoles TCP/IP est aujourd'hui le standard de fait sur tous les systèmes d'exploitation. Ces protocoles ont été développés principalement sous Unix et ils constituent, bien entendu, une composante fondamentale du système. Une version Unix ou Linux intègre donc directement un ensemble très complet de services :

- Applications standards (*telnet, ftp, remote commands, ssh...*)
 - Partage de disques (*NFS*)
 - Services d'annuaires (*NIS, NIS+, LDAP*)
 - Interfaces graphiques basées sur le protocole *X-Window*
 - Services de nommage et d'adressage (*DNS, DHCP*)
 - Services Internet/Intranet (*Web, Messagerie, FTP, News...*)
- etc...

.e Les interfaces de programmation

Une excellente portabilité des applications (au niveau des sources) est assurée par l'existence de bibliothèques normalisées (en langage C) :

- Programmation système, avec notamment les *IPCs* (InterProcess Communication)
- Programmation réseau (*sockets, TLI, RPCs*)

En complément du langage C, d'autres langages, plus ou moins d'actualité, sont disponibles (Fortran, Cobol, C++, Java...) pour réaliser divers types d'applications.

.f L'offre logicielle - les domaines d'utilisation

Unix, en tant que tel et via une offre très vaste de solutions logicielles, est utilisé aujourd'hui dans beaucoup de contextes variés :

- Unités de recherche et de développement
- Écoles et universités
- Centres de calcul
- Grandes entreprises
- PME

Unix peut remplir quasiment toutes les fonctions du système d'information :

- Serveurs d'infrastructure
- Services Internet/Intranet
- Serveurs de fichiers
- Serveurs de bases de données
- Serveurs d'applications
- Stations de travail graphiques (design, animations, CAO, calculs...)

La fiabilité et l'efficacité des solutions Unix sont aujourd'hui clairement reconnues dans les environnements de production à haute disponibilité.

4. Connexion et environnement de travail

Unix est un système **orienté ligne de commandes**, sur lequel on peut se connecter via un terminal texte asynchrone. Bien entendu, cette possibilité historique est quasiment abandonnée aujourd'hui au profit de connexions réseau via les applications TCP/IP.

.a Diverses possibilités de connexion à un système Unix

Les diverses possibilités de connexion à un système Unix sont les suivantes :

- depuis un **terminal texte asynchrone** (connecté sur un port série ou via un modem)

- depuis un **terminal X**

Un *terminal X* est un terminal graphique directement connecté au réseau. L'administrateur le configure pour y afficher une *boîte de connexion* sur un des systèmes de ce réseau. Une fois authentifié, l'utilisateur dispose d'un environnement de travail graphique multi-fenêtres (interface **X_Window**). En plus des véritables applications graphiques, il est possible de démarrer des émulateurs de terminaux. On peut disposer ainsi, sur ce seul poste, de plusieurs connexions simultanées, éventuellement sur des systèmes différents.

- depuis une station de travail (ou station X)

Sous ce terme officieux de station de travail, nous voulons désigner une machine Unix dotée d'un écran graphique en tant que console. On dispose alors du même environnement que celui décrit précédemment pour le terminal X.

- depuis un autre système qu'Unix

Depuis tout système d'exploitation implémentant les protocoles TCP/IP, il est possible de se connecter à une machine Unix du réseau. Cette connexion se fera, soit via les applications **telnet** ou **ssh** (mode texte), soit via un **émulateur X_Window** (mode graphique). Pour ce dernier aspect, l'offre logicielle est abondante, notamment dans les environnements *Microsoft Windows*.

.b Notions de compte utilisateur

Pour se connecter au système, il faut disposer d'un **compte utilisateur**.

L'administrateur se charge de créer les comptes qui sont caractérisés par un certain nombre d'attributs parmi lesquels :

- le nom de connexion (*login*)

Il s'agit d'une combinaison de lettres minuscules et de chiffres, souvent limitée à huit caractères.

- le mot de passe

Comme sur tous les systèmes d'exploitation, le mot de passe est l'élément de base de la sécurité. Unix permet différents niveaux de gestion, plus ou moins contraignants. Assez souvent, le mot de passe est limité à huit caractères significatifs.

- le **numéro interne (UID : User IDentificator)**

Il s'agit d'un entier qui identifie véritablement l'utilisateur au sein du système. Il est associé au nom de connexion, présent pour des raisons évidentes de commodité.

- le shell de connexion

Dans les versions commerciales d'Unix, le choix du *Korn shell* est recommandé, à la fois en interactif et dans l'activité de programmation de scripts. Dans certains contextes, le *C shell* conserve parfois ses adeptes. Le *Bourne-Again shell* est le shell par défaut dans les distributions Linux mais les autres shells y sont disponibles.

- le répertoire de connexion

L'espace disque Unix apparaît comme une arborescence de répertoires au sein de laquelle l'utilisateur dispose d'une sous-arborescence personnelle qui commence au répertoire de connexion.

- un ou plusieurs **groupes** (groupe primaire et éventuels groupes secondaires)

L'appartenance d'un *login* à des groupes permet une mise en œuvre plus aisée et cohérente des droits d'accès. À l'image des noms de connexion, les noms de groupes sont associés à des numéros internes (**GID** : *Group IDentificator*).

L'administrateur, déjà cité, correspond concrètement à un compte privilégié baptisé **root**. Son numéro interne est 0, ce qui lui donne les pleins pouvoirs sur le système.

Après authentification (nom de connexion et mot de passe), **une connexion en mode texte se traduit donc par l'exécution d'un shell**. Ce programme consiste en un traitement répétitif qui présente une invite de commandes (*prompt*) puis interprète les lignes de commandes validées par l'utilisateur jusqu'à demande de déconnexion.

.c Le fichier `/etc/passwd`

Chaque ligne du fichier `/etc/passwd` décrit un compte utilisateur. Elle est constituée de sept champs séparés par le caractère : (deux points).

nom:mop:uid:gid:commentaire:répertoire:shell

nom

Nom de connexion

mdp

La quasi-totalité des versions utilisent aujourd'hui un fichier spécifique sécurisé pour stocker les informations concernant le mot de passe. Ce champ historique ne donne plus aujourd'hui d'informations pertinentes et contient un caractère variable selon les versions (x ou ! ou *).

<i>uid</i>	Numéro interne associé au nom de l'utilisateur
<i>gid</i>	Numéro interne associé au groupe principal de l'utilisateur
<i>commentaire</i>	Commentaire associé à l'utilisateur
<i>répertoire</i>	Nom complet du répertoire de connexion
<i>shell</i>	Nom complet du shell de connexion

Exemple

```
stage01:x:1001:1000:Compte  
Formation:/home/stage01:/usr/bin/ksh
```

L'utilisateur *stage1* porte le numéro 1001. Son groupe principal porte le numéro 1000. Il se connecte en *Korn shell* dans le répertoire */home/stage1*.

.d Le fichier */etc/group*

Chaque ligne du fichier ***/etc/group*** décrit un groupe. Elle est constituée de quatre champs séparés par le caractère : (deux points).

nom:mop:gid:liste des membres

<i>nom</i>	Nom du groupe
<i>mdp</i>	Ce champ, anciennement destiné à stocker un mot de passe de groupe, est obsolète aujourd'hui et contient un caractère variable selon les versions (x ou ! ou *).
<i>gid</i>	Numéro interne associé au groupe
<i>liste des membres</i>	Ce quatrième champ peut être vide. Dans le cas contraire, il contient une liste de comptes (on sépare les noms par des virgules) pour lesquels le groupe est un groupe supplémentaire.

Exemple

```
gkn:!:1000:  
ecole:!:2000:stage1,stage2,stage3
```

Le groupe *gkn* porte le numéro 1000 et est le groupe principal des utilisateurs ayant la valeur 1000 comme quatrième champ du fichier */etc/passwd*. Le groupe *ecole* porte le numéro 2000 et est le groupe principal des utilisateurs ayant la valeur 2000 comme quatrième champ du fichier */etc/passwd*. En outre, le groupe *ecole* est un groupe supplémentaire pour les utilisateurs *stage1*, *stage2* et *stage3*.

5. La documentation

.a Ressources Internet

Après une période où les versions Unix proposaient l'accès à leur documentation complète via un environnement graphique spécifique, la consultation se fait tout naturellement aujourd'hui depuis un navigateur Web, soit en format *HTML*, soit en format *PDF*.

Parmi les nombreux sites intéressants, on peut citer :

docs.sun.com Documentations officielles pour Solaris

docs.hp.com Documentations officielles pour HP-UX

publib16.boulder.ibm.com/pseries/fr_FR/infocenter/base/aix.htm

Documentations pour AIX

En ce qui concerne AIX, le site **www.redbooks.ibm.com** constitue également un excellent centre de ressources.

Pour les distributions **Linux**, le site de chaque distribution constitue bien évidemment un excellent point d'entrée. D'autre part, le site **www.tldp.org** (The Linux Documentation Project) constitue un portail de référence.

.b La commande man

Sur toutes les versions, la documentation de référence est également disponible via la commande **man** qui est organisée en différents manuels. Son découpage historique a beaucoup évolué mais les **commandes publiques** correspondent pratiquement toujours au **volume 1** alors que les commandes administratives s'inscrivent dans le *volume 1M*.

Pour connaître le détail de l'implémentation sur une version donnée, le bon réflexe est de faire appel à : *man man*.

Exemples Solaris

```
$ man man
```

User Commands

man(1)

NAME

man - find and display reference manual pages

SYNOPSIS

```
man [-] [-adFlrt] [-M path] [-T macro-package] [-s section]
name...
```

```
man [-M path] -k keyword...
```

```
man [-M path] -f file...
```

DESCRIPTION

The man command displays information from the reference manuals. It displays complete manual pages that you select by name, or one-line summaries selected either by keyword (-k), or by the name of an associated file (-f). If no manual page is located, man prints an error message.

.....

Il est souvent nécessaire de connaître le nom de la commande dont on veut la description. Cependant, l'option **-k** permet une recherche sur mot clé.

Dans cet exemple, nous remarquons que deux rubriques trouvées sont des commandes publiques (*volume 1*). Les autres correspondent à des fonctions en langage C (*volume 3C*).

```
$ man -k calendar
```

```
cal          cal(1)          display a calendar
calendar     calendar(1)    reminder service
difftime     difftime(3c)   computes the difference between two calendar
times
mktime       mktime(3c)     converts a tm structure to a calendar time
$
```

L'option **-s (section)** fait référence aux différents manuels et peut être utile pour gérer, par exemple, les homonymies entre commandes et fichiers de configuration.

Nous obtenons, dans un premier temps, la documentation de la commande *passwd* (*volume 1*). Si nous souhaitons obtenir celle du fichier */etc/passwd* (fichier de description des comptes), nous devons demander explicitement le *volume 4* (fichiers de configuration).

```
$ man passwd
```

```
User Commands                                passwd(1)
```

```
NAME
```

```
passwd - change login password and password attributes
```

```
.....
```

```
$ man -s 4 passwd
```

```
File Formats                                passwd(4)
```

```
NAME
```

```
passwd - password file
```

```
SYNOPSIS
```

```
/etc/passwd
```

```
DESCRIPTION
```

```
The file /etc/passwd is a local source of information about users'
accounts. The password file can be used in conjunction with other
password sources, such as the NIS maps passwd.byname and passwd.bygid and
the NIS+ table passwd.
```

```
.....
```

Signalons également l'intérêt de la rubrique **See Also** qui permet d'identifier des commandes de même thème.

6. Premiers éléments de syntaxe

Unix fait la différence entre les minuscules et les majuscules. Tous les noms de commandes du système sont en minuscules. Il s'agit, la plupart du temps, de fichiers exécutables (programmes C compilés ou procédures écrites en shell) ou bien quelquefois de commandes internes, c'est-à-dire intégrées au programme shell lui-même.

Un nom de commande peut être suivi d'**options** introduites par le signe - (moins) ou le signe + (plus).

Il y a des **options simples** (un caractère alphanumérique ou un mot clé seul) et des **options à argument** constituées d'un mot clé et d'un paramètre associé.

Certaines versions, comme Linux, intègrent des options constituées d'un double signe - (moins) suivi d'un mot clé.

Après les éventuelles options, la commande peut également recevoir des **arguments** ou **paramètres**. Si ces arguments sont des noms de fichier, ils peuvent contenir des **caractères génériques** permettant des désignations abrégées.

📖 *Il est important de signaler dès maintenant que ces éventuels caractères génériques sont résolus par le shell avant l'exécution de la commande. Celle-ci reçoit toujours une liste complète de paramètres. Ce choix de conception simplifie avantageusement la programmation des commandes mais peut se révéler troublant dans la phase d'apprentissage.*

Quelques exemples (sans préciser ici le rôle des commandes)

- | | |
|---------------------------------------------------------|---------------------|
| - Commande sans option ni argument | who |
| - Commande avec une option simple | who -q |
| - Commande avec une option à argument | man -k user |
| - Commande avec une option de <i>style Linux</i> | date --help |
| - Commande avec une option simple et un argument | head -5 toto |
| - Commande avec un argument mais sans option | cal 1959 |
| - Commande avec options simples et arguments génériques | ls -ul toto* |

Certaines commandes ne suivent pas la syntaxe de base. D'autre part, des mécanismes plus avancés (redirections, *pipeline*...) viennent compliquer la ligne de commande. Les chapitres suivants et notamment l'étude du shell détailleront progressivement ces aspects complémentaires.

Lors de la frappe, il est possible d'**annuler un caractère** grâce à la touche *backspace* (le plus souvent associée à *Ctrl h*). L'**annulation de la ligne complète** s'obtient quelquefois par le caractère @, mais le plus souvent par *Ctrl u*.

Pour **se déconnecter**, nous pouvons utiliser la commande interne *exit* ou bien taper le caractère *Ctrl d* en début de ligne. Ceci provoque une lecture clavier vide et termine ainsi l'exécution du shell. Le système se charge alors de présenter à nouveau une bannière de connexion.

7. Premières commandes utiles

Nous présentons ici quelques premières commandes, d'une part pour leur intérêt fonctionnel, mais aussi pour nous familiariser avec la grammaire de base de la ligne de commandes Unix.

📖 *Les options ne sont pas présentées de façon détaillée car nous renvoyons le lecteur soit vers un aide-mémoire des commandes, soit vers la documentation en ligne du système via la commande `man`. Par souci d'homogénéité, la quasi-totalité des exemples de ce document sont issus d'une version Solaris 9.*

.a Effacement de l'écran

La commande **clear** permet d'effacer l'écran. Il s'agit d'une commande très simple sans options ni paramètres.

.b Calendrier perpétuel

La commande **cal** permet d'afficher un calendrier. Sans argument, elle affiche le mois courant. Si nous lui fournissons un seul argument, elle le considère comme une année complète. Si deux paramètres sont donnés, le premier représente le mois et le second représente l'année.

```
$ cal

    September 2004

S   M  Tu   W  Th   F   S
                1   2   3   4
5   6   7   8   9  10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28  29  30

$ cal 2 1959
```

February 1959

```
S  M Tu  W Th  F  S
1  2  3  4  5  6  7
8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28

$
```

.c Qui est connecté ?

La commande **who** comporte de nombreuses options. Elle permet ainsi d'obtenir, sous divers formats, des informations sur les utilisateurs connectés.

La forme particulière **who am i** identifie la connexion courante.

La commande **tty** affiche le nom du terminal.

Liste des utilisateurs connectés

```
$ who
root      console    sept  1 15:29    (:0)
mike      pts/2       sept  1 17:46    (192.168.0.2)
mike      pts/4       sept  1 17:49    (solaris9)

$
```

Liste résumée et nombre d'utilisateurs

```
$ who -q
mike  root  mike
# users=3

$
```

Identification de la connexion courante

```
$ who am i
mike      pts/2       Sep  1 17:46    (192.168.0.2)

$
```

Nom du terminal

```
$ tty
/dev/pts/2

$
```

.d Commandes d'identification

La commande **id** affiche le nom et le numéro de l'utilisateur connecté ainsi que les noms et numéros de ses groupes.

La commande **groups** affiche uniquement les noms des groupes.

La commande **uname** donne des informations sur le système et sa version.

La commande **hostname** affiche le nom réseau du système.

Identification de l'utilisateur courant

```
$ id
uid=2000(mike) gid=2000(mike)
$ id -a
uid=2000(mike) gid=2000(mike) groups=2000(mike),3000(gkn)
$ groups
mike gkn
$
```

Nom du système d'exploitation

```
$ uname
SunOS
$
```

L'option -n indique le nom réseau (équivalent à la commande hostname)

```
$ uname -n
solaris9
$
```

L'option -a donne le maximum d'informations

```
$ uname -a
SunOS solaris9 5.9 Generic_112234-10 i86pc i386 i86pc
$
```

.e Afficher du texte

La commande **echo** affiche la liste de ses paramètres en les séparant par **un seul espace** et en terminant par un saut de ligne.

La commande interprète quelques caractères conventionnels :

\n	saut de ligne supplémentaire
\b	backspace
\c	annulation du saut de ligne final
\t	tabulation
\r	retour chariot
\0n	caractère de code ascii <i>n</i> (en octal)

Il faut utiliser les " (doubles quotes) pour une bonne interprétation de ces caractères et pour éviter des conflits avec des caractères spéciaux du shell.

La commande **banner** affiche ses paramètres en gros caractères.

```
$ echo Bienvenue chez GK
Bienvenue chez GK
$
$ echo "\tPremiere ligne\n\tDeuxieme ligne"
    Premiere ligne
    Deuxieme ligne
$ banner unix
#   #   #   #   #   #   #
#   #   ##  #   #   #   #
#   #   #  #   #   #   ##
#   #   #  #  #   #   ##
#   #   #   ##  #   #   #
###  #   #   #   #   #
$
```

.f Modifier son mot de passe

Les diverses caractéristiques du mot de passe (durée de validité, contraintes syntaxiques...) sont liées aux possibilités de la version ainsi qu'aux éventuels choix de l'administrateur.

La commande **passwd** permet de modifier son mot de passe de manière interactive. La saisie du mot de passe se fait sans écho des caractères tapés. Sur beaucoup de versions, seuls les huit premiers caractères du mot de passe sont significatifs.

.g Afficher l'heure ou la date système

La commande **date** permet d'afficher la date et l'heure système sous des formats variés. L'étendue des possibilités nécessite de recourir à la commande *man* pour une utilisation efficace.

L'option de cette commande commence par le signe **+** (plus). Il faut ensuite préciser un format d'affichage qui pourra être constitué de texte libre et de spécificateurs introduits par le caractère **%**.

La commande sans option affiche la date et l'heure système sous un format de base.

```
$ date
Wed Sep  1 18:04:48 CEST 2004
$
```

Les spécificateurs **%H** et **%M** désignent respectivement les heures et les minutes.

```
$ date "+Il est %H:%M"
Il est 18:05
$
```

.h Récapitulatif des commandes à approfondir dans la documentation

banner	Affichage en gros caractères
cal	Affichage d'un calendrier
clear	Effacement de l'écran
date	Affichage de la date et de l'heure
echo	Affichage de texte

groups	Liste des groupes d'un utilisateur
hostname	Affichage du nom réseau
id	Identification d'un utilisateur
passwd	Choix d'un mot de passe
tty	Affichage du nom du terminal
uname	Informations sur le système
who	Liste des utilisateurs connectés

2775cbe5ce
Global Knowledge

8. TPs 1 et 2 – Ouverture de session et premières commandes

Ce qui est fait dans ce TP :

Le propos de cet exercice est de découvrir les étapes de connexion pour commencer une session d'utilisateur Unix, de se familiariser avec quelques commandes de base, l'usage de la documentation en ligne, quelques actions au clavier, puis de mettre fin à la session en se déconnectant.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- . Vous connecter, déconnecter et changer votre mot de passe,
- . Lancer quelques commandes de base,
- . Utiliser les utilitaires **mail** et **write** pour communiquer avec d'autres personnes,
- . Utiliser des actions claviers pour contrôler l'affichage généré par les commandes.

Remarque : Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

Introduction

Pour lancer une commande saisie sur la ligne de commande, vous utilisez la touche <Entrée> (ou <Enter>) et non pas <Ctrl> comme sur certains systèmes.

Pour corriger une faute de frappe utilisez la touche <BackSp> ou la touche < ← > (<retour arrière>), mais pas les flèches qui peuvent générer des comportements inattendus !

Limitez vous à l'utilisation de la partie gauche du clavier. Y compris pour les touches numériques. Les touches du pavé numérique situé à droite, peuvent ne pas fonctionner comme prévu ...

.a Connexion / Changement de mot de passe

- ___ 1. Ouvrez une session avec le nom et le mot de passe qui vous ont été donnés par l'animateur. Il est de la forme **stagexx** où **xx** est un nombre comme **01**, **02** etc... Changez votre mot de passe. Le mot de passe que vous indiquez n'est pas affiché.

- ___ 2. Vérifiez que le mot de passe a été défini en vous déconnectant puis en vous reconnectant de nouveau.

.b Commandes de base

- ___ 3. Affichez la date du système.

- ___ 4. Affichez le calendrier de l'année 2014.

- ___ 5. Affichez le mois de février 1682 (ou bien septembre 1752 si le système est basé sur les calendriers anglo-saxons). Vous remarquerez quelle particularité ?

- ___ 6. Affichez le mois de janvier pour l'année 1999 et pour l'année 99. Est-ce que 1999 et 99 affichent la même chose ? _____

- ___ 7. Lancez la commande qui affiche les informations des utilisateurs connectés.

- ___ 8. Affichez juste votre nom de connexion.

- ___ 9. Utilisez la commande **echo** pour afficher la chaîne de caractère **A table !**

- ___ 10. Utilisez la commande **clear** pour effacer l'écran.

.c La documentation en ligne : la commande `man`

- ___ 12. Affichez la page d'utilisation du manuel en ligne lui même par une commande `$man man <entrer>`.

Utilisez la barre d'espace pour avancer d'un écran et la touche **<Entrée>** pour avancer d'une ligne. Appuyez sur la touche **b** pour revenir en arrière. Lorsque vous avez assez lu, sortir du manuel à l'aide de la touche **q** ou **<CTRL-C>**.

- ___ 13. Recherchez les rubriques du manuel en ligne qui traite du calendrier (« calendar » si la documentation est en anglais).
- ___ 14. La documentation vous propose la commande **cal**. Utilisez **man**, sans option, pour afficher la syntaxe de la commande **cal**.

.d Envoyer et recevoir du courrier : commande **mail**

- ___ 12. Envoyez vous un courrier à vous même en utilisant la commande **mail**. Renseignez le sujet mais ignorez la copie vers un autre destinataire.
- ___ 13. Lancez la commande **mail** pour afficher la liste des messages de votre boîte à lettre. Ouvrez le message, le sauvegarder, puis quittez l'utilitaire **mail**. Pour afficher le résumé des sous-commandes de **mail**, tapez **?** à l'invite de **mail**.
- ___ 14. Ouvrez de nouveau votre courrier et supprimez le message que vous aviez sauvegardé. Quittez le programme **mail**.

.e Communiquer avec les autres utilisateurs

- ___ 15. Envoyez une note à toutes les personnes connectées comme quoi vous avez terminé l'exercice.
- ___ 16. Pour cet exercice pratiquez de concert avec votre voisin(e) **stageyy**. Ouvrez une ligne pour saisir un texte qui est envoyé vers l'écran de votre voisin(e). Attendez que s'affiche sur votre écran une réponse de sa part, comme quoi il(elle) ne peut pas communiquer pour le moment, suivi de l'indicateur de fin de texte. Fermez à votre tour la ligne de saisie de texte et mettez ainsi fin à votre session de messagerie instantanée.

.f Actions au clavier

Le but est de stoppez ponctuellement l'affichage d'un texte long qui défile à l'écran, puis de reprendre le défilement.

___ 17. Lancez la commande `ls -lR /` (option 'l' de lettre et non pas le chiffre 1, ni le 'l' en majuscule. La signification sera présentée plus tard). Par des actions au clavier: stoppez le défilement. Puis reprendre le défilement

___ 18. Relancez la commande **précédente**, mais en ne tapant que les quatre premiers caractères, SANS valider en N'appuyant PAS sur la touche **<Enter>**. Effacez la ligne saisie en tapant **<ctrl-u>**. Puis utilisez **echo** pour afficher **Fin de l'exercice**.

Cette fois faites une faute de frappe et utilisez la touche **<retour arrière>** pour corriger.

___ 19. Déconnectez vous.

Fin de l'exercice

Chapitre 2

Les systèmes de fichiers

2775cbe5ce
Global Knowledge

1. Vision globale de l'organisation des disques

Pour l'utilisateur, le système de fichiers Unix apparaît comme une **arborescence de répertoires** dont la racine se nomme **/**.

Cette arborescence globale est très fréquemment constituée de plusieurs sous-arborescences qui sont rassemblées, lors du démarrage du système, par une opération dite de **montage**. Une sous-arborescence est baptisée **filesystem**.

Selon les versions, les administrateurs système disposent de deux types d'organisation des disques . Il s'agit, d'une part, des **partitions** et, d'autre part, des **volumes logiques**.

Un filesystem correspond concrètement à une structure physique qui occupe l'espace disque correspondant, soit à une partition, soit à un volume logique. Plusieurs types de filesystems sont disponibles selon les versions.

L'organisation physique des disques doit être complètement transparente pour les utilisateurs et les programmes. Sa maîtrise constitue une compétence essentielle de l'administrateur système.

Dans tous les cas, Unix requiert une région disque complémentaire baptisée historiquement **zone de swap** mais nommée plus souvent aujourd'hui **espace de pagination**. Cet espace permet l'implémentation de la mémoire virtuelle. Il est géré directement par le noyau et ne comporte pas, bien entendu, de structure de filesystem.

Signalons enfin que certaines sous-arborescences peuvent parfois se trouver physiquement sur une autre machine. L'opération de montage se fait alors à travers le réseau grâce notamment à l'application **NFS** (Network FileSystem), standard de fait TCP/IP pour le partage de fichiers.

2. Notion de filesystem

Pour que les partitions ou les volumes logiques puissent être utilisés comme des arborescences, il est nécessaire d'y créer une structure de filesystem.

La structure physique d'un filesystem peut se résumer ainsi :

- des blocs de gestion
- une table des inodes



La notion d'inode est précisée dans la suite de ce chapitre. Dans le répertoire où il se trouve, chaque fichier Unix est associé à un numéro d'inode. La dimension de la table des inodes correspondra au nombre maximum de fichiers du filesystem.

- la zone des fichiers

L'espace disque est alloué par blocs. Les blocs disque occupés par un fichier sont retrouvés grâce aux informations d'adressage situées dans son inode. D'autre part, les blocs de gestion du filesystem permettent d'allouer rapidement des blocs libres.

Le détail de cette structure physique et les mécanismes d'allocation de l'espace disque se sont perfectionnés en terme de sécurité et de performance au fur et à mesure des années. Dans l'ordre chronologique, nous pouvons distinguer trois types de filesystems :

- type System V

Cette première famille est quasiment obsolète aujourd'hui.

- type Berkeley

Par rapport à la génération précédente, on dispose de meilleures performances (blocs disque plus grands, meilleurs algorithmes d'allocation...) et d'une plus grande fiabilité (redondance des blocs de gestion). Beaucoup de versions utilisent encore aujourd'hui ce type de filesystem.

- type journalisé

Il s'agit de la dernière génération qui tend à devenir le type par défaut sur la majorité des versions. Les performances en écriture sont meilleures et le mécanisme de journalisation accélère grandement les opérations de vérification lors du démarrage du système.

Les deux dernières familles fonctionnelles correspondent aujourd'hui à des types aux noms variés selon les versions :

AIX	jfs (journalisé) et jfs2 (évolution de jfs)
HP-UX	hfs (Berkeley) et vxfs (journalisé)
Solaris	ufs (Berkeley et journalisé dans les implémentations récentes)
Linux	ext2 (Berkeley) et ext3 ou reiserfs (journalisés)

.a Filesystems de type journalisé

Cette nouvelle génération de filesystems s'impose aujourd'hui comme standard de fait en améliorant encore les performances et la fiabilité. Les opérations en cours sur le filesystem sont conservées, sur disque, dans un journal des transactions. Une transaction est un ensemble d'opérations élémentaires qui doivent obligatoirement être effectuées en totalité pour garantir la cohérence de la structure du filesystem. En cas d'arrêt anormal du système, les procédures de reprise consistent à examiner le contenu du journal afin de traiter d'éventuelles transactions non terminées. Ce mécanisme s'avère beaucoup plus rapide qu'une véritable vérification de la structure physique et constitue, de ce fait, un avantage intéressant par rapport à la génération Berkeley. Il faut cependant bien préciser que la journalisation ne garantit aucunement le contenu des fichiers, son seul objectif est bien la cohérence de la structure.

📖 *Lorsqu'une version propose plusieurs types de filesystems, il convient, a priori, de choisir le plus récent (à savoir le type journalisé). Les filesystems de type Berkeley continuent, bien entendu, à être proposés à titre de compatibilité. Il se pourrait qu'un logiciel nécessite ce type particulier (ce cas est tout de même rare) ou bien encore que des utilitaires, tels que des scripts de sauvegardes, ne soient pas adaptés à cette dernière génération.*

.b Montage

L'opération de montage est nécessaire pour accéder aux fichiers d'un filesystem par les commandes Unix classiques. Elle consiste à accrocher l'arborescence dans un répertoire vide d'un filesystem lui-même déjà monté. Ce répertoire s'appelle le point de montage (**mount point**).

Le filesystem contenant la racine Unix et les répertoires de premier niveau est baptisé **filesystem root**. Il est monté dès le début de la phase de démarrage du système. Cela permet ensuite la constitution, via d'autres montages successifs, de l'arborescence complète vue par les utilisateurs. Un fichier de configuration, spécifique à chaque version, permet de décrire les montages automatiques au démarrage et de bénéficier d'abrégiés dans les commandes ultérieures de gestion.

La commande **mount** réalise l'opération de montage. Dans sa syntaxe complète (réservée à l'administrateur système), elle nécessite deux arguments :

- le nom de la partition ou du volume logique contenant le filesystem
- le répertoire de montage

Des options spécifiques au type de filesystem peuvent être ajoutées. L'option la plus intuitive permet d'effectuer le montage en lecture seule. Dans ce cas de figure, aucune modification, même par le compte *root*, ne peut avoir lieu dans l'arborescence.

La commande *mount* sans arguments donne la liste des filesystems montés. Les utilisateurs ordinaires n'ont accès qu'à cette seule forme de la commande.

```
$ mount
/ on /dev/dsk/c0d0s0 read/write/setuid/largefiles on Thu Sep 2 07:30:55
2004
.....
/home on /dev/dsk/c0d0s7 read/write/setuid/largefiles on Thu Sep 2
07:31:01 2004
$
```

A l'aide de ces premières informations, nous comprenons certains intérêts majeurs d'une organisation en filesystems :

- Mise à disposition maîtrisée (montage ou démontage)
- Séparation fonctionnelle du système proprement dit et des données ou logiciels
- Sécurité complète de certaines arborescences (montage en lecture seulement)

.c Filesystems de type CD-Rom

Chaque version Unix implémente un type de filesystem permettant l'accès aux CD-Rom, à la manière d'un disque dur, via une opération de montage.

Le nom de ce type de filesystem varie selon les versions :

AIX	cdarfs
HP-UX	cdfs
Solaris	hsfs
Linux	iso9660

Certains problèmes historiques liés à l'utilisation des CD-Rom sous Unix sont plus ou moins bien résolus dans les versions récentes. La commande *mount* étant réservée au compte *root*, cela peut empêcher un utilisateur ordinaire de se servir de lui-même du lecteur de CD-Rom. Le montage automatique, à l'aide d'un processus spécialisé, résout le plus souvent ce problème. De la même façon, la commande **umount**, nécessaire en fin d'utilisation, est réservée au compte *root*. D'autre part, il ne faut pas oublier de démonter l'arborescence avant d'éjecter physiquement le disque sous peine de rendre le lecteur inutilisable (*busy*) pour un autre montage. Un démontage automatique lié à l'éjection du disque résout également ce problème sur les implémentations récentes.

.d Pseudo filesystem /proc

Le répertoire **/proc** qui apparaît aujourd'hui dans les arborescences d'une grande majorité de versions Unix correspond à un pseudo-filesystem permettant une représentation interne des processus en activité, sous forme de fichiers. Il est clair qu'il ne s'agit pas d'un répertoire classique et qu'il est prudent de ne pas y effectuer de manipulations directes. Chaque processus correspond, en fait, à un sous-répertoire de */proc*. Ce sous-répertoire contient un ensemble de fichiers qui stockent diverses informations liées au processus concerné. On peut trouver également dans */proc* un certain nombre de fichiers individuels décrivant divers aspects de la configuration du système. Ces fichiers servent d'interface à beaucoup de commandes.

 *La notion de processus est détaillée dans le chapitre 4 : Processus et Mécanismes.*

3. Types et désignations des fichiers

Un fichier Unix possède un type.

Nous pouvons énumérer **quatre types** principaux de fichiers :

- Fichier ordinaire

Le noyau Unix considère un fichier comme une simple suite d'octets et ne distingue pas les types de contenu (texte, binaire...). Pour identifier la nature de ce contenu, il sera nécessaire de recourir à une commande qui lit le début du fichier (commande *file*).

- Répertoire

De par le concept même d'arborescence, tout fichier, quel que soit son type, appartient à un répertoire.

- Fichier spécial

Un fichier spécial est le nom d'un périphérique. Une des particularités des systèmes Unix consiste en effet à désigner les périphériques par l'intermédiaire de noms de fichiers situés dans le répertoire */dev*. Le fait de désigner un fichier spécial, dans une commande appropriée, active implicitement le programme de gestion (*driver*) de ce périphérique.

- Lien symbolique

Un lien symbolique contient le nom d'un autre fichier et joue le rôle de pointeur vers cet autre emplacement. Les liens symboliques sont apparus notamment pour permettre les évolutions de l'arborescence Unix en préservant la compatibilité avec les arborescences antérieures.

.a Les conventions à connaître

Le noyau n'intègre pas de sémantique sur les noms des fichiers et **la notion d'extension n'existe pas**.

Il n'y a **pas de caractères interdits** pour constituer un nom de fichier (sauf le */* qui sert de séparateur dans les chemins). Nous nous limiterons sans doute aux **lettres** et aux **chiffres** ainsi qu'à l'éventuel caractère *_* (tiret bas) si nous souhaitons constituer des noms composés.

Les minuscules et les majuscules sont différentes. Il vaut mieux nommer tous ses fichiers en minuscules et réserver l'utilisation des majuscules à des fichiers particuliers que l'on voudrait voir apparaître en tête de liste dans un affichage (*README* ou *LISEZMOI* par exemple).

Par tradition, certains fichiers d'initialisation ou de paramétrage ont des **noms qui commencent par un point**. Par exemple, le fichier de nom *.profile* sert à paramétrer le comportement de l'interpréteur de commandes. Il convient d'insister à nouveau sur le fait que ces noms particuliers ne signifient rien au niveau du noyau Unix. Ils ne remplissent un rôle particulier que pour tel ou tel programme. Les répertoires de connexion des utilisateurs sont les seuls endroits où se trouvent habituellement ces fichiers.

Sur d'anciennes versions, les noms de fichiers étaient limités à 14 caractères. Nous disposons maintenant de noms longs pouvant aller jusqu'à **255 caractères**.

.b Comment désigner les fichiers ?

Un fichier, quel que soit son type, appartient toujours à un répertoire au sein de l'arborescence Unix. Pour désigner un fichier, nous mentionnons donc (implicitement ou explicitement) le répertoire qui le contient. Les répertoires contiennent des entrées qui sont, en fait, des couples "*numéro d'inode, nom de fichier*". Le système, grâce au numéro d'inode associé au nom, trouve, dans la table des inodes du filesystem concerné, toutes les caractéristiques précises de ce fichier. Un inode contient en effet la description complète du fichier.

On y trouve notamment les informations suivantes :

- **Type** du fichier (ordinaire, répertoire, lien symbolique, fichier spécial...)
- **Taille** (en octets)
- Propriétaire et groupe
- Droits d'accès
- **Adresses des blocs** disque qui constituent le fichier
- Trois **dates** système
- date de dernière modification (la date de création n'est pas conservée)
- date de dernière consultation
- date de dernière modification de l'inode lui-même
- nombre de liens physiques

Un répertoire contient, dès sa création, deux références par défaut, utilisées pour parcourir l'arborescence :

- | | |
|-----------|-------------------------------------------------------------------------------------------------------|
| . (point) | Ce point désigne le répertoire lui-même. |
| .. | Ces deux points désignent le répertoire parent, c'est-à-dire le niveau supérieur dans l'arborescence. |

Nous disposons de deux façons de nommer les fichiers dans les lignes de commandes Unix :

- **Référence absolue** (nom complet ou chemin complet)

Nous désignons, de façon unique, le fichier à partir de la racine. Le nom commence alors par le caractère / .

Exemples

/home/mike/.profile

/usr/bin/date

- **Référence relative**

Nous désignons le fichier par rapport au répertoire courant.

Exemples

../mike/toto

../../usr/bin/date

./toto (équivalent à *toto*)

4. Parcours et visualisation de l'arborescence Unix

.a Commandes de base (pwd, cd, ls)

Pour découvrir concrètement l'organisation de l'arborescence Unix, nous allons introduire trois premières commandes de base :

pwd	Nom complet du répertoire courant
cd	Changement de répertoire courant
ls	Affichage du contenu d'un répertoire

```
$ pwd
/home/mike
$
```

Utilisation d'un chemin absolu

```
$ cd /usr
$ pwd
/usr
$
```

Sans argument, la commande cd nous repositionne dans le répertoire de connexion.

```
$ cd
$ pwd
/home/mike
$
```

Utilisation d'un chemin relatif

```
$ cd ../stage1
$ pwd
/home/stage1
$
```

La commande **ls** visualise, de façon plus ou moins détaillée, le contenu d'un répertoire donné en argument. Sans paramètre, elle traite le répertoire courant.

De nombreuses options sont disponibles :

-l	Affichage détaillé
-----------	--------------------

-d	Caractéristiques du répertoire lui-même (au lieu de lister son contenu)
-i	Affichage du numéro d'inode
-t	Tri par date de modification
-u	Affichage de la date de consultation (au lieu de la date de modification)
-a	Affichage des fichiers dont le nom commence par un . (point)
-R	Liste récursive de tous les sous-répertoires

Avec l'*option -l*, les renseignements affichés sont, de gauche à droite :

Type (sur une position)

d	Répertoire
-	Fichier ordinaire
l	Lien symbolique
b ou c	Fichier spécial (type <i>bloc</i> ou <i>caractère</i>)

Droits d'accès (sur 9 positions)

Il s'agit de trois combinaisons de trois permissions (*r w x*) correspondant respectivement au propriétaire, au groupe et aux *autres* (les comptes n'appartenant pas au groupe). La sémantique de ces droits d'accès sera détaillée à la fin de ce chapitre.

Nombre de liens

La notion de lien est également détaillée plus loin dans ce même chapitre.

Propriétaire et Groupe

Taille (en octets)

Date de dernière modification

L'*option -u* permet d'afficher la date de dernière consultation.

Nom du fichier

Exemples

Dans cet affichage, nous reconnaissons successivement un lien symbolique, un répertoire et un fichier ordinaire.

```
$ ls -l /etc
total 504
.....
lrwxrwxrwx  1 root  root   14 Jan 14  2004 aliases -> ./mail/aliases
drwxr-xr-x  2 root  bin   512 Jan 14  2004 apache
.....
-r-----  1 root  sys    478 Sep 12 11:01 shadow
.....
```

Cet autre exemple met en évidence le rôle de l'*option -a*. Un répertoire qui apparaît comme vide avec la seule *option -l* peut contenir, en fait, des fichiers dont le nom commence par un point. De plus, les références . (répertoire courant) et .. (répertoire parent) sont présentes dans tout répertoire et seront également visualisées par cette *option -a*.

```
$ ls -l
total 0
$ ls -al
total 8
drwxr-xr-x  2  mike  mike   512  Sep 12 16:43 .
dr-xr-xr-x 12  root  root   512  Sep  1 20:55 ..
-rw-r--r--  1  mike  mike   144  Aug 25 16:26 .profile
-rw-----  1  mike  mike     8  Sep 12 16:44 .sh history
$
```

.b Les principaux répertoires de la racine Unix

Dans l'exemple ci-dessous (issu d'une version **Solaris**), nous ne conservons volontairement que les répertoires les plus usuels, présents sur quasiment toutes les implémentations d'Unix.

```
$ ls -al /
.....
lrwxrwxrwx  1  root  root      9  jan  3 07:22  bin -> ./usr/bin
.....
drwxr-xr-x 14  root  sys    3584  jan 23 12:22  dev
.....
drwxr-xr-x 40  root  sys    3584  jan 23 11:58  etc
.....
dr-xr-xr-x  3  root  root     512  jan 21 18:34  home
.....
lrwxrwxrwx  1  root  root      9  jan  3 07:22  lib -> ./usr/lib
.....
drwxrwxr-x  6  root  sys     512  jan  3 08:12  opt
.....
drwxr-xr-x  2  root  sys    1024  jan  3 07:25  sbin
drwxrwxrwt  7  root  sys     452  jan 23 12:21  tmp
.....
drwxr-xr-x 32  root  sys    1024  jan  3 07:56  usr
drwxr-xr-x 30  root  sys     512  jan  3 08:35  var
.....
$
```

/bin

Commandes publiques (accessibles à tous les utilisateurs)

(*/bin* est très souvent un lien symbolique vers */usr/bin*.)

/dev

Fichiers spéciaux (noms des périphériques)

/etc

Fichiers de configuration et commandes administratives

(les commandes sont souvent des liens symboliques

vers leur homonyme dans */sbin* ou */usr/sbin*.)

/home

Répertoires de connexion des utilisateurs locaux

(le répertoire ***/export/home*** contiendra plutôt les répertoires

de connexion des comptes réseau, partagés via le service **NFS**.)

/lib

Librairies, programmes des compilateurs, commandes diverses...

(*/lib* est très souvent un lien symbolique vers */usr/lib*.)

/opt

Répertoire proposé pour l'installation de certains logiciels optionnels

/sbin

Commandes ou scripts administratifs utilisés au démarrage du système

/tmp

Répertoire public pour fichiers temporaires

/usr

Répertoire stable contenant les sous-répertoires correspondant aux diverses composantes du système (commandes, exécutable, bibliothèques, documentations...)

/var

Répertoire variable contenant les sous-répertoires associés à l'activité quotidienne du système (files d'attente, fichiers de statistiques et de compte-rendu...)

.c Créer ou supprimer des répertoires (mkdir, rmdir)

La commande **mkdir** permet la création de répertoires. La commande **rmdir** permet de supprimer des répertoires vides. La suppression de répertoires non vides se fait via la commande **rm** présentée plus loin dans ce chapitre.

```
$ mkdir rep
$ ls -ld rep
drwxr-xr-x  2  mike  mike   512   jan 23 12:26  rep
$
```

L'*option -p* est intéressante car elle permet de créer plusieurs niveaux de sous-répertoires en une seule commande.

```
$ mkdir -p rep/rep2/rep3
$
```

La commande **rmdir** ne permet pas la suppression d'un répertoire non vide. Dans notre cas, il nous faut supprimer successivement tous les sous-répertoires en partant du niveau le plus bas.

```
$ rmdir rep
rmdir: directory "rep": Directory not empty
$ cd rep/rep2
$ rmdir rep3
$ cd ..
$ rmdir rep2
$ cd ..
$ rmdir rep
$ ls -ld rep
ls: rep: No such file or directory
$
```

✎ *En fait, il aurait été possible de supprimer le répertoire rep via la seule commande **rmdir -p rep/rep2/rep3** tant que tous les répertoires mentionnés dans le chemin restent eux-mêmes vides.*

5. TP 3 – Nommages des fichiers et répertoires

Ce qui va être fait dans ce TP :

Ce TP présente la manipulation de répertoires et des fichiers qu'ils contiennent.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- afficher le nom du répertoire courant
- changer de répertoire
- utiliser plusieurs options de la commande **ls** pour afficher les propriétés de fichiers et de répertoires
- créer et supprimer des répertoires,
- créer des fichiers vides.

Remarque : Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

- ___ 1. Si ce n'est pas déjà fait, ouvrez une session
- ___ 2. Utilisez la commande **pwd**, et vérifiez que vous êtes dans votre répertoire d'accueil, **/home/stagexx**. C'est le dossier qui est ouvert lorsque vous vous connectez.
- ___ 3. Déplacez vous pour que le répertoire racine (**/**) devienne votre répertoire courant.
- ___ 4. Vérifiez que vous vous trouvez dans le répertoire racine et affichez la liste simple puis détaillée des fichiers présents dans ce répertoire.
- ___ 5. Lancez la commande **ls** avec les options **-a** et **-R**. Quelle est l'utilité de

chacune des options ? _____ (Remarque: la commande **ls -R** génère un grand nombre de lignes. Pour interrompre cette commande saisissez **<ctrl-c>** au clavier).

- ___ 6. Retournez dans le répertoire d'accueil (**/home/stagexx**) et listez son contenu y compris les fichiers cachés.

- ___ 7. Toujours dans votre répertoire d'accueil, créez un répertoire nommé **monrep**. Puis affichez sous forme d'une liste longue les attributs des deux répertoires **/home/stagexx** et **/home/stagexx/monrep**. Combien de fichiers sont contenus pour chacun des deux dossiers ? _____

- ___ 8. Déplacez-vous dans le répertoire **monrep**. En utilisant la commande **touch** sans option, créez deux fichiers **monfic1** et **monfic2** (cf le manuel en ligne pour l'utilisation de la commande **touch**).

- ___ 9. Affichez sous forme d'une liste longue le contenu de votre dossier **monrep**. Quelle est la taille des deux fichiers **monfic1** et **monfic2** ? _____
Affichez de nouveau la liste longue de votre répertoire en affichant aussi le numéro d'i-node des fichiers. Le numéro d'i-node de **monfic1** : _____, de **monfic2** : _____.

- ___ 10. Retournez dans votre répertoire d'accueil et utilisez la commande **ls -R** pour afficher votre arborescence de répertoires.

- ___ 11. Utilisez la commande **rmdir** pour supprimer le répertoire **monrep**. Est-ce possible ? _____ Vous constatez que la commande **rmdir** ne permet pas de supprimer un répertoire NON vide. Pour ce faire vous pourrez utiliser une commande présentée dans le module suivant : **rm -r**.

Fin de l'exercice

6. Commandes essentielles de manipulation des fichiers

.a Visualiser le contenu de fichiers de texte (cat, pg, more, less)

La commande **cat** visualise le contenu des fichiers sans marquer de pause.

Les commandes **pg** , **more** ou **less** permettent une visualisation écran par écran en offrant diverses possibilités de défilement. Elles sont assez équivalentes en terme fonctionnel et l'utilisation de l'une ou l'autre dépend surtout des préférences de l'utilisateur.

📖 *Le développement parallèle de plusieurs familles de systèmes, lors des premières années de l'histoire d'Unix, explique parfois l'existence de diverses commandes assez équivalentes pour réaliser telle ou telle fonctionnalité*

.b La commande pg

Au bas de chaque écran, on obtient le caractère : (deux points) avec de nombreuses possibilités de défilement. Chaque action doit être validée par la touche *Entrée* du clavier. Les actions les plus usuelles sont :

+n	Avancée de <i>n</i> écrans
(le seul caractère <i>Entrée</i> provoque l'affichage de l'écran suivant.)	
-n	Remontée de <i>n</i> écrans
n	Affichage de l'écran <i>n</i>
\$	Affichage du dernier écran
/expr	Recherche d'une expression vers le bas
?expr	Recherche d'une expression vers le haut
!cde	Exécution d'une commande du système
h	Aide en ligne
q	Fin de la commande

La commande peut être lancée avec des options :

-c	Affichage par effacements successifs
-p%d	Affichage du numéro de page au lieu du caractère :
+n	Affichage à partir de la ligne de numéro <i>n</i>
+/expr	Affichage à partir de la première apparition d'une expression

.c La commande more

Au bas de chaque écran, on obtient le message *More* suivi du pourcentage de fichier déjà affiché. Il existe alors de nombreuses possibilités de défilement qui n'ont pas besoin d'être validées par la touche *Entrée* du clavier. Les actions les plus usuelles sont :

Entrée	Avancée d'une ligne
Espace	Avancée d'un écran
b	Remontée d'un écran
=	Affichage du numéro de la ligne courante
v	Appel de l'éditeur de texte <i>vi</i> positionné sur la ligne courante
/expr	Recherche d'une expression vers le bas
!cde	Exécution d'une commande du système
.	Répétition de la dernière commande
h	Aide en ligne
q ou Q	Fin de la commande

La commande peut être lancée avec des options :

-c	Affichage par effacements successifs (équivalent à page)
-----------	------------------------------------------------------------------

+n	Affichage à partir de la ligne de numéro <i>n</i>
+/expr	Affichage deux lignes avant l'apparition d'une expression

.d La commande less

La commande *less* est assez équivalente à la commande *more*. Elle se révèle plus performante pour de gros fichiers car elle ne lit pas l'intégralité d'un fichier avant d'en commencer l'affichage.

📖 *Lorsqu'elles sont utilisées en dernier membre d'un pipe, les commandes *pg*, *more* ou *less* sont très utiles pour permettre le contrôle du défilement des sorties d'autres commandes. Le concept de pipeline sera présenté dans le chapitre 4 : Processus et Mécanismes. Dans ce contexte, l'avantage de la commande *less* réside dans la possibilité de remonter dans l'affichage.*

\$ **cal 1959 | more**

```

                                1959
                                Jan
S  M Tu W Th F S
      1 2 3
4  5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

                                Feb
S  M Tu W Th F S
      1 2 3 4 5 6 7
8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

                                Mar
S  M Tu W Th F S
      1 2 3 4 5 6 7
8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

                                Apr
S  M Tu W Th F S
      1 2 3 4
5  6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

                                May
S  M Tu W Th F S
      1 2
3  4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

                                Jun
S  M Tu W Th F S
      1 2 3 4 5 6
7  8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

                                Jul
S  M Tu W Th F S
      1 2 3 4 5 6
7  8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

                                Aug
S  M Tu W Th F S
      1 2 3 4 5 6
7  8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

                                Sep
S  M Tu W Th F S
      1 2 3 4 5 6
7  8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

--More--

```

2.2. Copier des fichiers (cp)

La commande **cp** permet la duplication de fichiers. Deux syntaxes sont proposées pour copier respectivement un seul ou plusieurs fichiers. D'autre part, l'*option -r* permet de copier une sous-arborescence complète.

Syntaxe résumée

cp [options] source cible

ou

cp [options] fic1 fic2 fic3 ... repertoire

Dans la seconde syntaxe où le dernier argument est un nom de répertoire existant, tous les fichiers cités sont copiés dans ce répertoire en y conservant leur nom.

Quelques options usuelles

-f Forcer l'écrasement sans confirmation

Certains schémas de permission provoquent l'échec de la copie si le fichier cible existe déjà. Cette option permet de passer outre certaines protections.

-i Demande de confirmation avant écrasement

-p Conservation des caractéristiques des fichiers sources (dates, permissions ...)

Si le fichier cible est créé, ses droits d'accès sont le résultat d'un choix concernant les droits par défaut des nouveaux fichiers. De plus, sa date de création est naturellement la date système. Cette option permet d'associer au nouveau fichier les propriétaires, droits et dates du fichier source.

-r Copie récursive de répertoires

Duplication simple de fichier

```
$ cp zorro zorrobis
$ ls -l zorro*
-rw-r--r--  1 mike  mike   1738 sep 12 12:23  zorro
-rw-r--r--  1 mike  mike   1738 sep 12 12:27  zorrobis
$
```

L'**option -i** protège contre les écrasements accidentels.

```
$ ls -l toto
-rw-r-----  1 mike  mike    916 Sep 12 18:10  toto
$ cp -i toto zorro
cp: overwrite zorro (yes/no)? n
$
```

Si le dernier argument est un répertoire, le ou les fichiers précédents de la liste d'arguments sont copiés dans ce répertoire et ils y conservent leur nom.

```
$ ls -l toto
-rw-r-----  1 mike  mike    916 Sep 12 18:10  toto
$ mkdir rep
$ cp toto rep
$ ls -lR
.:
total 4
drwxr-x---  2 mike  mike    512 Sep 12 18:11  rep
-rw-r-----  1 mike  mike    916 Sep 12 18:10  toto
./rep:
total 2
-rw-r-----  1 mike  mike    916 Sep 12 18:11  toto
$
```

L'**option -r** doit être suivie de deux noms de répertoires. Elle permet la copie de sous-arborescences complètes. Si le répertoire cible n'existe pas encore, il sera créé et contiendra une copie complète du contenu du répertoire source. Par contre, si le répertoire cible existe déjà, il contiendra une copie du premier répertoire lui-même et de son contenu.

📌 *Il faut faire attention à ne pas déclencher une commande récursive sans fin telle que `cp -r source source`.*

.f Gérer les liens (ln)

.g Liens physiques

Un même fichier physique peut être désigné par plusieurs noms externes que l'on appelle des **liens** (ou *liens physiques* ou encore *liens matériels*). Ceci peut notamment servir à éviter une copie physique si on souhaite désigner un même fichier à plusieurs endroits de l'arborescence. Un autre usage peut consister à désigner un même programme sous différents noms afin d'obtenir un paramétrage différent, dès le lancement, sans recourir à des options.

Les liens désignent un seul fichier physique représenté via un seul *inode*. Ils appartiennent nécessairement au **même filesystem** puisque chaque filesystem possède sa propre table des inodes. De plus, pour des raisons évidentes de cohérence de la structure d'arborescence, il n'est pas possible de créer explicitement un lien sur un répertoire. Un répertoire vide comporte cependant, dès sa création, deux liens. Il s'agit, d'une part, de son nom en clair et, d'autre part, de la référence `.` (point) qui permet de désigner ce répertoire si nous sommes positionnés dedans. Par la suite, son nombre de liens s'incrémentera à la création de chaque sous-répertoire car celui-ci comportera en effet la référence `..` (deux points) pour désigner le répertoire parent.

La commande **ls** affiche le nombre de liens entre les permissions et le nom de propriétaire. Dans l'exemple ci-dessous, la création d'un sous-répertoire *rep2* provoque l'incrémentation du compteur de liens du répertoire *rep*. En effet, la référence `..` créée dans *rep2* fournit maintenant une nouvelle façon de désigner le répertoire *rep*.

```
$ ls -ld rep
drwxr-xr-x  2 mike  mike   512 Sep 20 10:50 rep
$ mkdir rep/rep2
$ ls -ld rep
drwxr-xr-x  3 mike  mike   512 Sep 20 11:09 rep
$
```

📖 *Un répertoire qui ne contient pas de sous-répertoires possède donc exactement deux liens. Par la suite, le nombre de liens d'un répertoire Unix nous indique immédiatement le nombre de sous-répertoires de premier niveau qu'il contient. Il suffit de retrancher deux à cette valeur.*

.h Liens symboliques

Le lien symbolique poursuit le même objectif que le lien physique traditionnel. Il s'agit cependant d'un type de fichier. Il a une existence physique et le système lui attribue son propre *inode*. Le contenu d'un lien symbolique correspond à un chemin d'accès vers le fichier lié.

Les liens symboliques lèvent les limitations des liens physiques. Ils peuvent, en effet, désigner **des fichiers ou des répertoires situés dans un autre filesystem**.

Ces liens sont apparus plus récemment. Ils ont été imaginés principalement pour gérer la compatibilité ascendante lors des évolutions de l'arborescence Unix. Lorsque des fichiers ou des répertoires changent de nom ou d'emplacement, l'ancien nom devient un lien symbolique vers le nouveau fichier. Ceci rend le changement transparent pour les utilisateurs et les programmes.

Dans la pratique quotidienne, le lien symbolique est sans doute préférable aujourd'hui au lien traditionnel puisqu'il offre plus de souplesse. Le fait qu'il consomme un inode est négligeable par rapport aux avantages fournis.

Quelques inconvénients sont tout de même à signaler. À la création d'un lien symbolique, l'existence du fichier lié n'est pas vérifiée. D'autre part, l'utilisation de ces liens doit être bien maîtrisée lors des suppressions de fichiers ou encore dans les activités de sauvegarde.

.i La commande ln

La commande **ln** permet la création des liens (physiques ou symboliques via l'*option -s*). Deux syntaxes sont proposées pour lier respectivement un seul ou plusieurs fichiers.

Syntaxe résumée

ln [-s] actuel nouveau

ou

ln lien1 lien2 lien3 ... repertoire

Dans la seconde syntaxe où le dernier argument est un nom de répertoire existant, tous les fichiers cités sont liés dans ce répertoire en y conservant leur nom.

Dans ce premier exemple, nous créons un lien *toto2* sur le fichier existant *toto*. Le compteur de liens devient égal à 2. La visualisation du numéro d'inode (option *-i* de la commande *ls*) nous permet de vérifier qu'il n'existe qu'un seul fichier physique auquel correspondent deux noms différents.

```
$ ls -il toto
150647 -rw-r--r-- 1 mike mike 30 Sep 20 12:22 toto
$ ln toto toto2
$ ls -il toto*
150647 -rw-r--r-- 2 mike mike 30 Sep 20 12:22 toto
150647 -rw-r--r-- 2 mike mike 30 Sep 20 12:22 toto2
$
```

Dans ce deuxième exemple, nous créons un lien symbolique *stoto* vers le fichier existant *toto*. La visualisation des numéros d'inodes nous permet de constater que le lien symbolique possède son propre numéro car il s'agit bien d'un fichier à part entière. Son contenu est le chemin d'accès vers *toto*. Sa taille en octets est bien de 4 caractères (longueur du nom *toto*).

```
$ ln -s toto stoto
$ ls -il *toto*
150648 lrwxrwxrwx 1 mike mike 4 Sep 20 12:23 stoto -> toto
150647 -rw-r--r-- 2 mike mike 30 Sep 20 12:22 toto
150647 -rw-r--r-- 2 mike mike 30 Sep 20 12:22 toto2
$
```

L'utilisation de liens symboliques vers des répertoires peut quelquefois être troublante par rapport à l'utilisation de la commande **pwd**. La commande interne du shell est celle qui est appelée implicitement. Elle affichera le nom du lien symbolique lui-même. Par contre, la commande externe */usr/bin/pwd* affichera le véritable emplacement.

```
$ ln -s /home lhome
$ cd lhome
$ pwd
/home/mike/lhome
$ /usr/bin/pwd
/home
$
```

.j Renommer ou déplacer des fichiers (mv)

La commande **mv** permet de renommer ou de déplacer des fichiers. Deux syntaxes sont proposées pour respectivement changer le nom d'un seul fichier ou déplacer plusieurs fichiers dans un autre répertoire.

Syntaxe résumée

`mv [options] actuel nouveau`

ou

`mv [options] fic1 fic2 fic3 ... repertoire`

Dans la seconde syntaxe où le dernier argument est un nom de répertoire existant, tous les fichiers cités sont déplacés dans ce répertoire en y conservant leur nom.

Par défaut, si le nouveau nom correspond à un fichier ordinaire existant, l'ancien contenu de ce fichier est perdu.

Deux options permettent de prendre en compte cette situation :

- | | |
|-----------|----------------------------------------------------------------|
| -i | Demande de confirmation avant écrasement d'un fichier existant |
| -f | Forcer l'écrasement sans confirmation |

Dans ce premier exemple, nous changeons le nom d'un fichier ordinaire. La visualisation du numéro d'inode nous permet de vérifier qu'il s'agit toujours du même fichier physique.

```
$ ls -il bernard
```

```
150634 -rw-r--r-- 1 mike  mike  139 Sep 20 14:21 bernardo
```

```
$ mv bernardo zorro
```

```
$ ls -il zorro
```

```
150634 -rw-r--r-- 1 mike  mike  139 Sep 20 14:21 zorro
```

```
$
```

Dans ce deuxième exemple, nous voulons montrer qu'il est possible de renommer un lien symbolique ainsi qu'un répertoire.

```
$ ls -il
150636 lrwxrwxrwx 1 mike mike 4 Sep 20 14:30 lzorro -> zorro
150635 drwxr-xr-x 2 mike mike 512 Sep 20 14:30 rep
150634 -rw-r--r-- 1 mike mike 139 Sep 20 14:30 zorro

$ mv rep repbis
$ mv lzorro szorro
$ ls -il
150635 drwxr-xr-x 2 mike mike 512 Sep 20 14:30 repbis
150636 lrwxrwxrwx 1 mike mike 4 Sep 20 14:30 szorro -> zorro
150634 -rw-r--r-- 1 mike mike 139 Sep 20 14:30 zorro

$
```

Si nous renommons un fichier lié par un lien symbolique, le lien lui-même n'est pas modifié et ceci peut nous conduire à une incohérence dans l'arborescence.

```
$ mv zorro toto
$ ls -il
150635 drwxr-xr-x 2 mike mike 512 Sep 20 14:30 repbis
150636 lrwxrwxrwx 1 mike mike 4 Sep 20 14:30 szorro -> zorro
150634 -rw-r--r-- 1 mike mike 139 Sep 20 14:30 toto

$ cat szorro

cat: cannot open szorro

$
```

Dans ce nouvel exemple, nous déplaçons un fichier ordinaire dans un répertoire, en conservant son nom.

```
$ mv toto repbis
$ ls -il repbis
150634 -rw-r--r-- 1 mike mike 139 Sep 20 14:30 toto

$
```

.k Supprimer des fichiers (rm)

La commande **rm** permet de supprimer des fichiers. Lorsqu'il ne reste plus de lien sur le fichier concerné, celui-ci est supprimé physiquement. En dehors d'applications graphiques (gestionnaire de fichiers avec notion de *corbeille*), la suppression physique d'un fichier est irréversible.

La suppression d'un lien symbolique ne supprime pas le fichier pointé par ce lien. La suppression d'un fichier n'entraîne pas non plus la suppression des éventuels liens symboliques vers ce fichier. Ce dernier cas de figure peut générer des incohérences dans l'arborescence.

Quelques options

-f	Forcer la suppression sans confirmation
-i	Demande de confirmation avant suppression
-r	Suppression de répertoires (avec tout leur contenu)

Dans ce premier exemple, nous supprimons un lien sur un fichier ordinaire. Le compteur de liens passe de 2 à 1 et le fichier physique concerné n'est pas encore réellement supprimé.

```
$ ls -il toto*
150634 -rw-r--r--  2 mike  mike    139 Sep 20 14:57 toto
150634 -rw-r--r--  2 mike  mike    139 Sep 20 14:57 toto2
$ rm toto2
$ ls -il toto*
150634 -rw-r--r--  1 mike  mike    139 Sep 20 14:57 toto
$
```

Dans ce deuxième exemple, nous supprimons un fichier ordinaire sur lequel existe un lien symbolique. Le lien symbolique n'est pas supprimé mais il ne pointe plus sur un fichier valide.

```
$ ls -il *toto
150635 lrwxrwxrwx 1 mike  mike    4 Sep 20 15:04 stoto -> toto
150634 -rw-r--r-- 1 mike  mike   139 Sep 20 14:57 toto

$ rm toto

$ ls -il *toto
150635 lrwxrwxrwx 1 mike  mike    4 Sep 20 15:04 stoto -> toto

$ cat stoto
cat: cannot open stoto

$
```

Ce dernier exemple nous montre tout l'intérêt de l'option **-r** qui permet la suppression d'un répertoire et de tout son contenu. L'existence de cette option limite grandement l'usage de la commande **rmdir** qui ne peut supprimer que des répertoires vides.

```
$ ls -l rep
-rw-r--r--  1 mike  mike    30 Sep 20 15:11 fic1
drwxr-xr-x  2 mike  mike   512 Sep 20 15:11 repbis

$ rm -r rep

$ ls -l rep
rep: No such file or directory

$
```

.l D'autres commandes utiles

.m La commande head

La commande **head** permet d'extraire les premières lignes d'un fichier. Le nombre de lignes souhaité peut être fourni en option (précédé ou non du caractère *n*). Sans option, la commande affiche les 10 premières lignes.

```
$ head -4 /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
sys:x:3:3:/:

$
```

.n La commande tail

La commande **tail** permet d'extraire les dernières lignes d'un fichier. Le nombre de lignes souhaité peut être fourni en option. Sans option, la commande affiche les 10 dernières lignes. En outre, si l'option commence par le signe + (signe plus , et avec -n pour Linux), la commande affiche le fichier à partir d'un numéro de ligne donné.

```
$ tail -4 /etc/passwd
stage5:x:5005:5000::/home/stage5:/usr/bin/ksh
stage6:x:5006:5000::/home/stage6:/usr/bin/ksh
stage7:x:5007:5000::/home/stage7:/usr/bin/ksh
stage8:x:5008:5000::/home/stage8:/usr/bin/ksh
$ tail -n +4 /etc/passwd
sys:x:3:3::/
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
.....
.....
```

.o La commande wc

La commande **wc** fournit respectivement le nombre de lignes, de mots et de caractères qui constituent un fichier donné. Elle dispose d'options pour n'afficher que certaines de ces informations :

-l	nombre de lignes
-w	nombre de mots
-c	nombre de caractères

Exemples

```
$ wc /etc/passwd

 23      35    916 /etc/passwd

$ wc -l /etc/passwd

 23 /etc/passwd
```



```
$ wc -w /etc/passwd
```

```
35 /etc/passwd
```

```
$ wc -c /etc/passwd
```

```
916 /etc/passwd
```

```
$
```

Si plusieurs noms de fichiers sont fournis en argument, la commande effectue la somme de chacune des valeurs.

```
$ wc zorro*
```

```
1      6      30 zorro
41     469    1757 zorrobis
42     475    1787 total
```

```
$
```

2775cbe5ce
Global Knowledge

7. TP 4 – Manipulations de fichiers

Ce qui est fait :

Dans cet exercice vous utilisez des commandes Unix pour manipuler des fichiers ordinaires et des dossiers.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- copier, déplacer, renommer, créer un lien et supprimer des fichiers,
- afficher le contenu d'un fichier.

Remarque : Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

.a vérifiez votre environnement

- ___ 1. Si ce n'est pas déjà fait, ouvrez une session
- ___ 2. Utilisez la commande **pwd**, et vérifiez que vous êtes dans votre répertoire d'accueil, **/home/stagexx**. C'est le dossier qui est ouvert lorsque vous vous connectez.
- ___ 3. Listez le contenu de votre répertoire d'accueil (**/home/stagexx**), y compris les fichiers cachés.

.b manipulations de fichiers

- ___ 4. Utilisez successivement les commandes **cat**, **less** et **more** pour afficher le contenu des fichiers **/etc/hosts** et **/etc/profile**. Remarquez les différences de comportement.

- ___ 5. Copiez le fichier **/bin/cat** dans votre dossier courant (répertoire d'accueil)
- ___ 6. Copiez aussi le fichier **/usr/bin/cal** dans votre dossier courant (répertoire d'accueil)
- ___ 7. Affichez le contenu de votre répertoire courant. Vous devez voir les deux fichiers que vous venez de copier.

.c création et manipulation de répertoires

- ___ 8. Dans votre répertoire d'accueil, créez un sous répertoire nommez le **mescmds**.
- ___ 9. Déplacez y et renommez les deux fichiers que vous avez copiés dans votre répertoire d'accueil (**cat** et **cal**) vers **moncat** et **moncal**.
- ___ 10. Déplacez vous dans le répertoire **mescmds**.
- ___ 11. Listez le contenu du répertoire courant pour vérifier que les fichiers ont bien été déplacés.
- ___ 12. Utilisez la commande **./moncat** de votre répertoire **mescmds** pour afficher le contenu du fichier **.bash_profile** situé dans votre répertoire d'accueil.
- ___ 13. Retournez dans votre répertoire d'accueil.
- ___ 14. Dans ce répertoire d'accueil, créez un autre répertoire nommé **dubon**.
- ___ 15. Copiez le fichier **/etc/profile** dans le nouveau répertoire, et nommez la copie **nveauprofile**.
- ___ 16. Toujours depuis votre répertoire d'accueil, utilisez la commande **cat** pour

afficher la copie **nveauprofile**. Difficile de lire le début ? Utilisez **less** (ou more) à la place de **cat**.

- ___ 17. Le nom du fichier **nveauprofile** est trop long. Renommez le en **np**. Affichez la liste des fichiers du répertoire **dubon** pour vérifier que le changement de nom est effectif. Utilisez la commande **cat** pour afficher le contenu du fichier renommé.

- ___ 18. C'est le moment d'avoir une vision d'ensemble : en partant de votre répertoire d'accueil, affichez de façon hiérarchique, tous vos fichiers et sous-répertoires.

.d suppression de répertoires

- ___ 19. Assurez-vous d'être dans votre répertoire d'accueil. Essayez de supprimer votre répertoire **dubon**. Pouvez vous le supprimer ? Pourquoi ?

- ___ 20. Déplacez-vous dans le répertoire **dubon**. Listez son contenu y compris les fichiers cachés. Supprimez les fichiers. Listez de nouveau le contenu du répertoire courant. Vous remarquez que **.** et **..** sont toujours présents. Le répertoire est considéré comme vide si il ne contient que ces deux entrées. Supprimez le répertoire.

Fin de l'exercice

8. Droits d'accès

.a Sémantique des permissions de base

Un fichier Unix, quel que soit son type, possède **9 permissions de base**.

En effet, il faut considérer, vis à vis d'un fichier, trois catégories d'utilisateurs qui sont respectivement le **propriétaire**, le **groupe** et les **autres** (comptes n'appartenant pas au groupe).

Trois permissions seront éventuellement accordées à chacune de ces trois catégories. Il s'agit de la **lecture** (r), de l'**écriture** (w) et de l'**exécution** (x).

Ces permissions sont visualisées via l'*option -l* de la commande **ls**.

Exemple

```
$ ls -l fic
-rwxr-x--- 1 mike gkn 3053 Dec 13 14:43 fic
$
```

Nous constatons, dans cet affichage, que le fichier ordinaire *fic* appartient au compte *mike* et au groupe *gkn*. Son propriétaire possède les droits de lecture, d'écriture et d'exécution. Les membres du groupe *gkn* possèdent les droits de lecture et d'exécution, mais pas celui d'écriture. Les comptes n'appartenant pas au groupe *gkn* ne possèdent aucun droit.

La sémantique des permissions de base dépend du type de fichier.

En ce qui concerne les **liens symboliques**, nous pouvons dire que les permissions ne sont pas significatives. En effet, si nous positionnons des droits sur un lien symbolique, ces droits seront, en fait, attribués au fichier lié. Par tradition, la *commande ls* fait apparaître, pour un lien symbolique, un masque virtuel avec des permissions complètes :

```
$ ls -l *toto
lrwxrwxrwx 1 mike gkn 4 Sep 20 15:59 ltoto -> toto
-rw-r--r-- 1 mike gkn 1727 Sep 20 15:59 toto
$
```

Les **fichiers spéciaux** représentent, quant à eux, des noms de périphériques. La lecture et l'écriture ont leur sémantique naturelle tandis que le droit d'exécution n'est pas significatif.

En tant qu'utilisateur du système, nous sommes surtout concernés par les droits d'accès associés aux fichiers ordinaires et aux répertoires.

La sémantique des permissions pour un **fichier ordinaire** peut se résumer ainsi :

r (lecture)	Nous pouvons consulter ou copier le fichier.
w (écriture)	Nous pouvons modifier le contenu du fichier. La perte du contenu actuel d'un fichier (écrasement) est assimilée à une modification.
x (exécution)	Le fichier est considéré comme une commande. L'interpréteur de commandes essaiera de l'exécuter. Si ce fichier est un programme compilé (<i>exécutable C</i> par exemple), le droit de lecture n'est pas nécessaire à son exécution. Si le programme doit être interprété (<i>script shell</i> par exemple), le droit de lecture devra être associé au droit d'exécution.

La sémantique des permissions pour un **répertoire** peut se résumer ainsi :

r (lecture)	Nous pouvons obtenir la liste des fichiers du répertoire via la commande <i>ls</i> .
w (écriture)	Nous pouvons créer ou supprimer des fichiers dans le répertoire. Il faut noter qu'un fichier ordinaire, protégé individuellement en écriture, n'est pas modifiable mais que sa suppression est possible s'il se trouve dans un répertoire pour lequel nous disposons du droit d'écriture.

x (exécution)

Nous pouvons accéder aux fichiers de ce répertoire. Ce droit d'exécution est fondamental. En son absence, le répertoire est complètement inaccessible pour l'utilisateur concerné et, de fait, toute la sous-arborescence est également interdite. Si le droit d'exécution n'est pas accompagné du droit de lecture, les fichiers sont malgré tout accessibles dans la mesure où nous connaissons leur nom. En effet, l'absence de permission de lecture nous empêche seulement de visualiser la liste.

.b Permissions supplémentaires

Quelques **permissions supplémentaires** peuvent être associées à certains types de fichiers.

📖 *En interne, 12 bits de l'inode (et non pas seulement 9) sont utilisés pour représenter les permissions. Cependant, la commande `ls` ne visualise les droits que sur 9 positions grâce à des conventions de représentation.*

Certains droits complémentaires peuvent concerner les **commandes** (fichiers ordinaires exécutables) :

- bit SUID

Par défaut, une commande s'exécute avec l'identité du compte sous lequel nous sommes connectés. Si cette permission dite *SUID* est associée à la commande, nous prendrons plutôt l'identité du propriétaire de cette commande. Ce propriétaire sera, la plupart du temps, l'administrateur (compte *root* disposant des pleins pouvoirs). Par ce moyen, il est possible de disposer d'une commande publique capable d'effectuer des tâches administratives privilégiées, sans poser de problème de sécurité. La présence de cette permission est visualisée par la **lettre s** en lieu et place de la *lettre x* dans la partie *propriétaire*.

- bit SGID

Si cette permission dite *SGID* est associée à la commande, nous changerons également d'identité en terme de groupe. La présence de cette permission est visualisée par la **lettre s** en lieu et place de la *lettre x* dans la partie *groupe*.

```
$ ls -l /usr/bin/passwd
-r-sr-sr-x  1  root  sys   22168   Nov  4  2002   /usr/bin/passwd
$
```

Dans cet exemple, nous constatons que les bits *SUID* et *SGID* sont associés à la commande *passwd*. Cela signifie que tout utilisateur qui fait appel à cette commande est considéré comme ayant l'identité *root* et comme faisant partie du groupe *sys*. L'identité *root* est nécessaire et suffisante au bon fonctionnement de la commande. En effet, celle-ci doit stocker le cryptage du mot de passe dans des fichiers système protégés. Si nous avions gardé notre identité d'utilisateur du système, cette opération n'aurait pas été possible. Le fait d'être également intégré ponctuellement au groupe *sys* constitue un choix supplémentaire d'implémentation de la version concernée (Solaris dans cet exemple).

Certains autres droits complémentaires peuvent concerner les **répertoires** :

- bit SGID sur répertoire ou Inheritance flag

Cette permission complémentaire est utile si nous souhaitons que tous les fichiers créés dans le répertoire prennent comme groupe propriétaire celui du répertoire. Cette permission est, de plus, héritée par les sous-répertoires. Ceci permet d'obtenir une arborescence où tous les fichiers sont placés dans le même groupe quels que soient les propriétaires qui créent ces fichiers. La présence de cette permission est visualisée par la **lettre s** en lieu et place de la *lettre x* dans la partie *groupe*.

```
$ id
uid=2000(mike) gid=2000(mike)
$ ls -ld rep
drwxr-sr-x  2  mike  gkn          512 Sep 20 17:00  rep
$ cp /etc/passwd rep/toto
$ mkdir rep/repbis
$ ls -al rep
total 8
drwxr-sr-x  3  mike  gkn          512 Sep 20 17:02  .
drwxr-xr-x  3  mike  mike          512 Sep 20 17:00  ..
drwxr-sr-x  2  mike  gkn          512 Sep 20 17:02  repbis
-r--r--r--  1  mike  gkn          916 Sep 20 17:02  toto
$
```


Dans cet exemple, nous sommes connectés sous l'identité *mike* faisant partie d'un groupe de même nom. Si nous créons des fichiers, ceux-ci devraient tout naturellement être attribués à ce compte et à ce groupe. Le répertoire *rep* appartient au compte *mike* et au groupe *gkn*. De plus, il est doté de la permission dite *Inheritance flag*. De ce fait, quand nous créons un fichier et un sous-répertoire dans ce répertoire *rep*, ils sont attribués au groupe *gkn*. De plus, le sous-répertoire *repbis* se retrouve doté de la permission complémentaire pour propager ce comportement dans la sous-arborescence.

- Sticky Bit

Cette permission complète et affine le droit d'écriture pour un répertoire. Elle permet d'indiquer le fait que la suppression d'un fichier de ce répertoire sera réservée au seul propriétaire du fichier. La présence de cette permission est visualisée par la **lettre t** en lieu et place de la *lettre x* dans la partie *autres*.

```
$ ls -ld /tmp
drwxrwxrwt  5 root      sys      227 Sep 20 17:15 /tmp

$ id
uid=2000(mike) gid=2000(mike)

$ ls -l /tmp/t*
-rw-rw-rw-  1 mike      mike      1727 Sep 20 17:15 /tmp/toto
-rw-rw-rw-  1 root      other      30 Sep 20 17:15 /tmp/tutu

$ rm /tmp/toto

$ rm /tmp/tutu

rm: /tmp/tutu not removed: Permission denied

$
```

Le répertoire */tmp* est un répertoire public où tous les utilisateurs et les programmes peuvent créer des fichiers de travail (fichiers temporaires). En effet, nous constatons que le droit d'écriture est présent dans la partie *autres*. En l'absence de la permission complémentaire dite *sticky bit*, ce même droit d'écriture nous donnerait aussi la permission de supprimer n'importe quel fichier, y compris ceux des autres utilisateurs. Nous pouvons vérifier, dans cet exemple, que l'utilisateur *mike* ne peut supprimer que le fichier dont il est propriétaire.

📖 *Le sticky bit existe aussi pour les commandes (fichiers ordinaires exécutables). Il s'agit d'une fonctionnalité historique, obsolète aujourd'hui, qui indiquait la volonté de conserver le fichier en zone de swap pour obtenir un chargement plus rapide. Ce droit d'accès complémentaire est aujourd'hui réutilisé, avec une toute autre sémantique, pour les répertoires..*

.c Choisir les droits par défaut en création de fichiers (umask)

Au sein de l'arborescence Unix, l'utilisateur est responsable de la protection de ses propres fichiers. Pour faciliter cette gestion, il dispose de la commande **umask** qui lui permet d'indiquer quels seront les droits associés aux nouveaux fichiers et répertoires lors de leur création.

Pour manipuler cette commande, nous devons savoir associer aux 9 permissions de base, une **valeur octale** sur 3 chiffres. La lecture (r) prend la valeur **4**, l'écriture (w) prend la valeur **2** et l'exécution (x) prend la valeur **1**.

Le choix historique des concepteurs d'Unix a été d'associer des droits très permissifs aux nouveaux fichiers et répertoires, à savoir :

rw- rw- rw- pour les fichiers ordinaires (**666** en notation octale)

rxw rxw rxw pour les répertoires (**777** en notation octale)

La commande *umask* reçoit en argument une valeur octale qui correspond à ce que nous souhaitons enlever aux masques de droits par défaut. Il nous faudra choisir une valeur commune pour les deux types de fichiers (fichiers ordinaires et répertoires).

Quelques choix de valeurs réalistes pour cette commande :

umask 022	Fichiers ordinaires en	<i>rw- r-- r--</i> (644)
	Répertoires en	<i>rxw r-x r-x</i> (755)
umask 002	Fichiers ordinaires en	<i>rw- rw- r--</i> (664)
	Répertoires en	<i>rxw rxw r-x</i> (775)
umask 027	Fichiers ordinaires en	<i>rw- r-- ---</i> (640)
	Répertoires en	<i>rxw r-x ---</i> (750)
umask 007	Fichiers ordinaires en	<i>rw- rw- ---</i> (660)
	Répertoires en	<i>rxw rxw ---</i> (770)
umask 077	Fichiers ordinaires en	<i>rw- --- ---</i> (600)
	Répertoires en	<i>rxw --- ---</i> (700)

```

$ umask
022
$ touch f1
$ mkdir r1
$ umask 077
$ touch f2
$ mkdir r2
$ umask 0
$ touch f3
$ mkdir r3
$ ls -l
-rw-r--r--  1  mike  mike    0   Sep 20 18:01  f1
-rw-----  1  mike  mike    0   Sep 20 18:01  f2
-rw-rw-rw-  1  mike  mike    0   Sep 20 18:01  f3
drwxr-xr-x  2  mike  mike  512   Sep 20 18:01  r1
drwx-----  2  mike  mike  512   Sep 20 18:01  r2
drwxrwxrwx  2  mike  mike  512   Sep 20 18:01  r3
$

```

Dans cet exemple, nous découvrons la commande **touch** qui est destinée à changer la date système d'un fichier mais qui a pour effet complémentaire de le créer (de taille 0). Invoquée sans argument, la commande **umask** affiche la valeur courante du masque.

Avec la valeur **022**, le fichier **f1** et le répertoire **r1** prennent respectivement les masques **644** et **755**. Après affectation de la valeur **077**, le fichier **f2** et le répertoire **r2** prennent les masques **600** et **700**. Enfin, après affectation de la valeur **0**, le fichier **f3** et le répertoire **r3** prennent les masques **666** et **777** qui correspondent aux droits historiques non modifiés.

📖 Une valeur générale de umask est souvent positionnée par l'administrateur système et l'utilisateur peut ensuite faire son choix personnel. La valeur choisie est associée au shell courant. Elle disparaît donc avec la déconnexion. L'utilisateur devra rendre ce choix permanent via un appel explicite à la commande dans le fichier de connexion, lu par le shell lors de son démarrage. Nous préciserons ces aspects dans le chapitre 5 consacré à l'utilisation du shell.

.d Modifier les droits des fichiers existants (chmod)

La commande **chmod** permet de gérer les permissions des fichiers dont on est propriétaire. Le nouveau masque de permissions peut être donné soit sous forme octale, soit sous forme symbolique. La forme symbolique est bien adaptée lorsqu'il s'agit d'ajouter ou d'enlever des droits sans avoir à consulter les droits existants.

📖 *Le compte root dispose de tous les pouvoirs sur le système. Il peut notamment utiliser la commande chmod sur n'importe quel fichier. Il dispose aussi de deux commandes chown et chgrp qui permettent respectivement de changer le propriétaire et le groupe d'un fichier. Pour des raisons évidentes de sécurité, ces deux commandes ne sont pas accessibles aux utilisateurs du système.*

Syntaxe résumée

chmod [-R] mode_octal fichiers ...

ou

chmod [-R] mode_symbolique fichiers...

-R

Traitement récursif sur les répertoires

mode_octal

Valeur en octal du masque de permission

mode_symbolique

Combinaison : *qui opération droits*

[ugoa] +|-|= [rwxstugo]

qui

u propriétaire

g groupe

o autres

a tous

opération	+	ajout
	-	suppression
	=	affectation
droits	r	lecture
	w	écriture
	x	exécution
	s	bits SUID ou SGID
	t	sticky bit
	u	permissions de l'utilisateur
	g	permissions du groupe
fichiers...	o	permissions des autres
	Fichier(s) concerné(s)	

```

$ chmod 666 f1
$ ls -l f1
-rw-rw-rw- 1 mike mike 0 Sep 20 18:01 f1
$ chmod go-w f1
$ ls -l f1
-rw-r--r-- 1 mike mike 0 Sep 20 18:01 f1
$ chmod g+w,o-r f1
$ ls -l f1
-rw-rw---- 1 mike mike 0 Sep 20 18:01 f1
$

```

Dans ce premier exemple, nous faisons usage successivement d'un masque en octal (666) puis de deux masques en notation symbolique pour, dans un premier temps enlever le droit d'écriture au groupe et aux *autres* puis, dans un second temps, redonner le droit d'écriture au groupe et enlever le droit de lecture aux *autres*.

```
$ ls -ld r1
drwxr-xr-x  2  mike  mike  512  Sep 20 18:01  r1
$ chmod g+w,u+t r1
$ ls -ld r1
drwxrwxr-t  2  mike  mike  512  Sep 20 18:01  r1
$
```

Dans ce deuxième exemple, nous ajoutons, sur le répertoire *r1*, le droit d'écriture pour le groupe et nous positionnons le droit complémentaire *sticky bit*. Par la suite, dans ce répertoire, les membres du groupe auront la possibilité de créer des fichiers mais ils ne pourront supprimer que ceux dont ils sont propriétaires.

📖 *Les permissions supplémentaires (bits SUID, SGID et sticky bit) correspondent bien entendu à des valeurs octales qui nécessitent un quatrième chiffre en début de masque (exemple 1755 pour obtenir rwx rwx r-t). Des conventions de notation qui s'inspirent du mode de visualisation de la commande ls sont néanmoins disponibles, à savoir u+s pour le bit SUID, g+s pour le bit SGID et, curieusement, u+t pour le sticky bit. Certaines versions permettent aussi la notation o+t pour ce même sticky bit. Dans tous les cas, le bon réflexe est de consulter la documentation en ligne via la commande man.*

```
$ chmod 400 f1
$ ls -l f1
-r-----  1  mike  mike    0  Sep 20 18:01  f1
$ ls -ld
drwxr-xr-x  5  mike  mike  512  Sep 20 18:01  .
$ rm f1
rm: f1: override protection 400 (yes/no)? y
$ ls -l f1
ls: No such file or directory
$
```

Dans ce troisième exemple, nous avons pu supprimer le fichier *f1* protégé en écriture car il se trouve dans un répertoire lui-même accessible en écriture et permettant donc la suppression des fichiers. L'**option -f** de la commande *rm* aurait évité la demande de confirmation.

.e ACLs (Access Control List)

Afin d'améliorer le schéma de permissions standards, la plupart des versions Unix implémentent des permissions étendues qui permettent d'associer des droits particuliers à un utilisateur ou à un groupe vis à vis d'un fichier. Ces droits étendus, baptisés **ACLs** (Access Control Lists) peuvent compléter ou remplacer les permissions de base.

Malheureusement, l'implémentation de ces *ACLs* n'est pas normalisée et le jeu de commandes et de fonctionnalités varie selon les versions. Il conviendra d'étudier les commandes dans la documentation du système pour en maîtriser la mise en œuvre.

Le jeu de commandes sur les trois versions majeures est le suivant :

AIX	aclget, aclput, acledit
HP-UX	lsacl, chacl
Solaris	getfacl, setfacl

9. Récapitulatif des commandes à approfondir dans la documentation

cat	Affichage ou concaténation de fichiers
cd	Changement de répertoire courant
chmod	Gestion des permissions d'un fichier
cp	Copie de fichiers
head	Affichage des premières lignes
less	Affichage page par page
ln	Création de liens
ls	Affichage du contenu d'un répertoire
mkdir	Création de répertoires
more	Affichage page par page
mv	Changement de nom ou déplacement de fichiers
pg	Affichage page par page
pwd	Nom du répertoire courant
rm	Suppression de fichiers
rmdir	Suppression de répertoires vides
tail	Affichage des dernières lignes d'un fichier
touch	Modification de la date d'un fichier avec création éventuelle
umask	Choix des droits par défaut
wc	Nombre de lignes, de mots et de caractères

10. TP 5 – Droits d'accès des fichiers

Ce qui est fait :

Cet exercice fait manipuler les droits d'accès des fichiers et des répertoires. Dans le but d'acquérir les notions fondamentales de propriétaire et de comprendre les restrictions d'accès aux fichiers.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- de manipuler les droits d'accès de fichiers ordinaires et de répertoires,
- d'interpréter le sens des bits de droits d'accès,
- afficher la liste longue des fichiers ordinaires et répertoires pour interpréter leurs droits d'accès.

Remarques : Assurez-vous d'être positionné dans le bon répertoire pour lancer les commandes. N'hésitez pas à utiliser de façon intensive la commande **pwd** pour vérifier que vous êtes bien dans le bon répertoire.

Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

.a Affichez les attributs des fichiers

- 1. Ouvrez une session utilisateur. Déplacez-vous dans le répertoire **mescmds**. Affichez la liste longue du répertoire. Notez le nom du propriétaire et les droits d'accès aux fichiers copiés lors de l'exercice précédent.

Propriétaire et permissions de **moncat** _____

Propriétaire et permissions de **moncal** _____

- ___ 2. Affichez les attributs des fichiers originaux **/bin/cat** et **/usr/bin/cal** et comparez avec les copies du répertoire **mescmds**. Vous êtes propriétaire des copies mais pas des originaux.
- ___ 3. Changez l'heure de modification de **moncal** et de **moncat** dans le dossier **mescmds**. Vérifiez que l'heure de modification a bien été changée. Quel est l'autre usage de la commande **touch** ? _____
- ___ 4. Faire en sorte de pouvoir référencer aussi bien par son nom de fichier : **moncal**, dans le répertoire **mescmds**, que par autre nom : **home_moncal**, dans votre répertoire d'accueil.

Y a-t-il une différence d'attributs entre les deux noms ? _____

Quel est le nombre de liens ? _____

- ___ 5. Déplacez-vous dans votre répertoire d'accueil et lancez **home_moncal**.

Y a-t-il une différence de comportement avec **/usr/bin/cal** ? _____

Maintenant changez les droits d'accès de **home_moncal** de façon que vous, le propriétaire du fichier, n'ayez plus que le droit de lecture. Lancez de nouveau la commande **moncal**.

Est-ce possible ? _____

Pourquoi ? _____

- ___ 6. Supprimez **home_moncal**. Est-ce que cela supprime **mescmds/moncal** ?

Pourquoi ? _____

.b Travaillez avec les droits d'accès des fichiers ordinaires

- ___ 7. Déplacez-vous dans le dossier **mescmds**. Utilisez la syntaxe symbolique de la commande **chmod** pour supprimer le droit de lecture pour le bit *other* du fichier **moncat**. Vérifiez que les droits ont été changés.

- ___ 8. En utilisant la syntaxe octale, changez les droits d'accès de **moncat** de façon que le propriétaire ait le droit de lecture uniquement, avec aucun droit pour tous les autres. Vérifiez que les droits ont été changés.

- ___ 9. Utilisez **moncat** pour afficher le contenu du fichier **.bash_profile**. Est-ce possible ?

- ___ 10. Déplacez-vous dans votre répertoire d'accueil. Vérifiez que vous y êtes bien positionné.

.c Travailler avec les droits d'accès aux répertoires

- ___ 11. Modifiez les droits du répertoire **mescmds** pour que vous n'ayez que le droit de lecture.

- ___ 12. Affichez la liste des attributs pour vérifier que les droits ont été correctement modifiés.

- ___ 13. Cherchez à afficher une liste simple du contenu du répertoire **mescmds**. Puis une liste détaillée. Est-ce que c'est possible ?

Pourquoi ?

- ___ 14. Cherchez à lancer **moncal**. Est-ce que c'est possible ?

Pourquoi ?

___ 15. Cherchez à supprimer **moncal** Est-ce que c'est possible ?

Pourquoi ?

___ 16. Redonnez les droits d'origine **rwxr-xr-x** au répertoire **mescmds**, puis supprimez **moncal**.

___ 17. Si le temps le permet, testez d'autres combinaisons de droits d'accès.
Lorsque vous avez fini, assurez-vous de repositionner les droits **rwx** pour vous même le propriétaire.

Fin de l'exercice

Chapitre 3

L'éditeur de texte vi

2775cbe5ce
Global Knowledge

L'éditeur plein écran *vi* (video) est présent, depuis très longtemps, sur toutes les versions Unix. Il constitue l'éditeur standard de fait du système. Il peut fonctionner sur n'importe quel type de terminal en se contentant éventuellement d'un clavier minimum (sans touches de fonction, ni flèches, ni pavé numérique).

L'utilisateur pourra, bien entendu, préférer d'autres éditeurs plus conviviaux, notamment en environnement graphique. Cependant, une connaissance minimale de *vi* s'avère nécessaire pour être capable de créer ou modifier un fichier de texte quel que soit l'environnement de travail.

De plus, la maîtrise des commandes essentielles de l'éditeur permet d'être plus efficace pour exploiter l'historique des commandes dans le contexte *Korn shell* (voir chapitre 5).

1. Les conventions à connaître

Le fichier donné en argument au lancement de l'éditeur est visualisé à l'écran. Si ce fichier n'existe pas encore, il sera créé lors de la première sauvegarde et nous sommes devant un écran vierge. Le point de référence est toujours la position du curseur.

Si aucun nom de fichier n'est donné en argument, son nom pourra être choisi lors de la sauvegarde.

Il est également possible de donner plusieurs noms de fichiers en paramètres. Des commandes de l'éditeur permettront de passer de l'un à l'autre mais, à chaque instant, nous ne pourrons visualiser qu'un seul fichier à l'écran.

Contrairement à la majorité des éditeurs de texte, *vi* présente plusieurs modes de fonctionnement distincts. Cette caractéristique est essentielle à maîtriser pour surmonter les éventuelles difficultés ou désagréments qui en découlent.

Les modes de fonctionnement sont les suivants :

Mode commande (ou mode *vi*)

Chaque touche ou combinaison de touches est directement interprétée et provoque l'exécution immédiate d'une commande de l'éditeur. Nous sommes dans ce *mode commande* lors de l'appel de l'éditeur.

Mode insertion

Certaines commandes nous font passer dans le *mode insertion* dans lequel aucune touche n'est interprétée afin de permettre la saisie ou la modification de texte. Pour insérer une ligne, il faudra notamment penser à taper la touche *Entrée*. Le retour en *mode commande* se fera par la touche **Echappement (Escape)** qui est incontournable pour une utilisation correcte de l'éditeur.

Mode commande syntaxique (ou mode ex)

Il s'agit d'un cas particulier du *mode commande*. En effet, les commandes qui commencent par le caractère : (deux points) apparaissent en écho sur la dernière ligne de l'écran. Elles doivent être validées, de façon plus naturelle, par le caractère *Entrée*.



Au départ, l'éditeur est donc en mode commande et il est nécessaire de tout d'abord placer l'éditeur en mode insertion à l'issu duquel il sera obligatoire de revenir en mode commande pour enchaîner une autre action et notamment un déplacement. Il est donc incontournable d'adopter rapidement le réflexe de la touche Echappement (Escape).

L'éditeur ne permet pas de se déplacer au-delà du texte existant. Nous devons, dans ce genre de situation, penser en terme d'insertion de texte et utiliser une commande appropriée (insertion en fin de ligne, insertion avant ou après la ligne courante...).

Sur certaines émulations pauvres, il sera impossible d'utiliser le pavé numérique du clavier, ni même les flèches. Dans les tableaux de commandes qui suivent, nous constaterons que toutes les actions de l'éditeur peuvent être obtenues depuis un clavier minimum, tel qu'il se présentait dans les années où l'éditeur a été conçu.

2. Les commandes essentielles

a. Déplacements

La notation n indique un nombre.

h ou nh	Se déplacer vers la gauche (une ou n positions)
j ou nj	Se déplacer vers le bas (une ou n positions)
+ ou $n+$	Synonyme de j
k ou nk	Se déplacer vers le haut (une ou n positions)
- ou $n-$	Synonyme de k
l ou nl	Se déplacer vers la droite (une ou n positions)
On peut aussi, la plupart du temps, utiliser les touches flèches.	
0 (zéro)	Se positionner en début de ligne
^	Se positionner sur le premier caractère significatif de la ligne
\$	Se positionner en fin de ligne
b	Se positionner sur le mot précédent
w	Se positionner sur le mot suivant
nG	Aller à la ligne n
1G	Aller en début de fichier
G	Aller en fin de fichier
Ctrl d	Descendre d'une moitié d'écran
Ctrl u	Remonter d'une moitié d'écran
Ctrl f	Descendre d'un écran
Ctrl b	Remonter d'un écran

H	Aller en haut de l'écran
M	Aller au milieu de l'écran
L	Aller en bas de l'écran
{	Aller au paragraphe précédent
}	Aller au paragraphe suivant
%	Trouver la prochaine parenthèse correspondante

b. Insertions, suppressions, modifications

La notation n indique un nombre.

Escape	Revenir au mode commande
a	Insérer après le caractère courant (terminer par <i>Escape</i>)
A	Insérer en fin de ligne (terminer par <i>Escape</i>)
i	Insérer avant le caractère courant (terminer par <i>Escape</i>)
I	Insérer en début de ligne (terminer par <i>Escape</i>)
o	Insérer après la ligne courante (terminer par <i>Escape</i>)
O	Insérer avant la ligne courante (terminer par <i>Escape</i>)
x	Supprimer le caractère courant
s	Supprimer le caractère courant et passer en mode insertion
X	Supprimer le caractère précédent
D	Supprimer la fin de la ligne
dd ou ndd	Supprimer une ou n lignes

dw ou ndw	Supprimer un ou <i>n</i> mots
dG	Supprimer jusqu'à la fin de fichier
r	Remplacer le caractère courant par un autre caractère
cw	Remplacer le mot courant (terminer par <i>Escape</i>)
C	Remplacer la fin de ligne (terminer par <i>Escape</i>)
R	Se placer en mode <i>sur-impression</i> (terminer par <i>Escape</i>)
J	Regrouper la ligne courante avec la ligne suivante

c. Recherches d'expressions

La notation *n* correspond ici à la *lettre n*.

/expr	Rechercher une expression vers le bas
?expr	Rechercher une expression vers le haut
n	Chercher l'occurrence suivante de l'expression
N	Chercher l'occurrence précédente de l'expression

d. Substitutions répétitives

:s/expr1/expr2/	Sur la ligne courante, remplacer, une seule fois, la première expression par la seconde
:s/expr1/expr2/g	Sur la ligne courante, remplacer, plusieurs fois si nécessaire, la première expression par la seconde
:n,ms/expr1/expr2/	

:n,ms/expr1/expr2/g

Effectuer les remplacements entre les lignes n à m

☞ Dans ce contexte, la ligne courante se note . (point) et la dernière ligne se note \$ (dollar).

e. Duplications et déplacements de lignes (copier-coller et couper-coller)

La notation n indique un nombre.

Y

Mémoriser la ligne courante dans un tampon de travail

en vue d'un copier-coller

nY

Mémoriser n lignes à partir de la ligne courante

p (minuscule)

Insérer le tampon de travail après la position courante

P (majuscule)

Insérer le tampon de travail avant la position courante

☞ Les commandes de suppression, telles que notamment x et dd, mémorisent également dans le tampon de travail en vue d'un déplacement (couper-coller).

f. Autres commandes utiles

u

Annuler la dernière commande

. (point)

Répéter la dernière commande

!:commande

Exécuter une commande Unix et revenir dans l'éditeur

!!commande	Insérer le résultat d'une commande (la ligne courante est écrasée)
Ctrl g	Obtenir le numéro de la ligne courante
Ctrl l ou Ctrl r	Rafraîchir l'écran en cas de parasites d'affichage

g. Sorties et sauvegardes

:q ou :q!	Quitter l'éditeur sans sauvegarder le fichier courant (Le ! est nécessaire si des modifications ont eu lieu)
ZZ ou :wq ou :x	Quitter l'éditeur et sauvegarder le fichier courant
:x!	Forcer la sauvegarde (fichier en lecture seule)
:w	Sauvegarder sans sortir
:w <i>fic</i>	Sauvegarder sous le nom <i>fic</i>
:n1,n2w <i>fic</i>	Sauvegarder les lignes de numéros <i>n1</i> à <i>n2</i> dans le fichier de nom <i>fic</i>
:r <i>fic</i>	Insérer, après la ligne courante, le contenu du fichier <i>fic</i>

h. Édition de plusieurs fichiers

:e <i>fic</i>	Éditer le fichier <i>fic</i>
:e! <i>fic</i>	Éditer le fichier <i>fic</i> sans sauvegarder les modifications sur le fichier courant

:e#

Retour à l'édition du fichier précédent

Si l'appel de l'éditeur a lieu avec plusieurs noms de fichiers,

:n

Éditer le fichier suivant

:n!

Éditer le fichier suivant sans sauvegarder les modifications sur le fichier courant

i. Paramétrage de l'éditeur

:set all

Liste des options

Le détail des options s'obtient via la commande **man ex**.

\$ **man ex**

```
.....
autoindent    ai      supply indent
autowrite     aw      write before changing files
directory     pathname of directory for temporary work
                files
exrc          ex      allow vi/ex to read the .exrc in the
                current directory. This option is set in
                the EXINIT shell variable or in the .exrc
                file in the $HOMEdirectory.
ignorecase    ic      ignore case of letters in scanning
list          print ^I for tab, $ at end
magic         treat . [ * special in patterns
modelines     first five lines and last five lines
                executed as vi/ex commands if they are
                of the form ex:command: or vi:command:
number        nu      number lines
paragraphs    para    macro names that start paragraphs
redraw        simulate smart terminal
.....
```

:set option

Activer une option (exemple :set
autoindent)

:set nooption

Inhiber une option (exemple :set
nonumber)

`:set opt=val`

Donner une valeur de paramétrage
(exemple `:set tabstop=4`)

☞ On peut abréger le mot *set* en *se* ainsi que le nom de l'option s'il n'est pas ambigu (exemple `:se nonu`).

Nous pouvons constituer un fichier de nom **.exrc** et le placer dans notre répertoire de connexion pour obtenir un paramétrage permanent. En effet, ce fichier sera lu à chaque appel de l'éditeur.

Exemple de fichier

```
$ cat .exrc
set number
set autoindent
set tabstop=4
set showmode
$
```

Les deux premières lignes de ce fichier de paramétrage permettent de numéroter systématiquement les lignes à l'affichage (*set number*) et d'activer le mode auto-indentation (*set autoindent*). Dans ce mode, destiné surtout aux développeurs, la ligne suivante s'aligne automatiquement sur la ligne précédente. La touche **Ctrl d** permettra de revenir en début de ligne lorsque nous souhaiterons mettre fin à cet alignement. La troisième ligne (*set tabstop=4*) redéfinit la correspondance entre le caractère *tabulation* et le nombre d'espaces générés. Enfin, la quatrième ligne (*set showmode*) est destinée à mieux visualiser la différence entre le *mode commande* et le *mode insertion* en faisant apparaître une indication explicite en bas de l'écran lorsque nous nous trouvons en mode insertion.

3. L'éditeur vim des distributions Linux

Dans les distributions Linux, l'éditeur **vim** (VI Improved) est proposé en lieu et place de l'éditeur standard *vi*. Il s'agit d'un clone de l'éditeur standard mais il apporte quelques améliorations telles que la gestion de certains langages de programmation via la coloration syntaxique du code. Il offre, de plus, un historique complet des commandes pour des annulations ou répétitions plus performantes.

Les sources de cet utilitaire peuvent s'obtenir via le site officiel du produit **www.vim.org**.

2775cbe5ce
Global Knowledge

4. TP 6 – L'éditeur vi

Ce qui est fait dans ce TP :

Le but de cet exercice est de vous donner la possibilité de créer et modifier des fichiers en utilisant l'éditeur UNIX le plus courant : vi. Une bonne compréhension de l'éditeur vi est essentielle pour mener à bien le reste des exercices de ce stage.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- Créez un fichier avec vi
- Sauvegardez et quittez le fichier et quitter sans sauvegarder
- Manipuler un fichier en utilisant différentes touches de déplacement du curseur
- Ajouter, supprimer et faire des changements de texte dans un fichier
- Définir les options pour personnaliser la session d'édition
- Appelez le mode commande d'édition en ligne.

Introduction

L'éditeur **vi** est basé sur un logiciel développé par l'Université de Californie à Berkeley, division Informatique. L'éditeur **vi**, prononcez «vé-i » (abréviation de visuel), comporte des commandes pour créer, modifier, ajouter ou supprimer des fichiers. Les exercices suivants vous familiariseront avec quelques-unes des principales caractéristiques et des fonctions de **vi**.

.a Création d'un fichier

- ___ 1. Assurez vous d'être dans votre répertoire d'accueil. Avec **vi** créez un fichier nommé **vitest**.

»\$ cd

»\$ pwd

»\$ vi vitést

- 2. Lorsque vous lancez **vi** vous êtes en mode commande. Appuyez sur la touche **i** (insert) pour basculer en mode écriture. Vous pouvez aussi appuyer sur la touche **a** (ajout). L'utilisation de **i** ou de **a** fait que le début du texte saisi ensuite apparaît avant (insert) ou après (ajout) le curseur. Par défaut, il n'y a pas d'indication comme quoi vous avez basculé en mode écriture.

Basculez du mode écriture au mode commande en appuyant sur la touche <Echap> (Echappement) du clavier. Remarquez que si vous appuyez une deuxième fois sur la touche <Echap> alors votre matériel peut émettre un « bip » et / ou votre écran flasher. Cela indique que vous êtes déjà en mode commande et n'a pas d'autre conséquence. Appuyez de nouveau sur la touche **i** pour repasser en mode écriture, puis passez à l'étape suivante.

»i

»<Echap>

»<Echap> (vous entendez un “bip” et / ou un flash écran)

»i

- 3. Saisissez le texte ci-dessous *exactement* comme il est présenté. Ligne à ligne. Puis la liste des caractères de l'alphabet, un caractère par ligne. L'ajout de l'alphabet est un moyen facile de remplir plusieurs écrans de texte nécessaires à une utilisation ultérieure

This Ceci est une session de formation sur l'utilisation de l'editeur vi. On a besoin de plus de lignes pour apprendre les commandes les plus courantes de l'éditeur. On est maintenant dans le mode ecriture et nous allons passer tout de suite dans le mode commande.

**a
b
c
d
...
z**

- 4. Repassez en mode commande. Puis enregistrez et quittez **vi**. Remarquez que dès que vous appuyez sur la touche ":" (deux points), ces deux points apparaissent en bas de l'écran, en dessous de la dernière ligne saisie. Une fois le buffer vidé et le fichier fermé, vous verrez un message indiquant le nombre de lignes et de caractères enregistrés dans le fichier.

»<Echap> (retour en mode commande).

»:wq (ou bien <shift-zz>, ou bien ":x" qui est une autre façon d'enregistrer les modifications, puis de quitter).

.b Déplacement du curseur

- ___ 5. Ouvrez le fichier **vitest** avec **vi**. Remarquez sur la dernière ligne de l'écran l'affichage du nom du fichier et du nombre de caractères. Vous êtes en mode commande.

»\$ vi vitest

- ___ 6. Pour déplacer le curseur, vous pouvez utiliser les flèches du clavier ou les touches **h**, **j**, **k**, **l**. Déplacez le curseur d'une ligne vers le bas, de quelques caractères vers la droite, puis vers la gauche et remontez le curseur sur la ligne du dessus.

»j (une ligne vers le bas)

»k (une ligne vers le haut)

»l (un caractère à droite)

»h (un caractère à gauche)

»Pratiquez les mêmes déplacements avec les flèches du clavier.

- ___ 7. Plutôt que de déplacer le curseur ligne à ligne, ou caractère à caractère, vous allez provoquer un déplacement global sur l'écran ou sur les lignes. Réalisez les déplacements suivants :

. avancez d'une page,

. reculez d'une page,

. déplacez le curseur sur la dernière ligne du fichier,

. ramenez le curseur sur la première ligne du fichier,

. positionnez le curseur sur la 4eme ligne du fichier,

. déplacez le curseur à la fin de la ligne,

. ramenez le curseur au début de ligne.

»<Ctrl-f> ou éventuellement la touche <PgDn> qui elle peut ne pas fonctionner.

»<Ctrl-b> ou éventuellement la touche <PgUp> qui elle peut ne pas fonctionner.

»<Ctrl-u>

»<shift-g>

»1<shift-g> ou :1 <Entrée>

»4<shift-g> ou :4 <Entrée>

»\$

»0 (la touche zéro)

- ___ 8. Déplacez le curseur au début du fichier. Recherchez le mot `écriture`. Le curseur doit être sur le `e` initial. Basculez en insertion et ajoutez le mot `texte`. N'oubliez pas l'espace après le mot saisi.

»1<shift-g> ou :1

»/écriture

»i

»texte

- ___ 9. Déplacez le curseur sur l'espace après le mot `écriture` sur la même ligne. Insérez une virgule. Vous êtes toujours en mode écriture.

»<Echap>

»Positionnez le curseur après le mot `écriture`

»i, (virgule)

- ___ 10. Basculez en mode commande. Placez le curseur n'importe où sur la ligne qui commence par "`de l'éditeur vi`". Insérez en dessous une ligne vide pour former deux paragraphes.

»<Echap>

»Placez le curseur sur la ligne qui commence par "de l'editeur vi"

»o (o minuscule ouvre une ligne après le curseur)

- ___ 11. Ouvrir une ligne vide, comme vous venez de le faire, vous bascule dans le mode écriture ; donc, basculez en mode commande. Enregistrez vos modifications SANS QUITTER l'éditeur.

»<Echap>

» :w

- ___ 12. En restant en mode de commande, retirez les caractères c, e, g, et laissez les lignes vides à leur place, en d'autres termes, ne pas supprimer la ligne, juste le caractère. Ensuite, supprimer les lignes vides. Cela vous fait pratiquer les deux fonctions de suppression.

» Placez le curseur sur c; Press x

» Placez le curseur sur e; Press x

» Placez le curseur sur g; Press x

» Placez le curseur sur chaque ligne vide. Tapez dd

- ___ 13. Maintenant remplacez le caractère h par un z .

» Placez le curseur sur h

» Appuyez sur r (remplace 1 caractère)

»z

- ___ 14. Vous décidez de ne pas vouloir enregistrer les modifications apportées aux caractères alphabétiques. Quittez la session vi sans enregistrer les modifications apportées depuis la dernière sauvegarde.

» :q!

- ___ 15. Editez le fichier **vitest** une nouvelle fois. Tout d'abord, copiez le 1er paragraphe (y compris la ligne vide), une ligne à la fois, à la fin du fichier. Quand c'est terminé, copiez le second paragraphe à la fin du fichier, cette fois ci en une seule fois.

»\$ **vi vittest**

» Le curseur sur la première ligne; Tapez **yy**

»<shift-g>; Tapez **p**

»2<shift-g>; Tapez **yy**

»<shift-g>; Tapez **p**

»3<shift-g>; Tapez **yy**

»<shift-g>; Tapez **p**

»4<shift-g>; Tapez **3yy**

»<shift-g>; Tapez **p**

___ 16. Vous décidez que les lignes que vous venez d'ajouter à la fin du fichier sont de trop. Supprimez-les en une seule commande.

» Positionnez le curseur sur la première ligne des six lignes copiées à la fin du fichier.

»6dd

___ 17. Maintenant, avant de faire quoi que ce soit d'autre avec ce fichier, vous décidez que vous devez insérer la date et l'heure à la première ligne du fichier. Tout ça, sans quitter l'éditeur **vi**.

»:!date > datefic

» N'appuyez pas sur <Entrée> lorsque le message "Press return to continue" apparaît.

»:0r datefic

» Appuyez <Entrée> deux fois pour continuer.

.C Personnalisation

___ 18. Grâce à la commande **set**, on peut activer temporairement certaines options pour la durée d'une session de **vi**. Assurez vous d'être en mode commande et activez les options suivantes :

. césure automatique des mots à 15 espaces avant la marge droite,

. affichage du texte INPUT MODE lorsque vous êtes en mode écriture,

. affichage de la numérotation des lignes.

» **1<shift-g>**

» **<Echap>**

» **:set wrapmargin=15** (pas d'espace ni avant, ni après le signe =)

» **:set showmode**

» **:set number**

__ 19. Testez chacune des options que vous venez d'activer.

» Les lignes doivent être numérotées,

» Passez en mode écriture par **i** ou **a**. Vous devez voir apparaître INPUT MODE en bas à droite de votre écran.

» Tapez quelques lignes de texte pour tester le retour à la ligne automatique.

» Basculez en mode commande en appuyant sur **<Echap>**. Le message INPUT TEXTE doit disparaître.

__ 20. Enregistrez et quittez l'éditeur.

» **:wq**

.d Edition de l'historique de la ligne des commandes

__ 21. Maintenant que vous êtes familiarisé avec les différents modes **vi** et de ses commandes, passons à la manipulation de l'historique de la ligne de commande du shell. Pour activer ce mode de fonctionnement, sur une ligne de commande du terminal, saisissez la commande **set -o vi**

» **\$ set -o vi**

__ 22. Maintenant vous pouvez rappeler les commandes lancées auparavant, les modifier, et les relancer. Générez un historique de commandes. Affichez la liste des fichiers du répertoire **/usr** (juste les noms, pas tous les attributs). Avec la commande **cat** affichez le contenu du fichier **/etc/filesystems**. Lancez **echo hello ...**

»\$ **ls /usr**

»\$ **cat /etc/filesystems**

»\$ **echo hello**

- __ 23. Supposons que vous voulez modifier une des commandes que vous venez de lancer. Appuyez sur <Echap> pour basculer en mode de commande **vi**. Puis appuyez sur la touche **k** plusieurs fois pour remonter la liste des commandes. Appuyez sur **j** pour redescendre. Ce rappel de commandes est fait grâce à un buffer qui contient les commandes que vous avez déjà lancées. Ces commandes sont stockées dans votre fichier **.bash_history** situé dans votre répertoire d'accueil.

»<Echap>

»**k** (remonte l'historique des commandes stockées dans le buffer).

»**j** (descend dans l'historique des commandes).

- __ 24. Retrouvez la commande **ls**. Utilisez la touche **l** pour avancer le curseur sur le **/** de **/usr**. (remarque : les flèches du clavier peuvent faire quitter la ligne courante et insérer des caractères indésirables. Vous devez utiliser les touche **l** pour avancer vers la droite, et **h** pour reculer vers la gauche). Utilisez la touche **i** pour basculer en mode insertion et ajoutez **-l** pour afficher cette fois une liste détaillée des fichiers. Lancez la commande modifiée.

»**k** (pour retrouver la commande **ls /usr**)

»**l** (pour se placer sur le **/**)

»**i** (pour basculer en insertion. Ou bien utilisez **a** pour ajouter si le curseur est placé sur l'espace avant le **/**)

»**-l**

»<Echap>

- __ 25 . Rappelez la commande **cat**. Cette fois afficher le contenu du fichier **/etc/passwd**.

»<Echap>

»**k** (pour retrouver la commande **cat**)

»**l** (pour déplacer le curseur sur le **f** de **filesystems**)

»**D** (pour effacer le reste de la ligne, ou **dw** pour effacer le mot)

»a (pour ajouter du texte)

»passwd

»<Entrée>

__ 26. Rappelez la commande **cat**. Allez à la fin de la ligne (rappel : \$). Ajoutez un "|" ("pipe") à la fin de la commande pour envoyer le texte généré en sortie vers la commande **wc** et comptez le nombre de lignes.

»<Echap>

»k (pour remonter jusqu'à la commande **cat**)

»\$

»a

»| wc -l

»<Entrée>

Fin de l'exercice

Chapitre 4

Processus

et

Mécanismes

2775ce55ce
Global Knowledge

1. Quelques définitions

Un **programme** est un fichier de type ordinaire sur lequel est positionnée la permission d'exécution.

Un **processus** (ou **tâche**) est un objet système correspondant à l'environnement d'exécution du programme.

L'activité du système se traduit par la présence simultanée de nombreux processus dans une politique de **temps partagé** dans laquelle le noyau gère de façon équitable le partage des ressources.

Unix est, bien entendu, un système à **mémoire virtuelle**. Le noyau gère un espace mémoire constitué de pages qui séjournent soit en mémoire physique, soit dans un espace disque de pagination (appelé aussi *zone de swap*).

Depuis quelques années maintenant, les noyaux Unix intègrent également la notion de **threads** que l'on peut considérer comme des *sous-processus*.

Ces threads constituent des **activités indépendantes** à l'intérieur d'un processus. Ils permettent aux développeurs de concevoir des applications *client/serveur* plus efficaces car le coût de création d'un *thread* est inférieur à celui de création d'un processus. Les threads permettent également de mieux exploiter les architectures matérielles multi-processeurs.

Un processus en erreur ne peut pas compromettre l'intégrité du système car, dans ce cas de figure, le noyau met fin à ce processus via une interruption (ou signal).

Selon le type d'erreur, cette interruption peut s'accompagner de la génération d'un *dump* de l'image mémoire du processus (fichier **core** créé dans le répertoire courant). De tels fichiers peuvent ensuite être exploités à l'aide de débogueurs ou simplement éliminés.

2. Mécanisme de base fork+exec

Une **connexion** en mode texte correspond à la présence d'un processus qui exécute le **programme shell**. Nous l'appellerons, dans la suite, le processus **parent**.

Le programme shell présente une invite de commandes (*prompt*) et attend la frappe de l'utilisateur. Après l'analyse de cette ligne de commande, le programme shell réalise un premier appel système, baptisé **fork**, qui consiste en une duplication de processus (création d'un processus **fil**s).

Après le *fork*, deux processus, quasiment identiques, exécutent le même code. Il est nécessaire alors de les aiguiller vers deux endroits distincts dans la suite du programme. A cet effet, la primitive *fork* retourne un code différent pour chacun des deux processus.

Le processus parent exécute alors l'appel système **wait** qui le met en attente de la fin du processus fils.

De son côté, via l'appel système **exec**, le fils met en œuvre un mécanisme de recouvrement (*overlay*). L'ancien code, à savoir celui du programme shell, est complètement remplacé par le code de la commande souhaitée.

En fin de traitement, ce nouveau programme fera appel à la primitive **exit** qui mettra fin au processus fils en libérant les ressources systèmes et en permettant le déblocage de l'appel système *wait* sur lequel le processus parent est en attente.

La terminaison du processus fils met donc fin à l'attente du parent et permet également de récupérer un code retour au niveau de ce processus parent. Celui-ci peut maintenant enchaîner sur la présentation de l'invite de commande afin de recommencer un nouveau cycle jusqu'à demande de déconnexion.

Plus généralement, par rapport à ce mécanisme interactif, il est nécessaire, en programmation multi-tâches, que le processus parent effectue, à un moment donné, une attente (*wait*) sur les fils générés afin d'éviter la création de processus dits **zombies** (*defunct*). Un zombie est un processus qui a libéré ses ressources mais qui continue à occuper une entrée dans la table des processus du système.

La présence ponctuelle de zombies peut être normale suivant les rythmes respectifs des différents processus concernés. Par contre, la présence, voire la multiplication, de processus zombies sur une longue période mettrait en évidence un problème logiciel dans un schéma de programmation multi-tâches. Un processus zombie qui serait le résultat d'une erreur de programmation ne pourra être éliminé qu'au prix d'un redémarrage du système.

3. Attributs des processus

- PID (Process IDentificator)

Il s'agit d'un entier, attribué par le noyau, permettant de désigner le processus. Ce nombre est, bien entendu, unique à un instant donné.

- PPID (Parent Process IDentificator)

Il s'agit du numéro du processus parent. De par le mécanisme *fork+exec*, un processus est toujours créé par un autre processus. Dans certaines activités, il est intéressant de connaître l'identité du processus parent.

- UID (User IDentificator) et GID(Group IDentificator)

Il s'agit de numéros internes correspondant au propriétaire et au groupe effectif du processus. Un processus possède, en fait, deux propriétaires (respectivement deux groupes). Le propriétaire **effectif** détermine les droits du processus par rapport aux fichiers du système. Le propriétaire **réel** est pris en compte dans divers autres aspects de statistiques, de comptabilité ou de communications inter-processus. Par défaut, les propriétaires réel et effectif correspondent à l'utilisateur ayant lancé le programme. Mais, si le programme possède la permission **SUID** (voir chapitre 2), le propriétaire du fichier programme devient le propriétaire effectif du processus.

- TTY

Il s'agit du terminal de contrôle du processus. Le terminal de contrôle est une notion importante par rapport aux mécanismes de redirections, présentés plus loin dans ce chapitre. Intuitivement, le terminal de contrôle correspond au clavier et à l'écran pour les processus lancés en interactif. Certains processus, lancés par programme ou encore au démarrage du système, ne possèdent pas de terminal de contrôle. On les qualifie souvent de **daemons** (démons).

- Nice value

La *nice value* représente une valeur de priorité de départ. Les noyaux Unix attribuent les ressources système selon une politique de **temps partagé**. La priorité effective d'un processus évolue donc constamment, au fur et à mesure de l'exécution. Dans les faits, l'influence réelle de la *nice value* (priorité de départ, plus ou moins forte) est assez difficile à prévoir.

4. La commande ps

La commande **ps** (*process status*) permet de visualiser un certain nombre d'attributs concernant les processus en activité. Les options ainsi que le détail de l'affichage varient selon les versions. Une lecture attentive de la *page de man* est souvent nécessaire.

Sans option, la commande donne un affichage résumé des processus associés au terminal de l'utilisateur.

\$ **ps**

```
PID TTY      TIME CMD
463 pts/2    0:00 ksh
479 pts/2    0:00 ps
```

\$

Cette première liste visualise le processus correspondant au *shell* de connexion (ksh) ainsi que le processus correspondant à la commande *ps* elle-même. On y reconnaît les attributs PID et TTY. La colonne TIME correspond au temps d'exécution cumulé en minutes et secondes. Tant qu'un processus n'a pas consommé au moins une seconde de temps CPU, les compteurs restent à zéro. La colonne CMD correspond, bien entendu, au nom du programme.

L'option **-f** permet d'obtenir plus de détails.

\$ **ps -f**

```
UID    PID  PPID  C   STIME TTY      TIME CMD
mike   463   461   0  13:50:50 pts/2    0:00 -ksh
mike   480   463   0  15:09:43 pts/2    0:00 ps -f
```

\$

Cette deuxième liste visualise trois informations supplémentaires. On reconnaît l'attribut PPID (processus parent). L'examen des colonnes PID et PPID nous permet de vérifier que le processus correspondant à la commande *ps* est bien le fils de celui correspondant au programme *shell*. Cela concrétise le mécanisme de *fork+exec* évoqué précédemment. La colonne STIME correspond à la date de démarrage des processus. Pour le shell, cela correspond à la date de connexion. La colonne C représente une valeur instantanée de priorité, peu exploitable à notre niveau de discours.

L'option à argument **-u** doit être suivie d'un nom d'utilisateur. Elle permet d'obtenir tous les processus dont cet utilisateur est propriétaire, quels que soient les terminaux de contrôle.

\$ **ps -fu mike**

UID	PID	PPID	C	STIME	TTY	TIME	CMD
mike	331	309	0	09:31:54	?	0:03	/usr/openwin/bin/Xsun :0
mike	463	461	0	13:50:50	pts/2	0:00	-ksh
mike	620	463	0	15:24:44	pts/2	0:00	ps -fu mike
mike	547	533	0	15:20:39	pts/4	0:00	/usr/dt/bin/dtsession
mike	551	547	0	15:21:04	?	0:01	/usr/dt/bin/dtfile
mike	549	547	0	15:20:40	?	0:02	dtwm
mike	611	610	0	15:24:17	?	0:00	/usr/dt/bin/dtterm
mike	613	611	0	15:24:17	pts/5	0:00	/usr/bin/ksh

.....

Cette liste de processus est plus fournie que les précédentes puisqu'il s'agit bien de tous les processus dont l'utilisateur *mike* est propriétaire. En contexte graphique notamment, les fenêtres actives ainsi qu'un certain nombre de services système sous-jacents se traduisent assez vite par de nombreux processus en activité.

L'option **-e** (*everyone*) permet d'obtenir la liste complète des processus.

\$ **ps -ef**

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	0	0	0	09:30:52	?	0:03	sched
root	1	0	0	09:30:53	?	0:00	/etc/init -
root	2	0	0	09:30:53	?	0:00	pageout
root	3	0	0	09:30:53	?	0:00	fsflush
root	338	1	0	09:31:58	?	0:00	/usr/lib/saf/sac
root	341	338	0	09:31:58	?	0:00	/usr/lib/saf/ttymon

.....

Ce début de liste permet de remarquer plusieurs choses. Par tradition, le processus de **numéro 0** correspond au **noyau** Unix. Il s'agit du seul processus pour lequel le *PID* et le *PPID* sont identiques. Parmi les fils immédiats du noyau, le processus le plus important, pour les administrateurs système, correspond au *PID* **numéro 1** (programme *init*). Ce programme *init* est, en effet, l'ancêtre de tous les processus du système.

5. Signaux, interruption des processus

a. Description des signaux Unix

Un **signal** (ou **interruption**) correspond à une action dont la conséquence par défaut est l'**élimination** du processus concerné, à la condition d'en être propriétaire.

Les signaux peuvent être générés de différentes façons (action de l'utilisateur au clavier, commande **kill**, par programme, par le noyau souhaitant éliminer un processus en erreur).

Si rien n'a été prévu par le développeur, un processus est donc éliminé lors de la réception d'un signal provenant d'un autre processus, de même propriétaire ou de propriétaire *root*. Cependant, les programmes peuvent comporter des traitements associés à la réception de certains signaux, soit pour les ignorer, soit pour exécuter un sous-programme dit de déroutement, consistant souvent à supprimer des objets système avant de se terminer.

Certains signaux ont pour effet de générer un fichier **core** dans le répertoire courant. Ce fichier contient l'image mémoire du processus au moment de l'interruption et peut ainsi être examiné à l'aide d'un débogueur.

Les noms des signaux Unix sont standardisés au niveau des interfaces de programmation. Leur numéro interne peut, par contre, varier selon les versions. Cependant, pour les signaux qui concernent l'utilisateur, les numéros sont standards :

- | | |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| - INT (2) | Généré au clavier, le plus souvent par la touche Ctrl c . |
| - QUIT (3) | Généré au clavier, le plus souvent par la touche Ctrl \ . Par rapport au précédent, son objectif est l'obtention d'un fichier core . |
| - TERM (15) | Ce signal peut être généré via la commande kill . Il correspond à une méthode d'arrêt contrôlé d'un processus. En effet, le programme concerné peut éventuellement traiter ce signal pour effectuer, par exemple, la suppression de certains objets système avant de se terminer. |
| - KILL (9) | Ce signal peut être généré via la commande kill . Il correspond à un arrêt immédiat du processus car il ne peut pas être capté par programme. Ce signal ne |

doit donc être utilisé pour terminer un processus que dans le cas où le signal 15 aurait échoué.

- HUP (1)

Lors de la déconnexion (fin du *shell*), ce signal est systématiquement envoyé à tous les processus de même terminal de contrôle. Nous verrons, au cours de ce chapitre, que la prise en compte de ce signal est importante dans le lancement de processus en mode arrière-plan.

b. Comment éliminer un processus ?

La condition nécessaire pour éliminer un processus, via une interruption, est d'émettre le signal depuis un processus de même propriétaire ou depuis un processus de propriétaire *root*.

Le scénario classique d'élimination d'un processus est le suivant :

1) Recourir aux interruptions clavier

Si le processus visé est associé à notre terminal, nous allons utiliser les interruptions clavier, à savoir le **signal 2** (touche *Ctrl c*) ou le **signal 3** (touche *Ctrl *).

2) En cas d'échec des interruptions clavier

Si le programme est protégé contre les interruptions clavier, nous allons nous connecter "ailleurs" (nouvelle fenêtre X-Window ou nouvelle session *telnet* par exemple). Depuis cette nouvelle connexion, la commande **ps** (avec l'option *-u*) va nous permettre d'obtenir le **PID** du processus à éliminer.

3) Utilisation de la commande kill

Malgré son nom, la commande **kill** ne permet pas toujours d'éliminer un processus. Son rôle exact est, en fait, d'envoyer un signal vers un processus dont on donne le *PID* en argument. Le numéro du signal sera fourni en option à la commande. Quand on ne fournit pas d'options, la commande génère le **signal 15**.

4) En cas d'échec de la commande kill

Si le programme est protégé contre le signal 15, le dernier recours est de générer le **signal 9** qui ne pourra pas être traité par ce programme. Syntaxiquement, le signal 9 sera fourni en option à la commande **kill** : **kill -9 PID**.



Il est important d'insister sur le fait que le signal 9 ne doit être utilisé qu'en dernier recours. En effet, certains programmes peuvent souhaiter capter le signal 15 afin

de supprimer, avant de se terminer, des objets utilisés en programmation système (segments de mémoire partagée, sémaphores, files de message...). L'émission systématique du signal 9 correspond à un arrêt brutal et ne permet pas ce nettoyage logiciel.

6. Redirections

Les concepteurs d'Unix ont souhaité qu'une commande puisse disposer, dès son démarrage, de trois descripteurs logiques de fichiers, implicitement ouverts.

Ces descripteurs sont les suivants :

- | | |
|--------------------------|--------------------------------------------------|
| - entrée standard | La commande peut y lire des données. |
| - sortie standard | La commande peut y écrire des résultats. |
| - erreur standard | La commande peut y écrire des messages d'erreur. |

Ces descripteurs logiques seront mis en correspondance avec des fichiers physiques au niveau du shell, de façon transparente pour les commandes. Ils sont associés par défaut au **terminal de contrôle** du processus, si celui-ci est défini. Le shell de connexion possède, bien entendu, un terminal de contrôle et toutes les commandes de la session héritent naturellement de ce terminal (via le mécanisme du *fork*). Concrètement, l'entrée standard correspond par défaut au clavier tandis que la sortie et l'erreur standard ne sont pas différenciées et sont associées, toutes deux, à l'écran.

Les commandes Unix n'exploitent pas systématiquement les trois descripteurs. Elles adoptent en général les comportements suivants :

1) Une commande dont le rôle n'est pas de produire une liste comme résultat n'écrit rien sur la **sortie standard**. Elle préfère positionner un **code retour** pour les autres commandes qui souhaiteraient le tester. Cette tradition explique pourquoi une commande Unix qui se passe bien est souvent muette. Certains nouveaux utilisateurs d'Unix seront surpris, voire agacés, par ce comportement qui oblige souvent à s'assurer, par une autre commande, du bon fonctionnement d'une première commande.

```
$ mkdir rep
$ echo $?
0
$ ls -ld rep
drwxr-xr-x  2 mike  mike          512 Sep 21 17:39 rep
$
```

La commande *mkdir* a créé un répertoire mais n'a pas signalé la réussite de l'opération. Au niveau du shell, le code retour de la dernière commande est disponible via la notation **\$?** et son affichage, immédiatement après l'appel de la commande *mkdir*, nous assure déjà du succès de l'opération. En effet, une commande doit retourner zéro en cas de succès.

2) Les commandes prévoient un usage assez systématique de l'**erreur standard** pour compléter le code retour par un message sur ce descripteur.

```
$ mkdir /rep
mkdir: Failed to make directory "/rep"; Permission denied
$
```

La commande *mkdir* a échoué car l'utilisateur a tenté de créer un répertoire directement sous la racine Unix. Un message d'erreur est tout naturellement envoyé sur le descripteur erreur standard pour expliquer la cause de l'échec.

3) Une commande dont le rôle est de produire une liste comme résultat utilise, bien entendu, la sortie standard mais exploite aussi l'erreur standard pour séparer résultats proprement dits et messages d'erreur.

```
$ ls -l f1 f10
f10: No such file or directory
-rw-r--r--  1 mike  mike          30 Sep 21 17:49 f1
$
```

Dans cet exemple, le fichier *f10* n'existe pas. Le message correspondant est écrit sur l'erreur standard tandis que la description du fichier existant *f1* est écrite sur la sortie standard. Par défaut, les deux informations sont affichées à l'écran. Le mécanisme de redirection, déclenché au niveau du shell, permettra de bien différencier les deux notions si on le souhaite.

4) Il faut bien distinguer les arguments et les éventuelles données présentes sur l'**entrée standard**. La confusion peut avoir lieu dans la mesure où les deux entités sont tapées au clavier par l'utilisateur. Une commande essaye de traiter, la plupart du temps, une liste d'arguments. Dans le cas où l'utilisateur de la commande ne fournit pas d'argument, celle-ci décide souvent, en deuxième choix, de chercher ses données sur l'entrée standard. Quand l'entrée standard correspond au clavier, la fin de fichier devra être indiquée par la frappe de la touche **Ctrl d** en début de ligne.

\$ **cat**

```
je n'ai pas fourni d'argument a la commande cat
je n'ai pas fourni d'argument a la commande cat
elle a donc decide de lire son entree standard
elle a donc decide de lire son entree standard
elle affiche en echo ses donnees sur la sortie standard
elle affiche en echo ses donnees sur la sortie standard
elle va se terminer sur un ctrl d en debut de ligne
elle va se terminer sur un ctrl d en debut de ligne
$
```

Dans cet exemple, nous ne donnons pas d'argument à la commande *cat*. Elle se met donc en lecture sur son descripteur *entrée standard*. Tout ce qui est tapé au clavier, puis validé par la touche *Entrée*, se retrouve affiché sur la sortie standard (l'écran par défaut). Il s'agit bien là du rôle normal de la commande. La frappe du caractère *Ctrl d* en début de ligne provoque la fin du traitement en indiquant à la commande que les données sont terminées.



Certaines commandes (assez rares) peuvent utiliser simultanément des arguments et des données complémentaires sur l'entrée standard. D'autres (assez rares également) préfèrent n'utiliser que l'entrée standard comme source de données.

Une opération de **redirection** consiste à changer l'association entre le descripteur logique et le fichier physique. Une redirection peut s'effectuer vers un fichier ordinaire, un fichier spécial (un autre terminal par exemple) ou vers un tube. Nous n'avons pas encore évoqué le concept de *tube*. Ce terme fait référence au mécanisme de *pipeline* que nous présenterons plus loin dans ce chapitre.



Les redirections sont effectuées par le shell avant l'appel des commandes. Celles-ci n'ont pas connaissance des fichiers physiques associés aux descripteurs. Ce choix de conception facilite grandement le travail des développeurs sous Unix.

a. Redirection de l'entrée standard

La redirection de l'entrée standard est obtenue via le caractère **<**.

Le fichier concerné doit exister et être accessible en lecture. Les données seront donc lues dans ce fichier au lieu d'être tapées au clavier.

Nous allons illustrer la redirection de l'entrée via la commande **write**. Cette commande de messagerie instantanée est un exemple de commande Unix qui utilise simultanément argument et entrée standard. L'argument doit correspondre au nom d'un utilisateur connecté alors que le texte des messages sera lu sur l'entrée standard. Si la commande est utilisée de manière complètement interactive, les textes des messages seront tapés au clavier. La fin de conversation correspondra à la saisie de la touche *Ctrl d* en début de ligne. L'utilisateur destinataire conduira le dialogue de manière symétrique. Par contre, si un texte global doit être affiché ponctuellement sur le terminal du destinataire, il sera possible de l'avoir préparé dans un fichier et d'utiliser la redirection de l'entrée pour l'envoyer à la personne concernée.

```
$ cat message
```

```
J'ai prepare mon texte dans un fichier  
afin de pouvoir utiliser une redirection de l'entree  
lors de l'appel de la commande write
```

```
$ write mike < message
```

```
Message from mike on solaris9 (pts/2) [ Wed Sep 22 10:21:23 ]
```

```
J'ai prepare mon texte dans un fichier  
afin de pouvoir utiliser une redirection de l'entree  
lors de l'appel de la commande write
```

```
<EOT>
```

```
$
```

Dans cet exemple, l'utilisateur a préparé son texte dans le fichier *message* et il redirige l'entrée standard de la commande *write* vers ce fichier afin de fournir un texte complet à la commande, sans frappe interactive au clavier.

b. Redirection de la sortie standard

La redirection de la sortie est bien plus répandue que celle de l'entrée. L'objectif est d'archiver le résultat d'une commande dans un fichier. Si nécessaire, le fichier résultat sera créé et, selon la syntaxe utilisée, il sera, par la suite, écrasé ou complété.

Les redirections vers des fichiers spéciaux (des terminaux, par exemple) sont également possibles mais elles sont rarement pratiquées par l'utilisateur. En effet, celui-ci ne possède pas, en général, les permissions nécessaires pour écrire sur un autre terminal que le sien.

c. Redirection de la sortie en mode écrasement

La redirection de la sortie en mode écrasement est obtenue via le caractère **>**. Les résultats de la commande seront donc écrits dans un fichier.

Si ce fichier n'existe pas encore, il est créé avec les droits par défaut correspondant au choix effectué via la commande **umask** (voir chapitre 2). Il est nécessaire alors de posséder le droit d'écriture sur le répertoire concerné (il s'agit d'une création de fichier).

Si ce fichier existe déjà, son ancien contenu est remplacé par les résultats de la commande. En toute logique, le fichier conserve ses droits d'accès. Il est nécessaire alors de posséder le droit d'écriture sur le fichier lui-même (il s'agit d'une modification de contenu). Le droit d'écriture sur le répertoire où se trouve le fichier n'est pas requis.

d. Redirection de la sortie en mode ajout

La redirection de la sortie en mode ajout est obtenue via les caractères **>>**.

Dans cette deuxième forme, lorsque le fichier existe déjà, les résultats de la commande sont écrits à la fin du fichier et viennent donc compléter l'ancien contenu.

```
$ cal 2 1959 > monfichier
$ echo "je rajoute du texte en fin de fichier" >> monfichier
$
$ cat monfichier
    February 1959
S  M Tu  W Th  F  S
1  2  3  4  5  6  7
8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
je rajoute du texte en fin de fichier
$
```

```
$ cat > fichier
je tape des lignes au clavier et je les stocke en meme temps dans un
fichier
je termine la saisie par un ctrl d en debut de ligne
```

\$ **cat fichier**

je tape des lignes au clavier et je les stocke en meme temps dans un fichier

je termine la saisie par un ctrl d en debut de ligne

\$

Les symboles de redirection peuvent apparaître n'importe où dans la ligne de commande. Dans l'exemple ci-dessous, la commande *cal* est mentionnée après la redirection mais celle-ci est tout de même valide. Cette sorte de construction reste très marginale et ne facilite pas la lisibilité de la ligne de commande.

\$ **> monfichier cal**

\$ **cat monfichier**

September 2004

S	M	Tu	W	Th	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

\$

e. Élimination de la sortie

Lorsque, dans un contexte de script par exemple, nous ne sommes intéressés que par le code retour d'une commande, nous avons quelquefois envie ou besoin d'éliminer les sorties de cette commande. Il suffit d'effectuer alors une redirection vers le pseudo-périphérique **/dev/null** qui correspond à une sorte de poubelle sans fond où les sorties peuvent disparaître.

\$ **echo "Bienvenue au club Unix"**

Bienvenue au club Unix

\$ **echo "Bienvenue au club Unix" > /dev/null**

\$

f. Création de fichier

En utilisant une redirection sans mentionner de commande, il est possible de créer un fichier vide ou de purger un fichier existant. Dans l'exemple ci-dessous, le fichier *toto* est, tout d'abord, créé par la seule redirection et il est, bien entendu, vide. Par la suite, après avoir contenu des données (redirection de la commande *cal*), son contenu est réinitialisé.

\$ **ls -l toto**

toto: No such file or directory

```

$ > toto
$ ls -l toto
-rw-r--r--  1 mike  mike           0 Sep 22 15:07 toto
$ cal > toto
$ ls -l toto
-rw-r--r--  1 mike  mike        139 Sep 22 15:07 toto
$ > toto
$ ls -l toto
-rw-r--r--  1 mike  mike           0 Sep 22 15:07 toto
$

```

g. Protection contre l'écrasement accidentel

Il est difficile de mémoriser tous les noms de ses propres fichiers et le mécanisme des redirections peut vite conduire à des écrasements non maîtrisés de fichiers existants.

Le comportement du shell peut être modifié via de nombreuses options, positionnées à l'aide de la commande **set -o option**. L'option **noclobber** a pour conséquence d'empêcher la réinitialisation d'un fichier existant par une redirection. Cet écrasement doit être forcé explicitement par l'ajout du caractère **|** derrière le caractère de redirection. Cette contrainte évite bien évidemment un écrasement silencieux non maîtrisé.

La syntaxe **set +o option** a pour effet d'annuler l'activation de l'option concernée. Dans notre exemple, l'écrasement via une redirection redevient implicite après l'annulation de l'option *noclobber*.

```

$ set -o noclobber
$ date > monfichier
ksh: monfichier: file already exists
$ date >| monfichier
$ set +o noclobber
$ cal > monfichier
$

```

h. Redirection de l'erreur standard

La redirection de l'erreur standard est similaire à celle de la sortie. Il suffit d'ajouter le chiffre 2 devant les symboles de redirection, soit respectivement **2>** et **2>>** pour les deux modes *écrasement* ou *ajout en fin de fichier*.

📖 *Le choix du chiffre 2 provient du fait que les trois descripteurs prédéfinis (entrée, sortie et erreur) correspondent en interne aux numéros 0, 1 et 2. Les caractères de redirection peuvent être précédés syntaxiquement du numéro de descripteur concerné. Dans la pratique, les numéros 0 et 1 n'ont pas besoin d'être explicitement mentionnés. Cependant, les notations 1> et > seraient équivalentes tout comme le seraient les syntaxes < et 0<.*

En cas de mauvaise syntaxe d'appel, la commande `cp` produit un message sur l'erreur standard. Nous le redirigeons ici vers un fichier *erreur*.

```
$ cp
cp: Insufficient arguments (0)
Usage: cp [-f] [-i] [-p] [-@] f1 f2
        cp [-f] [-i] [-p] [-@] f1 ... fn d1
        cp -r|R [-f] [-i] [-p] [-@] d1 ... dn-1 dn

$ cp 2> erreur
$ cat erreur
cp: Insufficient arguments (0)
Usage: cp [-f] [-i] [-p] [-@] f1 f2
        cp [-f] [-i] [-p] [-@] f1 ... fn d1
        cp -r|R [-f] [-i] [-p] [-@] d1 ... dn-1 dn

$
```

Les redirections respectives de la sortie et de l'erreur peuvent, bien entendu, être combinées sur la ligne de commande, comme le montre très simplement l'exemple ci-dessous.

```
$ ls f1 f2

f2: No such file or directory

f1

$ ls f1 f2 > resul 2> erreur

$ cat resul

f1

$ cat erreur

f2: No such file or directory

$
```


Quand un nom de fichier a déjà été mentionné dans une redirection, on ne peut le désigner ensuite que par son numéro de descripteur précédé alors du caractère **&**. Une redirection combinée de la sortie et de l'erreur vers une même destination illustre cette contrainte syntaxique.

```
$ ls f1 f2

f2: No such file or directory

f1

$ ls f1 f2 > resul 2>&1

$ cat resul

f2: No such file or directory

f1

$
```

La notation **2>&1** (sans aucun espace) signifie que l'erreur standard doit être redirigée vers le fichier portant le numéro 1, à savoir la sortie standard, elle-même déjà redirigée vers le fichier *resul*.

7. Processus séquentiels

Plusieurs commandes peuvent être tapées sur la même ligne via le caractère ; (point-virgule). Les processus correspondants sont indépendants et s'exécutent séquentiellement, les uns après les autres.

```
$ id -a ; cal 2 1959
uid=2000(mike) gid=2000(mike) groups=2000(mike),3000(gkn)
  February 1959
  S  M Tu  W Th  F  S
  1  2  3  4  5  6  7
  8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
$
```

L'utilisation intuitive de parenthèses permet de réaliser des **redirections globales**. Dans l'exemple suivant, toutes les sorties des différentes commandes sont

redirigées séquentiellement dans un seul fichier résultat. Le regroupement de plusieurs commandes dans des parenthèses provoque, en fait, la création d'un shell fils permettant ce genre d'opérations combinées.

```
$ ( id -a ; cal 2 1959 ; tty ) > toto
$ cat toto
uid=2000(mike) gid=2000(mike) groups=2000(mike),3000(gkn)
    February 1959
  S  M Tu  W Th  F  S
  1  2  3  4  5  6  7
  8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
/dev/pts/2
$
```

8. Mécanisme du pipeline

.a Principes généraux

Le **pipeline** est un mécanisme très caractéristique de l'esprit "boîte à outils" Unix car il permet d'obtenir un traitement complet à partir d'une combinaison de commandes qui propagent leurs résultats intermédiaires. **Les résultats d'une commande deviennent les données de la commande suivante** et ainsi de suite jusqu'à obtenir le résultat recherché.

Pour concevoir un *pipeline* efficace, il convient, d'une part, de bien connaître les commandes disponibles et, d'autre part, d'avoir de bonnes idées afin de trouver des combinaisons intéressantes. Un peu d'astuce et d'expérience constituent les ingrédients nécessaires dans cette activité.

Syntaxiquement, le *pipeline* correspond au caractère | intercalé entre les différentes commandes.

La **sortie standard** du premier processus est redirigée vers un *pipe* (ou *tube* dans une traduction répandue) qui est l'équivalent d'un fichier temporaire se comportant comme une file (*FIFO: First In, First Out*).

L'**entrée standard** du processus suivant est redirigée vers ce même tube pour en réaliser la consommation.

Le nombre de commandes mises en jeu, ainsi que la quantité de données échangées, ne sont pas limités. Le shell initial crée un tube pour chaque couple de

commandes et effectuent les redirections. Grâce au principe des descripteurs logiques, les commandes s'utilisent sans changement. Le système assure une **synchronisation implicite** de tous les processus. Ceux-ci s'exécutent en parallèle, ils sont tous créés, dès le départ, par le shell initial. Le traitement a lieu aussi longtemps que le premier processus alimente le *pipeline*.

L'entrée standard du premier processus ainsi que la sortie du dernier restent disponibles pour des redirections classiques.



Toutes les commandes Unix ne se prêtent pas à une utilisation dans un pipeline. Pour jouer le rôle de producteur, il est clair que la commande doit utiliser sa sortie standard. Symétriquement, pour jouer le rôle de consommateur, il faut choisir des commandes qui lisent leur entrée standard, le plus souvent en deuxième choix quand elles ne reçoivent pas d'argument.

```
$ who
mike      pts/2          sept 22 15:17    (192.168.0.1)
root      pts/3          sept 22 17:42    (192.168.0.4)
mike      pts/4          sept 22 17:43    (localhost)
$ who | wc -l
3
$
```

Ce pipeline permet d'obtenir le nombre d'utilisateurs connectés. La commande *who* affiche une ligne par utilisateur connecté. La commande *wc* avec l'*option -l* donne un nombre de lignes. Elle ne reçoit pas d'argument, elle va donc compter les lignes sur son entrée standard. Concrètement, la commande *wc* nous permet de compter le nombre de lignes du résultat de la commande *who* et, par là même, nous donne le nombre d'utilisateurs.

Nous voulons savoir maintenant combien de fois est connecté l'utilisateur *mike*. Nous allons avoir besoin de la commande **grep** (détaillée dans le chapitre 6). Cette commande reçoit, en premier argument, une chaîne de caractères à chercher dans les fichiers donnés en arguments supplémentaires. Si elle ne reçoit pas de fichiers en argument, elle cherchera tout naturellement la chaîne sur son entrée standard. Il nous suffit donc, par rapport au pipeline précédent, d'intercaler un appel à la commande *grep* pour constituer ainsi un pipeline à trois commandes.

```
$ who
mike      pts/2          sept 22 15:17    (192.168.0.1)
root      pts/3          sept 22 17:42    (192.168.0.4)
mike      pts/4          sept 22 17:43    (localhost)
$ who | grep mike
mike      pts/2          sept 22 15:17    (192.168.0.1)
mike      pts/4          sept 22 17:43    (localhost)
```

```
$ who | grep mike | wc -l
```

```
2
```

```
$
```

Les commandes **head** et **tail** (présentées dans le chapitre 2) sont très fréquentes et utiles dans les pipelines. Dans le premier exemple ci-dessous, nous obtenons les cinq premières lignes du résultat de la commande `ps -ef` (liste de tous les processus). Dans le second exemple, nous intercalons la commande `tail -n +2` (affichage à partir de la deuxième ligne) pour éliminer la ligne de titre de la commande `ps` et obtenir ainsi vraiment les cinq premiers processus de la liste.

```
$ ps -ef | head -5
```

	UID	PID	PPID	C	STIME	TTY	TIME	CMD
	root	0	0	0	07:50:52	?	0:03	sched
	root	1	0	0	07:50:53	?	0:00	/etc/init -
	root	2	0	0	07:50:53	?	0:00	pageout
	root	3	0	0	07:50:53	?	0:00	fsflush

```
$ ps -ef | tail -n +2 | head -5
```

	root	0	0	0	07:50:52	?	0:03	sched
	root	1	0	0	07:50:53	?	0:00	/etc/init -
	root	2	0	0	07:50:53	?	0:00	pageout
	root	3	0	0	07:50:53	?	0:00	fsflush
	root	338	1	0	07:51:58	?	0:00	/usr/lib/saf/sac

```
$
```



L'erreur standard n'est pas redirigée vers le tube. Il faut en tenir compte, par exemple, lors de l'utilisation du très classique pipeline vers la commande `more` afin de ne pas subir de décalages de l'affichage qui seraient dus au fait que les messages d'erreur continueraient à s'afficher à l'écran, sans être filtrés par la commande `more`. La solution est alors d'utiliser la syntaxe `2>&1` qui permet de rediriger l'erreur au même endroit que la sortie et, en l'occurrence, vers le tube pour un traitement correct via la commande `more`. La syntaxe à utiliser serait donc : commande `2>&1 | more`.

.b Mémoriser les résultats intermédiaires

La commande **tee** reçoit un nom de fichier en argument. Elle écrit toutes les données lues sur son entrée standard, à la fois sur sa propre sortie standard et dans le fichier. Elle réalise donc une duplication de la sortie standard vers un fichier.

La redirection vers le fichier a lieu, par défaut, en mode écrasement mais peut avoir lieu en mode ajout en utilisant l'*option* **-a**.

La commande *tee* est utile pour mémoriser les résultats intermédiaires d'un pipeline. Dans l'exemple suivant, nous mémorisons dans le fichier *f1* le résultat de *who | grep mike*.

```
$ who
mike      pts/2          sept 22 15:17    (monpc)
root      pts/3          sept 22 18:46    (solaris9)
mike      pts/5          sept 22 18:47    (localhost)
root      pts/4          sept 22 18:47    (solaris9)
$ who | grep mike | tee f1 | wc -l
2
$ cat f1
mike      pts/2          sept 22 15:17    (monpc)
mike      pts/5          sept 22 18:47    (localhost)
$
```

La commande *tee* est très intéressante en tant que **dernier membre d'un pipeline**. Elle permet ainsi d'archiver le résultat final tout en le visualisant à l'écran, ce qui ne serait pas le cas si on terminait le pipeline par une redirection simple.

```
$ ps -ef | tail -n +2 | head -5 | tee resul
root      0          0 0 07:50:52 ?        0:03 sched
root      1          0 0 07:50:53 ?        0:00 /etc/init -
root      2          0 0 07:50:53 ?        0:00 pageout
root      3          0 0 07:50:53 ?        0:00 fsflush
root     338        1 0 07:51:58 ?        0:00 /usr/lib/saf/sac
$
```

9. Processus en arrière-plan

Un programme peut être lancé dans un mode baptisé **arrière-plan** (*background*). Cela signifie que le processus shell parent ne se met pas, comme à l'habitude, en attente de la fin du processus fils. Concrètement, nous gardons la main pour effectuer d'autres traitements pendant le déroulement de ce fils.

Cette possibilité d'arrière-plan était fondamentale, dans les premières années, lorsque les utilisateurs ne disposaient que d'un seul terminal, sans autres possibilités de connexions simultanées. Aujourd'hui, dans les environnements graphiques, cette fonctionnalité reste malgré tout d'actualité car elle évite de multiplier inutilement les fenêtres d'émulateurs de terminaux.

Pour lancer un programme en arrière-plan, il suffit de terminer la ligne de commande par le caractère **&**.

La seule utilisation de ce caractère **&** n'est cependant pas suffisante. En effet, le processus fils hérite des descripteurs de fichiers du shell parent. La sortie et l'erreur standard du programme lancé en arrière-plan sont donc toujours associées au terminal de l'utilisateur. Par conséquent, il s'avère indispensable d'effectuer des redirections. Cela évite, d'une part, une "pollution" plus ou moins pénible de son écran par les résultats du programme et cela permet, d'autre part, de mémoriser les résultats dans des fichiers, consultables a posteriori.

```
$ ls -alR /usr > resul 2> /dev/null &
[1]      386
$ ps -f
  UID    PID  PPID  C   STIME TTY      TIME CMD
  mike   358   356  0  09:19:36 pts/2    0:00 -ksh
  mike   387   358  0  10:02:32 pts/2    0:00 ps -f
  mike   386   358 10  10:02:28 pts/2    0:02 ls -alR /usr
$
[1] +  Done(2)      ls -alR /usr > resul 2> /dev/null &
$ ls -l resul
-rw-r--r--  1 mike   mike   5080786 Sep 23 10:03 resul
$
```

Dans l'exemple précédent, nous lançons en arrière-plan la commande *ls* pour obtenir la liste complète de l'arborescence */usr*. La sortie standard est redirigée vers le fichier *resul*. Les messages éventuels sur l'erreur standard sont éliminés. Lorsque le programme est terminé, nous en sommes avertis par un message lors d'une entrée clavier. Nous vérifions alors que le fichier *resul* a bien été créé.

a. Pouvoir se déconnecter (mode détaché)

À la déconnexion (fin du *shell*), le signal *HUP* (numéro 1) est systématiquement envoyé à tous les processus de même terminal de contrôle. Ceci provoque, par défaut, leur élimination.

La commande **nohup** a pour effet de protéger un processus contre le signal *HUP*. Nous devons donc utiliser cette commande pour nos processus lancés en arrière-plan afin de leur permettre de continuer leur exécution, même après déconnexion.

Le programme **init** (de *PID* égal à 1) se chargera de l'attente obligatoire de ces processus (notion de *zombie* évoquée en début de chapitre). D'autre part, il est à noter que les processus n'auront plus de terminal de contrôle, ce qui nous incitera, a fortiori, à effectuer des redirections explicites vers des fichiers résultats.

Si nous négligeons cet aspect, le shell décidera systématiquement de rediriger (en mode ajout) les résultats et les erreurs vers un fichier **nohup.out**, généré dans le répertoire courant.

☞ Si nous lançons en arrière-plan, via la commande **nohup**, un programme qui ne nécessite pas de redirections, un fichier **nohup.out** (de taille 0) sera quand même généré. Il s'agit là, en effet, d'une décision anticipée du shell.

```
$ nohup ls -alR /usr > resul 2> /dev/null &
[1]      428
$
$ exit
You have running jobs
$ exit
```

Dans cet exemple, nous utilisons *nohup* pour protéger la commande *ls* en cas de déconnexion. Lorsque nous voulons nous déconnecter, le shell nous envoie un avertissement (*You have running jobs*) mais il nous suffit d'insister par un deuxième *exit*. Il s'agit là du comportement normal de l'interpréteur.

Par la suite, après nous être connectés de nouveau, nous vérifions la présence de la commande *ls* dans la liste des processus. L'utilisation de l'option *-u* est nécessaire car le processus a été détaché du terminal et il n'apparaîtrait pas avec la seule option *-f*. Nous constatons également que le *PPID* du processus est maintenant égal à 1. Cela concrétise le fait que le processus système *init* se charge de l'attente finale correcte de notre processus. Lorsque le traitement sera terminé, nous n'en serons pas avertis à l'écran puisque ce programme n'a plus de terminal de contrôle. Il nous faudra consulter, de temps à autre, la liste des processus pour guetter sa disparition.

```
$ ps -fu mike
  UID    PID  PPID  C   STIME TTY      TIME CMD
mike   428      1  28 10:15:55 ?        0:18 ls -alR /usr
mike   436    431   0 10:16:44 pts/2    0:00 ps -fu mike
mike   431    429   0 10:16:37 pts/2    0:00 -ksh
$
```

b. Contrôle des tâches

Le Korn shell nous permet de stopper momentanément un processus via la touche **Ctrl z** et de relancer ensuite son exécution. Il y a plusieurs possibilités de reprise :

fg	Reprise de la dernière commande suspendue
fg pid	Reprise d'un processus désigné par son <i>PID</i>
fg %n	Reprise d'un processus désigné par son numéro dans la liste des travaux
bg pid	Reprise, en arrière-plan, d'un processus désigné par son <i>PID</i>
bg %n	Reprise, en arrière plan, d'un processus désigné par son numéro dans la liste des travaux

La commande interne **jobs** donne la liste des processus , soit stoppés, soit s'exécutant en arrière-plan. Elle indique un numéro d'ordre exploitable par les commandes *fg* ou *bg* ainsi que par la commande *kill*.

Dans cet exemple, nous démarrons une commande *ls* en premier plan. Cette commande est stoppée via la touche *Ctrl Z*. La commande *jobs* nous permet de visualiser le processus stoppé. Ce processus est ensuite relancé en arrière-plan via la commande *bg*.

```
$ ls -lR / > resul 2> /dev/null
^Z[1] + Stopped (SIGTSTP)      ls -lR / > resul 2> /dev/null
$ jobs
[1] + Stopped (SIGTSTP)      ls -lR / > resul 2> /dev/null
$ bg %1
[1]      ls -lR / > resul 2> /dev/null&
$ jobs
[1] +  Running               ls -lR / > resul 2> /dev/null
$
```


10. Récapitulatif des commandes à approfondir dans la documentation

grep	Recherche d'expressions
kill	Émission d'un signal
nohup	Protection d'une commande contre le signal HUP
ps	État des processus
tee	Duplication de la sortie standard
write	Dialogue avec un autre utilisateur (messagerie instantanée)

2775cbe5ce
Global Knowledge

11. TP 7 - Gestion des process et redirections

Ce qui est fait dans ce TP :

Cet exercice familiarise le stagiaire avec la gestion des process et la redirection des textes utilisés en entrée et en sortie des commandes.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- . Surveiller les process avec les commandes **ps** et **jobs**,
- . Contrôler les process avec les commandes **kill** et **jobs**,
- . Afficher l'ID du process courant.

Introduction

Dans cet exercice, dans un premier temps, vous allez utiliser des commandes pour comprendre l'environnement de vos process. Identifier les process associés à votre terminal. Manipuler des process en tâche de fond et mettre fin aux process que vous avez lancés. Dans un deuxième temps vous provoquez la redirection de l'entrée standard et des sorties d'affichage des commandes, depuis ou vers un fichier ordinaire. Enfin vous lancez plusieurs process sur une seule ligne de commande, interconnectés ou indépendants.

Remarque : Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

.a Process courant

- ___ 1. Ouvrez une session et affichez votre numéro de process courant

- ___ 2. Créez un sous-shell en lançant **bash** (ou **ksh** selon votre environnement). Quel est le PID de ce sous-shell ? Est-il différent de votre process obtenu à votre connexion ? _____

- ___ 3. Lancez la commande **ls -lR / > outfic 2> errfic &** puis affichez la liste des process lancés depuis votre terminal. Repérez la ligne de la commande **ls**. Cette commande **ls** se termine quand elle a fini de lister tous les fichiers de l'arborescence. (Remarque : si **ls** se termine trop vite, alors lancez la commande **sleep 30 &** qui provoque une pause de 30 secondes).
- ___ 4. Terminez votre shell enfant. Qu'est-ce qui se passerait si vous tapiez de nouveau **exit** ? _____

.b Optionnel - Contrôle des process lancés en tâche de fond.

- ___ 5. Avec **vi** créez un script de commandes shell nommé **sctest** qui contient les lignes suivantes:

sleep 30

ls -lR / &

Rendez-le exécutable. Puis lancez la commande

\$./sctest

- ___ 6. La pause de 30 secondes permet de suspendre le job. Suspendez le job que vous venez de lancer.
- ___ 7. Affichez la liste des jobs que vous avez lancés sur le système et relancez en tâche de fond celui que vous avez suspendu ci-dessus.
- ___ 8. Ramenez le job en avant plan.
- ___ 9. Lancez le script **sctest** avec **nohup** et en tâche de fond. Notez le numéro de job, son PID, puis déconnectez-vous.
- ___ 10. Connectez-vous de nouveau (ou bien depuis un autre terminal) et vérifiez que le process est toujours en cours d'exécution.
- ___ 11. Une fois le process terminé, affichez le fichier qui contient le texte de sortie **outfic**. (si vous n'avez pas explicitement redirigé la sortie, alors par défaut le système crée un fichier qui se nomme **nohup.out** dans le répertoire d'où a été

lancée la commande).

.c Terminer un process

___ 12. Lancez en tâche de fond, avec redirection des sorties, la commande **ls -lR /** . Cette commande met du temps avant de se terminer. Notez le PID du process

___ 13. Si vous n'avez pas pu noter le PID après avoir lancé la commande en tâche de fond, comment pouvez-vous le retrouver ? _____

Une fois le PID connu, tuez le process. Vérifiez qu'il a bien disparu.

.d Les redirections

___ 15. Avec la commande **cat** et le mécanisme de redirection, créez un fichier de texte nommé **soleil** qui contient quelques lignes de texte. Utilisez <Ctrl-d> au début d'une nouvelle ligne pour mettre fin à votre saisie et retourner à l'invite shell \$. Affichez le contenu du fichier pour vérifier.

___ 16. Avec la commande **cat** ajoutez plusieurs lignes de texte au fichier **soleil**. Vérifiez vos modifications en affichant le contenu de ce fichier.

___ 17. En utilisant la commande **mail** de la messagerie, envoyez-vous le contenu du fichier **soleil**. Patientez une minute puis ouvrez votre courrier, supprimez-le et quittez votre programme de messagerie.

.e Tubes (pipes) et filtres

___ 18. Affichez le contenu de votre répertoire courant avec la commande **ls**. Notez le nombre de fichiers: _____

___ 19. Listez de nouveau le contenu de votre répertoire courant, mais cette fois redirigez la sortie vers un fichier nommé **temp**.

- ___ 20. Utilisez la commande appropriée pour compter le nombre de mots du fichier **temp**. Est-ce le même nombre qu'à l'étape 18 ? _____ si non pourquoi ? _____

Affichez le contenu du fichier **temp**. Supprimez-le.

- ___ 21. Cette fois utilisez un "tube" "|" (pipe) pour compter le nombre de fichiers du répertoire courant. C'est bien le résultat attendu ? _____

C'est le même nombre qu'à l'étape 18 ?

- ___ 22. Reprenez la ligne de commande lancée à l'étape 21, mais cette fois insérez la commande **tee** entre les deux commandes pour écrire dans un fichier nommé **soleil2**. Est-ce que le nombre de fichier est affiché ?

Vérifiez que le fichier **soleil2** contient bien ce qui est prévu.

- ___ 23. Listez le contenu du répertoire courant en ordre inversé. Envoyez le résultat dans un fichier **soleil3**, et aussi vers une commande pour compter le nombre de mots de la liste inversée. Ajoutez ce nombre au fichier **soleil3**. N'oubliez pas d'utiliser la redirection en mode ajout, sinon vous pourriez avoir des résultats inattendus. On ne peut pas utiliser une simple redirection en création car le fichier est utilisé deux fois dans la même commande globale de la ligne. Vous pouvez tester les conséquences si vous êtes curieux.

- ___ 24. Dans le dossier **/dev** Il y a un fichier spécial qui représente votre terminal. Affichez le nom du fichier qui est associé à votre terminal. Il se présente sous la forme **tty0**, **lft0** ou **pts/x** . Répétez la commande de l'étape précédente, mais avec deux changements :

a. plutôt que d'utiliser le fichier ordinaire **soleil3**, la commande **tee** envoie le résultat à votre écran (**/dev/<nom de votre terminal>**)

b. ne renvoyez pas le résultat de la commande **wc** vers le fichier **soleil3** de façon à ce que le comptage soit envoyé à l'écran.

.f Optionnel - Commandes groupées et annulation de fin de ligne

- ___ 25. Sur la même ligne de commande affichez la date système, qui est connecté, le nom de votre répertoire courant et la liste des fichiers de ce répertoire. Est-ce que ces commandes ont un lien entre elles ?
- ___ 26. Cette étape à deux buts: le premier est d'annuler la fin de ligne de commande pour une commande trop longue pour tenir sur une seule ligne, le second est de tester ce que vous avez appris en vous faisant rédiger une très longue commande.

Vous pouvez choisir de couper la ligne où vous voulez, mais en tout cas ne tapez pas jusqu'à dépasser le bord droit de l'écran. Une fois terminé, tester votre sortie en affichant le contenu des fichiers qui ont été créés. C'est une longue commande reliée par des tubes et une redirection :

- a. Générez une liste détaillée de tous les fichiers du répertoire courant, y compris les fichiers cachés,
- b. récupérez le résultat pour l'enregistrer dans un fichier nommé **listing.inverse** et renvoyez ce même flux de texte vers une commande qui compte uniquement le nombre de mots,
- c. récupérez ce nombre de mots et enregistré le dans quatre fichiers nommés de fic1 à fic4,
- d. enfin, envoyez le résultat précédent vers une commande qui compte le nombre de ligne et renvoyez ce nombre vers un fichier nommé fic5.

Fin de l'exercice

Chapitre 5

Utilisation du shell

2775cbe5ce
Global Knowledge

1. Les différents shells

Le shell est à la fois un **interpréteur de commandes** et un **langage de programmation** permettant l'écriture de procédures utilitaires (**scripts**). Par ce deuxième aspect, il s'avère être un outil de travail important pour l'administrateur système.

Le shell est un **programme extérieur au noyau**. Il réalise l'interprétation d'un certain nombre de caractères spéciaux avant l'appel proprement dit des commandes. Cette caractéristique est essentielle et son intérêt est déjà apparu, dans le chapitre précédent, au niveau de mécanismes tels que les redirections et le *pipeline*.

Lors de la création d'un compte utilisateur, l'administrateur système associe un shell de connexion à cet utilisateur. Les principaux shells disponibles sont les suivants :

- **Bourne shell** (sh ou bsh)
Il s'agit du shell le plus ancien, créé par Steve Bourne. On ne l'utilise pratiquement plus en interactif à cause de son manque de confort (absence d'alias, impossibilité de rappel de commandes...).
- **C shell** (csh)
Ce shell est issu des versions *Unix Berkeley*. Il a été écrit par Bill Joy. Lors de son apparition, il a apporté de nettes améliorations par rapport au Bourne shell, notamment dans le confort interactif (fonctionnalités d'alias et de rappel de commandes).
- **Korn shell** (ksh)
Créé par David Korn, cet interpréteur constitue le shell *standard de fait* dans les versions Unix commerciales. Successeur du Bourne shell dont il étend, de façon compatible, les fonctionnalités de programmation, il propose également à l'utilisateur les améliorations qu'avaient apportées le C shell en son temps. L'activité de scripts en production se réalise aujourd'hui le plus souvent en Korn shell.

- Bourne again shell (bash)

Il s'agit d'un logiciel libre qui est le shell par défaut des distributions Linux. Sa syntaxe est très proche de celle du Korn shell.

Le shell de connexion est choisi par l'administrateur lors de la création du compte de l'utilisateur et il est visible dans le fichier **/etc/passwd** (fichier de description des comptes).



La suite du chapitre se concentre sur le Korn shell, standard de fait des versions Unix actuelles.

2. Variables et environnement

a. Variables

Exceptions faites de certaines variables spéciales utilisées principalement en programmation, le **nom d'une variable** peut être constitué de lettres, de chiffres et du caractère `_` (tiret bas). Il ne peut pas commencer par un chiffre. Une variable contient une **chaîne de caractères** quelconque.

Le caractère `=` (signe égal) est le symbole d'**affectation** de variable et permet la création de celle-ci.

Le caractère `$` permet de désigner la valeur d'une variable. Si la variable n'existe pas, nous obtiendrons la chaîne vide.

La commande **unset** permet d'annuler la définition d'une variable, ce qui est rarement utile.

```
$ var=toto
$ echo $var
toto
$
```

Dans certaines constructions, la désignation de la valeur d'une variable doit se faire à l'aide de la notation **`${ }`** qui permet de bien préciser le nom de la variable par rapport à une chaîne littérale.

```
$ a=pa
$ b=ul
$ echo $a$b
paul
$ echo $a$bette
pa
$ echo $a${b}ette
paulette
$
```

b. Environnement

L'**environnement** désigne l'ensemble des variables qui seront transmises au processus fils lors de sa création via le mécanisme du *fork* (voir chapitre 4). Au niveau du shell, une variable est placée dans l'environnement par la commande **export**. Pour désigner l'environnement, on utilise d'ailleurs quelquefois le terme de *variables exportées*.

Une variable non exportée n'est pas connue dans le processus fils. S'il s'agit d'une variable prédéfinie du shell, elle reprend sa valeur par défaut. Les éventuelles modifications, dans le processus fils, des variables de l'environnement s'opèrent sur des copies locales et n'altèrent donc pas les variables du processus parent. Il y a donc un mécanisme d'héritage du parent vers le fils mais il n'y a jamais de remontée du fils vers le parent.

La commande **env** permet d'afficher la liste des variables faisant partie de l'environnement.

La commande **set** permet d'afficher la liste de toutes les variables, faisant partie ou non de l'environnement.

La commande **export** permet d'insérer des variables dans l'environnement. Elle peut être utilisée seule ou combinée avec le symbole d'affectation. Si nous modifions la valeur d'une variable qui est déjà dans l'environnement, il est inutile de refaire appel à la commande *export*.

```
$ var1=toto
$ export var1
```

```
$ export var2=zorro
$ env | grep var
var1=toto
var2=zorro
$ var2=autre valeur
$ env | grep var2
var2=autre valeur
$
```

c. Quelques variables prédéfinies

La notion de variable concerne surtout l'aspect programmation. Cependant, un certain nombre de **variables prédéfinies** joue un rôle important pour l'utilisateur. Elles peuvent avoir une valeur par défaut ou être initialisées à diverses étapes de la procédure de connexion.

Nous pouvons citer quelques variables importantes.

HOME	Nom complet du répertoire de connexion
LOGNAME	Nom de l'utilisateur
OLDPWD	Nom du répertoire courant précédent (on peut utiliser la commande " cd - " pour y retourner)
PATH	Liste des répertoires pour la recherche des commandes par le shell
PS1	Prompt principal (par défaut : \$)
PS2	Prompt secondaire (par défaut : >)
PWD	Nom du répertoire courant
TERM	Type du terminal (émulation) utilisé notamment par l'éditeur <i>vi</i>
TMOUT	Nombre de secondes d'inactivité du shell avant déconnexion

```
$ PS1='unix> '
unix> echo "Bienvenue au
> club Unix"
```

```
Bienvenue au
club Unix
unix> PS2='suite: '
unix> echo "Bienvenue au
suite: club Unix"
Bienvenue au
club Unix
unix>
```

La variable **TERM** peut nous servir à illustrer la notion d'environnement. Cette variable contient le type du terminal. Elle doit se trouver dans l'environnement car elle sera consultée par les commandes qui vont exploiter les caractéristiques d'affichage, comme par exemple *vi*, *more*, *clear*... La valeur de **TERM** correspond, en effet, au nom d'un fichier administratif dans lequel seront décrites les caractéristiques de gestion du terminal.

Nous commençons par supprimer la définition de la variable **TERM**, ce qui fait échouer la commande *clear*. Nous recréons la variable sans l'insérer dans l'environnement. La commande n'y a toujours pas accès. Nous exportons la variable. La commande y accède mais sa valeur ne correspond pas à une vraie émulation. Nous affectons enfin la bonne valeur et la commande *clear* fonctionne à nouveau.

```
$ echo $TERM
xterm
$ unset TERM
$ clear
TERM environment variable not set.
$ TERM=bidon
$ clear
TERM environment variable not set.
$ export TERM
$ clear
'bidon': unknown terminal type.
$ TERM=xterm
$ clear
```

Effacement correct de l'écran

La variable **PATH** joue un rôle très important. Elle contient une liste de répertoires dans lesquels le shell cherche les commandes dont on ne donne pas le chemin complet. La recherche s'effectue de gauche à droite.

Nous faisons ici l'expérience d'un *PATH* où ne serait pas inclus le répertoire des commandes publiques */usr/bin*. La commande *id* n'est pas trouvée par le shell. Son lancement redevient possible dès que nous complétons notre variable d'environnement. Nous plaçons le répertoire */usr/bin* en tête car il contient la grande majorité des commandes publiques.

```
$ PATH=/usr/local/bin
$ id
ksh: id: not found
$ PATH=/usr/bin:$PATH
$ echo $PATH
/usr/bin:/usr/local/bin
$ id
uid=2000(mike) gid=2000(mike)
$
```



Le répertoire courant n'est pas implicite dans la recherche des commandes. L'utilisateur doit le mentionner explicitement dans la variable PATH. Pour des raisons de sécurité (appel par erreur d'une fausse commande), cette pratique peut être déconseillée. On peut la "tolérer" si le répertoire courant est placé en dernier. En ce qui concerne le compte sensible root, le répertoire courant n'est jamais intégré au PATH.

La commande interne **whence** (ou **which**) permet d'obtenir très rapidement le nom complet d'une commande si celle-ci est trouvée grâce au *PATH*. L'échec de la commande *whence* ne signifie pas que la commande n'est pas présente sur le système. Il se peut tout simplement qu'elle appartienne à un répertoire non intégré pour l'instant à la variable *PATH*. L'exemple ci-dessous nous montre cet aspect par rapport à la recherche de la commande de nom *ifconfig*. La recherche réelle d'un fichier sur le disque pourra se faire, en dernier recours, via la commande *find*, décrite dans le chapitre 6.

```
$ whence date
/usr/bin/date
$ whence ifconfig
$ PATH=$PATH:/sbin
$ whence ifconfig
/sbin/ifconfig
$
```

d. Internationalisation

Dans la mesure où l'administrateur aurait installé les paquets logiciels adéquats, l'utilisateur peut choisir la langue utilisée dans le cadre de sa connexion. La variable d'environnement **LANG** définit une valeur globale pour cette internationalisation. Elle est constituée du nom de la langue associé au pays concerné, éventuellement complété par le nom d'un jeu de caractères.

La langue par défaut, en production, est souvent *en_US* (english, United States) qui s'abrège en *C* (tout ceci est normalisé en langage C). La variable peut contenir des valeurs telles que *fr_FR* (french, FRance), *fr_BE*(french, BElgium) ...

```
$ export LANG=C
$ date
Sun Sep 26 16:42:20 CEST 2004
$ export LANG=fr FR
$ date
dimanche, 26 septembre 2004, 16:42:27 CEST
$
```

D'autres variables, moins souvent utilisées, portent des noms qui commencent par **LC_** (LC_MESSAGES, LC_TIME, LC_NUMERIC...). Elles permettraient de faire des choix différents selon qu'il s'agit de messages d'erreur, de dates, de nombres etc...

```
$ export LANG=C
$ export LC_TIME=fr FR
$ banner
Usage: banner "up to 10 char arg string" . . .
$ date
dimanche, 26 septembre 2004, 16:45:46 CEST
$
```

Dans cet exemple, les commandes Unix utilisent l'anglais, à l'exception des commandes qui donnent des informations correspondant à la variable **LC_TIME**.

3. Caractères spéciaux

Comme nous avons déjà eu l'occasion de le souligner, le shell interprète un certain nombre de caractères avant l'appel proprement dit des commandes.

Ce principe d'interprétation préliminaire de **caractères spéciaux** permet de faciliter l'utilisation des commandes, sans changement, dans un nombre varié de contextes (redirections, pipeline, arrière-plan...).

La liste d'arguments transmis aux commandes pourra également comporter des caractères **génériques** (ou **jokers**) pour désigner de façon abrégée des noms de fichiers. Ces caractères seront également traités par le shell afin de fournir aux commandes une liste de paramètres déjà résolue. Ce principe favorise grandement l'activité de programmation.

Tous ces aspects cumulés s'inscrivent parfaitement dans la démarche *boîte à outils* voulue par les concepteurs du système.

a. Rappel des caractères spéciaux déjà évoqués

<	Redirection de l'entrée standard
> et >>	Redirections de la sortie standard (écrasement ou ajout)
2> et 2>>	Redirections de l'erreur standard (écrasement ou ajout)
;	Processus séquentiels
	Mécanisme du <i>pipeline</i>
&	Mode arrière-plan
()	Groupement de commandes
=	Affectation de variable
\$	Contenu de variable

b. Désignations abrégées de noms de fichiers (jokers, caractères génériques)

Ces caractères spéciaux, souvent baptisés *jokers* ou *caractères génériques*, prennent un sens pour désigner des noms de fichiers existants. Si ces caractères ne sont pas résolus (aucun fichier ne leur correspond), ils sont, malgré tout, transmis tels quels à la commande.

Les caractères génériques sont les suivants :

*	Le caractère * remplace n'importe quelle suite de caractères (même vide) dans le nom de fichier. Utilisé seul, il désigne tous les noms de fichiers du répertoire courant, excepté ceux qui commencent par . (point).
?	Le caractère ? indique la présence d'un caractère quelconque dans le nom du fichier.
[]	Les crochets permettent de préciser un ensemble de caractères. Le signe - (moins) permet d'indiquer un intervalle dans le code ascii (exemple: [a-z] désigne une lettre minuscule). Cette notation indique la présence, dans le nom du fichier, d'un caractère parmi ceux cités dans les crochets.
[!]	Cette notation indique la présence, dans le nom du fichier, d'un caractère non cité dans les crochets.

```
$ echo *
fic1 fic11 fic2 titi toto zorro
$ echo .*
. .. .exerc .profile .sh history
$ echo .* *
. .. .exerc .profile .sh history fic1 fic11 fic2 titi toto zorro
$
```

L'affichage d'une liste de noms de fichiers au lieu du caractère ***** ne doit pas nous surprendre. En effet, le caractère générique ***** est traité par le shell. Il est remplacé par tous les noms de fichiers du répertoire courant avant l'appel de la commande *echo*. Celle-ci ne fait qu'afficher les arguments qui lui sont transmis. Nous constatons également que la notation **.*** est nécessaire pour sélectionner aussi les noms de fichiers qui commencent par **.** (point).

```
$ echo .[!.*]*
.exerc .profile .sh history
$
```

Les deux références implicites de répertoire (**.** et **..**) sont rarement souhaitées dans une liste d'arguments. Nous pouvons les éliminer via la notation ci-dessus qui sélectionne les noms qui commencent par un point suivi d'au moins un caractère qui n'est pas un point.

En combinant astucieusement les caractères génériques, nous pouvons assez souvent désigner une liste de noms de façon très abrégée.

```
$ echo *
fic1  fic11  fic2  toto  zorro  titi
$ echo fic*
fic1  fic11  fic2
$ echo fic?
fic1  fic2
$ echo *[12]
fic1  fic11  fic2
$ echo t[!o]*
titi
$ echo ?i*
fic1  fic11  fic2  titi
$ echo s*
s*
$
```

Le dernier exemple met l'accent sur le fait que le shell appelle les commandes même si les jokers ne sont pas résolus. En effet, il n'y a pas de fichier dont le nom commence par s et l'argument s* est, malgré tout, transmis à la commande `echo`.



Lorsque nous utilisons des jokers dans les arguments d'une commande, il est intéressant de visualiser d'abord la liste résolue avec la commande `echo`. Si nous constatons que la liste est bien celle attendue, nous pourrions ensuite remplacer `echo` par le nom de la commande souhaitée.

c. Substitutions de commandes

Le shell nous permet d'insérer dans la liste d'arguments d'une commande, le résultat d'une autre commande simple ou même d'un *pipeline*. Cette possibilité est surtout utile dans un contexte de programmation (script).

Deux syntaxes sont disponibles :

``commande``

La commande est encadrée par deux ` (quotes inverses). Il s'agit de la syntaxe historique du Bourne shell.

\$(commande)

Il s'agit d'une syntaxe supplémentaire apparue en Korn shell. La commande est encadrée de parenthèses précédées du caractère **\$** (dollar). Cette deuxième forme fait mieux apparaître l'analogie avec la substitution de variable.

```
$ echo "Nombre de fichiers de $PWD : $(ls | wc -w)"
Nombre de fichiers de /home/mike :          6
$
```

Avant d'appeler la commande *echo*, le shell va constituer la liste d'arguments avec successivement, le contenu de la variable *PWD* (nom du répertoire courant) et le résultat du pipeline *ls | wc -w* (nombre de fichiers).

d. Caractères de protections

Les caractères de protection (ou d'échappement) sont destinés à empêcher l'interprétation par le shell de certains autres caractères spéciaux.

Les différentes possibilités sont les suivantes :

**** Le caractère **** (antislash) permet d'annuler l'interprétation du seul caractère placé derrière lui.

' ' Tous les caractères situés entre deux simples quotes perdent leur rôle fonctionnel (excepté bien sûr la quote elle-même).

" " Tous les caractères situés entre deux doubles quotes perdent leur rôle fonctionnel, excepté la double quote elle-même et les trois caractères suivants : **\$**, **`commande`** et ****.

```
$ echo *
fic1 fic11 fic2 titi toto zorro
$ echo \*
*
$ echo \\
```

\
\$

Si le caractère `*` est précédé d'un `\`, il ne joue plus le rôle d'un joker. Le caractère `\` doit être doublé si nous souhaitons le désigner en tant que tel.

```
$ echo <$LOGNAME>
ksh: syntax error: `newline or ;' unexpected
$ echo '<$LOGNAME>'
<$LOGNAME>
$ echo "<$LOGNAME>"
<mike>
$
```

Dans le premier essai, les caractères `<` et `>` tentent d'être interprétés comme des redirections, ce qui provoque une incohérence syntaxique. Dans le deuxième essai, les simples quotes annulent malencontreusement tous les caractères spéciaux, y compris le `$`. Le troisième essai remplit son office puisque les redirections sont banalisées tandis que `$LOGNAME` donne bien le nom de l'utilisateur courant.



Les doubles quotes (les guillemets) sont faites pour être utilisées avec la commande `echo`. En effet, cette commande a souvent pour objectif d'afficher un contenu de variable (avec `$`) ou un résultat de commande (avec les ```). Tous les autres caractères spéciaux peuvent faire l'objet d'un affichage sans perturbation.

```
$ echo "<\\$LOGNAME\\$>"
<\mike$>
$
```

4. TP 8 – Éléments du Shell, variables et jokers

Ce qui est fait dans ce TP :

Cet exercice familiarise le stagiaire avec les opérations de base du shell.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- Afficher les variables prédéfinies du shell,
- Déclarer des variables et les exporter pour quelles soient transmises aux process enfants,
- Utiliser les jokers appelés aussi caractères spéciaux ou génériques,
- Utiliser la substitution de commande pour affecter le contenu d'une variable
- Pratiquer les trois caractères : apostrophe (quote), guillemets (double quote) et « anti-quote », pour l'utilisation littérale ou interprétée des caractères spéciaux .

Introduction

L'usage du shell (on dit aussi langage shell) est considéré comme une des bases de la compréhension de l'interface utilisateur d'Unix. Vous allez utiliser les commandes et les mécanismes découverts auparavant pour mettre en œuvre les mécanismes de base du shell.

.a Variables

- ___ 1. Afficher les variables déclarées dans votre shell. Contrôlez l'affichage avec la commande **more**.
- ___ 2. Déclarez une variable nommée **dejeuner** qui contient la valeur **pizza**. Et une variable **diner** qui contient **jambon**. Avec la commande **echo** affichez le contenu des variables. Retrouvez-les dans la liste des variables du shell courant.

- ___ 3. En utilisant les variables que vous venez de définir, affichez le message : **Ce jour au déjeuner il y a pizza et au diner jambon.**
- ___ 4. Supprimez les deux variables. Assurez-vous qu'elles ne sont plus dans la liste des variables du process courant.
- ___ 5. Affichez les valeurs de vos deux invites (prompt en anglais) de ligne de commande.
- ___ 6. Modifiez la première chaîne d'invite par **"vous desirez ? "**. Pourquoi faut-il des guillemets (avec des apostrophes cela fonctionne aussi) pour **"vous desirez ? "** _____
- ___ 7. Modifiez le second prompt (invite) par : **"Quoi d'autre ? "**. Testez en tapant une ligne de commande comme **ls -l**, mais en annulant le caractère de fin de ligne. Une fois fait, réaffectez les variables d'invite avec leurs valeurs initiales. Pourquoi faut-il utiliser les guillemets autour du signe > pour la variable PS2 ? _____
- ___ 8. Vérifiez la valeur de la variable qui contient le chemin à votre répertoire d'accueil. Modifiez la pour que votre répertoire d'accueil soit **/bin**. Utilisez les commandes **cd** et **pwd** pour constater les effets.
- ___ 9. Fermez votre session et reconnectez-vous. Quel est votre répertoire d'accueil ?
_____ Pourquoi ? _____

.b Jokers

- ___ 10. Tapez **cd** pour retourner dans votre répertoire d'accueil. (celui dans lequel vous étiez placé lors de votre connexion).
- ___ 11. Exécutez un simple **ls** pour afficher la liste des fichiers non cachés de votre répertoire d'accueil. Puis lancez de nouveau **ls** suivi du caractère joker pour lister ces mêmes fichiers. Quelles sont les différences d'affichage de ces deux commandes ?

Pourquoi ?

- ___ 12. Déplacez-vous dans le répertoire **/usr/bin**. Affichez la liste des fichiers dont le nom commence par la lettre **a**.

- ___ 13. Affichez la liste des fichiers dont le nom est composé de deux caractères.

- ___ 14. Affichez la liste des fichiers dont le nom commence par une des lettres suivantes : **a**, **b**, **c** ou **d**.

- ___ 15. Affichez la liste de tous les fichiers excepté ceux dont le nom commence par une lettre comprise entre **c** et **t**. Cette liste est très longue. Vous pouvez contrôler l'affichage en utilisant **more** (ou **pg**) grâce à un pipe.

- ___ 16. Retournez dans votre répertoire d'accueil.

.c Substitution de commande

- ___ 17. Affichez la liste des sessions ouvertes, i.e. des utilisateurs connectés et les terminaux qui leurs sont attribués. Relancez votre commande mais cette fois en transmettant le résultat à la commande **wc** pour compter le *nombre* de sessions en cours.

- ___ 18. En utilisant la substitution de commande, affichez le texte suivant : **Il y a # sessions en cours** où **#** représente le nombre de sessions en cours.

- ___ 19. Chaque compte utilisateur du système est représenté par une ligne dans le fichier **/etc/passwd**. En utilisant vos connaissances sur la substitution de commande, affichez le message suivant : **Il y a # utilisateurs de créés sur le système**, où **#** représente le nombre d'entrées dans **/etc/passwd**.

.d Environnement d'un process, export de variable

- ___ 20. Affichez toutes les variables d'environnement du process courant.
- ___ 21. Créez une variable **x** initiée à **10**. Vérifiez sa valeur et affichez de nouveau vos variables d'environnement.
- ___ 22. Créez un sous shell avec **ksh** (ou **bash**). Vérifiez si la variable **x** est connue dans le sous shell courant. Quelle est la valeur de **x** ? ____ Affichez la liste des variables du sous shell. Voyez vous la variable **x** ?
- ___ 23. Retournez dans votre process parent. Modifiez la description de la variable **x** de façon à ce quelle soit héritée par les process enfants. Vérifiez que la modification est prise en compte en créant un sous shell et dans ce nouveau process affichez de nouveau la valeur de **x**.
- ___ 24. Toujours dans le sous shell, changez la valeur de **x** à 200. Vérifiez que la valeur est changée.
- ___ 25. Retournez dans le process parent. Dans cet environnement vérifiez la valeur de **x**. Est-ce que la modification faite dans le sous shell a été exportée en amont vers le parent ? ____
- ___ 26. Créez un script shell nommé **sc1**. Son contenu :
- ```
pwd ; cd / ; pwd
```
- \_\_\_ 27. Modifiez les droits d'accès de **sc1** pour qu'il soit exécutable et lancez ce programme. Une fois le script terminé, dans quel répertoire êtes vous ? \_\_\_\_ Pourquoi ? \_\_\_\_
- \_\_\_ 28. Créez un autre script shell nommé **sc2**. Son contenu :
- ```
var1=bonjour ; var2=$LOGNAME ; export var1 var2
```

- ___ 29. Modifiez les droits de **sc2** pour qu'il soit exécutable puis lancez ce nouveau programme. Quand il est terminé, examinez le contenu des variables **var1** et **var2**. Quelles sont leurs valeurs ? _____ Pourquoi ? _____
- ___ 30. Relancez de nouveau le script **sc2**, mais cette fois forcez son exécution dans le process shell courant. Lorsqu'il a fini de s'exécuter, vérifiez les valeurs de **var1** et de **var2**. Quelles sont leurs valeurs ? _____ Pourquoi ? _____

.e Apostrophes, Guillemets et banalisation explicite (antislash)

- ___ 31. Utilisez trois façons pour affichez le caractère astérisque * en utilisant la commande **echo**.
- ___ 32. Assurez-vous d'être dans votre répertoire d'accueil. Créez un dossier nommé **quote**.
- ___ 33. Déplacez-vous dans le dossier **quote**. Créez un fichier vide nommé **fic**. Créez une variable nommée **n** qui contient **bonjour**. Vérifiez ce que vous avez fait en listant le contenu du dossier **quote** puis affichez le contenu de la variable **n**.
- ___ 34. Optionnel - Depuis le dossier **quote**, lancez les cinq commandes suivantes. Notez les résultats. Attention à bien différencier l'apostrophe (') de l'anti-apostrophe (`) <Alt-Gr> 7.

i.\$ **echo** '* \$n `ls` \$(ls) '

ii.\$ **echo** "* \$n `ls` \$(ls) "

iii.\$ **echo** * \\$n `ls` \\$\ (ls\)

iv.\$ **echo** * \$n `ls` \$(ls)

v.\$ **echo** * \$n ls

Fin de l'exercice

5. Fonctionnalités interactives

a. Alias

Le mécanisme d'alias permet de définir des abrégés pour des commandes souvent utilisées. Il permet aussi de surcharger des commandes, c'est-à-dire de forcer l'utilisation de certaines options.

La commande **alias** permet de définir un alias. Invoquée sans argument, elle donne la liste des définitions existantes.

```
$ alias l='/usr/bin/ls -l'
$ alias l
l='/usr/bin/ls -l'
$ l
total 6
-rw-r--r--  1 mike   mike           0 Sep 26 17:38 fic1
-r--r--r--  1 mike   mike        916 Sep 26 17:38 fic2
-rw-r--r--  1 mike   mike       106 Sep 26 17:38 toto
-r--r--r--  1 mike   mike       916 Sep 26 17:38 zorro
$
```

Nous obtenons ici un abrégé **l** pour exécuter, en fait, la commande **ls -l**.

Le shell examine les définitions d'alias avant le parcours des répertoires du PATH. La commande **type** permet de savoir si un nom de commande est un alias, une commande interne du shell ou bien une commande classique (fichier ordinaire exécutable).

```
$ type type
type is a shell builtin
$ type cd
cd is a shell builtin
$ type l
l is an alias for /usr/bin/ls -l
$ type cal
cal is /usr/bin/cal
$
```

La commande **unalias** permet la suppression d'un alias.

```
$ unalias l
$ l
ksh: l: not found
$
```

La surcharge de commande force l'utilisation systématique de certaines options. Elle est utile pour sécuriser l'emploi de certaines commandes sensibles.

```
$ alias rm='/usr/bin/rm -i'
$ rm fic1
rm: remove fic1 (yes/no)? n
$ alias cp='/usr/bin/cp -i'
$ cp fic1 toto
cp: overwrite toto (yes/no)? n
$
```

b. Historique des commandes

Par défaut, le Korn shell mémorise les 128 dernières commandes tapées par l'utilisateur dans un fichier **.sh_history** (ou **.bash_history** pour **bash**) créé dans le répertoire de connexion. Ce fichier est conservé au travers des connexions successives.

Deux variables d'environnement (rarement modifiées) permettraient de changer le comportement par défaut :

HISTFILE

Nom du fichier d'historique

HISTSIZE

Nombre de commandes mémorisées

Grâce à ce fichier, le shell propose diverses méthodes de rappel des commandes.

c. Rappel simple de commandes

La commande de gestion de l'historique s'appelle **fc**. Elle comporte beaucoup d'options. Pour faciliter son utilisation, le shell définit un certain nombre d'alias internes.

Affichage des dernières commandes

fc -l	Afficher les 16 dernières commandes avec un numéro d'ordre
history	Alias prédéfini pour <i>fc -l</i> (on pourra se définir un autre alias plus court : <i>alias h=history</i>)
fc -l n1 n2	Afficher les commandes de numéro <i>n1</i> à <i>n2</i>

Relancer des commandes

r	Relancer la dernière commande (r est un alias pour <i>fc -e -</i>)
r num	Relancer la commande d'indice <i>num</i> dans la liste
r string	Relancer la commande la plus récente commençant par <i>string</i>
r st1=str2	Relancer la dernière commande en remplaçant <i>str1</i> par <i>str2</i>

d. Rappel et édition de commandes avec l'éditeur intégré

Une connaissance minimale de l'éditeur *vi* est nécessaire pour savoir rappeler efficacement des commandes. En effet, le Korn shell, via la fonctionnalité baptisée *éditeur intégré*, permet de modifier la commande en cours de saisie ou, bien entendu, toute commande déjà tapée et présente dans le fichier d'historique.

Pour activer cette possibilité, nous devons, au choix, positionner la variable d'environnement *EDITOR* ou positionner une option au niveau du shell :

```
$ EDITOR=vi ; export EDITOR
```

ou bien

```
$ set -o vi
```

La frappe de la touche **Escape** au niveau du shell déclenche alors ce mode *éditeur intégré* sur le fichier d'historique et permet l'édition des commandes avec tous les ordres de l'éditeur *vi*.



*Il faut signaler que les déplacements par les flèches sont rarement opérationnels et qu'il est nécessaire d'utiliser les véritables commandes : *h* , *j* , *k* , *l*.*

L'éditeur de texte **emacs** est quelquefois disponible sur les systèmes Unix. Il est possible de l'activer en tant qu'éditeur intégré en utilisant la syntaxe : **set -o emacs**.

Les équivalents des déplacements *h*, *j*, *k*, *l* sont obtenus respectivement par les touches *Ctrl b*, *Ctrl n*, *Ctrl p* et *Ctrl f*. La modification de lignes est intuitive puisque l'utilisateur se trouve naturellement en mode insertion. En ce qui concerne les effacements, la touche *backspace* permet d'effacer le caractère précédent tandis que la touche *Ctrl d* efface le caractère courant.

e. Rappel et édition de commandes avec la commande interne **fc**

La commande **fc** de gestion de l'historique permet aussi d'éditer un ensemble de commandes dans un fichier temporaire. Pour activer cette possibilité, nous devons positionner la variable d'environnement *FCEDIT* avec le nom complet de l'éditeur choisi :

```
FCEDIT=$(whence vi) ; export FCEDIT
```

Il convient, dans un premier temps, d'obtenir le groupe de commandes souhaité.

fc	Seulement la dernière commande
fc n	Seulement la commande de numéro <i>n</i> (liste obtenue par <i>history</i>)
fc n1 n2	Les commandes de numéro <i>n1</i> à <i>n2</i> (liste obtenue par "history")

Après d'éventuelles modifications sur ce groupe de commandes, la sortie de l'éditeur en déclenche l'exécution. Il convient d'être vigilant dans cette manipulation car les commandes seront relancées même si nous quittons l'éditeur sans sauvegarder. Les commandes non souhaitées doivent donc absolument être supprimées du fichier temporaire.

6. Fichiers de connexion

Lors de la connexion, le shell exécute les commandes mentionnées dans des fichiers de paramétrage destinés à initialiser des variables d'environnement et à activer diverses fonctionnalités.

Le Korn shell exécute, en premier lieu, un fichier **/etc/profile**, géré par l'administrateur et commun à tous les utilisateurs. Ce fichier permet, entre autres, l'affichage d'informations générales, la recherche de courrier, le positionnement de certaines variables d'environnement.

Dans un deuxième temps, le shell exécute le contenu d'un éventuel fichier personnel **.profile**, situé dans le répertoire de connexion.

Parmi les tâches usuelles effectuées dans ce fichier, nous pouvons citer :

- le paramétrage du *prompt* (variables PS1 et PS2)
- le positionnement de certaines variables d'environnement
- le positionnement de certaines options du shell
- un appel éventuel à la commande *umask* (choix des droits par défaut)

En ce qui concerne l'activation de certaines fonctionnalités interactives propres au Korn shell (alias, éditeur intégré ...), il est recommandé d'utiliser un second fichier dont le nom (traditionnellement **.kshrc**) doit être indiqué dans la variable d'environnement **ENV**.

Ce deuxième fichier est exécuté après le fichier *.profile* lors de la connexion. Par contre, si le shell est démarré de façon interactive, le fichier *.profile* ne sera pas exécuté, au profit de ce seul deuxième fichier.

✎ *Il ne faut pas confondre la commande `env` avec la variable `ENV` qui contient donc le nom d'un fichier exécuté seul lors du lancement interactif du shell ou exécuté après le fichier *.profile* lors d'une véritable connexion.*

Exemple de fichier *.profile*

```
# personnaliser les prompts
PS1="`hostname`.\$PWD\$ "
PS2="suite: "
# completer son PATH
PATH=$PATH:.
# se proteger contre les ecrasements via des redirections
set -o noclobber
# choix des droits par default
umask 027
# choix d'un second fichier (lu apres le .profile)
ENV=$HOME/.kshrc ; export ENV
```

Exemple de fichier *.kshrc*

```
# mes alias
```

```
alias h=history
alias l='/usr/bin/ls -l'
alias rm='/usr/bin/rm -i'
alias cp='/usr/bin/cp -i'
alias mv='/usr/bin/mv -i'
alias p='/usr/bin/ps -fu $LOGNAME'
# historique des commandes
set -o vi
FCEDIT=$(whence vi) ; export FCEDIT
```

a. Prise en compte des modifications

Pour activer des modifications dans ces fichiers personnels, nous pouvons, bien entendu, effectuer une déconnexion puis une reconnexion immédiate. Pour éviter cela, nous pourrions utiliser la **commande interne** `.` (point) qui force l'interprétation du fichier dans le processus courant, sans création de fils.

Ceci est indispensable pour permettre la prise en compte d'éventuelles modifications de variables d'environnement. En effet, rappelons que s'il y a bien un mécanisme d'héritage du parent vers le fils, il n'y a jamais de remontée du fils vers le parent.

```
$ . .profile
ksh: .profile: not found
$ . ./profile
$
```

Assez souvent, l'utilisateur ne possède pas le répertoire courant dans son *PATH*. Dans ce cas, la première syntaxe échoue car la *commande* `.` (point) n'a pas su localiser le fichier *.profile*. La seconde syntaxe s'impose alors puisque nous y indiquons explicitement que le fichier se trouve dans le répertoire courant.

Pour résumer, le premier point désigne la commande interne, le second désigne le répertoire courant et le troisième n'est que le premier caractère du nom du fichier !!

7. TP 9 – Personnaliser son environnement

Ce qui est fait dans cet exercice :

Généralement lors de l'ouverture d'une session, les utilisateurs préfèrent personnaliser leur environnement. Dans cet exercice vous allez personnaliser votre environnement par quelques fonctionnalités usuelles qui sont prises en compte à chaque connexion.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- . Personnaliser les fichiers **.profile**, **.kshrc** ou **.bashrc**,
- . Définir des **alias**,

Introduction

A travers cet exercice vous modifiez votre invite (prompt en anglais) par défaut – le caractère **\$**, par une chaîne qui indique votre répertoire courant. Si vous travaillez en environnement graphique XWindows, alors plutôt que de vous déconnecter puis vous reconnecter pour tester vos modifications, simulez une nouvelle connexion en lançant explicitement la commande **login**.

Si c'est le Korn Shell **ksh** qui vous est attribué lors de votre connexion, alors vous mettez à jour le fichier **.kshrc**. Si c'est le Bourne Again Shell **bash** qui est lancé alors c'est le fichier **.bashrc** qui doit être modifié. Ces deux versions de shell présentent des fonctionnalités plus avancées (notamment l'affichage dynamique du répertoire courant à l'invite) que le shell Bourne historique. Ce dernier lui prend en compte le fichier **.profile** à la connexion.

Remarque : Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

.a Personnalisation de .kshrc ou .bashrc

- ___ 1. Pour personnaliser votre environnement à chaque connexion, autrement dit définir votre profile d'utilisateur, vous devez mettre à jour le fichier qui est lu à chaque connexion. Assurez-vous d'être dans votre répertoire d'accueil et selon le shell qui vous est attribué, modifiez soit le fichier **.kshrc**, soit **.bashrc** en ajoutant les fonctions suivantes :
- ___ a. modifiez l'invite de commande pour qu'à la place du **\$** par défaut , ce soit le nom du répertoire courant concaténé à '>_' qui apparaisse.
- ___ b. affichez un message de bienvenue personnalisé avec votre nom de login et la date et l'heure système.
- ___ c. déclarez un **alias dir** qui lance la commande **ls -l**,
- ___ d. activez l'édition de l'historique des commandes grâce aux commandes de l'éditeur **vi**.
- ___ 2. Testez votre personnalisation en ré-exécutant votre fichier de profile. Pour ce faire deux possibilités : soit vous vous déconnectez puis reconnectez, soit vous lancez l'interprétation du fichier profile avec la notation du point. Une fois cela fait, quelques questions :
- ___ a. est-ce que le message de bienvenue est affiché ?
- ___ b. est-ce que le nom du répertoire courant est affiché dans l'invite de commande ?
- ___ c. déplacez-vous dans le répertoire **/etc**. Est ce que l'invite est mise à jour ? (remarque : ceci ne serait pas fonctionnel avec le shell Bourne historique),
- ___ d. utilisez **dir** pour afficher la liste détaillée du répertoire courant,
- ___ e. pouvez-vous relancer **dir** en rappelant la dernière commande de l'historique ?

Si vous répondez non à au moins une des questions, alors rééditez votre fichier **.kshrc** ou **.bashrc** et corrigez le problème ...

.b Gestion des alias

- ___ 3. Affichez la liste des **alias** de votre environnement

- ___ 4. Supprimez **dir** de vos alias

- ___ 5. Tentez de relancer **dir**.

- ___ 6. L'alias **dir** est toujours défini dans votre fichier de connexion, mais n'est plus défini dans votre environnement. Pour qu'il soit de nouveau déclaré, soit vous vous déconnectez / reconnectez, soit vous relancez votre fichier dans votre shell courant par la syntaxe du point.

Fin de l'exercice.

2775cbe5ce
Global Knowledge

Chapitre 6

Sélection de commandes

2775cbe5ce
Global Knowledge

Ce chapitre est une sélection de commandes, utiles au quotidien, qui s'ajoutent à celles déjà rencontrées dans les chapitres précédents. Nous les présentons surtout via des exemples. Elles sont donc volontairement relativement peu détaillées en terme de syntaxe et d'options. Pour une étude plus complète, le lecteur se reportera à la documentation en ligne (commande **man**).

Pour exploiter au mieux Unix, nous devons pouvoir recenser les commandes importantes et en mémoriser le nom et le rôle. Au fur et à mesure de la pratique, nous nous souviendrons des options les plus usuelles et nous saurons notamment construire des *pipelines* intéressants. Il sera, par la suite, aisé de retrouver ou d'approfondir des options plus avancées, si le besoin se présente.

Les exemples proviennent d'une version Solaris 9 par souci d'homogénéité.

1. Commandes complémentaires sur les fichiers

a. La commande file

La commande **file** lit le début des fichiers pour essayer d'en déterminer le type de contenu.

```
$ file toto titi rep
toto:          empty file
titi:          ascii text
rep:           directory
$ file /usr/sbin/*
/usr/sbin/6to4relay: ELF 32-bit LSB executable 80386 Version 1,
dynamically linked, stripped
/usr/sbin/accept:   executable shell script
/usr/sbin/acctadm:  ELF 32-bit LSB executable 80386 Version 1,
dynamically linked, stripped
.....
```

 *La commande file nécessite le droit de lecture sur le fichier concerné.*

L'option **-h** est intéressante car elle permet d'identifier les liens symboliques.

```
$ file /bin
/bin:          directory
$ file -h /bin
/bin:          symbolic link to ./usr/bin
$
```

b. La commande nl

La commande **nl** permet de numéroté les lignes de façons variées grâce à de nombreuses options.

```
$ nl /etc/passwd | tail
14  mike:x:2000:2000::/home/mike:/usr/bin/ksh
15  mysql:x:50:50::/home/mysql:/bin/sh
16  stage1:x:5001:5000::/home/stage1:/usr/bin/ksh
17  stage2:x:5002:5000::/home/stage2:/usr/bin/ksh
18  stage3:x:5003:5000::/home/stage3:/usr/bin/ksh
19  stage4:x:5004:5000::/home/stage4:/usr/bin/ksh
20  stage5:x:5005:5000::/home/stage5:/usr/bin/ksh
21  stage6:x:5006:5000::/home/stage6:/usr/bin/ksh
22  stage7:x:5007:5000::/home/stage7:/usr/bin/ksh
23  stage8:x:5008:5000::/home/stage8:/usr/bin/ksh
$
```

Nous obtenons ici les 10 dernières lignes (*tail*) du fichier */etc/passwd*, préalablement numérotées par la commande *nl*.

c. La commande cmp

La commande **cmp** compare le contenu de deux fichiers et affiche la position de la première différence. Si elle n'affiche rien, cela signifie que les deux fichiers sont identiques. Il s'agit là d'un moyen très pratique et usuel de s'en assurer. Le code retour `$?` (voir chapitre 4) est positionné comme il se doit (valeur 0 si les deux fichiers sont identiques).

```
$ cp f1 f2
$ cmp f1 f2
$
```

Les deux fichiers *f1* et *f2* sont identiques. La commande *cmp* est donc muette.

```
$ cat f1 >> f2
$ cmp f1 f2
cmp: EOF on f1
$
```

La commande *cmp* nous indique maintenant que le contenu du fichier *f1* constitue le début du fichier *f2*. Malgré ce début commun, les deux fichiers sont différents. La commande rend un code retour non nul.

d. Diverses commandes complémentaires

Nous pouvons citer d'autres commandes intéressantes pour la manipulation des fichiers :

comm	Comparaison de fichiers triés
diff	Comparaison de fichiers
fold	Découpage de lignes
join	Jointure entre deux fichiers triés
od	Visualisation de fichiers sous différents formats
paste	Concaténation de lignes entre plusieurs fichiers
pr	Mise en forme de fichiers pour impression
split	Éclatement d'un fichier en plusieurs fichiers

2. La commande find

La commande **find** est très utile au quotidien. Elle permet la recherche de fichiers, suivant de multiples critères, à partir d'un ou plusieurs répertoires. La recherche a lieu dans toute la sous-arborescence.

Syntaxe résumée

`find repertoire(s) options actions`

Quelques options

-name fichier	Recherche sur le nom
-perm mode	Recherche sur les permissions
-newer fichier	Recherche des fichiers plus récents qu'un fichier donné
-user login	Recherche sur le propriétaire
-mtime +n	Fichiers modifiés depuis plus de <i>n</i> jours
-mtime -n	Fichiers modifiés depuis moins de <i>n</i> jours
-atime +n	Fichiers consultés depuis plus de <i>n</i> jours
-atime -n	Fichiers consultés depuis moins de <i>n</i> jours
-inum num	Recherche sur un numéro d'inode
-type t	Recherche sur le type de fichier (<i>t = b, c, d, f ou l</i>)

Actions (exclusives)

-print	Affichage du nom des fichiers
-exec	Exécution d'une commande pour chaque fichier trouvé
-ok	Exécution d'une commande avec demande de confirmation

```
$ pwd
/home/mike
$ echo .??*
.Xauthority .profile .sh history
$$ find /home -name .??* -print
find: bad option .Xauthority
find: path-list predicate-list
$ find /home -name '.??*' -print
/home/mysql/.profile
/home/mike/.profile
/home/mike/.sh history
/home/mike/.Xauthority
/home/stage1/.profile
/home/stage2/.profile
/home/stage3/.profile
/home/stage4/.profile
/home/stage5/.profile
/home/stage6/.profile
/home/stage7/.profile
/home/stage8/.profile
$
```

Nous voulons chercher, à partir du répertoire */home*, les fichiers dont le nom commence par un point, en éliminant les références de répertoire. Nous utilisons, pour ce faire, la notation *.??**.

Dans le premier essai, les caractères *jokers* *?* et *** sont tout d'abord traités, comme il se doit, par le shell. Nous sommes positionnés dans un répertoire où plusieurs fichiers correspondent à ces caractères spéciaux. Le shell appelle donc la commande *find* avec plusieurs noms de fichiers derrière l'option *-name*, ce qui provoque une erreur de syntaxe.

Dans le deuxième essai, nous prenons soin de protéger les caractères spéciaux qui seront donc transmis tels quels et ne perturberont pas l'option *-name* qui doit être suivie d'un seul argument. La commande *find* est une des rares commandes Unix qui va traiter, elle-même, ces *jokers*.

- ☞ *L'exemple précédent insiste sur la bonne utilisation de l'option -name. Il est important de penser à protéger d'éventuels caractères spéciaux utilisés dans la recherche de noms de fichiers afin d'empêcher le traitement préliminaire au niveau du shell.*

La commande *find* accepte plusieurs noms de répertoire avant les options. Ceci permet d'effectuer une recherche à plusieurs endroits de l'arborescence Unix. Nous cherchons les fichiers dont le nom est constitué de trois lettres exactement et se termine par *sh*. Nous protégeons, par précaution, le caractère spécial *?*. Dans le deuxième essai, nous décidons d'éliminer les messages d'erreur.

```
$ find /etc /usr/bin -name '?sh' -print
find: cannot read dir /etc/inet/secret: Permission denied
find: cannot read dir /etc/sfw/private: Permission denied
/etc/ssh
/usr/bin/csh
/usr/bin/jsh
/usr/bin/ksh
/usr/bin/rsh
/usr/bin/ssh
/usr/bin/zsh
$ find /etc /usr/bin -name '?sh' -print 2> /dev/null
/etc/ssh
/usr/bin/csh
/usr/bin/jsh
/usr/bin/ksh
/usr/bin/rsh
/usr/bin/ssh
/usr/bin/zsh
$
```

Nous recherchons maintenant, à partir du répertoire */home*, les fichiers qui se nomment *.profile* **et** qui appartiennent à l'utilisateur *mike*. Pour ce faire, il suffit de juxtaposer les deux options correspondantes.

```
$ find /home -name .profile -user mike -print
/home/mike/.profile
$
```

Nous recherchons maintenant, à partir du répertoire */home*, les fichiers qui se nomment *.profile* **ou** qui appartiennent à l'utilisateur *mike*. Le *ou* est obtenu par l'opérateur *-o*. Celui-ci n'est opérationnel qu'entre des parenthèses qu'il nous faut protéger au niveau du shell par le caractère **. Toutes ces contraintes nous donnent une syntaxe peu agréable.

```
$ find /home \( -name .profile -o -user mike \) -print
/home/mysql/.profile
/home/mike
/home/mike/.profile
/home/mike/.sh history
/home/mike/.Xauthority
/home/mike/toto
/home/mike/titi
/home/mike/rep
/home/mike/rep/f1
/home/mike/rep/f2
/home/stage1/.profile
/home/stage2/.profile
/home/stage3/.profile
/home/stage4/.profile
/home/stage5/.profile
/home/stage6/.profile
/home/stage7/.profile
/home/stage8/.profile
$
```

À la place de l'action **print** qui se contente d'afficher les noms des fichiers trouvés, il est possible d'utiliser les actions **exec** ou **ok** qui permettent d'exécuter une commande sur chacun des fichiers trouvés, respectivement sans ou avec demande de confirmation.

Le nom de la commande doit être suivie de la notation **{}** en lieu et place du nom du fichier et doit se terminer par les caractères **\;**.

Dans l'exemple suivant, nous illustrons les deux possibilités en exécutant la commande **wc** sur chaque fichier trouvé.

```
$ find . -name '.*?*' -exec wc {} \;
      5      27     154 ./profile
    169     644    3258 ./sh history
      0       1     101 ./Xauthority
$ find . -name '.*?*' -ok wc {} \;
< wc ... ./profile >?  y
      5      27     154 ./profile
< wc ... ./sh history >?  n
< wc ... ./Xauthority >?  y
      0       1     101 ./Xauthority
$
```

3. Sauvegardes

L'activité de sauvegarde est plutôt de la responsabilité de l'administrateur système. Elle se décline souvent en commandes spécifiques ou en solutions logicielles. Cependant, dans la mesure où les bonnes permissions lui seraient accordées sur les fichiers spéciaux (noms des périphériques de sauvegardes), l'utilisateur dispose de deux commandes publiques **tar** et **cpio**, présentes sur toutes les versions.

a. Les commandes de compression

Les commandes de compression de fichiers sont souvent utilisées dans le contexte des sauvegardes. Il y a plusieurs familles de commandes, chronologiquement de plus en plus performantes.

Si ces commandes reçoivent des arguments, elles remplacent le fichier initial par un fichier compressé, renommé avec une pseudo-extension propre à la commande. Symétriquement, les commandes de décompression restituent le fichier original.

Si les commandes ne reçoivent pas d'argument, elles traitent alors tout naturellement leur entrée standard et écrivent le résultat sur la sortie standard. Cette convention permet de combiner ces commandes avec les commandes de sauvegarde dans des *pipelines*. Compte-tenu de cette possibilité, les commandes de sauvegarde sont rarement dotées d'options de compression.

Le tableau suivant nous résume les trois familles de commandes les plus répandues.

	Origine System V	Origine Berkeley	Logiciel libre
Commande de compression	pack	compress	gzip
Extension des fichiers compressés	.z	.Z	.gz
Commande de décompression	unpack	uncompress	gunzip
Décompression sur sortie standard	pcat	zcat	

La commande **gzip** est la plus performante. Il s'agit d'une commande *open source* qui n'est pas toujours disponible de base sur les systèmes Unix. La commande **compress** constitue alors la deuxième meilleure solution tandis que la plus ancienne commande **pack** reste la moins performante.

b. Caractéristiques communes aux commandes de sauvegarde Unix

À partir d'une liste de fichiers à sauvegarder, les commandes fabriquent un fichier **archive** d'un certain format. Chaque commande a son propre format.

Concrètement, l'archive peut être un **fichier spécial** (nom d'un lecteur de disquettes ou de bandes), un **fichier ordinaire** sur disque (créé ou écrasé par la commande) ou encore la **sortie standard** (pour une utilisation dans un *pipeline*).

L'activité symétrique de **restauration** consiste à **recréer les fichiers avec le nom qu'ils ont dans l'archive**. Le chemin utilisé lors de la sauvegarde (chemin absolu ou relatif) est alors très important à vérifier pour ne pas commettre d'erreurs lors de cette restitution.

Le choix de noms relatifs, en se positionnant dans le répertoire souhaité au moment de la sauvegarde, est sans doute préférable car il donne plus de souplesse, même s'il convient d'être attentif au moment de la restauration en s'assurant d'être positionné dans le bon répertoire.

c. La commande tar

La commande **tar** est issu historiquement des *versions Berkeley*. Elle reçoit en argument une liste de fichiers à sauvegarder. Elle effectue un traitement récursif sur les répertoires, ce qui la rend très adaptée pour la sauvegarde d'arborescences complètes.

Le format de l'archive est indépendant du *hardware*. En conséquence, la commande **tar** est la solution classique pour effectuer des transferts entre machines Unix, quelle que soit la version. La commande s'impose aussi comme standard de fait pour proposer des archives Unix sur l'Internet.

En restauration, le comportement par défaut de la commande consiste à écraser les fichiers disques de même nom que sur l'archive. De plus; les fichiers sont restaurés avec leurs propriétaires, permissions et dates d'origine.

La commande comporte beaucoup d'options. Trois options principales (exclusives) sont à connaître : **c** pour effectuer une sauvegarde, **t** pour obtenir la liste du contenu de l'archive et **x** pour effectuer une restauration.

Nous effectuons la **sauvegarde** (*option c*) du répertoire courant. L'*option v* (*verbose*) demande à la commande d'afficher les noms des fichiers traités. L'*option f* est une option à argument. Elle est en dernière position car elle est suivie du nom de l'archive résultat. Nous effectuons ici la sauvegarde dans un fichier ordinaire `/tmp/montar` qui sera créé ou écrasé. La notation `.` désigne le répertoire courant, la commande traite toute la sous-arborescence.

```
$ tar cvf /tmp/montar .
a ./ 0K
a ./profile 1K
a ./sh history 4K
a ./rep/ 0K
a ./rep/fic3 2K
a ./rep/fic4 1K
a ./fic2 1K
a ./exrc 1K
a ./fic1 2K
$ file /tmp/montar
/tmp/montar:      USTAR tar archive
$
```

Pour obtenir ensuite la **liste** détaillée du contenu de l'archive, nous utilisons l'*option t* accompagnée des deux *options v et f* déjà mentionnées.

```
$ tar tvf /tmp/montar
drwxr-xr-x 2000/2000      0 Sep 24 21:25 2004 ./
-rw-r--r-- 2000/2000    154 Sep 23 10:13 2004 ./profile
-rw----- 2000/2000   4032 Sep 24 21:27 2004 ./sh history
drwxr-xr-x 2000/2000      0 Sep 24 21:25 2004 ./rep/
-rw-r--r-- 2000/2000   1242 Sep 24 21:24 2004 ./rep/fic3
-rw-r--r-- 2000/2000    916 Sep 24 21:25 2004 ./rep/fic4
-r--r--r-- 2000/2000    916 Sep 24 12:29 2004 ./fic2
-rw-r--r-- 2000/2000     64 Sep 21 11:09 2004 ./exrc
-rw-r--r-- 2000/2000   1242 Sep 24 10:19 2004 ./fic1
$
```

Pour effectuer la **restauration complète** de notre archive, nous devons nous positionner dans le répertoire où seront recréés les fichiers. En effet, nous avons effectué une sauvegarde avec des noms relatifs comme le montre bien la liste précédente. Après avoir supprimé tout le contenu du répertoire, nous restaurons l'intégralité de l'archive grâce à l'*option x*. Les fichiers retrouvent leurs propriétaires, permissions et dates d'origine.

```
$ pwd
/home/mike
$ rm -r *
$ ls
$ tar xvf /tmp/montar
x ., 0 bytes, 0 tape blocks
x ./profile, 154 bytes, 1 tape blocks
x ./sh history, 4032 bytes, 8 tape blocks
x ./rep, 0 bytes, 0 tape blocks
x ./rep/fic3, 1242 bytes, 3 tape blocks
x ./rep/fic4, 916 bytes, 2 tape blocks
x ./fic2, 916 bytes, 2 tape blocks
x ./exrc, 64 bytes, 1 tape blocks
x ./fic1, 1242 bytes, 3 tape blocks
$ ls -alR
.:
total 24
drwxr-xr-x  3 mike  mike      512 Sep 24 21:25 .
dr-xr-xr-x 12 root   root      512 Sep  1 20:55 ..
-rw-r--r--  1 mike  mike       64 Sep 21 11:09 .exrc
-rw-r--r--  1 mike  mike      154 Sep 23 10:13 .profile
-rw-----  1 mike  mike     4032 Sep 24 21:27 .sh history
-rw-r--r--  1 mike  mike     1242 Sep 24 10:19 fic1
-r--r--r--  1 mike  mike      916 Sep 24 12:29 fic2
drwxr-xr-x  2 mike  mike      512 Sep 24 21:25 rep
.....
```

Pour effectuer la **restauration d'un fichier** individuel, nous devons prendre soin de le nommer exactement comme dans la liste du contenu de l'archive. Dans notre exemple, il faut bien indiquer, en argument supplémentaire, le nom *./fic1* et non pas seulement *fic1* qui ne sera pas trouvé dans l'archive.

```
$ rm fic1
$ tar xvf /tmp/montar fic1
tar: 1 file(s) not extracted
$ tar xvf /tmp/montar ./fic1
x ./fic1, 1242 bytes, 3 tape blocks
$ ls -l fic1
-rw-r--r--  1 mike  mike      1242 Sep 24 10:19 fic1
$
```

Pour pouvoir collaborer avec les commandes de compression dans un *pipeline*, la commande *tar* considère le caractère - (signe moins) comme étant la sortie standard lors d'une sauvegarde et comme étant l'entrée standard lors d'une liste ou d'une restauration. Dans notre dernier exemple, nous effectuons une sauvegarde compressée via la commande *compress* et nous savons restaurer un fichier individuel.

```
$ tar cvf - . | compress > /tmp/montar.Z
a ./ 0K
a ./profile 1K
a ./sh history 5K
a ./rep/ 0K
a ./rep/fic3 2K
a ./fic2 1K
a ./exrc 1K
a ./fic1 2K
$ uncompress < /tmp/montar.Z | tar xvf - ./fic1
x ./fic1, 1242 bytes, 3 tape blocks
$
```

📖 *Les distributions Linux disposent d'une commande tar améliorée (GNU tar) qui dispose notamment d'options directes de compression.*

d. La commande **cpio**

La commande **cpio** est issue historiquement des *versions System V*. Elle ne reçoit pas en argument une liste de fichiers à sauvegarder. En effet, il faut lui fournir cette liste sur son entrée standard. De plus, la commande écrit systématiquement l'archive résultat sur sa sortie standard, ce qui nécessite une redirection finale vers, soit un fichier spécial, soit un fichier ordinaire. Une commande préliminaire (très souvent, la commande *find*) doit donc générer une liste de noms et l'envoyer sur l'entrée standard de *cpio* via un *pipeline*. Ce choix de conception rend la commande très adaptée à des sauvegardes sélectives, ne correspondant pas nécessairement à des arborescences complètes car elle bénéficie, en fait, de toute la richesse fonctionnelle de la commande *find*.

Le format de l'archive est, a priori, moins indépendant du *hardware* que celui de la commande *tar*. En conséquence, *cpio* est rarement choisi pour effectuer des transferts entre machines ou proposer des archives sur l'Internet.

En restauration, le comportement par défaut de la commande est également différent de celui de la commande *tar*. Les fichiers disques de même nom que sur l'archive ne sont pas remplacés. L'*option u* est nécessaire dans ce cas de figure. De plus, les fichiers sont restaurés avec la date système. L'*option m* est disponible pour obtenir la date d'origine.

La commande comporte, bien entendu, beaucoup d'autres options. Deux options principales (exclusives) sont à connaître : **o** pour effectuer une sauvegarde et **i** pour effectuer une restauration. L'option **t**, associée à l'option **i** permet d'obtenir la liste du contenu de l'archive.

Tout comme pour la commande *tar*, nous allons effectuer la **sauvegarde** (*option o*) du répertoire courant. La commande *find* préliminaire ne comporte que l'action *print* (sans options de sélection). Ceci nous permet de générer la liste complète de tous les fichiers de l'arborescence. L'*option v* (*verbose*) demande à la commande d'afficher les noms des fichiers traités. Nous redirigeons la sortie standard de *cpio* vers l'archive résultat, à savoir le fichier ordinaire */tmp/moncpio*.

```
$ find . -print | cpio -ov > /tmp/moncpio
.
.profile
.sh history
.....
$ file /tmp/moncpio
/tmp/moncpio:  byte-swapped cpio archive
$
```

Pour obtenir ensuite la **liste** détaillée du contenu de l'archive, nous utilisons les *options i et t* combinées, accompagnées aussi de l'*option v* déjà mentionnée. Nous devons aussi veiller à rediriger l'entrée standard de *cpio* vers le nom de l'archive.

```
$ cpio -itv < /tmp/moncpio
drwxr-xr-x    3 mike  mike      0 Sep 24 21:32 2004, .
-rw-r--r--    1 mike  mike    154 Sep 23 10:13 2004, .profile
-rw-----    1 mike  mike   4032 Sep 24 21:27 2004, .sh history
drwxr-xr-x    2 mike  mike      0 Sep 24 21:25 2004, rep
-rw-r--r--    1 mike  mike   1242 Sep 24 21:24 2004, rep/fic3
-rw-r--r--    1 mike  mike    916 Sep 24 21:25 2004, rep/fic4
-r--r--r--    1 mike  mike    916 Sep 24 12:29 2004, fic2
-rw-r--r--    1 mike  mike     64 Sep 21 11:09 2004, .exrc
-rw-r--r--    1 mike  mike   1242 Sep 24 10:19 2004, fic1
24 blocks
$
```

Nous illustrons le comportement en **restauration** (*option i* seule, sans l'*option t*) par celle d'un fichier individuel. Le nom de fichier se donne, cette fois, en argument. Le nom de l'archive doit toujours être l'objet d'une redirection de l'entrée standard. Les *options u et m* sont nécessaires pour, respectivement, permettre l'écrasement d'un fichier de même nom que sur l'archive et récupérer la date d'origine. Cela illustre la différence de comportement par défaut, en restauration, entre les commandes *tar* et *cpio*.

```
$ ls -l fic1
-rw-r--r--  1 mike  mike      1242  Sep 25 10:19 fic1
$ cpio -ivum < /tmp/moncpio fic1
fic1
24 blocks
$ ls -l fic1
-rw-r--r--  1 mike  mike      1242  Sep 24 10:19 fic1
$
```

Un autre exemple met l'accent sur l'*option d* qui est requise pour une restauration de fichiers individuels nécessitant de recréer des sous-répertoires. Cette option ne serait pas utile lors d'une restauration complète. Dans ce cas, tous les répertoires seraient recréés sans problème particulier.

```
$ rm -r rep
$ cpio -iv < /tmp/moncpio rep/fic3
cpio: Missing -d option.
cpio: Cannot open/create rep/fic3, errno 2, No such file or directory
24 blocks
2 error(s)
$ cpio -idv < /tmp/moncpio rep/fic3
rep/fic3
24 blocks
$ ls -l rep/fic3
-rw-r--r--  1 mike  mike      1242  Sep 24 21:44 rep/fic3
$
```

Tout comme la commande *tar*, *cpio* peut collaborer avec les commandes de compression dans un *pipeline*.

```
$ find . -print | cpio -ov | compress > /tmp/moncpio.Z
.
.profile
.sh history
.....
$ uncompress < /tmp/moncpio.Z | cpio -ivum fic1
fic1
17 blocks
$
```

4. Commandes d'impression

Les systèmes Unix sont dotés d'un service d'impression permettant l'accès à des imprimantes locales (sur port série ou parallèle), des imprimantes distantes (locales à une autre machine jouant le rôle de serveur d'impression), des imprimantes réseau (directement connectées au réseau).

Du point de vue de l'utilisateur, les imprimantes sont identifiées par des noms logiques attribués par l'administrateur système. L'utilisateur n'a donc pas à se préoccuper de la connexion physique et il ne lui est pas nécessaire de connaître les noms des fichiers spéciaux associés à ces imprimantes.

Trois services d'impression coexistent dans le monde Unix :

- Le spouleur **System V** (démon **lpsched**) utilisé notamment sur Solaris et HP-UX
- Le spouleur **Berkeley** (démon **lpd**) utilisé notamment sur Linux
- Le spouleur **AIX** (démon **qdaemon**)

Ces services correspondent à des jeux de commandes différents pour l'utilisateur. Le spouleur AIX reconnaît les commandes des deux autres services.

	Spouleur System V	Spouleur Berkeley	Spouleur AIX
Requêtes d'impression	lp	lpr	qprt
État des files d'attente	lpstat	lpq	qchk
Annulation de requêtes	cancel	lprm	qcan

5. Autres commandes utiles

a. La commande script

La commande **script** mémorise une session dans un fichier. Elle crée un processus shell fils et se termine à la fin de celui-ci. Le nom par défaut du fichier de capture est *typescript*. Un autre nom peut être donné en argument à la commande. L'*option -a* permet une utilisation du fichier en mode ajout.

```
$ script
Script started, file is typescript
$ echo bonjour
bonjour
$ who -q
mike
# users=1
$ exit
Script done, file is typescript
$ cat typescript
Script started on Sat Sep 25 09:17:41 2004
$ echo bonjour
bonjour
$ who -q
mike
# users=1
$ exit

script done on Sat Sep 25 09:18:09 2004
$
```

b. La commande **crypt**

Grâce à une clé de cryptage, choisie et tapée au clavier par l'utilisateur, la commande **crypt** permet de crypter et décrypter son entrée standard vers sa sortie standard. Elle s'utilise donc via des redirections, tel que dans l'exemple suivant.

Dans un premier temps, nous cryptons le contenu du fichier *fic* et obtenons le résultat dans le fichier *ficbis*. La commande nous demande de taper la clé de cryptage.

```
$ cat fic
Ceci est un fichier personnel
dont le contenu est confidentiel
meme vis a vis de l'administrateur systeme
$ crypt < fic > ficbis
Enter key:
$
$ file ficbis
ficbis:          data
$
```

Nous pouvons supprimer le fichier de départ. À tout moment, la commande *crypt* nous permettra de retrouver les données initiales, à la condition bien évidente de savoir redonner la clé de cryptage.

```
$ rm fic
$ crypt < ficbis
Enter key:
Ceci est un fichier personnel
dont le contenu est confidentiel
meme vis a vis de l'administrateur systeme
$
```

c. La commande du

La commande **du** (*disk usage*) nous informe sur la place disque réellement consommée par les fichiers. Elle fournit ses résultats sous la forme d'un nombre de blocs disques utilisés par les fichiers ou répertoires donnés en argument (assez souvent des blocs virtuels de 512 octets). La commande est dotée de nombreuses options permettant d'affiner son comportement.

Sans argument, la commande traite le répertoire courant et n'affiche que les sous-répertoires.

```
$ du
6      ./rep
34     .
$
```

L'*option -a* visualise tous les fichiers et l'*option -k* donne le résultat en blocs de 1K octets.

```
$ du -ak
1      ./profile
5      ./sh history
2      ./rep/fic3
3      ./rep
1      ./fic2
.....
```

L'*option -s* permet de n'afficher que le nombre total de blocs.

```
$ du /home
10     /home/mysql
6      /home/mike/rep
34     /home/mike
.....
$ du -s /home
126    /home
$ du -ks /home
63     /home
$
```

d. La commande su

La commande **su** reçoit en argument le nom d'un utilisateur. Elle permet d'endosser l'identité de cet autre utilisateur sans opérer une déconnexion puis une reconnexion complète. Sans argument, la commande sert à devenir *root* pour effectuer une opération privilégiée. Il s'agit là de son usage le plus fréquent. Le mot de passe du nouveau compte doit, bien entendu, être fourni. Si ce mot de passe est correct, la commande génère un processus shell fils dont il convient de sortir après avoir effectué les opérations souhaitées.

```
$ id
uid=2000(mike) gid=2000(mike)
$ su
Password:
su: Sorry
$
```

Dans ce premier essai, nous essayons de devenir **root** (*su* est appelé sans argument). Nous échouons car le mot de passe fourni est incorrect.

📖 *Tous les appels à la commande su sont archivés dans un fichier compte-rendu. L'administrateur est donc capable de surveiller les tentatives d'utilisation de cette commande.*

Dans cette deuxième tentative, nous fournissons le bon mot de passe de *root*. Il y a bien création d'un shell fils dans lequel nous avons changé d'identité. La commande *exit* nous ramène ensuite dans le shell initial, sous l'identité de l'utilisateur de départ.

```
$ id
uid=2000(mike) gid=2000(mike)
$ su
Password:
# id
uid=0(root) gid=1(other)
# exit
$ id
uid=2000(mike) gid=2000(mike)
$
```


L'option - (signe moins) est importante car elle permet l'exécution du fichier **.profile** de l'utilisateur cible afin, en général, de bien valider un nouvel environnement shell.

```
$ id ; pwd
uid=2000(mike) gid=2000(mike)
/home/mike
$ su
Password:
# id ; pwd
uid=0(root) gid=1(other)
/home/mike
# echo $PATH
/usr/bin:/bin:..
# exit
$
```

Dans ce premier essai, nous avons simplement endossé l'identité de *root* sans changer d'environnement (même répertoire courant et même *PATH*).

Si nous utilisons l'*option -*, nous constatons que le changement d'identité s'accompagne d'un changement d'environnement (nouveau répertoire courant, nouveau *PATH*). L'usage de cette option est très souvent indispensable à une utilisation efficace de la commande *su*.

```
$ su -
Password:
Sun Microsystems Inc. SunOS 5.9 Generic January 2003
# id ; pwd
uid=0(root) gid=1(other)
/
# echo $PATH
/usr/sbin:/sbin:/usr/bin:/bin
# exit
$
```

e. La commande cut

La commande traite les lignes d'un fichier ou, à défaut, son entrée standard. Elle permet d'extraire de chaque ligne un ensemble de caractères ou de mots selon les options utilisées.

Les options

-d car (l'espace devra être défini entre deux quotes)	Définition du séparateur de champs
-f liste	Liste des numéros des champs à extraire
-c liste extraire	Liste des positions des caractères à

Après avoir redéfini le séparateur de mots (*option -d*), nous sélectionnons les premier, troisième et septième champ du fichier `/etc/passwd`, à savoir le nom, le numéro et le shell de connexion des utilisateurs du système.

```
$ cut -d: -f 1,3,7 /etc/passwd | tail -5
stage4:5004:/usr/bin/ksh
stage5:5005:/usr/bin/ksh
stage6:5006:/usr/bin/ksh
stage7:5007:/usr/bin/ksh
stage8:5008:/usr/bin/ksh
$
```

L'*option -d* est indispensable pour le bon fonctionnement de l'*option -f*, même si les champs sont séparés par des espaces.

```
$ echo Bonjour cher ami | cut -f 2,3
Bonjour cher ami
$ echo Bonjour cher ami | cut -d" " -f 2,3
cher ami
$
```

Il faut savoir que la commande *cut* ne fonctionne correctement que si les champs ne sont séparés que par un seul délimiteur exactement.

Dans l'exemple ci-dessous, la première tentative de filtrage de la commande `ls -l` échoue car les mots sont séparés par un nombre variable d'espaces. Nous devons, au préalable, éliminer ce nombre variable grâce à la commande `echo` qui garantit toujours que les éléments affichés sont séparés d'un seul espace. Rappelons que la notation `$(...)` permet d'insérer le résultat d'une commande dans la liste d'arguments (voir chapitre 5).

```
$ ls -l fic1
-rw-r--r--  1 mike  mike      1242 Sep 24 10:19 fic1
$ ls -l fic1 | cut -d" " -f 3,5,9
mike
$ echo $(ls -l fic1)
-rw-r--r-- 1 mike mike 1242 Sep 24 10:19 fic1
$ echo $(ls -l fic1) | cut -d" " -f 3,5,9
mike 1242 fic1
$
```

La commande `cut` sait également effectuer un découpage selon les positions de caractères.

```
$ echo Bonjour cher ami | cut -c 1-3,14-16
Bonami
$ echo Bonjour cher ami | cut -c 9-
cher ami
$
```

6. Filtres

Le terme **filtre** désigne communément des utilitaires qui effectuent des traitements sur des fichiers ou des données contenant du texte. L'unité de traitement est en effet la **ligne** (présence du caractère `\n` (saut de ligne)). Les filtres constituent des outils très intéressants, notamment pour compléter les fonctionnalités du shell proprement dit, dans un contexte de *script*. Leur appellation provient sans doute du fait qu'ils apparaissent assez souvent dans des *pipelines* pour traiter des résultats de commandes.

Des commandes assez simples, déjà rencontrées, peuvent être qualifiées de filtre, telles que *head*, *tail*, *wc*, *more*, *nl*, *cut*...

Nous allons présenter ici des filtres complémentaires, plus élaborés, qui nécessitent souvent une prise en main syntaxique.

📄 *Les résultats des filtres sont toujours écrits sur la sortie standard. Les données ne sont pas modifiées. Il convient d'utiliser des redirections pour archiver ces résultats dans des fichiers.*

a. Tris avec sort

La commande **sort** permet d'effectuer des tris. L'unité de traitement peut être la ligne complète ou seulement certains champs. Le critère de tri par défaut est le code ascii. De nombreuses options permettent de choisir des critères plus réalistes (alphabétique, numérique...).

Syntaxe résumée

`sort [options] [+pos1 [-pos2]] fichiers...`

Si aucun fichier n'est donné en argument ou si on utilise le caractère - (signe moins), la commande traite son entrée standard.

Quelques Options

-u	Suppression de lignes multiples dans le résultat
-o fichier	Sauvegarde du résultat dans le fichier spécifié, qui peut être alors un des fichiers d'entrée. Une redirection de la sortie vers un des fichiers d'entrée se traduirait par la perte de celui-ci puisque le shell traiterait la redirection avant d'appeler la commande <i>sort</i> .
-d	Critère alphanumérique (lettres, chiffres, espaces, tabulations)
-f	Minuscules et majuscules confondues
-n	Critère numérique
-r	Résultat en ordre décroissant
-tcar	Redéfinition du séparateur de champs
-b	Bonne gestion du nombre variable d'espaces entre les champs

Utilisation des champs

Les notations **+pos1 [-pos2]** permettent d'indiquer que le tri doit s'effectuer uniquement sur certains champs. Il est possible d'indiquer plusieurs couples de position pour opérer un tri à plusieurs passes.

Ces positions s'expriment sous la forme **m.n** éventuellement suivie d'options.

m	Nombre de champs à sauter depuis le début de la ligne
(+2 -3 signifie tri sur le troisième champ)	
n	Nombre de caractères à sauter depuis le début du champ
(+3.0 -3.3 signifie tri sur les 3 premiers caractères du quatrième champ)	

Nous trions le fichier */etc/passwd* d'après le numéro d'utilisateur (troisième champ) en ordre décroissant. Nous devons redéfinir, avec l'*option -t*, le séparateur de champs qui est ici le caractère : (deux points). Nous choisissons d'associer les critères de tri aux positions (notation **+2nr**).

```
$ sort -t: +2nr -3 /etc/passwd | head
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody:x:60001:60001:Nobody:/:
stage8:x:5008:5000::/home/stage8:/usr/bin/ksh
stage7:x:5007:5000::/home/stage7:/usr/bin/ksh
stage6:x:5006:5000::/home/stage6:/usr/bin/ksh
stage5:x:5005:5000::/home/stage5:/usr/bin/ksh
stage4:x:5004:5000::/home/stage4:/usr/bin/ksh
stage3:x:5003:5000::/home/stage3:/usr/bin/ksh
stage2:x:5002:5000::/home/stage2:/usr/bin/ksh
$
```

Dans ce deuxième exemple, nous trions le résultat de la commande *ls -l* selon les tailles décroissantes des fichiers. En cas d'égalité, nous effectuons un tri sur les noms. Le passage préliminaire par la commande *tail -n +2* nous permet d'éliminer la première ligne du résultat de *ls* car cette ligne n'a pas le bon format pour le tri.

```
$ ls -l | tail -n +2 | sort +4nr -5 +8b -9
-rw-r--r-- 1 mike mike 1242 Sep 24 10:19 exemple
-r--r--r-- 1 mike mike 916 Sep 24 12:29 fic2
```

```
-r--r--r--  1 mike  mike      916 Sep 25 12:30 zorro
drwxr-xr-x  2 mike  mike      512 Sep 24 21:44 rep
-rw-r--r--  1 mike  mike      149 Sep 25 09:18 typescript
-rw-r--r--  1 mike  mike      106 Sep 25 09:23 fic
-rw-r--r--  1 mike  mike      106 Sep 25 09:23 ficbis
-rw-r--r--  1 mike  mike      106 Sep 25 12:30 toto
$
```

b. Transformations de caractères avec tr

La commande **tr** permet d'effectuer des substitutions ou des suppressions de caractères.

Syntaxe résumée

tr [options] [chaîne1]
[chaîne2]

Options

- d** Suppression des caractères apparaissant dans *chaîne1*
- s** Dans le résultat, les caractères consécutifs identiques sont réduits à un seul exemplaire
- c** On considère les caractères n'apparaissant pas dans *chaîne1*

La commande lit ses données systématiquement sur l'entrée standard. Dans la syntaxe de base, elle remplace tous les caractères de l'argument *chaîne1* par ceux de même position dans l'argument *chaîne2*.

Des abréviations sont possibles pour désigner des ensembles de caractères :

- [a-z] lettres minuscules (notation d'intervalle avec les crochets)
- [a*n] *n* fois le caractère *a* (* seul signifie *un nombre quelconque*)
- \xyz Code ascii en octal du caractère

Nous transformons la sortie de la commande `echo` en majuscules. Ce genre de transformation résume bien le rôle essentiel de la commande `tr`. Les simples quotes qui encadrent les arguments ne font pas partie de la syntaxe du filtre. Elles empêchent une tentative d'interprétation des crochets par le shell. Cette utilisation assez systématique des quotes va se retrouver souvent dans les syntaxes des autres filtres.

```
$ echo bienvenue au club Unix | tr '[a-z]' '[A-Z]'
```

```
BIENVENUE AU CLUB UNIX
```

```
$
```

L'*option -d* permet la suppression de caractères. La commande ne reçoit, dans ce cas, qu'un seul argument.

```
$ echo bienvenue au club Unix | tr -d " "
```

```
bienvenueauclubUnix
```

```
$
```

Dans le résultat de la commande `ls`, nous remplaçons tous les chiffres par la lettre X. La notation `*` est nécessaire pour *ajuster la longueur* des deux arguments.

```
$ ls -l | tr '[0-9]' '[X*]'
```

```
total XX
```

```
-rw-r--r--  X mike  mike      XXXX Sep XX XX:XX exemple
-rw-r--r--  X mike  mike      XXX Sep XX XX:XX fic
-r--r--r--  X mike  mike      XXX Sep XX XX:XX ficX
-rw-r--r--  X mike  mike      XXX Sep XX XX:XX ficbis
drwxr-xr-x  X mike  mike      XXX Sep XX XX:XX rep
-rw-r--r--  X mike  mike      XXX Sep XX XX:XX toto
-rw-r--r--  X mike  mike      XXX Sep XX XX:XX typescript
-r--r--r--  X mike  mike      XXX Sep XX XX:XX zorro
```

```
$
```

Dans le deuxième essai, l'*option -s* indique que, dans le résultat, si nous avons plusieurs exemplaires adjacents de la lettre X, nous n'en conserverons qu'un seul.

```
$ ls -l | tr -s '[0-9]' '[X*]'
```

```
total X
```

```
-rw-r--r--  X mike  mike      X Sep X X:X exemple
-rw-r--r--  X mike  mike      X Sep X X:X fic
-r--r--r--  X mike  mike      X Sep X X:X ficX
-rw-r--r--  X mike  mike      X Sep X X:X ficbis
drwxr-xr-x  X mike  mike      X Sep X X:X rep
-rw-r--r--  X mike  mike      X Sep X X:X toto
```

```
-rw-r--r--  X mike  mike      X Sep X X:X typescript
-r--r--r--  X mike  mike      X Sep X X:X zorro
$
```

c. Recherche d'expressions (grep, egrep, fgrep)

La commande **grep** effectue des recherches d'expressions littérales ou régulières dans des fichiers.

Les **expressions régulières** consistent en un vocabulaire permettant d'exprimer une sémantique puissante de recherche.

Il y a trois filtres :

grep	Recherche d'expressions littérales et régulières
egrep	Ce filtre connaît un peu plus d'expressions régulières que <i>grep</i> .
fgrep	Ce filtre recherche uniquement les expressions littérales, mais de façon plus performante

Quelques Options

-i	Minuscules et majuscules confondues
-v	Recherche des lignes ne comportant pas l'expression
-x	Recherche des lignes exactement égales à l'expression (fgrep)
-c	Affichage du nombre de lignes trouvées
-l	Affichage uniquement des noms de fichiers
-n	Indication du numéro de la ligne dans le fichier
-e expr	Recherche d'une expression commençant par - (signe moins) (ne fonctionne pas avec grep)

-f fichier

Expressions décrites dans un fichier (une par ligne) (ne fonctionne pas avec grep)

Quelques expressions régulières

^	Début de ligne
\$	Fin de ligne (se place en fin d'expression)
.	Présence d'un caractère quelconque
[...]	Un caractère parmi un ensemble
[^...]	Un caractère ne figurant pas dans l'ensemble
*	Un nombre quelconque d'apparitions d'un caractère (éventuellement aucune)
^\$	Ligne vide (un début et une fin tout de suite)
.*	Reste de la ligne ou ligne complète suivant le contexte
 	Le caractère signifie OU et relie deux expressions (egrep seulement)

Quelques exemples

Lignes qui commencent par une majuscule.

```
grep '^[A-Z]' fichier
```

Lignes qui se terminent par la chaîne *cd1* suivie d'un nombre quelconque d'espaces (éventuellement aucun). Le caractère *** (nombre quelconque) qualifie le caractère placé devant lui (espace).

```
grep 'cd1 *$' fichier
```

Lignes qui commencent par une majuscule ou bien qui se terminent par la chaîne *cd1*. Le caractère *|* signifie OU mais n'est interprété que par la commande *egrep*.

```
egrep '^[A-Z]|cd1$' fichier
```

Lignes qui commencent par une majuscule et qui se terminent par la chaîne *cd1*. La notation *.**, placée entre les deux critères, permet d'exprimer un ET puisqu'elle signifie littéralement que, à cet endroit de la ligne, il y aura un nombre quelconque de caractères quelconques.

```
grep '^[A-Z].*cd1$' fichier
```

d. Édition non interactive de fichiers avec sed

Le filtre **sed** est un éditeur non interactif qui copie les fichiers d'entrée sur la sortie standard après avoir appliqué un certain nombre d'actions.

Syntaxe résumée

```
sed [-n] 'commandes_sed' fichiers...
```

ou

```
sed [-n] -f fichier_commandes fichiers...
```

Syntaxe des commandes sed et principe de fonctionnement

```
[ adresse1 [,adresse2] ] action [ arguments ]
```

Les crochets indiquent un aspect facultatif et n'apparaissent pas dans les commandes. Tous ces éléments ne sont séparés par aucun espace. En l'absence d'adresses de sélection, l'action a lieu sur toutes les lignes des fichiers d'entrée.

L'action par défaut est d'afficher la ligne sur la sortie standard. L'*option -n* permet de supprimer cette sortie systématique des lignes.

Une **adresse** peut être, entre autres :

- un numéro de ligne,
- le caractère **\$** désignant la dernière ligne,
- le caractère **.** (point) désignant la ligne courante,
- une expression littérale ou régulière placée entre deux caractères **/**.

Quelques actions usuelles

d	Ne pas afficher la ligne
p	Afficher la ligne (s'utilise souvent avec l' <i>option -n</i>)

q	Abandonner le traitement
s/expr1/expr2/	Remplacer la première expression par la seconde,
une seule fois par ligne	
s/expr1/expr2/g	Remplacer toutes les occurrences de la première expression par la seconde
s/expr1//	Supprimer l'expression (une seule fois par ligne)
s/expr1//g	Supprimer toutes les occurrences d'une expression
s/expr1/...&.../	Expanser l'expression
La notation & signifie reprendre la première expression.	
=	Afficher le numéro de ligne

Quelques exemples

L'action *p* (afficher la ligne) n'est demandée que sur les lignes 5 à 10. L'option *-n* annule la sortie automatique des autres lignes (action par défaut de *sed*).

`sed -n '5,10p' fichier`

L'action *q* arrête le traitement dès qu'une ligne commence par une majuscule. Comme il se doit dans le contexte de la commande *sed*, l'expression régulière est placée entre deux caractères /. L'action par défaut a permis d'afficher les lignes jusqu'à rencontrer la ligne recherchée.

`sed '/^[A-Z]/q' fichier`

L'action *s* effectue le remplacement de la chaîne *cd* par la chaîne *code*. Cette transformation n'est demandée que sur les lignes qui commencent par une majuscule. L'action par défaut permet, par ailleurs, d'afficher les lignes sur la sortie standard.

`sed '/^[A-Z]/s/cd/code/' fichier`

La notation **&**, placée dans la deuxième expression de l'action *s*, signifie qu'il faut reprendre le résultat de l'évaluation de la première expression. La première expression *.** désigne ici la ligne complète. La commande *sed* place donc des parenthèses autour des lignes, affichées sur la sortie standard grâce à l'action par défaut.

`sed 's/.*/(&)/' fichier`

7. Récapitulatif des commandes à approfondir dans la documentation

cmp	Comparaison de deux fichiers
comm	Comparaison de fichiers triés
compress, uncompress	Compression, décompression de fichiers
cpio	Archivage et restauration de fichiers
crypt	Cryptage et décryptage
cut	Sélection de caractères ou de champs
diff	Comparaison de fichiers
du	Informations sur l'espace disque utilisé
file	Détermination du type de contenu
find	Recherche de fichiers
fold	Découpage de lignes

grep, egrep, fgrep	Recherche d'expressions
gzip, gunzip	Compression, décompression de fichiers
join	Jointure entre deux fichiers triés
lp, lpstat, cancel	Commandes d'impression du spouleur System V
lpr, lpq, lprm	Commandes d'impression du spouleur Berkeley
nl	Numérotation de lignes
od	Visualisation de fichiers sous différents formats
pack, unpack	Compression, décompression de fichiers
paste	Concaténation de lignes entre plusieurs fichiers
pr	Mise en forme de fichiers pour impression
qprt, qchk, qcan	Commandes d'impression du spouleur AIX
script	Capture d'une session sur un terminal
sed	Éditeur de texte non interactif
sort	Utilitaire de tri
split	Éclatement d'un fichier en plusieurs fichiers
strings	Recherche de texte dans des fichiers binaires
su	Changement d'identité
tar	Archivage et restauration de fichiers
tr	Transformation de caractères

8. TP 10 - Utilitaires

Ce qui est fait dans cet exercice :

Le but de cet exercice est de se familiariser avec quelques une des commandes utilitaires les plus utiles d'Unix.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable :

- Lancer des recherches de fichiers de façon récursive à travers des ensembles de dossiers, avec des critères variés,
- Rechercher des fichiers par leur nom, à partir de modèles de texte qu'il contiendrait,
- Comparez le contenu de deux fichiers pour savoir s'il est identique.
- Extraire des champs spécifiques d'un fichier de texte,
- Trier les lignes d'un fichier,
- Afficher les premières ou dernières lignes d'un fichier,
- Sauvegarder et restaurer des fichiers avec la commande **tar**.

Introduction

Cet exercice est conçu pour que vous expérimentiez avec les principaux utilitaires, notamment **find** et **grep**, et d'autres commandes qui une fois rassemblées dans des scripts shell, permettent de construire des commandes personnalisées.

L'édition de l'historique de la ligne de commande sera très utile ici pour modifier des commandes particulièrement longues à écrire, et souvent répétées.

La correction de cet exercice utilise le **\$** comme signe d'invite de la ligne de commande. Mais en fonction des exercices précédents, ou par votre choix de modifier la variable PS1, votre invite est peut être différente.

Remarque : Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

.a La commande find

- ___ 1. Recherchez tous les fichiers du répertoire **/tmp** et afficher leur chemin d'accès.
- ___ 2. A partir de votre répertoire d'accueil, recherchez tous les fichiers dont le nom commence par la lettre **s**, et pour chacun d'entre-eux lancez automatiquement la commande **ls -l**.
- ___ 3. Relancez la commande précédente, mais cette fois avec une demande de confirmation avant de lancer la commande **ls -l** sur chaque nom de fichier trouvé.
- ___ 4. . Toujours depuis votre répertoire d'accueil, recherchez à partir des répertoires **/var** et **/tmp**, les fichiers qui vous appartiennent. Puis modifiez la ligne de commande pour compter ce nombre de fichiers. Il peut y avoir des répertoires pour lesquels vous n'avez pas le droit de passage. Renvoyez les messages d'erreur vers un fichier nommé **errfic**.
- ___ 5. Affichez le fichier **errfic** indiqué à la commande précédente, pour vérifier s'il y a eu des messages d'erreur.
- ___ 6. Pour mettre en évidence que **find**, à partir d'un répertoire de départ , fonctionne de façon récursive dans tous les répertoires et sous-répertoires sous-jacents, effectuez les opérations suivantes:
 - a. assurez-vous d'être dans votre répertoire d'accueil,
 - b. créez un sous répertoire nommé **niveau1**,
 - c. dans ce sous répertoire **niveau1**, créez un fichier de taille vide nommé **nomfic1**,
 - d. déplacez-vous dans le sous répertoire **niveau1**,

- e. dans **niveau1** créez un sous répertoire **niveau2**,
- f. dans ce sous répertoire **niveau2**, créez un fichier de taille vide nommé **nomfic2**,
- g. retournez dans votre répertoire d'accueil,
- h. depuis votre répertoire d'accueil, lancez la commande qui affiche tous les fichiers dont le nom commencent par la lettre **n**. Notez les noms affichés.
- i. depuis votre répertoire d'accueil relancez la commande qui affiche les fichiers dont le nom commence par **n**, mais cette fois en ne recherchant que les fichiers ordinaires. Notez aussi cette nouvelle liste de noms et comparez avec la précédente.

.b La commande **grep**

- ___ 1. Dans le fichier **/etc/passwd**, recherchez toutes les lignes des utilisateurs dont le nom commence par **stage**,
- ___ 2. Dans le fichier **/etc/passwd** recherchez toutes les lignes qui commencent par la lettre **s**,
- ___ 3. Dans le fichier **/etc/passwd**, recherchez toutes les lignes qui contiennent un chiffre 0 – 9,
- ___ 4. Relancez la commande précédente, mais cette fois affichez uniquement le nombre de lignes qui contiennent un chiffre,
- ___ 5. Utilisez les commandes **ps** et **grep** pour afficher la liste des process associés à d'autres utilisateurs que vous mêmes.

.c La commande **sort**

- ___ 6. Affichez le contenu du fichier **/etc/passwd** trié par ordre alphabétique. Puis en ordre alphabétique inverse,

.d Les commandes **head** et **tail**

___ 7. Affichez les dix premières lignes de **/etc/passwd**,

___ 8. Affichez les cinq premières lignes de **/etc/passwd**,

___ 9. Affichez les dix dernières lignes de **/etc/passwd**

.e Les commandes `diff` et `cmp`

___ 10. Créez un fichier **liste1** qui contient les prénoms de personnes que vous connaissez, un prénom par ligne, au moins quatre prénoms différents. Copiez ce fichier **liste1** en un fichier **liste2**. Editez le fichier **liste2** et faites les modifications suivantes :

- . changez l'orthographe d'un des prénoms,
- . supprimez un des autres prénoms,
- . ajoutez un nouveau prénom.

___ 11. Utilisez la commande **diff** pour comparer les contenus de **liste1** et **liste2**,

___ 12. Utilisez la commande **cmp** et comparez de nouveau le contenu des fichiers **liste1** et **liste2**. Puis relancez la commande de façon à obtenir une liste longue des différences.

.f La commande `tar`

___ 13. Depuis votre répertoire d'accueil utilisez la commande **tar** pour archiver tous les fichiers de ce répertoire. Créez un fichier d'archive **/tmp/archivexx.tar** pour sauvegarder les fichiers en mode relatif.

___ 14. Vérifiez le contenu du fichier d'archive.

___ 15. Créez un répertoire **/tmp/restaurexx** (où xx est votre numéro de connexion). Déplacez-vous dans ce répertoire pour y restaurer les fichiers de votre archive.

Fin de l'exercice

2775cbe5ce
Global Knowledge

Chapitre 7

Commandes réseau,

Environnements

graphiques

277569550
Global Knowledge

1. Interfaces physiques

La commande **ifconfig** permet de rendre disponible une carte réseau pour son utilisation via les protocoles TCP/IP.

Le nom de l'interface de bouclage (127.0.0.1) est *lo0* (*lo* sous Linux). En ce qui concerne les noms des véritables cartes réseau, ils sont variables selon les versions.

Exemple de noms pour la première carte réseau (qui est souvent la seule)

AIX	en0 (Ethernet) ou tr0 (Token Ring)
HP-UX	lan0
Linux	eth0
Solaris	variable selon le type ou le constructeur de la carte

Les noms et caractéristiques de toutes les interfaces disponibles sont obtenus par la commande **ifconfig -a** ou bien encore par la commande **netstat -in** quand l'*option -a* n'est pas disponible (versions HP-UX).

Exemple HP-UX

```
$ ifconfig lan0
lan0: flags=843<UP,BROADCAST,RUNNING,MULTICAST>
      inet 172.16.1.42 netmask ffff0000 broadcast 172.16.255.255
$
```

Exemple AIX

```
$ ifconfig en0
en0: flags=8080863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST>
      inet 172.16.1.10 netmask 0xffff0000 broadcast 172.16.255.255
$
```

Exemple Linux**\$ ifconfig -a**

```

eth0  Link encap:Ethernet  HWaddr 00:60:97:FE:CB:96
      inet addr:172.16.1.2  Bcast:172.16.255.255  Mask:255.255.0.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:504 errors:0 dropped:0 overruns:0 frame:0
      TX packets:156 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      RX bytes:55428 (54.1 Kb)  TX bytes:11149 (10.8 Kb)
      Interrupt:11 Base address:0x300

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:10 errors:0 dropped:0 overruns:0 frame:0
      TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:700 (700.0 b)  TX bytes:700 (700.0 b)

```

\$

Exemple Solaris**\$ ifconfig -a**

```

lo0: <UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
      inet 127.0.0.1  netmask ff000000

elx10: <UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
      inet 172.16.1.3  netmask ffff0000  broadcast 172.16.255.255
      ether 0:b0:d0:1d:9:c1

```

\$ netstat -in

Name	Mtu	Net/Dest	Address	Ipkts	Ierrs	Opkts	Oerrs	Collis	Queue
lo0	8232	127.0.0.0	127.0.0.1	384	0	384	0	0	0
elx10	1500	172.16.0.0	172.16.1.3	9567	0	4093	0	0	0

\$

2. Résolution des noms

Dans leurs manipulations, les utilisateurs préfèrent mentionner les noms en clair des systèmes. Bien évidemment, ces noms doivent être mis en correspondance avec les adresses IP.

Il y a deux façons de gérer cet aspect :

- le fichier local **/etc/hosts**
- l'organisation en domaines **DNS**

a. Fichier **/etc/hosts**

Dans des réseaux locaux regroupant un petit nombre de machines, les noms consistent en de simples chaînes de caractères et la correspondance avec les adresses peut être indiquée via un simple fichier **/etc/hosts**.

La syntaxe des lignes du fichier est la suivante :

Adresse_internet Nom Alias...

Le deuxième champ représente le nom officiel du système. Il peut être suivi d'éventuels alias. Des commentaires (introduits par le caractère #) peuvent apparaître sur une ligne complète ou en fin de ligne.

Exemple de fichier **/etc/hosts**

```
# Internet host table
127.0.0.1      localhost
172.16.1.3     solaris9  loghost
172.16.1.240   aix43   ibm1      # serveur NFS
172.16.1.242   aix5    ibm2
172.16.1.245   hpux11         # machine de test
172.16.1.246   redhat9
#
```

b. Aspect client DNS

Le service **DNS** (Domain Name System) est un système hiérarchique distribué. Son utilisation est obligatoire sur le réseau Internet. Il s'avère également très intéressant pour un réseau d'entreprise en permettant de centraliser, sur des serveurs, les noms et adresses de tous les systèmes. Le DNS considère le réseau comme une arborescence de domaines. Le **nom complet** d'une machine prend en compte la hiérarchie de ces domaines.

Exemples

www.globalknowledge.fr
pc1.gestion.masociete.com

Une machine Unix devient client DNS dès la présence du fichier **/etc/resolv.conf** qui mentionne le nom du domaine et la liste des serveurs DNS à contacter.

Exemple de fichier **/etc/resolv.conf**

```
domain                masociete.com
nameserver             172.16.1.1
nameserver             172.16.2.10
```

domain Cette entrée définit le nom de domaine qui sera ajouté aux noms de machine ne se terminant pas par un point. Une entrée *search* est parfois préférée à celle-ci car elle permet d'indiquer une liste explicite de noms de domaines de recherche.

nameserver Ce type d'entrée indique l'adresse IP d'un serveur DNS. On place ces lignes (au maximum 3) dans l'ordre préférentiel d'interrogation.

L'ordre de recherche pour la résolution de noms est souvent déterminé dans un fichier complémentaire **/etc/nsswitch.conf**.

Par exemple, la ligne suivante indique que la résolution de noms se fait d'abord dans le fichier **/etc/hosts** puis ensuite auprès des serveurs DNS.

hosts: files dns

Les commandes **nslookup** ou **dig** permettent d'interroger les serveurs DNS.

3. Applications standards

a. Terminal virtuel (telnet)

La commande **telnet** permet d'établir une connexion sur une machine dont on donne le nom ou l'adresse IP en argument.

Invoquée sans argument, la commande présente un *prompt* et permet un certain nombre de commandes :

?	Aide en ligne
open host	Ouverture d'une connexion sur la machine <i>host</i>
set argument valeur	Positionnement des valeurs des <i>variables telnet</i> etc ...

```
$ telnet solaris9
Trying 192.168.0.6...
Connected to solaris9.
Escape character is '^]'.

SunOS 5.9
login: mike
Password:
Last login: Sat Sep 25 13:41:04 from 192.168.0.1
Sun Microsystems Inc.      SunOS 5.9          Generic January 2003
$
.....
.....
$ exit
Connection to solaris9 closed by foreign host.
$
```



```
$ telnet
```

```
telnet> ?
```

Commands may be abbreviated. Commands are:

close	close current connection
logout	forcibly logout remote user and close the connection
display	display operating parameters
mode	try to enter line or character mode ('mode ?' for more)
open	connect to a site
quit	exit telnet
send	transmit special characters ('send ?' for more)
set	set operating parameters ('set ?' for more)
unset	unset operating parameters ('unset ?' for more)
status	print status information
toggle	toggle operating parameters ('toggle ?' for more)
slc	change state of special characters ('slc ?' for more)
z	suspend telnet
!	invoke a subshell
environ	change environment variables ('environ ?' for more)
?	print help information
<return>	leave command mode

```
telnet> open solaris9
```

```
Trying 192.168.0.6...
```

```
Connected to solaris9.
```

```
Escape character is '^['.
```

```
SunOS 5.9
```

```
login:
```

```
.....  
.....
```

b. Les "remote commands"

On désigne sous le terme de **remote commands** un ensemble de trois commandes :

- | | |
|-----------------|----------------------------------|
| - rlogin | Connexion distante |
| - rsh | Exécution de commandes distantes |
| - rcp | Copie de fichiers |

L'utilisation de ces commandes est intéressante dès que l'on possède un compte de même nom sur un ensemble de machines. L'utilisation vers un autre compte est possible mais rarement mise en oeuvre en pratique.

Sur chacun des systèmes, un fichier **.rhosts** doit être placé dans le répertoire de connexion. Il faudra y mentionner le nom des autres *hosts* concernés. Le rôle de ce fichier est de permettre la connexion automatique vers le système distant, sans avoir à fournir le mot de passe. Cette autorisation n'est valable que pour un compte de même nom, cela ne pose donc pas de problème de sécurité.

Si le fichier **.rhosts** ne fait qu'apporter du confort dans l'utilisation de la commande **rlogin**, sa présence devient une obligation en ce qui concerne les deux autres commandes **rsh** et **rcp**. Le message *Permission denied* viendrait sanctionner l'absence du fichier sur la machine cible.

Un fichier général d'autorisation **/etc/hosts.equiv** pourrait être positionné par l'administrateur mais ceci est rarement mis en oeuvre pour des raisons de sécurité.

c. La commande rlogin

La commande **rlogin** permet la connexion vers une autre machine sur laquelle on possède, par défaut, un compte de même nom.

`rlogin [options] host`

Quelques options

- | | |
|-----------------|-----------------------------------------|
| -l login | Connexion sous un compte différent |
| -e car | Redéfinition du caractère d'échappement |

Par défaut, la suite de caractères **~.** permet d'interrompre la commande.

Dans sa première tentative, l'utilisateur *mike* doit fournir son mot de passe car il n'a pas encore mentionné le nom du système d'origine dans le fichier *.rhosts* du système cible. Une fois connecté, il complète ce fichier afin de pouvoir se connecter de manière plus confortable par la suite.

```
$ id
uid=2000(mike) gid=2000(mike)
$ uname -n
solaris8
$ rlogin solaris9
Password:
Last login: Sat Sep 25 18:34:00 from solaris8
Sun Microsystems Inc.      SunOS 5.9          Generic January 2003
$ id
uid=2000(mike) gid=2000(mike)
$ uname -n
solaris9
$ echo solaris8 >> .rhosts
$ exit
Connection to solaris9 closed.
$
.....
.....
$ rlogin solaris9
Last login: Sat Sep 25 18:35:21 from solaris8
Sun Microsystems Inc.      SunOS 5.9          Generic January 2003
$ id
uid=2000(mike) gid=2000(mike)
$ uname -n
solaris9
$
.....
.....
```

d. La commande rsh

La commande **rsh** permet d'exécuter une commande à distance. Sous certaines versions, la commande se nomme plutôt **remsh**, voire encore **rcmd**, *rsh* désignant alors un shell restreint (*restricted shell*).

rsh host commande

La commande *rsh* ne peut pas s'exécuter en l'absence d'autorisation dans le fichier *.rhosts* du système cible. En effet, la commande doit pouvoir effectuer une connexion sous-jacente automatique afin de lancer la commande distante.

Le message *permission denied* indiquerait l'échec de la connexion automatique.

```
$ rsh solaris9 uname -n
permission denied
$
```

☞ *L'entrée standard de rsh devient l'entrée standard de la commande à lancer. La sortie et l'erreur standard de la commande deviennent celles de rsh. L'exécution des commandes distantes s'intègre donc parfaitement dans des pipelines.*

Dans cet exemple, nous pouvons imprimer un fichier local sur l'imprimante par défaut du système distant.

```
$ cat toto | rsh solaris9 lp
request id is impr-989 (standard input)
$ rsh solaris9 lpstat
impr-989          mike              75    Sep 24 12:10
$
```

☞ *Les caractères spéciaux non banalisés sont traités au niveau de la machine locale. Il convient de maîtriser cet aspect en cas d'utilisation des redirections.*

rsh solaris9 cal 2004 > toto

Fichier créé sur la machine locale

rsh solaris9 'cal 2004 > toto'

Fichier créé sur la machine distante

e. La commande rcp

La commande **rcp** permet le transfert de fichiers entre systèmes Unix. La syntaxe est quasiment la même que la commande locale **cp**, il suffit de préfixer les noms de fichiers par celui de la machine concernée.

Tout comme la commande *rsh*, *rcp* ne peut pas s'exécuter en l'absence d'autorisation dans le fichier *.rhosts* du système cible. En effet, la commande doit pouvoir effectuer une connexion sous-jacente automatique.

Le message *permission denied* indiquerait l'échec de la connexion automatique.

☞ *Tout comme pour la commande rsh, les caractères spéciaux non banalisés sont traités au niveau de la machine locale.*

```
$ rcp toto solaris9:/tmp
$ rsh solaris9 'ls -l /tmp/toto'
-rw-r--r--  1 mike  mike  1264   Sep 14 18:45  toto
$
```

f. Transferts de Fichiers avec ftp (sftp)

La commande **ftp** (ou la version sécurisée *sftp*) permet d'effectuer des copies de fichiers entre machines du réseau.

`ftp [options] host`

La commande effectue une **connexion vers un compte avec mot de passe** de la machine cible. Après cette connexion, la commande s'exécute en mode interactif. Elle affiche le prompt *ftp>*. Certaines actions concernent alors la machine sur laquelle l'utilisateur s'est connecté (machine distante), certaines autres ont lieu sur la machine locale. Les transferts peuvent avoir lieu dans les deux sens, selon la commande utilisée.

Commandes essentielles

help ou ?	Aide en ligne
status	Affichage de l'état courant de la session
ascii	Transferts des fichiers en mode texte (bonne conversion entre Unix et Windows)
binary	Transferts des fichiers en mode binaire (recommandé pour des transferts entre deux systèmes Unix)
pwd	Affichage du nom du répertoire courant sur la machine distante
cd rep	Changement de répertoire sur la machine distante
ls	Liste des noms de fichiers sur la machine distante
!pwd	Affichage du nom du répertoire courant sur la machine locale
lcd rep	Changement de répertoire sur la machine locale
!ls	Liste des noms de fichiers sur la machine locale
get f1 f2	Récupération d'un fichier distant
recv f1 f2	Récupération d'un fichier (synonyme de <i>get</i>)
put f1 f2	Émission d'un fichier local
send f1 f2	Émission d'un fichier (synonyme de <i>put</i>)
glob	Réactive ou inhibe la résolution des caractères spéciaux
prompt	Activation du mode interactif pour un transfert multiple
mget fichiers	Récupération de fichiers vers le répertoire local courant
mput fichiers	Émission de fichiers vers le répertoire distant courant

bye

Sortie de ftp

quitSortie de ftp (synonyme de *bye*)**Exemple de session ftp**

```
$ ftp solaris9
Connected to solaris9.
220 solaris9 FTP server ready
Name (solaris9:mike):
331 Password required for root.
Password:
230 User mike logged in.
ftp> status
Connected to solaris9.
No proxy connection.
Mode: stream; Type: ascii; Form: non-print; Structure: file
Verbose: on; Bell: off; Prompting: on; Globbing: on
Store unique: off; Receive unique: off
Case: off; CR stripping: on
Ntrans: off
Nmap: off
Hash mark printing: off; Use of PORT cmds: on
ftp> binary
200 Type set to I.
ftp> get /etc/group /tmp/toto
local: /tmp/toto remote: /etc/group
200 PORT command successful.
150 Binary data connection for /etc/group (192.9.200.130,1069) (1264
bytes).
226 Binary Transfer complete.
1264 bytes received in 0.03 seconds (41 Kbytes/s)
ftp> mget /home/mike/.??*hrc
mget /home/mike/.exrc? y
200 PORT command successful.
150 Binary data connection for /home/mike/.exrc (192.9.200.130,1073) (196
bytes).
226 Binary Transfer complete.
196 bytes received in 0 seconds (0.19 Kbytes/s)
mget /home/stage1/.kshrc? n
ftp> quit
221 Goodbye.
```

\$

g. L'alternative sécurisée ssh

SSH (Secure SHell) est une alternative sécurisée aux "remote commands" traditionnelles : *rlogin*, *rsh*, *rcp*. Il s'agit de crypter le trafic réseau (notamment les mots de passe) et de fournir des possibilités complémentaires d'authentification.

Les versions Linux utilisent notamment **OpenSSH** (www.openssh.org) qui est une version libre de la suite d'outils du protocole.

L'utilisateur dispose de commandes similaires, en terme d'utilisation, aux commandes traditionnelles mais il bénéficie du cryptage intégral des données échangées.

Les commandes sont les suivantes :

- **ssh** (remplaçant de *telnet*, *rlogin* et *rsh*)
- **scp** (remplaçant de *rcp*)
- **sftp** (remplaçant de *ftp*)

4. Environnements graphiques

a. Protocole X-Window, schéma fonctionnel et terminologie

Les environnements graphiques Unix (baptisés historiquement **X-Window**) s'appuient sur les protocoles TCP/IP.

Un programme **serveur d'affichage** (**serveur X**) gère un poste de travail graphique que l'on nomme **DISPLAY** (écran-clavier-souris).

Les *serveurs X* s'exécutent toujours en local sur le poste de travail.

Si nous utilisons la console graphique d'un système Unix, le serveur X est un processus de cette machine.

Dans le cas d'un matériel de type *terminal X*, le serveur est chargé dans la mémoire du terminal au moment de son démarrage.

Le serveur X peut également faire partie d'un logiciel dit d'émulation qui s'exécute sur un autre système d'exploitation tel que *Microsoft Windows*. Ce dernier cas de figure est maintenant plus répandu que le terminal X spécialisé car il s'impose souvent en terme de coût et permet de faire cohabiter Windows et Unix en mode graphique sur un seul poste de travail.

Les applications sont associées à des **fenêtres** et sont considérées, dans le schéma fonctionnel, comme des **clients X** car elles sollicitent le serveur en terme de requêtes d'affichage.

En effet, les actions de l'utilisateur (entrées clavier, actions souris...) sont captées par le serveur X et transmises vers les applications concernées sous la forme d'événements. Une application répond au serveur sous la forme d'une requête qui traduit une demande d'action à effectuer à l'écran.

Clients et serveur communiquent via le **protocole X** qui peut lui-même s'appuyer sur les mécanismes locaux inter-processus ou sur les protocoles TCP/IP.



Les divers clients X peuvent donc s'exécuter sur des machines différentes du réseau en venant s'afficher sur le seul poste de travail graphique piloté par le serveur X. Cet aspect réseau constitue l'atout fonctionnel majeur de ce système de multi-fenêtrage.

Unix propose un environnement intégré prêt à l'emploi baptisé **desktop**. Sur les versions commerciales, une volonté d'unification s'est traduite, il y a quelques années, par la mise en oeuvre du **CDE** (Common Desktop Environment).

Outre, bien entendu, le serveur X, un *desktop* intègre un certain nombre de clients, parmi lesquels :

- un service de connexion

Il s'agit d'une boîte de connexion venant s'afficher sur le poste de travail graphique pour permettre l'authentification sur un des systèmes du réseau. L'administrateur Unix choisit et paramètre ce service de connexion (**dtlogin** pour le CDE).

- divers accessoires

émulateur de terminal, horloge, calculatrice, agenda, éditeur de texte...

- des clients plus élaborés

gestionnaire de fichiers, outils de documentation, menus d'administration...

- un gestionnaire de fenêtres (**window manager**)

Il s'agit d'un client particulier qui joue un rôle fondamental puisqu'il permet de gérer les autres clients (déplacement, dimensionnement, mise en icône...). Pour ce faire, il habille chaque fenêtre d'un contour et gère des menus. Le gestionnaire de fenêtre pilote également la session. Sa terminaison provoque le renouvellement du cycle de connexion.

b. Paramétrage et lancement de clients

Le comportement d'un client X est paramétrable, soit de façon classique via des **options** de la ligne de commande, soit via un mécanisme très complet de **ressources**.

Un client X doit implémenter un certain nombre d'options générales.

Quelques options de base

-fg	Couleur de premier plan
-bg	Couleur d'arrière plan
-title	Titre de la fenêtre
-geometry	Choix de la taille et de la position

Exemples:

```
xterm -fg yellow -bg blue -title "Emulation de terminal"
```

```
xterm -geometry +100+100
```

```
xclock -geometry 90x80-0+0
```

La notion de ressource permet de définir plus complètement des préférences usuelles ou d'adapter l'affichage des clients (taille, fontes...) aux différents types d'écran. Ces ressources sont décrites dans divers fichiers lus par le client lors de son démarrage. Il s'agit là d'un vaste sujet qui sort du cadre de notre présentation.

c. La variable DISPLAY

Les clients utilisent la variable d'environnement **DISPLAY** pour déterminer l'identité du serveur d'affichage.

Cette variable est correctement positionnée grâce au service de connexion.

La syntaxe de cette variable est la suivante :

```
[host]:server[.screen]
```

<i>host</i>	Nom du système sur lequel s'exécute le serveur X
<i>server</i>	Numéro du serveur
<i>screen</i>	Numéro d'écran

En pratique, il n'y a qu'un serveur gérant un seul écran, d'où la notation **0.0** pour ces deux derniers éléments.

Un client lancé depuis l'émulateur de terminal s'affiche tout naturellement sur le même DISPLAY (héritage de l'environnement shell).

Il convient de lancer les clients en arrière-plan pour garder la main dans la fenêtre d'origine.

d. Lancement de clients distants depuis un émulateur de terminal

Deux méthodes peuvent être utilisées pour réaliser le lancement de clients distants, c'est-à-dire d'applications qui s'exécutent sur un autre système que celui sur lequel nous nous sommes authentifiés lors de la connexion, tout en venant s'afficher sur notre écran, géré par le serveur X local.

La première méthode consiste à utiliser la commande *telnet* ou *rlogin*.

Les étapes sont alors les suivantes :

- Connexion distante


- La **variable DISPLAY** n'est pas transmise lors de la connexion distante. Il faut donc recréer et mettre dans l'environnement cette variable.

La seconde méthode consiste à utiliser la commande *rsh*.

Cette méthode fait appel à l'**option -display** que doit implémenter tout client X. Cette option permet d'indiquer explicitement le nom du serveur d'affichage.

Nous allons donc utiliser une syntaxe qui ressemble à :

```
rsh host client -display $DISPLAY &
```

 *Il convient de s'assurer que la variable d'environnement DISPLAY est correctement positionnée sur la machine de départ. Son contenu va être résolu par le shell local. En ce qui concerne le client X, il faut assez souvent fournir son nom complet pour éviter des problèmes éventuels de PATH différents sur les deux systèmes.*

Exemple

```
rsh solaris9 "/usr/X11R6/bin/xterm -display $DISPLAY" &
```

Chapitre 8

Bases de la

programmation

Korn shell

2775cb05ce
Global Knowledge

.1 Procédures et paramètres

Des séquences de commandes et d'instructions shell peuvent être regroupées dans un fichier qui devient ainsi une nouvelle commande (procédure ou **script**).

Pour lancer un script, deux méthodes sont disponibles :

Exécution explicite

vi proc

ksh proc

Exécution implicite

vi proc

chmod +x proc

proc

Dans les deux cas, **le script est interprété par un shell fils** du shell de connexion.

Le caractère **#** introduit un **commentaire** jusqu'à la fin de la ligne en cours. **La première ligne de la procédure** doit être un commentaire particulier qui précise l'interpréteur choisi pour exécuter le script.

#! Nom_complet_de_l'interpréteur

Exemple :

#! /usr/bin/ksh

Ces procédures, ainsi créées, sont évidemment paramétrables. Des notations sont directement disponibles, à l'intérieur de la procédure :

\$0

Nom de la procédure elle-même

\$1, \$2, \$3 ...

Valeurs des paramètres reçus

\$*

Liste complète des paramètres

\$#

Nombre de paramètres

\$?

Code retour de la dernière commande appelée.

Les scripts peuvent retourner un **code retour** via l'instruction **exit**.

Il convient de retourner **0 en cas de succès** et une valeur dans l'intervalle [1 , 127].

Exécutions conditionnelles :

Notations disponibles exploitant le code retour

<code>cde1 && cde2</code>	Lancer la deuxième commande si la première a réussi.
-----------------------------------	------------------------------------------------------

<code>cde1 cde2</code>	Lancer la deuxième commande si la première a échoué.
---------------------------	------------------------------------------------------

Premier exemple de procédure élémentaire

```
#!/usr/bin/ksh
echo "Je m'appelle $0"
echo "Nombre d'arguments : $#"
```

```
echo "Liste des arguments : $*"
```

.2 Instructions de Contrôle

.a Tests

.b Tests simples

If commande

then

liste_de_commandes

fi

ou bien :

if commande

then

liste_de_commandes

else

liste_de_commandes

fi

```
if who | grep toto > /dev/null
then
    echo "toto est connecte"
else
    echo "toto n'est pas connecte"
fi
```

NB: L'instruction *if* ne peut être suivie que d'une commande et pas directement d'une expression.

.c Test séquentiels

1) Tests imbriqués :

If commande

then

liste_de_commandes

else if commande

then

liste_de_commandes

else if commande

then

liste_de_commandes

else # Ce "else" est bien sur facultatif

liste_de_commandes

fi

fi

fi

2) *elif* plutôt que *esle if* :

L'utilisation de la syntaxe en deux mots *e/se if* nécessite autant de *fi* que de *if*. Il existe une autre forme *elif* qui ne nécessite qu'un seul *fi* pour clore l'instruction.

If commande

then

liste_de_commandes

Elif commande

then

.....

.....

fi

2775cbe5ce
Global Knowledge

.d La commande test

Cette commande évalue une expression booléenne et retourne la valeur 0 si cette expression est vraie.

Il s'agit d'une commande interne assez indispensable puisque l'instruction *if* ne peut être suivie que d'une commande et pas directement d'une expression.

test expression

ou

[expression] (Les crochets doivent être délimités par des espaces)

Quelques expressions

-r fichier	Droit de lecture sur le fichier
-w fichier	Droit d'écriture sur le fichier
-x fichier	Droit d'exécution sur le fichier
-f fichier	Il s'agit d'un fichier ordinaire
-d fichier	Il s'agit d'un répertoire
-L fichier	Il s'agit d'un lien symbolique
-z "chaîne"	La chaîne est de longueur nulle
-n "chaîne"	La chaîne est de longueur non nulle
chaîne1 = chaîne2	Les deux chaînes sont identiques
chaîne1 != chaîne2	Les deux chaînes sont différentes
entier1 -eq entier2	Les deux entiers sont égaux

Autres opérateurs sur les entiers : -ne -gt -ge -lt -le

!	Négation
-o	OU
-a	ET

Exemples simples de syntaxe

```
if    test $# -eq 0
then
    echo "Usage : $0 argument" ; exit 1
fi

if    test -f $1
then
    echo "$1 est un fichier ordinaire"
elif [ -d $1 ]
then
    echo "$1 est un repertoire"
else
    echo "Ni fichier ordinaire, ni repertoire"
fi

if    test -f $1 -a -x $1
then
    echo "$1 est un fichier ordinaire et executable"
fi

midi=12
heure=`date '+%H'`
if    test $heure -ge $midi
then
    echo "Il est plus de midi"
else
    echo "Il n'est pas encore midi"
fi
```

.e Boucles

.f La boucle for

Cette boucle permet d'exécuter plusieurs fois une liste de commandes. A chaque pas de boucle, la variable choisie comme indice prend une valeur dans une liste donnée.

for variable **in** liste_de_valeurs

do

liste_de_commandes

done

Exemples :

Boucler sur tous les paramètres de la procédure

```
for arg in $*
do
    liste de commandes
done
```

NB: La première ligne de cette boucle peut s'abréger en : `for arg`

Boucler sur tous les noms de fichiers du répertoire courant

```
for nomfic in *
do
    liste de commandes
done
```

Faire cinq fois un traitement

```
for i in 1 2 3 4 5
do
    liste de commandes
done
```

.g Les boucles *while* et *until*

La boucle *while* permet d'exécuter une liste de commandes tant qu'une commande est vraie. La boucle *until* permet d'exécuter une liste de commandes jusqu'à ce qu'une commande soit vraie.

La commande interne *test* reste assez indispensable ici puisque les instructions *while* ou *until* ne peuvent être suivies que d'une commande et pas directement d'une expression.

while commande

do

liste_de_commandes

done

ou

until commande

do

liste_de_commandes

done

```
who | grep $1 > /dev/null
while test $? -eq 0
do
    echo "$1 est connecte"
    sleep 15
    who | grep $1 > /dev/null
done
echo "$1 n'est pas ou n'est plus connecte"
```

.h Traitements associés aux boucles

Pour utiliser de façon réaliste les boucles, il est souvent nécessaire de savoir effectuer des lectures au clavier, des calculs élémentaires sur des entiers ou des branchements.

.i Lectures au clavier

read liste_de_variables

Cette commande lit une ligne au clavier. Chaque mot est respectivement affecté à chacune des variables de la liste.

```
reponse=o
until test "$reponse" = n
do
    echo "Voulez-vous continuer (o/n) : \c"
    read reponse
done
```

NB: Dans la commande test, il est important de gérer la possible lecture vide en utilisant judicieusement les doubles quotes ou via un test préalable comme ci-dessous. Il convient en effet de toujours fournir le nombre d'arguments attendus à cette commande.

```
reponse=o
until test $reponse = n
do
    echo "Voulez-vous continuer (o/n) : \c"
    read reponse
    if test -z "$reponse"
    then
        reponse=n
    fi
done
```

.j Expressions arithmétiques : let et expr

La commande intégrée **let** permet d'évaluer des expressions arithmétiques. On peut aussi utiliser la notation **((expression))**.

Les opérateurs utilisables sont pratiquement ceux du langage C.

```
x=10
```

```
let x=x+1
```

*On ne doit pas utiliser d'espaces avec la commande **let***

```
(( x = x * 2 ))
```

Les doubles parenthèses permettent de mettre des espaces

```
(( x = ( x - 2 ) * 3 ))
```

Les parenthèses modifient les priorités des opérateurs

On peut également déclarer des entiers (**integer**) pour bâtir des expressions directes.

```
integer i=0
```

```
while (( i < 100 ))
```

let ou (()) utilisés directement dans un test

```
do
```

```
.....
```

```
i=i+1
```

On peut bâtir une "expression entière" directement

```
done
```

Commande **expr** :

Syntaxe:

expr opérande1 opérateur opérande2

ou bien :

expr expression

Opérateurs:

+ - / * %

= != < <= > >=

substr length

Exemples:

```
$ v1=10 ; v2=15
```

```
$ expr $v1 + $2
```

```
35
```

```
$ expr length ``bonjour``
```

```
7
```


.k Instructions de branchement

Il est possible d'utiliser des instructions de branchement pour sortir brutalement des boucles ou pour sauter les instructions restantes du corps de boucle :

break [n]

Quitter brutalement une boucle.(éventuellement pour n niveaux)

continue [n]

Retourner au test (while et until) ou passer à la valeur suivante (for).
(éventuellement pour n niveaux)

2775cbe5ce
Global Knowledge

.I Aiguillage

```
case valeur in  
  
    motif1)                liste_de_commandes  
  
    ;;  
  
    motif2)                liste_de_commandes  
  
    ;;  
  
    .....  
  
esac
```

Le caractère | peut être utilisé pour réaliser un OU.

Les différents motifs sont examinés dans l'ordre. Si un des motifs correspond à la valeur de test, la liste de commandes correspondantes est exécutée et il y a ensuite sortie de l'instruction.

Le motif * peut être utilisé comme dernier motif et jouer ainsi le rôle de "cas par défaut".

```
case $# in  
    1|2) echo "On a reçu 1 ou 2 arguments"  
        ;;  
    3)   echo "On a reçu 3 arguments"  
        ;;  
    *)   echo "Mauvais appel, il faut 1 ou 2 ou 3 arguments"  
        exit 1  
        ;;  
esac  
echo "Suite du traitement"
```

.3 Quelques aspects complémentaires

.a Fonctions

function nom

{

.....

}

Les fonctions peuvent :

- Recevoir des arguments (\$1, \$2...)
 - Modifier un argument uniquement via l'instruction *read*
 - Retourner une valeur via l'instruction **return** (qui positionnera \$?)
 - Utiliser des variables locales définies via l'instruction **typeset**
- (par défaut, les variables du script (globales) sont accessibles dans les fonctions)

```
# Exemple d'une fonction "question - reponse"
function mafonction
{
    echo $1 ; read $2
}
# Exemple d'appel ($1 est ici l'argument fourni au script)
mafonction $1 rep
echo "Reponse = $rep"
```

.b Gestion des signaux

trap action événement

La commande interne **trap** permet d'exécuter une action lorsque se produit un certain événement (principalement l'arrivée de signaux).

action :

Une liste de commandes

La commande vide " (2 simples quotes), pour ignorer un événement

Le signe - ou une action non mentionnée, pour rétablir le comportement par défaut)

événement :

Nom ou numéro d'un signal

2 (INTR)

Interruption clavier (Ctrl c)

3 (QUIT)

Interruption clavier (Ctrl \) générant un fichier *core*

15 (TERM)

Commande kill sans option

0 (EXIT)

Déconnexion (fin du shell)

```
fictemp=/tmp/fic$$ # La notation $$ désigne le PID du shell courant
trap 'rm -f $fictemp ; exit 1' INTR
```

```
trap '' 2 3 # Protection contre les interruptions clavier
```

```
trap - 2 3 # Rétablissement du comportement par défaut
```

```
trap '$HOME/.logout' EXIT # à mettre dans son fichier ".profile"
```

.c Autres commandes internes

. (point) Interpréter la procédure sans créer de shell fils.

: (deux points) Commande vide toujours vraie (pour constituer des boucles infinies).

```
while :
do
.....
done
```

print Veut mettre fin aux variantes de la commande echo.

Elle accepte les caractères conventionnels suivants :

<code>\a</code>	Signal sonore
<code>\b</code>	Retour arrière
<code>\c</code>	Pas de saut de ligne
<code>\f</code>	Effacement d'écran ou saut de page
<code>\n</code>	Saut de ligne
<code>\r</code>	Retour chariot
<code>\t</code>	Tabulation
<code>\v</code>	Tabulation verticale

L'option **-u2** permet d'écrire sur l'erreur standard :

```
print -u2 "Ceci est un message d'erreur"
```

.4 TP 11 – Les bases de la programmation shell

Ce qui est fait dans cet exercice :

Après avoir pratiqué Unix pendant quelques temps, vous souhaitez automatiser ou personnaliser quelques tâches que vous exécutez régulièrement.

Cet exercice va vous présenter quelques-unes des techniques les plus couramment utilisées pour vous aider à écrire des scripts shell afin de personnaliser et d'automatiser votre environnement.

Ce qui est fait dans cet exercice :

Après avoir déroulé cet exercice vous serez capable de :

- Enumérer les concepts couramment utilisés dans l'écriture de scripts shell
- Créer et exécuter des scripts shell

Remarque : Les exercices de ce module dépendent des particularités des équipements et de la configuration des matériels de la salle.

.a Ecrire une procédure script shell

- ___ 1. Créer un script shell nommé **parametres** qui affiche les cinq lignes suivantes, en utilisant les variables prédéfinies du shell pour remplacer les "_____" par les paramètres transmis . Exécutez le script avec les arguments 10 100 1000.

Le nom de la procédure est _____.

Le premier paramètre reçu est le nombre _____.

Le deuxième paramètre le nombre _____.

Le troisième paramètre le nombre _____.

En tout il y a _____ paramètres de transmis.

- ___ 2. Créez une procédure **veriffic**, qui affiche le contenu du fichier **parametres** à condition qu'il existe. Utilisez un opérateur d'exécution conditionnelle.

- ___ 3. Modifiez la procédure **veriffic** pour remplacer le nom du fichier **parametres** par un nom de fichier qui n'existe pas **nonfic** (soyez certains qu'il n'y a pas de fichiers nommés nonfic dans votre répertoire courant). Toujours en utilisant un opérateur d'exécution conditionnelle, si la commande **ls** échoue, alors affichez un texte d'erreur **Le fichier n'a pas été trouvé**. Lancez la procédure.

- ___ 4. Modifiez la procédure **veriffic** pour que le message d'erreur affiché par la commande **ls** n'apparaisse pas à l'écran. Lancez la procédure.

- ___ 5. Modifiez la procédure **veriffic** pour pouvoir lui transmettre un argument sur la ligne de commande. Cet argument est utilisé, dans le code de la procédure, comme paramètre des commandes **ls** et **cat** . Lancez deux fois la procédure script shell, une fois avec **parametres**, la seconde fois avec **nonfic**.

.b Utilisation de **for**, **test** et **if**

- ___ 6. En utilisant la boucle **for**, modifiez la procédure **veriffic** pour qu'elle puisse recevoir plusieurs noms de fichier en paramètre au lieu d'un seul. Si les fichiers existent alors affichez leur contenu. Si les fichiers n'existent pas, alors affichez un message d'erreur comme quoi le fichier n'a pas été trouvé. Repérez quelques noms de fichier présents dans votre répertoire courant. Lancez la procédure en indiquant des noms de fichiers existants et de fichiers inexistant.

- ___ 7. Modifiez votre procédure **veriffic** pour utiliser l'opérateur **if** et la commande **test**, à la place des opérateurs d'exécution conditionnelle, pour vérifier que le nom de fichier est celui d'un fichier existant ou pas. Lancez la procédure comme vous l'avez fait à l'étape précédente.

Astuce : les codes retours sont utilisés dans cette procédure.

.c Utilisation de **while** et de **expr**

- ___ 8. Créez une procédure nommée **manger**, qui utilise une boucle **while** infinie, pour afficher **A table !** toutes les deux secondes. Lancez ce script shell. Quand vous en avez assez, interrompez la boucle.

- ___ 9. Directement sur la ligne de commande, affichez le résultat de la multiplication

de 5 fois 6.

- 10. Maintenant, toujours en utilisant **expr**, créez une procédure nommée **math** qui accepte deux nombres comme arguments sur la ligne de commande et qui affiche leur multiplication. Lancez la procédure pour calculer 5 fois 6. Testez avec d'autres nombres.

Fin de l'exercice

2775cbe5ce
Global Knowledge

ANNEXE

Correction des Travaux

Pratiques

2775cbe50e
Global Knowledge

.1 Introduction

Cette annexe rassemble les corrigés des travaux pratiques parfois appelés exercices.

.2 TPs 1 et 2 – Ouverture de session et premières commandes

.a Connexion / Changement de mot de passe

- ___ 1. Ouvrez une session avec le nom et le mot de passe qui vous ont été donnés par l'animateur. Il est de la forme **stagexx** où **xx** est un nombre comme **01**, **02** etc... Changez votre mot de passe. Le mot de passe que vous indiquez n'est pas affiché.

```
»login: stagexx ( à l'invite de connexion, où xx est le nombre qui vous a été communiqué par le formateur ou formatrice )
```

```
Password: password
```

```
»$ passwd
```

```
Changing password for user stagexx.
```

```
Changing password for stagexx.
```

```
(current) UNIX password: password
```

```
New password: stagexx
```

```
Enter new password again: stagexx
```

- ___ 2. Vérifiez que le mot de passe a été défini en vous déconnectant puis en re-connectant de nouveau.

```
»$ exit
```

```
login: stagexx
```

```
Password:
```

.b Commandes de base

- ___ 3. Affichez la date du système.

```
»$ date
```

__ 4. Affichez le calendrier de l'année 2014.

»\$ cal 2014

__ 5. Affichez le mois de février 1682 (ou bien septembre 1752 si le système est basé sur les calendriers anglo-saxons). Vous remarquez quelle particularité ? *Certains jours sont manquants. Pour cause de décalage entre les calendriers Julien et Grégorien;-)*

»\$ cal 2 1682 - ou -

»\$ cal 9 1752

__ 6. Affichez le mois de janvier pour l'année 1999 et pour l'année 99. Est-ce que 1999 et 99 affichent la même chose ? *Non il y a 1900 ans d'écart !*

»\$ cal 1 1999

»\$ cal 1 99

__ 7. Lancez la commande qui affiche les informations des utilisateurs connectés.

»\$ who

__ 8. Affichez juste votre nom de connexion.

»\$ who am i

__ 9. Lancez la commande **banner** (si elle est implémentée) pour afficher **A table !**

»\$ banner A table !

__ 10. Utilisez la commande **echo** pour afficher la chaîne de caractère **A table !**

»\$ echo A table !

__ 11. Utilisez la commande **clear** pour effacer l'écran.

»\$ clear

.c La documentation en ligne : la commande `man`

- ___ 12. Affichez la page d'utilisation du manuel en ligne lui-même par une commande `$man man <entrer>`.

Utilisez la barre d'espace pour avancer d'un écran et la touche **<Entrée>** pour avancer d'une ligne. Appuyez sur la touche **b** pour revenir en arrière. Lorsque vous avez assez lu, sortez du manuel à l'aide de la touche **q** ou **<CTRL-C>**.

- ___ 13. Recherchez les rubriques du manuel en ligne qui traitent du calendrier (« `calendar` » si la documentation est en anglais).

```
»$ man -k calendrier
```

- ___ 14. La documentation vous propose la commande **cal**. Utilisez **man**, sans option, pour afficher la syntaxe de la commande **cal**.

.d Envoyer et recevoir du courrier : commande `mail`

- ___ 12. Envoyez-vous un courrier à vous-même en utilisant la commande **mail**. Renseignez le sujet mais ignorez la copie vers un autre destinataire.

```
»$ mail stagexx ( où stagexx est votre nom de connexion )
Subject: rappel
rappel à moi même
la réunion commence à 10h00
<Ctrl-d> ( <Ctrl-d> doit être tapé en début de ligne )
EOT
```

- ___ 13. Lancez la commande **mail** pour afficher la liste des messages de votre boîte à lettre. Ouvrez le message, le sauvegardez, puis quittez l'utilitaire **mail**. Pour afficher le résumé des sous-commandes de **mail**, tapez **?** à l'invite de **mail**.

```
»$ mail
Heirloom Mail version 12.5 7/5/10. Type ? for help.
"/var/spool/mail/stage01": 1 message 1 new
>N 1 STAGE01 Wed Jan 4 17:02 19/614 "rappel"
& 1
Message 1:
From stage01@srv01.localdomain Wed Jan 4 17:02:24 2017
Return-Path: <stage01@srv01.localdomain>
X-Original-To: stage01
```

```
Delivered-To: stage01@srv01.localdomain
Date: Wed, 04 Jan 2017 17:02:23 +0100
To: stage01@srv01.localdomain
Subject: rappel
User-Agent: Heirloom mailx 12.5 7/5/10
Content-Type: text/plain; charset=iso-8859-1
From: stage01@srv01.localdomain (STAGE01)
Status: R
```

```
rappel à moi même
la réunion commence à 10h00
```

```
& s
```

```
"/home/stagexxxx/mbox" [New file] ( ce message est affiché )
```

```
& q
```

```
»$
```

- 14. Ouvrez de nouveau votre courrier et supprimez le message que vous aviez sauvegardé. Quittez le programme **mail**. Si d'autres personnes sont connectées à votre système, alors leur envoyer un courrier.

```
»$ mail -f
```

```
& d
```

```
& q
```

```
»$
```

e Communiquer avec les autres utilisateurs

- 15. Envoyez une note à toutes les personnes connectées comme quoi vous avez terminé l'exercice.

```
»$ wall
```

```
Exercice terminé ! ( je suis pas sûr qu'ils apprécient ... )
```

- 16. Pour cet exercice pratiquez de concert avec votre voisin(e) **stageyy**. Ouvrez une ligne pour saisir un texte qui est envoyé vers l'écran de votre voisin(e). Attendez que s'affiche sur votre écran une réponse de sa part, comme quoi il(elle) ne peut pas communiquer pour le moment, suivi de l'indicateur de fin de texte. Fermez à votre tour la ligne de saisie de texte et mettez ainsi fin à votre session de messagerie instantanée.

```
»$ write stageyy
```

```
C'est tout bon ?
```

```
»$ write stagexx
```

```
Trop occupé pour l'instant
```

```
<Ctrl-d>
```

.f Actions au clavier

Le but est de stopper ponctuellement l'affichage d'un texte long qui défile à l'écran, puis de reprendre le défilement.

- 17. Lancez la commande **ls -lR /** (la signification sera présentée plus tard). Par des actions au clavier: stoppez le défilement. Puis reprendre le défilement

```
»$ ls -lR /
<Ctrl-s> ( stop le défilement à l'écran )
<Ctrl-q> ( pour relancer)
<Ctrl-c> ( stop, arrête la commande )
```

- 18. Relancez la commande **ls -lR /**, mais en ne tapant que les quatre premiers caractères, SANS valider en N'appuyant PAS sur la touche **<Enter>**. Effacez la ligne saisie en tapant **<ctrl-u>**. Puis utilisez **echo** pour afficher **Fin de l'exercice**.

Cette fois faites une faute de frappe et utilisez la touche **<retour arrière>** pour corriger.

```
»$ ls -l <Ctrl-u>
»$ echo Fin de l'exercice
```

- 19. Déconnectez vous.

```
<Ctrl-d> ( fin de saisie de la ligne de commande, donc exit du
shell )
```

Fin de l'exercice

.3 TP 3 – Nommages des fichiers et répertoires

___ 1. Si ce n'est pas déjà fait, ouvrez une session

»Login: stage**xx**

»stage**xx**'s Password: stage**xx**

___ 2. Utilisez la commande **pwd**, et vérifiez que vous êtes dans votre répertoire d'accueil, **/home/stagexx**. C'est le dossier qui est ouvert lorsque vous vous connectez.

»\$ pwd

___ 3. Déplacez-vous pour que le répertoire racine (**/**) devienne votre répertoire courant.»

\$ cd /

___ 4. Vérifiez que vous vous trouvez dans le répertoire racine et affichez la liste simple puis détaillée des fichiers présents dans ce répertoire.

»\$ pwd

»\$ ls

»\$ ls -l

___ 5. Lancez la commande **ls** avec les options **-a** et **-R**. Quelle est l'utilité de chacune des options ? *-a : Affichage des fichiers "cachés" dont le nom commence par "." , -R : affichage récursif du contenu de tous les répertoires et sous-répertoires* (Remarque: la commande **ls -R** génère un grand nombre de lignes. Pour interrompre cette commande saisissez **<ctrl-c>** au clavier).

»\$ ls -a

»\$ ls -R

___ 6. Retournez dans le répertoire d'accueil (**/home/stagexx**) et listez son contenu y compris les fichiers cachés.

»\$ cd

»\$ ls -a

- ___ 7. Toujours dans votre répertoire d'accueil, créez un répertoire nommé **monrep**. Puis affichez sous forme d'une liste longue les attributs des deux répertoires **/home/stagexx** et **/home/stagexx/monrep**.

```
»$ mkdir monrep
»$ ls -ld /home/stagexx /home/stagexx/monrep
- ou bien -
»$ ls -ld . monrep
```

- ___ 8. Déplacez-vous dans le répertoire **monrep**. En utilisant la commande **touch** sans option, créez deux fichiers **monfic1** et **monfic2** (cf le manuel en ligne pour l'utilisation de la commande **touch**).

```
»$ cd monrep
»$ touch monfic1 monfic2
```

- ___ 9. Affichez sous forme d'une liste longue le contenu de votre dossier **monrep**. Quelle est la taille des deux fichiers **monfic1** et **monfic2** ? 0 Affichez de nouveau la liste longue de votre répertoire en affichant aussi le numéro d'i-node des fichiers. Le numéro d'i-node de **monfic1** : *dépend du système*, de **monfic2** : _____ *même numéro que pour monfic1*.

```
»$ ls -l
```

- ___ 10. Retournez dans votre répertoire d'accueil et utilisez la commande **ls -R** pour afficher votre arborescence de répertoires.

```
»$ cd
»$ ls -l
```

- ___ 11. Utilisez la commande **rmdir** pour supprimer le répertoire **monrep**. Est-ce possible ? *Non* Vous constatez que la commande **rmdir** ne permet pas de supprimer un répertoire NON vide. Pour ce faire vous pourrez utiliser une commande présentée dans le module suivant : **rm -r**.

```
»$ rmdir monrep ( message d'erreur comme qui monrep n'est pas vide )
»$ rm -r monrep
```

Fin de l'exercice

.4 TP 4 – Manipulations des fichiers et répertoires

.a vérifiez votre environnement

- __ 1. Si ce n'est pas déjà fait, ouvrez une session»

```
Login: stagexx
```

```
»stagexx's Password: stagexx
```

- __ 2. Utilisez la commande **pwd**, et vérifiez que vous êtes dans votre répertoire d'accueil, **/home/stagexx**. C'est le dossier qui est ouvert lorsque vous vous connectez.

```
»pwd
```

- __ 3. Listez le contenu de votre répertoire d'accueil (**/home/stagexx**), y compris les fichiers cachés.

```
»ls -a
```

.b manipulations de fichiers

- __ 4. Utilisez successivement les commandes **cat**, **less** et **more** pour afficher le contenu des fichiers **/etc/hosts** et **/etc/profile**. Remarquez les différences de comportement.

```
»$ cat /etc/hosts
```

```
»$ cat /etc/profile
```

```
»$ less /etc/hosts
```

```
»$ less /etc/profile
```

```
»$ more /etc/hosts
```

```
»$ more /etc/profile
```

- __ 5. Copiez le fichier **/bin/cat** dans votre dossier courant (répertoire d'accueil)

```
»$ cp /bin/cat .
```

- ___ 6. Copiez aussi le fichier **/usr/bin/cal** dans votre dossier courant (répertoire d'accueil)

```
»$ cp /usr/bin/cal .
```

- ___ 7. Affichez le contenu de votre répertoire courant. Vous devez voir les deux fichiers que vous venez de copier.

```
»$ ls
```

.c création et manipulation de répertoires

- ___ 8. Dans votre répertoire d'accueil, créez un sous répertoire nommez le **mescmds**.

```
»$ mkdir mescmds
```

- ___ 9. Déplacez y et renommez les deux fichiers que vous avez copiés dans votre répertoire d'accueil (**cat** et **cal**) vers **moncat** et **moncal**.

```
»$ mv cat mescmds/moncat
```

```
»$ mv cal mescmds/moncal
```

- ___ 10. Déplacez-vous dans le répertoire **mescmds**.

```
»$ cd mescmds
```

- ___ 11. Listez le contenu du répertoire courant pour vérifier que les fichiers ont bien été déplacés.

```
»$ ls
```

- ___ 12. Utilisez la commande **./moncat** de votre répertoire **mescmds** pour afficher le contenu du fichier **.bash_profile** situé dans votre répertoire d'accueil.

```
»$ ./moncat ../.bash_profile
```

- ___ 13. Retournez dans votre répertoire d'accueil.

```
»$ cd
```

- ___ 14. Dans ce répertoire d'accueil, créez un autre répertoire nommé **dubon**.

```
»$ mkdir dubon
```

- ___ 15. Copiez le fichier **/etc/profile** dans le nouveau répertoire, et nommez la copie **nveauprofile**.

```
»$ cp /etc/profile dubon/nveauprofile
```

- ___ 16. Toujours depuis votre répertoire d'accueil, utilisez la commande **cat** pour afficher la copie **nveauprofile**. Difficile de lire le début ? Utilisez **pg** (ou **more**) à la place de **cat**.

```
»$ cat dubon/nveauprofile
```

```
»$ less dubon/nveauprofile
```

- ___ 17. Le nom du fichier **nveauprofile** est trop long. Renommez le en **np**. Affichez la liste des fichiers du répertoire **dubon** pour vérifier que le changement de nom est effectif. Utilisez la commande **cat** pour afficher le contenu du fichier renommé.

```
»$ mv dubon/nveauprofile dubon/np
```

```
»$ ls dubon
```

```
»$ cat dubon/np
```

- ___ 18. C'est le moment d'avoir une vision d'ensemble : en partant de votre répertoire d'accueil, affichez de façon hiérarchique, tous vos fichiers et sous-répertoires.

```
»$ cd
```

```
»$ ls -R
```

.d suppression de répertoires

- ___ 19. Assurez-vous d'être dans votre répertoire d'accueil. Essayez de supprimer votre répertoire **dubon**. Pouvez-vous le supprimer ? *Non Pourquoi ? Il n'est pas vide.*

```
»$ rmdir dubon
```

- ___ 20. Déplacez-vous dans le répertoire **dubon**. Listez son contenu y compris les fichiers cachés. Supprimez les fichiers. Listez de nouveau le contenu du répertoire courant. Vous remarquez que **.** et **..** sont toujours présents. Le répertoire est considéré comme vide si il ne contient que ces deux entrées. Supprimez le répertoire.

```
»$ cd dubon  
»$ ls -a  
»$ rm np  
»$ ls -a  
»$ cd ..  
»$ rmdir dubon
```

Fin de l'exercice

2775cbe5ce
Global Knowledge

.5 TP 5 – Droits d'accès des fichiers

.a Affichez les attributs des fichiers

- ___ 1. Ouvrez une session utilisateur. Déplacez-vous dans le répertoire **mescmds**. Affichez la liste longue du répertoire. Notez le nom du propriétaire et les droits d'accès aux fichiers copiés lors de l'exercice précédent.

Propriétaire et permissions de **moncat** stagexx

Propriétaire et permissions de **moncal** stagexx

- ___ 2. Affichez les attributs des fichiers originaux **/bin/cat** et **/usr/bin/cal** et comparez avec les copies du répertoire **mescmds**. Vous êtes propriétaire des copies mais pas des originaux.

```
»$ cd mescmds
```

```
»$ ls -l
```

- ___ 3. Changez l'heure de modification de **moncal** et de **moncat** dans le dossier **mescmds**. Vérifiez que l'heure de modification a bien été changée. Quel est l'autre usage de la commande **touch** ? *Une façon de créer des fichiers vides.*

```
»$ touch *
```

- ___ 4. Faire en sorte de pouvoir référencer aussi bien par son nom de fichier : **moncal**, dans le répertoire **mescmds**, que par autre nom : **home_moncal**, dans votre répertoire d'accueil.

Y a-t-il une différence d'attributs entre les deux noms de fichier ? *Aucune ce sont les mêmes fichiers physiques, mais avec des noms d'accès différents*

Quel est le nombre de liens ? 2

```
»$ ln moncal /home/teamxx/home_moncal
```

-ou bien-

```
»$ ln moncal ../home_moncal
```

```
»$ ls -l moncal
```

__ 5. Déplacez-vous dans votre répertoire d'accueil et lancez **home_moncal**.

Y a-t-il une différence de comportement avec **/usr/bin/cal** ?

Aucune, ce sont les mêmes binaires

Maintenant changez les droits d'accès de **home_moncal** de façon que vous, le propriétaire du fichier, n'ayez plus que le droit de lecture. Lancez de nouveau la commande **moncal**.

Est-ce possible ? Pourquoi ? *Oui pour le changement de droit, car vous en êtes le propriétaire. Mais pas pour l'exécution.*

```
»$ cd
```

```
»$ home_moncal
```

```
»$ chmod 455 home_moncal
```

```
»$ ls -l home_moncal
```

```
»$ mescmds/moncal ( erreur il n'y a plus de droit x )
```

__ 6. Supprimez **home_moncal**. Est-ce que cela supprime **mescmds/moncal** ?

non

Pourquoi ? *Ce n'est que le nom dans le répertoire qui est supprimé. Pas le fichier sur le disque.*

```
»$ rm home_moncal
```

```
»$ ls -l mescmds/moncal
```

.b Travaillez avec les droits d'accès des fichiers ordinaires

__ 7. Déplacez vous dans le dossier **mescmds**. Utilisez la syntaxe symbolique de la commande **chmod** pour supprimer le droit de lecture pour le bit *other* du fichier **moncat**. Vérifiez que les droits ont été changés.

```
»$ cd mescmds
```

```
»$ chmod o-r moncat
```

```
»$ ls -l moncat
```

- __ 8. En utilisant la syntaxe octale, changez les droits d'accès de **moncat** de façon à ce que le propriétaire ait le droit de lecture uniquement, avec aucun droit pour tous les autres. Vérifiez que les droits ont été changés.

```
»$ chmod 400 moncat
```

```
»$ ls -l moncat
```

- __ 9. Utilisez **moncat** pour afficher le contenu du fichier **.bash_profile**. Est-ce possible ?

```
»$ moncat ../.bash_profile
```

-ou bien-

```
»$ moncat /home/teamxx/.bash_profile
```

- __ 10. Déplacez-vous dans votre répertoire d'accueil. Vérifiez que vous y êtes bien positionné.

```
»$ cd
```

```
»$ pwd
```

.c Travailler avec les droits d'accès aux répertoires

- __ 11. Modifiez les droits du répertoire **mescmds** pour que vous n'ayez que le droit de lecture.

```
»$ chmod u-wx mescmds
```

-ou bien-

```
»$ chmod u=r mescmds
```

-ou bien-

```
»$ chmod 455 mescmds
```

- __ 12. Affichez la liste des attributs pour vérifier que les droits ont été correctement modifiés.

```
»$ ls -l /home/stagexx
```

-ou bien-

```
»$ ls -ld mescmds
```

- __ 13. Cherchez à vous déplacer dans le répertoire **mescmds** pour afficher une liste simple du contenu du répertoire. Puis une liste détaillée. Est-ce c'est possible ?

Pourquoi ?

Non vous ne pouvez pas vous y déplacer car il manque le droit de passage

```
»$ ls mescmds
```

```
»$ ls -l mescmds
```

- __ 14. Cherchez à lancer **moncal**. Est-ce que c'est possible ?

Pourquoi ?

Non car il manque le droit de passage

```
»$ mescmds/moncal
```

- __ 15. Cherchez à supprimer **moncal** Est-ce que c'est possible ?

Pourquoi ?

Non car il manque le droit de passage.

```
»$ rm mescmds/moncal
```

- __ 16. Redonnez les droits d'origine **rwxr-xr-x** au répertoire **mescmds**, puis supprimez **moncal**.

```
»$ rm mescmds/moncal
```


- ___ 17. Si le temps le permet, testez d'autres combinaisons de droits d'accès.
Lorsque vous avez fini, assurez-vous de repositionner les droits **rwX** pour vous même le propriétaire.

Fin de l'exercice

2775cbe5ce
Global Knowledge

.6 TP 6 – L'éditeur vi

Ce qui est fait dans ce TP :

Le but de cet exercice est de vous donner la possibilité de créer et modifier des fichiers en utilisant l'éditeur UNIX le plus courant : vi. Une bonne compréhension de l'éditeur vi est essentielle pour mener à bien le reste des exercices de ce stage.

Ce que vous allez savoir faire :

Après avoir complété ces travaux pratiques, vous serez capable de :

- Créez un fichier avec vi
- Sauvegardez et quittez le fichier et quitter sans sauvegarder
- Manipuler un fichier en utilisant différentes touches de déplacement du curseur
- Ajouter, supprimer et faire des changements de texte dans un fichier
- Définir les options pour personnaliser la session d'édition
- Appelez le mode commande d'édition en ligne.

Introduction

L'éditeur **vi** est basé sur un logiciel développé par l'Université de Californie à Berkeley, division Informatique. L'éditeur **vi**, prononcez «vé-i » (abréviation de visuel), comporte des commandes pour créer, modifier, ajouter ou supprimer des fichiers. Les exercices suivants vous familiariseront avec quelques-unes des principales caractéristiques et des fonctions de **vi**.

.a Création d'un fichier

- ___ 1. Assurez-vous d'être dans votre répertoire d'accueil. Avec **vi** créez un fichier nommé **vitest**.

»\$ cd

»\$ pwd

»\$ vi vittest

- ___ 2. Lorsque vous lancez **vi** vous êtes en mode commande. Appuyez sur la touche **i** (insert) pour basculer en mode écriture. Vous pouvez aussi appuyer sur la touche **a** (ajout). L'utilisation de **i** ou de **a** fait que le début du texte saisi ensuite apparaît avant (insert) ou après (ajout) le curseur. Par défaut, il n'y a pas d'indication comme quoi vous avez basculé en mode écriture.

Basculez du mode écriture au mode commande en appuyant sur la touche <Echap> (Echappement) du clavier. Remarquez que si vous appuyez une deuxième fois sur la touche <Echap> alors votre matériel peut émettre un « bip » et / ou votre écran flasher. Cela indique que vous êtes déjà en mode commande et n'a pas d'autre conséquence. Appuyez de nouveau sur la touche **i** pour repasser en mode écriture, puis passez à l'étape suivante.

»i

»<Echap>

»<Echap> (vous entendez un "bip" et / ou un flash écran)

»i

- ___ 3. Saisissez le texte ci-dessous *exactement* comme il est présenté. Ligne à ligne. Puis la liste des caractères de l'alphabet, un caractère par ligne. L'ajout de l'alphabet est un moyen facile de remplir plusieurs écrans de texte nécessaires à une utilisation ultérieure

This Ceci est une session de formation sur l'utilisation de l'editeur vi. On a besoin de plus de lignes pour apprendre les commandes les plus courantes de l'éditeur. On est maintenant dans le mode ecriture et nous allons passer tout de suite dans le mode commande.

**a
b
c
d
...
z**

- ___ 4. Repassez en mode commande. Puis enregistrez et quittez **vi**. Remarquez que dès que vous appuyez sur la touche ":" (deux points), ces deux points apparaissent en bas de l'écran, en dessous de la dernière ligne saisie. Une fois le buffer vidé et le fichier fermé, vous verrez un message indiquant le nombre de lignes et de caractères enregistrés dans le fichier.

»<Echap> (retour en mode commande).

»:wq (ou bien <shift-zz>, ou bien ":x" qui est une autre façon d'enregistrer les modifications, puis de quitter).

.b Déplacement du curseur

- 5. Ouvrez le fichier **vitest** avec **vi**. Remarquez sur la dernière ligne de l'écran l'affichage du nom du fichier et du nombre de caractères. Vous êtes en mode commande.

»\$ vi vitest

- 6. Pour déplacer le curseur, vous pouvez utiliser les flèches du clavier ou les touches **h**, **j**, **k**, **l**. Déplacez le curseur d'une ligne vers le bas, de quelques caractères vers la droite, puis vers la gauche et remontez le curseur sur la ligne du dessus.

»j (une ligne vers le bas)

»k (une ligne vers le haut)

»l (un caractère à droite)

»h (un caractère à gauche)

»Pratiquez les mêmes déplacements avec les flèches du clavier.

- 7. Plutôt que de déplacer le curseur ligne à ligne, ou caractère à caractère, vous allez provoquer un déplacement global sur l'écran ou sur les lignes. Réalisez les déplacements suivants :

. avancez d'une page,

. reculez d'une page,

. déplacez le curseur sur la dernière ligne du fichier,

- . ramenez le curseur sur la première ligne du fichier,
- . positionnez le curseur sur la 4eme ligne du fichier,
- . déplacez le curseur à la fin de la ligne,
- . ramenez le curseur au début de ligne.

»<Ctrl-f> ou éventuellement la touche <PgDn> qui elle peut ne pas fonctionner.

»<Ctrl-b> ou éventuellement la touche <PgUp> qui elle peut ne pas fonctionner.

»<Ctrl-u>

»<shift-g>

»1<shift-g> ou :1 <Entrée>

»4<shift-g> ou :4 <Entrée>

»\$

»0 (la touche zéro)

- ___ 8. Déplacez le curseur au début du fichier. Recherchez le mot `écriture`. Le curseur doit être sur le `e` initial. Basculez en insertion et ajoutez le mot `texte`. N'oubliez pas l'espace après le mot saisi.

»1<shift-g> ou :1

»/écriture

»i

»texte

- ___ 9. Déplacez le curseur sur l'espace après le mot `écriture` sur la même ligne. Insérez une virgule. Vous êtes toujours en mode écriture.

»<Echap>

»Positionnez le curseur après le mot `écriture`

»i, (virgule)

- ___ 10. Basculez en mode commande. Placez le curseur n'importe où sur la ligne qui commence par `"de l'éditeur vi"`. Insérez en dessous une ligne vide pour former deux paragraphes.

»<Echap>

»Placez le curseur sur la ligne qui commence par "de l'editeur vi"

»o (o minuscule ouvre une ligne après le curseur)

- __ 11. Ouvrir une ligne vide, comme vous venez de le faire, vous bascule dans le mode écriture ; donc, basculez en mode commande. Enregistrez vos modifications SANS QUITTER l'éditeur.

»<Echap>

»:w

- __ 12. En restant en mode de commande, retirez les caractères c, e, g, et laissez les lignes vides à leur place, en d'autres termes, ne pas supprimer la ligne, juste le caractère. Ensuite, supprimer les lignes vides. Cela vous fait pratiquer les deux fonctions de suppression.

» Placez le curseur sur c; Press x

» Placez le curseur sur e; Press x

» Placez le curseur sur g; Press x

» Placez le curseur sur chaque ligne vide. Tapez dd

- __ 13. Maintenant remplacez le caractère h par un z .

» Placez le curseur sur h

» Appuyez sur r (remplace 1 caractère)

»:z

- __ 14. Vous décidez de ne pas vouloir enregistrer les modifications apportées aux caractères alphabétiques. Quittez la session vi sans enregistrer les modifications apportées depuis la dernière sauvegarde.

»:q!

- __ 15. Editez le fichier **vitest** une nouvelle fois. Tout d'abord, copiez le 1er paragraphe (y compris la ligne vide), une ligne à la fois, à la fin du fichier. Quand c'est terminé, copiez le second paragraphe à la fin du fichier, cette fois ci en une seule fois.

»\$ **vi vtest**

» Le curseur sur la première ligne; Tapez **yy**

»**<shift-g>**; Tapez **p**

»**2<shift-g>**; Tapez **yy**

»**<shift-g>**; Tapez **p**

»**3<shift-g>**; Tapez **yy**

»**<shift-g>**; Tapez **p**

»**4<shift-g>**; Tapez **3yy**

»**<shift-g>**; Tapez **p**

___ 16. Vous décidez que les lignes que vous venez d'ajouter à la fin du fichier sont de trop. Supprimez-les en une seule commande.

» Positionnez le curseur sur la première ligne des six lignes copiées à la fin du fichier.

»**6dd**

___ 17. Maintenant, avant de faire quoi que ce soit d'autre avec ce fichier, vous décidez que vous devez insérer la date et l'heure à la première ligne du fichier. Tout ça, sans quitter l'éditeur **vi**.

»**!:date > datefic**

» N'appuyez pas sur <Entrée> lorsque le message "Press return to continue" apparaît.

»**:0r datefic**

» Appuyez <Entrée> deux fois pour continuer.

.C Personnalisation

___ 18. Grâce à la commande **set**, on peut activer temporairement certaines options pour la durée d'une session de **vi**. Assurez-vous d'être en mode commande et activez les options suivantes :

. césure automatique des mots à 15 espaces avant la marge droite,

. affichage du texte INPUT MODE lorsque vous êtes en mode écriture,

. affichage de la numérotation des lignes.

```
»1<shift-g>
```

```
»<Echap>
```

```
»:set wrapmargin=15 ( pas d'espace ni avant, ni après le signe = )
```

```
»:set showmode
```

```
»:set number
```

__ 19. Testez chacune des options que vous venez d'activer.

» Les lignes doivent être numérotées,

» Passez en mode écriture par **i** ou **a**. Vous devez voir apparaître INPUT MODE en bas à droite de votre écran.

» Tapez quelques lignes de texte pour tester le retour à la ligne automatique.

» Basculez en mode commande en appuyant sur <Echap>. Le message INPUT TEXTE doit disparaître.

__ 20. Enregistrez et quittez l'éditeur.

```
»:wq
```

.d Edition de l'historique de la ligne des commandes

__ 21. Maintenant que vous êtes familiarisé avec les différents modes **vi** et de ses commandes, passons à la manipulation de l'historique de la ligne de commande du shell. Pour activer ce mode de fonctionnement, sur une ligne de commande du terminal, saisissez la commande **set -o vi**

```
»$ set -o vi
```

__ 22. Maintenant vous pouvez rappeler les commandes lancées auparavant, les modifier, et les relancer. Générez un historique de commandes. Affichez la liste des fichiers du répertoire **/usr** (juste les noms, pas tous les attributs). Avec la commande **cat** affichez le contenu du fichier **/etc/filesystems**. Lancez **echo hello ...**

```
»$ ls /usr
```


»\$ **cat /etc/filesystems**

»\$ **echo hello**

- 23. Supposons que vous vouliez modifier une des commandes que vous venez de lancer. Appuyez sur <Echap> pour basculer en mode de commande **vi**. Puis appuyez sur la touche **k** plusieurs fois pour remonter la liste des commandes. Appuyez sur **j** pour redescendre. Ce rappel de commandes est fait grâce à un buffer qui contient les commandes que vous avez déjà lancées. Ces commandes sont stockées dans votre fichier **.bash_history** situé dans votre répertoire d'accueil.

»<Echap>

»**k** (remonte l'historique des commandes stockées dans le buffer).

»**j** (descend dans l'historique des commandes).

- 24. Retrouvez la commande **ls**. Utilisez la touche **l** pour avancer le curseur sur le **/** de **/usr**. (remarque : les flèches du clavier peuvent faire quitter la ligne courante et insérer des caractères indésirables. Vous devez utiliser les touche **l** pour avancer vers la droite, et **h** pour reculer vers la gauche). Utilisez la touche **i** pour basculer en mode insertion et ajoutez **-l** pour afficher cette fois une liste détaillée des fichiers. Lancez la commande modifiée.

»**k** (pour retrouver la commande **ls /usr**)

»**l** (pour se placer sur le **/**)

»**i** (pour basculer en insertion. Ou bien utilisez **a** pour ajouter si le curseur est placé sur l'espace avant le **/**)

»**-l**

»<Echap>

- 25 . Rappelez la commande **cat**. Cette fois afficher le contenu du fichier **/etc/passwd**.

»<Echap>

»**k** (pour retrouver la commande **cat**)

»**l** (pour déplacer le curseur sur le **f** de **filesystems**)

»**D** (pour effacer le reste de la ligne, ou **dw** pour effacer le mot)

»**a** (pour ajouter du texte)

»**passwd**

»<Entrée>

- 26. Rappelez la commande **cat**. Allez à la fin de la ligne (rappel : **\$**). Ajoutez un "**|**" ("pipe") à la fin de la commande pour envoyer le texte généré en sortie vers la commande **wc** et comptez le nombre de lignes.

»<Echap>

»**k** (pour remonter jusqu'à la commande **cat**)

»**\$**

»**a**

»**| wc -l**

»<Entrée>

Fin de l'exercice

2775cbe5ce
Global Knowledge

.7 TP 7 - Gestion des process et redirections

.a Process courant

___ 1. Ouvrez une session et affichez votre numéro de process courant

»\$ `ps` → la 1ere ligne décrit le process courant, shell de connexion. Le PID du process courant est aussi stocké dans la variable `$$` dont on affiche le contenu par `echo $$`

___ 2. Créez un sous-shell en lançant **bash** (ou **ksh** selon votre environnement). Quel est le PID de ce sous-shell ? Est-il différent de votre process de obtenu à votre connexion ? _____

»\$ `ps -f` → Ce sous-shell a pour PPID le process dont le PID a été relevé à l'étape précédente. Une fois cette ligne repérée, on obtient le PID du sous-shell courant. Autre façon: ce sous-shell courant a son PID maintenant stocké dans `$$`. On affiche le contenu par `echo $$`

___ 3. Lancez la commande **ls -lR / > outfile 2> errfile &** puis affichez la liste des process lancés depuis votre terminal. Repérez la ligne de la commande **ls**. Cette commande **ls** se termine quand elle a fini de lister tous les fichiers de l'arborescence. (Remarque : si **ls** se termine trop vite, alors lancez la commande **sleep 30 &** qui provoque une pause de 30 secondes).

»\$ `ls -lR / > outfile 2> errfile &`

»\$ `ps -f`

___ 4. Terminez votre shell enfant. Qu'est-ce qui se passerait si vous tapiez de nouveau **exit** ? _____

.b Optionnel - Contrôle des process lancés en tâche de fond.

___ 5. Avec **vi** créez un script de commandes shell nommé **sctest** qui contient les lignes suivantes:

sleep 30

ls -lR / &

Rendez le exécutable. Puis lancez la commande

\$./sctest

en avant-plan.

```
»$ vi sctest
```

(appuyez sur la touche **i** pour passer en mode écriture)

```
sleep 120
```

```
ls -lR / &
```

(appuyez sur la touche <Echap> suivi par **:wq** pour sauve et quitte)

```
»chmod 700 sctest
```

```
»$ ./sctest
```

___ 6. ___ 6. La pause de 30 secondes permet de suspendre le job. Suspendez le job que vous venez de lancer.

```
»$ <Ctrl-z>
```

___ 7. Affichez la liste des jobs que vous avez lancés sur le système et relancez en tâche de fond celui que vous avez suspendu ci-dessus.

```
»$ jobs
```

```
»$ bg %jobno
```

___ 8. Ramenez le job en avant-plan.

```
»$ fg %jobno
```

___ 9. Lancez le script **sctest** avec **nohup** et en tâche de fond. Notez le numéro de job, son PID, puis déconnectez vous.

```
»$ nohup ./sctest &
```

```
»$ jobs -l
```

```
»$ exit ( un message informe qu'il y a des jobs en cours d'exécution )
```

```
»$ exit
```

___ 10. Connectez-vous de nouveau (ou bien depuis un autre terminal) et vérifiez que le process est toujours en cours d'exécution.

»Login: stagexx

»stagexx's Password: stagexx

»\$ ps -ef (retrouvez le PID relevé à l'étape précédente)

- ___ 11. Une fois le process terminé, affichez le fichier qui contient le texte de sortie **outfic**. (si vous n'avez pas explicitement redirigé la sortie, alors par défaut le système crée un fichier qui se nomme **nohup.out** dans le répertoire d'où a été lancée la commande).

»\$ less /home/stagexx/outfic

.c Terminer un process

- ___ 13. Lancez en tâche de fond, avec redirection des sorties, la commande **ls -lR /**. Cette commande met du temps avant de se terminer. Notez le PID du process _____

»\$ ls -lR / > outfic 2> errfic &

- ___ 14. Si vous n'avez pas pu noter le PID après avoir lancé la commande en tâche de fond, comment pouvez-vous le retrouver ? _____

Une fois le PID connu, tuez le process. Vérifiez qu'il a bien disparu.

»\$ ps -f

»\$ kill <pid>

»\$ ps -f

.d Les redirections

- ___ 15. Avec la commande **cat** et le mécanisme de redirection, créez un fichier de texte nommé **soleil** qui contient quelques lignes de texte. Utilisez <Ctrl-d> au

début d'une nouvelle ligne pour mettre fin à votre saisie et retourner à l'invite shell **\$** . Affichez le contenu du fichier pour vérifier.

```
»$ cat > soleil
```

Saisissez plusieurs lignes de texte

<Ctrl-d> au début d'une nouvelle ligne pour retourner à l'invite **\$** du shell.

```
»$ cat soleil
```

- ___ 16. Avec la commande **cat** ajoutez plusieurs lignes de texte au fichier **soleil**. Vérifiez vos modifications en affichant le contenu de ce fichier.

```
»$ cat >> soleil ( pas d'espace entre les >> )
```

```
»$ cat soleil
```

- ___ 17. En utilisant la commande **mail** de la messagerie, envoyez vous le contenu du fichier **soleil**. Patientez une minute puis ouvrez votre courrier, supprimez le et quittez votre programme de messagerie.

```
»$ mail stagexx < soleil
```

```
»$ mail
```

```
»? t
```

```
»? d
```

```
»? q
```

.e Tubes (pipes) et filtres

- ___ 18. Affichez le contenu de votre répertoire courant avec la commande **ls**. Notez le nombre de fichiers: _____

```
»$ ls
```

- ___ 19. Listez de nouveau le contenu de votre répertoire courant, mais cette fois redirigez la sortie vers un fichier nommé **temp**.

```
»$ ls > temp
```

- ___ 20. Utilisez la commande appropriée pour compter le nombre de mots du fichier **temp**. Est-ce le même nombre qu'à l'étape 18 ? _____ si non pourquoi ? _____

Affichez le contenu du fichier **temp**. Supprimez-le.

```
»$ wc -w temp
```

```
»$ cat temp
```

```
»$ rm temp
```

- __ 21. Cette fois utilisez un "tube" "|" (pipe) pour compter le nombre de fichiers du répertoire courant. C'est bien le résultat attendu ? _____

C'est le même nombre qu'à l'étape 18 ?

```
»$ ls | wc -w
```

- __ 22. Reprenez la ligne de commande lancée à l'étape 21, mais cette fois insérez la commande **tee** entre les deux commandes pour écrire dans un fichier nommé **soleil2**. Est-ce que le nombre de fichier est affiché ?

Vérifiez que le fichier **soleil2** contient bien ce qui est prévu.

```
»$ ls | tee soleil2 | wc -w
```

```
»$ cat soleil2
```

- __ 23. Listez le contenu du répertoire courant en ordre inversé. Envoyez le résultat dans un fichier **soleil3**, et aussi vers une commande pour compter le nombre de mots de la liste inversée. Ajoutez ce nombre au fichier **soleil3**. N'oubliez pas d'utiliser la redirection en mode ajout, sinon vous pourriez avoir des résultats inattendus. On ne peut pas utiliser une simple redirection en création car le fichier est utilisé deux fois dans la même commande globale de la ligne. Vous pouvez tester les conséquences si vous êtes curieux.

```
»$ ls -r | tee soleil3 | wc -w >> soleil3
```

```
»$ cat soleil3
```

- __ 24. Dans le dossier **/dev** Il y a un fichier spécial qui représente votre terminal. Affichez le nom du fichier qui est associé à votre terminal. Il se présente sous la forme **tty0**, **lft0** ou **pts/x** . Répétez la commande de l'étape précédente, mais avec deux changements :

a. plutôt que d'utiliser le fichier ordinaire **soleil3**, la commande **tee** envoie le

résultat à votre écran (`/dev/<nom de votre terminal>`)

b. ne renvoyez pas le résultat de la commande **wc** vers le fichier **soleil3** de façon à ce que le comptage soit envoyé à l'écran.

```
»$ who am i
```

-ou-

```
»$ tty
```

```
»$ ls -r | tee /dev/pts/xx | wc -w
```

.f Optionnel - Commandes groupées et annulation de fin de ligne

- ___ 25. Sur la même ligne de commande affichez la date système, qui est connecté, le nom de votre répertoire courant et la liste des fichiers de ce répertoire. Est-ce que ces commandes ont un lien entre elles ?

```
»$ date ; who ; pwd ; ls
```

- ___ 26. Cette étape a deux buts : le premier est d'annuler la fin de ligne de commande pour une commande trop longue pour tenir sur une seule ligne, le second est de tester ce que vous avez appris en vous faisant rédiger une très longue commande.

Vous pouvez choisir de couper la ligne où vous voulez, mais en tout cas ne tapez pas jusqu'à dépasser le bord droit de l'écran. Une fois terminé, testez votre sortie en affichant le contenu des fichiers qui ont été créés. C'est une longue commande reliée par des tubes et une redirection :

a. Générez une liste détaillée de tous les fichiers du répertoire courant, y compris les fichiers cachés,

b. récupérez le résultat pour l'enregistrer dans un fichier nommé **listing.inverse** et renvoyez ce même flux de texte vers une commande qui compte uniquement le nombre de mots,

c. récupérez ce nombre de mots et enregistrez-le dans quatre fichiers nommés de fic1 à fic4,

d. enfin, envoyez le résultat précédent vers une commande qui compte le nombre de ligne et renvoyez ce nombre vers un fichier nommé fic5.

```
»$ ls -al | tee listing.inverse | wc -w | tee fic1 \<Entrée>
```

```
> | tee fic2 | tee fic3 | tee fic4 | wc -l > fic5
```

(le symbole > qui débute la deuxième ligne n'est pas à saisir, il représente le second prompt qui apparaît lorsque le système interprète que la commande n'est pas terminée).

Fin de l'exercice

2775cbe5ce
Global Knowledge

.8 TP 8 – Eléments du Shell, variables et jokers

.a Variables

__ 1. Affichez les variables déclarées dans votre shell,

```
»$ set | more
```

__ 2. Déclarez une variable nommée **dejeuner** qui contient la valeur **pizza**. Et une variable **diner** qui contient **jambon**. Avec la commande **echo** affichez le contenu des variables. Retrouvez-les dans la liste des variables du shell courant.

```
»$ déjeuner=pizza
```

```
»$ diner=jambon
```

```
»$ echo $déjeuner ; echo $diner
```

```
»$ set
```

__ 3. En utilisant les variables que vous venez de définir, affichez le message : **Ce jour au déjeuner il y a pizza et au diner jambon.**

```
»$ echo Ce jour au déjeuner il y a $déjeuner et au diner $diner
```

__ 4. Supprimez les deux variables. Assurez-vous qu'elles ne sont plus dans la liste des variables du process courant.

```
»$ unset déjeuner
```

```
»$ unset diner
```

```
»$ set
```

__ 5. Affichez les valeurs de vos deux invites (prompt en anglais) de ligne de commande.

```
»$ echo $PS1
```

```
»$ echo $PS2
```

- ___ 6. Modifiez la première chaîne d'invite par **"vous desirez ? "**. Pourquoi faut-il des guillemets (avec des apostrophes cela fonctionne aussi) pour **"vous desirez ? "** *Si il n'y a pas de guillemets alors le ? est interprété comme un joker.*

```
»$ PS1="Vous desirez ? "
```

- ___ 7. Modifiez le second prompt (invite) par : **"Quoi d'autre ? "**. Testez en tapant une ligne de commande comme **ls -l**, mais en annulant le caractère de fin de ligne. Une fois fait, réaffectez les variables d'invite avec leurs valeurs initiales. Pourquoi faut-il utiliser les guillemets autour du signe > pour la variable PS2 ? *Sinon le > serait interprété comme la redirection de la sortie standard.*

```
»Vous desirez ? PS2="Quoi d'autre ? "
```

```
»Vous desirez ? ls -l \
```

```
Quoi d'autre ?
```

```
»<Ctrl-c>
```

```
»Vous desirez ? PS1="$ "
```

```
»$ PS2="> "
```

- ___ 8. Vérifiez la valeur de la variable qui contient le chemin à votre répertoire d'accueil. Modifiez-la pour que votre répertoire d'accueil soit **/bin**. Utilisez les commandes **cd** et **pwd** pour constater les effets.

```
»$ echo $HOME ( ou bien utilisez set )
```

```
»$ HOME=/bin
```

```
»$ cd ; pwd
```

- ___ 9. Fermez votre session et reconnectez-vous. Quel est votre répertoire d'accueil ? **/home/styagexx** Pourquoi ? *La variable HOME est réinitialisée à la connexion.*

```
»$ exit
```

```
»login: stagexx
```

```
»stagexx's Password:
```

```
»$ cd ; pwd
```

```
»$ echo $HOME
```

.b Jokers

__ 10. Tapez **cd** pour retourner dans votre répertoire d'accueil. (celui dans lequel vous êtes placé lors de votre connexion).

```
»$ cd
```

```
»$ pwd
```

__ 11. Exécutez un simple **ls** pour afficher la liste des fichiers non cachés de votre répertoire d'accueil. Puis lancez de nouveau **ls** suivi du caractère joker pour lister ces mêmes fichiers. Quelles sont les différences d'affichage de ces deux commandes ?

Liste en colonne des fichiers du répertoire courant.

Pourquoi ?

Liste d'une suite de noms.

```
»$ ls
```

```
»$ ls *
```

__ 12. Déplacez-vous dans le répertoire **/usr/bin**. Affichez la liste des fichiers dont le nom commence par la lettre **a**.

```
»$ cd /usr/bin
```

```
»$ ls a*
```

__ 13. Affichez la liste des fichiers dont le nom est composé de deux caractères.

```
»$ ls ??
```

__ 14. Affichez la liste des fichiers dont le nom commence par une des lettres suivantes : **a, b, c** ou **d**.

```
»$ ls [abcd]*
```

-OU-

```
»$ ls [a-d]*
```

- ___ 15. Affichez la liste de tous les fichiers excepté ceux dont le nom commence par une lettre comprise entre **c** et **t**. Cette liste est très longue. Vous pouvez contrôler l'affichage en utilisant **more** (ou **pg**) grâce à un pipe.

```
»$ ls [!c-t]* | more
```

- ___ 16. Retournez dans votre répertoire d'accueil.

```
»$ cd
```

.C Substitution de commande

- ___ 17. Affichez la liste des sessions ouvertes, i.e. des utilisateurs connectés et les terminaux qui leurs sont attribués. Relancez votre commande mais cette fois en transmettant le résultat à la commande **wc** pour compter le *nombre* de sessions en cours.

```
»$ who
```

```
»$ who | wc -l
```

- ___ 18. En utilisant la substitution de commande, affichez le texte suivant : **Il y a # sessions en cours** où # représente le nombre de sessions en cours.

```
»$ echo Il y a `who | wc -l` session en cours
```

- ou -

```
»$ echo Il y a $( who | wc -l ) sessions en cours
```

- ___ 19. Chaque compte utilisateur du système est représenté par une ligne dans le fichier **/etc/passwd**. En utilisant vos connaissances sur la substitution de commande, affichez le message suivant : **Il y a # utilisateurs de créés sur le système**, où # représente le nombre d'entrées dans **/etc/passwd**.

```
»$ echo Il y a `cat /etc/passwd | wc -l` utilisateurs de créés  
sur le système
```

- ou -

```
»$ echo Il y a $( cat /etc/passwd | wc -l ) utilisateurs de créés  
sur le système
```

.d Environnement d'un process, export de variable

__ 20. Affichez toutes les variables d'environnement du process courant.

```
»$ set
```

__ 21. Créez une variable **x** initiée à **10**. Vérifiez sa valeur et affichez de nouveau vos variables d'environnement.

```
»$ x=10
```

```
»$ echo $x
```

```
»$ set
```

__ 22. Créez un sous shell avec **ksh** (ou **bash**). Vérifiez si la variable **x** est connue dans le sous shell courant. Quelle est la valeur de **x** ? *vide* Affichez la liste des variables du sous shell. Voyez vous la variable **x** ? *non*

```
»$ ksh -ou- $ bash
```

```
»$ echo $x
```

```
»$ set
```

__ 23. Retournez dans votre process parent. Modifiez la description de la variable **x** de façon à ce quelle soit héritée par les process enfants. Vérifiez que la modification est prise en compte en créant un sous shell et dans ce nouveau process affichez de nouveau la valeur de **x**.

```
»$ exit
```

```
»$ export x=10
```

```
»$ ksh
```

```
»$ echo $x
```

__ 24. Toujours dans le sous shell, changez la valeur de **x** à 200. Vérifiez que la valeur est changée.

»\$ **x=200**

»\$ **echo \$x**

- __ 25. Retournez dans le process parent. Dans cet environnement vérifiez la valeur de **x**. Est-ce que la modification faite dans le sous shell a été exportée en amont vers le parent ? *non*

»\$ **exit**

»\$ **echo \$x**

- __ 26. Créez un script shell nommé **sc1**. Son contenu :

pwd ; cd / ; pwd

»\$ **vi sc1** <Entrée> **i pwd ; cd / ; pwd**

»\$ appuyez sur <Echap>, suivi de **:wq** pour sauver et quitter.

- __ 27. Modifiez les droits d'accès de **sc1** pour qu'il soit exécutable et lancez ce programme. Une fois le script terminé, dans quel répertoire êtes vous ? *Toujours dans le même répertoire. Pourquoi ? Le changement n'a eu lieu que le temps d'exécution de sc1.*

»\$ **chmod 700 sc1**

»\$ **sc1**

»\$ **pwd**

- __ 28. Créez un autre script shell nommé **sc2**. Son contenu :

var1=bonjour ; var2=\$LOGNAME ; export var1 var2

»\$ **vi sc2** **i var1=bonjour ; var2=\$LOGNAME ; export var1 var2**

»\$ appuyez sur <Echap>, suivi de **:wq** pour sauver et quitter.

- __ 29. Modifiez les droits de **sc2** pour qu'il soit exécutable puis lancez ce nouveau programme. Quand il est terminé, examinez le contenu des variables **var1** et **var2**. Quelles sont leurs valeurs ? *vide Pourquoi ? var1 et var2 n'existent que le temps d'exécution du sous-shell.*

```
»$ chmod 700 sc2
```

```
»$ sc2
```

```
»$ echo $var1 $var2
```

- ___ 30. Relancez de nouveau le script **sc2**, mais cette fois forcez son exécution dans le process shell courant. Lorsqu'il a fini de s'exécuter, vérifiez les valeurs de **var1** et de **var2**. Quelles sont leurs valeurs ? *var1=bonjour, var2=stagexx*
Pourquoi ? *Le code a été interprété par le shell courant*

```
»$ . sc2
```

```
»$ echo $var1 $var2
```

.e Apostrophes, guillemets et banalisation explicite (antislash)

- ___ 31. Utilisez trois façons pour affichez le caractère astérisque ***** en utilisant la commande **echo**.

```
»$ echo '*'
```

```
»$ echo "***"
```

```
»$ echo \*
```

- ___ 32. Assurez-vous d'être dans votre répertoire d'accueil. Créez un dossier nommé **quote**.

```
»$ cd
```

```
»$ pwd
```

```
»$ mkdir quote
```

- ___ 33. Déplacez-vous dans le dossier **quote**. Créez un fichier vide nommé **fic**. Créez une variable nommée **n** qui contient **bonjour**. Vérifiez ce que vous avez fait en listant le contenu du dossier **quote** puis affichez le contenu de la variable **n**.

```
»$ cd quote
```

```
»$ touch fic
```

```
»$ n=bonjour
```

```
»$ ls
```


»\$ echo \$n

- 34. **Optionnel** - Depuis le dossier **quote**, lancez les cinq commandes suivantes. Notez les résultats. Attention à différencier les apostrophes (') des anti-apostrophes(`)<Alt-Gr> 7.

i.\$ echo '* \$n `ls` \$(ls) '

ii.\$ echo "* \$n `ls` \$(ls) "

iii.\$ echo * \\$n \'ls\' \' \\$\ (ls\)

iv.\$ echo * \$n `ls` \$(ls)

v.\$ echo * \$n ls

Fin de l'exercice

.9 TP 9 – Personnaliser son environnement

.a Personnalisation de .kshrc ou .bashrc

___ 1. Pour personnaliser votre environnement à chaque connexion, autrement dit définir votre profil d'utilisateur, vous devez mettre à jour le fichier qui est lu à chaque connexion. Assurez-vous d'être dans votre répertoire d'accueil et selon le shell qui vous est attribué, modifiez soit le fichier **.kshrc**, soit **.bashrc** en ajoutant les fonctions suivantes :

___ a. modifiez l'invite de commande pour qu'à la place du \$ par défaut , ce soit le nom du répertoire courant concaténé à '>_' qui apparaisse.

___ b. affichez un message de bienvenue personnalisé avec votre nom de login et la date et l'heure système.

___ c. déclarez un **alias dir** qui lance la commande **ls -l**,

___ d. activez l'édition de l'historique des commandes grâce aux commandes de l'éditeur **vi**.

```
»$ cd
```

```
»$ pwd
```

```
    - version si ksh ( Korn Shell ) -
```

```
»$ vi .kshrc
```

```
PS1="$PWD >_ " ( pas d'espace entre le '=' et le '>' )
```

```
echo Bonjour $LOGNAME , nous sommes le $(date)
```

```
alias dir='ls -l'
```

```
set -o vi
```

```
<Echap>:wq
```

```
    - version si bash ( Bourne Again Shell ) -
```

```
»$ vi .bashrc
```

```
PS1="\w>_ " ( pour découvrir tous les codes utilisables, voir l'aide via le man sur bash,
                rubrique PROMPTING )
```

```
echo Bonjour $LOGNAME , nous sommes le $(date)
```

```
alias dir='ls -l'
```

```
set -o vi
```

<Echap>:wq

___ 2. Testez votre personnalisation en ré-exécutant votre fichier de profile. Pour ce faire deux possibilités : soit vous vous déconnectez puis reconnectez, soit vous lancez l'interprétation du fichier de profile avec la notation du point. Une fois cela fait, quelques questions :

___ a. est-ce que le message de bienvenue est affiché ?

___ b. est-ce que le nom du répertoire courant est affiché dans l'invite de commande ?

___ c. déplacez-vous dans le répertoire **/etc**. Est-ce que l'invite est mise à jour ? (remarque : ceci ne serait pas fonctionnel avec le shell Bourne historique),

___ d. utilisez **dir** pour affichez la liste détaillée du répertoire courant,

___ e. pouvez-vous relancez **dir** en rappelant la dernière commande de l'historique ?

Si vous répondez non à au moins une des questions, alors rééditez votre fichier **.kshrc** ou **.bashrc** et corrigez le problème ...

»\$ logout

»Login: stagexx

»stagexx Password: stagexx

- ou -

»\$. .kshrc (version ksh) ----- »\$. .bashrc (version bash)

.b Gestion des alias

___ 3. Affichez la liste des **alias** de votre environnement

»/home/stagexx=> alias

___ 4. Supprimez **dir** de vos alias

»/home/stagexx=> unalias dir

___ 5. Tentez de relancez **dir**.

```
»/home/stagexx=> dir
```

___ 6. L'alias **dir** est toujours défini dans votre fichier de connexion, mais n'est plus défini dans votre environnement. Pour qu'il soit de nouveau déclaré, soit vous vous déconnectez / reconnectez, soit vous relancez votre fichier dans votre shell courant par la syntaxe du point.

Fin de l'exercice.

2775cbe5ce
Global Knowledge

.10 TP 10 – Utilitaires

.a La commande find

__ 1. Recherchez tous les fichiers du répertoire **/tmp** et affichez leur chemin d'accès.

```
»$ find /tmp
```

__ A partir de votre répertoire d'accueil, recherchez tous les fichiers dont le nom commence par la lettre s, et pour chacun d'entre eux lancez automatiquement la commande **ls -l**.

```
»$ find . -name 's*' -exec ls -l {} \;
```

OR

```
»$ find . -name "s*" -ls
```

__ 3. Relancez la commande précédente, mais cette fois avec une demande de confirmation avant de lancer la commande **ls -l** sur chaque nom de fichier trouvé.

```
»$ find . -name 's*' -ok ls -l {} \;
```

__ 4. Toujours depuis votre répertoire d'accueil, recherchez à partir des répertoires **/var** et **/tmp**, les fichiers qui vous appartiennent. Puis modifiez la ligne de commande pour compter ce nombre de fichiers. Il peut y avoir des répertoires pour lesquels vous n'avez pas le droit de passage. Renvoyez les messages d'erreur vers un fichier nommé **errfic**.

```
»$ find /var /tmp -user `logname` 2> errfic | wc -l
```

__ 5. Affichez le fichier **errfic** indiqué de commande précédente pour vérifier s'il y a eu des messages d'erreur.

```
»$ less errfic - ou bien - $ more errfic
```

- ___ 6. Pour mettre en évidence que **find**, à partir d'un répertoire de départ, fonctionne de façon récursive dans tous les répertoires et sous-répertoires sous-jacents, effectuez les opérations suivantes:
- assurez-vous d'être dans votre répertoire d'accueil,
 - créez un sous répertoire nommé **niveau1**,
 - dans ce sous répertoire **niveau1**, créez un fichier de taille vide nommé **nomfic1**,
 - déplacez-vous dans le sous répertoire **niveau1**,
 - dans **niveau1** créez un sous répertoire **niveau2**,
 - dans ce sous répertoire **niveau2**, créez un fichier de taille vide nommé **nomfic2**,
 - retournez dans votre répertoire d'accueil,
 - depuis votre répertoire d'accueil, lancez la commande qui affiche tous les fichiers dont le nom commencent par la lettre **n**. Notez les noms affichés.
 - depuis votre répertoire d'accueil relancez la commande qui affiche les fichiers dont le nom commence par **n**, mais cette fois en ne recherchant que les fichiers ordinaires. Notez aussi cette nouvelle liste de noms et comparez avec la précédente.

```

»$ cd
»$ mkdir niveau1
»$ touch niveau1/nomfic1
»$ cd niveau1
»$ mkdir niveau2
»$ touch niveau2/nomfic2
»$ cd
»$ ls n*
»$ find . -name 'n*' -type f

```

.b La commande **grep**

- ___ 1. Dans le fichier **/etc/passwd**, recherchez toutes les lignes des utilisateurs dont le nom commence par **stage**,

```
»$ grep "^stage" /etc/passwd
```

- ___ 2. Dans le fichier **/etc/passwd** recherchez toutes les lignes qui commencent par la lettre **s**,

```
»$ grep '^s' /etc/passwd
```

- ___ 3. Dans le fichier **/etc/passwd**, recherchez toutes les lignes qui contiennent un chiffre 0 – 9,

```
»$ grep '[0-9]' /etc/passwd
```

- ___ 4. Relancez la commande précédente, mais cette fois affichez uniquement le nombre de lignes qui contiennent un chiffre,

```
»$ grep -c '[0-9]' /etc/passwd
```

- ___ 5. Utilisez les commandes **ps** et **grep** pour afficher la liste des process associés à d'autres utilisateurs que vous mêmes.

```
»$ ps -ef | grep -v stagexx | more
```

où stagexx est votre nom de connexion.

.c La commande **sort**

- ___ 6. Affichez le contenu du fichier **/etc/passwd** trié par ordre alphabétique. Puis en ordre alphabétique inverse,

```
»$ sort /etc/passwd
```

```
»$ sort -r /etc/passwd
```

.d Les commandes **head** et **tail**

- ___ 7. Affichez les dix premières lignes de **/etc/passwd**,

```
»$ head /etc/passwd
```

__ 8. Affichez les cinq premières lignes de **/etc/passwd**,

```
»$ head -5 /etc/passwd
```

__ 9. Affichez les dix dernières lignes de **/etc/passwd**

```
»$ tail /etc/passwd
```

.e Les commandes **diff** et **cmp**

__ 10. Créez un fichier **liste1** qui contient les prénoms de personnes que vous connaissez, un prénom par ligne, au moins quatre prénoms différents. Copiez ce fichier **liste1** en un fichier **liste2**. Editez le fichier **liste2** et faites les modifications suivantes :

- . changez l'orthographe d'un des prénoms,
- . supprimez un des autres prénoms,
- . ajoutez un nouveau prénom.

```
»$ vi list1 ( ajoutez au moins quatre prénoms, un par ligne)
```

```
»$ cp list1 list2
```

```
»$ vi list2 ( modifiez le contenu comme indiqué ci-dessus )
```

__ 11. Utilisez la commande **diff** pour comparer les contenus de **liste1** et **liste2**,

```
»$ diff list1 list2
```

__ 12. Utilisez la commande **cmp** et comparez de nouveau le contenu des fichiers **liste1** et **liste2**. Puis relancez la commande de façon à obtenir une liste longue des différences.

```
»$ cmp list1 list2
```

```
»$ cmp -l list1 list2
```

.f La commande **tar**

- __ 13. Depuis votre répertoire d'accueil utilisez la commande **tar** pour archiver tous les fichiers de ce répertoire. Créez un fichier d'archive **/tmp/archivexx.tar** pour sauvegarder les fichiers en mode relatif.

```
»$ tar -cvf /tmp/archivexx.tar *
```

- __ 14. Vérifiez le contenu du fichier d'archive.

```
»$ tar -tvf /tmp/archivexx.tar
```

- __ 15. Créez un répertoire **/tmp/restaurexx** (où xx est votre numéro de connexion). Déplacez-vous dans ce répertoire pour y restaurer les fichiers de votre archive.

```
»$ mkdir /tmp/restaurexx
```

```
»$ cd /tmp/restaurexx
```

```
»$ tar -xvf ../archivexx.tar
```

2775cbe5ce
Global Knowledge

.11 TP 11 – Les bases de la programmation shell

.a Ecrire une procédure script shell

- __ 1. Créez un script shell nommé **parametres** qui affiche les cinq lignes suivantes, en utilisant les variables prédéfinies du shell pour remplacer les "_____" par les paramètres transmis . Exécutez le script avec les arguments 10 100 1000.

Le nom de la procédure est _____.
 Le premier paramètre reçu est le nombre _____.
 Le deuxième paramètre le nombre _____.
 Le troisième paramètre le nombre _____.
 En tout il y a _____ paramètres de transmis.

»\$ **vi parameters**

```
echo Le nom de la procédure est $0.
echo Le premier paramètre reçu est le nombre $1.
echo Le deuxième paramètre le nombre $2.
echo Le troisième paramètre le nombre $3.
echo En tout il y a $# paramètres de transmis.
```

»\$ **chmod +x parameters**

»\$ **parameters 10 100 1000**

- __b 2. Créez une procédure **veriffic**, qui affiche le contenu du fichier **parametres** à condition qu'il existe. Utilisez un opérateur d'exécution conditionnelle.

»\$ **vi veriffic**

```
ls parameters && cat parameters
```

»\$ **chmod +x veriffic**

»\$ **veriffic**

- ___ 3. Modifiez la procédure **veriffic** pour remplacer le nom du fichier **parametres** par un nom de fichier qui n'existe pas **nonfic** (soyez certains qu'il n'y a pas de fichiers nommés nonfic dans votre répertoire courant). Toujours en utilisant un opérateur d'exécution conditionnelle, si la commande **ls** échoue, alors affichez un texte d'erreur **Le fichier n'a pas été trouvé**. Lancez la procédure.

```
»$ vi veriffic
ls nonfic && cat nonfic || echo "Le fichier n'a pas été trouvé"
»$ veriffic
```

- ___ 4. Modifiez la procédure **veriffic** pour que le message d'erreur affiché par la commande **ls** n'apparaisse pas à l'écran. Lancez la procédure.

```
»$ vi veriffic
ls nonfic 2> /dev/null && cat nonfic || echo "Le fichier n'a pas
été trouvé"
»$ veriffic
```

- ___ 5. Modifiez la procédure **veriffic** pour pouvoir lui transmettre un argument sur la ligne de commande. Cet argument est utilisé, dans le code de la procédure, comme paramètre des commandes **ls** et **cat**. Lancez deux fois la procédure script shell, une fois avec **parametres**, la seconde fois avec **nonfic**.

```
»$ vi veriffic
ls $1 2> /dev/null && cat $1 || echo "Le fichier n'a pas été
trouvé"
»$ veriffic parametres
»$ veriffic nonfic
```

.b Utilisation de **for**, **test** et **if**

- ___ 6. En utilisant la boucle **for**, modifiez la procédure **veriffic** pour qu'elle puisse recevoir plusieurs noms de fichier en paramètre au lieu d'un seul. Si les fichiers existent alors affichez leur contenu. Si les fichiers n'existent pas, alors affichez un message d'erreur comme quoi le fichier n'a pas été trouvé. Repérez quelques noms de fichier présents dans votre répertoire courant. Lancez la procédure en indiquant des noms de fichiers existants et de fichiers inexistantes.

```
»$ vi veriffic
```

```

for x in $*
do
    ls $x 2> /dev/null && cat $x || echo $x pas trouvé
done
»$ ls
»$ veriffic fic1 fic2 fic3

```

(où *fic1 fic2 fic3* sont remplacés par des noms de fichier, existants ou pas, dans votre répertoire courant).

- 7. Modifiez votre procédure **veriffic** pour utiliser l'opérateur **if** et la commande **test**, à la place des opérateurs d'exécution conditionnelle, pour vérifier que le nom de fichier est celui d'un fichier existant ou pas. Lancez la procédure comme vous l'avez fait à l'étape précédente.

Astuce : les codes retours sont utilisés dans cette procédure.

```

»$ vi veriffic
for x in $*
do
    ls $x 2> /dev/null
    if [[ $? -eq 0 ]]
    then cat $x
    else echo $x pas trouvé
    fi
done
»$ veriffic fic1 fic2 fic3

```

.c Utilisation de **while** et de **expr**

- 8. Créez une procédure nommée **manger**, qui utilise une boucle **while** infinie, pour afficher **A table !** toutes les deux secondes. Lancez ce script shell. Quand vous en avez assez, interrompez la boucle.

```

»$ vi manger
while true
do
    echo A table !

```

```
sleep 2
done
»$ chmod +x manger
»$ manger
<Ctrl-c>
```

- 9. Directement sur la ligne de commande, affichez le résultat de la multiplication de 5 fois 6.

```
»$ expr 5 \* 6
```

- 10. Maintenant, toujours en utilisant **expr**, créez une procédure nommée **math** qui accepte deux nombres comme arguments sur la ligne de commande et qui affiche leur multiplication. Lancez la procédure pour calculer 5 fois 6. Testez avec d'autres nombres.

```
»$ vi math
expr $1 \* $2
»$ chmod +x math
»$ math 5 6
```

Fin de l'exercice