



Université Cadi Ayyad
Faculté des Sciences Semlalia
Département d'Informatique
Marrakech



Système d'exploitation

UNIX

Utilisation - Développement

Azzeddine LAZREK



Table des matières

Introduction	3
1. Présentation d'un système d'exploitation	4
1.1. Rôles d'un système d'exploitation	
1.2. Fonctions d'un système d'exploitation	
1.3. Classification des systèmes d'exploitation	
1.4. Historique des systèmes d'exploitation	
1.5. Conception d'un système d'exploitation	
2. Présentation d'UNIX	16
2.1. Historique	
2.2. Normalisation	
2.3. Architecture	
2.4. Caractéristiques	
2.5. Linux	
2.6. Systèmes d'exploitation mobiles	
Partie I : Utilisation	
3. Session de travail	22
3.1. Compte d'un utilisateur	
3.2. Connexion	
3.3. Commande	
3.4. Déconnexion	
4. Système de gestion de fichiers	25
4.1. Structure arborescente	
4.2. Gestion des répertoires	
4.3. Gestion des fichiers	
5. Gestion de processus	44
5.1. Processeur	
5.2. Interpréteur de commande	
5.3. Manipulation de processus	
5.4. Historique	
5.5. Alias de commandes	
5.6. Redirection des entrées-sorties	
5.7. Tube entre les commandes	
6. Commandes étendues	64
6.1. Impression de fichiers	
6.2. Aide électronique	
6.3. État des utilisateurs	
6.4. Gestion de sessions de travail	
6.5. Lien de ficheirs	
6.6. Divers	

7. Communication électronique	72
7.1. Courrier	
7.2. Dialogue	
7.3. Agenda	
8. Sécurité de fichiers	77
8.1. Mot de passe	
8.2. Modes de protection	
8.3. Cryptage	
 Partie II : Développement	
9. Éditeur de texte	83
9.1. Éditeur vi	
9.2. Éditeur ed	
10. Langages de commandes	91
10.1. Interpréteur de commandes	
10.2. Procédure de commandes	
10.3. Vocabulaire	
10.4. Utilitaires	
10.5. Fonction	
11. C Shell	105
11.1. Variables	
11.2. Structures de contrôle	
11.3. Utilitaires	
11.4. Cas traités	
12. Bourne Shell	127
12.1. Variables	
12.2. Structures de contrôle	
12.3. Utilitaires	
12.4. Cas traités	
Bibliographie	138
Annexes	
1. Liste des symboles utilisés	139
2. Liste de fichiers particuliers	140
3. Liste de répertoires particuliers	143
4. Exercices	144
5. Travaux Pratiques	147
6. Projets	148

Introduction

Ce document est un support de cours spécifique au module **Système d'exploitation - UNIX**.

Les objectifs de ce module sont d'une part la maîtrise de l'utilisation d'un système d'exploitation (aspects utilisateur) et d'une autre part la connaissance des concepts de base et le fonctionnement des systèmes d'exploitation lors de la programmation (aspects développement), à travers UNIX. La conception et la réalisation d'un système d'exploitation sont l'objet d'un autre module. L'installation de Linux est présentée en travaux pratiques. Cependant, la maintenance d'un système d'exploitation peut faire l'objet d'un autre module.

Le système d'exploitation occupe une partie vitale d'un système informatique. Le système d'exploitation présente une interface simplifiée, efficace et agréable entre l'utilisateur et les ressources de la machine. La majorité des fonctionnalités du système d'exploitation offerte à la plupart des utilisateurs est transparente et mystérieuse. Malgré l'objectif principal d'un système d'exploitation, à savoir la maintenance au maximum de cette transparence, il est très utile, si ce n'est pas nécessaire pour les informaticiens, de comprendre les concepts de base et les algorithmes associés qui entrent en jeu lors de son fonctionnement.

Le déroulement est comme suit : une présentation des systèmes d'exploitation et leur fonctionnement. Ensuite, la première partie dédiée à l'utilisation, à travers la maîtrise des commandes de base d'UNIX. Et enfin, la deuxième partie consacrée au développement, à travers la manipulation des interpréteurs et des langages de commandes C Shell et Bourne Shell.

Le premier chapitre donne une présentation générale d'un système d'exploitation. Le deuxième chapitre donne une présentation générale du système d'exploitation UNIX.

Le troisième chapitre présente la notion de session de travail, de la connexion à la déconnexion avec la syntaxe générale d'une commande. Le quatrième chapitre donne les différentes commandes de la gestion de répertoires et de fichiers. Le cinquième chapitre présente les différents modes d'exécution de processus, avant plan, arrière plan, en différé, etc. avec les commandes de transition. Le sixième chapitre est consacré à un certain nombre d'utilitaires offerts par le système d'exploitation, comme par exemple : l'impression de fichier, l'aide électronique, etc. Le septième chapitre présente les différentes commandes de communication électronique, comme par exemple : le courrier et le dialogue électronique, etc. Le huitième chapitre est consacré à la sécurité de fichiers où sont étudiés les commandes de protection de fichiers et de répertoires contre la lecture, l'écriture et l'exécution ainsi que la protection du compte d'un utilisateur.

Le neuvième chapitre présente deux éditeurs de texte vi et ed. vi, comme éditeur de texte pleine page, pour l'édition de fichiers. ed, comme éditeur de texte ligne, utilisé dans les procédures de commandes. Le dixième chapitre étudie la notion de langage de commande. Le onzième chapitre présente l'interpréteur de commandes C Shell. Le douzième chapitre présente l'interpréteur de commandes Bourne Shell.

Le système d'exploitation étudié dans ce document est **ULTRIX-32** version 4.4 de **DEC** sous le système **VAX 8350**. Certaines particularités de **Solaris** version 2.3 (**SunOS 5.3**) de **Sun** sous le système **Sparc 20** sont mentionnées. Certaines particularités de **Linux** sont aussi présentées.

Un certain nombre de cas et de questions sont traités sans chercher ni l'efficacité, ni l'optimisation, ni le parfait.

1. Présentation d'un système d'exploitation

Un **système informatique** est un ensemble de **matériels (Hardware)** et de **logiciels (Software)** qui permet le traitement automatique de l'**information (Data)**.

Les principales fonctions d'un système informatique sont :

- la gestion des informations : création, stockage, mise à jour, consultation, protection, etc. ;
- la gestion des programmes : conception, mise au point, exploitation, maintenance, etc.

Le matériel peut être constitué de :

- les dispositifs physiques : contenant des circuits, des fils, des tubes, ... ;
- le logiciel microprogramme : résidant dans la mémoire morte (ROM Read Only Memory) qui est un interpréteur des instructions en langage machine ;
- le langage machine : constitué d'un ensemble d'instructions, en langage binaire, supportées par la machine.

Une classification des logiciels peut être la suivante :

- les **logiciels de base** :
 - les **utilitaires** : ce sont les outils indispensables pour l'écriture et la mise au point des programmes d'une application (Interpréteur de commandes, Compilateur, Interpréteur, Assembleur, Éditeur de liens, Chargeur, Éditeur de texte, Traitement de texte, Tableur, ...). Ils fonctionnent en **mode utilisateur** ;
 - les **systèmes d'exploitation** : ensemble des programmes de base d'une machine permettant d'utiliser tous les services disponibles (la gestion des travaux, les opérations d'entrées-sorties sur les périphériques, l'affectation des ressources aux différents processus, l'accès aux bibliothèques de programmes et aux fichiers, la comptabilité des travaux, ...). Ils fonctionnent en **mode noyau** ou **superviseur** (c'est-à-dire avec permission absolue mais en protection) ;
- les **logiciels d'applications**, progiciel, logiciel sur mesure : ce sont des programmes de résolutions de problèmes particuliers et spécifiques (Comptabilité, Paie, Gestion de stock, Bibliothèque numérique, ...).

Un **système d'exploitation (Operating System)** est un ensemble de programmes de grandes tailles et de complexité élevée. En effet, sa taille et sa complexité dépendent des fonctionnalités qu'il offre, des applications qu'il supporte et des caractéristiques des ressources qu'il gère. C'est un véritable système de composantes cohérentes et interconnectés entre eux. Il permet de faire fonctionner les éléments internes de l'ordinateur et de commander les éléments externes. C'est aussi une interface entre l'ordinateur et l'utilisateur.

Exemples:

CP/M, PCOS, UCSD P-System, OS/2, PROLOGUE, MS-DOS, MULTICS 1965, VM 1970, UNIX 1971, VMS 1977, ...

Actuellement, toute machine est dotée d'un système d'exploitation. Il est primordial pour son exploitation. C'est le premier programme qui se déroule lors de la mise en route de l'ordinateur. Le système d'exploitation se charge partiellement en mémoire centrale. Une partie reste permanente en mémoire. Une autre partie se charge et se décharge suivant les services demandés.

1.1. Rôles d'un système d'exploitation

Les **rôles** principaux d'un système d'exploitation d'une machine sont :

- la supervision des **ressources** de la machine par le superviseur (**Supervisor**). Entre autre, l'exécution des éléments exécutables des applications et la gestion des activités des utilisateurs par le système exécutif (**Executive system**) ;
- la présentation d'une **interface** adaptée aux besoins des utilisateurs. En particulier, il offre à l'utilisateur une **machine virtuelle** plus performante et praticable que la machine physique.

Les **ressources** d'une machine sont l'ensemble des matériels (mémoire, processeur(s), unités d'entrées-sorties, périphériques), des informations (données) et des programmes (logiciels/applications) disponibles dans l'environnement de la machine.

Le système d'exploitation offre des services à l'utilisateur pour lui faciliter l'exécution de ses programmes. Ces services sont similaires à ceux offerts par les compilateurs et les assembleurs pour lui éviter de programmer en langage binaire.

Les services/caractéristiques principaux d'un système d'exploitation sont :

- simplifier les opérations d'entrées-sorties : soulager l'utilisateur de la maîtrise du fonctionnement des opérations de lecture/écriture des informations à travers chaque périphérique (une commande unique) ;
- contrôler les erreurs : aider les utilisateurs à détecter leurs éventuelles erreurs dans les programmes et les commandes, et à les corriger et éviter les effets de bord ;
- gérer les fichiers : offrir un système de gestion de fichiers efficace (un nom au fichier au lieu de l'adresse des secteurs en mémoire) pour les utilisateurs tout en le protégeant contre les erreurs et les malversations ;
- permettre la multi-tâches : offrir la possibilité d'exécuter plusieurs programmes en parallèle, avec la possibilité de communication entre eux d'une façon sécurisante ;
- permettre le multi-utilisateurs : offrir la possibilité d'utiliser simultanément l'ordinateur, et ses ressources, par plusieurs utilisateurs, avec la possibilité de communication entre eux d'une façon sécurisante (protection totale) et performante (temps de réponse acceptable) ;
- contrôler l'activité : améliorer les performances du système informatique, soit optimiser/rentabiliser toutes les ressources de l'ordinateur ;
- gérer les ressources : attribuer, restituer, collaborer, coordonner, partager, sécuriser, ... les ressources de la machine entre les processus de(s) l'utilisateur(s) et du système.

Remarque :

Certaines caractéristiques/fonctionnalités ne sont pas communes à tous les systèmes d'exploitation.

1.2. Fonctions d'un système d'exploitation

Les principales **fonctions** d'un système d'exploitation d'une machine sont :

- la gestion du/des processeur(s) ;
- la gestion du/des processus ;
- la gestion de la mémoire ;
- la gestion des entrées-sorties des périphériques ;
- la gestion des fichiers des informations ;
- la gestion des utilisateurs.

1.2.1. Gestion du processeur

Les principales fonctions du noyau du système d'exploitation sont :

- l'attribution du processeur par le moniteur ;
- la synchronisation et le parallélisme ;
- la gestion des tâches ;
- la gestion des déroutements (**Traps**) ;
- la gestion des interruptions.

Le fonctionnement simplifié d'un processeur (**processor**) se décompose en les étapes séquentielles suivantes :

- la lecture en mémoire centrale de l'instruction désignée par le compteur ordinal (**Program Counter**) ;
- l'analyse de l'instruction ;
- la récupération des opérandes ;
- l'exécution de la fonction ;
- le transfert du résultat ;
- la modification du compteur ordinal ;
- le retour à la première l'étape.

1.2.2. Gestion de processus

Un **processus (process)** est un programme en cours d'exécution. Un processus est une entité dynamique représentant l'exécution d'un programme (ensemble d'instructions sur des données).

Un processus est réalisé par une suite de **tâches (tasks)**. Une tâche est définie par son contexte qui est principalement formé par :

- l'état de la tâche ;
- le compteur ordinal ;
- les registres ;
- le niveau de priorité.

La description d'un processus est principalement formée par :

- l'état du processus ;
- l'identité du programme ;
- les droits ;
- les ressources requises ;
- le code ;
- les données.

Les fonctions de la gestion de processus sont :

- la création du processus ;
- le changement d'état du processus ;
- la communication entre les processus ;
- la destruction du processus.

Un processus naît à sa création, vit durant son exécution et meurt à sa terminaison.

Les principaux états d'un processus sont :

- **actif** : en exécution (en possession du processeur) ;
- **prêt** : en attente du processeur ;
- **en attente** : en attente de ressource ;
- **interrompu** : interrompu par l'utilisateur ;
- etc.

Il y a deux niveaux de processus :

- le niveau des processus, **travaux (Job-Scheduling)** ;
- le niveau des **tâches (Task-Scheduling)**.

Il y a deux modes d'exécution :

- le **mode maître**, système ;
- le **mode esclave**, utilisateur.

Une commande est de deux types :

- un calcul : besoin du processeur ;
- une entrée-sortie : besoin du superviseur d'entrées-sorties.

Les problèmes suivants se posent continuellement :

- l'inter-blocage (**Deadlock**) : un processus a besoin d'une ressource détenue par un autre en attente ;
- la boucle infinie : comme par exemple : 100 GOTO 100 ;
- la synchronisation : des processus dépendants qui communiquent entre eux ;
- la sécurité : protection entre les processus et contre les processus malfaisants.

Pour exécuter un programme, il faut le charger en mémoire centrale. Le programme et les données se trouvent dans des segments différents de la mémoire centrale. Un segment est un ensemble d'emplacements mémoire contigus. Les segments programme ne sont pas modifiés au cours de l'exécution.

Les étapes d'un appel de procédure sont :

- la préparation des paramètres effectifs à transmettre ;
- la sauvegarde du contexte du programme appelant ;
- le chargement du contexte de la procédure appelée ;
- la préparation des résultats à transmettre au programme appelant ;
- la restauration du contexte du programme appelant.

Les opérations de chargement et de restauration des contextes se font dans le segment de pile d'exécution.

Un processus est un modèle représentant l'activité résultante de l'exécution d'un programme sur une machine.

L'état de la machine est déterminé par l'état du processus (contenu des registres adressables et internes, ...) et de l'état de la mémoire (contenus des emplacements, ...).

Une **action**, exécution d'une instruction, modifie l'état de la machine. Une instruction est indivisible.

Un **événement** est un état daté de la machine. Le début et la fin d'une action sont des événements.

L'exécution d'un programme se traduit par une suite d'actions, telle que le début d'une action se fait après la fin de l'action précédente. La fin d'une action et le début de l'action suivante sont deux événements différents, bien que les états correspondants soient identiques.

Le contexte d'un processus est l'ensemble des informations cumulées ou modifiées par les actions du processus.

Il y~~o~~trois types d'exécution :

- l'exécution séquentielle : les processus sont exécutés entièrement séquentiellement ;
- l'exécution pseudo-parallèle : les processus sont exécutés partiellement alternativement ;
- l'exécution parallèle : les processus sont exécutés entièrement simultanément.

Deux processus sont indépendants si leurs contextes sont disjoints. Ils n'ont pas d'interactions mutuelles. Deux processus sont en exclusion mutuelle, en compétition, s'ils demandent l'usage d'une même ressource critique en même temps. Une ressource est critique, si elle ne peut être utilisée que par un seul processus à la fois.

Deux processus sont en coopérations, s'ils participent ensemble à la réalisation d'un travail commun.

1.2.3. Gestion de la mémoire

Les principales fonctions de la gestion de la mémoire - centrale - sont :

- la désignation, la conservation et la protection de l'information ;
- l'allocation et la libération de la mémoire ;
- la gestion de l'espace libre.

Les deux problèmes suivants se posent continuellement :

- le manque d'espace libre : qui peut se résoudre par des transferts, ou l'extension, ou l'étendu à la mémoire secondaire ;
- le morcellement de la mémoire, fragmentation : qui peut se résoudre par le tassage, la défragmentation.

La désignation de l'information est l'allocation des objets définis par un langage de programmation dans une mémoire physique directement adressable.

La notion de mémoire virtuelle fait une abstraction sur le transfert de l'information entre la mémoire centrale et la mémoire secondaire en donnant un espace d'adressage plus important et uniforme.

Tout programme à exécuter doit être en mémoire centrale. Les problèmes suivants se posent alors :

- l'allocation et la restitution de la mémoire aux programmes ;
- le mécanisme d'adressage : transformation d'une adresse virtuelle à une adresse réelle ;
- la stratégie d'allocation de la mémoire : transfert de l'information entre la mémoire virtuelle et la mémoire centrale.

Pour une machine 32 bits, le champ d'adressage d'une instruction est de longueur de 32 bits. Ce qui implique un espace d'adressage composé de plusieurs Giga-octets. ($2^{32} = 4 \text{ Go}$).

La mémoire secondaire est décomposée en zones de tailles variables ou en cases de tailles fixes.

La mémoire virtuelle est décomposée en segments de tailles variables ou en pages de tailles fixes.

La mémoire d'un processus est un vecteur d'emplacements décomposé en zones de tailles variables.

UNIX est un système **paginaire**, tout processus est découpé en un ensemble de pages d'une taille déterminée. Par exemple, une page est de taille **4 Ko**. Le système charge uniquement les pages nécessaires à l'exécution (**swap-in**). Il les décharge une fois utilisées (**swap-out**) puis charge les pages suivantes et ainsi de suite. C'est la gestion de la mémoire virtuelle.

La gestion de la mémoire virtuelle est implantée dans un fichier à accès spéciale, la partition **swap**, en mémoire secondaire.

1.2.4. Gestion des entrées-sorties des périphériques

Les fonctions de la gestion des entrées-sorties sont :

- le fonctionnement du superviseur d'entrées-sorties ;
- la gestion du contrôleur des périphériques.

Le déroulement d'une entrée-sortie se décompose :

- l'exécution de l'instruction de commencement d'entrées-sorties par le processeur ;
- l'exécution de la commande de transfert de l'information par le contrôleur de périphérique après réception des paramètres de l'instruction ;
- l'exécution de l'ordre de transfert de l'information du support d'enregistrement par le périphérique.

1.2.5. Gestion des fichiers des informations

Les fonctions de la gestion des informations sont :

- la gestion du système de gestion de fichiers ;
- la gestion des entrées-sorties logiques.

En général, les systèmes de fichiers sont structurés sous une forme d'arborescence.

Les rôles d'un système de gestion de fichiers sont :

- la gestion de l'espace physique pour le stockage de l'information ;
- la gestion de l'accès physique aux fichiers ;
- la gestion des transferts logiques d'informations ;
- la gestion des protections.

L'espace physique est découpé en blocs de tailles fixes. Les problèmes suivants se posent alors continuellement :

- la gestion de l'espace libre ;
- l'allocation des blocs au niveau physique ;
- l'allocation des fichiers au niveau logique.

1.2.6. Gestion des utilisateurs

La machine physique est partagée entre plusieurs utilisateurs. Chaque utilisateur dispose d'une machine virtuelle de même comportement que la machine physique.

En général, le partage se fait par **multiplexage** qui consiste à allouer la machine physique aux utilisateurs, pendant des tranches de temps successives, avec éventuellement des priviléges.

Le partage de la machine entre plusieurs utilisateurs génère les problèmes suivants :

- la dégradation continue des performances de la machine avec l'augmentation du nombre d'utilisateurs ;
- la nécessité de la protection de certaines ressources, des commandes privilégiées et de certaines zones mémoires.

1.3. Classification des systèmes d'exploitation

Il y a plusieurs façons de classifier les systèmes d'exploitation. Une classification suivant leurs fonctionnalités, par rapport à la machine associée, est la suivante :

- système individuel ;
- système en temps réel ;
- système transactionnel ;
- système en temps partagé ;
- système réseau ;
- système vectoriel.

1.3.1. Système individuel

Une composition simplifiée d'un ordinateur individuel est :

- un terminal : écran, clavier et souris ;
- une unité centrale : processeur et mémoire centrale ;
- des périphériques : imprimante et mémoire secondaire.

L'interface présentée par le système à l'utilisateur est le langage de commandes.

Exemple : Micro-ordinateur.

Auparavant, les principales propriétés d'un système individuel sont :

- mono-utilisateur ;
- mono-tâche.

Un système mono-programmé est caractérisé par l'exécution séquentielle des différents processus. À un instant donné, un seul processus réside en mémoire centrale en mobilisant toutes les ressources de la machine lors de son exécution.

La **commutation de tâches**, amélioration faite sur les systèmes mono-utilisateur, permet un traitement concurrent de plusieurs tâches d'un même utilisateur.

Exemples :

CP/M, MS-DOS

1.3.2. Système en temps réel

Une composition simplifiée d'un système en temps réel est :

- une installation industrielle : qui émet des signaux de mesures ;
- un ordinateur individuel : qui transforme les signaux de mesures en des signaux de commandes.

Les principales propriétés d'un système en temps réel sont :

- la gestion de l'information ;
- le traitement en temps réel ;
- la communication avec les organes extérieurs.

Exemple : Le détecteur de panne d'un procédé industriel commandé par ordinateur.

1.3.3. Système transactionnel

Les principales propriétés d'un système transactionnel sont :

- la gestion des bases de données ;
- la préservation des contraintes d'intégrité ;
- la gestion des transactions ;
- le temps de réponse relativement court.

Exemples :

- Système de réservation de places d'avion.
- Système de gestion des comptes bancaires.
- Système de gestion documentaire.

1.3.4. Système en temps partagé

Les processus partagent l'allocation du processeur en tranches de temps. À intervalle de temps régulier, et suivant la priorité, l'ordonnanceur (**Schedule**) suspend le processus en cours d'exécution pour affecter le processeur au processus prêt suivant.

Le processeur est en attente lors des transferts des informations. En effet :

- le processeur traite les informations à un débit élevé ;
- les périphériques transfèrent les informations à un débit faible.

Les unités d'échanges sont introduites pour assurer la synchronisation et le transfert des informations simultanément avec le traitement. Elles utilisent des tampons d'entrées-sorties pour :

- maximiser l'exploitation du processeur : un processus en attente de traitement peut utiliser le processeur libéré par un autre processus en attente d'entrée de données ;
- minimiser le temps de réponse : le processeur peut suspendre le traitement d'un long processus pour traiter un autre processus plus court.

Les principales propriétés d'un système en temps partagé sont :

- la multi-programmation : une exécution concurrente de plusieurs processus utilisateurs ;
- la multi-tâches : une exécution concurrente entre un processus utilisateur et des tâches de système.

Lors des situations suivantes :

- la présence de plusieurs programme en mémoire centrale : on a besoin d'une augmentation de la taille de la mémoire centrale ;
- le transfert d'information entre la mémoire centrale et la mémoire secondaire : on a besoin d'un système de pagination évolué ;
- la multi-programmation : on a besoin d'une rapidité d'accès à la mémoire secondaire.

Une machine en temps partagé offre en plus du service d'une machine individuelle un service commun de communication et de partage d'informations et de ressources.

1.3.5. Système réseau

Un réseau local est la connexion de plusieurs machines individuelles de grandes performances.

Les principales propriétés d'un système réseau sont :

- la communication entre les utilisateurs ;
- l'accès aux services communs ;
- le partage des ressources.

L'introduction des systèmes clients/serveurs a eu lieu en début des années 1980. Les clients s'adressent aux serveurs pour satisfaire leurs demandes de services.

Dans un système réparti, plusieurs processeurs partagent la mémoire.

1.3.6. Système vectoriel

Les principales propriétés d'un système vectoriel sont :

- le multi-processeurs : présence de plusieurs processeurs qui travaillent en parallèle ;
- la programmation parallèle : exécution de plusieurs instructions indépendantes en parallèles (Par exemple, le calcul des éléments de la matrice produit de deux matrices).

1.4. Historique des systèmes d'exploitation

Les premiers ordinateurs n'étaient pas dotés d'un système d'exploitation, car en particulier les ordinateurs ne possédaient pas de mémoire centrale. L'exécution se fait pas-à-pas avec une interaction directe de l'utilisateur.

L'étude de l'histoire des systèmes d'exploitation montre leur évolution petit à petit. Ce qui met en évidence leurs faiblesses et éclaircie les améliorations apportées au cours du temps pour les surmonter. Puisque la structure du système d'exploitation dépend de l'architecture de l'ordinateur qu'il abrite, l'histoire de l'évolution des systèmes d'exploitation est liée à celle des générations des ordinateurs.

Les générations des ordinateurs sont :

- avant ordinateur : conception sans réalisation d'une première **machine analytique mécanique** par Charles Babbage (1792-1871).
- 1^{ère} génération (1945-1955) : construction d'une première machine à calculer basée sur des **tubes à vide électroniques** par John Von Neumann et autres. Ils étaient énormes, lentes, ... L'enregistrement des programmes passe des cartes enfichables aux cartes perforées permettant la détection des erreurs au préalable.
- 2^{ème} génération (1955-1965) : construction d'une première machine à **transistors**. Ils étaient commercialisés pour la première fois, mais à un prix désorbitant. Le chargement des programmes, des données et des logiciels se fait manuellement par des opérateurs spécialisés. Ils sont basés sur le traitement par lots. La production en parallèle de deux types de machines : les gros ordinateurs de calculs scientifiques, orientés mot (comme IBM 7094) et les ordinateurs de traitement de données, orientés caractère (comme IBM 1401).
- 3^{ème} génération (1965-1980) : construction d'une première machine à **circuits intégrés**. Ils sont plus performants et moins chers (comme IBM 360). Ils sont basés sur la multi-programmation, où les calculs et les entrées-sorties se font en parallèle, et le spool (Simultaneous Peripheral Operation On Line). Ils sont dotés d'un système d'exploitation très volumineux, bourrés d'erreurs d'une façon continue dans les différentes versions.
- 4^{ème} génération (1980-1990) : construction des premières machines personnelles à circuits LSI (Large Scale Integration). Ils sont des micro-ordinateurs à bas prix. Ils sont basés sur MS-DOS et UNIX ou XENIX.
- 5^{ème} génération (1990-) : construction des premières machines avec de l'**intelligence artificielle**.

Un facteur essentiel qui entre en jeu direct dans l'évolution des systèmes d'exploitation est l'évolution technologique considérable de la performance du matériel des ordinateurs. Cependant, les utilisateurs et leurs programmes deviennent de plus en plus gourmands et exigeants. Un autre facteur aussi important est la diminution considérable du prix du matériel qui l'a accompagné. Ce qui a entraîné un profond changement de préoccupation dans la conception des systèmes d'exploitation.

En 1953 : l'apparition du premier système d'exploitation **Job By Job Processing**. Il était très long et nécessitait une intervention manuelle pour les opérations d'entrées-sorties.

En 1958 : l'apparition des moniteurs d'enchaînement permettant l'enchaînement de l'exécution d'un ensemble de programmes avec leurs données. Ils ont évolué vers les systèmes de traitement par lots (**Batch**). Les systèmes de traitement par lots sont caractérisés par le fait que l'utilisateur ne peut plus intervenir pendant le déroulement de son travail après sa soumission. Puis l'apparition de systèmes de traitement par lots à distant **RJE (Remote Job Entry)**. Ils comportent plusieurs stations d'entrées-

sorties éloignées, permettant la soumission des travaux à distance via des lignes téléphoniques. Puis l'apparition des systèmes multi-accès. Ils permettent aux utilisateurs de contrôler leurs travaux même après leurs soumissions à partir de leurs terminaux. Puis l'apparition des systèmes de traitement par lots (pour les travaux routinières non interactifs, exemple : paie, gestion de stock) et multi-accès (pour les travaux interactifs, exemple : développement de programmes ou de documents) sur une ou plusieurs machines interconnectées.

En 1960 : l'apparition du système d'exploitation de lot **PRECOCE**. Il utilise les cartes perforées.

En 1960 : l'apparition du système de contrôles des entrées-sorties, **IOCS** (Input/Output Control Services).

En 1965 : le commencement d'une étude théorique des systèmes d'exploitation.

En 1962 : l'apparition de la première mémoire paginée **ATLAS**.

Entre 1965 et 1968 : l'élaboration des notions de processus séquentiels, d'exclusion mutuelle, de synchronisation et de sémaphores.

En 1965 : l'introduction de la notion de segmentation pour la structuration et la désignation de l'information.

En 1966 : l'apparition du premier système d'exploitation en temps partagé **CTSS** et l'introduction des unités d'échanges.

En 1967 : le **THE** est un système d'exploitation utilisant les sémaphores.

En 1967 : le **MULTICS** (Multiplexed Information and Computing Service) est un système d'exploitation avec un noyau de gestion de processus.

1.5. Conception d'un système d'exploitation

Un système d'exploitation occupe une place dans la mémoire centrale (**over-head place**) et utilise du temps de l'unité centrale (**over-head time**). L'attention est focalisée sur la minimisation de ces deux paramètres lors de la conception d'un système d'exploitation.

Un système d'exploitation est décomposé en un ensemble de modules sous forme hiérarchique. Chaque module est menu d'une spécification fonctionnelle où il décrit son comportement sans se préoccuper des détails de sa réalisation. Cela assure une très grande sécurité par la limitation des erreurs au détriment de la perte d'efficacité par l'augmentation de l'autoconsommation en place mémoire et en temps d'exploitation.

La structure d'un système d'exploitation peut être représentée sous forme d'une suite de couches à plusieurs niveaux qui s'organisent autour d'un noyau.

Les concepts de base sur lesquels reposent les systèmes d'exploitation ne changent que sur le plan de la forme, malgré la grande évolution de l'informatique et son environnement.

Lors de la conception d'un système d'exploitation, plusieurs stratégies de confection surgissent :

- sur-mesure : système dédié ;
- universelle : système généraliste ;
- semi-confection : système dual/mixte.

Lors de la réalisation d'un système d'exploitation, il faut faire attention aux :

- *Tout faire pour ne rien faire* : vouloir tout faire pour tout le monde peut aboutir à ne pas satisfaire personne ;
- *Perfection est l'ennemie du bien* : prévoir tous les cas peut le rendre compliqué ;
- *Utiliser un marteau pour tuer une mouche* : la majorité des options est utilisée uniquement par la minorité des utilisateurs ;
- *La vie n'est pas toujours rose* : chercher un compromis entre les caractéristiques qui sont souvent contradictoires ;
- *Informatique/ordinateur pour tous* : populariser l'ordinateur en offrant à l'utilisateur une machine virtuelle plus performante et praticable que la machine physique.

Nous allons nous positionner dans le cadre de système multi-programmation en se restreignant aux systèmes centralisés, à mémoire commune, sans s'étendre aux systèmes répartis.

Paradoxe : Un ou Une ordinateur ?

En français, de quel genre le nom ordinateur, masculin ou féminin ?

Réponse 1 : "ordinateur" est du genre féminin (une ordinateur) parce que :

1. Personne d'autre que son créateur ne comprend sa logique intérieure.
2. Le langage de base qu'elle utilise avec les autres est incompréhensible pour quiconque.
3. Même la plus petite erreur rencontrée est conservée en mémoire à long terme pour être ramenée à la surface plus tard.
4. Aussitôt que vous l'utilisez régulièrement, vous vous exposez à dépenser la moitié de votre salaire pour acheter des accessoires pour elle.

Réponse 2 : "ordinateur" est du genre masculin (un ordinateur) parce que :

1. Afin d'accomplir quoi que ce soit avec lui, tu dois l'allumer.
2. Il est bourré de matériel de base, mais ne peut penser par lui-même.
3. Il est sensé régler beaucoup de problèmes, mais la moitié du temps, c'est lui le problème.
4. Aussitôt que tu en utilises un régulièrement, tu te rends compte que si tu avais attendu un peu, tu aurais obtenu un meilleur modèle.

Laquelle des deux réponses est juste !

2. Présentation d'UNIX

2.1. Historique

En 1966 : le développement de **CTSS** (Compatible Time Sharing System) au **MIT** (Massachusetts Institute of Technology). C'est un système à temps partagé et multi-programmation.

En 1967 : la conception du système **MULTICS** (Multiplexed Information and Computing Service) dans le cadre du projet MAC au MIT. C'est un système à multiple usage, multi-utilisateurs, multi-tâches, à temps partagé et opérationnel sur de puissants calculateurs centralisés.

En 1969 : la conception d'un système de fichiers mono-utilisateur en assembleur comme partie du système **GECOS**. Il est conçu par Ken Thompson et plus tard en compagnie de Dennis Ritchie sur une machine **PDP-7** de **DEC** (Digital Equipment Corporation) dans les laboratoires **BELL** de **AT&T** (American Telephon & Telegraph Campany). Par la suite, ils ont enrichi ce système par un ensemble de commandes de base avec la possibilité d'avoir 2 utilisateurs simultanés.

En 1970 : l'appellation de ce système **UNIX** par Brian Kernighan.

En 1971 : la sortie de la Version 1 qui est caractérisée par : l'écriture de la première documentation, le système de fichiers, la gestion de processus, l'interface avec l'utilisateur, les commandes et les utilitaires.

En 1972 : la sortie de la Version 2 qui est caractérisée par : la notion de tube, la réécriture en **B** (langage sans type).

En 1973 : la sortie de la Version 3 qui est caractérisée par : la réécriture en **C** (générateur de code exécutable et langage **NB** (langage **B** avec type)). **C**, inventé par D. Ritchie, est un langage destiné à la programmation système (programmes spécifiques à une machine).

En 1975 : la sortie de la **V 6** : commercialisation du système.

En 1979 : la sortie de la Version **BSD** (Berkeley Software Distribution) pour PDP-11 qui est caractérisée par : des améliorations qui sont apportées au domaine des réseaux.

En 1979 : la sortie de la **V 7** qui est caractérisée par : la transportabilité du système.

En 1979 : la sortie de la **PWB/UNIX** (Programmer's Work Bench) qui est caractérisée par : l'élaboration d'un atelier du programmeur par un autre laboratoire de **BELL**.

En 1982 : la sortie du **SYSTEM III** : version propre de **AT&T** basée sur **V 7**, c'est la première version avec un environnement et une documentation complète du système.

En 1982 : la sortie de la première version d'**ULTRIX**. **ULTRIX** est une marque déposée de **DEC**.

En 1983 : la sortie du **SYSTEM V** des laboratoires **BELL** : défini par la norme **SVID** (System V Interface Definition). Après quelques années, il y avait plus d'un million d'installations et 80 % des utilisateurs professionnels d'**UNIX**.

En 1983 : la sortie du **BSD** Version 4.1 qui est caractérisée par : la gestion de la mémoire virtuelle. Il est conçu par l'Université Berkeley de Californie pour les machines **VAX**, développées par **DEC** en 1978.

BSD Version 4.2 qui est caractérisée par : l'éditeur de texte **vi** et l'intégration dans un réseau.

BSD Version 4.3 qui est caractérisée par : le **Kerner**. Elle tourne sur toute la gamme **VAX** et sur la gamme **RISC**. MIPS Corporation est le concepteur des puces des ordinateurs RISC, MIPS 2000 et MIPS 3000.

ULTRIX-32 : la dérivée de la version **BSD** pour les machines 32 bits. **ULTRIX** est compatible avec la spécification **POSIX** (Portable Operating System for Computer Environnements).

L'apparition des systèmes **UNIX** répartis sur les noyaux Mach et Chorus.

Il y a plusieurs systèmes "**UNIX-derived**" dérivés du système **UNIX** de **AT&T**. Les systèmes existants se présentent sous les deux familles suivantes :

- "**UNIX-based**" : ce sont les systèmes à base d'**UNIX**, comme **XENIX** de **Microsoft Corporation**, **Solaris** et **SunOS** de **Sun Microsystems, Inc.** ;
- "**UNIX-like**" : ce sont les systèmes ressemblant à **UNIX**, comme **ULTRIX-32** de **DEC** et **AIX** d'**IBM**.

Il y a aussi **HP-UX** de **HP**, **SPIX** de **Bull**, **LINUX** sur **PC** à partir de microprocesseur 80386.

En général, le noyau du **SYSTEM V** et celui du **4BSD** sont les bases des différents systèmes actuellement existants dans le marché.

UNIX est devenu une marque déposée de **AT&T**.

UNIX est une marque déposée sous licence exclusive de **X-OPEN Company Ltd.**

UNIX est une marque déposée de **UNIX System Laboratories, Inc.**

2.2. Normalisation

Plusieurs groupes sont créés pour étudier la normalisation des différents systèmes d'exploitation dérivés du **System V** et de **BSD**.

En 1984, **X-OPEN** : un consortium indépendant à but non lucratif de fournisseurs internationaux de systèmes informatiques, 19 constructeurs (**AT&T**, **SUN**, etc.). Il cherche à investir des ressources techniques et de commercialisation dans un environnement commun d'applications indépendamment du fournisseur en se basant sur des normes internationales. Il a donné naissance au **SYSTEM V 4.0** à la base de **SYSTEM V**, **SUN-OS** et **XENIX**.

En 1988, **OSF** (Open Software Fondation) : une association pour assurer l'ouverture et la standardisation d'**UNIX**. Le nombre de constructeurs qu'elle réunit est passé de 8 à 76 constructeurs, (**DEC**, **IBM**, etc.). Elle a pour but la mise en place d'un environnement logiciel de base ouvert et portable au profit de l'industrie de l'informatique. Ce qui fait aboutit à la création d'un système à base de **4BSD** et **AIX**. **OSF** est devenu un système d'exploitation de **DEC**.

En 1989, **UNIX International** : association pour assurer l'intégrité de **SYSTEM V** et orienter son évolution.

DECUS (DEC Users Society) : association des utilisateurs du matériel, des logiciels et des services de **DEC**.

AFUU (Association Française des Utilisateurs d'**UNIX**).

2.3. Architecture

Un système **UNIX** regroupe 3 parties :

- le noyau (**Kernel**) :
 - le gestionnaire du système de fichiers ;
 - le gestionnaire de processus ;
 - le gestionnaire de la mémoire ;
 - le gestionnaire des périphériques ;
 - le gestionnaire des entrées-sorties de bas niveaux.
- l'interpréteur de commandes :
 - l'interface pour l'interprétation des commandes utilisateurs ;
 - le langage de commandes (**Shell**) pour le développement de procédures de commandes.
- les utilitaires :
 - l'éditeur de texte ;
 - le système de traitement de textes ;
 - les compilateurs ;
 - le compilateur de compilateurs ;
 - l'assembleur et l'éditeur de lien ;
 - la communication entre les utilisateurs ;
 - l'atelier de programmation ;
 - la documentation en ligne ;
 - etc.

2.4. Caractéristiques

Une liste, non exhaustive, des caractéristiques du système **UNIX** est :

- l'écriture en un langage évolué (langage **C**) ;
- la hiérarchisation du système de fichiers en forme d'arborescence ;
- le comportement uniforme des opérations d'entrées-sorties par rapport à l'utilisateur ;
- la communication entre les processus par le mécanisme de production-consommation (tube) ;

- la simplicité et la puissance du langage de commandes, ce qui permet la possibilité d'étendre l'ensemble des commandes disponibles sans modifier le noyau du système ;
- la possibilité de l'exécution en avant plan, en arrière plan et en différés, ce qui permet l'exécution de processus asynchrones ;
- la protection efficace des fichiers et des répertoires, aussi bien entre les utilisateurs et les non utilisateurs ;
- le partage efficace des ressources ;
- l'utilisation du temps partagé ;
- l'interaction du système ;
- la richesse de ses utilitaires à travers une large bibliothèque de logiciels;
- l'appel procédural du noyau et de la bibliothèque ;
- le mode esclave et mode maître ;
- la présence de deux méthodes d'accès aux informations d'un fichier : accès séquentiel et accès direct ;
- le multi-utilisateurs ;
- la multi-tâches ;
- l'introduction d'un système de messagerie pour la communication entre les utilisateurs ;
- l'aide automatique ;
- la présence sur toutes les gammes d'ordinateurs (micro, mini et super-ordinateur), mais plus particulièrement sur les stations de travail en réseau et micro-ordinateurs multi-postes ;
- etc.

UNIX est le leader mondial des systèmes d'exploitation au niveau international. UNIX est devenue une norme pour les systèmes d'exploitation et un enjeu primordial.

2.5. Linux

D'après l'encyclopédie Wikipedia : <http://fr.wikipedia.org/wiki/Linux>

Au sens strict, Linux est le nom du noyau de système d'exploitation libre, multi-tâches et multi-utilisateurs de type UNIX créé par Linus Torvalds en 1991, souvent désigné comme le noyau Linux. Par extension, Linux désigne couramment le système d'exploitation libre combinant le noyau et un ensemble d'utilitaires systèmes. Pour désigner cet ensemble, la Free Software Foundation (FSF) soutient la désignation GNU/Linux afin de rappeler que le noyau Linux est généralement distribué avec de nombreux logiciels ainsi que l'infrastructure du projet GNU.

Pour l'utilisateur final, Linux se présente sous la forme d'une distribution Linux, c'est-à-dire du système d'exploitation accompagné d'une collection de logiciels très variés. Originellement développé pour les compatibles PC, Linux est utilisé sur tous types de matériel, du téléphone portable au superordinateur. Son premier marché est celui des serveurs informatiques, suivi par les systèmes embarqués. La mascotte de Linux est Tux, un manchot dessiné par Larry Ewing en 1996. Linux est une marque déposée au nom de Linus Torvalds.

GNU (GNU's Not UNIX), a été créé par Richard Stallman en 1984 sous le principe : « ramener l'esprit de coopération qui prévalait dans la communauté informatique dans les jours anciens ». Parmi les composants de base du système GNU, on retrouve : la collection de compilateurs GNU (GCC), les

outils binaires GNU (binutils), le Shell Bash, la bibliothèque C GNU (glibc), les outils de base GNU (coreutils), l'éditeur de texte (Emacs), etc. Cependant, le principal composant encore manquant étant le noyau. S'agissant d'un véritable projet Open Source, le code source correspondant aux paquetages propres à GNU se relève de la licence GNU GPL (General Public License).

Parmi les étapes marquantes, on peut citer le lancement de l'environnement graphique KDE par Matthias Ettrich en 1996 puis de son concurrent GNOME par Miguel de Icaza en 1997, les deux étant basés sur le système de fenêtrage X11 issu des travaux du MIT (Massachusetts Institute of Technology).

Parmi les plus célèbres distributions, on peut citer la Slackware qui est la première distribution Linux apparue en 1992, toujours activement maintenue par Patrick J. Volkerding ; la Debian, éditée par une communauté de développeurs ; la Red Hat, éditée par l'entreprise américaine du même nom qui participe également au développement de Fedora Core ; ou encore la SuSE, à l'origine dérivée de Slackware avec ajout de certains sous-système issus de Red Hat, aujourd'hui éditée par la société Novell.

De nombreuses autres distributions plus ou moins spécialisées existent, étant pour la plupart dérivées des projets sus-cités. Par exemple, voici quelques distributions spécialisées « environnement de bureau » : Ubuntu, éditée par Canonical Ltd qui est dérivée de Debian ; Mepis également basée sur Debian ; Zenwalk dérivée de Slackware ; Mandriva, dérivée de Red Hat, aujourd'hui éditée par la société française de même nom et impliquée dans plusieurs projets libres. Il existe également des distributions dites LiveCD, dont la plus célèbre est Knoppix, qui offrent la possibilité de démarrer un système d'exploitation Linux complet et d'accéder à de nombreux logiciels à partir du support (CD ou DVD) sans installation préalable sur le disque dur, et sans altérer son contenu. Cette souplesse d'utilisation a fait qu'elles sont devenues un support très populaire de démonstration d'utilisation de Linux, et sont même utilisées comme outils de maintenance système.

Il y a des émulateurs du système Linux sous Windows : **CygWin**

2.6. Systèmes d'exploitation mobiles

Les systèmes d'exploitation mobiles sont apparus : Symbian OS (Nokia), Firefox OS (Mozilla), Android (Google), iOS (Apple), Blackberry OS (RIM), Windows Phone (Microsoft), Bada (Samsung) ... Ils sont utilisés initialement par les applications des Smartphones et les tablettes, puis étendu aux montres, appareils photo, etc. Ils se différencient par leurs objectifs et leurs philosophies suivants ceux de leurs concepteurs. La plupart de ces systèmes d'exploitation se basent sur Linux. Cependant, iOS, dérivé de Mac OS X, se base sur BSD.

Partie I : Utilisation

3. Session de travail

3.1. Compte d'un utilisateur

Un utilisateur est identifié par rapport aux autres utilisateurs par son nom de connexion. Le système attribue à chaque utilisateur un numéro appelé **UID (User IDentifier)**. Le système identifie un utilisateur par son **UID**.

Les utilisateurs sont organisés sous forme de groupes. Un utilisateur peut être affecté à un ou plusieurs groupes lors de l'ouverture de son compte, ou même après, par l'administrateur du système, ou par lui-même. Un utilisateur est membre d'un **groupe principal** et, éventuellement, à d'autres **groupes secondaires**. Lors de la création des groupes, l'administrateur du système attribue à chaque groupe un nom et un numéro appelé **GID (Group IDentifier)**.

3.2. Connexion

Si le système est en marche, dès que l'utilisateur allume un terminal connecté au système, le message suivant s'affiche sur l'écran, éventuellement après avoir appuyé sur la touche **Return** :

login:

L'utilisateur peut taper sur le clavier le nom de connexion, appelé aussi nom de login, **nom_u**, juste après **login:** qui a été communiqué par l'administrateur du système. Il faut appuyer sur la touche **Return** à la fin pour valider la commande.

Selon qu'il y ait un mot de passe ou non, le message suivant s'affiche :

Password :

Il faut taper alors le mot de passe suivi toujours de la touche **Return**. L'écho étant coupé, les lettres du mot de passe tapées n'apparaissent pas sur l'écran. Le mot de passe est personnel et confidentiel. Pour modifier ou créer un mot de passe, chose qui doit être faite périodiquement par mesure de sécurité, il y a la commande **passwd**, étudiée dans le chapitre **Sécurité de fichiers**.

Le nom de connexion est en général en minuscule. **UNIX** est sensible à la casse des lettres. Il faut respecter la casse des lettres du nom de connexion et du mot de passe et faire attention à l'activation de la touche **Caps lock**.

Une fois l'utilisateur connecté sur le système, un ensemble de messages du système est affiché sur l'écran. Cela dépend d'une part du message du jour d'un fichier particulier maintenu par l'administrateur du système et d'autre part des fichiers d'environnement de l'utilisateur qu'il peut modifier à son gré. Ceci est étudié dans le chapitre **Langages de commandes**.

Exemples :

Last login : Fri 25 19 :25 :13 on tty 05
Last login : Fri Sep 25 19 :25 :13 on term/5

À la dernière ligne, l'invite (**prompt**), ou le message d'attente, apparaît au début et le curseur clignote juste à côté. L'invite est un ensemble de caractères que l'utilisateur peut modifier dans ses fichiers d'environnement.

Exemples :

Science>
/usr/home/omar>

S'il y a une erreur de connexion le message suivant s'affiche :

Login incorrect
login:

Cela peut être dû soit à un mauvais nom de connexion, soit à un mauvais mot de passe, soit aux deux à la fois.

3.3. Commande

La session de travail commencera réellement à cet instant. L'utilisateur peut taper maintenant toutes sortes de commandes qu'il désire, suivant ses droits. L'interpréteur de commandes **Shell** va les interpréter pour que le système les exécute.

La première tâche qui s'exécute à la connexion est l'exécution des fichiers d'environnement de l'utilisateur puis la commande **Shell**.

Il y a 3 types de commandes :

- les commandes systèmes livrées, du noyau ;
- les programmes exécutables, compilés et liés ;
- les procédures de commandes.

La plupart des commandes du système sont soient des procédures **Shell**, soient des programmes C. Toutes les commandes correspondent à des fichiers sauf pour les commandes internes aux interpréteurs de commandes.

En général, une commande est l'appel de l'exécution du programme correspondant au fichier exécutable.

UNIX fait la différence entre les minuscules et les majuscules dans une ligne de commandes.

En général, la syntaxe d'une commande est la suivante :

chemin_c [options] [paramètres]

Exemples :

ls
ls -l
ls -li rapport
ls -l -i rapport
ls --help

Les différents éléments composant la commande sont séparés par au moins un espace.

En général, une erreur de syntaxe génère un message d'erreur.

Exemple :

lps

En **C Shell** : **lps: command not found**

En **Bourne Shell** : **lps: not found**

Exemple Solaris :

lps: Command not found

Les options peuvent être combinées entre elles avec ou sans le signe moins.

Sous Linux :

Ctrl Alt F2 : Passe du mode graphique au mode terminal.

Ctrl Alt F7 : Passe du mode terminal au mode graphique.

3.4. Déconnexion

Pour terminer la session de travail, il suffit de taper **logout** ou **Ctrl d** une plusieurs fois.

Un ensemble de messages va apparaître sur l'écran de l'utilisateur en provenance de ses fichiers d'environnement.

Le fait d'éteindre un terminal ne termine pas la session de travail et n'arrête pas la machine.

4. Système de gestion de fichiers

Un **système de gestion de fichiers** permet de conserver un ensemble d'informations dans un **fichier** en leur donnant un nom.

Le système de gestion de fichiers d'**UNIX** est caractérisé par :

- un mécanisme de sécurité efficace ;
- une interface unique pour toutes les opérations d'entrée-sortie.

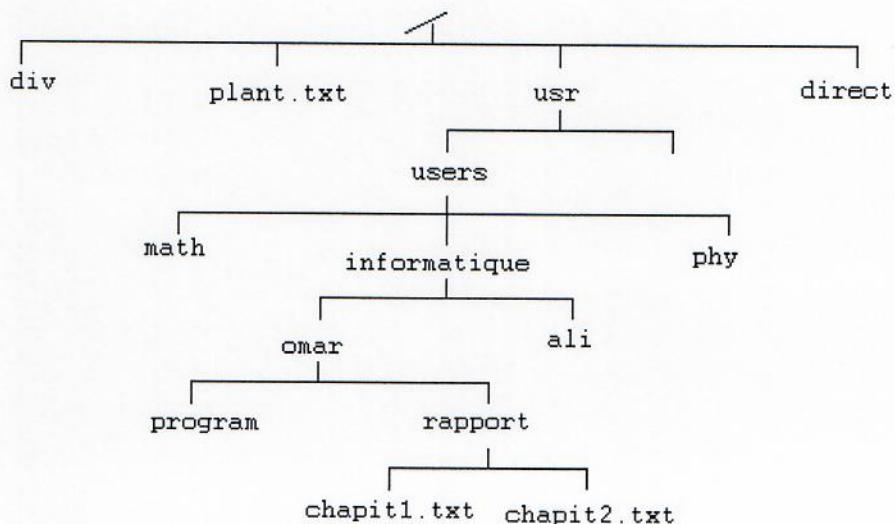
Il y a 3 types de fichiers :

- les fichiers ordinaires (**file**).
- les répertoires (**directory**).
- les fichiers spéciaux (**device**) qui correspondent aux périphériques d'entrée-sortie et aux outils de communications entre processus. Les périphériques sont considérés comme des fichiers normaux, au point de vue interne.

4.1. Structure arborescente

Le système de gestion de fichiers d'**UNIX** range les fichiers sous une structure arborescente, à la manière après de **MS-DOS**. Les fichiers sont regroupés dans des répertoires. Les répertoires peuvent regrouper également d'autres répertoires, dits sous répertoires, etc. Ainsi on obtient une structure hiérarchisée de fichiers et de répertoires. Dans cet arbre, la **racine** et les nœuds sont des répertoires et les feuilles sont des fichiers ou des répertoires vides.

Exemple :



Un exemple d'une branche de cet arbre est le suivant :

/usr/users/informatique/omar/rapport/chapit1.txt

Le **nom complet**, le **chemin d'accès**, d'un fichier ou d'un répertoire est une liste de noms de répertoires séparés par le symbole / et se terminant par le nom du fichier ou du répertoire désigné. Un chemin est un seul mot sans espace.

Il y a 2 types de chemin d'accès :

- le chemin d'accès **absolu**, commençant par / qui est le répertoire racine (**root**) ;
- le chemin d'accès **relatif**, par rapport au répertoire courant.

Exemple :

Le chemin d'accès absolu et relatif du fichier **chapit1.txt** sont respectivement :

- **/usr/users/informatique/omar/rapport/chapit1.txt**
- **rapport/chapit1.txt** si le répertoire courant est **/usr/users/informatique/omar**

Chaque utilisateur a son propre répertoire, appelé : **répertoire de base**, créé par l'administrateur du système lors de l'ouverture de son compte. Tous les répertoires et les fichiers d'un utilisateur sont regroupés dans son répertoire de base. En général, le répertoire de base d'un utilisateur porte son nom de connexion.

L'utilisateur peut se déplacer dans l'arborescence, avec éventuellement des restrictions de protection. À un instant donné, le répertoire où se retrouve l'utilisateur est le **répertoire de travail** ou encore dit : **le répertoire courant**. Il y a un pointeur qui indique le nom du répertoire courant.

Le système affecte à chaque fichier ou répertoire un numéro unique, appelé : **inode (index node, nœud d'index)**.

La description d'un **inode** est donnée par :

- le nom du fichier ou du répertoire ;
- le nom du propriétaire ;
- le nom du groupe principal du propriétaire ;
- le numéro d'inode ;
- le jeu des protections ;
- la taille en octet pour les fichiers et en bloc (ex. de 512 octets) pour les répertoires ;
- le nombre de liens durs ou d'entrées ;
- le fichier ou le répertoire d'origine lors d'un lien symbolique ;
- le type : fichier, répertoire, lien symbolique, etc. ;
- la date de la création ;
- la date de la dernière modification ;
- la date du dernier accès.
- etc.

Le nom d'un fichier est un identificateur (ex. d'au plus 14 caractères). Il y a une différence entre les lettres en majuscule et celles en minuscule. Eventuellement, il ne porte pas le caractère espace.

Au point de vue de système, un fichier est une chaîne de caractères non structurés et un répertoire est un fichier contenant une suite de couples formés d'un numéro d'inode et d'un nom.

Suivant l'encodage, un caractère est codifié dans un octet ou plus.

Les caractères spéciaux sont :

- . : le répertoire courant ;
- .. : le répertoire père d'un répertoire ;
- / : le répertoire racine, et joue le rôle du séparateur entre les répertoires dans un chemin ;
- ~ : le répertoire de base de l'utilisateur ;
- ~nom_u** : le répertoire de base de l'utilisateur **nom_u**.

Les caractères génériques sont :

- ? : un et un seul caractère ;
- * : une chaîne de caractères ;
- [...] : une liste restreinte de caractères.

Exemples :

[a-z]*[0-9]

tous les fichiers dont les noms commencent par une lettre en minuscule et se terminent par un chiffre.

.

tous les fichiers dont les noms avec extension.

tous les fichiers dont les noms sans extension (**Solaris** : tous les fichiers avec ou sans extension).

F?

tous les fichiers dont les noms commencent par la lettre F puis un autre caractère quelconque.

/usr/

le répertoire **usr** fils de la racine. / peut être ajouté à la fin d'un chemin de répertoire.

Remarques :

Pour manipuler les fichiers et les répertoires, il n'est pas obligatoire de se déplacer dans leurs répertoires pères, il est même déconseillé.

Pour manipuler les fichiers et les répertoires, l'utilisateur doit posséder les permissions requises.

4.2. Gestion des répertoires

mkdir [-p] chemin_r_1 [chemin_r_2 ...]

crée des nouveaux répertoires de nom **chemin_r_1**, **chemin_r_2**, Suivant la version, les répertoires parent doivent existés (**Linux**) ou seront créés aussi.

p : crée les répertoires parents nécessaires.

mv chemin_r_a chemin_r_n

renomme le répertoire **chemin_r_a** par **chemin_r_n** si **chemin_r_n** n'existe pas sinon elle déplace le répertoire **chemin_r_a** dans **chemin_r_n** comme fils tout en gardant sa description. Suivant la version, s'il existe déjà, il le remplace par le nouveau nom ou bien ne fait rien.

rmdir chemin_r_1 [chemin_r_2 ...]

détruit les répertoires **chemin_r_1**, **chemin_r_2**, ... s'ils ne contiennent aucun fils, s'ils ne sont pas le répertoire courant ou un ascendant.

rm -r *chemin_r_1* [*chemin_r_2* ...]

détruit les répertoires ***chemin_r_1*, *chemin_r_2*, ...** avec leur contenu s'ils ne sont pas le répertoire courant ou un descendant.

En général, un fichier détruit ne peut plus être récupéré. Les fichiers stockés dans la poubelle peuvent être récupérés tant que la poubelle n'est pas vidée et l'espace mémoire déjà occupé par ces fichiers n'est pas encore attribué à un autre.

cp -r[i] *chemin_r_1* [*chemin_r_2* ...] *chemin_r*

recopie la liste des répertoires ***chemin_r_1*, *chemin_r_2*, ...** dans le répertoire ***chemin_r***. Si les répertoires ***chemin_r_1*, *chemin_r_2*, ...** existent auparavant dans le répertoire ***chemin_r*** ils sont remplacés par défaut. Si le répertoire ***chemin_r*** n'existe pas il sera créé.

i : mode interactif, avec demande de confirmation.

cd [*chemin_r*]

change le répertoire courant et qui devient le répertoire ***chemin_r*** s'il existe. Le répertoire courant devient le répertoire de base par défaut.

Exemples :

cd ..

change le répertoire courant par le père du répertoire courant, c'est à dire à l'étage au-dessus, si l'utilisateur possède la permission.

cd /

change le répertoire courant par la racine, si l'utilisateur possède la permission.

cd ~[*nom_u*]

change le répertoire courant par le répertoire de base de l'utilisateur ***nom_u***. Elle se déplace au répertoire de base de l'utilisateur par défaut, si l'utilisateur possède la permission.

pwd

(Print Working Directory)

affiche le chemin absolu du répertoire courant.

ls [-lrR[F1acdgiu]] [*chemin_f_r*]

(Linux : **dir** aussi)

liste les fichiers et les sous répertoires du répertoire ***chemin_f_r*** ou bien la description du fichier ***chemin_f_r***. Elle liste le contenu du répertoire courant par défaut.

I : les champs listés sont : le mode, le nombre de liens, le propriétaire, la taille, la date et l'heure de la dernière modification et le nom du fichier ou du sous répertoire et le nom du fichier de lien symbolique s'il y a lieu. La liste est triée par ordre croissant par nom ;

r : liste les fichiers et les sous répertoires à l'envers de l'ordre alphabétique des noms ou des dates de modification ;
R : liste les fichiers d'une façon récursive ;
1 : liste un nom par ligne, jusqu'à 5 noms par ligne par défaut ;
a : liste tous les fichiers, même les fichiers cachés qui commencent par un « . » ;
lc : trie la liste par ordre décroissant de la date de la dernière modification, du nom par défaut;
(Solaris et Linux : t : trie de la liste par ordre décroissant de la date de la dernière modification, du nom par défaut)
c : trie la liste par ordre alphabétique croissant des noms des fichiers (**Linux** : par ordre croissant des dates de la dernière modification des fichiers) ;
ld : décrit le répertoire courant ;
g : donne aussi le groupe ;
i : donne aussi le numéro d'inode dans la première colonne ;
lu : trie la liste par ordre décroissant de la date du dernier accès au lieu de la date de la dernière modification (**Solaris** : n'existe pas) ;
F : indique la nature des fichiers au début (**Linux** : à la fin) de chaque élément :
/ : pour un répertoire ;
* : pour un fichier exécutable ;
= : pour un fichier **socket** ;
@ : pour un lien symbolique ;
Rien : pour un fichier ordinaire.

Exemples :

ls -lig plan.txt

235 lrwx-wx--x 1 omar informatique 9 Dec 3 14:54 plan.txt -> /plan.txt

ls -ligu plan.txt

235 lrwx-wx--x 1 omar informatique 9 Dec 5 11:52 plan.txt -> /plan.txt

ls -l a*

affiche la description de tous les fichiers et les répertoires dont le nom commence par a.

Le mode d'un fichier regroupe le type du fichier, représenté par le premier caractère, et la protection qui est représenté par les 9 caractères suivants.

Les types de fichiers sont :

- : pour un fichier ordinaire ;

d : pour un répertoire ;

I : pour un lien symbolique ;

c : pour un fichier spécial structuré avec transfert de caractères pour les entrées-sorties sur disque ;

b : f pour un fichier spécial non structuré avec transfert de blocs pour les entrées-sorties sur terminal ;

s : pour un fichier **socket** ;

(Solaris : p : pour un fichier spécial de pipe line)

Les protections, ou droits d'accès, sont représentées par 9 caractères, regroupés en 3 champs de 3 caractères, chacun de l'ensemble { r , w , x , - , s , t }, respectivement pour le propriétaire, le groupe et les autres utilisateurs.

Le nombre de liens est 1 pour les fichiers normaux, le nombre de liens durs pour les fichiers à lien dur et le nombre d'entrées pour un répertoire.

Un répertoire possède les liens, les entrées, suivants :

- un lien pour lui-même ;
- et un lien vers son père ;
- plus un lien vers chacun de ses sous répertoires ;
- plus autant de liens durs qu'il possède.

En **Linux**, après la modification d'un fichier de nom « f », l'ancienne version est sauvegardée dans le fichier nommé « f~ ».

La taille exprime le nombre d'octets pour les fichiers, le nombre de blocs pour les répertoires, le nombre de caractères du nom du chemin complet du fichier avec lequel le fichier en question a un lien symbolique et le numéro primaire et le numéro secondaire pour les fichiers spéciaux. Le numéro primaire, d'un fichier spécial, identifie une classe de périphériques. Le numéro secondaire identifie un numéro de cette classe. Pour les répertoires, **Total** exprime la taille totale, en blocs, du répertoire en question. En général, un bloc est de taille 1 Ko (512 o en **Linux**).

du [-as] [chemin_r]

(Disk Usage)

donne la taille en bloc du répertoire **chemin_r** avec ses sous répertoires, du répertoire courant par défaut.

a : pour chaque fichier en plus ;

s : uniquement le total.

diff [-lr] chemin_r_1 chemin_r_2

(Solaris)

compare le contenu des deux répertoires **chemin_r_1** et **chemin_r_2**. Les résultats sont présentés sous la forme suivante (**Linux** : la présentation est différente) :

La liste des fils de **chemin_r_1** et **chemin_r_2** en même temps

La liste des fils de **chemin_r_1** uniquement

La liste des fils de **chemin_r_2** uniquement

I : affiche les résultats différemment ;

r : compare récursivement aussi les sous répertoires deux à deux (**Linux** : ce qui est par défaut).

diremp [-d] chemin_r_1 chemin_r_2

(Solaris) (Linux : non)

compare le contenu des deux répertoires **chemin_r_1** et **chemin_r_2**. Les résultats sont présentés sous la forme suivante :

La liste des fils de **chemin_r_1** uniquement | La liste des fils de **chemin_r_2** uniquement

La liste des fils de **chemin_r_1** et **chemin_r_2** en même temps

d : compare aussi les fichiers de même nom deux à deux à la façon de la commande **diff** pour les fichiers.

3.3. Gestion des fichiers

cp [-i] chemin_f_1 [chemin_f_2 ...] chemin_r

recopie les fichiers *chemin_f_1*, *chemin_f_2*, ... dans le répertoire *chemin_r* avec les mêmes noms. Si ces fichiers existent auparavant dans *chemin_r* ils sont remplacés par défaut.

i : mode interactif, elle demande une confirmation avant de recopier sur un fichier existant.

cp [-i] chemin_f_s chemin_f_d

recopie le fichier *chemin_f_s* dans le fichier *chemin_f_d*. Si *chemin_f_d* existe auparavant il est remplacé par défaut.

Exemple :

cp /dev/null plan.txt

crée un nouveau fichier vide de nom plan.txt ou le vide s'il existe auparavant.

rm [-if] chemin_f_1 [chemin_f_2 ...]

détruit les fichiers *chemin_f_1*, *chemin_f_2*, ... si l'utilisateur a la permission d'écriture, sinon il demande une confirmation s'il est propriétaire des fichiers. Les fichiers ne doivent pas être en court d'utilisation.

- : pour un fichier dont le nom commence par - ;

i : mode interactif, avec demande de confirmation ;

f : force la suppression, sans demande de confirmation. C'est ce qui est par défaut.

Remarques :

L'option **-if** tient compte de **f** alors que **-fi** tient compte de **i**.

Un fichier détruit ne peut plus être récupéré sauf en cas de présence de gestion de poubelle, tant que la poubelle n'est pas vidée et l'espace mémoire déjà occupé par ce fichier n'est pas encore attribué à un autre.

mv [-if] chemin_f_a nom_f_n

change le nom du fichier *chemin_f_a* qui devient *nom_f_n*. Si *nom_f_n* existe auparavant il est remplacé, sauf s'il est protégé en écriture une confirmation est demandée. Les caractéristiques du fichier restent inchangées.

mv [-if] chemin_f_a chemin_f_n

déplace le fichier *chemin_f_a* qui devient *chemin_f_n*, si *chemin_f_n* est un chemin avec un répertoire autre que celui de *chemin_f_a* se dernier sera déplacé dans le répertoire spécifié. Si *chemin_f_n* existe auparavant il est remplacé, sauf s'il est protégé en écriture une confirmation est demandée. Les caractéristiques du fichier restent inchangées.

mv ch_r_a ch_r_n

mv [-if] *chemin_f_1* [*chemin_f_2* ...] *chemin_r*

déplace les fichiers *chemin_f_1*, *chemin_f_2*, ... dans le répertoire *chemin_r*. Si ces fichiers existe auparavant dans *chemin_r* ils sont remplacés, sauf s'ils sont protégés en écriture une confirmation est demandée. Les caractéristiques de ces fichiers restent inchangées.

more [+num +/chaîne] [*chemin_f*]

liste le contenu du fichier *chemin_f* page par page, c'est à dire 23 lignes ou écran par écran. Elle liste l'entrée-sortie standard par défaut.

+num : commence à partir de la ligne numéro *num*, la première ligne par défaut. La première ligne est de numéro 1 ;

+/chaîne : commence deux lignes avant la première ligne qui contient au moins une occurrence de *chaîne*.

--More--(num_1 %)

[Return [num_2]Space q Q = v h ? num/chaîne !chemin_c Ctrl l ...]

num_1 : exprime le pourcentage du fichier déjà listé.

Return : affiche la ligne suivante ;

[num_2]Space : affiche les *num_2* lignes suivantes, la page suivante par défaut ;
q ou **Q** : quitte **more** ;

= : donne le numéro de la ligne courante, la dernière ligne affichée ;

v : appel l'éditeur de texte **vi** à la ligne courante sans quitter **more** ;

h ou **?** : donne un aide de **more** ;

[num]/chaîne : cherche l'occurrence numéro *num* de *chaîne*. Elle cherche la première occurrence par défaut ;

!chemin_c : exécute la commande *chemin_c* d'Unix sous **Shell** sans quitter **more** ;
Ctrl l : réaffiche l'écran.

less [*chemin_f*]

liste le contenu du fichier *chemin_f* page par page, c'est à dire 23 lignes ou écran par écran. Elle liste l'entrée-sortie standard par défaut.

cat [-bn] *chemin_f_1* [*chemin_f_2* ...]

affiche le contenu des fichiers *chemin_f_1*, *chemin_f_2*, ... d'un seul coup et d'une façon séquentielle.

b : ignore les lignes blanches, en les affichant sans les numérotées, et numérote les lignes, à partir du début du fichier, au début ;

n : numérote les lignes au début.

Remarque :

more, **less** et **cat** sont utilisées uniquement pour les fichiers textes, de plus sans pouvoir les modifier.

Remarques :

- La taille d'un fichier vide est 0.
- La taille de la touche *entrée* est 2 caractères sous MS-DOS et 1 caractère sous Linux.
- En code ASCII, la taille d'un fichier contenant n caractères est n+1 octets sous MS-DOS et n octets sous Linux.

tee [-a] *chemin_f*

transmet les sorties dans le fichier ***chemin_f*** jusqu'à **Ctrl d**, en plus de la sortie standard. Si le fichier existe déjà il sera détruit par défaut.

a : ajoute à la fin du fichier. Si le fichier n'existe pas il sera créé.

Exemples :

ls -l | tee f.txt

.

Ctrl d

tee f.txt

bien
bien

Ctrl d

crée un fichier **f.txt** contenant **bien**, une seule fois.

tee -a f.txt

bien
bien

Ctrl d

ajoute **bien** à la fin du fichier **f.txt**.

head [-num] [*chemin_f_1* [*chemin_f_2* ...]]

liste les **num** premières lignes des fichiers ***chemin_f_1*, *chemin_f_2*, ...**, (Linux : **-n num** aussi). Elle affiche les 10 premières lignes par défaut de valeur. Elle liste l'entrée-sortie standard par défaut de fichiers.

tail [-num] [*chemin_f_1* [*chemin_f_2* ...]]

liste les **num** dernières lignes des fichiers ***chemin_f_1*, *chemin_f_2*, ...**, (Linux : **-n num** aussi). Elle affiche les 10 dernières lignes par défaut de valeur. Elle liste l'entrée-sortie standard par défaut de fichiers.

wc [-lwc] [chemin_f_1 [chemin_f_2 ...]]

(Word Count)

donne le nombre de lignes, de mots et de caractères, ou d'octets, suivant l'ordre des options, (Linux : toujours dans cet ordre), qui se trouvent dans les fichiers **chemin_f_1**, **chemin_f_2**, Elle donne toutes les informations par défaut d'options. Elle liste l'entrée-sortie standard par défaut de fichiers.

Exemples :

wc plan.txt

25 123 1265 plan.txt

wc -l plan.txt

25 plan.txt

wc plan.txt trait.txt

25 123 1265 plan.txt

15 13 128 trait.txt

40 136 1393 total

diff [-bie] chemin_f_1 chemin_f_2

donne toutes les lignes des fichiers **chemin_f_1** et **chemin_f_2** qui représentent au moins une différence entre eux. En particulier, elle donne toutes les modifications qu'il faut portées à **chemin_f_1** pour obtenir **chemin_f_2** à la manière des commandes de l'éditeur de texte **ed**.

b : ignore les chaînes d'espaces et les tabulations ;

i : ignore la différence entre les lettres majuscules et les lettres minuscules ;

e : produit une procédure de commandes pour obtenir **chemin_f_2** à partie de **chemin_f_1**.

Dans le résultat, on a :

Les lignes en provenance de **chemin_f_1** commencent par le symbole <.

Les lignes en provenance de **chemin_f_2** commencent par le symbole >.

num_1 a num_3,num_4 : ajouter dans **chemin_f_1** les lignes de l'intervalle

[num_3,num_4] de **chemin_f_2** après la ligne **num_1** de **chemin_f_1** ;

num_1,num_2 d num_3 : détruire dans **chemin_f_1** les lignes de l'intervalle

[num_1,num_2] de **chemin_f_1** en référence à la ligne **num_3** de **chemin_f_2** ;

num_1,num_2 c num_3,num_4 : changer dans **chemin_f_1** les lignes de l'intervalle

[num_1,num_2] de **chemin_f_1** par les lignes de l'intervalle **[num_3,num_4]** de **chemin_f_2**.

Exemple :

(Solaris)

Dans **f1** :

C

Dans **f2** :

A

B
C
A

diff f1 f2

0a1,2
>A
>B
1a4
>A

cmp chemin_f_1 chemin_f_2

compare les deux fichiers *chemin_f_1* et *chemin_f_2* et donne le numéro du caractère et de la ligne où se trouve la première différence entre les deux fichiers.

Exemple :

cmp plan.txt trait.txt
plan.txt trait.txt differ : char 5, line 1

comm [-123] chemin_f_1 chemin_f_2

compare les deux fichiers *chemin_f_1* et *chemin_f_2* de données triés par ordre ASCII puis affiche 3 colonnes en lignes, contenant respectivement les données appartenant à *chemin_f_1* uniquement, à *chemin_f_2* uniquement puis à *chemin_f_1* et à *chemin_f_2* en même temps. Les colonnes sont interclassées.

- 1** : supprime la colonne 1.
2 : supprime la colonne 2.
3 : supprime la colonne 3.

uniq [-udc] chemin_f_1 chemin_f_2

(Solaris)

uniq [-udc] chemin_f_1
(Linux)

élimine les lignes répétées contigües de *chemin_f_1* et met le résultat dans *chemin_f_2*, (Linux : affiche le résultat). *chemin_f_1* est considéré trier auparavant, c'est-à-dire les mêmes lignes considérées sont contigües.

- u** : seulement les lignes non répétées contigües sont reproduites ;
d : seulement les lignes répétées contigües sont reproduites, mais une seule fois ;
c : précède chaque ligne par le nombre de répétitions contigües.

file chemin_f_r

donne le type du fichier ou du répertoire *chemin_f_r*.

La liste des types des fichiers est :

- Directory ;
- No such file or directory ;
- Vax demand paged pure executable not stripped ;
- ASCII text ;
- Permission denied ;
- Symbolic link to ***chemin_f_r*** ;
- (**Solaris** : empty file)
- (**Solaris** : Commands text)
- etc.

sort [-bfnr...] [*chemin_f_1* [*chemin_f_2* ...]]

trie les lignes des fichiers ***chemin_f_1*, *chemin_f_2*, ...**, avec fusion, par ordre alphabétique **ASCII** croissant, vers la sortie standard. Elle trie l'entrée standard par défaut.

b : ignore les espaces lors de la détermination de la position des champs lors du trie (par défaut en **Linux**) ;

f : ignore la différence entre les lettres majuscules et celles minuscules lors du trie ;

n : trie en fonction des valeurs numériques et non des valeurs **ASCII** ;

r : trie par ordre décroissant ;

m : interclasse les fichiers supposés déjà triés ;

o *chemin_f* : met le résultat dans le fichier ***chemin_f*** ;

t *car* : trie en fonction des champs séparés par le caractère ***car***. Les espaces et les tabulations sont les séparateurs par défaut ;

d : tienne en compte seulement les lettres, les chiffres, les espaces et les tabulations lors du trie (par défaut en **Linux**) ;

num : trie en fonction de la valeur du champ ***num***. Le premier champ ou colonne est d'ordre 1 ;

+num_1-num_2 : restreint le trie du champ ***num_1*** compris au champ ***num_2*** non compris (**Solaris**) ;

+num_1- : restreint le trie du champ ***num_1*** compris jusqu'à la fin de la ligne (**Solaris**) ;

k num_1,num_2 : restreint le trie aux champs ***num_1*** et ***num_2*** (**Linux**).

Exemple :

7 est plus grand que 11 en **ASCII**.

sort f.txt →	11	sort -n f.txt →	AZ
	12		Fgr
	123		hr
	13		hr f
	7		7
	AZ		11
	Fgr		12
	hr		13
	hr f		123

ls -l | sort -b -n +4 -5

trie le contenu du répertoire courant en fonction de la taille des fichiers, quatrième champ.

split [-num] chemin_f [chaîne]

partage horizontalement le fichier **chemin_f** en plusieurs sous fichiers de même taille **num** lignes et de noms **chaîneaa**, **chaîneab**, ... La taille est 1000 lignes et les noms sont **xaa**, **xab**, ... par défaut.

cat chemin_f_1 chemin_f_2 ...

fusionne horizontalement les lignes des fichiers **chemin_f_1**, **chemin_f_2**, ...

cut [-bef...] chemin_f

partage verticalement en affichant certains champs, ou colonnes, des lignes du fichier **chemin_f**. Il faut spécifier la nature des colonnes.

b num : l'octet **num** ;

c num : le caractère **num**. Dans un fichier texte en ASCII, un caractère est codifié sur un octet ;

f num : le champ **num** ;

d car : les champs sont délimités par le caractère **car**. Le délimiteur est la tabulation par défaut.

Exemple :

wc f.txt | cut -f 1 -d '' ↔ **wc -l f.txt**

affiche le nombre de lignes du fichier f.txt.

Exemple :

La liste des numéros de colonnes **num** :

3-9 : de 3 à 9 ;

-5 : de 1 à 5 ;

5- : de 5 à la fin de la ligne ;

1,3,5 : 1, 3 et 5.

Exemple :

Le délimiteur **car**, est entre cote éventuellement :

'' : l'espace ;

:: le symbole ::

paste [-sd...] chemin_f_1 [chemin_f_2 ...] (Solaris)

fusionne verticalement les champs, ou les colonnes, des lignes des fichiers **chemin_f_1**, **chemin_f_2**, ...

- : pour une lecture à partir de l'entrée standard ;

s : fusionne les lignes d'un même fichier ;

d car : remplace la fin des lignes par le caractère **car**. Elle remplace la fin des lignes par une tabulation par défaut.

car peut être un caractère normal ou bien :

\n : nouvelle ligne ;
\t : tabulation ;
\\" : backslash ;
\0 : empty string, not a nul character ;
etc.

chemin_c | paste [...] -

fusionne verticalement les champs, ou les colonnes, des lignes du résultat de la commande *chemin_c*.

Exemples :

ls | paste -d " " -

affiche le contenu du répertoire courant dans une seule colonne.

ls | paste -d " " - -

affiche le contenu du répertoire courant dans 3 colonnes séparées par des tabulations.

paste -s -d "\t\n" f.txt

combine chaque ligne impaire avec la ligne paire qui la suit du fichier f.txt, en les séparant par le caractère de tabulation et en les terminant par le caractère de nouvelle ligne.

joint [-t...] chemin_f_1 chemin_f_2 ...
(Solaris)

join [-t...] chemin_f_1 chemin_f_2 ...
(Linux)

joint les lignes qui ont les mêmes champs des fichiers *chemin_f_1*, *chemin_f_2*, ... supposés triés par ordre croissant du code ASCII. Les champs sont numérotés à partir de l'ordre 1.

- : pour une lecture à partir de l'entrée standard pour un des fichiers ;

t car : sépare les lignes jointes par le caractère *car* ;

a1 : ajoute les lignes appartenant uniquement à *chemin_f_1* ;

a2 : ajoute les lignes appartenant uniquement à *chemin_f_2* ;

j1num : joint le champ numéro *num* du fichier *chemin_f_1* (Linux : **-1num**) ;

j2num : joint le champ numéro *num* du fichier *chemin_f_2* (Linux : **-2num**) ;

o chemin_f_num_1.champ_num_1 chemin_f_num_2.champ_num_2 ... : spécifie les champs des fichiers à joindre sous cet ordre : le champ numéro *num_1* du fichier *chemin_f_num_1* puis le champ numéro *num_2* du fichier *chemin_f_num_2*, ...

Exemple :

join -o 1.2 2.1 1.3 f1 f2

joint le champ 2 de f1 puis le champ 1 de f2 puis le champ 3 de f1.

look [-f...] chaîne chemin_f
(Solaris)

retourne les lignes du fichier trié (Linux : le tri n'est pas obligatoire) *chemin_f* qui commencent par *chaîne*.

- f** : ignore la distinction entre la majuscule et le minuscule ;
- t car** : ignore tous les caractères après le caractère *car* ;
- d** : prend en compte uniquement les lettres, les chiffres, les espaces et la tabulation lors de la comparaison.

Remarque :

Le tri est important puisque le système ne parcourt pas entièrement le fichier. Il s'arrête dès qu'il arrive au niveau du fichier où doit être présent la chaîne.

basename chemin_f

retourne le nom du fichier du chemin de fichier *chemin_f*.

Exemple :

basename /usr/bin/f.txt → f.txt

dirname chemin_f

retourne le chemin du répertoire père du chemin de fichier *chemin_f*.

Exemple :

dirname /usr/bin/f.txt → /usr/bin

find chemin_r expr

cherche récursivement tous les fichiers et les répertoires, à partir du répertoire *chemin_r*, qui vérifient l'expression *expr*.

La forme de *expr* est :

- name nom_f_r** : de nom *nom_f_r* ;
- iname nom_f_r** : de nom *nom_f_r* sans sensibilité à la casse ;
- group nom_g** : de groupe *nom_g* ;
- inum num** : de numéro d'inode égal à *num* ;
- perm num** : de permission *num* avec *num* en octal ;
- size num** : de taille *num* blocs (ex. 1 bloc = 512 o) ;
- size numc** : de taille *num* caractères ;
- links num** : de *num* liens ;
- type car** : de type *car* avec :
 - car = b* : fichier spécial en bloc (par exemple le disque) ;
 - car = c* : fichier spécial en caractère (par exemple le terminal) ;
 - car = d* : répertoire ;

car = f : fichier ordinaire ;
car = l : lien symbolique ;
car = p : port de tube, pipeline ;
car = s : socket ;
-user nom_u : de propriétaire *nom_u* ;
-atime num : accédés depuis *num* jours ;
-mtime num : modifiés depuis *num* jours ;
-maxdepth num : descend jusqu'à la profondeur *num*. **-maxdepth 1** pour une recherche non récursive ;
-print : affiche le résultat de **find** sur l'écran ;
-exec chemin_c {} \; : exécute la commande *chemin_c* sur le résultat de **find**. La commande doit être terminée par \;. Il faut ajouter {}, ou \{}, dans la commande, si elle nécessite des arguments et le résultat de **find** sera l'argument de la commande. Il faut respecter la position de {} ou \{} suivant la commande à exécuter.

La valeur numérique de **num** peut être précédée par le signe + pour exprimer plus que la valeur ou le signe - pour exprimer moins que la valeur. Une valeur sans signe exprime exactement la valeur.

'expr' : l'expression *expr* ;
\(expr\) : l'expression *expr* ;
! expr : la négation de l'expression *expr* ;
expr_1 expr_2 : la conjonction des 2 expressions *expr_1* et *expr_2* ;
expr_1 -o expr_2 : la disjonction des 2 expressions *expr_1* et *expr_2*.

Exemples :

find . -size -20 -print

liste tous les fichiers descendant du répertoire courant et de taille inférieure à **20 blocs**.

find . -size +20 -exec rm {} \;

détruit tous les fichiers descendant du répertoire courant et de taille supérieure à **20 blocs**.

find . \(-name a.out -o -name '*.exe' \) ! -atime +3 -exec rm {} \;

(Linux : **find . \(-name a.out -o -name "*.exe" \) ! -atime +3 -exec rm {} \;**)

(Linux : **find . \(-name a.out -o -name *.exe \) ! -atime +3 -exec rm {} \;**)

détruit tous les fichiers descendant du répertoire courant de nom **a.out** ou d'extension **exe** et qui ne sont pas accédés depuis plus de **3 jours**.

find . -size 0 -exec mv {} R \;

déplace tous les fichiers vides du répertoire courant au répertoire **R**.

find . -perm -421 -print

liste tous les fichiers descendant du répertoire courant et de permission moins que **421**.

grep [-chlnvi] expr [chemin_f_1 [chemin_f_2 ...]]
(Global Regular Expression Printer)

donne la liste des lignes des fichiers *chemin_f_1*, *chemin_f_2*, ... qui contiennent au moins une occurrence de l'expression *expr*, de l'entrée standard par défaut. S'il y a plusieurs fichiers comme argument, elle affiche le nom du fichier et la ligne pour chaque occurrence. La forme de l'expression *expr* est comme celle de l'éditeur ed.

- c** : compte le nombre de lignes où apparaît l'argument pour chaque fichier ;
- h** : supprime le nom du fichier placé devant chaque ligne ;
- I** : liste les noms des fichiers où apparaît l'argument, un nom de fichier par ligne et sans répétition ;
- n** : précède chaque ligne par le numéro d'ordre dans le fichier, en commençant par 1 ;
- v** : affiche les lignes qui ne contiennent pas l'argument *expr* ;
- i** : ignore la distinction entre le majuscule et le minuscule lors de la recherche ;
- w** : cherche l'expression sous forme d'un mot.

L'expression *expr* peut prendre l'une des formes suivantes :

- \car : le caractère *car*. Il est obligatoire de mettre \car si car est dans :
 - { ; , \$, * , [,] , ^ , | , (,) , ? , \ , } ;
 - ^ : le début d'une ligne ;
 - \$: la fin d'une ligne ;
 - . : un caractère quelconque ;
- [c1c2...] : un caractère de {c1,c2,...} ;
- [a-zA-Z0-9] : un caractère de {a,...,z,0,...,9} ;
- 'expr' : l'expression *expr*.

Exemples :

- [1-79] : {1 , ... , 7 , 9 }
- [01][0-9] : { 0 , ... , 19 }
- [a-c][dk] : { ad , bd , cd , ak , bk , ck }
- [^a] : tout caractère autre que a.

Exemples :

grep '^l' plan.txt

affiche les lignes qui commencent par l du fichier plan.txt, fils du répertoire courant, si l'utilisateur possède la permission de lecture du fichier.

grep '[rk]' plan.txt

affiche les lignes du fichier plan.txt qui contient r ou k.

grep -l 'ba' *

affiche les noms des fichiers du répertoire courant qui contiennent au moins une occurrence de ba.

grep '\...' plan.txt

affiche les lignes du fichier **plan.txt** qui contiennent "." puis au moins deux autres caractères quelconques.

grep '^a.*b' plan.txt

affiche les lignes du fichier **plan.txt** qui commencent par **a**, puis au moins un caractère quelconque, puis **b**.

grep '^a.*b\$' plan.txt

affiche les lignes du fichier **plan.txt** qui commencent par **a**, puis au moins un caractère quelconque, puis se termine par **b**.

grep '^[a-k]' plan.txt

affiche les lignes du fichier **plan.txt** qui commencent par l'une des lettres suivantes : { **a,...,k** }.

grep -v 'a\$' plan.txt

affiche les lignes du fichier **plan.txt** qui ne se termine pas par **a**.

grep '[^a]\$' plan.txt

affiche les lignes du fichier **plan.txt** qui ne contiennent que **a** ou qui sont vides.

grep -v '^bien\$' plan.txt

affiche les lignes du fichier **plan.txt** qui ne contiennent pas uniquement **bien**.

grep '^...\$' plan.txt

affiche les lignes du fichier **plan.txt** de taille exactement **3**.

grep '...\$' plan.txt

affiche les lignes du fichier **plan.txt** de taille supérieure ou égale à **3**.

grep '> '\$1'\$' plan.txt

affiche les lignes du fichier **plan.txt** qui contient > puis espace puis le contenu du premier paramètre de position à la fin de la ligne.

grep ^\$a plan.txt

affiche les lignes du fichier **plan.txt** qui commencent par le contenu de la variable **a**.

grep ^\$a[2] plan.txt

affiche les lignes du fichier **plan.txt** qui commencent par le contenu de la deuxième case de la variable **a**.

fgrep [-chlnvief] expr [chemin_f_1 [chemin_f_2 ...]]
(Fast Global Regular Expression Printer)

donne la liste des lignes des fichiers **chemin_f_1, chemin_f_2, ...** qui contiennent au moins une occurrence de l'expression **expr**, de l'entrée standard par défaut. **expr** est une chaîne de caractères sans caractères spéciaux ni caractères génériques. S'il y a plusieurs fichiers comme argument, elle affiche le nom du fichier et la ligne pour chaque occurrence. L'expression **expr** est entre simple cote.

egrep [-chInvie] expr [chemin_f_1 [chemin_f_2 ...]]
(Expression Global Regular Expression Printer)

donne la liste des lignes des fichiers *chemin_f_1*, *chemin_f_2*, ... qui contiennent au moins une occurrence de l'expression *expr*, de l'entrée standard par défaut. S'il y a plusieurs fichiers comme argument, elle affiche le nom du fichier et la ligne pour chaque occurrence.

La forme de *expr* est :

*expr** : une séquence de 0 ou plusieurs occurrences de *expr* ;

expr+ : une séquence d'au moins une occurrence de *expr* ;

expr? : une séquence de 0 ou 1 occurrence de *expr* ;

expr_1expr_2 : la conjonction de *expr_1* et *expr_2* ;

expr_1|expr_2 : la disjonction de *expr_1* et *expr_2* ;

expr_1 op expr_2 : le résultat de l'expression logique avec *op* un opérateur relationnel dans : { <, <=, >, >=, = } ;

e *expr* : l'expression spéciale *expr* ;

f *chemin_f* : l'expression prise du fichier *chemin_f*.

Exemples :

egrep '1|3' plan.txt

affiche les lignes du fichier **plan.txt** qui contient **1** ou **3**.

egrep '1''3' plan.txt

affiche les lignes du fichier **plan.txt** qui contient **13**.

egrep '1+' plan.txt

affiche les lignes du fichier **plan.txt** qui contient au moins une occurrence de **1**.

egrep "^\w{fr}|\w{an}" plan.txt

affiche les lignes du fichier **plan.txt** qui commencent par « **fr** » ou « **an** ».

Remarque :

L'ordre de précédence des opérateurs est :

[]

* ? +

Concaténation

|

Remarque :

Le code de retour est 0 s'il y a au moins une occurrence de l'expression, 1 s'il n'y a pas d'occurrence et 2 s'il y a une erreur de syntaxe dans la commande.

5. Gestion de processus

5.1. Processus

En **UNIX**, et autre, toute activité s'exécute sous forme d'un **processus**. Un processus est l'enchaînement des phénomènes répondant à un certain schéma et aboutissant à un résultat déterminé. L'exécution d'une commande ou d'un programme correspond à un processus qui peut générer d'autres.

Exemples :

date
racine.exe
lien

Un processus est l'exécution d'une **image**. Une image est composée d'un programme en exécution et de son environnement système. Elle est principalement caractérisée par :

- le code programme, les données et la pile ;
- les registres système ;
- les états des fichiers ouverts ;
- les variables d'environnement :
 - le répertoire courant ;
 - le **Shell** utilisé ;
 - etc.

Le système attribue à chaque processus un numéro appelé **PID (Process IDentifier)** lors de sa création par rapport au système entier. Ce compteur est mis à jour à chaque démarrage du système.

Tout processus a un niveau de priorité. Les processus des utilisateurs ont un niveau de priorité dans un intervalle (ex. entre -20, le plus rapide, et +20, le plus lent). Un processus est plus prioritaire qu'un autre s'il a un niveau de priorité inférieur à l'autre.

À la création d'un processus d'un utilisateur, le système lui affecte un niveau de priorité (ex. 4 en **C Shell** et 10 en **Bourne Shell**). Un utilisateur peut exécuter un processus avec un niveau supérieur à une valeur (ex. 0). Il peut aussi augmenter le niveau de priorité d'un processus au cours de son exécution. La diminution du niveau de priorité ne peut être faite que par l'administrateur du système.

UNIX est un système à **temps partagé**, à un instant donné, il y a un seul processus qui s'exécute dans un système mono processeur. Tant que le processus n'utilise pas d'entrée-sortie, le système lui donne une fraction du temps **CPU** et une priorité. Le niveau de priorité change automatiquement lors de l'exécution d'un processus. Il est fonction : du niveau de priorité de base, du niveau de priorité courant, de l'utilisation de **CPU** et de la commande **nice**. Le système favorise les processus qui consomment peu de temps **CPU** ou peu d'entrées-sorties par rapport au temps **CPU**. En effet, plus il y a des demandes de temps **CPU** plus le niveau de priorité diminue et plus il y a des demandes d'entrée-sortie plus le niveau de priorité augmente.

5.2. Interpréteur de commandes

Au lancement d'une commande, celle-ci est exécutée comme le processus fils d'un processus parent. Un processus peut créer un autre processus lors de son exécution. Tout processus a un parent sauf le processus **init** de **PID** égal à 1. Pour un utilisateur, le premier processus exécuté est l'interpréteur de commandes **Shell** de connexion qui est fils de **init**, (**Solaris** : Chaque utilisateur, terminal, a un processus **login** à la connexion. Le processus **Shell** de connexion est fils du processus **login**. **login** est un processus système fils de **init**). Tout processus d'un utilisateur sera fils du processus **Shell** courant ou du processus qui l'a lancé. (**Linux** : **init** >>> **gnome-terminal** >>> **bash** >>> ... ou **init** >>> **kde** >>> **bash** >>> ...)

La description d'un processus est donnée par :

- le numéro d'identification **PID** ;
- le numéro d'identification du père **PPID** ;
- le terminal d'attachement pour les entrées-sorties, mais un processus peut ne plus être attaché à aucun terminal ;
- le temps **CPU** ;
- la taille mémoire requise pour son exécution ;
- les fichiers auxquels il accède ;
- etc.

chemin_s

crée un processus exécutant le **Shell** ***chemin_s*** après empilement de l'environnement courant et la création d'un nouvel environnement. C'est sous cette commande que l'utilisateur se trouve implicitement à la connexion, ou lors de l'appel du **Terminal**. ***chemin_s*** peut être égal à **csh**, **sh** ou **sh5** (**Solaris** : ksh) (**Linux** : bash, tcsh, ...). En général, l'invite de **csh** est % (**Solaris** : ISMAILIA) et l'invite de **sh** ou **sh5** (**Solaris** : ksh) est \$.

Ctrl d

tue le processus **Shell** courant. Si c'est le processus **Shell** de connexion, la session de travail sera terminée. S'il y a des processus en exécution en arrière plan, il affiche un message d'erreur (ex. **There are stopped jobs**). Il faut faire une deuxième fois **Ctrl d** pour tuer les tâches en court afin de terminer la session de travail. **Ctrl d** est équivalent au signal **eof**.

5.3. Manipulation de processus

Pour exécuter une commande, l'interpréteur de commandes crée un nouveau processus fils et attend sa fin. La création et l'attente sont deux primitives du système. Le processus **Shell** est endormi et l'utilisateur n'a plus le message d'attente jusqu'à la fin de la commande.

Une commande qui ne peut pas tenir sur une ligne est terminée par \.

Une commande peut être exécutée **en avant plan (foreground)** dite aussi en mode interactif, conversationnel, dialogué, ou **en arrière plan (background)** dite aussi en mode parallèle, c'est-à-dire le lancement d'exécution d'une commande sans attendre sa fin d'exécution. Une commande peut aussi

être exécutée **en différée** à une date donnée par l'utilisateur ou **en soumission** c'est-à-dire laisser le soin au système de la lancer au moment propice.

Un processus en arrière plan peut devenir en avant plan et inversement. Un processus peut se terminer normalement, être suspendu, être arrêté, etc.

Un processus en exécution en arrière plan utilise la sortie standard pour ses résultats et pour les messages d'erreur par contre, il utilise le fichier spécial vide **/dev/null** comme fichier d'entrée. Lors de la demande d'une entrée, le processus sera suspendu. Pour entrer les données, il faut placer le processus en avant plan. Normalement, il faut utiliser une redirection des entrées-sorties lors d'une exécution en arrière plan ou en soumission.

Les processus **actifs**, en cours d'exécution en arrière plan ou en suspension, sont appelés des **tâches** (**job**). Le système attribue à chaque tâche un numéro d'ordre par rapport à la session de travail courante de l'utilisateur, en commençant par 1 et en l'incrémentant de 1 à chaque attribution.

Une tâche **détachée** est une tâche qui continue à s'exécuter en arrière plan après déconnexion de l'utilisateur. En général, elle devient attachée au processus **init**.

ps [-#acegklstxuvwx]

(**Processus Status**)

donne la liste ponctuelle des processus en cours qui sont attachés au terminal courant (ex. avec les informations suivantes : **PID**, **TTY**, **TIME**, **COMMAND**).

a : tous les processus, même ceux des autres utilisateurs et des autres terminaux si l'utilisateur a la permission, uniquement les processus propres à l'utilisateur et reliés au même terminal par défaut (**Solaris** : tous les processus sauf ceux non attachés à un terminal) ;

x : tous les processus, même les processus en arrière plan ;

u : avec les informations : **USER**, **PID**, **%CPU**, **%MEM**, **SZ**, **RSS**, **TT**, **STAT**, **TIME**, **COMMAND**. Les informations : **PID**, **TT**, **STAT**, **TIME**, **COMMAND** par défaut (**Solaris** : **unom_u** tous les processus de l'utilisateur **nom_u**) ;

I : avec les informations : **F**, **S**, **UID**, **PID**, **PPID**, **CP**, **PRI**, **NI**, **ADDR**, **SZ**, **RSS**, **WCHAN**, **STAT**, **TTY**, **TIME**, **COMMAND**. Les informations sont : **PID**, **TTY**, **STAT**, **TIME**, **COMMAND** par défaut ;

tnum : tous les processus du terminal numéro **num** uniquement, avec ? pour aucun terminal et co pour la console ;

e : affiche l'environnement du **Shell** (**Solaris** : tous les processus en exécution) ;

#num_p : uniquement le processus numéro **num_p** (**Linux** : **num_p** sans #) ;

pnum_p : uniquement le processus numéro **num_p** ;

gnum_g : uniquement les processus propre au groupe **num_g** ;

f : affiche l'état du processus avec toutes les informations.

Le résultat en colonnes de la commande **ps** est de la forme :

F : le numéro du drapeau (**flag**) du processus indiquant son état en mémoire ;

USER : le nom du propriétaire ;

UID : le numéro du propriétaire ;

PID : le numéro du processus ;

PPID : le numéro du processus auquel il est attaché ;
GID : le numéro du groupe du propriétaire ;
SID : le numéro de la session ;
CP : le facteur en temps **CPU** utilisé pour le calcul du niveau de la priorité d'exécution ;
PRI : le niveau de priorité d'exécution ;
NI : le niveau de priorité d'exécution au départ (ex. entre -20 et 20), ou celui donné par **nice** ;
ADDR : l'adresse du processus en mémoire ;
%CPU : le pourcentage du temps **CPU** utilisé ;
%MEM : le pourcentage de la taille réelle de la mémoire utilisée ;
SZ : la taille de l'image mémoire, la mémoire virtuelle, du processus (ex. en **Ko**) ;
RSS : la taille de la mémoire réelle du processus (ex. en **Ko**) ;
TT ou **TTY** : le numéro du terminal de contrôle de processus avec **co**, ou ?, pour la console et ? les processus non attachés à un terminal ;
WCHAN : le nombre de fois que le processus à demander des paginations ;
STAT ou **S** : le statut du processus, formé par 5 lettres au maximum ;

1ère lettre : l'état du processus :

- R** : en exécution ;
- T** : en suspension par une interruption (par **Ctrl z** ou **stop** par exemple) ;
- Z** : tué (par **kill -9** par exemple) ;
- P** : en attente de page lors d'un va et vient ;
- D** : en attente d'une entrée-sortie du disque ;
- S** : endormi de moins de 20 s ;
- I** : endormi de plus de 20 s ;

2ème lettre : la mémoire centrale :

- W** : processus déchargé par **swap** ;
- Z** : tué mais pas encore éliminé ;
- rien** : dans le fichier **core** ;

3ème lettre : le niveau de priorité du processus :

- N** : priorité a été diminuée ;
- <** : priorité a été augmentée ;
- rien** : en exécution sans traitement spécial ;

4ème lettre : la mémoire virtuelle :

- A** : VA-ANOM ;
- S** : VA-SEQL ;
- rien** : VA-NORM ;

5ème lettre : le processus vectoriel :

- V** : processus utilisant un matériel vectoriel ;
- rien** : processus n'utilisant pas un matériel vectoriel. ;

ou bien, l'état du processus est exprimé par :

- O** : en exécution ;
- S** : endormi (en attente d'un événement) ;
- R** : dans la queue d'exécution ;
- I** : en état de création ;
- Z** : en état de terminaison ;
- T** : stoppé ;
- X** : en attente de la mémoire primaire.

TIME : temps d'exécution cumulé, de l'utilisateur ajouté au temps système en minutes et en secondes ;

COMMAND ou **COMD** : le nom de la commande avec ses options et ses paramètres correspondant.

pstree

affiche la structure arborescente des processus.

top [-] [d delay] [p pid] [q] [c] [C] [S] [s] [i] [n iter] [b]
décrit des processus.

chemin_x &

exécute **chemin_x** en arrière plan. Elle retourne le numéro de la tâche et du processus correspondant attribué par le système en **C Shell**. Elle retourne uniquement le numéro du processus en **Bourne Shell**.

Exemples en C Shell :

rm f.txt &

[2] 18457

...

[2] Done rm f.txt

rm g.txt &

[2] 18457

g.txt not found

rm &

[2] 18457

[2] + Stopped (tty input) rm

racine.exe &

[1] 12458

Donner la valeur :

[1] + Stopped (tty input) racine.exe

fg

racine.exe

25

La racine est : 5

[1] Done racine.exe

ls -l &

[1] 14578

...

[1] Done ls -l

```
ls -l > f.txt &
[1] 14578
...
[1] Done ls -l > f.txt
```

```
vi &
vi f.txt &
[1] +Stopped (tty output) vi
```

Exemples en Bourne Shell :

```
vi &
14578
Visual needs adressible cursor or upline capability
```

```
racine.exe &
14578
Donner la valeur :
Standard input : Tried to read past end of file
14578 Trace/BPT trap - core dumped
```

chemin_x_1 ; chemin_x_2
exécute *chemin_x_1* puis *chemin_x_2* d'une manière séquentielle.

(chemin_x_1 ; chemin_x_2) &
exécute *chemin_x_1* puis *chemin_x_2* d'une manière séquentielle en arrière plan.

nohup chemin_x &

exécute la commande *chemin_x* en arrière plan sans interruption du processus lors d'une déconnexion. La sortie, résultat et erreur, est redirigée vers le fichier **nohup.out** créé dans le répertoire courant (**Linux** : vers la fin du fichier **nohup.out** dans le répertoire de base) avec incrémentation du niveau de priorité (ex. de 5) en cas de **Bourne Shell**. La sortie est redirigée vers l'écran en cas de **C Shell** en gardant le niveau de priorité égal à 0. La commande est immunisée des signaux : **HUP**, **QUIT** et **TERM** (**Linux** : seulement **HUP**). *chemin_x* est une commande simple (ni liste ni pipe line) sinon il faut la mettre dans un fichier et l'appeler comme une procédure de commandes.

Exemple :

```
nohup ls -lR / >& f.txt &
(C Shell)
[1] 14578
avec NI dans le résultat de ps est égal à 0.
```

Exemple :

```
nohup ls -lR / &
(Bourne Shell)
14578
avec NI dans le résultat de ps est égal à 5.
```

jobs [-l]

donne la liste des tâches en suspension ou en exécution en arrière plan avec leurs numéros et les numéros des processus correspondants, sans les numéros des processus par défaut.

La liste des états des processus est :

Running : processus en exécution ;

Stopped : processus en suspension par **Ctrl z** par exemple ;

Stopped (signal) : processus en suspension par **stop** par exemple ;

Stopped (tty output) : **vi** par exemple ;

Stopped (tty input) : processus en attente d'une entrée ;

Killed : processus en terminaison brutale par **kill -9** par exemple ;

Terminated : processus en terminaison normale par **kill -15** par exemple ;

Done : fin normale ou non d'un processus ;

Exit 1 : exécution en arrière plan d'un fichier non exécutable ou n'existe pas.

+ : la **tâche courante**, la plus récente, en suspension ou en exécution en arrière plan.

Exemples :

jobs -l

[1] + 450 Stopped vi f.txt

[2] - 455 Stopped (tty output) vi ff.txt

[3] 457 Done rm

[4] + 460 Running (sleep (5 ; pwd) &)

La tâche 1 est suspendue par **Ctrl z** et la tâche 2 est lancée en arrière plan.

pidof nom

retourne le numéro du processus correspondant au programme ou à la commande **nom**, plusieurs numéros s'il y a plusieurs commandes de même nom.

batch num_1 [num_2] chemin_x

at num_1 [num_2] chemin_x

soumet l'exécution du fichier exécutable **chemin_x** jusqu'à une heure et une date données. Elle retourne le numéro de la tâche soumise qui lui est attribué par le système.

num_1 : de la forme **hhmm**, heure et minute.

num_2 : de la forme **mmjj**, mois et jour.

Exemples :

at 2am trait.exe > f1.txt

94.296.0200.21

lance l'exécution de trait.exe à 2 h du matin.

at 0135 lien > f2

94.296.0135.22

lance l'exécution de lien à 1 h 35 mn du matin.

at 135 jan 21 lien > f3

95.020.0135.23

lance l'exécution de lien à 1 h 35 mn du matin de 21 janvier l'an 95.

at [-csm f *chemin_f*] num_1 [num_2] [+ num_3 nom]

(Solaris)

batch -f *chemin_f*

at num_1 [num_2] [+ num_3 nom] < *chemin_f*

(Linux)

soumet l'exécution de l'entrée standard jusqu'à **Ctrl d**, à une heure et une date données. Elle retourne le numéro de la tâche soumise qui lui est attribué par le système.

c : en **C Shell**, suivant la valeur de la variable d'environnement **SHELL** par défaut ;

s : en **Bourne Shell**, suivant le **Shell** spécifié dans la variable d'environnement **SHELL** par défaut ;

m : avertir l'utilisateur de la fin d'exécution par un message ;

f *chemin_f* : exécute le fichier ***chemin_f***, l'invite local **at>** est affiché par défaut ;

num_1 : l'heure qui peut être : now, noon, midnight, h, hh, hhmm, h:m, h:mm, hh:mm,... avec am ou pm ;

num_2 : le jour qui peut être : today, tomorrow, ... ;

num_3 nom : le nombre à incrémenter à partir du moment de lancement avec l'unité **nom** qui peut être : minute, day, week, month, year, minutes, days, weeks, months, years, ...

Exemples :

(Solaris)

at 0815am Jan 25

at 5 pm Friday

at now + 1 minute

at noon

Une tâche soumise n'est plus attachée au terminal où elle a été lancée, il faut donc utiliser une redirection des entrées-sorties. En général, le système envoie un mail à l'utilisateur concernant les entrées-sorties s'il n'y a pas eu de redirections. Par exemple, les tâches soumises sont examinées toutes les 15 minutes. L'utilisateur doit être autorisé à utiliser **at**.

Le fichier **/usr/lib/cron/at.allow** (Linux : **/etc/at.allow**) contient les utilisateurs qui peuvent soumettre des processus. Le fichier **/usr/lib/cron/at.deny** (Linux : **/etc/at.deny**) contient les utilisateurs qui ne le peuvent pas. Si ces deux fichiers n'existent pas, seul l'administrateur du système peut soumettre des processus. Si le fichier **/usr/lib/cron/at.deny** (Linux : **/etc/at.deny**) existe et vide, tous les utilisateurs peuvent soumettre des processus. (Solaris : **/usr/sbin/cron.d/at.allow**, ou **/etc/sbin/cron.d/at.allow**, est le fichier qui contient la liste des utilisateurs autorisés à utiliser **at**, à raison d'un utilisateur par ligne. **/usr/sbin/cron.d/at.deny**, ou **/etc/sbin/cron.d/at.deny**, est le fichier qui contient la liste des utilisateurs non autorisés à utiliser **at**).

at et **batch** ajoutent un fichier, au répertoire **/usr/spool/at** (Solaris : **/var/spool/cron/atjobs**) (Linux : **/var/spool/at**), contenant, en plus du fichier exécutable, la commande **cd** vers le répertoire courant et l'environnement **Shell**.

at -l

liste les tâches en soumission, par **at** ou **batch**, de l'utilisateur courant.

Exemple :**at -l**

94.296.0200.21

94.296.0135.22

at -r num_s

(Solaris)

at -d num_s

(Linux)

supprime la tâche numéro **num_s** en soumission, par **at** ou **batch**, de l'utilisateur.

En Solarise, **num_s** est de la forme : **yy.ddd.hhmm.num** avec :

yy : l'année ;

ddd : le numéro du jour à partir de 0 ;

hhmm : l'heure en heure et en minute ;

num : le numéro d'ordre.

Exemple :**at -r 94.296.0200.21****atq [nom_u]**

(Solaris)

liste les tâches en soumission, par **at** ou **batch**, de l'utilisateur **nom_u**, de tous les utilisateurs par défaut

atrm [-afi] [num_s] [nom_u]

(Solaris)

supprime la tâche numéro **num_s** en soumission, par **at** ou **batch**, de l'utilisateur **nom_u**.

a : supprime toutes les tâches ;

f : force la suppression ;

i : suppression en mode interactif, ce qui est par défaut.

batch chemin_x(Solaris : **batch <chemin_x**)

laisse le soin au système de lancer l'exécution du fichier exécutable **chemin_x** au moment propice.

Exemple :**batch racine.exe**

94.312.0000.25

ls -l racine.exe

-rwxr-xr-- 1 omar 21504 Nov 22 19 :55 racine.exe

ls -l /usr/spool/at/94.312.0000.25

-rw----- 1 omar 21755 Dec 12 14 :18 94.312.0000.25

Ctrl c

tue le processus courant, en avant plan.

Ctrl z

suspend l'exécution du processus courant, en avant plan.

kill -l

liste tous les noms des signaux ou des interruptions existants.

Les 30 signaux, numérotés de 1 à 30, du système sont : **HUP INT QUIT ILL TRAP IOT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM URG STOP TSTP CONT CHLD TTIN TTOU IO XCPU XFSZ VTALRM PROF WINCH LAST USR1 USR2**.

HUP numéro **1** : arrête brutalement le processus ;

KILL numéro **9** : arrête immédiatement le processus ainsi que ses fils ;

TERM numéro **15** : arrête le processus en différé.

kill [-num_nom_s] num_p

envoie le signal de numéro ou de nom **num_nom_s** au processus numéro **num_p**, le signal **TERM** de numéro **15** par défaut.

kill [-num_nom_s] %num_t (C Shell)

envoie le signal de numéro ou de nom **num_nom_s** à la tâche numéro **num_t**.

Exemple :

kill -9 %1

[1] Killed vi

kill %2

kill -15 %2

kill -TERM %2

[2] Terminated vi

kill -1 %2

arrête la tâche sans rien signaler.

fg [num_t]

fg [%num_t]

(C Shell)

reprend l'exécution de la tâche d'ordre **num_t** en avant plan, la dernière tâche par défaut si elle existe sinon elle affiche : **fg : No current job.**

Exemples :

fg 2

fg %2

reprend la tâche d'ordre 2.

fg -2

reprend la tâche avant la dernière tâche.

bg [num_t]**bg [%num_t]****(C Shell)**

reprend l'exécution de la tâche d'ordre **num_t** en arrière plan, la dernière tâche par défaut si elle existe sinon elle affiche : **bg : No current job.**

Exemples :**bg 2****bg %2**

reprend la tâche d'ordre 2.

bg -2

reprend la tâche avant la dernière tâche.

bg %**bg %+****bg %%**

reprend la dernière tâche.

bg %-

reprend la tâche précédente.

bg %num_t

reprend la tâche numéro **num_t**.

bg %chainé

reprend la tâche commençant par **chainé**.

bg %?chainé

reprend la tâche dont la commande est égale à **chainé**.

stop num_p**(C Shell)****(Solaris : non)**

suspend le processus numéro **num_p**.

stop %num_t**(Bourne Shell)****(Solaris : C Shell)**

suspend la tâche numéro **num_t**.

Ctrl s

suspend le défilement du résultat d'un processus sur l'écran.

Ctrl q

reprend l'édition arrêtée par **Ctrl s**.

nice [+num] [chemin_x]

(C Shell)

(valeur standard UNIX de gentillesse)

exécute le fichier exécutable *chemin_x*, le **Shell** courant par défaut, avec le niveau de priorité égal à **num** (**Solaris** : augmente le niveau de priorité de **num**, de 10 par défaut). (ex. La valeur de **num** est comprise entre 1 et 19, la priorité est 4 par défaut.)

/etc/renice +num -p num_p

(C Shell)

(Solaris : non)

change le niveau de priorité du processus numéro **num_p** pour devenir égal à **num**. (ex. La valeur de **num** est comprise entre 1 et 19.)

nice [-num] chemin_x

(Bourne Shell)

exécute le fichier exécutable *chemin_x* avec le niveau de priorité égal à **num** (**Solaris** : augmente le niveau de priorité de **num**). La valeur de **num** est comprise entre 1 et 19, la priorité est 10 par défaut. (**Solaris** : une incrémentation du niveau de priorité de 10 par défaut ou de 19 si **num** > 19).

time chemin_x

(C Shell)

exécute le fichier exécutable *chemin_x* puis affiche les informations suivantes :

Uu Ss E P% X+Dk I+Oio Fpf+Ww

U : temps CPU utilisateur en seconde, ou en h:mn:s ;

S : temps CPU système en seconde, ou en h:mn:s ;

E : temps écoulé de la machine en seconde, ou en h:mn:s (**E=U+S**) ;

P : pourcentage de cycle de CPU utilisé ;

X : espace occupé par le programme résident en **Ko** ;

D : espace occupé par les données et la pile résident en **Ko** ;

I : nombre d'opérations de lecture ;

O : nombre d'opérations d'écriture ;

F : nombre de pages utilisées ;

W : nombre de va et vient effectués.

time

(C Shell)

affiche le temps utilisé pour le processus **C Shell** courant et ses fils.

/bin/time *chemin_x*

(Bourne Shell)

(Solaris : /usr/bin/time *chemin_x*)

exécute le fichier exécutable *chemin_x* puis affiche approximativement le temps globale, le temps d'exécution puis le temps passé dans le système en seconde (c'est le temps déroulé pour l'utilisateur du lancement jusqu'à la terminaison).

Exemples :

time date

0.0u 0.1s 0 :00 100% 26+12k 0+1io 0pf+0w

/bin/time date

0.2 real 0.0 user 0.1 sys

suspend

(C Shell)

suspend le **Shell** courant. **Ctrl c** l'annule. Les commandes en arrière plan continues à s'exécuter.

sleep num

suspend le travail sur le terminal, en ne prenant plus de nouvelles commandes, pour une durée de *num* secondes. **Ctrl c** l'annule. L'interpréteur de commandes ne reçoit plus de nouvelles commandes mais continue pour les anciennes commandes.

Exemples :

sleep 120 ; *chemin_x*

reporte l'exécution de la commande *chemin_x* après 120 s, tout en refusant de nouvelles commandes pendant ce temps.

(sleep 120 ; *chemin_x*)&

reporte l'exécution en arrière plan de la commande *chemin_x* après plus que 120 s.

wait

(C Shell)

suspend le travail sur le terminal jusqu'à ce que tous les processus en arrière plan soient exécutés. L'effet de **wait** peut être annulé par **Ctrl c**.

wait [*num_p*]

(Bourne Shell)

suspend le travail sur le terminal jusqu'à ce que le processus numéro *num_p* soit terminé, tous les processus en arrière plan soient exécutés par défaut. L'effet de **wait** peut être annulé par **Ctrl c**.

Exemple :

wait 12345 ; *chemin_x*

reporte l'exécution de la commande *chemin_x* après la terminaison du processus 12345. Ce qui est important si la commande *chemin_x* nécessite les résultats du processus 12345.

notify [%num_t] (C Shell)

signale la fin d'exécution de la tâche en arrière plan numéro **num_t**, des tâches en cours par défaut. Elle signale tout changement de l'état de la tâche **num_t**.

gcore num_p

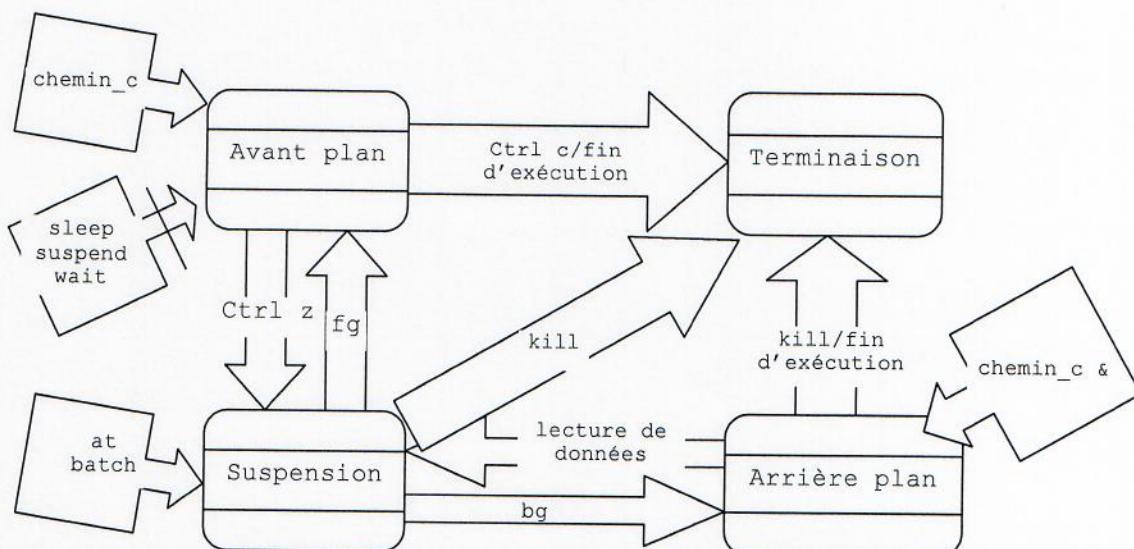
génère un fichier de nom **core.num_p**, contenant l'image du processus, suspendu ou en exécution en arrière plan, **num_p**.

Ctrl d

envoie un **eof** (End Off File) au fichier d'entrée standard. Elle termine l'interpréteur de commandes courant (ex. tcsh, bash, ...) ou la commande courante qui fait une lecture (ex. tee, mail, ...).

Ctrl o

accélère, ou bien décélère, l'affichage des résultats d'une commande.



5.4. Historique

La notion d'historique des commandes exécutées est introduite par le **C Shell**. Elle est reprise par les autres Shells.

history [num]

donne la liste des **num** dernières commandes de l'utilisateur courant, durant la session courante, sous le terminal courant. La valeur de **num** est précisée dans le fichier **.login** par la variable **C Shell history** par défaut.

! : le caractère spécial de substitution des commandes de l'historique des commandes. Il peut être changé à partir de la variable d'environnement **histchars**.

!nom

affiche la commande la plus récente dont le nom commence par le mot **nom** dans l'historique des commandes si elle existe.

?nom?

affiche la commande la plus récente qui contient au moins une occurrence du mot **nom** dans l'historique des commandes si elle existe.

!num

affiche la commande numéro **num** dans l'historique des commandes.

!-num

affiche la commande numéro **num** dans l'historique des commandes, en partant de la dernière commande.

!!

affiche la dernière commande exécutée.



se déplace vers le haut dans la liste des commandes de l'historique.



affiche la dernière commande exécutée.

← et →

se déplace dans la commande avec la possibilité d'utiliser les commandes suivantes, comme celles utilisées dans l'éditeur de texte **vi** :

x : efface le caractère courant ;

r : remplace le caractère courant par le caractère suivant tapé ;

i : passe en mode insertion ;

a : passe en mode insertion en déplaçant le curseur après le caractère courant.

5.5. Alias de commandes

La notion d'alias des commandes est introduite par le **C Shell**. Elle est reprise par les autres Shells.

alias identif chemin_c

(Dans certaines versions Linux : **alias identif = chemin_c** sans espace ni avant ni après = et les cotes)

donne une nouvelle identification au nom de la commande **chemin_c** qui est **identif**. **chemin_c** reste toujours valide. Cette nouvelle identification est valable uniquement durant le **C Shell** courant. La récursivité et l'utilisation d'un alias dans la définition d'un autre sont interdites. La commande **chemin_c** peut ne pas exister.

\!:0 : le nom de la commande ;
\!:num : le paramètre ou l'option numéro **num**, en commençant par le numéro 1;
\!:num_1-num_2 : les paramètres ou les options de numéro **num_1** au numéro **num_2** ;
\!^ : le premier paramètre ou la première option ;
\!\$: le dernier paramètre ou la première option ;
\!* : la liste des paramètres et des options.

Exemples :

```

alias rm rm -i
alias np 'ps -aux | wc -l'
alias vt100 set term = vt100
alias cd 'cd\!:1 ; pwd ; set n='ls -l | wc -l' ; echo Il y a $n[1] fichier(s).'
alias cd 'cd\!:1 ; pwd ; set n='ls | wc -w' ; echo Il y a $n[1] fichier(s).'
alias cd "cd ; dir"
alias cd 'cd \!:1 ; set prompt = `pwd`">"'
alias lm 'ls -l \!* | more'
alias dir 'ls -l \!:1-3'
alias dir "ls -l \!:1-3"
dir a b c d      →      ls -l a b c
alias dir ls
  dir /          →      ls /
  dir a b c d    →      ls a b c d

```

alias *identif*

affiche l'alias **identif** avec sa commande correspondante.

alias

affiche la liste des alias avec leurs commandes correspondantes.

unalias *identif*

supprime l'identification de **identif**.

5.6. Redirection des entrées-sorties

Lors de l'exécution d'un programme, il prend ses données du fichier d'entrée, envoi ses résultats sur le fichier de sortie et envoi les messages d'erreur sur le fichier de sortie erreur. En général, le clavier et l'écran du terminal sont respectivement le fichier d'entrée et le fichier de sortie standard et de sortie erreur standard pour un programme.

Exemples :

ls -l	→ le résultat de la commande est sur l'écran.
mail omar	→ l'entrée de la lettre à partir du clavier.
lks -l	→ le message d'erreur est sur l'écran, si lks n'est pas une commande.

UNIX permet de rediriger les entrées, les sorties et les sorties erreurs d'un fichier exécutable ou d'une commande à travers autres fichiers que les fichiers standards. C'est le remplacement des fichiers des entrées-sorties standards.

Les descripteurs des fichiers standards d'entrée-sortie sont :

- 0 : l'entrée de données (**stdin**) ;
- 1 : la sortie de résultats (**stdout**) ;
- 2 : la sortie d'erreurs (**stderr**).

chemin_x < chemin_f_i

exécute le fichier ***chemin_x*** en prenant ses entrées du fichier ***chemin_f_i***. Le fichier ***chemin_f_i*** doit exister auparavant sinon un message d'erreur est affiché, avant même l'exécution de ***chemin_x***.

chemin_x << car

exécute le fichier ***chemin_x*** en prenant ses entrées dans la séquence de lignes qui suivent jusqu'à la ligne composée par ***car***.

Exemple :

cat << @

...

@

lecture à partir du clavier jusqu'à la rencontre de la ligne composée par le caractère **@** au début d'une ligne.

tee f << @

...

@

chemin_x > chemin_f_o

exécute le fichier ***chemin_x*** en mettant ses sorties vers le fichier ***chemin_f_o***. Si le fichier ***chemin_f_o*** existe auparavant il est détruit tout d'abord sinon il est créé, si la variable **noclobber** n'est pas positionnée.

Exemples :

cat f1.txt > f2.txt

vide **f2.txt** même si **f1.txt** est vide ou n'existe pas.

cat > f.txt

...

Ctrl d

crée un fichier **f.txt** contenant tout ce qui est tapé jusqu'à **Ctrl d**.

cat f.txt > /dev/tty3

transmet le fichier **f.txt** au terminale numéro 3.

cat f.txt > /dev/lp

imprime le fichier **f.txt**.

chemin_x > chemin_f_o

exécute le fichier *chemin_x* en mettant ses sorties vers la fin du fichier *chemin_f_o*. Si le fichier *chemin_f_o* n'existe pas auparavant il est créé tout d'abord si la variable **noclobber** n'est pas positionnée.

Exemples :

```
cat f1.txt f2.txt > f3.txt  
cat f1.txt f2.txt >> f3.txt
```

Remarque :

```
cat f1.txt f2.txt > f1.txt
```

La création ou la recréation d'un fichier se fait avant l'exécution de la commande. Si **cat** est utilisée pour la redirection des entrées-sorties, la redirection doit être faite dans un autre fichier que ceux listés sinon le fichier est détruit.

chemin_x >& chemin_f_o

(C Shell)

exécute le fichier *chemin_x* en mettant ses sorties et les sorties erreurs vers le fichier *chemin_f_o*. Si le fichier *chemin_f_o* existe auparavant il est détruit tout d'abord si la variable **noclobber** n'est pas positionnée.

chemin_x >>& chemin_f_o

(C Shell)

exécute le fichier *chemin_x* en mettant ses sorties et les sorties erreurs vers la fin du fichier *chemin_f_o*.

Exemples :

```
at 2400 (racine.exe <donnee.txt) > resultat.txt  
(pwd ; date) > f.txt  
(pwd ; dat) > f.txt  
(pwd ; dat) >& f.txt
```

(chemin_x) 2> chemin_f_o

(Bourne Shell)

exécute le fichier *chemin_x* en mettant les messages d'erreurs dans le fichier *chemin_f_o*. Si le fichier *chemin_f_o* existe auparavant il est détruit tout d'abord si la variable **noclobber** n'est pas positionnée. Le fichier *chemin_f_o* est vidé même s'il n'y a pas de messages d'erreur.

(chemin_x) 2>> chemin_f_o

(Bourne Shell)

exécute le fichier *chemin_x* en mettant les sorties erreurs vers la fin du fichier *chemin_f_o*.

(chemin_x > chemin_f_o) 2> chemin_f_e

(Bourne Shell)

exécute le fichier *chemin_x* en mettant les sorties résultats dans le fichier *chemin_f_o* et les sorties erreurs dans le fichier *chemin_f_e*.

chemin_x >&1

exécute le fichier ***chemin_x*** en mettant ses sorties résultats vers le fichier de sortie déjà ouvert.

Remarque :

(***chemin_c > chemin_f***) 2>> ***chemin_f*** en Bourne Shell, est équivalente à ***chemin_c >& chemin_f*** en C Shell.

Exemples :

```
(ls -l f1 f2 > f3) 2> f4  
(pwd ; date) > f.txt  
(pwd ; date) 2> f.txt  
(pwd ; dat) > f.txt  
(pwd ; dat) 2> f.txt
```

Remarques :

>! et >&! (C Shell) >| et >&| (Bourne Shell) : obligent la destruction de fichier existant même si la variable **noclobber** est positionnée.

>>! et >>&! (C Shell) >>| et >>&| (Bourne Shell) : obligent la création de fichier inexistant même si la variable **noclobber** est positionnée.

Il n'y a pas d'espace entre 2 et >, ni entre > et &, ni entre & et !, ...

Remarques :

Il y a une différence majeure entre :

- Les commandes qui ont besoin obligatoirement d'une donnée comme paramètre (ex. **rm f.txt**) (ex. **racine.exe 5** avec l'utilisation de main(argc,argv,envp)) ;
- Les commandes qui ont besoin obligatoirement d'une donnée comme une entrée (**tee f.txt** puis **bien Ctrl d**) (ex. **racine.exe** puis **5** ou bien **racine.exe < f.txt** avec l'utilisation de **scanf(...)**) ;
- Les commandes qui ont besoin obligatoirement d'une donnée comme paramètre (ex. **wc -l f.txt**) ou comme entrée (ex. **wc -l < f.txt**) (ex. **ls -l | wc -l**) (ex. **wc -l** puis ... puis **Ctrl d**) ;
- Les commandes qui ont besoin éventuellement d'une donnée comme paramètre (ex. **ls -l r** ou bien **ls -l**).

5.7. Tube entre les commandes

UNIX permet de construire des chaînes de programmes appelées des **tubes (pipes)**, où les sorties d'un programme sont les entrées du programme suivant. Les tubes permettent une exécution concurrente de processus.

Le tube est un tuyau, considéré comme un fichier virtuel, qui permet la communication entre des processus. Les deux processus, producteur et consommateur, forment un tube.

chemin_x_1 | chemin_x_2 [| chemin_x_3 ...]

les sorties de la commande ***chemin_x_1*** seront les entrées de la commande ***chemin_x_2*** dont ses sorties seront les entrées de la commande ***chemin_x_3*** et ainsi de suite.

Le père du processus ***chemin_x_2*** est ***chemin_x_1*** et le père du processus ***chemin_x_3*** est ***chemin_x_2***.

chemin_x_1 |& chemin_x_2

les sorties résultats et les sorties erreurs de la commande ***chemin_x_1*** seront les entrées de la commande ***chemin_x_2***.

Exemples :

ls | wc -w

ls -l | wc -l

affiche le nombre de fils du répertoire courant (plus la ligne Total !).

ls -l > f ; wc -l f ; rm f

affiche le nombre de fils du répertoire courant plus le fichier **f**, sauf s'il existe déjà il sera remplacé.

ls -l | sort -n +4 > f

f contient le contenu du répertoire courant trié par taille.

ls -l | tee f.txt

date | tee -a f.txt

ls -l | tail -5

ls -l | head -5

ls -l | grep '^->'

ps -aux | wc -l

ps | more

cat f.txt | more

ls -lt | head -1

ls -lrt | tail -1

affiche la description du fichier le plus récent, le dernier fichier créé.

cmd | kill -9

génère une erreur.

set q = `cmd`

kill -9 \$q

6. Commandes étendues

6.1. Impression de fichiers

pr [-t] [chemin_f_1 [chemin_f_2 ...]]

imprime les fichiers **chemin_f_1**, **chemin_f_2**, ... d'une façon séquentielle, imprime l'écran par défaut. Le système met les fichiers à imprimer dans une queue en affectant un numéro d'ordre à chacun.

t : enlève l'en-tête.

lpr [-#num] chemin_f

imprime le fichier **chemin_f num** fois, une seule fois par défaut.

lpq [-l] [nom_u]

donne la liste des fichiers en queue de l'imprimante, uniquement de l'utilisateur **nom_u** par défaut. S'il n'y a pas de fichier en queue le message suivant est affiché : **no entries**.

-l : affiche l'état détaillé des fichiers en attente d'impression.

lprm [-] [num]

élimine le fichier numéro **num** de la queue de l'imprimante, le fichier en cours par défaut. Si ce n'est pas un fichier de l'utilisateur il affiche : **Permission denied**.

- : tous les fichiers de l'utilisateur de la queue.

6.2. Aide électronique

chemin_c

en général, donne la syntaxe de la commande **chemin_c** si elle nécessite des paramètres sinon elle est exécutée.

Exemple :

more

usage : more [-dfln] [+linenum | +/pattern] name1 name2 ...

help

(Solaris : /usr/ccs/bin/help)

affiche une explication générale sur les commandes existantes.

chemin_c --help**(Linux)**

affiche une explication générale sur la commande ***chemin_c***, si cette option est prévue dans la commande.

info chemin_c**(Linux)**

affiche une explication restreinte de la commande ***chemin_c***.

man [num] nom_c

affiche une explication détaillée de la commande ***nom_c*** du chapitre numéro ***num*** du manuel.

C'est la documentation en ligne des commandes d'UNIX. ***nom_c*** doit être une commande externe, c'est-à-dire qui correspond à un fichier propre.

Exemples :**man more****man tcsh****man bash****Remarque :**

Pour les commandes internes (ex. cd, fg, ...), c'est-à-dire qui ne correspondent à aucun fichier propre, leurs aides électroniques sont regroupées dans l'aide de leur interpréteur de commandes correspondant (ex. tcsh, bash, ...).

whatis chemin_c**(Solaris : ne donne rien)**

donne une explication sommaire de la commande ***chemin_c***, la première ligne du manuel.

Exemple :**whatis more**

more, page(1) -display file data at your terminal

which nom_c

cherche l'emplacement dans le système de fichiers (**Solaris** : dans les chemins du **path**) du fichier qui correspond à la commande ***nom_c***.

Exemple :**which more**

/usr/ucb/more

whereis nom**(Solaris : /usr/ucb/whereis)**

donne la liste des répertoires où se trouvent tous les fichiers correspondants aux commandes qui ont comme nom ***nom***, avec n'importe quelle extension.

Exemple :

whereis more

more : /lib/more.help /usr/ucb/more /usr/lib/more.help /usr/man/man1/more.1

6.3. État des utilisateurs

ID

(Linux : **id**)

affiche le UID de l'utilisateur. Dans certaines versions, c'est une variable et non pas une commande.

useradd

permet d'ajouter un compte d'utilisateur par le super-utilisateur.

adduser

(Linux)

permet d'ajouter un compte d'utilisateur par le super-utilisateur, avec plus d'options, en appelant **useradd**.

groupadd

permet d'ajouter un groupe d'utilisateurs par le super-utilisateur.

addgroup

(Linux)

permet d'ajouter un groupe d'utilisateurs par le super-utilisateur, avec plus d'options, en appelant **groupadd**.

dispuid

(Solaris)

affiche la liste des utilisateurs du système en une seule colonne.

listusers [-g nom_g_1 , nom_g_2 , ...] [-u nom_u_1 , nom_u_2 , ...]

(Solaris **listusers [-g nom_g_1 , nom_g_2 , ...] [-l nom_u_1 , nom_u_2 , ...]**)

affiche la liste des utilisateurs connectés (Solaris même ceux non connectés) appartenant aux groupes **nom_g_1**, **nom_g_2**, ... et de nom **nom_u_1**, **nom_u_2**, ... en deux colonnes : nom de connexion puis liste d'informations correspondant.

users

(Solaris : **/usr/ucb/users** : donne la liste des propriétaires des processus en court)

donne la liste des utilisateurs connectés. Plusieurs noms dans la même ligne.

Exemple :

users

omar ali root

dispid

(Solaris)

affiche la liste des groupes des utilisateurs du système en une seule colonne.

groups [nom_u]

(Solaris : /usr/ucb/groups [nom_u])

affiche la liste des groupes auxquels l'utilisateur **nom_u** est membre, de l'utilisateur courant par défaut.**who**

donne la liste des utilisateurs connectés avec pour chaque utilisateur : le nom de connexion, le numéro du terminal et la date de connexion.

Exemple :**who**

```
omar  tty02  Oct 12 19 :45
ali   tty16  Oct 12 11 :02 (SERV2)
```

who i me**who am i**(Solaris comme **who**)

donne le nom de la machine, le nom de connexion, le numéro du terminal et la date de connexion de l'utilisateur courant connecté.

Exemple :**who i me**

```
science!omar  tty02  Oct 12 19 :45
```

w [nom_u]donne la liste des informations suivantes pour l'utilisateur **nom_u** : le nom de connexion, le numéro du terminal, l'heure de connexion, la durée écoulée depuis la dernière frappe en minutes, le temps consommé de l'unité centrale depuis la connexion, le temps consommé de l'unité centrale du processus courant et le nom de la commande courante ou -, ou -csh pour l'interpréteur de commandes **Shell**. Elle donne cette liste d'informations pour chaque utilisateur connecté par défaut.**Exemple :****w**

User	tty	from	login@	idle	JCPU	PCPU	What
omar	tty02		7 :45pm		29	5	-csh
ali	tty18	SERV2	7 :45am	3	49	15	vi f.txt

finger [-l] [nom_u]donne des informations supplémentaires de l'utilisateur **nom_u** de tous les utilisateurs connectés par défaut.**-l** : les informations détaillées (Login Name TTY Idle When Where).

chfn

(Solaris : non)

permet de changer ou de mettre des informations supplémentaires sur l'utilisateur en cours, en demandant le mot de passe.

logname

donne le nom de connexion de l'utilisateur du terminal courant.

Exemple :**logname**
omar**last [nom_u] [num]**

donne la liste de toutes les sessions de travail faites sur le système : nom de connexion, numéro du terminal, date de début de connexion et la date de la fin ou le message : **still logged in** s'il est encore connecté, triée par ordre décroissant de leur date de début de connexion, de l'utilisateur de nom de connexion **nom_u** et dans le terminal numéro **num**. Elle donne cette liste d'informations pour tous les utilisateurs par défaut, depuis la réinitialisation de la machine.

tty

affiche le chemin absolu du fichier spécial correspondant au terminal de connexion.

Exemple :**tty**
/dev/tty04**Exemple :**

cat f.txt > /dev/tty03

6.4. Gestion de sessions de travail

login nom_u

permet de terminer la session immédiatement et de se connecter sous le nom de connexion **nom_u**. (Solaris : **nom_u** ne doit pas être root et l'interpréteur de commandes doit être le **Shell** de base lui-même et non pas un fils) (Linux : valable uniquement par le super utilisateur).

su [nom_u]

se connecte sous l'utilisateur de nom de connexion **nom_u** sans terminer la session courante, tout en demandant le mot de passe et ne change ni l'environnement ni le répertoire courant mais appelle un nouveau **Shell**. **Ctrl d** permet de revenir à la session précédente. Elle permet de se connecter comme administrateur du système par défaut.

sudo -V [-u nom_u | #uid] chemin_cexécute la commande **chemin_c** sous un autre utilisateur spécifié par **nom_u** ou **uid**.

logout

déconnecte du système si aucun processus **Shell**, autre que celui de connexion, n'est empilé, sinon affiche le message suivant : **Not login shell**. Il suffit de faire **logout** une deuxième fois pour terminer la session.

script [-a] [chemin_f]

met une image de tout ce qui se passe dans le terminal de l'utilisateur dans le fichier **chemin_f** jusqu'à l'exécution de la commande **exit**, dans le fichier **typescript** du répertoire courant par défaut. Évidemment, il faut sortir de script pour pouvoir exploiter le fichier. Le mot de passe ne s'enregistre plus dans le fichier !

a : l'ajoute à la fin du fichier.

6.5. Lien de fichiers

Un fichier physique peut avoir plusieurs références logiques, ou points d'entrées, par l'intermédiaire de lien de fichiers. Cela permet d'avoir des chemins de fichiers restreints. Toute modification portée sur le fichier d'origine sera lisible dans son image et inversement. Le partage d'un fichier entre un groupe d'utilisateurs nécessite la permission d'écriture pour le groupe.

Il y a deux types de lien de fichiers :

- le **lien dur** ou physique : lien vers le numéro d'inode d'un fichier ;
- le **lien symbolique** ou logique : lien vers le chemin d'un fichier.

Le but de lien (**link**) de fichiers est d'avoir un fichier partagé par plusieurs utilisateurs (cas de lien dur) ou une référence simplifiée à un fichier (cas de lien symbolique). L'avantage des liens est de ne pas avoir une duplication du même fichier par mesure de gain de l'espace mémoire et une cohérence du contenu du fichier et de diminuer le risque de destruction involontaire du fichier.

En général, le lien symbolique est un lien entre des fichiers appartenant au même système de fichiers. Par contre, le lien dur est un lien entre des fichiers appartenant à des systèmes de fichiers différents.

Dans le cas de lien dur, la création d'un nouveau lien avec un fichier existant augmente le nombre de liens de 1. Les deux fichiers portent la même description, c'est-à-dire le même numéro d'inode, même nombre de liens après son incrémentation de 1, même taille, même date, etc. La destruction d'un fichier de lien décrémente le nombre de liens de 1. Le fichier n'est réellement supprimé que lorsque le nombre de liens devient nul. Il n'y a pas de distinction entre le fichier d'origine et le lien dur.

Dans le cas de lien symbolique, la création d'un nouveau lien avec un fichier existant crée un fichier de type lien avec un nouveau numéro d'inode, la date de création du lien et qui contient simplement le chemin d'accès du fichier avec une taille égale au nombre de caractères du chemin du fichier origine. Dans la description, il y a la mention 1 dans la partie type du champ de la protection et le chemin d'accès du fichier origine à la fin précédé du symbole ->. Il y a une distinction entre le fichier d'origine et le lien symbolique.

ln *chemin_f_o* *chemin_f_r*

(Linux : **cp -l *chemin_f_o* *chemin_f_r***)

crée un lien dur du fichier ***chemin_f_o*** dans le répertoire ***chemin_f_r*** avec le même nom, ou dans le répertoire courant avec le nom de fichier ***chemin_f_r***. ***chemin_f_o*** doit exister auparavant.

ln *chemin_r_o* *chemin_r*

crée un lien dur du répertoire ***chemin_r_o*** dans le répertoire ***chemin_r***. ***chemin_r_o*** doit exister auparavant. (*non Linux*)

ln -s *chemin_f_r_o* *chemin_f_r*

(Linux : **cp -s *chemin_f_r_o* *chemin_f_r***)

crée un lien symbolique du fichier ou du répertoire ***chemin_f_r_o*** dans le répertoire ***chemin_f_r*** avec le même nom, ou dans le répertoire courant avec le nom ***chemin_f_r***. ***chemin_f_r_o*** peut ne pas exister auparavant. ***chemin_f_r_o*** doit être un chemin absolu.

Exemples :

ls -li f1.p

534 -rwxrwxrwx 1 omar 231 Oct 25 16 :40 f1.p

1 ln f1.p f2.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f1.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f2.p

2 ln f2.p f3.p

ou bien **ln f1.p f3.p**

534 -rwxrwxrwx 3 omar 231 Oct 25 16 :40 f1.p

534 -rwxrwxrwx 3 omar 231 Oct 25 16 :40 f2.p

534 -rwxrwxrwx 3 omar 231 Oct 25 16 :40 f3.p

3 rm f2.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f1.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f3.p

4 ln -s f2.p f4.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f1.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f3.p

537 lrwxrwxrwx 1 omar 4 Oct 27 6 :40 f4.p -> f2.p

Si l'utilisateur a déjà détruit, ou déplacé, **f2.p** auparavant, il ne peut pas accéder à **f4.p** même si **f2.p** avait un lien dur avec **f3.p**.

Si l'utilisateur détruit **f2.p** après, il ne peut plus accéder à **f4.p** même si **f2.p** avait un lien dur avec **f3.p**.

5 ln -s f4.p f5.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f1.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f3.p

537 lrwxrwxrwx 1 omar 4 Oct 27 6 :40 f4.p -> f2.p

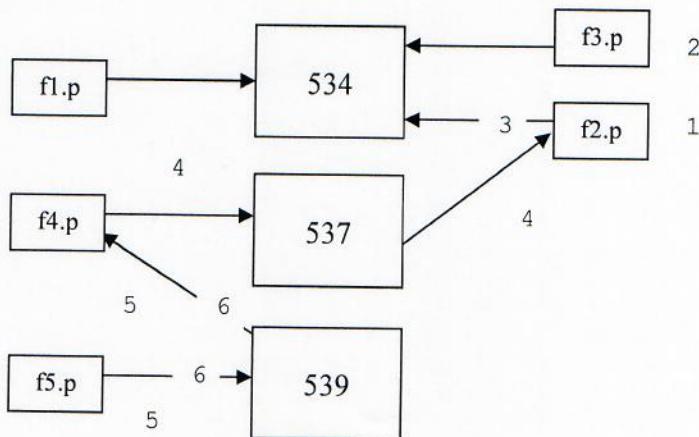
539 lrwxrwxrwx 1 omar 4 Oct 29 9 :40 f5.p -> f4.p

6 rm f5.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f1.p

534 -rwxrwxrwx 2 omar 231 Oct 25 16 :40 f3.p

537 lrwxrwxrwx 1 omar 4 Oct 27 6 :40 f4.p -> f2.p



6.6. Divers

date

donne la date et l'heure courante.

cal [num_1 [num_2]]

donne le calendrier du mois **num_2** de l'année **num_1**, de toute l'année **num_1** par défaut, du mois courant de l'année courante par défaut.

bc

appelle une calculatrice arithmétique simple. Pour sortir, il faut taper **Ctrl d**.

stty [-a]

(Solaris : non)

donne la vitesse de connexion.

-a : toutes les informations.

Exemple :

stty

new tty, speed 9600 baud ; -tabs ff1 crtbs crterase prterase brk = ^[

stty all

donne la liste des touches de contrôles utilisées.

stty everything

donne toutes les informations sur le terminal en cours.

setxkbmap fr|us

change la carte du clavier QWERTY|AZERTY.

Loadkeys fr|us

7. Communication électronique

UNIX est un système d'exploitation très convivial. Il permet des communications faciles et puissantes entre les utilisateurs.

7.1. Courrier

UNIX permet d'établir une communication indirecte entre les utilisateurs en leur permettant d'envoyer des messages, ou des lettres, électroniques aux autres utilisateurs. Pour envoyer une lettre à un ou plusieurs utilisateurs, et à soi même pour garder une trace, il suffit de connaître leurs noms de connexion de ces correspondants. Le système envoie automatiquement des messages dans le cas de certaines erreurs ou de certains problèmes du système. Un message est composé de deux parties : l'en-tête et le corps du message.

mail nom_u_1 [nom_u_2 ...]

envoie un message aux utilisateurs de nom de connexion **nom_u_1, nom_u_2, ...**.

Un ensemble de questions à qui l'utilisateur doit répondre sont affichées :

Subject: [nom]

sujet de la lettre est **nom**.

Cc: [nom_u_1 [nom_u_2] ...]

(Carbon Copy)

redirige la lettre aux autres utilisateurs **nom_u_1, nom_u_2, ...**.

Puis l'utilisateur tape librement sa lettre mais avec un certain nombre de limitations. Tant qu'il est dans une ligne il peut la modifier. Une fois il appuie sur **Return**, il ne peut plus accéder à cette ligne ni aux lignes précédentes. L'utilisateur ne peut jamais commencer une ligne par un . ou **Ctrl d**. Pour remédier à ces contraintes, l'utilisateur peut taper sa lettre dans un fichier via un éditeur de texte puis l'envoyer directement.

Ctrl c Ctrl c

permet d'annuler la lettre en cours d'édition. Elle est sauvegardée dans le fichier **dead.letter** dans le répertoire courant ou le répertoire de base.

Les commandes internes, au début de la ligne, sont :

~r chemin_f : insert le contenu du fichier **chemin_f** à la position courante ;

~! : appelle le **Shell** ;

~v : appelle l'éditeur de texte **vi** ;

~? : liste les commandes de ~ ;

~p : liste le message tapé ;

~w chemin_f : sauvegarde le message dans le fichier **chemin_f** ;

s chaîne : positionne le sujet de la lettre qui sera **chaîne** ;

c nom_u_1 [nom_u_2 ...] : redirige la lettre aux autres utilisateurs **nom_u_1, nom_u_2,...**.

Si **nom_u** n'existe pas il affiche : **nom_u User unknown**

Lorsque la lettre est envoyée le système émet le message suivant à l'attention du ou des destinataires: **You have mail**

mail

affiche la liste des messages reçus. Si aucun courrier n'est arrivé le message suivant s'affiche : **No mail for nom_u**

Si l'utilisateur a reçu un courrier le message suivant s'affiche :

Mail version 2.18 5/19/83. Type ? for help.

"/usr/spool/mail/omar": 1 message 1 new

>N 1 omar Mon Dec 3 11:11 14/154 "sujet"

N 2 MAILER-DAEMON Mon Dec 3 11:11 30/538 "Returned mail:User unknown"

> : indique le message courant.

Puis la prompt du mail & s'affiche :

& [Space d r R x q s chemin_fz help ? !]

Space : affiche le message comme suite :

Message 1:

From: nom de l'expéditeur et date de réception ;

Date: date de l'envoi ;

From: nom complet de l'expéditeur ;

To: nom du récepteur ;

Subject: sujet du message ;

Cc: liste des autres utilisateurs qui ont reçu aussi ce message ;

Status: état du message: non encore lu, déjà lu, etc.

Content-length: taille du message en nombre de caractères, chaque retour chariot compte pour 2 caractères ;

.

.

.

&

d num : détruit le message numéro **num** du **spool** ;
u num : reprend le message numéro **num** déjà détruit ;
r num : répond au message numéro **num** à l'expéditeur et aux autres utilisateurs qui l'ont reçu ;
R num : répond au message numéro **num** à l'expéditeur uniquement ;
x : sort sans changer l'état du **mail** ;
Ctrl d ou **q** : quitte **mail** en laissant les messages dans le **spool** puis affiche :
 Held **num** messages in /usr/spool/mail/omar
s [num] chemin_f : sauvegarde le message numéro **num** à la fin du fichier **chemin_f**, le message courant par défaut ;
t num : liste le message numéro **num** ;
! chemin_c : exécute la commande **chemin_c** sous le **Shell** ;
? ou help : liste ces commandes ;
: affiche le nombre de messages ;
Retour chariot : affiche le message suivant ;
w[chemin_f] : sauvegarde le message courant dans le fichier **chemin_f**, à la fin du fichier **mbox** par défaut.

Si l'utilisateur n'a pas détruit les messages, lorsqu'il quitte **mail**, les messages sont sauvegardés à la fin du fichier **mbox** qui doit se trouver dans le répertoire de base de l'utilisateur.

Exemples :

mail 'f1' < f2

envoie le message qui se trouve dans le fichier **f2** à la liste des utilisateurs qui se trouvent dans le fichier **f1**.

mail -u omar -f mes.txt -s rendez-vous
(Solaris : non)

envoie le message qui se trouve dans le fichier **mes.txt** à l'utilisateur **omar** sous le sujet **rendez-vous**.

En Solarise :

-t nom_u_1 [nom_u_2 ...]

envoie le message aux utilisateurs **nom_u_1, nom_u_2, ...**

-w

envoie le message au fur et à mesure de sa saisie, jusqu'à la fin de la saisie par défaut.

-m [text|binary|multipart]

spécifie le type du contenu du message.

-f [chemin_f]

sauvegarde le message dans le fichier **chemin_f**, dans le fichier **mailfile** par défaut.

-F "nom_u_1[, nom_u_2, ...]"

répond aux utilisateurs **nom_u_1, nom_u_2, ...**".

7.2. Dialogue

write *nom_u* [*num*]

permet d'envoyer un message à un utilisateur de nom de connexion ***nom_u*** et qui travaille sous le terminal numéro ***num***, s'il est connecté et s'il permet la réception de messages.

Le message tapé sur le clavier est envoyé au fur et à mesure, précédé de : **Message from *nom_u* tty *num* at *num_1***. Il est terminé dès que l'utilisateur tape **Ctrl d**, puis l'indication **EOT** est envoyé pour marquer la fin du message.

Si le destinataire ne donne pas la permission de réception de messages le message suivant s'affiche chez l'expéditeur : **Your party is refusing messages**.

Par convention :

oo (ove rand out) pour terminer définitivement le message.

o (over) pour terminer momentanément le message et attendre une réponse du destinataire.

talk *nom_m!nom_u* [*num*]**

permet d'ouvrir un dialogue écrit avec l'utilisateur de nom de connexion ***nom_u*** dans la machine ***nom_m*** et qui travaille sous le terminal numéro ***num***, s'il est connecté et s'il permet la réception de messages.

Au début, le message suivant apparaît chez le destinataire : **Talk from *nom_m!**nom_u* tty *num* at *num***

Le destinataire doit répondre par la commande : **talk *nom_u***

Les écrans des deux utilisateurs se divisent en deux parties : la partie supérieure de l'écran pour l'émission et la partie inférieure pour la réception. La communication continue entre eux en changeant des messages jusqu'à ce que l'un des deux tape **Ctrl c** et le message suivant s'affiche.

Ringing your party again
Connection closing. Existing
Connection established

Si le destinataire ne donne pas la permission de réception de messages le message suivant s'affiche chez l'expéditeur : **Permission denied**.

! : au début d'une ligne pour appeler le **Shell**.

!chemin_c : exécute la commande **chemin_c** sans sortir du message.

mesg

donne le statut de la permission de réception de messages dans l'écran.

Exemple :

mesg

retourne **is y** ou **is n**.

mesg -y

(Linux)

mesg y

donne la permission de réception de messages dans l'écran.

mesg -n

(Linux)

mesg n

suspend la permission de réception de messages dans l'écran.

wall

(Solaris : /usr/sbin/wall)

permet d'envoyer un message à tous les utilisateurs connectés. Tout le texte tapé sera envoyé jusqu'à **Ctrl d**.

Le message suivant s'affiche : **Broadcast Message from nom_m!nom_u(tty) at hh:mm** ou **Cannot send to nom_u**

7.3. Agenda

calendar [-]

consulte le fichier **calendar**, s'il existe dans le répertoire de base, puis affiche les lignes qui portent la date courante ou le lendemain.

Exemples :

12/27 Rendez-vous avec le dentiste.

Le 27 décembre.

***/3 Payer le loyer.**

Le 3 de n'importe quel mois.

Tous les minuits, le système consulte tous les fichiers **calendar** de tous les utilisateurs et envoie un mail aux utilisateurs dont au moins une ligne de leur **calendar** correspond à la date courante ou à celle du lendemain.

Remarque :

Aug. 5, august 5 et 8/5 sont équivalents.

8. Sécurité de fichiers

Le mot de passe permet de protéger le système et en particulier les fichiers d'un utilisateur contre les non-utilisateurs du système. Il permet au système de savoir que l'utilisateur est bien celui qu'il prétend être. La protection des fichiers permet de les protéger contre l'utilisateur lui-même et contre les autres utilisateurs. À la création d'un fichier, le système affecte le **UID** et le **GID** de l'utilisateur au fichier créé et lui attribue la permission par défaut spécifiée par la variable d'environnement **umask**.

8.1. Mot de passe

passwd

(Linux : **passwd [nom_u]**, par le super-utilisateur uniquement)

change ou installe un mot de passe en demandant d'abord l'ancien mot de passe (pour s'assurer que celui qui demande l'exécution de cette commande est bien l'utilisateur lui-même) puis le nouveau 2 fois par sécurité. Le mot de passe doit être un mot composé de lettres, de chiffres et de certains symboles (ex. d'au moins 6 caractères et d'au plus 16 caractères). L'utilisateur doit répondre aux questions, pendant que la fonction **echo** du terminal est supprimée.

Exemple :

passwd

Echanging password for omar

Old password:

Enter new password:

Verify:

keylogin

installe une clé de sécurité de cryptage demandée en mode interactif sans écho. La clé de sécurité est dans le fichier **/etc/publickey**.

keylogout

désinstalle la clé de cryptage.

8.2. Modes de protection

chmod [ugoa] [+--] [rwx] chemin_f_r_1 [chemin_f_r_2 ...] (CHange MODe)

permet le changement des permissions des fichiers ou des répertoires **chemin_f_r_1**, **chemin_f_r_2**, ..., si l'utilisateur est propriétaire de ces fichiers ou ces répertoires.

u : (User) l'utilisateur propriétaire lui-même ;

g : (Group) les membres du groupe principal du propriétaire ;

o : (Other) les autres utilisateurs ;

a : (All) tous les utilisateurs, ce qui est par défaut.

- + : ajouter une permission à ce qui existe déjà, ce qui est par défaut ;
- : retirer une permission de ce qui existe déjà ;
- = : limiter la permission à ce qui est spécifiée après, pour une permission absolue.

Dans le cas d'un fichier, on a :

- r** : (**R**ead) l'utilisateur peut lire le fichier pour une consultation s'il a la permission d'exécution du répertoire qui le contient ;
- : l'utilisateur ne peut ni le lire, ni avoir une copie mais il peut changer son nom s'il est propriétaire ;
- w** : (**W**rite) l'utilisateur peut écrire sur le fichier s'il a la permission d'exécution du répertoire qui le contient, ce qui est par défaut ;
- : l'utilisateur ne peut ni changer son contenu ni le supprimer ;
- x** : (**eXecution**) l'utilisateur peut exécuter le fichier s'il est exécutable ;
- : l'utilisateur ne peut pas l'exécuter en général.

Dans le cas d'un répertoire, on a :

- r** : l'utilisateur peut lire ou consulter le contenu du répertoire, par **ls** par exemple ;
- w** : l'utilisateur peut créer, renommer ou supprimer des fichiers et des sous répertoires du répertoire, sans se préoccuper des permissions propres à ces fichiers et à ces sous répertoires, ce qui est par défaut ;
- x** : l'utilisateur peut faire figurer ce répertoire dans un chemin comme paramètre d'une commande, se déplacer vers ce répertoire et lire et obtenir les informations détaillées de la description des fichiers et des sous répertoires qu'il contient, par **ls -l** par exemple (**L**inux : possible même sans le **x**).

Il faut avoir la permission d'écriture pour modifier un fichier qui a un lien avec un fichier d'un autre utilisateur.

Exemple :

Par u2 :

ls -li ~u2

9220 drwxr-xr-x 2 u2 512 Oct 23 2:20 r2

ls -li ~u2/r2

9225 -rw-r--r-- 1 u2 15 Oct 23 2:25 f2

Par u1 :

ls -li ~u1

9120 drwx----- 2 u1 512 Oct 13 2:20 r1

ln ~u2/r2/f2 ~u1/r1/f1

ls -li ~u1/r1

9225 -rw-r--r-- 2 u2 15 Oct 23 2:30 f1

ln -s ~u2/r2/f2 ~u1/r1/f3

9240 lrw-r--r-- 1 u1 32 Oct 23 2:40 f3 -> /usr/users/informatique/u2/r2/f2
rm ~u1/r1/f3

destruction de f3.

rm ~u1/r1/f1

rm: override protection 644 for f1?

y → destruction de f1.

n → pas de destruction de f1.

vi f1

ouverture en lecture seulement.

Exemples :

chmod u+x f.txt

chmod o=x f.txt

chmod +x f.txt ↔ chmod a+x f.txt

chmod g= f.txt ↔ chmod g-rwx,g+w f.txt

chmod u+x,g-w,o-rw f.txt

chmod ug+rw f.txt

chmod f.txt ↔ chmod a+w f.txt

chmod num_1 num_2 num_3 chemin_f_r_1 [chemin_f_r_2 ...]

permet le changement des permissions des fichiers ou des répertoires **chemin_f_r_1**, **chemin_f_r_2**, ... **num_1**, **num_2** et **num_3** représentent la permission absolue respectivement de l'utilisateur, du groupe et des autres utilisateurs, si l'utilisateur est propriétaire de ces fichiers ou ces répertoires.

La forme numérique, en numérotation octale, sous 3 chiffres, est :

r + w + x	111	7
r + w	110	6
r + x	101	5
r	100	4
w + x	011	3
w	010	2
x	001	1
rien	000	0

Exemple :

chmod 761 f.txt ↔ chmod a+rwx , g-x , o-rw f.txt

umask [num]

(User file creation MASK)

(set umask [num])

spécifie la permission à la création des fichiers et des répertoires. La permission est égale au complément à **666** pour les fichiers et à **777** pour les répertoires et les liens symboliques de **num** avec toute différence négative sur un chiffre est nulle. Elle affiche le mode de permission par défaut. Le mode de permission de création est stocké dans la variable d'environnement **umask**.

Exemples :

umask 000

Pour les fichiers, la permission est **666 : rw-rw-rw-**

Pour les répertoires, la permission est **777 : rwxrwxrwx**

umask 022

Pour les fichiers, la permission est 644 : rw-r--r--

Pour les répertoires, la permission est 755 : rwxr-xr-x

umask 077

Pour les fichiers, la permission est 600 : rw-----

Pour les répertoires, la permission est 700 : rwx-----

**chown [-R] nom_u chemin_f_r_1 [chemin_f_r_2 ...]
(CHange OWNer)**

change le propriétaire des fichiers ou des répertoires *chemin_f_r_1*, *chemin_f_r_2*, ... qui devient *nom_u*. Il faut être propriétaire des fichiers ou des répertoires *chemin_f_r_1*, *chemin_f_r_2*, ... pour pouvoir réaliser cette opération. L'utilisateur ne peut plus modifier ces fichiers mais il peut toujours les détruire (permission donnée par le répertoire père du fichier).

R : changement récursif pour les sous-répertoires.

**chgrp [-R] nom_g chemin_f_r_1 [chemin_f_r_2 ...]
(CHange GRouP)**

change le groupe des fichiers ou des répertoires *chemin_f_r_1*, *chemin_f_r_2*, ... qui devient *nom_g*. Il faut être membre du groupe *nom_g* et propriétaire des fichiers ou des répertoires *chemin_f_r_1*, *chemin_f_r_2*, ... pour pouvoir réaliser cette opération.

usermod [...] nom_u

change les propriétés de l'utilisateur *nom_u*.

g nom_g : change le groupe de l'utilisateur *nom_u* au groupe *nom_g*.

G nom_g_1, nom_g_2, ... : ajoute l'utilisateur *nom_u* aux groupes *nom_g_1, nom_g_2, ...*.

groupmod**gpasswd****Remarque :**

La commande **cp** garde la même description du fichier original pour le nouveau fichier.

8.3. Cryptage

crypt [chaîne] <chemin_f_s> chemin_f_d

crypte, ou décrypte, le fichier *chemin_f_s* dans *chemin_f_d* en utilisant la clé de cryptage *chaîne*. Elle demande la clé de cryptage en mode interactif sans écho par défaut.

tr [-d] chaîne_1 chaîne_2 <chemin_f_s> chemin_f_d

codifie le texte du fichier *chemin_f_s* dans *chemin_f_d* en remplaçant le premier caractère de *chaîne_1* par le premier caractère de *chaîne_2*, le deuxième caractère de *chaîne_1* par le deuxième caractère de *chaîne_2*, et ainsi de suite.

d : détruit les caractères au lieu de les remplacer.

Exemples :

tr abc xyz < f1.txt > f2.txt

codifie le fichier **f1.txt** dans **f2.txt** en remplaçant **a** par **x**, **b** par **y** et **c** par **z**.

tr xyz abc < f2.txt > f3.txt

codifie le fichier **f2.txt** dans **f3.txt** en remplaçant **x** par **a**, **y** par **b** et **z** par **c**.

tr abc xy < f1.txt > f2.txt

codifie le fichier **f1.txt** dans **f2.txt** en remplaçant **a** par **x**, **b** par **y** et **c** par **y**.

(Solaris : codifie le fichier **f1.txt** dans **f2.txt** en remplaçant **a** par **x**, **b** par **y** et **c** reste inchangé).

tr a-z A-Z < f1.txt > f2.txt

codifie le fichier **f1.txt** dans **f2.txt** en remplaçant les lettres en minuscule en lettres en majuscule.

tr -d ab < f1.txt > f2.txt

codifie le fichier **f1.txt** dans **f2.txt** en détruisant les caractères **a** et **b**.

des -e [-k chaîne] chemin_f_s chemin_f_d

crypte le fichier *chemin_f_s* dans *chemin_f_d* en utilisant la clé de cryptage *chaîne*. Elle utilise l'algorithme de cryptage NBS-DES (Data Encryption Standard). Elle demande la clé de cryptage en mode interactif sans écho par défaut.

des -d [-k chaîne] chemin_f_s chemin_f_d

décrypte le fichier *chemin_f_s* dans *chemin_f_d* en utilisant la clé de cryptage *chaîne*. Elle utilise l'algorithme de cryptage NBS-DES (Data Encryption Standard). Elle demande la clé de cryptage en mode interactif sans écho par défaut.

Partie II : Développement

9. Éditeurs de texte

On doit faire la distinction entre les **visualiseurs**, les **éditeurs**, les **formateurs**, les **filtres** et les **traitements** de texte. **UNIX** propose un ensemble d'éditeurs de texte qui permettent d'écrire, modifier ou supprimer une partie d'un fichier texte. Parmi ces éditeurs, il y a : **ed**, **edit**, **ex**, **emacs**, **vi**, (**Linux** : **vim** (vi improve), **nano**, **pico**, **gedit**, **kedit**, **kwrite**, etc.) etc. On se contente ici d'exposer les deux éditeurs de texte **vi** et **ed**.

9.1. Éditeur vi

vi (**VI**suel) est un éditeur de texte pleine page très commode pour la manipulation d'un texte. **vi** est une extension de l'éditeur de texte **ex**, il est développé par l'Université de Berkeley.

view *chemin_f*

visualise le contenu du fichier *chemin_f* sous l'éditeur de texte **vi** en mode lecture uniquement.

vi [-rR] [*chemin_f*]

appelle **vi** avec le fichier *chemin_f* s'il existe sinon il le crée, de **vi** seulement par défaut.

r : restaure les dernières modifications effectuées sur le fichier avant l'arrêt brutal du système.

R : ouvre le fichier en mode lecture seulement.

Il y a 3 espaces mémoire, ou tampon (**buffer**) :

- le **tampon document** où il est rangé tout le texte ;
- le **tampon texte** où il est rangé le dernier texte éliminé ou copié ;
- le **tampon commande** où il est rangé la ou les dernières opérations effectuées.

Il y a trois modes de traitement : le mode commande, le mode remplacement et le mode insertion. À l'appel de **vi**, l'utilisateur est en mode commande.

9.1.1. Mode commande

A. Déplacement du curseur

Les commandes de déplacement du curseur sont :

k : déplace le curseur en haut d'une ligne ;

j : déplace le curseur en bas d'une ligne ;

h : déplace le curseur à gauche d'un caractère ;

l : déplace le curseur à droite d'un caractère ;

w : déplace le curseur au début du mot suivant ;

b : déplace le curseur au début du mot courant ;

e : déplace le curseur à la fin du mot courant ;

numG : déplace le curseur à la ligne numéro **num** ;

\$: déplace le curseur à la fin de la ligne courante ;
^ : déplace le curseur au début de la ligne courante ;
Ctrl b : déplace le curseur à la page précédente ;
Ctrl f : déplace le curseur à la page suivante ;
G : déplace le curseur à la fin du fichier.

B. Modification du texte

Les commandes de modification du texte sont :

x : élimine le caractère qui est sous le curseur puis le range dans le tampon texte ;
X : élimine le caractère qui est avant le curseur puis le range dans le tampon texte ;
dw : élimine le reste du mot qui suit le curseur puis le range dans le tampon texte ;
dd : élimine la ligne sous le curseur puis la range dans le tampon texte ;
D : élimine tous les caractères après le curseur jusqu'à la fin de la ligne puis les range dans le tampon texte ;
p : reprend le contenu du tampon texte après la ligne sous le curseur ;
P : reprend le contenu du tampon texte avant la ligne sous le curseur ;
Y : copie la ligne courante dans le tampon texte ;
J : joint la ligne courante avec la ligne suivante ;
u : annule la dernière modification effectuée sur la ligne courante ;
U : supprime toutes les modifications effectuées sur la ligne courante ;
. : répète la dernière modification ;
~ : inverse le type du caractère courant, majuscule-minuscule.

On peut placer un nombre devant une requête. Il indique combien de fois la requête sera répétée, une seule fois par défaut.

Exemples :

dd ... : détruit 4 lignes.
5Y : copie 5 lignes à partir de la ligne courante dans le tampon texte.
YYYYY : répète la copie de la ligne courante dans le tampon texte 5 fois (une seule fois suffit).

C. Recherche

Les commandes de recherche de texte sont :

/[chaîne] : cherche la première occurrence de la chaîne de caractères **chaîne**, reprend la recherche de l'occurrence suivante vers le bas et d'une façon cyclique par défaut ;
?[chaîne] : cherche en arrière la première occurrence de la chaîne de caractères **chaîne**, reprend la recherche de l'occurrence suivante vers le haut et d'une façon cyclique par défaut.

D. Commandes fichier

Les commandes fichier, précédées par le symbole :, doivent être validées par **Return**.
Les commandes fichier sont :

:r chemin_f : insert le contenu du fichier **chemin_f** après la ligne courante ;

:r !chemin_c : insert le résultat de la commande *chemin_c* après la ligne courante ;
:w [chemin_f] : enregistre le contenu du tampon document dans le fichier *chemin_f*, dans le même nom du fichier déjà dans le tampon par défaut ;
:num_1,num_2w chemin_f : enregistre les lignes de *num_1* à *num_2* dans le fichier *chemin_f* ;
:q : quitte vi s'il n'y avait pas de modification portée dans le tampon document ;
:q! : quitte vi sans sauvegarder les modifications portées sur le fichier depuis l'appel de vi ou la dernière sauvegarde ;
:sh : suspend vi, Ctrl d pour le reprendre ;
:!chemin_c : exécute la commande *chemin_c* d'Unix sous Shell, sans modifier le fichier ;
:e chemin_f : édite un autre fichier de nom *chemin_f* ;
:e# : se déplace à l'autre fichier édité ;
:/[chaîne/] : cherche en avant la première ligne contenant une occurrence de *chaîne*, reprend la recherche en avant par défaut ;
:?/[chaîne?] : cherche en arrière la première ligne contenant une occurrence de *chaîne*, reprend la recherche en arrière par défaut ;
:s/[chaîne_1/chaîne_2/] : substitue la première occurrence de *chaîne_1* par *chaîne_2* sur la ligne courante, reprend la substitution par défaut ;
:s/[chaîne_1/chaîne_2/]g : substitue toutes les occurrences de *chaîne_1* par *chaîne_2* sur la ligne courante, reprend la substitution par défaut.

E. Divers

Les commandes diverses sont :

Ctrl g : affiche la ligne d'état ;
Ctrl l : réaffiche la page courante ;
Ctrl z : suspend vi, la commande **fg** permet de le reprendre.

9.1.2. Mode remplacement

Les commandes de remplacement sont :

r car : remplace le caractère sous le curseur par le caractère *car* ;
s : remplace le caractère sous le curseur par le texte tapé jusqu'à **Ctrl c** ;
R : remplace les caractères à partir du caractère sous le curseur par les caractères tapés jusqu'à **Ctrl c** ;
cw : remplace le mot courant par le texte tapé jusqu'à **Ctrl c** ;
cc : remplace la ligne courante par le texte tapé jusqu'à **Ctrl c**.

9.1.3. Mode insertion

L'insertion d'un texte se fait toujours avant la position du curseur en poussant le texte placé à droite du curseur.

Les commandes d'insertion sont :

- i** : passe au mode insertion sans déplacement du curseur.
- I** : passe au mode insertion en déplaçant le curseur au début de la ligne sous le curseur.
- a** : passe au mode insertion en déplaçant le curseur après le caractère sous le curseur.
- A** : passe au mode insertion en déplaçant le curseur à la fin de la ligne sous le curseur.
- o** : passe au mode insertion en déplaçant le curseur au début de la ligne après le curseur.
- O** : passe au mode insertion en déplaçant le curseur au début d'une nouvelle ligne créée avant le curseur.

Ctrl c : passe du mode insertion au mode commande.

Remarque :

En général, en mode insertion, le déplacement du curseur, avec le pavé de déplacement, n'est pas opérationnel.

9.2. Éditeur ed

L'éditeur de texte **ed** est l'éditeur de base le plus simple. C'est un éditeur ligne à ligne et interactif. Il est utilisé surtout pour traiter des fichiers texte de manière automatique à travers les procédures de commandes.

ed [-] *chemin_f*

appelle l'éditeur de texte **ed** en ouvrant le fichier ***chemin_f*** s'il existe, sinon il crée un nouveau fichier de nom ***chemin_f***.

- : supprime l'affichage de la taille, en nombre de caractères, du fichier ***chemin_f***.

Exemples :

ed f.txt

25

ed g.txt

?g.txt

(Solaris : n'affiche pas ce message)

Il y a trois modes de traitement : le mode commande, le mode remplacement et le mode insertion. À l'appel de **ed**, l'utilisateur est en mode commande.

9.2.1. Mode commande

A. Adresse

Un indicateur de ligne se déplace dans les lignes du fichier. La ligne courante est la ligne où l'indicateur est positionné. Au début, la ligne courante est la première ligne du fichier de numéro **1**.

Les adresses utilisables sont :

- : la ligne courante ;

num : la ligne numéro **num** ;

num_1,num_2 : de la ligne numéro **num_1** à la ligne numéro **num_2** ;

,**num_2** : de la première ligne jusqu'à la ligne numéro **num_2** (**Solaris** : non) ;

num_1, : de la ligne numéro **num_1** à la dernière ligne (**Solaris** : non) ;

1 : la première ligne ;

\$: la dernière ligne ;

/chaîne/ : la première ligne, après la ligne courante vers le bas, contenant au moins une occurrence de **chaîne**, d'une façon cyclique ;

?chaîne? : la première ligne, avant la ligne courante vers le haut, contenant au moins une occurrence de **chaîne**, d'une façon cyclique ;

^ : l'avant dernière ligne ;

/\$: la ligne suivante.

B. Caractères spéciaux

Les caractères spéciaux sont : { . , * , [, \ , ^ , \$ }.

. : un caractère quelconque ;

^ : le début de la ligne ;

\$: la fin de la ligne ;

[...] : un caractère de la suite ;

[^...] : un caractère autre que les caractères de la suite.

C. Expressions

L'expression **expr** peut avoir une des formes suivantes :

expr* : zéro, une ou plusieurs occurrences de **expr** ;

expr\{num\} : exactement **num** occurrences de **expr** ;

expr\{num_1,num_2\} : entre **num_1** et **num_2** occurrences de **expr** ;

\(expr) : expression **expr** qu'on peut récupérer dans \1, \2, ... ;

expr_1expr_2 : conjonction de **expr_1** et **expr_2** ;

^expr : **expr** au début de la ligne ;

expr\$: **expr** à la fin de la ligne ;

// : l'expression précédente ;

\num : **num** occurrences de l'expression précédente (**Solaris** : l'expression numéro **num**).

Exemples :

.* : répétition d'un caractère.

[a-z]* : répétition d'une lettre minuscule.

D. Commandes

La commande en cours est appliquée à la ligne courante sauf mention dans la commande. Une mnémonique est affichée au lieu d'un caractère non affichable (tab, ...). La ligne courante devient la

dernière ligne affectée par une commande avec mention. Une erreur dans une commande provoque l'affichage de ?.

Les commandes sont :

= : affiche le nombre de lignes du fichier ;

num : se positionne sur la ligne numéro **num** en affichant son contenu ;

num_1,num_2 : se positionne sur la ligne numéro **num_2** en affichant son contenu ;

/chaîne/ : se positionne sur la première ligne, après la ligne courante vers le bas, contenant au moins une occurrence de **chaîne** en affichant son contenu ;

?chaîne? : se positionne sur la première ligne, avant la ligne courante vers le haut, contenant au moins une occurrence de **chaîne** en affichant son contenu ;

[num_1,num_2]l : liste les lignes de **num_1** à **num_2**, la ligne courante par défaut, en les terminant chacune par le caractère \$ (Linux : les lettres accentuées ne sont pas affichées correctement) ;

[num_1,num_2]p : affiche les lignes de **num_1** à **num_2**, la ligne courante par défaut ;

[num_1,num_2]d : détruit les lignes de **num_1** à **num_2**, la ligne courante par défaut ;

u : annule l'effet de la dernière commande ed modifiant le tampon ;

w : sauvegarde le fichier puis affiche sa taille en nombre de caractères. La taille n'est pas affichée si l'option - est utilisée lors de l'appel de ed ;

q : quitte l'éditeur de texte si aucune modification n'est opérée depuis la dernière sauvegarde ;

Q : quitte l'éditeur de texte dans tous les cas ;

!chemin_c : exécute la commande UNIX **chemin_c** sans sortir de l'éditeur de texte et sans modifier le fichier ;

[num]r chemin_f : insère le contenu du fichier **chemin_f** après la ligne numéro **num**, après la ligne courante par défaut, en affichant sa taille ;

[num_1,num_2]w chemin_f : sauvegarde les lignes de **num_1** à **num_2** dans le fichier **chemin_f**, la ligne courante par défaut ;

[num_1,num_2]m num_3 : déplace les lignes de **num_1** à **num_2** après la ligne numéro **num_3**, la ligne courante par défaut ;

[num_1,num_2]t num_3 : copie les lignes de **num_1** à **num_2** après la ligne numéro **num_3**, la ligne courante par défaut ;

[num_1,num_2]s/chaîne_1/chaîne_2/ : substitue la première occurrence de la chaîne de caractères **chaîne_1** par **chaîne_2** pour toutes les lignes de **num_1** à **num_2**, de la ligne courante par défaut ;

[num_1,num_2]s/chaîne_1/chaîne_2/g : substitue toutes les occurrences de la chaîne de caractères **chaîne_1** par **chaîne_2** pour toutes les lignes de **num_1** à **num_2**, de la ligne courante par défaut ;

[num_1,num_2]g/chaîne/nom_c : exécute la commande **nom_c** de ed pour toutes les lignes de **num_1** à **num_2** contenant au moins une occurrence de **chaîne**, la ligne courante par défaut ;

[num_1,num_2]v/chaîne/nom_c : exécute la commande **nom_c** de ed pour toutes les lignes de **num_1** à **num_2** ne contenant aucune occurrence de **chaîne**, de la ligne courante par défaut ;

h : affiche un message, au lieu du symbole ?, pour la dernière erreur détectée ;
H : affiche un message, au lieu du symbole ?, pour chaque erreur détectée. Elle alterne ce mode pour la deuxième **H** ;

Remarques :

(Linux : non)

num_1 est par défaut **1**, c'est-à-dire la première ligne.

num_2 est par défaut **\$**, c'est-à-dire la dernière ligne.

num_1 et **num_2** sont par défaut tout le fichier.

Exemples :

1,\$s// /

insert un espace au début de chaque ligne.

1,\$s/^./f/

remplace le premier caractère par **f** dans toutes les lignes.

1,\$s// //

détruire le premier caractère dans toutes les lignes.

g/^/s/^/ /p

affiche toutes les lignes en insérant un espace au début de chaque ligne.

g/ bien / s// très bien /g

remplace toutes les occurrences de " bien " par " très bien " dans toutes les lignes.

g/([aeiou])^/s/\\o'\\1 ^'/g

remplace **car** par **\o'car** avec **car** dans { a , e , i , o , u }.

v/^3/d

détruit toutes les lignes qui ne commencent pas par **3**.

g/a/s//A/g

remplace toutes les occurrences de **a** par **A** dans toutes les lignes.

'^\\([A-Z][A-Za-z]*\\), ^\\([A-Z][A-Za-z]*\\)^2 \\1'

remplace Nom, Prénom par Prénom, Nom par exemple.

^\\(.*)\\1\$

ligne qui contient uniquement une chaîne de caractères répétée 2 fois.

/\\(bien\\).*\\(mal\\)/

le mot bien puis une chaîne de caractères quelconque puis le mot mal.

Exemple :

Début

ed - test.txt << @

v/^\$1/d

g/\$2/s//\$/g

w

q

@

Fin

Exemple :

Début

```
set a = bien
ed - test.txt << @
v/$a/d
w
q
@a
# Fin
```

Exemple :

```
# Début
ed - test.txt << @@
1,$d
w
q
@a
# Fin
```

Remarques :

En **C Shell**, il faut mettre **1,\\$.**

En **Bourne Shell**, il faut mettre **1,\$.**

9.2.2. Mode replacement

[num]c : passe en mode remplacement sur la ligne numéro **num** jusqu'à **Ctrl c**, sur la ligne courante par défaut.

9.2.3. Mode insertion

[num]i : passe en mode insertion avant la ligne numéro **num**, avant la ligne courante par défaut.

[num]a : passe en mode insertion après la ligne numéro **num**, après la ligne courante par défaut.

Ctrl c : passe du mode insertion au mode commande.

(**Solaris** : . au début de la ligne puis **Return** permet de passer du mode insertion au mode commande).

10. Langages de commandes

10.1. Interpréteur de commandes

UNIX offre une interface avec l'utilisateur qui est l'interpréteur de commandes **Shell**. Le **Shell** est à la fois un langage de programmation de commandes et un interpréteur de commandes. Les commandes exécutées sont lues à partir du terminal ou d'une procédure de commandes.

Le nom générique de **Shell** associé aux interpréteurs des langages de commandes d'UNIX provient de l'analogie avec les mollusques (ex. les escargots). L'interface entre ces animaux et le monde extérieur est formée par la coquille (**shell**) qui les protège.

Il y a plusieurs interpréteurs de commandes : **sh**, **sh5**, **csh**, **ksh**, ... Chaque interpréteur de commandes offre un environnement de travail différent.

Le processus **shell** est un processus comme n'importe quel autre processus de commande.

/bin/sh

/usr/bin/sh

appelle l'interpréteur de commandes **Bourne Shell**. C'est l'interpréteur de commandes **shell standard**, le premier langage de programmation de commandes. Il est développé sur **UNIX System 7** par Steve Bourne.

/usr/bin/sh5

appelle l'interpréteur de commandes **SYSTEM V Bourne Shell**. C'est un langage développé sur **UNIX SYSTEM V**.

/usr/bin/csh ou /bin/csh ou /usr/local/bin/csh

appelle l'interpréteur de commandes **C Shell**. Il est développé à l'Université de Berkeley par Bill Joy. Il est près syntaxiquement du langage de programmation C.

/usr/bin/ksh

appelle l'interpréteur de commandes **Korn Shell**.

/bin/bash

(Linux)

appelle l'interpréteur commandes **Bourne again Shell**.

/bin/tcsh

(Linux)

appelle l'interpréteur de commandes **TENEX C Shell**. Il est développé initialement par Ken Greer sous le système d'exploitation TENEX dans les années 70 puis repris par d'autres. Il est basé et compatible avec **C Shell**.

/bin/zsh

appelle l'interpréteur de commandes **Z Shell**. C'est un interpréteur interactif et un langage de commandes puissant, réunissant les avantages des autres Shell. Il a été développé par Paul Falstad en 1990.

/usr/lib/rsh

appelle l'interpréteur de commandes **restricted Shell**. C'est un **Shell** réduit qui permet d'exécuter uniquement les commandes du répertoire de base et celles qui sont accessibles par le **path**. Il ne permet ni de changer de répertoire, ni de rediriger les sorties, ni de modifier la valeur du **path**. C'est une restriction du **Bourne Shell**.

/usr/bin/rsh

appelle l'interpréteur de commandes **remote Shell**. C'est un **Shell** distant.

Lors de la connexion d'un utilisateur au système, un environnement de travail s'établit en exécutant automatiquement un ensemble de fichiers de démaragements suivant le **Shell de connexion** utilisé. Le type du **Shell de connexion**, **Shell de base**, pour chaque utilisateur, se trouve dans sa ligne de description dans le fichier **/etc/passwd**.

chsh

(Solaris : non)

(Linux : **changesh** ou **chsh**)

affiche le **Shell** de connexion et permet de le changer.

Exemple :

chsh

Changing login shell for omar

Shell [/bin/csh] : sh

Dans une **session de travail**, le processus **Shell** initial est le **processus maître** (process leader) sous le terminal de l'utilisateur qui est le **terminal de contrôle**.

Un processus peut créer une nouvelle session dont elle est le maître mais sous le même terminal de contrôle.

Un **groupe de processus** est un ensemble de processus d'une même session et dont le maître est leurs créateurs.

En **C Shell**, le lancement d'une commande crée un processus maître d'un nouveau groupe de processus dans la même session du processus **Shell**.

En **Bourne Shell**, le lancement d'une commande crée un processus dans le groupe de processus **Shell** dans la même session du processus **Shell**.

10.2. Procédure de commandes

Une procédure **Shell (Shell Script)**, procédure de commandes, traitement par lots (Batch processing), est un ensemble de commandes d'**UNIX**, de commandes **Shell** ou de programmes exécutables qui sont regroupées dans un fichier texte. Le fichier d'une procédure de commandes peut prendre n'importe quelle extension. Cependant, l'extension **sh** est utilisée pour **Bourne Shell** et **csh** pour **C Shell**. En général, lorsqu'une erreur est trouvée lors de l'exécution d'une commande d'une procédure **Shell**, son exécution est arrêtée.

Exemple :

Soit la procédure **proc** contenant :

```
clear  
date > f.txt  
echo fin  
echo 'date'  
echo $path
```

Exemple :

Soit la procédure **proc1** contenant :

```
cat f  
date
```

Soit la procédure **proc2** contenant :

```
cat g  
date
```

Soit la procédure **proc** contenant :

```
cat $1  
date
```

La commande **proc1** est équivalente à la commande **proc f**

La commande **proc2** est équivalente à la commande **proc g**

Exemple :

Soit la procédure **proc** contenant :

```
echo $1  
more $1  
wc $1
```

Exemple :

Soit la procédure **ligne** contenant :

```
head -$1 $2 | tail -1  
# echo 'head -$1 $2 | tail -1'
```

Remarque :

Le principal avantage des **langages de scripts**, tels que : Perl et Python, sur les **langages systèmes**, tels que : VBS, les fichiers batch sous DOS, ou bash/tcsh sous Linux, c'est qu'ils sont portables sous tous les systèmes d'exploitation évolués (Windows, Linux, ...) alors que cela n'est pas le cas avec VBS, batch ou bash/tcsh qui sont spécifiques à leur système de conception.

10.2.1. Procédures de commandes particulières

Dans le cas de **C Shell** :

- à la connexion, il y a : `~/.login` puis `~/.cshrc` (**C SHeLL Restart Commands**)
- à chaque lancement de **C Shell**, il y a : `~/.cshrc`
- à la déconnexion, il y a : `~/.logout`

Dans le cas de **Bourne Shell** ou **Korn Shell** :

- à la connexion, il y a : `~/.profile`

Dans le cas de **Based Bourne Shell** :

- pour le super-utilisateur, il y a : `/etc/profile`
- pour un utilisateur, il y a : `~/.bash_profile`
- au lancement, il y a : `~/.bashrc`

Dans le cas de **TENEX C Shell** :

- au lancement, il y a : `~/.tcshrc`, s'il n'existe pas alors il y a : `~/.cshrc`

Ces fichiers doivent être des fichiers texte et existés dans le répertoire de base de l'utilisateur, dont le propriétaire est l'utilisateur et le groupe coïncide avec le groupe principal/réel de l'utilisateur.

Exemple : `.login` peut contenir :

```
stty crt
tset -I -Q
echo Bonjour
date
w
mail
```

Exemple : `.cshrc` peut contenir :

```
alias rmdir rm -ir
alias rm rm -i
set history = 30
biff n
```

Exemple : `.logout` peut contenir :

```
date
w
echo Au revoir
```

Exemple : `.profile` peut contenir :

```
PATH=.:$HOME/bin:/usr/bin
MAIL=/usr/spool/mail/$USER
tset -n -I
export TERM MAIL PATH
biff n
```

10.2.2. Exécution d'une procédure de commandes

Les étapes de l'exécution d'une procédure de commandes sont :

- la recherche de l'emplacement de la procédure à partir du **path** ;
- le test de la permission de l'exécution, ou d'ouverture, de la procédure, dans certains cas ;
- la substitution des paramètres de positions ;
- l'appel du **Shell** pour l'analyse de chaque ligne de commandes.

Les étapes de l'analyse d'une ligne de commandes sont :

- la lecture de la ligne ;
- la séparation des commandes ;
- le placement de la séquence des commandes dans la liste de l'historique, dans certains cas ;
- l'exécution de chaque commande.

Les étapes de l'exécution d'une commande sont :

- la résolution des alias, dans certains cas ;
- la redirection des entrées-sorties ;
- la substitution des différentes variables ;
- lancement de l'exécution ;

Les étapes de lancement d'exécution d'une commande simple sont :

- le **Shell** ouvre les fichiers d'entrées, de sorties et d'erreurs de description 0, 1 et 2 respectivement, en cas de la non redirection des entrées-sorties ;
- le **Shell** lance **fork** pour se dupliquer, puis **wait** pour attendre la fin de l'exécution ;
- le **Shell** dupliqué lance **exec** sur la commande en question pour l'exécuter. Grâce à **exec**, à la fin de l'exécution de la commande, le **Shell** dupliqué meurt et le **Shell** original reprend son exécution.

À la fin de l'exécution d'une commande, le système retourne une valeur appelée **code de retour** (**exit status**). Cette valeur indique l'état de la terminaison de l'exécution de la commande. Dans le cas d'une terminaison normale, c'est à dire sans erreur et sans interruption, cette valeur est en général égale à 0.

Il y a 3 types de commandes :

- les commandes simples, entre parenthèses éventuellement ;
- le tube, ou pipe line : séquence de commandes simples séparées par | ou |& ;
- la liste de commandes : séquence de commandes simples ou de tubes séparés par l'un des symboles suivants : ; , & , && ou || et qui peuvent être entre parenthèses.

chemin_c_1 | chemin_c_2

lance l'exécution de **chemin_c_1** puis lance l'exécution de **chemin_c_2** en considérant les sorties résultats de **chemin_c_1** comme les entrées de **chemin_c_2**.

chemin_c_1 |& chemin_c_2

lance l'exécution de **chemin_c_1** puis lance l'exécution de **chemin_c_2** en considérant les sorties résultats et les sorties erreurs de **chemin_c_1** comme les entrées de **chemin_c_2**.

chemin_c_1 ; chemin_c_2

exécute séquentielle *chemin_c_1* puis *chemin_c_2*.

chemin_c_1 & chemin_c_2

lance l'exécution en arrière plan de *chemin_c_1* puis lance l'exécution en avant plan de *chemin_c_2* sans attendre la fin de la précédente.

chemin_c_1 && chemin_c_2

lance l'exécution de *chemin_c_1* puis si elle retourne la valeur **0** il lance l'exécution de *chemin_c_2*.

chemin_c_1 || chemin_c_2

lance l'exécution de *chemin_c_1* puis si elle ne retourne pas la valeur **0** il lance l'exécution de *chemin_c_2*.

Exemple :

(cd r1 && rm -r r2) 2> /dev/null || echo "Le répertoire r1 est inaccessible ou r2 n'existe pas"

Une commande simple est une suite de mots séparés par des caractères spéciaux. Le premier mot est le nom de la commande et les autres mots sont les options et les paramètres.

Il y a plusieurs façons pour lancer l'exécution d'une procédure de commandes :

chemin_s [-vxieu...] [*chemin_f*] [*par*]

appelle l'exécution de la procédure de commandes *chemin_f* avec les paramètres *par* en créant un nouveau **Shell** *chemin_s*, de l'entrée standard par défaut. *chemin_f* n'est pas obligatoirement exécutable. L'appel de l'interpréteur de commandes se fait après l'exécution des procédures de commandes particulières. Et sera tué à la fin en cas d'une procédure de commandes.

- : supprime l'effet des options (**Solaris** : non) ;
- v** : affiche les commandes de la procédure, avant toute substitution, au fur et à mesure de leur exécution ;
- x** : affiche les commandes avec leurs paramètres, après leurs substitutions, au fur et à mesure de leur exécution ;
- i** : exécute le **Shell** en mode conversationnel, l'entrée et la sortie standards sont associées au terminal ;
- e** : termine l'exécution du **Shell** en cas d'erreur dans l'exécution d'une commande. Ce qui est par défaut ;
- u** : provoque une erreur en cas d'utilisation d'une variable indéfinie (**Solaris** : non). C'est ce qui est par défaut.

En **C Shell** sous **Solaris**, il y a aussi les options suivantes :

- c** *chemin_f [par]* : exécute la procédure de commandes *chemin_f* avec les paramètres *par* ;
- f** : exécute la procédure de commande d'une façon rapide, c'est-à-dire sans **.cshrc** ;
- n** : analyse syntaxiquement les commandes de la procédure sans les exécutées.

Exemple :

```
csh -x
echo $shell
echo /bin/csh
/bin/csh
```

Exemple :

Soit la procédure **ligne** contenant :

```
echo 'head -$1 $2 | tail -1'
csh -v -c ligne 1 ligne
echo 'head -$1 $2 | tail -1'
head -1 ligne
tail -1
echo 'head -$1 $2 | tail -1'
```

Exemple :

csh -i ligne 1 ligne

ne retourne rien

Exemple :

Soit la procédure **ligne** contenant :

```
echo 'head -$1 $2 | tail -1'
csh -xv ligne 1 ligne
echo 'head -$1 $2 | tail -1'
echo 'head -$1 $2 | tail -1'
head -1 ligne
tail -1
echo 'head -$1 $2 | tail -1'
```

Remarque :

Soit la procédure **proc** contenant :

```
set a=5
echo $a
```

la commande **echo \$a** ne sera pas exécutée si elle se trouve à la fin du fichier **proc** et ne se termine pas par retour chariot.

source *chemin_f*

(Linux : **source *chemin_f* [par]**)

appel l'exécution de la procédure de commandes ***chemin_f*** sous le même **C Shell** courant. Cette commande ne peut être utilisée que si la procédure ne nécessite pas de paramètres de positions (Linux : les paramètres peuvent exister et l'exécution est en **C Shell** ou **Bourne Shell**). ***chemin_f*** n'est pas obligatoirement exécutable. L'exécution se termine dès qu'une erreur est rencontrée dans une ligne.

exec *chemin_f* [par]

appel de l'exécution de ***chemin_f*** sous le **Shell** courant puis termine ce processus **Shell** courant (Linux : l'exécution est en **Bourne Shell** uniquement). ***chemin_f*** doit être exécutable.

chemin_f [par]

appelle l'exécution de ***chemin_f*** avec les paramètres de positions ***par***. Cette façon d'exécution ne peut être utilisée que si ***chemin_f*** est exécutable. Cette commande est exécutée dans **Bourne Shell** sauf si ***chemin_f*** commence par un commentaire, #, avant même tout espace alors elle est exécutée en **C Shell**. (**Linux** : # !/bin/csh ou /usr/local/bin/csh pour une exécution sous **C Shell** et # !/bin/sh ou /usr/local/bin/sh pour une exécution sous **Bourne Shell**).

`chemin_c`

exécute le résultat de l'exécution de la commande ***chemin_c***. Cette façon n'est pas possible pour une procédure de commandes.

Exemple :

'date'

Vendredi : commande not found

shell *chemin_c*

(**Linux** : **Shell *chemin_c***)

exécute ***chemin_c*** sous l'interpréteur de commande qui se trouve dans la variable **shell** ou sous **/usr/bin/sh** si elle n'est pas définie.

Lors de l'exécution d'une procédure de commandes, le mode d'exécution et le type de l'interpréteur de commandes ont des effets sur l'environnement de travail, les variables définies et les variables **Shell**. En effet, il est soit :

- *le même interpréteur de commandes* : on retrouve dans la procédure toutes les variables définies avant et on retrouve à l'extérieur de la procédure tous les changements des variables faits dans la procédure mais on ne récupère pas les anciennes valeurs (cas de **source *chemin_f***).
- *un clone de l'interpréteur de commandes* : on retrouve dans la procédure toutes les variables exportables définies avant et on ne retrouve pas à l'extérieur de la procédure les changements des variables faits dans la procédure mais on récupère les anciennes valeurs (cas de ***chemin_f***). En C Shell, .tcshe rc est exécuté. En Bourne Shell, .bashrc n'est pas exécuté.
- *un nouvel interpréteur de commandes* : on retrouve dans la procédure toutes les variables exportables définies avant et on ne retrouve pas à l'extérieur de la procédure les changements des variables faits dans la procédure mais on récupère les anciennes valeurs (cas de **bash *chemin_f***).

L'espace mémoire d'un processus est composé d'une zone de code, une zone de données et une zone d'environnement. Lors de la duplication d'un processus, seule la zone d'environnement est dupliquée.

Lors de l'appel de l'exécution d'une procédure de commandes, la variable **path** est consultée pour le chemin de la procédure dans le cas de l'appel direct ou avec **exec**. Cependant, Elle n'est pas consultée dans le cas de **source** ou avec un **Shell**.

10.3. Vocabulaire

10.3.1. Caractères spéciaux

La liste des caractères spéciaux est : { ; , & , > , < , | , ? , * , ~ , [,] , { , } , # , % , \ , \$, " , ' , ` }.

\car : le caractère car normal sans son effet spécial ;

? : remplace n'importe quel caractère ;

* : remplace n'importe quelle chaîne de caractères, y compris la chaîne vide ;

~ : remplace le répertoire de base de l'utilisateur ;

[...] : un caractère de la séquence ;

[car_1-car_2] : un caractère de l'intervalle suivant l'ordre ASCII ;

[chaîne] : un caractère de la chaîne de caractères ;

{chaîne_1,chaîne_2,...} : les chaînes de caractères de la séquence prises une par une ;

\ : à la fin d'une ligne pour pouvoir poursuivre l'écriture d'une commande trop longue pour tenir sur une seule ligne.

Une chaîne de caractères entre " , ' ou ` forme un seul mot lors de la séparation des mots dans une ligne de commande.

Exemples :

[ac] : un caractère de l'ensemble { a , c }.

[a-c] : un caractère de l'ensemble { a , b , c }.

[aeiøy]* : toutes les chaînes de caractères commençant par une voyelle.

?? : toutes les chaînes de caractères ayant 2 caractères.

?b? : toutes les chaînes de caractères ayant 3 caractères dont le deuxième est b.

b : toutes les chaînes de caractères contenant b.

b* : toutes les chaînes de caractères commençant par b.

*b : toutes les chaînes de caractères se terminant par b.

*[a-d] : toutes les chaînes de caractères se terminant par une lettre de { a , b , c , d }.

'a{b,c,d}e' : la chaîne de caractères 'abe ace ade'.

'/usr/{users *b*}' : la chaîne de caractères '/usr/users /usr/bin /usr/basic'.

[!a-c] : tous les caractères sauf {a , b , c }.

La liste des **caractères génériques** est : { ? , * , ~ , [,] , { , } }

La liste des **caractères séparateurs** est : { Espace , Tabulation , & , | , ; , < , > , (,) }.

10.3.2. Paramètres de positions

La liste des paramètres de positions est :

\$0 ou \${0} : le nom de la procédure en cours ;

\$num ou \${num} : le paramètre de position numéro num, avec num compris entre 1 et 9, de la procédure en cours ;

\$* : la liste des paramètres de positions \$1, ... , \$9 ;

En **C Shell**, il y a aussi :

\$#argv : le nombre de paramètres de positions de la procédure en cours ;

\$argv[num] : le paramètre de position numéro **num**, avec **num** compris entre 1 et 9, de la procédure en cours ;

\$argv ou **\$argv[*]** : la liste des paramètres de positions **\$1, ..., \$9** ;

\$@ : un paramètre de position quelconque.

\$# : le nombre de paramètres de position.

En **Bourne Shell**, il y a aussi :

\$@ : la liste des paramètres de positions.

Remarques :

"\$*" ↔ "\$1 \$2 ..."

"\$@" ↔ "\$1" "\$2" ...

10.3.3. Variables

Il y a deux types de variable :

- les **variables définies** ;
- les **variables Shell**.

Il y a deux sortes de variable, pour chacun des deux types précédents :

- les **variables locales** ;
- les **variables exportables**.

Chaque processus, créé par le **Shell**, s'exécute en héritant d'un environnement par défaut. Un processus hérite de son ascendant un certain nombre de variables appelées les **variables exportables**. Par contre, les modifications portées à des variables dans un processus ne sont pas héritées par ses processus descendants.

La création d'une variable local rend celle-ci accessible uniquement dans le cadre du processus en cours d'exécution. La création d'une variable exportable rend celle-ci accessible dans le cadre du processus en cours d'exécution et les processus fils créés.

À la création d'un **Shell**, les variables locales sont masquées et les variables exportables sont héritées. À la destruction d'un **Shell**, les variables locales, du **Shell** père, sont accessibles et les variables exportables reprennent leurs valeurs.

Par convention, les variables **Shell** exportables, sont en majuscule. Certaines variables **Shell** exportables en majuscule, prennent leurs valeurs automatiquement des variables **Shell** locales en minuscule de même nom.

10.3.4. Divers

\$\$: le numéro de processus du **Shell** en cours d'exécution ou de son père si c'est un sous processus.

En **Bourne Shell**, il y a aussi :

\$? : la valeur renvoyée de la dernière commande exécutée ;

\$! : le numéro du processus de la dernière commande exécutée en arrière plan ;

\$- : les options du **Shell** en cours d'exécution.

Remarques :

/tmp
un répertoire système des fichiers temporaires créés par l'utilisateur ou par le système.

/tmp/\${0}\$\$
un fichier temporaire de nom *chemin_cPID* dans le répertoire **/tmp**.

10.4. Utilitaires

10.4.1. Écriture

echo [-n] expr

affiche un message suivant le type de l'argument *expr*.

L'argument *expr* est d'une des formes suivantes :

nom : *nom* après l'interprétation des caractères spéciaux ;

'nom' : *nom* sans interprétation des caractères spéciaux ;

"nom" : *nom* (un seul mot avec des espaces éventuellement) après interprétation des caractères spéciaux sauf ` et \ ;

'nom' : le résultat de l'exécution de la commande **nom** (séquence de mots) après interprétation des caractères spéciaux sous le **Shell** courant ;

\$nom : le contenu de la variable **nom** ;

\$num : le contenu du paramètre de position numéro **num** ;

-n : laisse le curseur dans la même ligne. Il est retourné à la ligne suivante par défaut.

**echo " ... **
 ... "

\ permet de poursuivre l'écriture de la chaîne de caractères si elle est trop longue pour tenir sur une ligne.

Exemples :

echo "date ; # La date courante" > test.txt

echo 125\$

echo `date`

echo \$a \$1 > f.txt

echo *** Bonjour ***	→ erreur à cause du caractère spécial *.
set a = 7	
echo \$a	→ 7
echo '\$a'	→ \$a
echo "\$a"	→ 7
echo `\$a`	→ 7 : Command not found.
echo \\$a	→ \$a
echo \'\\$a'	→ \\$a
echo "\\$a"	→ \7
echo *.*	→ plan.txt prog.exe
echo *.*'	→ *.*
echo *.*"	→ *.*
echo *.*`	→ *.* : Ambiguous
echo *	→ la liste des fichiers ou no match
echo '*'	→ *
echo \'	→ '
echo \!	→ !

10.4.2. Commentaire

chaîne

chaîne est un commentaire.

chemin_c ; # chaîne

chaîne est un commentaire.

10.4.3. Divers

clear

efface l'écran.

expr expr

évalue l'expression *expr* puis retourne son résultat. Elle retourne 0 si le résultat d'évaluation est une chaîne vide. *expr* est une suite d'expressions séparées par des espaces.

L'argument *expr* est d'une des formes suivantes :

- expr_1 \| expr_2* : *expr_1* si *expr_1* n'est ni 0 ni une chaîne vide, *expr_2* sinon ;
- expr_1 \& expr_2* : *expr_1* si *expr_1* n'est ni 0 ni une chaîne vide, 0 sinon ;
- expr_ent_1 op_1 expr_ent_2* : résultat d'une comparaison arithmétique ;
- expr_chaine_1 op_1 expr_chaine_2* : résultat d'une comparaison lexicale ;
- expr_ent_1 op_2 expr_ent_2* : résultat d'une opération arithmétique ;

expr_1 : expr_2 : le nombre de caractères communs des deux expressions (Linux : non) ;
'expr' : l'expression *expr*.

op_1 dans l'ensemble { = , > , >= , < , <= , != }
op_2 dans l'ensemble { + , - , * , / , % }

Exemples :

set a = 4

expr \$a + \$a → 8

expr \$a+\$a → 4+4

set a='expr \$a + 1' → incrémente la valeur de a

expr \$a: '.*' → nombre de caractères de a

test expr

teste la validité de l'expression logique *expr*. En cas de validité, elle retourne 0 comme valeur du code de retour.

L'argument *expr* est d'une des formes suivantes :

- r *chemin_f_r* : teste pour l'utilisateur la permission de lecture de *chemin_f_r* ;
- w *chemin_f_r* : teste pour l'utilisateur la permission d'écriture de *chemin_f_r* ;
- x *chemin_f_r* : teste pour l'utilisateur la permission d'exécution de *chemin_f_r* ;
- c *chemin_f_r* : teste la structure caractère de *chemin_f_r* ;
- b *chemin_f_r* : teste la structure bloc de *chemin_f_r* ;
- f *chemin_f_r* : teste l'existence du fichier *chemin_f_r* ;
- d *chemin_f_r* : teste l'existence du répertoire *chemin_f_r* ;
- s *chemin_f_r* : teste si le fichier ou répertoire *chemin_f_r* est de taille non nulle ;
- z *chaîne* : teste l'égalité de la longueur de la chaîne de caractères *chaîne* à 0 ;
- n *chaîne* : teste la non égalité de la longueur de la chaîne de caractères *chaîne* à 0 ;
- chain_1 = chain_2 : teste l'égalité des 2 chaînes de caractères *chain_1* et *chain_2* ;
- chain_1 != chain_2 : teste la non égalité des 2 chaînes de caractères *chain_1* et *chain_2* ;
- expr_arith_1 -eq expr_arith_2 : teste si *expr_arith_1* = *expr_arith_2* ;
- expr_arith_1 -ge expr_arith_2 : teste si *expr_arith_1* >= *expr_arith_2* ;
- expr_arith_1 -gt expr_arith_2 : teste si *expr_arith_1* > *expr_arith_2* ;
- expr_arith_1 -le expr_arith_2 : teste si *expr_arith_1* <= *expr_arith_2* ;
- expr_arith_1 -lt expr_arith_2 : teste si *expr_arith_1* < *expr_arith_2* ;
- expr_arith_1 -ne expr_arith_2 : teste si *expr_arith_1* <> *expr_arith_2* .

'*expr*' : l'expression *expr* ;

\(*expr*\) : l'expression *expr* ;

!*expr* : la négation de l'argument *expr* ;

expr_1 -a *expr_2* : la conjonction des 2 arguments *expr_1* et *expr_2* ;

expr_1 -o *expr_2* : la disjonction des 2 arguments *expr_1* et *expr_2* ;

est selent
expr_1 -o expr_2 : la disjonction des 2 arguments *expr_1* et *expr_2*.

10.5. Fonction

Pour la définition d'une nouvelle fonction :

[function]nom_fonc (...) { ... }

Pour l'utilisation d'une fonction définie :

nom_fonc [par]

Remarques :

- La définition d'une fonction doit être, au début de la procédure de commandes, avant son utilisation.
- Le nom de la fonction ne doit pas être un nom de commande, sinon la fonction masque la commande.
- Les parenthèses sont obligatoires, même en absence de paramètres.
- Une fonction peut appeler une autre, mais cette dernière doit être déjà définie.
- Les paramètres de positions \$1, ... seront substituées par les paramètres *par*.
• *une fonction peut retourner un valeur : return V*

Exemple :

Soit la procédure **proc** contenant :

```
# Debut
fonct_test () {
    echo var x in $x
    x=5
    echo par1 in $1
}
```

```
echo par1 out before $1
x=3
fonct_test Teste
echo par1 out after $1
echo var x out $x
# Fin
```

proc A B → par1 out before A
var x in 3
par1 in Teste
par1 out after A
var x out 5

11. C Shell

Le **C Shell** a une structure qui ressemble à celle du langage **C**. Il incorpore, en plus de toutes les fonctionnalités du **Bourne Shell**, un mécanisme de l'histoire des commandes, un mécanisme d'**alias** ou d'identification de commandes et un contrôleur de tâches. Les versions actuelles de **Bourne Shell** ont aussi introduites ces utilitaires.

11.1. Variables

11.1.1. Variables Shell

Une liste non exhaustive des variables **Shell** est :

- **path** : la liste des répertoires utilisés lors d'une recherche des fichiers exécutables, correspondant aux commandes et non pas à leurs paramètres. Le répertoire courant "." n'est pas obligatoire de le spécifier dans **path** (**Linux** : c'est obligatoire). **unset path** oblige la spécification des chemins absous pour les commandes, même pour le répertoire courant. Un mot nul désigne le répertoire courant.
- **cdpath** : la liste des répertoires de recherche des paramètres de la commande **cd**.
- **mail** : le nom du fichier de réception de messages par **mail**. Le système cherche l'arrivée des nouveaux messages d'une façon périodique (ex. toutes les 5 mn), puis annonce leurs arrivés.
- **term** : le nom du terminal utilisé.
- **tty** : le numéro du terminal utilisé.
- **biff** : le message à afficher immédiatement lors de la réception d'un **mail** pendant une session de travail.
- **history** : le nombre maximum de commandes à sauvegarder dans la liste des historiques, (ex. **20** par défaut).
- **cwd** : le chemin complet du répertoire de travail courant. La commande **pwd** retourne le contenu de la variable **cwd**.
- **shell** : le chemin complet du **Shell** de base.
- **home** : le chemin complet du répertoire de base de l'utilisateur. **~** désigne le contenu de la variable **home**.
- **prompt** : le message d'attente, l'invite. Par défaut, **hostname#** pour le super utilisateur et **hostname%** pour les autres utilisateurs.
- **status** : le code de retour de la dernière commande exécutée.
- **user** : le nom de connexion de l'utilisateur.
- **argv** : la liste des paramètres de positions.
- **savehist** : la sauvegarde de la liste des commandes de l'historique dans le fichier **.history** lors de la déconnexion.
- **echo** : le choix du mode de visualisation des commandes après la substitution et avant l'exécution.
- **verbose** : le choix du mode de visualisation des commandes après la substitution des paramètres et avant l'exécution.
- **notify** : l'autorisation de la réception d'un message lors de la fin d'exécution d'un processus en arrière plan.

- **noglob** : l'interdiction de l'interprétation des caractères génériques.
- **noclobber** : l'interdiction de l'écriture sur un fichier existant lors d'une redirection par `>`, elle peut être forcée par `>!`. C'est aussi, l'interdiction de l'écriture sur un fichier non existant lors d'une redirection par `>>`, elle peut être forcée par `>>!`.
- **ignoreeof** : l'interdiction de l'interprétation de `Ctrl d` pour la terminaison du **Shell**, elle peut être forcée par la commande **logout**.
- **no beep** : l'interdiction de la réception du signal sonore en cas d'erreur.
- **histchar** : le caractère de substitution de **history**. C'est le caractère `!` par défaut.

Il faut faire la différence entre les variables : **history**, **echo**, **notify**, ... et les commandes qui portent les mêmes noms.

Les variables : **savehist**, **echo**, **verbose** **notify**, **noglob**, **noclobber** et **ignoreeof** sont des variables logiques.

Les variables : **argv**, **cwd**, **home**, **path**, **prompt**, **shell** et **status** sont initialisées automatiquement par l'appel du sous **Shell**.

Les valeurs des variables **Shell** locales : **user**, **term**, **path** et **home** sont transmises automatiquement aux variables **Shell** exportables : **USER**, **TERM**, **PATH** et **HOME** respectivement.

Les variables : **tty** et **history** ne sont pas exportables.

Les variables : **path**, **cdpath** et **argv** sont considérées comme des tableaux.

Remarque :

Pour la variable **Shell prompt**, il y a :

`\!` : désigne le numéro de commande depuis le début du processus **C Shell** en cours.

11.1.2. Contenu d'une variable

\$var

\${var}

exprime le contenu de la variable **var** si elle est définie sinon le message suivant : **var** :

Undefined variable.

\$?var

\${?var}

exprime **1** si la variable **var** est définie et **0** sinon.

\$#var

\${#var}

exprime la taille du tableau **var**, soit le nombre de mots du contenu de **var**.

Exemples :

pwd	↔ echo \$cwd
cd ~	↔ cd \$home
set a = date	
\$a	
set d = `date`	
\$d	
echo \$a	→ date
echo \$?a	→ 0 ou 1
echo \$d	→ Thu Feb 2 21:44:53 WET 2017
echo \$d	↔ \$a (équivalence suivant la date !)

11.1.3. Variables locales

Le nom d'une variable est une liste de caractères alphanumériques ou le symbole souligné qui commence par une lettre. Les variables sont toujours de type chaîne de caractères, les entiers peuvent être obtenus à travers la commande **expr** ou **@**. Il y a une différence entre les lettres en majuscule et les lettres en minuscule pour le nom d'une variable.

A. Définition

a. Variables chaînes

set var = expr
 affecte la valeur de l'expression **expr** à la variable **var**. Si un espace se trouve avant = il faut qu'il y a un autre après et inversement.

set var
 affecte la chaîne vide à la variable **var**.

Exemples :

set prompt = date
 set prompt = "<!>"
 set term = vt100
 set history = 30

Exemples :

set a = `date`	→ a contient Thu Feb 2 21:44:53 WET 2017
set a = date	→ a contient date
set a = 'date'	→ a contient date
set a = "date"	→ a contient date
set a = 1 + 2	→ a contient 1
set a = '1 + 2'	→ a contient 1 + 2

```
set a = "1 + 2"      → a contient 1 + 2
set a = ls -l        → a contient ls
set a = ls\ -l       → a contient ls -l
set a = `expr 1 + 2` → a contient 3
```

b. Variables logiques

set var = 1

set var

positionne/active la variable logique *var*.

Exemples :

set noclobber = 1

set noclobber

set -o noclobber (**Linux**)

c. Variables tableaux

set var = (expr_1 expr_2 ...)

affecte la liste d'expressions *expr_1, expr_2, ...* à la variable *var*. *var* est considéré comme un tableau de taille au moins égale à 2. Le premier rang est 1.

set var[num] = expr

change la valeur de la composante *num* de la variable *var* par *expr*, si le rang *num* est inférieur ou égal à la taille de *var*.

Exemples :

set path = (. / /usr/bin ~/bin)

set path = (\$path /users/bin)

set T = (" " " " " ") → crée un tableau vide de nom T et de taille 5.

set T[2] = bien

→ set : Subscript out of range.

set T[6] = 3

set T = (TB B AB P A)

set a = (3 5)

set b = \$a[1] → b contient 3

Exemples :

set c = a*b → c contient 'afb agb ahb' si le répertoire courant contient les fichiers afb, agb et ahb comme seuls fichiers dont le nom commence par a et se termine par b.

set c = 'a*b'

set c = "a*b"

Exemples :

set c = `cat f` → c contient (1 2 3) si f est un fichier qui contient 1 2 dans la première ligne et 3 dans la deuxième ligne
 set d = `date` → d contient (Vendredi 8 janvier 2021)

d. Variables numériques

@ var = expr

affecte la valeur de l'expression numérique entière *expr* à la variable *var*. Il faut mettre un espace après @ et un autre entre les opérateurs.

expr : *expr_1 op expr_2*
op : + - * / %

Remarque :

@ var op=expr ↔ @ var = \$var op expr

Exemples :

@ A = 7 ↔ set A = 7
 @ B = \$A + 1
 @ C += 2
 @ D = 9 (7 * 6)
 @ E = expr 7

e. Paramètres de position

set `chemin_c`

affecte à \$1, \$2, ... les mots du résultat de la commande *chemin_c* (Linux : non).

Exemples :

set `date` ==> \$1 contient Vendredi, \$2 contient 8, etc.

B. Destruction

unset var

efface la variable *var*.

Exemples :

set a = bien
 setenv a "mal"
 echo \$a → bien
 unset a
 echo \$a → mal
 a = 3 → a: command not found

C. Lecture

set *var* = \$<
lit le contenu de la variable *var* à partir du terminal.

D. Écriture

set

@

affiche la liste des variables définies et les variables **Shell** non exportables avec leurs valeurs par ordre alphabétique.

Exemple :

```
set
A    bien
B    7
T    (1 5 9)
argv  0
cwd   /usr/users/informatique/omar
history 20
home  /usr/users/informatique/omar
mail   /usr/spool/mail/omar
notify (. ./usr/users/informatique/omar/bin /usr/bin)
path   >
prompt >
shell  /bin/csh
status  0
term   vt100
user   omar
```

11.1.4. Variables exportables

A. Définition

setenv *var* *expr*
affecte la valeur de l'expression *expr* à la variable exportable *var*.

setenv *var*
affecte la chaîne de caractères vide à la variable exportable *var*.

setenv *var* = \$<
lit le contenu de la variable exportable *var* à partir du terminal.

B. Destruction

unsetenv *var*
efface la variable exportable *var*.

C. Écriture

printenv

env

setenv

affiche la liste des variables définies et des variables **Shell** exportables avec leurs valeurs.

Exemple :

setenv

```
HOME=/usr/users/informatique/omar
SHELL=/bin/csh
TERM=vt100
USER=omar
PATH=.:./usr/users/informatique/omar/bin:/usr/bin
r=mal
```

Remarque :

Soit la procédure **proc** contenant :

```
echo $a
set c = 3
echo $b
setenv d 4
```

L'exécution des commandes suivantes donne :

```
csh
set a = 1
setenv b 2
proc
```

→ a : Undefined variable.

echo \$c → c : Undefined variable.

echo \$d →

source proc → 1

echo \$c → 3

echo \$d → 2

exec proc → a : Undefined variable.

echo \$c → c : Undefined variable.

echo \$d → 2

csh proc → a : Undefined variable.

echo \$c → c : Undefined variable.

echo \$d → 2

Remarque :

Soit la procédure **proc** contenant :

echo \$a

L'exécution des commandes suivantes donne :

csh

set a = 1

setenv a 2

proc → 2

source proc → 2

exec proc → 2

csh proc → 2

Remarque :

Les commandes **set**, **if**, ... sont internes à l'interpréteur de commandes. Elles ne correspondent pas individuellement à des fichiers dans le système. Leurs explications se trouvent dans **man tcsh**.

11.2. Structures de contrôle

Les structures suivantes sont considérées comme des commandes. La valeur renvoyée par une structure est égale à la valeur renvoyée par la dernière commande simple exécutée dans la structure.

Remarques :

if, **else**, **endif**, **foreach**, **while** et **end** doivent être au début de ligne.

identif:, **then**, **case**, **foreach** et **while** ne sont pas suivies de commandes dans la même ligne, sinon elles ne seront pas prises en compte.

11.2.1. Traitement conditionnel

Les formes d'expressions logiques sont :

expr_arith_1 op_rel_1 expr_arith_2

expr_chaine_1 op_rel_2 expr_chaine_2

expr_chaine op_rel_3 expr_méta_chaine

expr_log_1 op_log expr_log_2

-car chemin_f_r

\$?var

!(expr_log)

(expr_log)

test option par

op_rel_1 : { == , != , < , <= , > , >= }

op_rel_2 : { == , != }

op_rel_3 : { =~ , !~ }

op_log : { || , && }

expr_méta_chaine est une expression chaîne de caractères contenant des caractères génériques.

Le caractère **car** peut être :

- r** : si l'utilisateur à la permission de lecture de *chemin_f_r* ;
- w** : si l'utilisateur à la permission d'écriture de *chemin_f_r* ;
- x** : si l'utilisateur à la permission d'exécution de *chemin_f_r* ;
- e** : si *chemin_f_r* existe ;
- o** : si l'utilisateur est propriétaire de *chemin_f_r* ;
- z** : si *chemin_f_r* est vide ;
- f** : si *chemin_f_r* est un fichier normal ;
- l** : si *chemin_f_r* est un lien symbolique ;
- d** : si *chemin_f_r* est un répertoire.

Tout opérateur est entre au moins un espace.

Exemples :

"bien" =~?i*
"bien" !~ ???

goto identif

branchement à la commande étiquetée par *identif*.

identif:

chemin_c
étiquette la commande *chemin_c* par *identif*. Il n'y a pas d'espaces entre *identif* et :.

Exemple :

```
...
goto Fin
...
Fin:
echo Fin.
```

if (*expr_log*) *chemin_c*
exécute *chemin_c* si *expr_log* est égale à vrai ou à 1 ou différent de 0. Cette structure est
valable si *chemin_c* est une commande simple.

Exemples :

if (\$?a) date
affiche la date courante si la variable **a** est définie.
if (-f a.txt) rm a.txt
if (test -f a.txt) rm a.txt
test -f a.txt && rm a.txt
appellent la destruction de **a.txt** si **a.txt** est un fichier normal qui existe sinon elles n'affichent pas
de message d'erreur.

```
if (expr_log) then  
    liste_1  
    [else liste_2]  
endif
```

(ou Linux) exécute *liste_1* si *expr_log* est égale à vrai ou à 1 ou différent de 0, sinon elle exécute *liste_2*.

```
if (expr_log) ;  
    then liste_1  
    [else liste_2]  
endif
```

Exemple :

```
if (-f a.txt) then  
    echo a.txt est un fichier.  
else echo a.txt n'est pas un fichier.  
endif
```

```
if (expr_log_1) then  
    liste_1  
else if (expr_log_2) then  
    liste_2  
else liste_3
```

endif

Remarque : En général, dans certaines versions de C Shell, il y a un seul **endif** dans une structure conditionnelle imbriquée.

```
switch (expr)  
    case expr_1:  
        liste_1  
        breaksw  
  
    ...  
    [default:  
        liste  
        breaksw]  
    endsw
```

Remarques :

L'exécution est séquentielle jusqu'à la rencontre de **breaksw** ou **endsw**.

breaksw : permet de sortir de la commande **switch**.

default : permet d'exécuter toutes les commandes jusqu'à **breaksw** ou **endsw**.

Exemple :

```
switch ($a)  
    case 1:  
        echo 1  
        breaksw  
    case 2:  
        echo 2  
        breaksw  
    default:  
        echo autre  
    endsw
```

11.2.2. Traitement itératif

foreach *var* (*liste_1*)

liste_2

end

exécute *liste_2* pour chaque valeur de *liste_1* prise par *var*.

Exemples :

```
foreach i ($argv)
```

```
    echo $i
```

```
end
```

```
foreach i ('users')
```

```
    echo $i
```

```
end
```

while (*expr_log*)

liste

end

exécute *liste* tant que *expr_log* est égale à vrai.

Exemple :

```
set i = 10
```

```
while ($i >= 0)
```

```
    echo $i
```

```
    @ i -= 1
```

```
end
```

repeat *num* *chemin_c*

répète la commande *chemin_c num* fois. *chemin_c* doit être une commande simple.

Exemple :

```
repeat 3 date
```

break [*num*]

(Solaris : **break**)

sort de la commande d'itération, **foreach** ou **while**, de niveau *num*, de la commande d'itération courante par défaut. Les commandes qui se trouvent après **break** dans la même ligne, sont exécutées avant de sortir.

continue [*num*]

(Solaris : **continue**)

sort de l'étape courante de la commande d'itération, **foreach** ou **while**, de niveau *num*, de la commande d'itération courante par défaut. Les commandes qui se trouvent après **continue** dans la même ligne, sont exécutées avant de sortir.

11.3. Utilitaires

shift [var]

décrémente tous les rangs de la variable tableau *var*, des paramètres de positions d'une unité ainsi que le nombre de paramètres par défaut. Le nombre de paramètres ne doit pas dépasser 9. Cependant, il peut être dépassé en Linux (ex. \${10} est possible).

Exemple :

	proc	1	2	3	4	5	6	7	8	9	10	11
	\$1 = 1	\$2 = 2	\$3 = 3	\$4 = 4	\$5 = 5	\$6 = 6	\$7 = 7	\$8 = 8	\$9 = 9			
A←\$1 shift	\$1 = 2	\$2 = 3	\$3 = 4	\$4 = 5	\$5 = 6	\$6 = 7	\$7 = 8	\$8 = 9	\$9 = 10			
B←\$1 shift	\$1 = 3	\$2 = 4	\$3 = 5	\$4 = 6	\$5 = 7	\$6 = 8	\$7 = 9	\$8 = 10	\$9 = 11			

exit [num]

exit [(num)]

permet de terminer la procédure de commandes courante avec *num* comme code de retour. Le code de retour est celui de la dernière commande exécutée par défaut.

Exemples :

trap `rm /tmp/\$\$; exit` 2 3

détruit le fichier /tmp/\$\$ puis termine le processus du Shell lors de la réception de l'interruption du signal 2 ou 3.

trap `` 2 3

ignore les signaux 2 et 3.

trap 2 3

rétablit le comportement standard à la réception des signaux 2 et 3.

onintr -

ignore toute interruption qui survient.

onintr identif

branchement à la commande étiquetée par *identif* lors de l'interception d'une interruption.

eval expr

évalue l'expression *expr*.

11.4. Cas traités

Cas 1 :

Écrire une procédure Shell qui affiche la date courante en français.

Exemple :

Sat Feb 22 16 :25 :13 GMT 1986
Samedi 22 février 1986 16 :25 :13 GMT

Solution 1 :

```
# Début
set `date`
#set D='date'
switch ($1)
#switch ($D[1])
case Mon :
    set j = Lundi
    breaksw
...
endsw
switch ($2)
#switch ($D[2])
case Jan :
    set m = janvier
    breaksw
...
endsw
echo $j $3 $m $6 $4 $5
#echo $j $D[3] $m $D[6] $D[4] $D[5]
# Fin
```

Solution 2 :

```
# Début
date > f
set mois=`cut -d '-' -f 1 f`  
mois
# Fin
```

Solution 3 :

```
# Début
foreach v (`date`)
$v
...
end
...
# Fin
```

set ordre = 0
foreach v (`date`)
 set ordre += 1
 switch(\$ordre)
 case 1:
 set jour = \$v
 breaksw
 case 2:
 set mois = \$v
 breaksw
 others:
 continue
 endsw
end

Cas 2 :

Écrire une procédure **Shell** appelée soit avec 3 arguments, soit sans argument avec la lecture de 3 chaînes de caractères. Etant les 3 chaînes de caractères, elle teste si les 3 sont identiques, si deux parmi les 3 sont identiques ou si les 3 sont différentes.

Solution :

```
# Début
switch ($#argv)
case 0 :
    echo -n Chaîne 1 :
    set ch1 = $<
    echo -n Chaîne 2 :
    set ch2 = $<
    echo -n Chaîne 3 :
    set ch3 = $<
    breaksw
case 3 :
    set ch1 = $1
    set ch2 = $2
    set ch3 = $3
    breaksw
default :
    echo Syntaxe : compare [chaîne_1 chaîne_2 chaîne_3]
    exit
endsw
if($ch1 == $ch2) && ($ch1 == $ch3) then
    echo Les trois chaînes sont égales.
else if ($ch1 == $ch2) then
    echo Seulement la première et la deuxième chaîne sont égales.
else if ($ch1 == $ch3) then
    echo Seulement la première et la troisième chaîne sont égales.
else if ($ch2 == $ch3) then
    echo Seulement la deuxième et la troisième chaîne sont égales.
else echo Les trois chaînes sont différentes.

endif
endif
endif
endif
# Fin
```

Cas 3 :

- a. Ecrire une procédure **Shell** appelée soit avec 2 arguments soit avec un argument. Elle affiche tous les liens durs du fichier en premier argument dans le répertoire en deuxième argument, le répertoire courant par défaut.
- b. Généralisation pour tout le sous arbre de racine le répertoire en deuxième argument ou le répertoire courant par défaut.

Solution 1 :

```
# Début
switch ($#argv)
case 1 :
    set f = $1
    set r = .
    breaksw
case 2 :
    set f = $1
    set r = $2
    breaksw
default :
    echo Syntaxe : lien chemin_f [chemin_r]
    exit
endsw
if !(-d $r) then
    echo Le répertoire $r n'existe pas.
    exit
else if !(-f $r/$f) then
    echo Le fichier $f n'existe pas. dans $r
    exit
endif
endif
set t = `ls -i $r/$f
set i = ${t[1]}
#foreach j (`ls -i $r/$f)
#set i = $j
#break
#end
@l = 0
foreach j (`ls -il $r | grep "^\$i")  

    if ($l%9 == 8) && ($j != $f) echo $j
    @l += 1
end
#foreach j (`ls $r`)
#if (-f $j) then
#set t = `ls -i $r/$j`
#if (${t[1]} == $i) && (${t[2]} != $f) echo ${t[2]}
#endif
#end
# Fin
```

Solution 2 :

```
set t = `ls -i $r/$f
set i = ${t[1]}
foreach ff (`ls $r`)
    if (-f $ff) then
        set T = `ls -i $r/$ff`
        if (${T[1]} == $i) echo ${T[2]}
```

```

else if (-d $ff) then lien $f $r/$ff
# else if (-d $ff) then $0 $f $r/$ff
    endif
end

```

Solution 3 :

```

set t = `ls -i $r/$f
set i = ${t[1]}
find $r -inum $i -print

```

Cas 4 :

- a. Écrire une procédure **Shell** appelée soit avec 2 arguments soit avec un argument. Elle affiche tous les liens symboliques du fichier en premier argument dans le répertoire en deuxième argument, le répertoire courant par défaut.
- b. Généralisation pour tout le sous arbre de racine le répertoire en deuxième argument ou le répertoire courant par défaut.

Solution :

```

# Début
switch ($#argv)
case 1 :
    set f = $1
    set r =
    breaksw
case 2 :
    set f = $1
    set r = $2
    breaksw
default :
    echo Syntaxe : lien chemin_f [chemin_r]
    exit
endsw
if !(-d $r) then
    echo Le répertoire $r n'existe pas.
    exit
else if !(-f $r/$f) then
    echo Le fichier $f n'existe pas.
    exit
endif
endif
ls -l $r | grep '^->""$f"
# Fin

```

Cas 5 :

- a. Écrire une procédure **Shell** appelée soit avec 2 arguments soit avec un seul argument. Elle affiche tous les processus de l'utilisateur en premier argument et lancés depuis le terminal dont le numéro est en deuxième argument n'importe quel terminal par défaut.
- b. Spécifier le lien de parenté des processus.

Solution :

```
# Début
set a = 0
foreach u (`users`)
    if ($u == $1) then
        set a = 1
        break
    endif
end
if ($a == 0) then
    echo L'utilisateur $1 n'est pas connecté.
    exit
endif
if ($#argv == 2) then
    ps -aux | grep "^\$1""\$2" > /tmp/f1{\$0}$$
else if ($#argv == 1) then
    ps -aux | grep "^\$1" > /tmp/f1{\$0}$$
        else echo Syntaxe : process nom_u [num_t]
        exit
    endif
endif
set l = `wc -l /tmp/f1{\$0}$$`
@ i = $l[1]
while ($i != 0)
    set t = `head -$i /tmp/f1{\$0}$$ | tail -1`
    echo $t[2] >> /tmp/f2{\$0}$$
    @ i -= 1
end
sort -n /tmp/f2{\$0}$$ > /tmp/f3{\$0}$$
cat /tmp/f3{\$0}$$
rm /tmp/f?{\$0}$$
# Fin
```

Cas 6 :

Écrire une procédure Shell limitant le nombre de processus de chaque utilisateur connecté à 3. Toute tentative de lancement d'un 4ème processus par un utilisateur le déconnecte.

Solution 1 :

```
# Début
while (0 == 0)
    foreach u (`users`)
        set n = `ps -aux | grep "^\$u" | wc -l`
        if ($n > 3) then
            deconnexion
        endif
    end
    sleep 300
end
```

```

# Fin

# Procédure deconnexion
# Début
set p = ps -aux | grep "^\$u" ("-csh" -o "-sh")
# kill -9 $p[1]           csh      bash
# Fin

```

Solution 2 :

```

ps -aux > /tmp/f
set p = `wc -l /tmp/f
set ng = ${p[1]}
foreach u ('users')
set n = 0
set I = 2
while ($I <= $ng)
set l = `head -$I /tmp/f | tail -1`
if ($l[1]== $u) then @n += 1
...

```

Cas 7 :

Écrire une procédure **Shell** permettant la gestion d'un calepin. L'exécution de la procédure offre un menu permettant les opérations suivantes :

- L'obtention du nombre de notes contenues dans le calepin.
- La création d'une nouvelle note.
- La visualisation d'une note.
- La suppression d'une note.
- La sortie de la commande et le retour au **Shell** appelant.

Solution :

```

# Début
set f=~{$user}/calepin
debut :
clear
echo 1. Affichage du nombre de notes.
echo 2. Création d'une nouvelle note.
echo 3. Visualisation d'une note.
echo 4. Suppression d'une note.
echo 5. Fin.
echo
echo
echo
echo -n Votre choix :
set c=$<
switch($c)
case 1 :
    set n = `wc -l $f
    echo -n Nombre de notes est :

```

```

echo ${n[1]}
goto debut
breaksw
case 2 :
    echo Donner la nouvelle note :
    set note = $<
    echo $note >> $f
    goto debut
    breaksw
case 3 :
    echo Donner le numéro de la note :
    set num = $<
    head -$num $f | tail -1 > f{$$}.txt
    cat f{$$}.txt
    rm f{$$}.txt
    goto debut
    breaksw
case 4 :
    echo Donner le numéro de la note :
    set num = $<
    set n = `wc -l $f
    @ t = $num-1
    head -$t $f > f{$$}.txt
    @ q = ${n[1]}-$num
    tail -$q $f >> f{$$}.txt
    mv f{$$}.txt $f
    goto debut
    breaksw
case 5 :
    exit
    breaksw
default :
    echo Erreur.
    goto debut
    breaksw
endsw
# Fin

```

*# Debut
ret Res = `head --
echo \$res*

Cas 8 :

Écrire une procédure **Shell** permettant de supprimer toutes les références dans un répertoire donné de fichiers dont le propriétaire du processus est propriétaire. La procédure admettra comme argument le nom du répertoire à traiter. Par défaut, ce répertoire sera **/tmp**. Il faut prévoir une option **-i** assurant une suppression interactive. L'option doit être située avant l'argument.

Solution :

```

# Début
switch ($#argv)
case 0 :

```

```

set rep = /tmp
set int = 0
breaksw
case 1 :
    if ($1 == "-i") then
        set rep = /tmp
        set int = 1
    else set rep = $1
        set int = 0
    endif
    breaksw
case 2 :
    if ($1 == "-i") then
        set rep = $2
        set int = 1
    else echo "Syntaxe : supprime [-i] [chemin_r]"
        exit (2)
    endif
    breaksw
default :
    echo "Syntaxe : supprime [-i] [chemin_r]"
    exit (1)
    breaksw
endsw
if !(-d $rep) then
    echo Le répertoire $rep n'existe pas.
    exit (3)
endif
Trait
# Fin

```

Solution 1 : Trait

```

set t = `ls -l $0`
set nom = ${t[4]}
if ($int == 0) then
    find $rep -type f -user $nom -exec rm {}|;
else find $rep -type f -user $nom -exec rm -i {}|;
endif

```

Solution 2 : Trait

```

set t = `ls -l $0`
set nom = ${t[4]}
ls -l $rep | grep $nom > f{$$.txt
#ls -l $rep > f{$$.txt
#ed f{$$.txt << @@
#v/$nom/d
#w
#q
#@#
set l = `wc -l f{$$.txt`
```

```

@ i = ${!#1}
@ j = 1
while ($j <= $i)
    set t = `head -$j f{$$.txt | tail -1`
    if ($int == 0) then
        rm $rep/${t[8]}
    else rm -i $rep/${t[10]}
    endif
end
rm f{$$.txt
#if (${t[4]} == $nom) then
#...

```

Cas 9 :

Ecrire une procédure **Shell** permettant de donner la description détaillée d'un fichier ou du contenu d'un répertoire donné comme argument.

Solution :

```

# description
if ($#argv != 1) then
echo erreur de syntaxe
echo description chemin_f_r
exit
else
if (-f $1) then
echo la description du fichier $1 est :
information $1
else if (-d $1) then
echo la description du répertoire $1 est :
# find $1 -type f -exec information {} \;
# find $1 -type d -exec description {} \;
foreach fr (`ls -1 $1`)
if (-f $1/$fr) then
echo la description du fichier $1/$fr est :
information $1/$fr
else if (-d $1/$fr) then
description $1/$fr
endif
endif
endif

# information
set T = `ls -l $1`
if ($T[1] == -*) then echo le type est un fichier normal
if ($T[1] == l*) then echo le type est un lien symbolique
#grep ^l $T[1] > /tmp/f.txt ; set nb = `wc -l /tmp/f.txt`
#if ($nb[1] == 1) then echo le type est un lien symbolique
if ($T[ ] == ->*) then echo le type est un lien dur

```

```
#if ((expr $T[2]) > 1) then echo le type est un lien dur
if ($T[1] == c*) then echo le type est un fichier spécial caractère
if ($T[1] == b*) then echo le type est un fichier spécial bloc
if ($T[1] == s*) then echo le type est un socket
echo le propriétaire est $T[3]
echo la permission est : $T[1] ; # moins le caractère du type de fichier
echo la date de la création ou de la dernière modification est : $T[5] $T[6] $T[7]
```

12. Bourne Shell

12.1. Variables

12.1.1. Variables Shell

La liste non exhaustive des variables **Shell** en majuscule est :

- **PATH** : liste des répertoires utilisés lors de recherche des fichiers exécutables ;
- **MAIL** : fichier de réception de messages par **mail** ;
- **USER** : nom de connexion de l'utilisateur ;
- **TERM** : nom du terminal utilisé ;
- **SHELL** : chemin du **Shell** de base ;
- **IFS** : liste des séparateurs de champs. **Space**, **Tab** et **Newline** par défaut ;
- **PS1** : premier symbole d'attente, **\$** par défaut ;
- **PS2** : deuxième symbole d'attente, **>** par défaut. Il apparaît dans le cas où la commande tapée nécessite plus d'une ligne ;
- **PWD** : chemin absolu du répertoire courant.
- **LOGNAME** : nom de connexion de l'utilisateur.
- **HOME** : nom du répertoire de base de l'utilisateur.

Exemple :

```
IFS = '
'
for i in `ls -l`
    do echo $i
done
# i prendra ligne par ligne et non pas mot par mot
```

12.1.2. Contenu d'une variable

\$identif ou **\${identif}** : le contenu de la variable **identif**, la chaîne vide si elle n'est pas définie.

\$#identif ou **\${#identif}** : le nombre de mots dans le contenu de la variable **identif**.

\$identif[num] ou **\${identif [num]}** : le mot numéro **num** de la variable **identif**.

Le premier mot a comme numéro **0**.

\$?identif : la chaîne '1' si la variable **identif** est définie, '0' sinon.

\$# : nombre de paramètres de position.

Exemples :

```
echo $ab
echo ${a}b
```

12.1.3. Variables locales

A. Déclaration

readonly identif_1 [identif_2 ...]

rend les identificateurs *identif_1*, *identif_2*, ... comme des constantes.

Exemple :

V=5

V=7

readonly V

V=9 # erreur

readonly identif = val

(Linux)

déclare l'identificateur *identif* comme une constante initialisée à *val*.

Exemple :

readonly V=9

Remarque :

LOGNAME une fois initialisée lors de la connexion devient une constante.

B. Définition

identif=[expr]

affecte l'expression *expr* à la variable d'identificateur *identif*, la chaîne vide par défaut. Il n'y a pas d'espaces avant et après le signe =.

Exemples :

a=bien

a=pwd

a="pwd"

a='pwd'

b=\$b

c=`\$b`

d=`pwd`

a=\$HOME

c=\${a}b

set a=3 → set: command not found (dans certaines versions)

identif=(expr)

affecte l'expression *expr* à la variable d'identificateur *identif*, considéré comme un tableau. Le premier élément est de rang 1. (o en Linux)

identif = \$(commande)

Exemples :

```
a=x y  
echo $a → x y  
echo ${a[1]} → x y[1]  
ta=($a)  
echo ${ta[1]} → x
```

Exemples :

```
set -o noclobber  
      active la variable logique noclobber.  
set +o noclobber  
      désactive la variable logique noclobber.
```

Remarque :

La variable **noclobber** n'a pas d'effet sur l'option **-o** d'envoie des résultats dans un fichier (ex. la commande **find**).

set *expr_1 [expr_2 ...]*

affecte la liste d'expressions ***expr_1, expr_2, ...*** aux paramètres de positions **\$1, \$2, ...** respectivement.

set `*chemin_c*`

affecte le résultat de la commande ***chemin_c*** aux paramètres de positions **\$1, \$2, ...**

set *identif_1 [identif_2 ...]*

(Linux : non)

affecte la liste d'expressions des paramètres de positions **\$1, \$2, ...** aux variables d'identificateurs ***identif_1, identif_2, ...*** respectivement.

C. Lecture

read *identif_1 [identif_2 ...]*

lecture des valeurs des variables ***identif_1, identif_2, ...*** à partir du terminal.

D. Écriture

set

affiche la liste des variables définies avec leurs valeurs.

Exemple :

```
set  
EXINIT=set redraw wn=8  
HOME=/usr/users/informatique/omar  
IFS=  
MAIL=/usr/spool/mail/omar
```

```
PATH=.: /usr/users/informatique/omar/bin:/usr/bin  
PS1=$  
PS2=>  
SHELL=/bin/csh  
TERM=vt100  
USER=omar  
A=bien
```

12.1.4. Variables exportables

A. Exportation

export *identif_1* [*identif_2* ...]

rend les variables *identif_1*, *identif_2*, ... exportables.

export *identif*=*expr*

(Linux)

définit la variable *identif* comme exportable en lui affectant *expr*.

B. Écriture

env

export

affiche la liste des variables exportables ainsi que leurs valeurs.

Exemple :

env

```
EXINIT=set redraw wn=8  
HOME=/usr/users/informatique/omar  
MAIL=/usr/spool/mail/omar  
PATH=.: /usr/users/informatique/omar/bin:/usr/bin  
SHELL=/bin/csh  
TERM=vt100  
USER=omar  
B=mal
```

12.1.4. Variables numériques

declare -x *identif*=*expr*

(Linux)

définit la variable *identif* comme exportable en lui affectant *expr*.

declare -i *identif*=*expr*

(Linux)

définit la variable *identif* comme numérique en lui affectant *expr*.

Exemple :

```
# Dans une expression, a et $a représentent la même chose.
declare -i a=3
a=a+1
a=$a+1
echo $a      → 5
```

Exemple :

```
i=3
let j=$i + 1
echo $j      → 4
```

Remarque :

En Linux, \ n'est obligatoire ni dans une ligne de commandes ni dans une procédure de commandes.

Exemple :

```
i=`echo 3/4 | bc -l`
echo $i      → .7500000
```

typeset [-aAfFilrtux] [-p] identif[=val] ...
 (Linux)

définit une variable.

12.2. Structures de contrôle

```
if liste_1
then liste_2
[elif liste_3
then liste_4
...
[else liste]]
fi
```

exécute *liste_1*. Si elle retourne la valeur 0 alors *liste_2* est exécutée sinon elle exécute *list_3* et ainsi de suite.

Exemple :

```
if test -f $1
# if (ls $1 > /dev/null) 2> /dev/null
then echo $1 existe
else echo $1 n'existe pas
fi
```

Remarques :

chemin_c_1 && chemin_c_2

est équivalent à

```
if chemin_c_1
then chemin_c_2
fi
```

<pre><i>chemin_c_1</i> <i>chemin_c_2</i></pre>	est équivalent à	<pre>if ! <i>chemin_c_1</i> then <i>chemin_c_2</i> fi</pre>
case <i>identif</i> in		case \$<i>identif</i> in
<i>nom_1_1</i> [, <i>nom_1_2</i> , ...]) <i>liste_1</i> ; ;		<i>nom_1_1</i> [<i>nom_1_2</i> ...]) <i>liste_1</i> ; ;
<i>nom_2_1</i> [, <i>nom_2_2</i> , ...]) <i>liste_2</i> ; ;		<i>nom_2_1</i> [<i>nom_2_2</i> ...]) <i>liste_2</i> ; ;
...		...
<i>[liste];;</i>		<i>[*) liste;;]</i>
esac		esac
		(Linux)

exécute *liste_1* si la valeur de *identif* est égale à l'une des valeurs de *nom_1_1*, *nom_1_2*, Sinon elle exécute *liste_2* si la valeur de *identif* est égale à l'une des valeurs de *nom_2_1*, *nom_2_2*, Et ainsi de suite jusqu'à l'exécution de *liste* s'il n'y avait aucune unification auparavant.

Exemple :

```
case i in
  1) echo 1;;
  2) echo 2 ;;
  3,4) echo 3 ou 4 ;;
    echo autre;;
esac
```

Exemple :

```
case f in
  *.c) echo Fichier source c ;;
  *.p) echo Fichier source Pascal ;;
  *.o) echo Fichier objet ;;
  *.a) echo Fichier bibliothèque ;;
  *) echo Autres fichiers ;;
esac
```

Exemple : (Linux)

```
case "$suite" in
  [A-Za-z]*) echo "$suite" >>$1 ;;
  [0-9]*) echo "$suite" >>$2 ;;
  *) echo "$suite" >>$3 ;;
esac
```

```
for identif in [nom_1 [ , nom_2 , ... ] ]
[ do liste ]
done
```

exécute *liste* pour chaque valeur de *nom_1*, *nom_2*, ..., pour chaque paramètre de position par défaut.

Exemple :

```
for i in 1 2 3
    do echo Cas $i
done
```

Exemple :

```
for i in
    do echo Cas $i
done
```

est équivalent à

```
for i in "$@"
    do echo Cas $i
done
```

for i in \$@
=

for i
=

Exemple : (non Linux)

```
for f in *
    do echo $f
done
```

affiche tous les noms des fichiers du répertoire courant.

```
while liste_1
[ do liste_2 ]
done
```

exécute *liste_1*. Si elle retourne la valeur 0 alors *liste_2* est exécutée puis elle recommence, sinon la commande est terminée. La valeur renvoyée de cette commande est la valeur de la dernière exécution de *liste_2* qui est celle de la dernière commande exécutée.

Exemple :

```
while :
    do date
done
```

Exemple :

Créer une liste de fichiers vides de nom **fi** avec **p <= i <= q** ; **p** et **q** sont des données.

```
i = $p
while test $i -le $q
do
> f$i
i = `expr $i + 1'
done
```

ou

```
i = $p
while $i <= $q
do
> f$i
i = `expr $i + 1'
done
```

Remarque :

: est la commande vide.

```
until liste_1
[ do liste_2 ]
done
```

exécute *liste_1*. Si elle ne retourne pas la valeur **0** alors *liste_2* est exécutée puis elle recommence, sinon la commande est terminée. La valeur renvoyée de cette commande est la valeur de la dernière commande exécutée de *liste_2*.

Exemple :

Lire une réponse jusqu'à ce qu'elle soit « oui » ou « non ».

```
question = "taper votre réponse (oui ou non)"
rep = ""
until ($rep = oui -o $rep = non)
do
    echo -n $question
    read rep
done
ou
question = "taper votre réponse (oui ou non)"
until
    echo -n $question
    read rep
    test $rep = "oui" || test $rep = "non"
done
```

12.3. Utilitaires

chemin_f

exécute le fichier **Shell** *chemin_f*, *chemin_f* peut être le nom du fichier où il se trouve pour une récursivité.

(*liste_c*)

exécute la liste de commandes *liste_c* dans un sous **Shell**.

Exemples :

cd ; pwd

change le répertoire courant puis affiche son nom.

(cd ; pwd) ; pwd

à la fin, le répertoire courant reste inchangé.

{ *liste_c* ; }

exécute la liste de commandes *liste_c*.

`*chemin_c*`
exécute la commande *chemin_c*.

eval *chemin_c*
évalue la commande *chemin_c*.

Exemple :
eval date
affiche la date courante.

eval echo \$identif
affiche *\$identif* car le **Shell** n'effectue l'évaluation que sur un seul niveau.
(**Solaris** : affiche la valeur de la variable *identif*.)

Exemple :
ls \$rep > f
set `cat f`
lig = `wc -l f`
tail = \$lig[1]
i = 0
while test \$i -le \$tail
 i = `expr \$i + 1`
 dola = \\$
 fich = \$dola\$i
 echo \$rep`eval echo \$fich`
done

12.4. Cas traités

Cas 1 :
Ecrire une procédure **Shell** qui affiche la date courante en français.

Exemple :
Sat Feb 22 16 :25 :13 GMT 1986
Samedi 22 février 1986 16 :25 :13 GMT

Solution :
Début
date > f{\$\$}.txt
read a b c d e f < f{\$\$}.txt
rm f{\$\$}.txt
date | read a b c d e f
set date
set read a b c d e f
case a in
 Mon) j=Lundi;;

```

...
esac
case b in
    Jan) m=janvier;;
...
esac
echo $j $c $m $f $d $e
# Fin

```

Cas 2 :

Ecrire une procédure **Shell** permettant de lancer l'exécution d'un logiciel s'il n'est pas déjà en exécution.

Solution :

```

# Début
for m
do while true
    do if expr `ps -aux | fgrep -c "lctm/bin/Eam"`` = 0 > /dev/null
        then break
    fi
    sleep 300
done
~lctm/bin/ampac $m < rep > /dev/null
done
# Fin

```

Cas 3 :

Ecrire une procédure **Shell** limitant le nombre de processus de chaque utilisateur connecté à 3. Toute tentative de lancement d'un 4ème processus par un utilisateur le déconnecte.

Solution :

```

# Début
while true
do for u in `users`
    do if expr $u != root > /dev/null
        then if expr `ps -aux | grep ^$u | wc -l` > 3 > /dev/null
            then sh deconnexion $u
        fi
    fi
done
sleep 300
done
# Fin

# Procédure deconnexion
# Début
for u
do q=

```

```

for p in `ps -aux | grep ^$u | fgrep -e '-csh (csh)'
do if expr $q = $u > /dev/null
then kill -9 $p
fi
q=$p
done
done
# Fin

```

Solution :

```

# Début
while true
do for u in `users`
do if expr `ps -aux | grep ^$u | wc -l` > 3 > /dev/null
then q=
for p in `ps -aux | grep ^$u`
do if expr $q = $u > /dev/null
then kill -9 $p
fi
q=$p
done
fi
done
sleep 300
done
# Fin

```

Bibliographie

- [1] *UNIX : Premier contact*, Yukihiro Nishinuma et Robert Espesser, Eyrolles, 1989.
- [2] *UNIX*, Humberto Lucas, Bernard Martin et Georges De Sablet, Eyrolles, 1990.
- [3] *Programmer UNIX*, Richard Stoeckel, Arnand Colin, 1992.
- [4] *L'environnement de programmation UNIX*, Brian Kernighan et Rob Pike, Traduit par Eric Horlait, InterEditions, 1986.
- [5] *Le système UNIX*, Steve Bourne, Traduit par Michel Dupuy, InterEditions, 1985.
- [6] *Conception du système UNIX* Maurice J. Bach, Traduit par Guillaume Fellah, Masson, 1993.
- [7] Sécurité Et Performances Des Systèmes UNIX, Richard Stoeckel, Eyrolles, 1993.
- [8] *UNIX Programmation et communication*, Jean-Marie Rifflet et Jean-Baptiste Yunès, Dunod, EAN : 9782100079667, 2003.
- [9] *UNIX Shell*, Abdelmadjid Berlat, Jean-François Bouchaudy et Gilles Goubet, Eyrolles, 2002.
- [10] *Systèmes d'exploitation*, Bart Lamiroy, Laurent Najman et Hugues Talbot, Collection Syntex, Pearson Education France, <http://bart.lamiroy.free.fr>
- [11] <http://www.pearsoneducation.fr/espace/livre.asp?idEspace=75&idLivre=2808&dep=0>
- [12] *Système d'Exploitation*, Juan Manuel Torres, Hanifa Boucheneb et Michel Gagnon, <http://www.cours.polyml.ca/inf3600/>
- [13] *Systèmes d'exploitation : concepts et algorithmes*, Joffroy Beauquier, Béatrice Bérard, McGraw-Hill 1990.
- [14] *Conception du système UNIX*, Maurice J. Bach, Masson, 1989.
- [15] *La programmation sous UNIX*, J.-M. Rifflet, Ediscience international, 1993, <http://www.pps.jussieu.fr/~rifflet/book1.html>
- [16] *L'histoire d'UNIX et ses dérivées*, <http://perso.wanadoo.fr/levenez/UNIX/>
- [17] *Systèmes d'exploitation et réseaux*, Bart Lamiroy, Mines de Nancy, Polycopié du cours, <http://www.mines.inpl-nancy.fr/~lamiroj/ENSEIGNEMENT/SI132/poly.pdf>
- [18] *Historique et évolution des ordinateurs*, Djamal Rebaïne, wwwens.uqac.ca/~rebaïne/8INF212/chapitre1.ppt
- [19] <http://www.basiclinuxcommand.com/>
- [20] Documentations d'**ULTRIX, Solaris et Linux**.

Annexes

1. Liste des symboles utilisés

chemin_f : chemin d'accès d'un fichier.
chemin_r : chemin d'accès d'un répertoire.
chemin_f_r : chemin d'accès d'un fichier ou d'un répertoire.
chemin_x : chemin d'une commande, d'une procédure de commandes ou d'un fichier exécutable résultant d'une compilation.
chemin_c : chemin d'une commande.
chemin_s : chemin d'un **Shell**.
nom_g : nom d'un groupe.
nom_m : nom d'une machine.
nom_u : nom de connexion d'un utilisateur.
nom_p : nom d'un périphérique.
num : nombre entier.
nom : nom.
chaîne : chaîne de caractères.
expr : expression.
var : variable.
par : paramètre.
arg : argument.
identif : identificateur.
car : un caractère.
tty : numéro d'un terminal Télétype ou la console.
num_t : numéro d'une tâche.
num_s : numéro d'une tâche en soumission.
num_nom_s : numéro ou nom d'un signal envoyé à un processus.
num_p : numéro d'un processus.
... : suite de termes.
[...] : partie optionnelle.
terme : terme, en **gras**, à mettre.
terme : terme, en *italique*, à remplacer.

2. Liste des fichiers particuliers suivant certaines versions

core

l'image de la mémoire générée lors de la présence d'une erreur.

/etc/passwd

la liste des utilisateurs avec d'autres informations pour chaque utilisateur : nom de connexion, mot de passe codé par le système, numéro d'identification de l'utilisateur, numéro d'identification du groupe, informations propres à l'utilisateur, interpréteur de commandes et répertoire de base.

/etc/group

la liste des groupes avec d'autres informations pour chaque groupe : nom du groupe, mot de passe, *, numéro d'identification du groupe, liste des utilisateurs membres de ce groupe.

/dev/null

le fichier de périphérique toujours vide.

/dev/ttynum

le nom de la sortie synchrone de numéro **num**.

dead.lettre

le fichier généré lors de la présence d'une erreur au cours de la rédaction d'un message et qui contient le message.

.cshrc

les commandes à exécuter à chaque appel du **C Shell**.

.login

les commandes à exécuter au début d'une session dont le **Shell** est **C Shell**.

.profile

la liste des commandes à exécuter au début d'une session dans le **Shell** est le **Bourne Shell** ou le **Korn Shell**.

.sendrc

la liste des utilisateurs auxquels l'utilisateur envoie un **mail**.

mbox

le fichier des messages.

/usr/bin/calendar

le fichier agenda.

a.out ou obj

le fichier de sortie d'un assemblage et d'une édition de lien.

/etc/hosts

la liste des machines connectées en réseau.

/etc/shells

la liste des Shells existants.

/usr/pub/ascii

le tableau des caractères avec leur code ASCII.

/usr/include/machine/param.F

la configuration de la machine.

/usr/adm/shutdownlog

le message envoyé avant l'arrêt du système après un **shutdown**.

/dev/lp

le nom de la sortie imprimante parallèle.

quotas

le fichier des quotas.

/etc/ttys

la liste des terminaux reconnus par le système. Si dans une ligne le 5ème champ est **secure** l'utilisateur peut entrer par **root** dans le terminal correspondant. S'il y a **not secure** l'utilisateur ne pourra pas le faire.

/etc/rc

le fichier de commandes à exécuter à chaque initialisation du système :

- la destruction des fichiers temporaire ;
- le montage des volumes ;
- l'initialisation des imprimantes et des lignes de communication ;
- la création des processus **update** et **cron**.

/etc/fstab

(File System TABLE)

la liste des volumes montés ou à monter.

/etc/utmp

la liste de toutes les connexions des utilisateurs et leur numéro de terminal.

/etc/adm/lastlog

le fichier contenant, pour chaque utilisateur, la date, l'heure et le numéro du terminal de la dernière connexion.

/etc/tty?

le fichier de tous les terminaux du système.

/usr/lib/crontab

le fichier de commandes à exécuter périodiquement.

/usr/lib/cron/at.allow

la liste des utilisateurs autorisés à exécuter **at** ou **batch**.

/usr/lib/cron/at.deny

la liste des utilisateurs non autorisés à exécuter **at** ou **batch**.

printcap

les caractéristiques des différentes imprimantes.

/usr/var/adm/wtmp

l'historique des commandes exécutées par les utilisateurs.

/dev/kmem

la mémoire physique.

/etc/dumpdates

la trace des archivages des volumes. Il contient dans chaque ligne : le nom du volume, niveau d'incrément et la date d'archivage.

3. Liste des répertoires particuliers suivant certaines versions

/tmp

le répertoire système qui permet à tout utilisateur de créer des fichiers. Il est vidé automatiquement à chaque démarrage du système.

/usr/tmp

le répertoire des fichiers temporaires.

/usr/spool/at

le répertoire des fichiers contenant les exécutions différées.

lost+found

le répertoire des fichiers perdus.

/usr/dict

le répertoire des fichiers de la correction d'orthographe.

/usr/etc/subsets

le répertoire des logiciels installés.

/dev

le répertoire des fichiers spéciaux, pilotes (**device** ou driver).

/bin

le répertoire des commandes de bases, développées par les laboratoires BELL.

/usr/bin

le répertoire des commandes de bases, développées par les laboratoires BELL.

/usr/ucb

le répertoire des commandes de bases, développées par l'Université de Berkeley.

/lib

le répertoire des bibliothèques de bases.

/etc

le répertoire des fichiers exécutables et des fichiers de gestion du système.

/usr

le répertoire des outils et des fichiers supplémentaires.

4. Exercices avec des indications suivant certaines versions

Exercice 1 :

Comment interdire l'exécution d'une commande ?

Indications :

- Détruire le fichier contenant la commande.
- Renommer le fichier contenant la commande.
- Éliminer la permission d'exécution du fichier contenant la commande.

Exercice 2 :

Comment interdire la connexion d'un ancien utilisateur ?

Indications :

- Changer son mot de passe.
- Enlever son compte.
- Enlever son **Shell** de connexion du fichier **/etc/passwd**.
- Remplacer son **Shell** de connexion du fichier **/etc/passwd** par une commande.

Exercice 3 :

Comment interdire toutes les manipulations du contenu du compte d'un utilisateur ?

Indications :

- Éliminer la permission **rwx** de son répertoire de base de l'utilisateur.
- Changer le propriétaire du répertoire de base de l'utilisateur.
- Éliminer le répertoire de base de l'utilisateur.

Exercice 4 :

Comment interdire toutes les manipulations du contenu du compte d'un utilisateur par tout autre utilisateur ?

Indication :

Éliminer la permission **rwx** du groupe et des autres utilisateurs du répertoire de base de l'utilisateur.

Exercice 5 :

Quand est ce qu'un utilisateur peut détruire un fichier ?

Indications :

- Si l'utilisateur est propriétaire et il a la permission d'écriture du fichier.
- Si l'utilisateur est propriétaire et il a la permission d'écriture du répertoire père du fichier.
- Si le fichier est un lien dur et l'utilisateur est propriétaire du répertoire père du fichier et même s'il n'est pas propriétaire du fichier.

Exercice 6 :

Donner les différentes étapes qu'il faut suivre pour partager un fichier **f** d'un utilisateur **u1** avec deux autres utilisateurs **u2** et **u3**.

Indications :

- Créer un groupe **g**.
- Mettre **u1**, comme membre de **g**.
- Créer un fichier vide **f** par **u1**, **u2** et **u3**.
- Changer le groupe de **f** par **g**.

- Ajouter la permission de lecture et d'écriture du fichier **f** pour le groupe **g**.
- Créer un fichier **f** par **u2** et **u3** qui a un lien dur ou symbolique (non obligatoire).

Exercice 7 :

Comment l'utilisateur peut garder la validité d'une nouvelle identification d'une commande même après la création d'un nouvel interpréteur de commandes ?

Indication :

Il suffit de la mettre dans le fichier **.cshrc**.

Exercice 8 :

Décrire les différentes commandes du changement de la protection, du propriétaire et du groupe propriétaire d'un fichier ou d'un répertoire.

Exercice 9 :

L'interruption anormale d'un processus génère le fichier **core** dans le répertoire courant. Ce fichier contient une image mémoire du processus et par conséquent sa taille est très grande. Comment un utilisateur peut-il empêcher sa création ?

Exercice 10 :

Quelles sont les différentes méthodes pour rendre une procédure **Shell** exécutable ?

Exercice 11 :

Quelles sont les différentes étapes suivies par le système pour exécuter une procédure **Shell** ?

Exercice 12 :

Décrire les différentes commandes de la communication électronique en **UNIX**.

Exercice 13 :

Donner une description détaillée de la gestion des processus en **UNIX** ainsi que les différentes commandes associées.

Exercice 14 :

Donner une version "la plus complète" des 3 fichiers suivants :

.login
.logout
.cshrc

Exercice 15 :

Écrire une procédure **Shell** permettant d'afficher le nombre de processus ainsi que le numéro du processus de connexion de chaque utilisateur connecté. Rendre la procédure en exécution continuellement.

Exercice 16 :

Décrire la gestion de processus.

Exercice 17 :

Décrire les étapes de l'exécution d'une commande.

Exercice 18 :

Écrire une procédure **Shell** permettant la gestion d'un calepin. Les arguments de la procédure, lors de son exécution, sont les suivants :

- "**-n**" : pour obtenir le nombre de notes.
- "**-v m**" : permet de visualiser la **m**ème note.
- "**-v m n**" : pour visualiser les notes de numéro compris entre **m** et **n**.
- "**-e**" : pour vider le calepin de son contenu.
- "**-e m**" : pour enlever la **m**ème note.
- "**-e m n**" : pour enlever toutes les notes de la **m**ème à la **n**ème.

Exercice 19 :

Écrire une procédure **Shell** permettant d'afficher le nombre de paramètres entrés dans une ligne de commandes de l'appel de cette procédure de commandes.

Exercice 20 :

Écrire une procédure **Shell** permettant de tester l'existence d'un fichier dans un répertoire. Le fichier et le répertoire dans deux paramètres de la procédure.

Exercice 21 :

Comment l'utilisateur peut limiter les commandes utilisables par un utilisateur ?

Indications :

- Mettre dans la ligne correspondante à l'utilisateur du fichier **/etc/passwd** : **rsh** comme **Shell** de connexion.
- Mettre dans un répertoire particulier, ou le répertoire de base de l'utilisateur, les fichiers correspondants aux commandes autorisées.

Exercice 22 :

Discuter en détail les différents cas possibles de l'exécution de la commande : **rm t**

Indications :

- **rm** est-il un fichier dans le répertoire courant ?
- si oui, le répertoire . est-il dans la variable **path** ?
- si oui, le fichier **rm** est-il exécutable ?
- **rm** est-il un fichier dans le répertoire **/bin** ?
- si oui, le répertoire **/bin** est-il dans la variable **path** ?
- si oui, le fichier **rm** est-il inchangé ? le fichier **rm** est-il exécutable ? le fichier **rm** est-il en permission d'exécution ?
- si oui, **t** est-il un fichier ? **t** est-il dans le répertoire courant ? **t** n'est-il pas en exécution en arrière plan ou suspendu ? **t** n'est-il pas ouvert par un autre utilisateur ou un autre processus ?
- si oui, le répertoire courant est-il protégé contre l'écriture ?
- si oui, si l'utilisateur répond par **y** de la demande de confirmation de destruction de **t**, alors il est détruit.

Exercice 23 :

Discuter en détail les différentes causes de l'obtention du résultat : **f: command not found** de l'exécution de la commande **f**.

Indications :

- **f** se trouve dans le répertoire courant et il est un fichier binaire, résultat d'une compilation et d'une édition de lien, ou une procédure de commande qui permet d'afficher le message : **f: command not found** ;
- **f** se trouve dans le répertoire courant mais le répertoire **.** ne se trouve pas dans la variable **path**.
(set path = (. \$path) ou ./f) ;
- **f** ne trouve pas dans les répertoires du **path**.

La valeur du code de retour de l'exécution de **f** permet de distinguer si le résultat est un message à afficher (**\$status = 0**) ou un message d'erreur (**\$status ≠ 0**).

5. Travaux Pratiques

1. Tester les différentes commandes d'**Unix** dans de différentes situations.
2. Lire le manuel d'utilisation des différentes commandes : **man nom_c**
3. Lire le manuel d'utilisation des différents interpréteurs de commandes : **man nom_s**
4. Combiner les différentes commandes entre elles : **nom_c_1 | nom_c_2**
5. Résoudre les différents exercices précédents.
6. Développer les cas traités dans les chapitres précédents.

6. Projets

Projet 1 :

Expliquer, d'une façon concise, ce que fait chacune des commandes suivantes, en discutant les différents cas possibles :

1. **find a -type f | grep c**
2. **wc -l < ps**
3. **cut -f 1 a/b**
4. **sort -n a b > c**
5. **rm a/b*c/ ?d[ef]**
6. **rm ../*b**
7. **md a**
8. **uniq a/b c/d**
9. **mv a/b c/d**
10. **ps a**
11. **a < b > c 2> d**
12. **a || b ; c && d**
13. **cd _abc/x**
14. **cut -z ./x/y**
15. **di a b**
16. **uniq a b**
17. **look a b**
18. **rm -r a/b/c d**
19. **a < b > c 2> d**
20. **join a b**

Projet 2 :

Donner la (ou les) commande(s) qui peuvent répondre aux questions suivantes, en décrivant en détail pour chaque commande, les conditions d'exécution, les permissions minimales, les options, les paramètres et les résultats, de plusieurs façons quand cela est possible :

1. Créer un fichier **f1** contenant les 5 premières lignes de numéro pair du fichier **f2**.
2. Compter le nombre des lignes communes aux deux fichiers **f1** et **f2**.
3. Partager le fichier **f** avec l'utilisateur **u**.
4. Interdire la modification du fichier **f** pour tous les utilisateurs sauf pour les membres du groupe **g**.
5. Déterminer le type fichier ou répertoire de **fr**.
6. Compter le nombre de lignes où apparaît la chaîne de caractères :
*** C'est facile ! *** dans le **f**.
7. Créer un fichier **f** contenant la description des fichiers vides d'un répertoire **r**.
8. Compter le nombre de lignes contenant la chaîne **UNIX** dans un fichier **f**.
9. Partager le fichier **f** avec uniquement les utilisateurs **u1**, **u2** et **u3**.
10. Interdire toutes opérations sur les fichiers et les répertoires d'un utilisateur **u**.
11. Reprendre l'exécution d'un processus de numéro **p**.
12. Renommer la commande de création d'un lien d'un fichier par le nom **liendur** pour le lien dur et le nom **liensym** pour le lien symbolique.
13. Compter le nombre de sous répertoires d'un répertoire **r**.

14. Fusionner verticalement les lignes des deux fichiers **f1** et **f2**.
15. Trier les lignes d'un fichier **f**, de données en colonnes, en fonction de la 3ème colonne par ordre décroissant.
16. Lancer l'exécution du programme **trait** en arrière plan tout en redirigeant les données dans le fichier **don** et les résultats dans le fichier **res**.
17. Afficher l'UID d'un utilisateur dont le nom de connexion est **tp3**.
18. Créer un fichier **f** contenant "**Bon courage !**", sans utiliser un éditeur de texte.
19. Insérer le message **Bon courage !** dans une ligne après la 3ème ligne du fichier **f** de 7 lignes, sans utilisation d'éditeur de texte.
20. Comparer les deux fichiers **f1** et **f2**.
21. Comparer les deux répertoires **r1** et **r2**.
22. Afficher la description du 3ème plus petit fichier du répertoire **r**.
23. Afficher les lignes du fichier **f** contenant uniquement la chaîne **Admis** ou **Admise**, en précédant chaque ligne par son numéro de ligne.
24. Ajouter le contenu du fichier **f** au début de celui de **g**.
25. Afficher uniquement la taille du fichier **f**.

Projet 3 :

Etudier l'exécution d'une procédure de commandes, suivant les différents Shell : C Shell et/ou Bourne Shell. En particulier :

- Le contenu diversifié des procédures de commandes particulières.
- Les différentes façons de lancer l'exécution d'une procédure de commandes.
- Les différentes variables d'environnement.
- Les différentes sortes de variables définies : locales et/ou exportables.

Projet 4 :

Écrire une procédure **Shell** qui permet de chercher tous les liens symboliques et/ou durs d'un fichier dans un répertoire suivant un niveau de profondeur.

- Les arguments sont : le fichier, le répertoire (répertoire courant, répertoire père du fichier, ...), le choix de liens (symboliques et/ou durs), le niveau de profondeur. Ils sont soient des paramètres soient des données, éventuellement certains par défaut.
- Utiliser le maximum de procédures et éventuellement certaines fonctions : lien, liend, liens, ...
- Utiliser plusieurs méthodes : find, les structures de contrôles, ed, ...
- Généraliser pour une recherche récursive : recherche dans un répertoire ou à partir d'un répertoire.
- Utiliser plusieurs Shell : C Shell et/ou Bourne Shell.