Le système UNIX

I. Ferrané et M.C. Lagasquie

28 septembre 2018

Table des matières

1 Les prérequis

Il faut savoir ce que sont un ordinateur et un système d'exploitation.
Ce document a été écrit en s'appuyant en partie sur [Rif93].

2 Présentation du système

UNIX est un système d'exploitation très répandu dans le monde scientifique. Une de ses particularités consiste dans le fait que c'est un système multi utilisateur et qui, par conséquent, permet à plusieurs personnes de travailler simultanément sur la même machine. De ce fait, c'est également un système multi tâches exécutant plusieurs programmes à la fois.
Plus qu'un système UNIX unique, il existe un ensemble de systèmes UNIX issus du même noyau initial, mais ayant évolué différemment. Deux grandes familles de systèmes existent : UNIX System V et UNIX BSD. Chaque constructeur a adapté un de ces systèmes à son matériel : Solaris chez Sun, AIX pour IBM, ...Linux est la version PC du système UNIX. Ces systèmes ont des noyaux différents mais la norme POSIX (Portable Operating System Interface for computer environment unIX) garantit à l'utilisateur un environnement de travail quasi identique d'un système à l'autre. De plus, le fait qu'à l'origine le logiciel correspondant au système UNIX ait été écrit en langage C a permis d'avoir un système facilement portable.

2.1 Rôle du système d'exploitation

Un ordinateur est un assemblage de composants électroniques. Le système d'exploitation est la couche logicielle chargée de faire fonctionner l'ensemble et de permettre son utilisation. Les trois constituants d'un ordinateur appelés également *ressources de la machine* sont l'unité centrale, la mémoire et les périphériques. Le rôle d'un système d'exploitation quel qu'il soit est de gérer l'utilisation de ces ressources.

- la gestion de l'*unité centrale* permet de contrôler l'exécution des programmes ;
- la gestion de la *mémoire* concerne la mémoire centrale directement sollicitée par l'unité centrale lors de l'exécution de programmes et la mémoire secondaire qui permet le stockage de données enregistrées sous forme de fichier (*système de gestion de fichiers* : SGF) ;
- la gestion des *périphériques* permet à la machine de communiquer avec l'utilisateur via les périphériques d'entrée-sortie (clavier, écran, souris, scanner, imprimante, haut parleur, ...) ou bien avec d'autres machines via l'accès à un réseau de communication.

2.2 Utilisation du système

Du fait de l'aspect multi utilisateur, chaque personne devant utiliser une machine gérée par un système UNIX doit disposer d'un compte attribué par l'administrateur du système. Ce compte est identifié par un login et un mot de passe. Ceux-ci seront systématiquement demandés à l'utilisateur lorsqu'il souhaitera se connecter à la machine pour accéder à ses ressources. Si le système ne peut identifier l'utilisateur, la connexion lui sera refusée.

2.3 Les concepts de base du système UNIX

La notion de processus Le système UNIX étant un système multi tâches, il faut qu'il puisse identifier chaque programme en cours d'exécution. Un *processus* correspond à l'exécution d'un programme à un moment donné. C'est un objet dynamique qui est créé au moment du lancement d'une commande ou d'un programme, qui évolue au fur et a mesure que l'exécution avance et qui est supprimé automatiquement à la fin de l'exécution. Un processus est identifié de manière unique par le système. Il est caractérisé notamment par un numéro unique appelé `PID`. Seuls l'administrateur du système et le propriétaire du processus (utilisateur unix qui a lancé la commande) peuvent agir sur le processus. Un processus peut être interrompu, supprimé, combiné avec d'autres processus.

La notion d'entrée et sortie standard Une particularité du système UNIX est de considérer chaque périphérique comme un fichier logique physiquement relié au dispositif matériel qui constitue le périphérique. Ainsi tout processus lit les données qui lui sont nécessaires dans le fichier logique nommé stdin (*entrée standard*), écrit les résultats qu'il produit dans le fichier logique nommé stdout (*sortie standard*). Si au cours

de l'exécution du processus des erreurs se produisent, les messages correspondants sont écrits sur le fichier logique stderr (*sortie standard d'erreur*).
Par défaut : le fichier logique d'entrée (stdin) est relié au dispositif physique correspondant au clavier, les fichiers logiques de sortie (stdout et stderr) sont reliés au dispositif physique correspondant à l'écran.
De ce fait, il est facile de modifier la source et/ou la destination des données. Ceci peut être fait par le mécanisme de redirection (voir section 5.2 page 9).
De même, on peut "brancher" un processus pour qu'il "lise" les données produites par un autre processus. C'est la communication de processus.
La plupart des commandes externes d'UNIX sont des *filtres*. Elles assurent le traitement d'informations lues sur l'entrée et écrivent les résultats sur la sortie standard ; les éventuels messages d'erreur sont écrits dans la sortie erreur standard. Une commande qui ne ferait qu'une seule de ces opérations de communication lecture ou écriture n'est pas un filtre.

3 Le shell : définitions générales

C'est le premier niveau d'interface entre l'utilisateur et le noyau du système chargé de l'exécution des commandes. Les aspects liés à l'utilisation d'une interface graphique ne sont pas abordés dans ce document. Le but étant de connaître les commandes essentielles à une utilisation convenable du système.
Le *shell* est un *interpréteur de commandes*, c'est-à-dire qu'il exécute les commandes une à une après avoir vérifié leur syntaxe et les avoir traduites en langage machine. Plusieurs types de shells existent, ils sont généralement fonction de la famille UNIX à laquelle appartient le système : `sh`, `ksh`, `bash`, `csh` et `tcsh`. Dans ce document, nous présentons les principales fonctionnalités du shell `csh` dont la particularité est d'utiliser les mêmes opérateurs que le langage C.
Le langage de commandes associé au shell est composé de 2 catégories de commandes qui permettent 2 modes de communication différents avec le système :

- Le mode interactif qui fait appel aux commandes externes qui sont des commandes prédéfinies, utilisables directement et généralement communes aux différents shells (`sh, ksh, csh, tcsh`,...). Les différences se situent au niveau des options de certaines commandes. Lors de l'exécution de ce type de commandes un processus est créé.
- Les *scripts* constituent de petits programmes écrits grâce aux commandes internes. Celles-ci permettent de structurer les commandes de façon à effectuer des tests, des répétitions, ...Un script est donc un fichier de commandes UNIX exécutable directement. Il n'est pas compilé mais directement interprété par le shell. La syntaxe des commandes internes dépend du shell utilisé. Contrairement aux commandes externes, le shell ne crée aucun processus lors de leur exécution. Par contre, un processus est créé globalement pour le script.

4 Principales commandes externes du shell

4.1 Syntaxe générale d'une commande UNIX

```
nom_commande [-option(s)] [argument(s)]
```

nom_commande : il est prédéfini et écrit en règle générale en lettres minuscules. Une commande peut être accompagnée d'options qui modifieront alors son mode de fonctionnement et/ou d'arguments sur lesquels elle s'appliquera (ici, les `[]` indiquent qu'options et arguments sont facultatifs en fonction de la commande).

option(s) : une commande possède généralement plusieurs options. Celles-ci doivent être précédées d'un tiret ('`-`') lors de leur utilisation. Si plusieurs options sont utilisées simultanément, elles peuvent être groupées.

argument(s) : certaines commandes nécessitent des arguments. Ceux-ci doivent alors être spécifiés après les options lorsqu'il y en a.

Exemples d'utilisation de commandes UNIX

`date`	commande seule
`more fichier1`	commande + 1 argument
`cp fich1 fich2`	commande + 2 arguments
`ls -a`	commande + 1 option
`ls -a -F ou ls -aF`	commande + 2 options
`ls -aF repert1`	commande + 2 options + 1 argument

Accéder à la syntaxe d'une commande particulière `?` L'utilisateur peut à tout moment consulter le *manuel d'aide en ligne* pour s'informer ou vérifier la syntaxe d'une commande UNIX ou bien d'une fonction standard du langage C.

`man nom_commande`

Exemples :

`man ls`	donne les caractéristiques de la commande `ls`.
`man scanf`	donne les caractéristiques de la fonction standard C `scanf` ainsi que celles des fonctions appartenant à la même bibliothèque.

4.2 Quelques commandes élémentaires

`who` donne la liste des utilisateurs connectés sur le même système.

`tty` donne le nom du fichier spécial associé au terminal.

`stty` donne la liste des caractéristiques du terminal utilisé.

`stty commande_clavier code_touche` associe à la commande clavier le code de la touche ou combinaison de touches qui permettront à l'utilisateur de l'activer.

`date` affiche la date courante.

`more fichier` affiche le contenu du fichier (mode page/page).

`cat fichier(s)` affiche le contenu du ou des fichiers.

4.3 Commandes de gestion des fichiers (SGF)

4.3.1 Quelques définitions

Fichiers Unix : Sous UNIX la notion de fichier regroupe :
- les fichiers ordinaires : fichiers textes et fichiers binaires,
- les répertoires,
- les fichiers spéciaux : associés aux dispositifs physiques d'entrée-sortie.

Arborescence du SGF : Le SGF est la partie du système UNIX chargée de la gestion des fichiers. Il est basé sur une organisation arborescente où :
- les *noeuds* correspondent systématiquement à des répertoires,
- et les *feuilles* à des fichiers ordinaires ou spéciaux.

Répertoires : Un ensemble de symboles prédéfinis permet à l'utilisateur de faire référence aux différents répertoires qui jouent un rôle spécifique dans l'arborescence du système de gestion de fichiers :

`/`	désigne le répertoire *racine* du système de fichiers,
`.`	désigne le répertoire de *travail* ou répertoire *courant*,
`..`	désigne le répertoire *père* du répertoire de travail,
`~`	désigne le répertoire *d'accueil* de l'utilisateur.

Chemin d'accès : Suite de répertoires qu'il faut emprunter pour accéder à un fichier. Chaque répertoire est désigné par son nom, ou bien par le symbole qui le représente, (cf. ci-dessus) et est séparé du suivant par le caractère `/`.

On parle de :

chemin absolu ou référence absolue si le chemin est donné en partant du répertoire racine (donc débute par le symbole `/`),

chemin relatif ou référence relative s'il est donné à partir du répertoire de travail.

Exemples de chemin d'accès :

`/home/si1g0/TP_UNIX` (chemin absolu),

`../TP_UNIX` (chemin relatif).

ATTENTION ! Le symbole `/` a donc deux interprétations : en tout début de chemin, il représente le répertoire racine ; utilisé à l'intérieur d'un chemin, il est considéré comme un séparateur.

4.3.2 Manipulation de répertoires

Notation : dans les éléments de syntaxe décrits ici, `rep` et `fichier` désigneront respectivement le chemin d'accès (absolu ou relatif) à un répertoire et à un fichier ; `fichier(s)` désignera un ensemble de fichiers.

Créer et supprimer un répertoire : `mkdir` et `rmdir`

`mkdir rep`	crée le répertoire indiqué,
`rmdir rep`	supprime le répertoire indiqué (seulement s'il est vide).

Changer de répertoire de travail : `pwd` et `cd`

`pwd`	affiche la référence absolue du répertoire de travail,
`cd`	pour se positionner sur le répertoire d'accueil,
`cd /`	pour se positionner sur le répertoire racine,
`cd ..`	pour se positionner sur le répertoire père du répertoire courant,
`cd rep`	pour se positionner sur le répertoire indiqué,
`cd ~/rep`	pour se positionner sur le sous-répertoire `rep` du répertoire d'accueil.

Le répertoire mentionné dans la commande `cd` devient le nouveau répertoire de travail. Cette commande permet de parcourir l'arborescence du Système de Gestion de Fichiers.

Visualiser le contenu d'un répertoire : `ls`

`ls`	affiche la liste des noms des fichiers et sous-répertoires contenus dans le répertoire courant,
`ls rep`	même chose, mais pour un répertoire donné.

Principales options :

`a` (all) affiche également les fichiers cachés (commençant par '.'),

`d` (directory) affiche seulement les répertoires,

`F` permet de distinguer les fichiers de données, des répertoires (`/`), des exécutables(`*`) et des liens (`@`),

`i` affiche le numéro d'`inode` (numéro grâce auquel le système identifie le fichier) de tous les fichiers,

`l` (long) affichage détaillé : nom du fichier, droits d'accès, nombre de liens, propriétaire, groupe, taille en octets [1], date de la dernière modification,

`R` affiche récursivement le contenu des sous-répertoires se trouvant dans le répertoire traité,

`t` affiche le contenu du répertoire trié en fonction de la date de modification (la plus récente en tête),

`u` affiche le contenu du répertoire trié en fonction de la date de consultation (la plus récente en tête).

4.3.3 Manipulation de fichiers

Mécanisme d'expansion des noms de fichiers Ensemble de caractères spéciaux qui permettent de spécifier un motif de sélection des fichiers qui seront traités par une commande. Tout fichier dont le nom vérifie le motif est sélectionné.

`?` désigne un caractère quelconque,

`*` désigne une chaîne de caractères quelconque,

1. 1 octet = 8 bits, 1 kibioctet (Kio) = 2^{10} octets = 1024 octets, 1 mebioctet (Mio) = 2^{20} octets = 1024*1024 octets, 1 gibioctet (Gio) = 2^{30} octets = 1024*1024*1024 octets. Il existe aussi les unités suivantes : 1 kilooctet (ko) = 1000 octets, 1 megaoctet (Mo) = 1000*1000 octets, 1 gigaoctet (Go) = 1000*1000*1000 octets. Attention, les affichages ne sont pas toujours conformes (par exemple, utilisation de Mo alors qu'il s'agit en fait de Mio).

`[ ]` définit une liste ou un intervalle de valeurs possibles pour un même caractère. Ex : `[Tt] = T` ou `t`, `[0-9] = 0` ou `1`, ..., ou `9`,

`[! ]` définit un ensemble de valeurs interdites pour un caractère donné (seuls les caractères n'apparaissant pas dans cette liste sont autorisés).

Exemples d'utilisation de ce mécanisme :

`ls *.c`	seuls les fichiers suffixés par `.c` sont affichés,
`rm tp[0-9].exe`	supprime les fichiers dont le nom commence par la chaîne `tp` suivi d'un chiffre entre 0 et 9 et suffixé par `exe`,
`ls [Aa]*`	sélectionne les noms commençant par `A` ou `a`.

Créer un fichier : `cat`

La commande `cat` peut être utilisée de différentes manières. L'une d'elle consiste à créer un fichier à partir du contenu d'un ou de plusieurs fichiers déjà existants (1), ou bien à partir de données saisies au clavier (2). Pour cela il faut utiliser le mécanisme de redirection de la sortie standard (`>` : voir la section 5.2 page 9).

`cat fichier(s) > fichier_cree`	le fichier créé est le résultat de la concaténation des fichiers mentionnés (1).
`cat > fichier_cree`	le fichier créé contient le texte saisi au clavier ;
`.. texte .. ^D`	la séquence `^D` indiquant la fin de la saisie (2).

Remarque : La méthode (2) permet de créer rapidement un fichier de quelques lignes, mais ne présente aucun des avantages proposés par un éditeur de textes.

Remarque : La combinaison `^`x signifie que les touches "control" et la touche correspondant au caractère x sont enfoncées ensemble !

Copier un fichier : `cp`

`cp fichier_source fichier_destination`	copie le fichier source dans le fichier destination,
`cp fichier_source rep_destination`	copie le fichier source dans le répertoire destination et la copie conserve le même nom,
`cp fichier_source rep_destination/nouveau_nom`	copie le fichier source dans le répertoire destination, la copie prenant alors un nouveau nom.

Principales options :

`i` demande confirmation de la copie lorsque le fichier destination existe déjà,

`R` **ou** `r` copie récursive des sous-répertoires et de leur contenu ; les 2 arguments doivent être des répertoires.

Détruire un fichier : `rm`

`rm fichier(s)` supprime les fichiers donnés en paramètre.

Principales options :

`i` demande confirmation de la suppression,

`f` force la suppression sans demander confirmation,

`r` suppression récursive du contenu du répertoire et des sous-répertoires : le sous-arbre associé est supprimé.

Déplacer et/ou renommer un fichier : `mv`

`mv fichier autre_nom`	renomme le fichier et lui attribue un autre nom,
`mv fichier rep`	déplace le fichier sous le répertoire indiqué ; le fichier garde le même nom,
`mv fichier rep/autre_nom`	déplace et renomme le fichier.

Changer les droits d'accès : `chmod` Seul le super-utilisateur ou le propriétaire du fichier peut modifier les droits d'accès. Ceux-ci sont visualisables par la commande `ls -l`.

`chmod mode fichier(s)`	modifie les droits d'accès du ou des fichiers ; le mode est exprimé par une valeur octale ou symbolique.

Une valeur symbolique est composée de 3 parties : les catégories d'utilisateurs concernés par la modification (`u` = propriétaire, `g` = groupe, `o` = autres utilisateurs, `a` = tous), l'opération à effectuer (`+` ajouter, `-` enlever, `=` affecter un droit), le ou les droits concernés (`r` = lecture, `w` = écriture, `x` = exécution).

Créer un lien sur un fichier : `ln` Définition d'un lien : association d'un nom de fichier (chaîne de caractères) et d'un numéro d'inode. Possibilité de définir plusieurs liens sur un même fichier.

`ln fichier autre_nom` crée un nouveau lien associant un autre nom de fichier au numéro d'inode déjà associé au premier fichier (même fichier physique).

Principale option :

`s` le lien créé est un lien symbolique associant non pas le numéro d'inode, mais le chemin d'accès au fichier. Ceci afin de créer des liens avec des répertoires ou bien des fichiers se trouvant sur un autre système de fichiers.

Comparer 2 fichiers ligne à ligne : `diff`

`diff fichier1 fichier2` compare les 2 fichiers et affiche le numéro de la première ligne ainsi que la position du premier caractère qui diffère. Un message signale si un des fichiers correspond au début de l'autre.

Principales options :

`l` toutes les différences sont signalées par le numéro du caractère et sa valeur en octal,

`s` aucune précision n'est donnée sur la différence ; seul un code de retour est renvoyé : 0 si fichiers identiques ; 1 si fichiers différents ; 2 si erreur.

Rechercher un fichier dans le système de fichier : `find`

`find liste_rep liste_expr` procède à la recherche récursive, dans les différents répertoires mentionnés, des fichiers qui satisfont l'ensemble des expressions booléennes spécifiées.

Expressions booléennes de la commande find :

`-name fichier` vrai si le fichier trouvé porte le nom spécifié,

`-perm nbre_octal` vrai si les droits d'accès du fichier trouvé correspondent au nombre octal donné,

`-type val_type` vrai si le type du fichier trouvé correspond à la valeur spécifiée : `b` pour fichier spécial, `d` pour répertoire, `f` pour fichier ordinaire,

`-atime nb_jours` vrai si le fichier a été consulté depuis un certain laps de temps exprimé en nombre de jours,

`-mtime nb_jours` idem, mais pour la date de modification,

`-print` nécessaire sur certains systèmes pour afficher le chemin du fichier si celui-ci correspond aux critères de recherche exprimés.

4.3.4 Traitements sur le contenu des fichiers

Afficher le contenu d'un fichier : `cat` Il s'agit d'une autre utilisation de la commande `cat` (voir les commandes de manipulation de fichiers).

`cat` copie sur la sortie standard les données provenant de l'entrée standard (exemple de filtre).

`cat fichier(s)` copie sur la sortie standard le contenu des fichiers mentionnés (pas de mode page).

Extraire des colonnes de données : `cut`

`cut -c deb-fin` lit des données sur l'entrée standard, extrait les caractères situés entre la position `deb` et la position `fin` et affiche le résultat sur la sortie standard,

`cut -c deb-fin fichier` idem, mais sur le fichier nommé.

On peut extraire plusieurs colonnes en donnant une liste de couples, chacun délimitant une colonne :

```
cut -c deb1-fin1,deb2-fin2, ...
```

Extraire des lignes contenant un motif particulier : `grep`

`grep expr fichier(s)`	recherche dans l'ensemble des fichiers indiqué les lignes contenant le motif spécifié par l'expression régulière `expr` (*) ; chaque ligne sélectionnée est alors affichée sur la sortie standard.

(*) *Expressions régulières* = chaîne de caractères fixe ("chaîne") ou bien écrite à partir des symboles décrits ci-dessous (ex : "`^{*;$`" représente toute ligne commençant par le caractère { et se terminant par un ;).

Symboles utilisés pour l'écriture des expressions régulières :

`.` tout caractère excepté newline,

`*` toute suite de caractères y compris la chaîne vide,

`^` début de ligne,

`$` fin de ligne,

`\<` début de mot,

`\>` fin de mot,

`[ ]` un des caractères de l'ensemble,

`[^ ]` un caractère non inclus dans l'ensemble,

`\` prise en compte d'un caractère spécial comme caractère normal.

Principales options :

`c` affiche le nombre de lignes qui satisfont l'expression,

`i` pas de distinction entre majuscules et minuscules,

`n` chaque ligne est précédée de son numéro dans le fichier qui la contient,

`v` affiche les lignes qui ne satisfont pas l'expression,

`w` la recherche ne porte que sur des mots complets.

Autres commandes similaires : `egrep` et `fgrep` (cf manuel en ligne)

Trier et combiner des données : `sort`

`sort`	trie les différentes lignes de texte lues sur l'entrée standard et affiche le résultat sur la sortie standard ; le tri se fait sur la totalité de la ligne et s'effectue en fonction du code ASCII des caractères,
`sort fichier(s)`	idem, mais appliqué aux fichiers spécifiés,
`sort +nb1 -nb2 fichier(s)`	idem mais le tri se fait sur la portion de ligne allant du mot situé à la position `nb1` (premier mot en position 0) à celui situé en position `nb2` non compris (= tri sur une suite de champs).

Principales options :

`b` ignore les espaces situés en début de ligne,

`d` seuls les lettres, chiffres et espaces sont significatifs,

`f` pas de distinction entre minuscules et majuscules,

`n` tri numérique,

`r` inverse l'ordre du tri,

`t car` permet de définir le caractère `car` comme délimiteur afin d'isoler les mots et de pouvoir faire le tri sur une partie de la ligne,

`o fichier` stocke le résultat du tri dans le fichier mentionné.

Compter les lignes, les mots et les caractères : `wc`

`wc`	affiche le nombre de lignes, de mots et de caractères lus sur l'entrée standard,
`wc fichier(s)`	idem, mais le traitement se fait sur chacun des fichiers mentionnés.

Principales options :

`l` affiche uniquement le nombre de lignes,

`w` affiche uniquement le nombre de mots,

`c` affiche uniquement le nombre de caractères.

Afficher le début d'un fichier : `head`

`head fichier`	affiche les 10 premières lignes du fichier,
`head -nb fichier`	affiche les `nb` premières lignes du fichier.

Afficher la fin d'un fichier : `tail`

`tail fichier`	affiche les 10 dernières lignes du fichier,
`tail +nb fichier`	affiche le contenu du fichier à partir de la ligne portant le numéro `nb` (`+` situe `nb` par rapport au début du fichier),
`tail -nb fichier`	affiche les `nb` dernières lignes du fichier (`-` situe `nb` par rapport à la fin du fichier).

4.4 Principales commandes de gestion des processus et des tâches

Lorsqu'une commande Unix (ou un programme) est exécutée, un processus lui est associé.
Processus = image, à un instant donné, de l'état d'avancement de l'exécution de la commande (ou du programme) à laquelle le processus est associé.

Obtenir la liste des processus : `ps`

`ps` affiche les principales caractéristiques des processus courants appartenant à l'utilisateur qui effectue la commande.

Ces caractéristiques sont :
— `PID` : identifiant du processus,
— `TT` : numéro du terminal d'où le processus a été lancé,
— `STAT` : état du processus,
— `TIME` : temps d'exécution cumulé,
— `COMMAND` : commande associée au processus.
Principales options :

`e` lister l'ensemble des processus actifs (tous utilisateurs confondus),

`u` lister les processus d'un utilisateur donné et identifié par son login,

`l` affiche des informations complémentaires (`UID`, `PPID`, ...),

`f` affiche toutes les caractéristiques des processus.

La liste des options varie suivant le système utilisé, pour plus de détails consulter le manuel en ligne : `man ps`.

Interrompre un processus : `intr`, `quit`

`intr (taper ^C)`	arrêt du processus de premier plan,
`quit (taper ^\)`	idem, avec en plus génération d'un fichier `core` (image mémoire au moment de l'interruption).

Supprimer un processus : `kill` Cette commande s'applique à tout processus quel que soit son état : premier ou arrière plan, suspendu (pour plus de détails voir paragraphe suivant).

`kill -signal PID`	envoie un signal au processus identifié par son `PID` (les signaux sont identifiés par des numéros ou des chaînes de caractères (15 = `TERM`, 9 = `KILL`, ...),
`kill -l`	donne la liste des signaux disponibles,
`kill PID`	envoie au processus mentionné le signal fin de processus (15, `TERM`) par défaut (ce signal peut ne pas être suffisant),
`kill -9 PID`	envoie le signal indiquant qu'il faut supprimer le processus identifié par son `PID`.

Suspension d'un processus : `susp`

`susp (taper ^Z)`	suspend le processus de premier plan ; celui-ci est alors en attente d'être supprimé (`kill`) ou d'être relancé (`fg, bg`).

Lancement d'un processus en tâche de fond

`&` opérateur de lancement en tâche de fond.

Le système n'attend pas la fin de l'exécution pour redonner la main à l'utilisateur (voir ci-après pour plus de détail sur la notion de tâche de fond – tâche en arrière plan).

Le mécanisme de gestion des tâches est associé à un type d'interprète shell. Celui-ci n'est pas disponible dans tous les shells : existe dans `csh` ; pas dans `sh`.

Tâche (ou job) = correspond à un processus ou à un groupe de processus si ceux-ci sont combinés par l'opérateur `|` (processus concurrents : voir section 5.3 page suivante). La notion de tâche est définie par rapport à un shell donné. Chaque tâche est identifiée par un numéro d'identification local au shell de lancement de la commande associée.

Désignation d'une tâche = soit par le `PID` associé, soit par le numéro de tâche. Pour les distinguer l'un de l'autre, le numéro d'une tâche doit être précédé de `%` :

`%1` désigne la tâche dont le numéro dans le shell courant est [1].

État d'une tâche = caractérise l'activité de la tâche et son contrôle possible par l'utilisateur :

— premier plan (ou "foreground") : tâche active interruptible directement par l'utilisateur (voir les paragraphes précédents pour les commandes d'interruption),
— arrière plan (ou "background") : tâche active, mais ne peut pas être interrompue directement par l'utilisateur,
— suspension : tâche inactive attendant soit d'être relancée (en premier ou arrière plan), soit d'être supprimée.

Afficher la liste des tâches liées au shell courant : `jobs`

`jobs`	affiche le numéro (entre `[]`), l'état, la commande associée à chaque tâche du shell courant ; la tâche courante est marquée par `+`,
`jobs -l`	affiche en plus le `PID` du processus associé.

Passage au premier plan : `fg` Ce changement d'état s'applique à une tâche suspendue ou bien à une tâche en arrière plan.

`fg`	la tâche courante (marquée par `+`) est placée au premier plan,
`fg %num_tâche`	la tâche désignée par son numéro local est placée au premier plan.

Passage en arrière plan : `bg` Ce changement d'état s'applique à une tâche suspendue ou bien en premier plan.

`bg`	la tâche courante (marquée par `+`) est placée en arrière plan,
`bg %num_tâche`	la tâche désignée par son numéro local est placée en arrière plan.

Suppression d'une tâche : `kill` Cette commande d'arrêt s'applique à toute tâche quel que soit son état (voir le paragraphe sur la suppression des processus pour plus de détail sur la commande `kill`).

`kill %num_tâche`	la tâche désignée par son numéro local est supprimée.

Exécution différée L'exécution d'un fichier de commandes peut être lancée à une date précise.

`at heure date fic_com`	exécute le fichier de commandes à la date et à l'heure données ; l'heure peut comporter 4 chiffres et la date est de la forme mois jour (ex : jan 4).

5 Enchaînement, redirection et communication de processus

Philosophie UNIX : réutiliser les outils existants en les combinant pour effectuer une tâche plus complexe.

5.1 Enchaînement de processus

L'opérateur `;` est utilisé pour enchaîner 2 processus. Il s'agit du lancement séquentiel et indépendant des différents processus : le deuxième processus n'est lancé qu'après la fin de l'exécution du premier, ...

```
commande1 ; commande2 ;..; commandeN
```

5.2 Redirection des fichiers standards

Redirection de l'entrée standard d'un processus

L'opérateur de redirection de l'entrée standard est `<`. Les données qui doivent être traitées par le processus ne sont pas lues sur le clavier, mais dans le fichier mentionné.

```
commande < fichier_entree
```

Redirection de la sortie standard d'un processus

L'opérateur `>` est utilisé pour rediriger la sortie standard. Le résultat obtenu n'est pas écrit sur l'écran mais dans le fichier mentionné. Si celui-ci n'existe pas, il est créé, sinon son contenu est effacé (si le fichier existe déjà, en fait deux cas peuvent se présenter : soit il est écrasé, soit la création du fichier est refusée. Il faut alors utiliser `>!` pour forcer l'opération d'écrasement du fichier).

```
commande > fichier_sortie
```

Exemple de redirection

```
cat fic1 fic2 > fic3
```

Concatène le contenu de `fic1` et `fic2` et met le résultat dans `fic3` au lieu de l'afficher à l'écran.

Utiliser le symbole `>>` pour concaténer le résultat à la fin d'un fichier déjà existant.

Exemple de redirection avec concaténation

```
cat fic2 >> fic1
```

Concatène le contenu `fic2` à la fin de `fic1`. Le contenu de `fic1` est modifié.

Redirection de la sortie erreur standard d'un processus

L'opérateur pour la redirection de la sortie d'erreur standard est `>&` . Les messages d'erreur résultant de la mauvaise exécution de la commande sont écrits dans le fichier mentionné. Si le contenu du fichier ne doit pas être écrasé, utiliser `>>&`.

```
commande >& fichier_erreur
```

Exemple de redirection de la sortie d'erreur standard

```
cc prog.c >& erreur_compil
```

Les erreurs de compilation issues de la commande `cc` sont récupérées dans le fichier `erreur_compil`.

5.3 Communication de processus (tube ou pipe)

L'opérateur `|` est utilisé pour faire communiquer 2 processus. C'est-à-dire, le lancement simultané de plusieurs processus communiquant entre eux deux à deux, par l'intermédiaire d'un tube de communication (ou "pipe"). Le système assure la synchronisation des processus. Les processus sont dits concurrents.

Tube = dédié à la communication entre 2 processus. La sortie standard du premier processus est redirigée sur l'entrée du tube et l'entrée standard du second est redirigée sur la sortie du tube. Le résultat du premier processus sert donc d'entrée au second.

```
commande1 | commande2 |..| commandeN
```

Ceci n'a de sens que si les commandes 2 à `N` lisent effectivement sur leur entrée standard et que les commandes 1 à `N-1` écrivent sur leur sortie standard. Les commandes 2 à `N-1` sont donc des filtres.

Exemple de communication entre processus

```
head -9 fic_test | tail +5
```

Affiche les lignes 6 à 9 du fichier `fic_test` ; le premier processus extrait les 9 premières lignes du fichier `fic_test` ; la seconde commande ne s'applique pas à un fichier mais au résultat du premier processus.

6 Commandes internes du shell (C-Shell)

`man nom_shell`	donne la liste des commandes internes relatives au shell indiqué (`csh, tcsh, sh, ksh`, ...sont les noms respectifs des processus réalisant l'exécution des interpréteurs de commandes C-Shell, Tc-Shell, Bourne-Shell, Korn-Shell,...).

Les commandes internes font du langage de commandes associé au shell un véritable langage de programmation.

Script Shell = programme effectuant un traitement particulier et composé d'une suite d'instructions utilisant des variables, des expressions, des commandes, des paramètres et des structures de contrôle.

6.1 Variables Shell prédéfinies

Les variables Shell se répartissent en deux catégories. Certaines d'entre-elles sont dédiées à la configuration du shell et d'autres permettent à l'utilisateur de modifier son environnement de travail en fonction de ses préférences.
Important : Si `nom_var` représente le nom d'une variable shell, `$nom_var` représente sa valeur (mécanisme de substitution).

Variables de configuration Ces variables sont locales au shell. Suivant les valeurs qui leur sont attribuées ou leur état d'activation, ces variables modifient le comportement du shell. Elles doivent être initialisées à chaque lancement de celui-ci. Pour que cela soit fait systématiquement, les instructions d'initialisation sont généralement incluses dans le fichier lancé au démarrage du shell (ici `.cshrc`). Il y a 2 sortes de variables de configuration : les variables de type "commutateur" et celles contenant une valeur.

Variables de type "commutateur" Elles n'ont pas de valeur proprement dite, elles sont seulement activées ou désactivées. Signification des principales variables lorsqu'elles sont actives :

`echo` affiche à l'écran chaque commande avant son exécution,

`filec` permet la complétion automatique des noms (utilisation de la touche `Esc`),

`noglob` interdit l'utilisation du mécanisme d'expansion des noms de fichiers,

`ignoreeof` interdit la déconnexion par `^D`,

`nobeep` aucun bip n'est émis lors de la détection d'ambiguïté dans la complétion automatique,

`nonomatch` l'échec d'une expansion de noms de fichiers ne provoque pas d'erreur,

`notify` signale la fin d'exécution d'un processus,

`verbose` affiche la commande complète après substitution dans l'historique.

Commandes de manipulation des variables "commutateurs" :

`set nom_var`	active la variable,
`unset nom_var`	désactive la variable,
`set`	affiche la liste des variables actives.

Variables contenant une valeur Elles n'existent que si une valeur leur est affectée soit par l'utilisateur soit directement par le système. Signification des principales variables lorsqu'elles sont définies :

`argv` liste des arguments de la commande courante,

`cdpath` liste des répertoires à parcourir pour trouver un fichier,

`cwd` référence absolue du répertoire de travail,

`home` référence absolue du répertoire d'accueil de l'utilisateur,

`path` liste des répertoires à parcourir pour trouver une commande ou un programme,

`prompt message` affiché par le shell pour indiquer qu'il attend une commande ('`%`' par défaut pour `csh`),

`shell` chemin d'accès au programme shell,

`status` code de retour de la dernière commande exécutée,

`term` type de terminal utilisé,

`history` nombre de lignes de commandes mémorisées dans l'historique,

`savehist` sauvegarde de commandes extraites de l'historique en vue de leur réutilisation lors d'autres sessions.

Les variables `cwd`, `argv`, `status` sont directement mises à jour par le shell, les autres doivent être définies par l'utilisateur dans un fichier de démarrage ou au cours de la session.
Commandes de manipulation de ces variables :

`set nom_var=valeur`	définit la variable et lui affecte une valeur,
`unset nom_var`	supprime la variable,
`set`	affiche la liste des variables définies,
`echo $nom_var`	affiche la valeur de la variable (mécanisme de substitution).

Ces commandes permettent également à l'utilisateur de définir, affecter, afficher et supprimer toute variable qui lui est nécessaire. Exemple :

```
set NB = 5 ; echo $NB ; unset NB
```

Variables d'environnement Reconnaissables parce qu'écrites en majuscules, ces variables sont utilisées pour communiquer des informations aux autres programmes ou à n'importe quel autre shell. Elles sont donc "exportables". Leur valeur doit être définie à la connexion. Par conséquent, les instructions d'initialisation sont en règle générale incluses dans le fichier lancé au démarrage (.login).
Les variables d'environnement sont modifiables par l'utilisateur, cependant certaines étant liées aux variables de configuration portant le même nom, toute modification de la valeur de l'une entraîne la modification de la valeur de l'autre. C'est le cas des variables : `HOME, PATH, SHELL, TERM, USER`,...
Signification d'autres variables d'environnement :

`LOGNAME` contient le nom de l'utilisateur,

`MANPATH` contient la liste des répertoires donnant accès aux pages du manuel d'aide en ligne,

`DISPLAY` utilisée lorsque l'utilisateur souhaite que l'affichage s'effectue sur un écran autre que celui de la machine ou du terminal où s'exécute le programme (environnement graphique). La variable doit alors contenir l'adresse de l'écran où l'affichage doit s'effectuer.

Commandes de manipulation de ces variables :

`setenv NOM_VAR valeur`	définit la variable et lui affecte une valeur (attention : pas de = entre la variable et la valeur!!!),
`setenv NOM_VAR`	supprime la valeur de la variable,
`unsetenv NOM_VAR`	supprime la variable,
`printenv` ou `setenv`	affiche la liste des variables définies,
`printenv NOM_VAR`	affiche la valeur de la variable,
`echo $NOM_VAR`	idem (mécanisme de substitution).

6.2 Expressions arithmétiques

Évaluation d'une expression L'opérateur `@` est utilisé pour évaluer une expression.

```
@ nom_var op_affectation expression
```

La valeur de l'expression est évaluée puis affectée à la variable `nom_var`. Toute définition de variable accompagnée de l'affectation d'une valeur calculée doit être faite à l'aide de `@` et non de `set`.
Les opérateurs arithmétiques utilisés dans les expressions (1) et les opérateurs d'affectation (2) sont identiques à ceux utilisés dans le langage C :

(1) : `+, -, *, /, %, >, >=, <, <=, ==, !=, !, &&, ||, ...`
(2) : `=, +=, -=, *=, /=, %=`

Exemples d'instructions d'évaluation

```
set X = 5
```
définition de la variable X et affectation de la valeur 5

```
@ Y = $X * 2
```
définition de la variable Y et affectation de la valeur obtenue en multipliant par 2 la valeur de X

```
@ X += $Y
```
modification de la valeur de la variable X à laquelle on ajoute la valeur de Y

Règles d'évaluation

- tout nombre commençant par `0` est considéré comme un nombre octal,
- tout argument absent est nul par défaut,
- tout résultat est décimal,
- tous les composants d'une instruction d'évaluation doivent être séparés par des caractères d'espacement,
- la valeur booléenne *faux* est représentée par 0 et *vrai* par 1.

Statut d'un fichier Certaines caractéristiques associées aux fichiers peuvent être testées grâce à l'utilisation d'une expression constituée d'un tiret ('-') suivi d'un caractère spécifiant le test à effectuer sur le fichier mentionné.

```
-car_test nom_fic
```

Le résultat d'une telle expression est une valeur booléenne. Signification du test effectué suivant la valeur de `car_test` :

- `d` le fichier est un répertoire,
- `e` le fichier existe,
- `f` le fichier est un fichier ordinaire,
- `o` l'utilisateur est le propriétaire du fichier,
- `r` l'utilisateur a l'accès en lecture sur le fichier,
- `w` l'utilisateur a l'accès en écriture sur le fichier,
- `x` l'utilisateur a l'accès en exécution sur le fichier,
- `z` le fichier est vide.

Exemple :

`-e fich_test`
1 (vrai) si le fichier `fich_test` existe
0 (faux) sinon

`-o fictest`
1 (vrai) si l'utilisateur est propriétaire du fichier `fich_test`
0 (faux) sinon

6.3 Mécanismes de base

Alias Surnom utilisé comme raccourci d'une commande ou d'un enchaînement de commandes particulier. Pour être connu dans chaque processus shell, les alias peuvent être définis dans le fichier de configuration (ici .cshrc).
Commandes de manipulation des alias :

`alias nom_alias commande`	crée ou modifie un alias en l'associant à la commande spécifiée,
`alias`	affiche la liste des alias déjà définis,
`alias nom_alias`	affiche l'alias existant,
`unalias nom_alias`	supprime l'alias.

Un alias peut être paramétré et ses paramètres intégrés à la commande qui lui correspond.
Symboles désignant les paramètres d'un alias :

`\!^`	premier paramètre	`\!$`	dernier paramètre
`\!*`	tous les paramètres	`\!:nb`	paramètre de rang nb

Exemple de définition d'un alias :

```
alias h history
```

définition de l'alias `h` comme raccourci de la commande history

```
alias cd 'cd \!:1; pwd'
```

définition de l'alias `cd` comme l'enchaînement de la
commande `cd` appliquée au premier paramètre de l'alias
et de la commande `pwd`
`cd rep` équivaut alors à `cd rep ; pwd`

Historique Possibilité de mémoriser les dernières lignes de commandes.

`set history = nb`	activation du mécanisme d'historique avec conservation des `nb` dernières commandes,
`history`	donne la liste des lignes de commandes mémorisées.

Chaque ligne de commande mémorisée est désignée par son numéro d'ordre. Celui-ci est calculé par le shell et incrémenté de un à chaque nouvelle commande.
Rappel d'une commande mémorisée :

`!!`	dernière commande,
`!ch`	dernière commande commençant par la chaîne `ch`,
`!nb`	commande désignée par son numéro d'ordre,
`^ch1^ch2`	dernière commande dans laquelle on substitue la chaîne `ch1` par la chaîne `ch2`.

Complétion des noms de fichiers La variable de configuration `filec` doit être active. La touche `Esc` permet alors de compléter le nom dont l'écriture a été ébauchée. Si la variable de configuration `nobeep` est désactivée, toute ambiguité sera signalée par un bip sonore.

Substitution Mécanisme qui consiste à substituer par une valeur particulière toute expression commençant par le caractère `$`.
Expressions et valeurs de substitution :

`$0`	nom de la commande en cours d'exécution,
`$$`	numéro du processus shell en cours,
`$<`	ligne lue sur l'entrée standard,
`$var`	valeur de la variable `var`,
`$?var`	0 si la variable n'est pas définie, 1 sinon.

Si `var` est une liste ou un tableau alors :

`$var[I]`	Ième élément de la variable `var`,
`$var[I-J]`	sous-liste allant du Ième au Jème élément de `var`,
`$var[*]`	tous les éléments de la liste `var`,
`$#var`	nombre d'éléments de la variable `var`.

Délimiteurs de chaînes de caractères L'interprétation d'une chaîne de caractères est différente suivant le caractère employé pour la délimiter : quote (`'`), double-quote (`"`) ou backquote (`‘`).
Délimiteurs et interprétation :

`'chaine'`	les caractères dits spéciaux sont considérés comme des caractères normaux,
`"chaine"`	les caractères `$`, `\`, `'` et `‘` sont considérés comme des caractères spéciaux,
`‘chaine‘`	la chaîne est considérée comme une commande. Celle-ci est exécutée par un sous-shell et la chaîne est remplacée par le résultat de la commande.

Exemples d'utilisation des délimiteurs de chaînes de caractères :

```
set X=5
```

```
echo 'X = $X'
```

pas de substitution ; X = $X est affiché tel quel

```
echo "X = $X"
```

$X est substitué par 5 ; X = 5 est affiché

```
echo ‘set X=3 ; echo $X‘
```

la chaîne est considérée comme une commande ;
elle est exécutée et remplacée par son résultat : 3,
la nouvelle valeur de X est affichée

7 Fichiers de commande ou Scripts shell

Un script shell est conçu pour effectuer une commande complexe résultant de la combinaison de commandes plus simples. Il peut faire appel à des commandes externes, des commandes internes (structures de contrôles) ou d'autres scripts shell.
Un script shell est donc un programme. Il peut être commenté (les commentaires étant introduits par le caractère `#`) et surtout paramétré.

7.1 Paramètres d'un script shell

La variable prédéfinie `argv` contient la liste des paramètres de la commande en cours d'exécution. Il suffit de lui appliquer le mécanisme de substitution pour accéder à un paramètre précis (voir le paragraphe sur la substitution dans la section 6.3 page 13).

Désignation des paramètres dans un script shell

`$argv` ou `$*`	représente la liste des paramètres de la commande en cours d'exécution,
`$argv[I]` ou `$I`	correspond au Ième paramètre de la liste représentée par `argv` ; si I=0 il s'agit du nom du script en cours d'exécution,
`$#argv`	nombre de paramètres.

7.2 Structures de contrôle

Si leur fonctionnement est proche des instructions de contrôle du langage C, leur syntaxe diffère quelque peu. En particulier, il faut noter que dans la majorité des cas, un mot-clé indique la fin de la séquence d'instructions qui relève de la structure de contrôle : `endif`, `end`, `endsw`.

Instruction conditionnelle simple

```
if (expression) commande
```

Si l'expression est vraie (différente de 0), la commande est exécutée ; celle-ci doit être une commande simple. Dans ce cas le mot `then` est omis.

Alternative et alternatives imbriquées

```
if (expression) then
...
else [ if(expression2) then
...
else
...]
endif
```

Si l'expression relative à un `if` donné est vraie (différente de 0), les commandes situées entre les mots-clés `then` et `else` correspondants sont exécutées, sinon ce sont celles situées entre `else` et `endif` qui le sont.
Attention : Les mots-clés `if`, `else` et `endif` doivent être écrits en début de ligne et le mot-clé `then` doit terminer la ligne (1ère instruction sur la ligne suivante).

Instructions itératives

```
foreach var (liste_valeur)
...
end
```

La variable `var` prend successivement chacune des valeurs de la liste `liste_valeur` et la séquence d'instructions terminée par le mot-clé `end` est exécutée pour chaque nouvelle valeur de `var`.

```
repeat nombre commande
```

La commande est exécutée un nombre donné de fois. Il s'agit obligatoirement d'une commande simple.

```
while (expression)
...
end
```

Tant que l'expression est vraie (différente de 0), les commandes situées entre le `while` et le `end` sont exécutées.
Attention : Les mots-clés `while` et `end` doivent être écrits en début de ligne.

Interrompre l'exécution d'une boucle `foreach` ou `while`

`break`	interruption de la boucle et reprise après le `end`,
`continue`	interruption et reprise au début de la boucle (itération suivante).

Sélection Fonctionnement similaire à l'instruction correspondante dans le langage C.
Attention : Les mots-clés `case`, `default` et `endsw` doivent être écrits en début de ligne.

```
switch (chaine)
case etiquette1 :
...
breaksw
case etiquette2 :
...
breaksw
...
default :
...
breaksw
endsw
```

`breaksw` permet de reprendre l'exécution après le `endsw`.

Autres commandes internes

`echo texte`	permet l'affichage d'une ligne de texte sur la sortie standard,
`exit [expr]`	provoque la sortie du shell en renvoyant soit la valeur de l'expression si elle spécifiée, soit la valeur de la variable `status`.

En complément, se référer au manuel en ligne : `man csh`.

7.3 Exécution d'un script shell

À la différence d'un programme exécutable directement issu de la compilation d'un programme source, un script shell n'est pas un fichier binaire mais un fichier texte qui est interprété par le shell. Par conséquent, il ne possède pas initialement le droit d'accès en exécution. Suivant le cas, la méthode de lancement d'un script shell est différente.

Exécution d'un script shell avec droit en lecture seulement Le lancement ne se fait pas directement. Il faut utiliser une des commandes suivantes en passant en paramètre le nom et, lorsque c'est possible, les paramètres du script.

`csh nom_script [paramètre(s)]`	exécution par un sous-processus `csh`, fils du processus de lancement,
`source nom_script`	exécution par le processus shell courant utilisé notamment pour l'exécution de fichiers système (.login, `.cshrc`, ...),
`exec nom_script [paramètre(s)]`	recouvrement par un autre shell `csh`.

Exécution d'un script shell avec droit en exécution Le nom du script est directement utilisé comme une commande.

```
nom_script [paramètre(s)]
```

Tout dépend à ce moment-là du contenu de la première ligne. Celle-ci détermine alors le mode d'interprétation du script.

`#! chemin_shell`	exécution par le shell dont on précise le chemin d'accès,
`# texte`	exécution par un sous-processus `csh`,
`pas de commentaire`	exécution par un sous-processus `sh`.

Un script écrit avec le langage de commandes associé à C-Shell doit être exécuté par un processus C-shell (`csh`). Il ne peut pas être exécuté par un processus Shell (`sh`) et réciproquement car la syntaxe des commandes internes est différente selon le shell employé.

8 Exercices

8.1 Les commandes liées au SGF

Utiliser les commandes adéquates pour :

1. Afficher la date du jour.
2. Visualiser la liste des utilisateurs connectés.
3. Consulter le manuel pour obtenir la syntaxe exacte de la commande `ls`.
4. Afficher la référence absolue de votre répertoire de travail.
5. Afficher le contenu de ce répertoire.
6. Créer le fichier de nom exemple et contenant 2 à 5 lignes de texte.
7. Copier le fichier exemple dans le fichier exemple.txt.
8. Créer le répertoire TP_UNIX et y déplacer le fichier exemple.txt.
9. Afficher le contenu du répertoire courant en utilisant l'option qui permet de distinguer les fichiers des répertoires.
10. Afficher le contenu du répertoire TP_UNIX en utilisant l'affichage long.
11. Changer de répertoire de travail et se positionner sur TP_UNIX.
12. Répéter les questions (4) et (5).
13. Afficher le contenu du fichier exemple.txt.
14. Revenir dans votre répertoire privé.
15. Créer 2 nouveaux fichiers appelés respectivement fic1.ex et fic2.ex.
16. Lister les noms des fichiers du répertoire de travail qui contiennent la chaîne de caractères ex.
17. Lister les noms des fichiers du répertoire de travail qui sont suffixés par ex.
18. Supprimer le fichier exemple.
19. Créer le répertoire EXEMPLE.
20. Déplacer le fichier fic1.ex dans ce répertoire.
21. Déplacer, en une seule commande, le fichier fic2.ex dans EXEMPLE en le renommant fichier.ex.
22. Se positionner sur le répertoire EXEMPLE et lister son contenu.
23. Sans changer de répertoire de travail, lister le contenu du répertoire TP_UNIX.
24. Aller sous le répertoire racine et lister son contenu.
25. Revenir sur votre répertoire privé.

8.2 Les commandes externes du shell

Redirection de fichiers standards

1. Créer un fichier de nom fictrav et contenant la ligne de texte "premiere ligne".
2. Utiliser le mécanisme de redirection pour ajouter les lignes "deuxième ligne" et "troisième ligne" à ce fichier.
3. Sachant que la commande `echo` affiche a l'écran la ligne de texte qui lui est fournie en paramètre, comment l'utiliser pour ajouter une "quatrième ligne" dans le fichier fictrav.
4. Récupérer, dans le fichier contenu_rep, le contenu en format long du répertoire courant.
5. Envoyer ce fichier par courrier électronique (mail) à un autre utilisateur d'adresse *adresse1*.

Modification des droits d'accès

1. Donner l'accès en lecture sur le fichier fictrav aux membres de votre groupe.
2. Consulter le fichier fictrav d'un autre utilisateur du groupe (*nom_user*). Essayer de modifier son contenu. Y-a t'il des problèmes ?
3. Remédier au problème survenu à la question précédente et essayer à nouveau de modifier le fichier fictrav du voisin.
4. Enlever si ce n'est pas déjà fait les droits d'accès sur tous vos fichiers et répertoires aux utilisateurs n'appartenant pas au groupe.

Traitements sur les fichiers

1. Combien y a-t-il de lignes dans le fichier contenu_rep ?
2. Créer 2 nouveaux fichiers non vides appelés respectivement fic1 et fic2
3. Créer en une seule ligne de commande le fichier temp en concaténant les 3 premières lignes d'un fichier fic1 et les 5 dernières lignes d'un fichier fic2.
4. Créer le fichier fic3 en concaténant le contenu d'un fichier fic1 et d'un fichier fic2.

Recherche d'un "motif" dans le contenu de fichiers existants

1. Se placer dans le répertoire TP_UNIX. Puis rechercher le mot "fictrav" dans les fichiers situés dans le répertoire père du répertoire courant.
2. Même chose :
   — sachant que le mot peut être écrit en majuscules ou en minuscules,
   — en précisant les numéros des lignes où apparaît le mot recherché.
3. Revenir sous votre répertoire d'accueil et créer le fichier coordonnees contenant vos nom, prénom et email.
4. Ajouter plusieurs lignes de la forme "nom prenom email" au fichier coordonnees.
5. Rechercher les coordonnées complètes à partir d'un nom.

Copie, déplacement et suppression de fichiers

1. Dans le répertoire d'accueil, créer les répertoires REP1, REP2 et REP3 ; puis se positionner sur le répertoire REP1. Effectuer toutes les opérations suivantes depuis ce répertoire de travail.
2. Copier dans le répertoire courant le fichier coordonnees situé dans votre répertoire d'accueil en nommant la copie coord_bis.
3. Déplacer coord_bis dans le répertoire REP2.
4. Déplacer le fichier coordonnees du répertoire d'accueil vers le répertoire REP3 en le renommant coord_ter.
5. Supprimer tous les fichiers qui se trouvent dans le répertoire REP2.

Tri du contenu d'un fichier

1. Créer le fichier etudiant contenant plusieurs lignes, chaque ligne étant composée des 3 champs (nom, prénom, age) avec le caractère : étant utilisé comme séparateur.
2. Afficher le contenu de ce fichier trié par ordre alphabétique.
3. Trier ce fichier selon l'âge (ordre croissant) et récupérer le résultat du tri dans le fichier etud_age.
4. Modifier le contenu du fichier etud_age (depuis un éditeur) de façon à ce que les colonnes nom, prénom et age soient bien alignées.
5. Afficher le nom et l'âge de chaque étudiant enregistré dans ce fichier.
6. Afficher seulement les noms et prénoms.
7. Récupérer la liste des étudiants (nom, prénom) dans le fichier liste_etud.

Processus concurrents

1. Envoyer par mail, à un utilisateur d'adresse *adresse1*, le contenu en format long de votre répertoire de travail.
2. Combien de fichiers se trouvent dans votre répertoire courant ?
3. Afficher les informations relatives aux fichiers qui se trouvent dans votre répertoire de travail et qui ne sont pas des répertoires.
4. Même question que précédemment mais en se limitant aux fichiers qui sont exécutables pour le propriétaire et en récupérant le tout dans le fichier info_exec.
5. Reprendre la question 6) de l'exercice précédent sur les étudiants en affichant le résultat trié par ordre alphabétique.
6. Reprendre la question 7) de l'exercice précédent sur les étudiants mais en récupérant le résultat trié par ordre alphabétique dans le fichier liste_etud2.
7. Afficher à partir du fichier etudiant les noms et prénoms des 3 étudiants les plus âgés.

Processus en tâche de fond et Job control

1. Lancer un éditeur de texte depuis votre fenêtre de commande (`gedit` par exemple).
2. Sans fermer la fenêtre de l'éditeur, lancer la commande `jobs`. Que se passe-t'il ?
3. Fermer l'éditeur. Que constatez-vous ?
4. Lancer à nouveau l'éditeur en le mettant en arrière plan. Recommencer 2).
5. Consulter la liste des processus en cours d'exécution, puis celle des tâches (avec `PID`). Comparer les 2 listes. Consulter la liste des processus (avec `PPID`).
6. Fermer l'éditeur et recommencer 5).
7. Créer le répertoire TP_C et s'y placer, saisir et compiler le programme C suivant (fichier boucle.c) :

```
void main(void)
{ while (1); }
```

8. Consulter la liste de vos fichiers pour connaître le nom de l'exécutable généré. Lancer l'exécution de ce programme en tâche de fond.
9. Consulter la liste des processus et noter le numéro du processus correspondant à l'exécution que vous venez de lancer.
10. Supprimer ce processus.
11. Lancer à nouveau ce programme, mais cette fois-ci au premier plan.
12. Suspendre l'exécution.
13. Visualiser la liste des jobs (avec `PID`) et noter l'état du processus.
14. Relancer le job au premier plan et effectuer à nouveau 12) et 13).
15. Relancer le job en tâche de fond et effectuer la question 13) à nouveau.
16. Supprimer le job et effectuer 13) à nouveau.

8.3 Programmation Shell

Considérons le shell `csh`.

Expressions arithmétiques et script shell

1. Quelles sont les lignes de commandes qui permettent d'effectuer les opérations suivantes :

   `X = 5, Y = 3, Z1 = X + Y et Z2 = X * Y`

   et d'afficher le contenu de `Z1` et de `Z2`.
2. Ecrire un script shell qui permet d'effectuer le traitement décrit ci-dessus.
3. Partant de ce script, écrire les deux versions suivantes :
   — Version 1 : les valeurs de X et de Y sont saisies au clavier,
   — Version 2 : les valeurs de X et de Y sont données en paramètre lors de l'exécution du fichier de commandes.

Script shell et statut des fichiers

1. Ecrire le script shell `conv_taille` qui, pour un fichier donné en paramètre, affiche la taille du fichier en octets et effectue la conversion de celle-ci comme le montre l'exemple suivant :

```
conv_taille fic1
------- Taille en octets du fichier fic1
       (1234567) soit : 1Mio, 181 Kio et 647 octets
```

   Un message sera également affiché dans les cas suivants :
   — le fichier n'existe pas,
   — le fichier est vide,
   — le fichier est un répertoire : dans ce cas, la taille n'est pas affichée.

2. Modifier ce script de façon à ce que, lorsqu'il s'agit d'un répertoire, la taille affichée corresponde réellement à la somme des tailles des fichiers qui appartiennent au répertoire en question.

Références

[Rif93] Rifflet (Jean-Marie). – *La programmation sous UNIX*. – Ediscience international, Mc Graw Hill, 1993. ISBN : 2-84074-013-3.

Index pour le système UNIX

T

U