Support du cours

Système d'exploitation UNIX/Linux - Scripting en Perl/Python

Pierre Pompidor
-
LIRMM, 161 rue Ada,
34 392 Montpellier Cedex 5
téléphone : 04 67 41 85 36, fax : 04 67 41 85 00
(ces numéros de téléphone apparaissent ici à titre décoratif)
adresse électronique : **pompidor@lirmm.fr**
URL : http://www.lirmm.fr/~pompidor (le caractère mystérieux avant pompidor est un tilda)

Ce polycopié accompagne la première partie du cours **Système d'exploitation/Unix/Linux** malheureusement dispensé à Polytech et en Licence et Master d'Informatique.

Un système d'exploitation consiste en un **ensemble de programmes** (terme vague pour décrire le maquis de commandes éventuellement interfacées sur le bureau, de bibliothèques obscures, de démons incontrôlables, de pilotes récalcitrants, ...) qui se terrent dans votre machine), ces programmes permettant d'utiliser un ordinateur pour une myriade de tâches (pour développer de nouveaux programmes, faire de la bureautique, employer des logiciels ingrats, communiquer, gérer d'autres ordinateurs ou bien entendu jouer), cet ensemble étant plus ou moins riche suivant ce qui est offert (ou chèrement acquis ?) sur le système employé (Unix, Windows, MacOs ...).
Ce plaisant préambule appelle immédiatement les remarques suivantes :

- d'une part, ce cours (et a fortiori ce polycopié), ne prétend pas du tout couvrir l'ensemble de ces programmes (ce qui dénoterait un détachement définitif d'une vie sociale normale), mais simplement donner quelques informations sur les commandes (à exécuter sur un terminal) les plus utiles pour essayer d'amadouer un ordinateur géré par **Linux**. A la question, "pourquoi Linux ?" (qui est un des "portages" d'Unix sur micro-ordinateurs), je répondrai d'une part que c'est le système installé sur la quasi-totalité des machines des salles de TP du département informatique, et d'autre part que ce système vous permettra de pratiquer tous les langages et quasiment toutes les technologies nécessaires à une vie professionnelle heureuse, que celle-ci soit émaillée de rencontres avec des machines sous Linux ou sous Windows. Par ailleurs, ce système nous permet de faire des économies qui frisent l'indécence. A savoir pourquoi apprendre des commandes et non pas une manipulation experte et envoûtante de la souris (ce qui est bien entendu aussi possible sous Linux), je répondrai que non seulement tout n'est pas interfacé par un bureau, et que de toute façon cliquer dans tous les coins fait perdre beaucoup trop de temps à un informaticien digne de ce nom.
- d'autre part, une partie importante de cette première partie du cours sera dévolue à l'enseignement du langage de script **Python** qui vous permettra de réaliser des scripts systèmes (c'est à dire de nouvelles commandes). Ce langage interprété remplace les shells Unix (tcsh, bash ...), mais pas le merveilleux mais déroutant langage Perl qui n'aura jamais de remplaçant.

Pour la partie scripting, voir le polycopié complémentaire sur l'initiation aux technologies du Web par la création de pages CGI (écrites en Perl ou en Python) :

- *HTML* ;
- *JavaScript et le DHTML (réduit aux divisions)* ;
- le cas des *CGI*.

Cette initiation aux technologies du web sera la base, lors du cursus de la Licence Informatique et du Master IFPRU à d'autres enseignements trsitement dispensés par mes soins :

- la programmation web "dynamique" via l'accès au *DOM* de la page web ;
- le duo *PHP / MySQL* ;
- la galaxie XML : *XML / XSL* et les "tiers" présentation basés sur XML : *SVG*, *Flex* et *Open Laszlo* ;
- la programmation en *ActionScript 3* (y compris pour la 3D) pour un *FlashPlayer* ;
- les *clients riches*.

Voici quelques remarques sur les chapitres de cette première partie :

- Chapitre 1 : **Introduction**
  Présentation générale d'*Unix*.
- Chapitre 2 : **Installation et découverte de *Ubuntu***
  Quelques informations relatives à l'installation et la prise en main de la distribution Ubuntu sont présentées.
- Chapitre 3 : **Environnement**
  Présentation générale de *Linux* et de son environnement.
- Chapitre 4 : **Commandes**
  A exécuter virilement sur un terminal.
  En cours, nous ne verrons que les commandes les plus importantes (celles qui sont encadrées).
- Chapitre 5 : **Outils**
  Ce chapitre présente d'une part les macro-commandes permettant de se connecter sur une machine distante (*ssh* et *ncftp*), et d'autre part les éditeurs de textes (pour la programmation et pour la bureautique) les plus importants intégrés à Linux.
- Chapitre 6 : **Interpréteur de commandes** (*tcsh* et *bash*)
  Les exemples de scripts donnés soit en *tcsh*, soit en *bash* ne le sont qu'à titre informatif, nous utiliserons l'interpréteur *Python* beaucoup plus puissant pour écrire nos propres scripts système.
- Chapitre 7 : **Ecriture de scripts Python et Perl**
  Présentation rapide de la syntaxe du langage de script Python. En parallèle, le langage Perl plus efficace mais d'une syntaxe bien plus déroutante sera présenté (et puis personne ne peut rompre avec Perl sans nostalgie).
- Chapitre 8 : **Appels système**
  Liste des appels systèmes qui permettraient de réécrire un interpréteur de commandes en C.
  Nous ne les verrons pas en cours.
- Chapitre 9 : **X Window**
  Quelques informations sur le serveur de gestion de fenêtres.
  Nous ne les verrons pas en cours.

Les mentions "*Nous ne les verrons pas en cours.*" n'ont de raison d'être que leur vertu vexatoire et frustante.

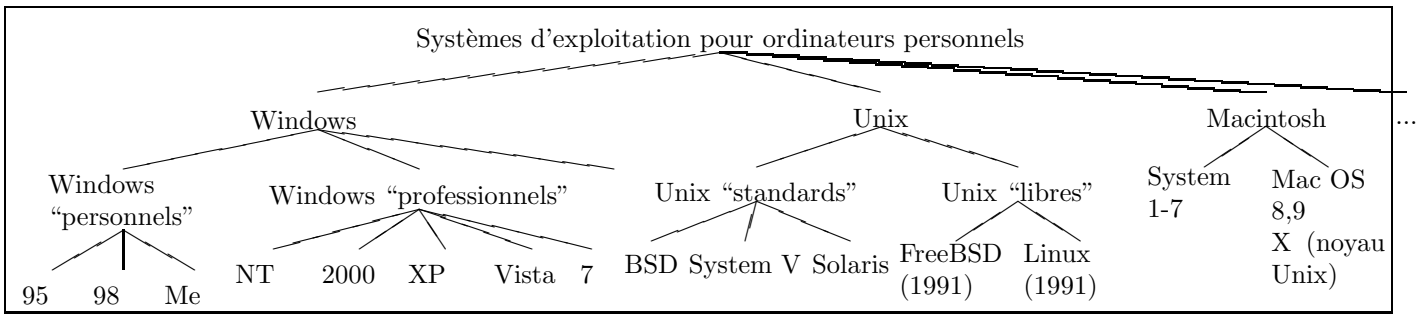
les parties importantes sont encadrées.

Les touches du clavier sur lesquelles appuyer sont entourées par des "¡" et "¿" ; exemple : *¡Tab¿*.

Table des matières

1 Introduction :

1.1 Panorama des systèmes d'exploitation pour ordinateurs personnels :



Part de marchés estimées pour les postes clients :
- XITI avril 2008 : Windows : 94%, MacOS : 4%, Linux : **1%** ;
- IDC novembre 2008 : Windows : 90.5%, MacOS : 8.5%, Linux : **1%** ;

Part de marchés estimées pour les postes serveurs (IDC août 2008) : Windows : 36,5%, Unix **46,1%** (Linux : 13,4% et autres systèmes Unix : 32,7%), Z serveur d'IBM (dont certains sous sont Linux par le biais de VM) : 11,8%

1.2 Bref historique d'Unix et de Linux :

UNIX est un système d'exploitation très fréquemment répandu dans les environnements de recherche, d'enseignement ou de développement. Il n'est lié à aucune architecture ou constructeur particulier. Son interface, longtemps austère, a été améliorée avec le système de gestion de fenêtres XWindow.
Il a été créé en 1969 au Bell Laboratories par Ken Thompson. Il est rapidement réécrit en C et différentes versions, SYSTEM-V(ATT et Bell Laboratories), BSD (Berkeley), XENIX (Microsoft) ... voient le jour. Une procédure de normalisation a homogéinisé toutes ces versions (normalisation des services offerts et de leurs accès, portabilité au niveau du code source). A partir de 1988 est effectuée l'intégration dans l'environnement de UNIX du système de gestion de fenêtre XWindow (projet Athena du MIT) et notamment de la bibliothèque de widgets MOTIF.

Ce cours sera plus spécialement dédié à **Linux** (créé en 1991 par Linus Torvalds), une implémentation libre d'UNIX pour les ordinateurs personnels. Linux est **gratuit**, puissant, stable, interfacé par différents "Windows managers" comme **KDE** ou **GNOME** (qui eux-mêmes reposent sur le système de gestion de fenêtres XWindow).
Linux peut être installé parallèlement à d'autres partitions MS-DOS/Windows ou tout autre système d'exploitation.
Différentes distributions de Linux existent, dont les plus célèbres sont :
- Slackware (plus ancienne distribution encore en activité)
- S.u.S.E.
- La distribution Linux de Debian
- Red Hat Linux, Fedora Core, CentOS et Mandrake devenu Mandriva
- **Ubuntu**

Remarque : RedHat avec le format **RPM** et Debian ou Ubuntu avec le format **DEB**, ont conçu des archives qui permettent d'effectuer une installation, une désinstallation ou une mise à jour de n'importe quel logiciel très facilement. Ils vérifient les dépendances nécessaires et les installent directement au bon endroit. Ubuntu intègre un logiciel de téléchargement de logiciels (**Synaptic**) très convivial.

1.3 Vue générale :

Unix est un :
- Système **multi-utilisateurs**,
- Système **multi-tâches** :
  - avec X11, plusieurs fenêtres représentants autant de terminaux différents,
  - processus en arrière plan,
  - bonne répartition des ressources de l'ordinateur,
- Système **de développement** :
  - interpréteurs de commandes puissants appelés shells,
  - systèmes de fichiers et de processus hiérarchisés,
  - vision unique des différents types d'entrées-sorties,
  - réallocation des entrées-sorties des processus (filtres et redirections),
  - points d'accès aux services offerts par le noyau dans des langages évolués (appels systèmes)

2 Installation et découverte d'Ubuntu :

(prévilégiez les versions stables des mois d'avril et d'octobre (4 et 10 ;-))

Ubuntu (mot bantou dont la signification nous rappelle délicieusement que nous faisons tous partie de la même famille), est une distribution de Linux (ou plutôt des distributions...) qui a les mérites :
– d'être distribuée en Live CD ce qui permet de l'essayer avant de l'installer ;
– d'être basée sur la solide distribution *Debian* ;
– de posséder de multiples pilotes lui permettant de s'adapter à un nombre considérable de machines ;
– d'offrir un système convivial de téléchargement de logiciels (Synaptic) ;
– et de me plaire.

2.1 Installation d'Ubuntu :

2.1.1 Vérification des partitions existantes sous Windows (facultative) :

Sous Windows, vérifiez l'etat des partitions :
(de memoire) Panneau de configuration → Outils d'administration → Gestion de l'ordinateur → Gestion des disques.
Vous pouvez également exécuter la commande **fdisk** dans une invite de commande DOS. Deux cas sont possibles :
– une partition a été créée pour installer Linux ;
– seules les partitions Windows sont présentes.
Dans le second cas, une opération recommandée (optionnelle mais confortable) est de défragmenter vos disques, puis de libérer de la place sous Windows en utilisant soit **Partition Magic**, soit **Gparted-Live**.

2.1.2 Repartitionnement :

Booter sur le disque Ubuntu (si votre ordinateur ne démarre pas directement sur le lecteur de CD/DVD, changer l'ordre de la séquence de boot dans le BIOS (en y accédant lors du démarrage de votre ordinateur par [F12], ou par [F2], [F8] suivant le bref message affiché...)). Comme vous pouvez le constater, Ubuntu vous offre directement un système Linux en "live cd" à partir duquel vous pouvez lancer son installation en DUAL BOOT.
Arrivé à la phase de partitionnement, choisissez :
– "Utiliser le plus grand espace disque disponible" si de la place libre existait sur votre disque ;
– sinon, choisissez le partitionnement manuel.

Dans le second cas, vous allez créer au moins trois partitions supplémentaires : une **partition** est une sectorisation virtuelle d'un disque (pour éviter que des informations corrompues altèrent par exemple l'ensemble du disque). Primitivement un disque ne peut être partitionné qu'en quatre partitions : ce sont les **partitions physiques**. Or Windows occupe déjà une ou deux partitions et il est conseillé de créer trois ou quatre partitions pour Linux. Ubuntu va donc vous permettre de transformer une partition physique en **partition étendue** qui va contenir des sous-partitions : les **partitions logiques**.
– **/** : partition principale (associée au système de fichiers **ext3** ;
– **/home** : partition utilisateur (également associée à ext3) ;
– **swap** : partition d'échange pour gérer la mémoire virtuelle ;
– *vous pourriez également créer une partition supplémentaire d'échange avec Windows en FAT32.*

2.1.3 Installation/Configuration du gestionnaire d'amorçage Grub :

Grub est installé directement par Ubuntu dans le cas d'une installation standard. Ce qui suit ne vous sera utile qu'en cas d'installation atypique...

Grub est un mini-système d'exploitation qui permet de gérer le démarrage d'autres systèmes : c'est un **gestionnaire d'amorçage**. Il est générique (pas forcément dédié à Linux) ce qui lui confère un avantage par rapport à Lilo. Il dispose d'un petit interpréteur de commandes permettant l'exécution de commandes internes (cet interpréteur se nomme **grub** ;-)).
Grub se décompose en plusieurs "morceaux" (stage1, stage1.5, stage2) la place mémoire disponible par exemple dans le MBR ne lui permettant pas d'y être logé dans son intégralité ;-).
– phase 1 : chargement, soit dans le MBR (Master Boot Record) ou dans le secteur de boot d'une partition, de **stage1** ;
– phase 2 : le **stage1** accèdera au **stage1_5** correspondant au système d'exploitation de la partition sur laquelle va se trouver stage2 (le stage1_5 est généralement placé en mémoire juste après le stage1) ;
– phase 3 : **stage1_5** qui lui-même accèdera à **stage2** (le vrai grub).

MBR (Master Boot Record) : Premier secteur adressable d'un périphérique amorçable (par exemple un disque dur). Ce secteur contient une routine d'amorçage du système d'exploitation ou du **chargeur d'amorçage**. En effet le **BIOS** lit les premiers secteurs des périphériques définis par l'utilisateur jusqu'à en rencontrer un qui contient le nombre magique lui indiquant qu'il doit charger le code d'amorçage. C'est également le MBR qui contient la table des partitions primaires.

Adressage des disques :
- 1ère partition du 1er disque (hd0,0)
- 2ème partition du 1er disque (hd0,1)
- 1ère partition du 2nd disque (hd1,0)

Commandes :
- **root** (hddisque,partition) : activation d'une partition en tant que partition de démarrage ;
- **kernel** /boot/vmlinuz... root=/dev/hd... options : chargement de noyau ;
- **initrd** /boot/initrd.img... : le noyau monte de façon temporaire un système racine en utilisant une image compressée
- **chainloader +1** : chargement d'un fichier ou des blocs qui jouent le rôle de chargeur secondaire (utilisé pour activer Windows) ;
- **rootnoverify** (hddisque,partition) : activation de la partition de démarrage mais sans la monter (l'utiliser pour les OS qu'on ajoute dans le fichier de configuration).

Installation de grub à partir d'un CD (tapez grub pour rentrer sous l'interpréteur et quit pour en sortir : Sur le MBR (Master Boot Record) (taille de 512 octets) :

```
root (hd0,0)
setup (hd0)
```

Sur le secteur de boot d'une autre partition : (par exemple pour conserver le MBR Windows) et que celui-ci puisse assurer le chaînage jusqu'à lui :

```
root (hdx,0)
makeactive  --> cette partition est déclarée active
setup (hdx,0)
```

Pour installer Grub sur un second disque dur à partir du grub initial : appuyez sur la touche c pour passer sous l'interpréteur, ESC pour en sortir.

Contenu du répertoire /boot/grub : Entre autres :
- **stage1** : premier chaînon menant au chargeur ;
- **systeme_stage1_5** : passage optionnel par un second chaînon gérant le système de fichiers sur lequel se trouve le dernier chaînon ;
- **stage_2** : grub en tant que tel.
- **menu.lst** : déclaration des différents systèmes ;
- ...

Paramétrages de menu.lst :

Windows et Linux sur le même disque
- Windows se trouve sur la première partition du premier disque **(hd0,0)** ;
- Linux se trouve sur la seconde partition du premier disque **(hd0,1)**.

```
title         Windows 95/98/NT/2000/XP
root          (hd0,0)
makeactive
chainloader   +1

title         Linux
root          (hd0,1)
kernel        /boot/vmlinuz_... root=/dev/hda2 ro
initrd        /boot/initrd.img...
```

Linux sur un second disque (mais Grub sur le MBR du premier) :

```
title         Windows 95/98/NT/2000/XP
root          (hd0,0)
makeactive
chainloader   +1

title         Linux
root          (hd1,0)
kernel        /boot/vmlinuz_... root=/dev/hdb1 ro
initrd        /boot/initrd.img...
```

Windows sur un second disque :

```
title         Windows 95/98/NT/2000/XP
root          (hd1,0)
makeactive
map (hd0) (hd1)
map (hd1) (hd0)
chainloader   +1
```

Il faut virtuellement échanger les deux disques...

2.1.4 Connexion :

Vous devez vous identifier avec le login et le mot de passe que vous avez choisis lors de l'installation du système. Vous êtes par défaut un **utilisateur particulier** qui peut avoir momentanément les droits de l'administrateur (appelé sous les autres distributions **root**). Dans un terminal, vous pourrez exécuter une tâche administrative nécessitant les droits de l'administrateur de deux manières différentes :
– pour lancer une commande non interfacée par : sudo [commande] ;
– pour lancer une application interfacée par : gksudo [application] .

2.1.5 Opération système avec les droits de l'administrateur

Par exemple, à partir d'un terminal (cherchez-le ;-)), visualisez l'état des partitions par : **sudo fdisk -l**.

2.1.6 Découverte du bureau (Gnome par defaut) :

Bureaux virtuels :
– CTRL + ALT + flèches pour vous déplacer d'un bureau à l'autre ;
– CTRL + ALT + SHIFT + flèches pour déplacer la fenêtre active dans un autre bureau.

Tableaux de bord :
– clic-droit sur le tableau pour changer ses propriétés ;
– pour déplacer/enlever un applet (élément d'un tableau de bord) : clic-droit sur celui-ci ;
– pour insérer un applet : clic-droit sur une zone vide du tableau de bord.

Lanceur (raccourci vers une application) : Clic-droit sur le bureau → Créer un lanceur...

2.1.7 Utilisation d'un navigateur Internet :

Si vous n'êtes pas déjà familiarisé avec un navigateur web, lancez le navigateur **firefox** soit grâce à son icône, soit grâce au chargeur d'application : appuyez simultanément sur **[ALT]** et **[F2]**, puis tapez **firefox** dans la zone de saisie et appuyez sur retour-chariot.
Sous Firefox, tapez dans la zone de saisie les URL suivantes :
– `http://www.ubuntu-fr.org` pour connaître les informations données sur Ubuntu ;
– `http://www.linux-france.org` pour connaître les informations données sur Linux ;
– http ://www.lirmm.fr/~pompidor pour accéder à des pages très très haut de gamme.
(Mémorisez les URLs qui vous intéressent comme **"signets"**/**"bookmarks"**)

2.1.8 Paramétrage du réseau :

Pour configurer la connexion au réseau, vous devez ouvrir la fenêtre de configuration par **Système → Administration → Reseau**. Cela-dit, votre poste peut être géré par un serveur **DHCP**, Ubuntu a se debrouillant alors tout seul... Si vous deviez configurer une adresse IP fixe, outre celle-ci paramétrez également le sous-réseau et la passerelle.

2.1.9 Parcours de l'arborescence des répertoires du système de fichiers :

Selectionnez : **Raccourcis → Poste de travail**, pour ouvrir **l'explorateur de fichiers**.
Remontez jusqu'au dossier (repertoire) racine du système de fichiers, et explorez la hierarchie de repertoires.
Dans la barre d'icônes du haut du navigateur (en dessous des menus) :
- [**F9**] permet de desafficher/visualiser les repertoires dans le panneau de gauche
- la **flèche vers le haut** permet de remonter au répertoire père
- la **flèche vers la gauche** permet de revenir au positionnement précédent
- (n'arrivant pas à trouver mes mots pour expliquer ce que fait la **flèche vers la droite**, essayez la après être revenu en arrière)
- en cliquant avec le bouton droit de la souris sur l'icône d'un fichier, vous pouvez opérer un certain nombre de manipulations (que je vous laisse découvrir)

Vous pouvez également utiliser le panneau de gauche pour vous positionner directement sur un répertoire.
Par celui-ci, vous pouvez également fermer ou ouvrir un répertoire.

2.2 Installation de logiciels :

2.2.1 Installation de paquets (paquetages) par Synaptic :

- **/etc/apt/sources.list** : fichier de configuration décrivant l'ensemble des **serveurs** connus contenant des paquetages
- **apt-get install** permet d'ajouter un média mais cette commande est difficile à manipuler
  → préférez l'utilisation du gestionnaire de paquets (attention si vous avez changé la liste des sources, n'oubliez-pas de cliquer sur le bouton "recharger").

2.2.2 Installation de sources archivées (codes à compiler) :

- Téléchargement d'une archive **.tar.gz** ou **.tgz**, l'archive est recopiée dans le répertoire **Desktop**
- **tar -xzvf nom_de_l_archive** : désarchivage + décompression → voir ci-après la documentation de cette commande
- lire les fichiers README et INSTALL s'ils existent
- **./configure** : création du fichier de gestion de la compilation **Makefile**
- **./make** : compilation (au sens large du terme (compilation + édition de liens))
- **./make check** : test du résultat de la compilation
- **./make install** : déplacement des binaires dans les répertoires systèmes

2.2.3 Installation et configuration d'Apache2 :

Apache2 est le **serveur HTTP** qui permet de servir des ressources web (pages HTML, script PHP...).

- Installer Apache → à télécharger via le gestionnaire de paquets
- Si vous voulez que les utilisataurs puissent entreposer leurs ressources web dans le répertoire **public_html** dans leur répertoire d'accueil, suivez les explications données à l'URL suivante :
  `http://lists.debian.org/debian-user-french/2006/04/msg00760.html`
- Lancer ou relancer Apache :
  - Pour arrêter Apache : **/etc/init.d/apache2 -k stop**
  - Pour lancer Apache : **/etc/init.d/apache2 -k start**
  - Pour relancer Apache : **/etc/init.d/apache2 -k restart**
- Déposez les ressources web communes dans **/var/www**
- Tester la page web que vous avez créée (voir paragraphe suivant) avec : **http ://localhost/essai.html**
- Les principaux fichiers de configuration d'Apache sont les suivants :
  - **/etc/apache2/apache2.conf** : configuration générale du serveur
  - **/etc/apache2/httpd.conf** : ajouts à la configuration générale du serveur (vide au départ)
  - **/etc/apache2/sites-enabled/000-default** : configuration de l'**hôte virtuel** standard
    remarquez la ligne : `ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/` qui localise par défault les scripts CGI.

Page HTML statique de test (essai.html)

```
<html>
   <body>
      <font color="red"> Bonjour Maître, le serveur Apache est content de te répondre </font>
   </body>
</html>
```

2.3 Gestion des utilisateurs et des groupes :

– Créez un nouvel utilisateur avec le gestionnaire des utilisateurs et groupes
– Accordez-lui le privilède d'administrer le système
– Vérifiez les informations le concernant dans le fichier `/etc/passwd`
– Vérifiez les informations le concernant dans le fichier `/etc/group`
Attention les fichiers `/etc/passwd` et `/etc/group` n'ont de sens que si vous gérez une machine isolée.

3 Environnement de base :

Vous disposez sous Linux de deux environnements graphiques possibles (nommés **bureaux**) : **GNOME** et **KDE**. Leur fonctionnement intuitif ne nécessite pas d'explication détaillée dans ce polycopié, hormis deux facilités :
- pour lancer une application qui ne serait pas représentée par une icône, tapez simultanément **Alt** et **F2**, puis dans la zone de texte l'application à lancer (par exemple le navigateur **Firefox** (nouvelle "version" de **Mozilla**) ou l'éditeur **xemacs**).
- pour explorer les répertoires et ouvrir les fichiers, vous pouvez utiliser l'explorateur de répertoires, c'est à dire la navigateur *Konqueror*. Pour cela cliquez sur l'icône représentant une petite maison.

Par ailleurs vous pouvez également ouvrir des **terminaux** où vous saisirez des commandes textuelles qui seront interprétées par un **interpréteur de commandes**. Cette façon de procéder est pour un utilisateur aguerri, la plus rapide, et la plus efficiente, une minorité de commandes étant interfacée sur le bureau.

3.1 Ouverture d'un compte, connexion, environnement, vision des utilisateurs, communication :

3.1.1 Ouverture d'un compte :

Demander l'ouverture d'un compte à l'administrateur qui attribue :
- un numéro individuel (uid : user identification),
- une chaîne de caractères de **login** (**l'/les initiale(s) de votre/vos prénom(s) suivie(s) des premières lettres de votre nom, le tout ne devant excéder 8 caractères**),
- et un mot de passe,

ce qui se traduit par l'ajout d'une ligne dans le fichier `/etc/passwd` ou dans le fichier `/var/yp/passwd.byname` (machines en réseau), cette ligne étant composée comme suit :
`nom : mot de passe crypté : uid : gid : infos libres : répertoire de travail : shell`
(uid = numéro d'utilisateur, gid = numéro de groupe)
Le mot de passe crypté peut être occulté en mode "shadow".

Au département informatique de l'université Montpellier 2, l'ouverture d'un compte représente en fait beaucoup plus d'étapes, et sera initiée par l'utilisation d'un petit programme d'inscription "inscription" et la signature d'une feuille faisant état de vos devoirs concernant l'usage de ce compte.

3.1.2 Connexion :

La connexion s'effectue soit sur une machine isolée ou sur un réseau de machines (généralement gérées par **NIS** : Network Information Service). Votre connexion est validée si votre login est connu et votre mot de passe encrypté similaire à celui mémorisé dans :
- le fichier */etc/passwd* dans le cas d'une connexion sur une machine isolée
- le fichier *passwd* géré par le serveur NIS dans le cas d'une connexion sur un réseau de machines

Après votre connexion, vous pouvez être face :
- à une interface graphique (appelée bureau)
- à une interface de type console

Dans le second cas, pour lancer l'interfaçage graphique utilisez la commande **startx**.
La commande **startx** lance le **serveur X** qui s'occupe de l'affichage de toutes les fenêtres graphiques...

Remarque : Vous pouvez vous connecter plusieurs fois en parallèle sur la même machine en ouvrant jusqu'à six consoles textuelles et une seule graphique :
- consoles textuelles : de **CTRL ALT F1** à **CTRL ALT F6**
- console en mode graphique : **CTRL ALT F7**

3.1.3 Manipulation d'un terminal :

Après familiarisation avec l'interfaçage graphique ayant les mêmes fonctionnalités que celles des autrez systèmes pour micro-ordinateurs (Windows ou Mac OS), lancez un terminal (nous allons aimer ensemble utiliser des terminaux) :
- en cliquant sur une icône représentant un écran d'ordinateur si elle existe
- en sélectionnant un item "terminal" dans un sous-menu "Système" dans le menu principal de votre bureau

Lors de la création d'un terminal, les opérations suivantes sont effectuées :
- Création d'un processus exécutant un **interpréteur de commandes** ou **shell** :
  L'interpréteur de commandes est le programme qui va "comprendre" vos commandes.
  Il existe plusieurs implémentations possibles d'interpréteurs de commandes, les deux les plus fréquentes étant **bash** et **csh**. Pour connaître l'interpréteur de commandes que vous utilisez par défaut, vous effectuerez la commande suivante dans votre terminal : **echo $SHELL**.
  Sa première implémentation ayant été appelée **shell**, l'interpréteur de commandes est également souvent appelé comme cela (par extension le terminal est souvent aussi appelé ainsi ce qui rend les choses encore plus confuses).
- Configuration de ce processus suivant le contenu d'un fichier de configuration nommé
  - **.bashrc** si votre interpréteur de commandes est **bash**
  - **.cshrc** si votre interpréteur de commandes est **csh** ou **tcsh**
- Affichage de la bannière (message du terminal modifiable par l'utilisateur) :
  - voir à **PS1** si votre interpréteur de commandes est **bash**
  - ou à **prompt** si votre interpréteur de commandes est **csh** ou **tcsh**

Après lecture d'une ligne, l'interpréteur de commandes l'analyse pour exécuter les commandes qui y sont recélées.
Il existe deux types de commandes :
- les commandes internes : exécutions directes (comme la commande *cd* vue plus loin),
- et les commandes externes : créations de nouveaux processus.

3.1.4 Syntaxe et documentation sur les commandes :

Une commande Unix saisie dans un terminal a différentes syntaxes, notamment :
- **commande -c_1...c_n paramètres** (où c_i est un caractère définissant une option)
- **commande –mot paramètres** (où mot est un mot explicite)

Pour lire la documentation relative à une commande :

apropos :	liste toutes les pages du manuel de toutes les commandes comportant le mot clef donné en paramètre exemple : *apropos directory* (directory signifie répertoire en anglais)
man :	affiche les pages du manuel de la commande donnée en paramètre exemple : *man mkdir*

Par ailleurs, si vous ne tapez que les premières lettres d'une commande, l'interpréteur de commande va :
- compléter son nom (s'il n'y a pas d'ambiguïté) si vous appuyez une fois sur <**Tab**>
- lister toutes les commandes commençant par ces lettres si vous appuyez deux fois sur <**Tab**>

3.1.5 Environnement :

date :	affichage de la date et de l'heure
hostname :	nom de la machine

3.1.6 Vision des utilisateurs :

whoami :	login de l'utilisateur
id :	uid et gid de l'utilisateur
who :	affichage des autres utilisateurs de la machine locale et des numéros de lignes (terminaux) login terminal date_de_connexion
finger :	même fonction que *who*, mais permet en plus : de visionner les connectés d'une machine distante :*finger @machine* de demander des informations sur un utilisateur : *finger login@nommachine*
rwho : ou **rusers**	permet de connaître les utilisateurs des machines du réseau local (interrogation de chaque machine)
last :	permet d'avoir l'historique des dernières connexions sur la machine

3.1.7 Communication par courrier électronique :

Vous allez disposer d'une **adresse électronique** sur le modèle suivant :
votre_login@adresse_internet_du_serveur_de_courriels_du_département_informatique
(A l'heure où j'écris ce document, l'adresse internet du serveur de courriels du département informatique est : info-ufr.univ-montp2.fr (cela peut changer))

Suivant ce schéma mon adresse électronique serait donc : ppompido@info-ufr.univ-montp2.fr
(ce qui n'est d'ailleurs pas le cas vu son esthétisme discutable)

Pour envoyer ou recevoir des courriers électroniques (**email**, **mail** ou **courriel**), de nombreux **gestionnaires de mails** existent. Sous le bureau KDE, je vous propose d'utiliser **KMAIL**.
Pour le paramétrer, réalisez les opérations suivantes (seules les étapes clés sont décrites) :
- sélectionner l'item **Configuration** dans le menu **Fichier**
- dans l'adresse électronique, supprimez le nom de la machine locale
- **ajouter** une boîte aux lettres **POP3**
  - Utilisateur : **votre login**
  - Mot de passe : rien (cad vous ne tapez rien)
  - Serveur : **mail**

Pour lire du courriel d'une boîte aux lettres extérieure au département informatique, vous pouvez utilisez un navigateur web.

Pour information, la commande Unix de base **mail** s'utilisait ainsi :

mail : envoi de courrier :
`mail login_utilisateur_système_local`
`mail login@nom_machine.nom_site.suffixe_pays`
retour du courrier électronique si le correspondant n'est pas trouvé
nslookup pour connaître le numéro internet d'une machine

3.1.8 Utilisation d'un navigateur web :

Pour surfez sur le web, vous devez utiliser un navigateur (arpenteur, butineur, browser ...).
Sur Linux, vous avez le choix entre :
- **Konqueror** : léger, utilisé comme explorateur du système de fichiers
- **Netscape** : ancien, obsolète, pour les nostalgiques
- **Firefox** (Mozilla) : à la pointe, version OpenSource de Netscape, futur noyau d'Internet Explorer ;-)
- **Opera** : rapide

Pour surfer, il suffit que vous inscriviez une **URL**, c'est à dire une adresse internet, dans la zone de saisie prévue à cet effet, (exemple d'URL impressionnante : http ://www.lirmm.fr/~pompidor), après avoir configuré le navigateur : allez dans : Edition → Préférences → Avancées → Proxy, saisissez dans la zone de saisie : **http ://www/proxy.pac** et validez.

3.1.9 Autres modes de communication (désuets) :

write : communication directe pour systèmes non distants :
(vous devez auparavant vous connecter sur la machine de votre interlocuteur ...)
`write login_utilisateur_machine`
`write login_utilisateur_machine terminal`

talk : communication directe pour systèmes distants :
`talk login_utilisateur_machine`
`talk login_utilisateur_machine terminal`
`talk adresse`

mesg : autorisation ou interdiction des communications extérieures (sauf mail)
demande de l'état actuel : `mesg`
positionnement : `mesg y` ou `mesg n`
`ctrl d` (fin de fichier = EOF), `exit` ou `logout` : déconnexion

3.2 Le système de gestion des processus :

Un prcessus est :
- un programme en train d'être exécuté,
- les données manipulées par ce programme,
- son contexte d'exécution (registre, pile, liens utilisateurs et système d'E/S ...).

Les processus sont ordonnancés par un ordonnanceur.
Le processus **init 1** est parent des processus shells créés par l'utilisateur.
Caractéristiques d'un processus :
- identification (**PID**),
- identification du processus parent (**PPID**),
- propriétaires,
- groupes propriétaires,
- éventuellement terminal d'attachement (**TTY**),
- priorité,
- différents temps ...

Accès à une table des fichiers manipulés héritée du processus père.

3.2.1 Quelques commandes de base sur les processus :

ps : liste des processus appartenant à un ensemble particulier
sans options : processus attachés au même terminal (hors processus XWindow)
- numéro d'identification (**PID**),
- numéro terminal (**TTY**)
- infos sur l'exécution (4 lettres : stoppé, arrêté, inactif, priorité réduites ...) (**STAT**)
- temps cumulé d'exécution (**TIME**)
- nom du fichier correspondant au prog. exécuté par le processus (**COMMAND**)

-u (f sur Linux) : format long (avec informations sur la taille mémoire, numéro du père, etc ...)
-a (e sur Linux) : liste de tous les processus s'exécutant sur la machine
-x (e sur Linux) : liste des processus XWindow

kill : pour envoyer un signal à un processus
-9 numéro_processus pour le tuer de manière certaine

3.3 Gestion des fichiers :

3.3.1 Le système de gestion des fichiers :

Un fichier Unix est un fichier sur disque ou une ressource du système (device).
Il peut avoir ou non un contenu sur disque :
- fichier régulier,
- répertoire,
- lien symbolique,
- tube de communication,
- ressource (terminaux, claviers, imprimantes, disques physiques et logiques).

Chaque fichier correspond à une entrée (i-noeud, i-node ou index node) dans une table contenant l'ensemble de ses attributs :
- type,
- propriétaire,
- droits d'accès (lecture, écriture, exécution),
- taille (pour les fichiers sur disque),
- nb de liens physiques,
- dates de lecture, de modification du fichier et du noeud,
- adresse des blocs utilisés sur le disque (pour les fichiers sur disque),
- identification de la ressource associée (pour les fichiers spéciaux).

Un fichier a donc pour identification le couple (table, entrée).
Les répertoires permettent d'organiser une arborescence de fichiers qui peuvent avoir des références absolues (/usr/toto) ou relatives (../toto), (. répertoire courant, .. répertoire père).
Un répertoire UNIX n'est jamais vide (.. et .).

Les répertoires usuels :
Sous la dépendance de l'administrateur, mais quelques conventions sont respectées :

/bin et /usr/bin :	commandes Unix non internes au shell
/usr/local/bin :	commandes particulières à un site donné
/dev :	noms des fichiers spéciaux associés aux ressources
/tmp :	fichiers temporaires
/usr :	fourre-tout des outils de développement (fichiers inclus du C, de X etc ...)
/var :	NIS
/users, /home..., ... :	répertoires de travail des utilisateurs

3.3.2 Quelques commandes de base sur les fichiers :

ls : liste du contenu d'un répertoire
l : type, droits, nb_liens, nom_proprio, groupe_proprio, taille, date
d : n'explore pas les répertoires
a : liste les fichiers "systèmes" commençant par un "."
R : liste récursive des répertoires

cat : liste le contenu d'un fichier
more : liste le contenu d'un fichier page par page
appuyez sur la barre d'espacement pour passer d'une page à l'autre
sur "q" pour quitter

cp : copie un fichier (*cp -R répertoire_source répertoire_destination* copie récursivement un répertoire)
ln : crée un lien (référence sur un fichier)
ln -s nom_fichier nouveau_nom (création d'un nouvel inode → lien symbolique)
mv : change le nom ou déplace un fichier
mv nom_fichier nouveau_nom_du_ficher → changement de nom
mv nom_fichier répertoire → déplacement
rm : supprime un fichier
rm -R répertoire détruit récursivement un répertoire
pwd : donne la réference absolue du répertoire courant
cd : changement de répertoire
mkdir : crée un répertoire
rmdir : détruit un répertoire

chmod : change les droits d'un fichier (mode symbolique ou numérique)
r pour read, **w** pour write, **x** pour execute, **u** pour user, **g** pour group, **o** pour others, **a** pour all :
chmod go+rx fichier
4 pour read, **2** pour write, **1** pour execute : *chmod 755 fichier*

umask : note les droits par défaut d'un fichier (umask 022 pour rwxr-xr-x)
chown : change le propriétaire d'un fichier (réservé au superutilisateur)
chgrp : change le groupe d'appartenance d'un fichier

3.3.3 Les mécanismes de redirection, de communication par tubes et de détachement :

Les applications développées doivent pouvoir être composées entre elles sans être retouchées.
Les applications sont des boîtes noires :
– lisent leurs données sur un fichier logique appelé entrée standard
– écrivent leurs données sur un fichier logique appelé sortie standard
qui sont par défaut associés au clavier et au terminal : ce sont les fichiers de descripteurs 0 et 1 de la table des fichiers liés au processus.
L'association des fichiers logiques aux fichiers physiques peut être modifiée par un processus de redirection.
Une sortie erreur standard est aussi définie (descripteur 2) (par défaut associée au terminal).
Redirection de :

l'entrée standard : commande < nom_fichier_lisible
la sortie standard : commande > nom_fichier_autorisé_en_écriture
la sortie standard (en ajout) : commande >> nom_fichier_autorisé_en_écriture
la sortie erreur standard : commande 2> nom_fichier_erreur

Enchaînement de processus en séquence :
Le déroulement du premier processus n'influe pas sur le déroulement du second (pas d'échanges d'informations) :
`commande_1 ; commande_2`
Avec parenthésage pour une redirection :
`(commande_1 ; commande_2) > nom_fichier`

Processus concurrents et communiquant entre eux :

Emploi de tubes (encore nommés pipes) : commande_1 |commande_2
Le système assure les tâches de synchronisation des écritures et des lectures.

Possibilité de lancer des processus en mode détaché :
l'utilisateur peut continuer à soumettre des commandes à l'interprète shell qui a lancé la commande : processus en arrière-plan ou en background :
– `commande &`
– `(commande_1; commande_2) &`
La terminaison d'une session tue les processus sauf ceux lancés par l'intermédiaire de la commande **nohup** :
`nohup commande [<données] [>résultats] &`
Les erreurs sont redirigées sur le fichier **nohup.out**.

Lancement à une date donnée :
at heure :minute am/pm 3_premières_lettres_mois jour
`at 9:45 16/09/02 < echo "rentrée des IAO"`
les utilisateurs autorisés à utiliser cette command doivent être référencés dans le fichier verb—/etc/at.allow/—
now un incrément en minutes/hours/days/weeks/months/years est autorisé
batch dès que le système le permet

Planification de tâches :
crontab :
crontab -e appelle l'éditeur de tâches à planifier
(export VISUAL=xemacs pour remplacer vi par xemacs sous bash)
format d'une ligne : `minute heure jour_du_mois mois jour_semaine commande`
exemples : `0 0 * * mon find / -atime 7 exec rm {} $backslash$;`
suppression de tous les fichiers non lus depuis une semaine
chaque lundi à 0 heures 0 minutes
crontab -l visualise la planification courante

Remarque : les commandes `nohup`, `at` et `batch` sont autorisées si le démon **cron** existe.

4 Principales commandes du système :

Pour ce qui suit, un fichier départements est pris comme exemple :
01 :50 :5756 :Ain :Bourg-en-Bresse :Rhône/Alpes
etc ...

4.1 Relatives à l'administration système :

passwd : changement du mot de passe sur la machine locale

yppasswd : changement du mot de passe dans les pages jaunes (commande NIS)
ypwhich : affichage du nom de la machine serveur de pages jaunes (commande NIS)
ypcat : affichage d'un fichier page jaune (commande NIS)
Un fichier "page jaune" est un fichier administratif géré par le serveur.
Quelques exemples de fichiers pages jaunes :
passwd : fichier des utilisateurs
group : fichier des groupes
hosts : fichiers des machines du site
ypmatch : recherche les lignes d'un fichier "page jaune" contenant un champ particulier
ypmatch champ fichier_page_jaune

4.2 Relatives aux shells :

Commandes externes : l'exécution de chacune de ces commandes correspond à un processus dédié dont l'exécutable se trouve dans les répertoires /bin ou /usr/bin.

chsh : changement de shell pour les connexions ultérieures
nom_login référence_absolue_shell : `chsh pompidor /bin/csh`
généralement : sh (Bourne-Shell), csh (C-Shell) et tcsh (Turbo-C-Shell)

chfn : changement du nom complet de l'utilisateur

sh : appel de l'interpréteur de commandes désuet **sh (Bourne-Shell)**

bash : appel de l'interpréteur de commandes **bash (Bourne-Shell Again)**
(par défaut celui de nombreuses distributions de Linux)
l'exécution du processus commence par l'exécution du fichier de nom **.bashrc**
→ voir la section *Scripts exécutés par l'interpréteur de commandes*

csh : appel de l'interpréteur de commandes **csh** (shell ayant une syntaxe liée à celle du langage C)
l'exécution du processus commence par l'exécution du fichier de nom **.cshrc**
(**-f** non-exécution de ce fichier)
→ voir la section *Scripts exécutés par l'interpréteur de commandes*

export, set ou setenv : accès aux variables système simples ou d'environnement
`export var_env=valeur` pour bash
`set var_env=valeur` pour csh/tcsh
`setenv var_env valeur` pour csh/tcsh
Les variables d'environnement définissent l'environnement de l'utilisateur.
Par ex. la variable **PATH** liste les répertoires susceptibles de contenir les exécutables appelés par l'utilisateur (répertoires séparés par :).

printenv ou setenv sans arguments : visualisation des valeurs des variables d'environnement.

4.3 Relatives aux terminaux :

logname : affichage du nom de login

clear : effaçage de l'écran

su : modification de son identification (création d'un sous-shell)

tty : écrit sur la sortie standard la référence absolue du terminal associée à l'entrée standard (not a tty si ce n'est pas un terminal)

stty : visualisation et modification des paramètres de la voie de communication entre le terminal et le système
stty : valeurs d'un ensemble réduit de paramètres
stty -a : tous les paramètres
stty sane : la commande magique pour tout remettre en état ...
exemples de paramètres :
erase : effacement du dernier caractère (en mode non canonique)
werase : effacement du dernier mot (en mode canonique)
kill : effacement de la dernière ligne (en mode canonique)
echo / **-echo** : écho / non écho à l'écran des caractères saisis au clavier
lcase : passage en mode minscule par défaut
icanon / **-icanon** : passage en mode canonique / non canonique
(en mode canonique attente d'un retour chariot de validation)
Exemples de modifications : `stty erase '^H'`
stty echo
STTY -LCASE

tabs : définition des tabulations
tabs et liste de nombres en ordre croissant séparés par des virgules :
par défaut tous les huit caractères
(Cette commande n'est pas implémentée dans toutes les versions d'Unix)

indent : indentation d'un programme C/C++

4.4 Relatives aux places mémoires occupées :

df : état des disques logiques (de la partition) notamment nb de blocs libres (512 k ou 1024 k)
du : espace alloué aux différents fichiers en nombre de blocs de 512 ou 1024 octets
pstat -s : statistiques sur la mémoire centrale (pas sous Linux)
top : liste triée des processus exploitant la mémoire centrale
quota : affichage de vos quotas et de la place mémoire que vous occupez

4.5 Relatives aux fichiers :

4.5.1 Sur leur recherche :

find : recherche récursive de fichiers dans une arborescence, par rapport à un nom :
find répertoire -name nom_du_fichier_recherché -print
exemple : *find . -name coucou -print*
si aucun répertoire n'est précisé, la recherche s'effectue à partir de la racine
et d'autres critères, comme une date de dernière modification :
trouver tous les fichiers réguliers modifiés depuis 24 heures :
find répertoire -mtime 1 -type f -print
trouver tous les fichiers et répertoires ne vous appartenant pas :
find $HOME! -user $LOGNAME -print
find permet d'exécuter une commande sur tous les fichiers sélectionnés :
find répertoire critères_de_sélections exec ... {} \ ;

whereis : recherche du fichier binaire, du fichier source et de la page du manuel d'une commande
whereis commande

locate : recherche de tous les fichiers comportant dans leurs noms, une sous-chaîne de caractères
locate sous-chaîne
locate -d répertoire sous-chaîne
Cette recherche exploite un index qui ne peut être créé que par l'administrateur par *locate -u*

file : classification d'un fichier

grep : sélection de lignes satisfaisant un motif particulier fichier ou entrée std
-c : nombre de lignes satisfaisant l'expression
-i : pas de distinction minuscules/majuscules
-v : lignes ne satisfaisant pas le motif
-r : recherche récursive à partir d'un répertoire de départ
grep options expression_régulière fichier

egrep : grep en mieux (utilisation de la norme "Perl" pour les expressions régulières)

Compléments sur l'expression régulière spécifiant le motif de la commande grep :

expression régulière :
[: début de la définition d'un ensemble de caractères
] : terminaison de la définition d'un ensemble de caractères
[abc] : soit a, soit b, soit c
. : un caractère quelconque
***** : indicateur d'itérations (de 0 à n fois le caractère précédent)
- : définition d'intervalles
[0-9] : un chiffre quelconque
^ : début de ligne en début d'expression, complément assembliste après un [
[^0-9] : n'importe quel caractère sauf un chiffre
$: fin de ligne en fin d'expression
**** : déspécialisation d'un caractère
Recherche des lignes correspondant à des départements de superficie comprise
entre 1000 et 4999 :
`grep "^[^:]*:[^:]*:[1-4][0-9][0-9][0-9]:" départements`
Nombre de départements n'appartenant pas à la région Provence/Côte d'Azur
`grep -c -v  'C\^ote d'Azur$' départements` (ici ' pour éviter que $ soit mal interprété)

4.5.2 Sur des visualisations avec mise au format :

od : visualisation d'un fichier sous différents formats
-a : les mots sont visualisés en ASCII
-o : les mots sont visualisés en octals
-x : les mots sont visualisés en hexadécimal

pr : découpage en pages et en colonnes avec en-têtes
-n : nombre de colonnes
-m : un fichier par colonne
-w : nombre de caractères par ligne
-h : entête à imprimer àprès la date en tête de chaque page

psf : formatage d'impression
`psf options fichiers_à_imprimer`
-w : impression en largeur
-p : fonte
(utilisation courante avec la commande d'impression lpr décrite plus loin)

lwf : autre formateur d'impression

tr : substitution ou suppression de caractères
`tr chaîne1 chaîne2`
[...-...] : intervalle, [[:classe :]] (digit, alpha, alphanum, upper, lower)
-d pour la suppression des caractères
tr ab AB fichier

sed : éditeur standard ne travaillant pas en mode interactif fichier ou entrée std
-e `s / expression_régulière / chaîne_de_remplacement / g` (g=toutes les occurr.)
`sed -e s/a/A/g fichier`

sort : tri les lignes du fichier
-f : minuscules et majuscules identiques
-r : inversement de l'ordre
-u : un seul exemplaire des lignes est conservé
-o : pour préciser le fichier de sortie

uniq : élimination des lignes redondantes successives
-c : chaque ligne est précédée de son nombre d'occurrences

4.5.3 Sur des comparaisons :

cmp : comparaison des contenus de deux fichiers
affichage du numéro de la ligne et du caractère de la première différence
`cmp fichier1 fichier2` → differ : char ..., line ...

diff : affichage de toutes les lignes différentes
diff fichier1 fichier2

comm : affichage sur trois colonnes des différences
lignes appartenant au premier, au second, aux deux
`comm fichier1 fichier2`

4.5.4 Sur des transformations :

tee : redirections multiples
`... | tee fichier1 ...`
-a pour ne pas écraser les fichiers
copier les lignes entre 10 et 20 du fichier fich dans 2 fichiers fich1 et fich2
tout en les visualisant à l'écran :
`head -20 fich | tail -10 | tee fich1 fich2`

head : écriture sur la sortie standard des n premières lignes des fichiers (par défaut 10)
`head -n fichier1 ...`

tail : écriture sur la sortie standard des n dernières lignes des fichiers (par défaut 10)
`tail -n fichier1 ...`
lignes 10 à 20 : `head -20 fichier1 | tail -10`

split : découpage en séquence d'un fichier
`split -nombre_de_lignes fichier`
les fichiers produits ont pour suffixes de .aa à .zz

csplit : découpage en séquence d'un fichier avec identification d'une expression
-f préfixe des fichiers résultats
`/ expression régulière /`
`csplit fichier -f fich /^a/` (toutes les lignes commençant par a)

cat : `cat fichier1 ... fichiern > nouveau_fichier`

cut : sous-ensemble de colonnes d'un fichier
-c : liste de portions (c1-c2 : intervalle, c1- : jusqu'à la fin)
-d : séparateur
-f : dans la cas d'un séparateur, numéros des champs à extraire
`cut -c1-10 départements` (caractères 1 à 10)
`cut -f1,4- -d: départements` (champs 1, 4 et suivants)

paste : juxtaposition en colonnes de fichiers
-d : suite des séparateurs
`paste -d+= fichier1 ...`

4.5.5 Sur des cryptages, compressions, impressions, archivages :

touch : modification de la date de dernière modification d'un fichier

crypt : cryptage / décryptage de fichiers
`crypt clé <fichier_en_clair >fichier_encrypté`
`crypt clé <fichier_encrypté`
n'est pas implémenté sur tous les systèmes

compress : compression → suffixe .Z
uncompress : décompression

gzip : compression → suffixe .z ou .gz
décompression option **-d**, **gunzip** ou **unzip**

lpr : impression
lpr -Pcode imprimante -#nombre_impressions fichier ...
lpq : affichage de la queue d'impression
lprm : suppression d'un job dans la queue d'affichage
tar : lecture/écriture sur supports magnétiques (disquettes, bandes ...)
archivage (compilation de fichiers en un seul) et désarchivage

lecture : *tar -tvf archive* (d'une archive)
tar -tvf /dev/fd0 (d'un support magnétique)

archivage : *tar -cvf archive fichier_1 ... fichier_n*
tar -cvf archive répertoire_1 ... répertoire_n
→ - après l'option *f* signifie que *tar* lit/écrit sur l'entrée/la sortie standard :
tar -cvf - fichiers ¿archive
tar -cvf /dev/fd0 ... (création sur disquette)
tar -cvMf /dev/fd0 ... (en multivolumes)
tar -rvf /dev/fd0 ... (ajout sur disquette)

désarchivage : *tar -xvf archive* (fichiers d'une archive)
tar -xvf /dev/fd0 (fichiers d'une disquette)
→ option *z* pour décompresser une archive compressée avec *gzip*
tar -xzvf archive
→ option *k* pour conserver des fichiers préexistants
tar -xkvf archive

remarque : le lecteur de disquettes d'une station se nomme */dev/rfd0* ou */dev/fd0*
sur Linux il se nomme */dev/fd0*

4.5.6 Montage/démontage de systèmes de fichiers :

mount : montage d'une arborescence de fichiers
forme générale : *mount device répertoire*
du lecteur de disquette : *mount /dev/fd0 /mnt/floppy.vfat*
du lecteur de disquette "msdos" : *mount -o conv=auto -t msdos /dev/fd0 /mnt*
(ici les fichiers de format msdos seront automatiquement convertis au format Unix)

/etc/fstab :
les montages doivent être préalablement spécifiés dans ce fichier, dont voici un exemple sous Linux :

```
...
/dev/hda6      /home            ext2      defaults      1 2
/dev/fd0       /mnt/floppy.vfat vfat      noauto,user   0 0
/dev/cdrom     /mnt/cdrom       iso9660   noauto,ro     0 0
...
```

→ la seconde partition logique du premier disque dur (hda6) est montée sur le répertoire /home
→ la disquette pourra être montée (noauto) sur le répertoire /mnt/floppy.vfat par n'importe quel utilisateur (user)
→ le cdrom pourra être monté (noauto) sur le répertoire /mnt/cdrom en lecture seulement (ro)

4.5.7 ... et par conséquent lecture/écriture sur une disquette, un cédérom, une clef USB ... :

Vous devez, si le montage et le démontage ne sont pas fait automatiquement (ce qui est la cas sur les versions récentes) :
- monter le périphérique (utilisez **usermount**)
- opérer les commandes adéquates (via l'interpréteur de commandes (avec cp ...) ou le bureau)
- et ne pas oublier de démonter le périphérique toujours en utilisant **usermount**

Pour les clefs USB, le montage est spécifié dans */etc/fstab* en les désignant comme disque amovible avec */dev/sda1*

5 Outils :

5.1 Outils de communication :

5.1.1 Introduction aux réseaux dans le monde Unix :

Les réseaux servent :
– aux transferts d'informations,
– aux partages de ressources.

La normalisation est effectuée par l'International Standards Organization (ISO) qui a édicté la norme OSI (cela-dit cette normalisation n'est absolument pas respectée) :

Open Systems Interconnection

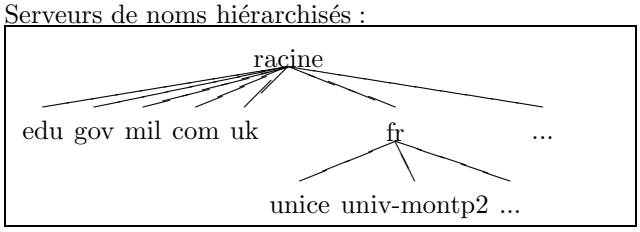
Norme OSI	Type des services	Monde Unix/Internet
1 : Physique	Suite de bits	Ethernet
2 : Liaison de données	Paquets entre deux points	Réseaux publics
3 : Réseau	Routage entre deux nœuds	IP
4 : Transport	Transport entre deux points (et donc multiplexage)	TCP-UDP
5 : Session	Session de communication	RPC
6 : Présentation	Codage normalisé des infos	XDR
7 : Application	Services standards	FTP, TELNET, SMTP, RLOGIN, RCP, RSH, SSH, X11

Internet :
Adresses sur 32 bits :

	identifiant réseau	machine
Grands réseaux (classe A) :	bits (0...)	24 bits
Réseaux moyens (classe B) :	16 bits (10...)	16 bits
Petits réseaux (classe C) :	24 bits (110...)	8 bits

n1.n2.n3.n4 (4 nombres représentant 4 groupes de 4 octets)
machine.site.pays ou machine.site1.site2.pays

Serveurs de noms hiérarchisés :



– Adresse symbolique convertie dans le fichier /etc/hosts
  Fichier /etc/hosts ou hosts.byname
  193.49.105.136 ncda6 # NCD P. Pompidor
– Adresse symbolique convertie sur le serveur de noms le plus proche (INRIA)
– Recherche par nslookup

Ethernet :
réseau local développé par Rank Xerok
– Novel (dirigé PC)
– AppleTalk (dirigé Mac)

5.1.2 Les outils de communication multi-plateformes (pour Internet) :

ssh : *ssh -l login machine* ou *ssh login@machine*

a rendu obsolète et dangereux :

telnet : *telnet machine*
Création d'un shell sur la machine distante
Caractère d'échappement `^]` pour passer en mode commande
- help
- !Commande sur le shell local
- set escape ...
qui n'encrypte pas les trames réseaux

ncftp :	Transfert de fichiers sur une machine distante	
Connexion :	**ncftp machine**	connexion en tant qu'anonyme
	ncftp -u login machine	connexion avec login
Site local :	**l**commande_shell	
	lcd	changement de répertoire
	lls	liste des fichiers
	lmkdir	création d'un répertoire
	...	
Site distant :	**cd**	changement de répertoire
	ls	liste des fichiers
	mkdir	création d'un répertoire
	...	
Transferts	**get *.c**	importation en utilisant le joker "*"
	put *.java	exportation en utilisant le joker "*"
Complétion des noms de fichiers avec ¡Tab¿		

a rendu obsolète

ftp :	Transfert de fichiers sur une machine distante	
Connexion :	*ftp machine*	puis saisie du mot de passe
Site local :	**!**commande_shell	
	lcd	changement de répertoire
Site distant :	**cd**	changement de répertoire
	ls	liste des fichiers
	ls répertoire	
	mkdir	création d'un répertoire

Utilisation de ftp en mode non interactif :

```
ftp -n > f << //
? open adresse_IP_machine
? user anonymous mot_de_passe
? ls -lR
? quit
? //
```

Copie de fichiers de la machine locale vers la machine distante :

put	nom_fichier_local [nom_copie]
append	nom_fichier_local nom_fichier_distant
mput	nom_fichier_local1 ... nom_fichier_local_n

Copie de fichiers de la machine distante vers la machine locale :

get	nom_fichier_distant [nom_copie]
mget	nom_fichier_distant1 ... nom_fichier_distantn
prompt	bascule mode interactif / non interactif

Sessions :

open :	ouverture de la connexion
close :	fermeture de la connexion sans quitter ftp
quit ou **bye** ou **ctrl D** :	fin de ftp

glob (dés)activation du mécanisme d'expansion (* et ?)

5.1.3 Les outils de communication Unix :

uname -a : affichage des informations sur la machine locale
nom_système nom_machine révision version Nom_modèle_machine
exemple : SunOs larzac 4.1.2 2 sun4c

rlogin : `rlogin adresse\_internet [-l login]`
Ouverture sans mot de passe si le login et le nom de la machine de l'utilisateur qui se connecte sur un compte d'un utilisateur de la machine distante sont référencés dans le fichier système **.rhosts** du répertoire courant du second utilisateur.
Caractère d'échappement tilda
!tilda sous le shell local
ctrl D retour au shell distant
rlogin transmet l'environnement UNIX

rsh (ou remsh) : `rsh adresse_internet [-l nom_login] commande`
Exécution d'une commande sur un système distant si référence dans .rhosts
Déspécialisation des caractères spéciaux pour la machine distante
`rsh mimosa.unice.fr who > fich` → machine locale
`rsh mimosa.unice.fr who \> fich` → machine distante

rcp : *rcp source cible* avec source ou cible login@machine :fichier
Autorisation et déspécialisation similaires

5.2 Editeurs de texte :

5.2.1 Liste des principaux éditeurs disponibles :

ed : le plus ancien
vi : éditeur pleine page

xedit : éditeur X très simple
(Couper/Copier/Coller à la souris,
Ctrl W pour effacer une zone sélectionnée,
Ctrl R pour rechercher ou remplacer une chaîne de caractères)
dxnotepad : éditeur X évolué
axe : le prochain

(x)emacs : complexe mais intéressant (macros ...)

OpenOffice/StarOffice : suite bureautique
ooffice ou *soffice*

5.2.2 Aperçu de vi :

Cet éditeur possède deux modes :
– mode commande (positionné au lancement)
– mode insertion

Passage du mode commande au mode insertion :
– i : insertion avant la position du curseur
– a : insertion après la position du curseur
– A : insertion à la fin de ligne
– o : insertion après la ligne
– O : insertion avant la ligne

Passage du mode insertion au mode commande :
Touche Escape ou F11

Déplacements :
Par les flêches sur certains terminaux

1G :	au début du texte
nG :	à la nième ligne
G :	à la fin du texte
Ctrl P :	à la ligne précèdente
Return / Ctrl N :	à la ligne suivante
Blanc :	au caractère suivant
Backspace :	au caractère précèdent
Ctrl B :	à la page précèdente
Ctrl F :	à la page suivante

Suppressions :

x :	suppression du caractère
dd :	suppression de la ligne
D :	suppression de la fin de ligne

Copier / Coller une ligne :

Y :	pour copier
p :	pour coller

Sauver les modification : :w nom_fichier

Quitter :
:q

5.2.3 Aperçu de (x)emacs :

Emacs est un éditeur de texte exclusivement réservé aux programmeurs car en fait, le texte à éditer est une programme qui va être compilé ;-) (et en plus en plusieurs étapes...). Ce monde à part ne sera pas vraiment évoqué ici.

Liste des raccourcis claviers les plus fréquemment utilisés (C- signifie contrôle) :

<**C-x**> <**C-f**>	pour ouvrir un fichier (existant ou nouveau) le nom du fichier peut être préfixé par un chemin de répertoire quelconque
<**C-x**> <**C-s**>	pour sauver
<**C-x**> <**C-c**>	pour quitter
<**C-s**>	pour chercher une chaîne de caractères en avant dans le texte
<**C-r**>	pour chercher une chaîne de caractères en arrière dans le texte
<**Alt-x**>	pour passer en mode commande
<**Alt-x**> **compile**	pour compiler le fichier courant remplacer *make -k* par la commande de compilation ad hoc au langage que vous utilisez une nouvelle fenêtre est créée par emacs dans laquelle vont s'afficher les erreurs de compilation sélectionner la ligne avec le bouton central de la souris (ou en cliquant avec les deux boutons pour les souris à deux boutons) pour se positionner sur la ligne fautive
<**Alt-g**>	pour se positionner sur une ligne par son numéro
<**C-g**>	pour annuler le passage en mode commande

Couper/Copier/Coller :	<**Sélection à la souris**> <**C-w**>	pour couper
	<**C-y**>	pour coller/copier
Subdivision d'Emacs :	<**C-x**> **2**	pour créer deux fenêtres faire <**C-x**> <**C-f**> pour ensuite charger le fichier désiré
	<**C-x**> **n**	pour créer *n* fenêtres
	<**C-x**> **0**	pour détruire la fenêtre où se trouve le curseur actif
	<**C-x**> **1**	pour ne garder que la fenêtre où se trouve le curseur actif (bien entendu, une fenêtre est active si vous avez cliqué en son sein)

6 Scripts exécutés par l'interpréteur de commandes :

Les syntaxes des scripts décrites ici sont celles pouvant être interprétées par un shell :
- **csh** (ou tcsh), s'il débute par `#!/bin/csh`
- **bash** (par défaut sous de nombreuses distributions), s'il débute par `#!/bin/bash`

Voici les fichiers de configuration exécutés :
- Sous **csh** ou **tcsh** :
  - /etc/csh.cshrc
  - /etc/csh.login
  - **.cshrc** (dans votre répertoire d'accueil)
- Sous **bash** :
  - */etc/profile*
  - *.bash_profile* (dans votre répertoire d'accueil)
  - **.bashrc** (dans votre répertoire d'accueil)

Ces fichiers de configuration vous permettent de :
- compléter la variable d'environnement **PATH** qui liste tous les répertoires dans lesquels des exécutables peuvent être retrouvés par l'interpréteur de commandes :

  Sous csh/tcsh : **setenv PATH "${PATH} :."**
  Sous bash : **export PATH="$PATH :."**
- de définir des alias :

  Sous csh/tcsh : alias nouvelle_commande 'ancienne_commande'
  alias c 'clear'
  alias bye exit

  Sous bash : alias nouvelle_commande='ancienne_commande'
  alias c='clear'
  Pour créer des alias avec paramètres, il faut créer des fonctions :
  if () { find . -name "$1" -print 2>/dev/null}
- changer la valeur du prompt :
  - Sous csh/tcsh, par exemple avec **set prompt='%~ $ '**
    - %~ : nom du répertoire courant
    - %t : heure
  - Sous bash, par exemple avec **PS1='\W $ '**
    - \W : nom du répertoire courant
    - \w : chemin du répertoire courant
    - \u : login de l'utilisateur

Analysez la configuration suivante sous csh : `alias cd 'chdir \!1; set prompt="‘pwd‘> "'`

6.1 Les variables :

L'accès à la valeur d'une variable est signifié en faisant précéder le nom de la variable par $.

6.1.1 Les variables d'environnement système :

Voici les principales variables d'environnement système :

DISPLAY :	adresse_IP :numéro_du_serveur_X.numéro_d_écran où les applications X vont s'afficher
HOME :	répertoire d'accueil
PATH :	liste des répertoires où le shell cherche les exécutables à exécuter
TERM :	type de terminal (par exemple xterm)
LOGIN :	login
SHELL :	shell (par exemple /bin/bash)
PWD :	répertoire courant
PS1 :	prompt (sous *bash*)

6.1.2 Déclaration / initialisation des variables :

Déclaration / initialisation d'une variable : `set nom_variable = valeur`
Une variable peut être instanciée par le résultat d'une commande : `set variable = ‘commande‘`

6.1.3 Calcul arithmétique :

Sous *(t)csh* : @ : `@ nom_variable = expression arithmétique`
Sous *bash* : directement

6.1.4 Paramètres de la ligne de commande :

Les paramètres de la ligne de commande sont accessibles par les variables suivantes :

(t)csh	bash	
`$#argv` :	`$#`	nombre de paramètres
`$argv` :	`$*`	liste des paramètres
`$0` :	`$0`	nom du script
`$argv[n]` :	`$n`	nième paramètre (raccourci $n)

6.2 Structures de contrôles des scripts :

if :
Sous *(t)csh* :

```
if (expression conditionnelle) then
        commandes
else                            (facultatif)
        commandes
endif
```

Sous *bash* :

```
if [ expression conditionnelle ]
then
        commandes
else                            (facultatif)
        commandes
fi
```

switch :

```
switch (valeur d'une variable)
        case ...:
                commandes
        breaksw
        ...
        default :               (facultatif)
                commandes
endsw
```

repeat :

```
repeat entier commande
```

while :
Sous *(t)csh* : while :

```
while (expression conditionnelle)
        commandes
end
```

Sous *bash* :

```
while [expression conditionnelle]
do
        commandes
done
```

foreach, for :
foreach :
Sous *(t)csh* :

```
foreach variable (liste de valeurs)
        commandes
end
```

Sous *bash* :

```
for variable in liste
do
        commandes
done
```

6.3 Opérateurs :

Les opérateurs sont ceux du C :
– ==, !=, <, <=, >, >=
– !
– +, -, *, /
– ++, –

Un type d'expression spécifique permet de tester le statut d'un fichier.
Ce type d'expression a la forme générale : **-spécification référence**

d	type répertoire
f	type ordinaire (fichier régulier)
e	existence du fichier
o	propriété du fichier
r	droit de lecture
w	droit d'écriture
x	droit d'exécution
z	taille nulle

Par exemple, lister tous les répertoires du répertoire courant :
Sous *csh* :

```
!#/bin/csh
set liste=‘ls‘
foreach i ($liste)
     if (-d $i) then
          echo $i est un répertoire
     endif
end
```

Sous *bash* :

```
!#/bin/bash
for i in *
do
     if [ -d $i ]
     then
          echo $i est un répertoire
     fi
done
```

6.4 Divers :

echo :	affichage à l'écran de ce qui suit (**-n** pour ne pas passer à la ligne)
exit entier :	sortie du script avec instanciation de la variable système **status**
$status :	valeur retournée par le dernier processus

6.5 Exemples de scripts (*tcsh*) :

Script *liste_paramètres* :

```
#! /bin/csh
echo Le script s\'appelle $0
echo Il a $#argv parametres qui sont : $argv
echo A l\'envers :
set i=$#argv
while ($i > 0)
      echo $argv[$i]
      @ i--
end
```

Script *somme* (réalise la somme de ses arguments) :

```
#! /bin/csh
clear
set resultat = 0
foreach i ($argv)
        @ resultat = $resultat + $i
end
echo Le résultat est $resultat
```

Script *compte* (compte les fichiers réguliers dans la sous-arborescence passée en paramètres) :

```
#! /bin/csh
cd $1
set cumul = 0
set liste = `ls`
foreach i ($liste)
        if (-d $i) then
                compte $i  # Appel récursif du script compte
                @ cumul = $cumul + $status
        else
                @ cumul++
        endif
end
echo il y a $cumul fichiers dans `pwd`
exit $cumul
```

7 Ecriture de scripts PYTHON et PERL :

7.1 Introduction :

Python et **Perl** sont les deux langages interprétés les plus adéquats pour réaliser des scripts système, portables sur Unix et sur Windows. Tous les deux gèrent automatiquement la mémoire et sont très conciliants sur les déclarations de types (Perl d'ailleurs beaucoup plus que Python).
Depuis cette année Python est utilisé pour l'écriture de scripts système à la place de Perl qui lui même avait frappé d'obsolescence les scripts (t)csh ou bash. En effet, synthèse surprenante de C, d'interpréteurs de commandes SHELL et d'utilitaires UNIX (sed, awk ...), Perl est un langage d'une syntaxe difficile. Ainsi, bien que moins efficace en terme de rapidité d'exécution, Python lui est de plus en plus préféré pour sa syntaxe objets "propre" (mais néanmoins imparfaite pour les puristes "objets"). **La définition de classes ne sera pas décrite dans ce polycopié, mes compétences en programmation "objets" me dissuadant de les mettre sur papier.**

7.2 Références bibliographiques principales

Pour Python :
Introduction à Python, Mark Lutz & David Ascher, O'Reilly
Programming Python, 2nd Edition, Mark Lutz, O'Reilly

Pour Perl :
L'émouvant Programmation en PERL, troisième édition en français, Larry Wall, O'Reilly

7.3 Comment exécuter un script

Il y a deux modes de lancement possibles pour un script Python ou Perl :
– en le passant en paramètre à l'interpréteur Python ou à l'interpréteur Perl :

  Python : python script_Python.py

  Perl : perl script_Perl.pl

– en l'exécutant directement sur la ligne de commande à condition que le script soit exécutable et qu'il comporte une première ligne particulière qui permette à l'interpréteur de commandes de savoir à qui il doit déléguer le script :

  Python : #!/usr/bin/python ou mieux encore **#!/usr/bin/env python**

  Perl : #!/usr/bin/perl ou mieux encore **#!/usr/bin/env perl**

Que ce soit en Python ou en Perl, il est également possible d'exécuter des instructions de manière interactive sous l'interpréteur (tapez respectivement **Python** ou **Perl** dans le terminal).

Le script peut être accompagné de paramètres ou inséré dans une ligne de commandes :

directement :	**script**
directement avec paramètres :	**script paramètre_1 paramètre_2 ...**
avec en entrée le contenu d'un fichier :	**cat fichier | script**
avec en entrée le contenu d'un fichier et des paramètres :	**cat fichier | script paramètre_1 paramètre_2 ...**

Les paramètres sont accessibles :
En Python dans la liste **sys.argv** (voir paragraphe 6.5.2)
En Perl dans le tableau **@ARGV** (voir paragraphe 6.5.2)

Que ce soit en Python ou en Perl, les lignes (ou fin de lignes) en commentaires sont précédées par #

7.4 Définitions des blocs d'instructions :

En Python :
Les instructions qui définissent les blocs (gouvernés par des structures conditionnelles ou itératives) doivent débuter sur la même colonne. Par ailleurs les expressions conditionnelles n'ont pas besoin d'être encadrées par des parenthèses.

Voici l'exemple d'un test ("if") inséré dans une boucle ("for") :

```
for ligne in lignes:
    res = re.search(r"^(.*)\s+verbe", ligne)  # application d'une expression régulière
    if res:
        print res.group(1)
```

En Perl :
Les instructions qui définissent les blocs doivent être encadrées par des accolades (très classiquement).
Voici la traduction de l'exemple précédent en Perl :

```
for (@lignes) {
    if (/^(.*)\s+verbe/o) {  # application d'une expression régulière
        print "$1\n";
    }
}
```

Le test "if" et la boucle "for" seront décrits plus loin.

7.5 Les variables :

7.5.1 Les variables individuelles (ou scalaires) :

En Python :
- Elles ne sont pas typées
  - chaine = "chaîne"
  - reel = 3.14
- Les chaînes de caractères peuvent être encadrées soit par des guillemets, soit par des quotes simples, ce qui permet, par exemple, l'inclusion de quotes entre guillemets :
  chaine = "J'aime les brocolis"
- Le transtypage n'est pas automatiquement effectué suivant le contexte :
  variable = '123' ; print int(variable) + 1, '\n'

En Perl :
- Leurs noms sont **toujours** préfixés par **$**
- Elles ne sont pas typées
  - $chaine = "chaîne" ;
  - $reel = 3.14 ;
  - $resultat_de_commande = `pwd` ;
- Les chaînes de caractères peuvent être encadrées soit par des guillemets, soit par des quotes simples, mais dans le cas où elles sont encadrées par des guillemets, elles autorisent l'interpolation de variables :
  $var1 = 'vous' ;
  $var2 = "Bonjour $var1" ;
- Le transtypage est automatiquement effectué suivant le contexte :
  $variable = '123' ; print $variable + 1, "\n" ;

7.5.2 Les listes (appelées tableaux simples en Perl) :

Les tableaux simples sont des tableaux d'éléments indicés par des entiers à partir de 0.

En Python :
- Ils doivent être initialisés (au pire à rien : liste = [])
- Ils peuvent être initialisés par une liste de valeurs :
  jours = ['dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi']
- Longueur d'une liste (appelée ici L) : len(L)
- Tranche d'une liste : L[i :j]
- Concaténation de listes : L1 + L2
- Tri d'une liste : L.sort()
- Inversion d'une liste : L.reverse()

En Perl :
- ils sont préfixés par **@**
- Ils peuvent être initialisés par une liste de valeurs :
  `@jours = ('dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi')`
- ($sunday, $monday, ...) = @jours ;
- `$#nom_tableau` donne l'indice du dernier élément du tableau
- **@ARGV** est instancié par les paramètres du script (**$ARGV[0]**, ...)
- Ils peuvent être créés à la volée

7.5.3 Les dictionnaires (appelés tableaux associatifs en Perl) :

Les éléments des dictionnaires sont **indexés** par des chaînes de caractères au lieu d'être indicés.

En Python :
- Ils doivent être initialisés (au pire à rien : dictionnaire = {})
- Ils peuvent être initialisés par une liste de couples de valeurs (clef, valeur) :
  mois = { 'janvier' : 31, "février" : 28, ... }
- Pour accéder à la valeur d'un élément, donnez sa clef entre crochets :
  print mois['janvier']
- Liste des clefs avec la méthode **keys()** : noms_mois = mois.keys()
- Liste des valeurs avec la méthode **values()** : nb_jours = mois.values()
- Test de l'existence d'une clef : **has_key(clef)**

En Perl :
- Ils sont préfixés par **%**
- Initialisation par liste de couples de valeurs (clef, valeur) :
  %mois = ('janvier' =¿ 31, 'février' =¿ 28, ...)
- Exemple d'utilisation : $mois{janvier} ou $mois{'janvier'}

7.6 Les structures conditionnelles et itératives :

7.6.1 Les structures conditionnelles :

En Python :

if *expression conditionnelle* **:**
<tabulation> *instruction*
<tabulation> *instruction*
else :
<tabulation> *instruction*
<tabulation> *instruction*

La réalisation d'un SWITCH peut être effectué par l'instruction **elif** :
if *expression conditionnelle* **:**
<tabulation> *instruction*
<tabulation> *instruction*
elif *expression conditionnelle* :
<tabulation> *instruction*
<tabulation> *instruction*

En Perl :

if (*expression conditionnelle*)
{ ... }
else
{ ... }

La réalisation d'un switch peut être effectué par l'utilisation de labels :

```
SWITCH :
{
        if ( ... ) { ... ; last SWITCH ; }
        if ( ... ) { ... ; last SWITCH ; }
        ...
}
```

Pas d'équivalence du switch C/JAVA
*Les labels (ici SWITCH) et la commande **last** sont présentés plus loin*

7.6.2 Les structures itératives :

En Python :

Les boucles sont réalisées soit par un “while”, soit par un “for” :
while *expression conditionnelle* **:**
<tabulation> *instruction*
<tabulation> *instruction*
<tabulation> ...

for *variable* in *liste de valeurs* **:**
<tabulation> *instruction*
<tabulation> *instruction*
<tabulation> ...

Exemple sur la liste des jours avec un “while” :

```
jours = ['dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi']
num_jour = 0
while num_jour < 7 :
    print jours[num_jour]
    num_jour = num_jour + 1
```

Exemple sur la liste des jours avec un “for” :

```
jours = ['dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi']
for jour in jours :
    print jour
```

En Perl :

while (*expression conditionnelle*)
{ ... }

Un cas fréquent : la recherche d'un motif sur les lignes d'un fichier passé sur l'entrée standard :
while (<STDIN>)
{
if (/expression régulière/) {...}
}

for (initialisation de l'indice ; expression conditionnelle ; incrémentation/décrémentation)
{ ... }

for *($i=0 ; $i < 10 ; $i++)*
{ ... }

foreach $variable (*liste de valeurs*)
{ ... }

Traitement sur une liste d'utilisateurs :
foreach $user (@users)
{...}

7.6.3 Les commandes de contrôle des structures itératives :

En Python :

break	sort de la boucle
continue	retourne au début de la boucle

En Perl :

chaîne de caractères :	pose une marque (un label)
goto	permet de sauter au label concerné
last	sortie immédiate de la boucle
next	saut immédiat à la prochaine itération
redo	saut immédiat à la prochaine itération sans évaluer l'expression conditionnelle

7.7 Les opérateurs de comparaison :

En Python :

nombres ou chaînes	
==	vrai si le premier opérande est égal au second
!=	vrai si le premier opérande n'est pas égal au second
<	vrai si le premier opérande est inférieur au second
>	vrai si le premier opérande est supérieur au second
<=	vrai si le premier opérande est inférieur ou égal au second
>=	vrai si le premier opérande est supérieur ou égal au second

En Perl :

nombres	chaînes	
==	eq	vrai si le premier opérande est égal au second
!=	ne	vrai si le premier opérande n'est pas égal au second
<	lt	vrai si le premier opérande est inférieur au second
>	gt	vrai si le premier opérande est supérieur au second
<=	le	vrai si le premier opérande est inférieur ou égal au second
>=	ge	vrai si le premier opérande est supérieur ou égal au second
<=>	cmp	0 si égal, 1 si le premier opérande plus grand, -1 si c'est le second

Un type d'expression spécifique permet de tester le statut d'un fichier.
Ce type d'expression a la forme générale : **-spécification référence** :
d type répertoire, **f** type ordinaire (fichier régulier), **T** type fichier texte, **e** existence du fichier,
r droit de lecture, **w** droit d'écriture, **x** droit d'exécution

7.8 Les expressions régulières :

Une expression régulière définit une chaîne de caractère quelconque qui va être recherchée dans une autre chaîne de caractères (qui peut être la valeur d'une variable, d'une ligne d'un fichier ...).

En Python :
- Le module **re** doit être importé : **import re**
- Pour plus d'efficacité les expressions régulières doivent être compilées avec la méthode **compile**
- Elles sont appliquées avec la méthode **search**
- Les motifs retrouvés sont accessibles avec la méthode **group**

```
recherche = re.compile(r"endives (.*)")  # l'expression régulière est compilée
res = re.search(recherche, "J'aime aussi les endives au jambon")
if res:
    print res.group(1)  # le motif extrait est ''au jambon''
```

En Perl :

Par défaut, elle s'applique sur la variable $_ (qui contient notamment la dernière ligne lue sur un fichier).
Les expressions régulières sont encadrées par des /.
La variable $& va contenir la chaîne de caractères appariée.
Les expressions régulières sont appliquées à une variable par le "binding opérator" :

expression appliquée sur une variable	`$variable =~ /expression régulière/`
comptage du nombre d'appariement	`$variable = /expression régulière/`
motifs appariés dans un tableau	`($var1, $var2, ...) = /...(...)...(...).../`
	`@tab = /...(...)...(...).../`
exemple	`($membres_des_groupes) = /^\w+:[^:]*:${ARGV[0]}:(.*)$/`

7.8.1 Syntaxe normalisée des expressions régulières :

.	n'importe quel caractère
|	exprime l'alternative
	pierre |paul |jacques ↔ pierre ou paul ou jacques
()	groupement de caractères
	(effet secondaire : le groupement de caractères est accessible par la méthode re.group(...) en Python
	mémorisé dans une variable nommée de $1 à $9 en Perl)
(? :)	supprime la mémorisation dans une variable
(?=)	les caractères appariés ne sont pas mémorisés dans $&
[]	constitution d'un ensemble de caractères dans lequel **un** caractère pourra être choisi
	[a-zA-Z] : **un** des caractères alphabétiques
^	appariement en début de ligne
$	appariement en fin de ligne
*****	répétition de 0 à n fois du caractère (ou du groupement de caractères) précédent
+	répétition de 1 à n fois du caractère (ou du groupement de caractères) précédent
?	répétition de 0 à 1 fois du caractère (ou du groupement de caractères) précédent
{n}	répétition n fois du caractère (ou du groupement de caractères) précédent
{n,}	répétition au moins n fois du caractère (ou du groupement de caractères) précédent
{n,m}	répétition de n à m fois du caractère (ou du groupement de caractères) précédent
	moo{3} ↔ moooo, (moo){3} ↔ moomoomoo
\d	un chiffre, soit un élément de l'ensemble suivant : $[0-9]$
\w	un caractère alphanumérique, soit un élément de l'ensemble suivant : [a-zA-Z_0-9]
\w+	un mot composé des caractères alphanumériques précédents
\W	un caractère non alphanumérique
\b	une frontière de mot (entre \w et \W)
\s	un élément de l'ensemble suivant : [\t\n\r\f]
\S	tout élément non compris dans l'ensemble précédent

7.8.2 Remplacement de chaînes de caractères :

En Python :

Les remplacements sont effectués par la méthode **sub** du module **Python re** :
resultat = re.sub(sous-chaîne_à_remplacer, sous-chaîne_de_remplacement, chaîne_cible)

```
chaine = "Mais ce que j'adore le plus ce sont les anchois"
chaine = re.sub ('anchois', 'anchois marinés à la catalane', chaine)
```

En Perl :

chaîne_cible =~ **s**/chaîne_à_remplacer/chaîne_de_remplacement/options

Principales options possibles :
g remplace toutes les occurrences, **i** ne distingue pas les minuscules des majuscules

Exemples :

s/^([^]+) +([^]+)/$2 $1/	inversion des deux premiers mots
s/\bmot1\b/mot2/g	remplacement de tous les "mot1" par "mot2"
$nombre =~ s/\bmot1\b/mot2/g	remplacement et comptage du nombre de remplacement

7.8.3 Remplacement de caractères (en Perl) :

tr/ensemble de caractères à remplacer/ensemble de caractères de remplacement/
Exemple :

tr/`[A-Z]`/[a-z]/	remplacement des majuscules par des minuscules

7.9 Quelques fonctions prédéfinies de base :

	en Python	En Perl
appel d'un exécutable	**os.system("...")**	**system** "..."
résultat d'un exécutable	liste = os.popen("...").readlines()	@tableau = '...'
affichage sur la sortie standard	**print** liste_de_valeurs_à_afficher	**print** liste_de_valeurs_à_afficher
découpage suivant un motif	liste = string.split(chaîne, séparateur)	@tableau = **split** /motif de séparation/, chaîne

7.10 Quelques fonctions prédéfinies sur les tableaux (en Perl) :

Simples :

extraction du premier élément	**shift** tableau
insertion en tête	**unshift** tableau, éléments
extraction du dernier élément	**pop** tableau
insertion en queue	**push** tableau, éléments
tri d'un tableau	**sort** tableau

Associatifs :

accès aux clefs	**keys** tableau_associatif
	`foreach $clef (%tableau_asso) {print "$clef: $tableau_asso{$clef}\n";}`
accès aux valeurs	**values** tableau_associatif
accès au binôme clef/valeur	($clef, $valeur) = **each** tableau_associatif
	`while (($clef, $valeur) = each %tableau_asso) {print "$clef: $valeur\n";}`

7.11 Quelques fonctions prédéfinies sur les fichiers :

En Python :
- Ouverture d'un fichier avec **open** : **fd = open("fichier", "mode d'ouverture")**
- Lecture d'une ligne dans une variable : variable = **fd.read()**
- Lecture de toutes les lignes dans une liste : liste = **fd.readlines()**
- Ecriture d'une ligne : **fd.write(variable)**
- Ecriture de plusieurs lignes : **fd.writelines(liste)**
- Fermeture : **fd.close()**

Ouverture et mémorisation de toutes les lignes d'un fichier dans une liste, puis affichage :

```
lignes = open("essai", "r").readlines()
for ligne in lignes :
    print ligne,  # la virgule empêche la fonction print de rajouter un retour-chariot après chaque
                  # affichage (il y en a déjà un en fin de chaque ligne du fichier)
```

En Perl :

lecture sur l'entrée standard	<>	`$entree = <>` ou `$entree = <STDIN>`
	boucle de lecture sur un tube	`while (<STDIN>) {...}`
ouverture d'un fichier	**open** descripteur_fichier, nom_fichier	
	ouverture en lecture seule	`open FICHIER, "fichier"`
	ouverture en écriture	`open FICHIER, ">fichier"`
	ouverture en ajout	`open FICHIER, ">>fichier"`
	ouverture en lecture écriture	`open FICHIER, "+<fichier"`
	en lecture d'une commande	`open GROUPES, "ypcat group|";`
lecture d'une ligne du fichier	<descripteur_fichier>	
écriture dans le fichier	**print** descripteur_fichier liste	
fermeture d'un fichier	**close** descripteur_fichier	

La variable **$/** détermine la chaîne de caractères marquant les fins de ligne lors de la lecture d'un fichier.
Elle peut être indéfinie pour lire le fichier en une seule fois : `undef $/; $_ = <FICH>;`
ou par exemple, pour s'arrêter sur les points d'un fichier texte : `$/="."; while (<FICH>) {...};`

7.12 Comparatif Python/Perl/Java :

Affichage de tous les verbes d'un dictionnaire bilingue de 35000 entrées, exemple :
(`abaisser verbe -06 push_down\etre/obj pull_down\etre/obj lower/valeur reduce/tempe'rature`)
Les temps d'exécution réels ont été mesurés avec la commande Linux **time**.
Les instructions d'affichage ont été mises en commentaires.

En Python → **entre 50 et 80 millisecondes** :

```
#!/usr/bin/env python2

import re, os

recherche = re.compile(r"^(.*)\s+verbe")
fichier = open(os.environ['HOME']+'/RECHERCHE/FONGUS/ANALYSEUR/dictionnaire_francais_anglais', 'r')
lignes = fichier.readlines()
fichier.close()

for ligne in lignes:
    res = re.search(recherche, ligne)
    #if res:
       # print res.group(1)
```

En Perl→ **entre 10 et 40 millisecondes** :

```
#!/usr/bin/env perl

open F, $ENV{HOME}.'/RECHERCHE/FONGUS/ANALYSEUR/dictionnaire_francais_anglais';
@lignes = <F>;
close F;
for (@lignes) {
    if (/^(.*)\s+verbe/o) {
        # print "$1\n";
    }
}
```

En Java → **entre 50 et 100 millisecondes** :

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class test_re {
  public static void main(String[] params) throws java.io.IOException {
     Pattern pattern = Pattern.compile("^(.*)\\s+verbe");

     String filename = "/auto/pompidor/RECHERCHE/FONGUS/ANALYSEUR/dictionnaire_francais_anglais";
     java.io.BufferedReader fichier = new java.io.BufferedReader(new java.io.FileReader(filename));

     String ligne;
     Matcher match;
     while ( (ligne = fichier.readLine()) != null) {
         match = pattern.matcher(ligne);
         if(match.find()) {   // ou matches
             // System.out.println(match.group(1));
         }
     }
     fichier.close();
  }
}
```

7.13 Exemples de scripts :

Affichage des lignes numérotées du fichier passwd d'un ordinateur isolé :

En Python :

```
#!/usr/bin/env python

i = 1
fd = open ('/etc/passwd', 'r')   # ouverture du fichier en lecture
for ligne in fd.readlines() :    # lecture de chaque ligne
    print i, ' : ', ligne        # numérotation de chaque ligne
    i+=1
fd.close()
```

En Perl :

```
#!/usr/bin/env perl

open F, '/etc/passwd';           # ouverture du fichier en lecture
while (<F>) {                    # lecture de chaque ligne
    print ++$i, " : ", $_}       # numérotation de chaque ligne
close F
```

Affichage des logins du fichier passwd géré par NIS et redirigé sur l'entrée standard :

Exemple d'appel : `ypcat passwd | logins.p[y|l]`

En Python :

```
#!/usr/bin/env python2

import sys, re
recherche = re.compile(r"^([^:]+)")

for ligne in sys.stdin.readlines() :   # lecture de chaque ligne de l'entrée standard
    res = re.search(recherche, ligne)  # application de l'expression régulière sur la ligne
    if res:
       print res.group(1)              # affichage du login
```

En Perl :

```
#!/usr/bin/env perl

while(<STDIN>) {
   print /^([^:]+)/, "\n" # Les parenthèses permettent l'extraction du champ et donc son affichage
}
```

Affichage du nom complet (et de l'UID) des utilisateurs dont les logins sont passés en paramètres :

Exemple d'appel : `ypcat passwd | noms_complets.p[y|l] aubert pompidor`

En Python :

```
#!/usr/bin/env python2

import sys, re

for ligne in sys.stdin.readlines() :        # lecture de chaque ligne de l'entrée standard
    for user in sys.argv :                  # pas très malin, pourquoi ?
        recherche = re.compile(user+":\w+:(\w+):\w+:(\w*) (\w*)")
        res = re.search(recherche, ligne)   # application de l'expression régulière sur la ligne
        if res:
            print "Nom complet =", res.group(3), res.group(2), "de uid =", res.group(1)
            # affichage du prénom et du nom
```

En Perl :

```
#!/usr/bin/env perl

while(<STDIN>) {
  foreach $user (@ARGV) {
    if (/^${user}:\w+:(\w+):\w+:(\w*) (\w*)/) {   # \w = caractère alphanumérique [a-zA-Z_0-9]
       print "Nom complet = $3 $2 de uid = $1 \n";
       last
    }
  }
}
```

Affichage des dates et du nombre de connexions de la machine :

Exemple : pompidor : 2 Nov (2 fois), 3 Nov (3 fois), 12 Nov (3 fois), 15 Nov (4 fois)

Exemple d'appel : `last | script_dates_connexions.p[y|l]`

En Python :
Cela sera l'objet d'un TP ....

En Perl :

```
#!/usr/bin/env perl

while (<STDIN>)
{
   /^(\w+)[^A-Z]*\w+\s(\w+)\s+(\w+)/;
   $logins{$1} .= "$3 $2, ";
}

foreach $user (sort keys %logins)
{
   @dates = split /,/, $logins{$user};
   pop @dates;

   undef %dates_uniques;
   foreach $date (@dates) { $dates_uniques{$date}++; }

   print "\n$user : ";
   foreach $date (keys %dates_uniques)
   {
      print "$date($dates_uniques{$date} fois), ";
   }
}
```

8 Appels système :

8.1 Rappels de quelques fonctions C (nécessaires à la réécriture d'un shell) :

main : `int main (int argc, char *argv[], char *arge [])`
argc : nombre de composantes de la commande
éléments de la commande :
argv : tableau de pointeurs sur caractères (fin : pointeur NULL)
nouvel environnement :
arge : tableau de pointeurs sur caractères

printf : affichage à l'écran
`printf ("format", variables ...)`

strcmp : comparaison de deux chaînes de caractères
`#include <string.h>`
`int strcmp (char *string1, char *string2)`
Compare string1 avec string2

strcpy : copie d'une chaîne de caractères dans une autre
`#include <string.h>`
`char* strcpy (char *string1, char *string2)`
Copie string2 dans string1 et retourne string2

exit : abandon du programme en cours
exit (int status)
status = 0 si arrêt normal

getenv : récupération de la valeur d'une variable d'environnement
`#include <stdlib.h>`
`char* getenv (const char *name)`
Retourne la chaîne d'environnement associée à name ou NULL
ligne_de_chemins = (char*) getenv ("PATH") ;

8.2 Appels système à partir d'un programme C :

8.2.1 Les primitives de base sur les fichiers :

access : Test de l'existence d'un fichier
`#include <unistd.h>`
`access (chemin, X_OK)  (R_OK ou W_OK)`
retourne 0 en cas de succès ou -1 sinon

creat : Création de nouveaux fichiers ou réécriture d'anciens
`int creat (char *name, int perms)`
`retourne un descripteur de fichier ou -1`
permission standard : 0755

open : Ouverture de fichiers existants
`int open (char *name, int flags, 0)`
retourne un descripteur de fichier ou -1
si `#include<fcntl.h>`, flags = O_RDONLY, O_WRONLY ou O_RDWR

read : Lecture dans un fichier
`int n_lus = read (int fd, char *buffer, int n)`

write : Ecriture dans un fichier
`int n_écrits = write (int fd, char *buffer, int n)`

close : Fermeture d'un fichier
`close (int fd)`

8.2.2 La primitive de création de processus :

fork : Création dynamique d'un nouveau processus s'exécutant de façon concurrente avec le processus qui l'a créé.

```
#include <unistd.h>
int pid fork ()
```

La valeur de retour est différente dans le processus père et dans le processus fils :
→ 0 dans le processus fils
→ l'identité du processus fils créé dans le processus père
→ -1 si échec

8.2.3 Les primitives de synchronisation père/fils :

Tout processus se terminant passe dans l'état zombie dans lequel il reste aussi longtemps que son père n'a pas pris connaissance de sa terminaison.

wait : Attente des processus fils

```
#include <sys/types.h>
#include <sys/wait.h>
int pid wait (int *pointeur_status)
```

→ Si le processus possède des fils mais aucun zombie, le processus est bloqué jusqu'à ce que l'un de ses fils devienne zombie.
→ Si le processus possède au moins un fils zombie, le processus renvoie l'identité de l'un de ses fils zombie et si l'adresse de pointeur_status est différente de NULL des informations sont fournies sur la terminaison du processus zombie.

waitpid : Attente d'un processus

```
#include <sys/types.h>
#include <sys/wait.h>
int pid waitpid (pid_t pid, int *pointeur_status, int options)
```

Attente bloquante ou non bloquante d'un processus fils particulier.
pid : -1 tout processus fils, ¿0 processus fils particulier
options : 0 bloquant, WNOHANG non bloquant

8.2.4 Les primitives de duplication de descripteurs :

Acquisition par un processus d'un nouveau descripteur synonyme d'un descripteur déjà existant.
retour : descripteur synonyme ou -1 en cas d'échec.

dup : Duplication de descripteurs

```
#include <unistd.h>
int dup (int desc)
```

Force le plus petit descripteur disponible à devenir un synonyme du descripteur desc

dup2 : Duplication de descripteurs

```
#include <unistd.h>
int dup2 (int desc1, int desc2)
```

Force le descripteur desc2 à devenir un synonyme du descripteur desc1

8.2.5 Les primitives de recouvrement :

exec.. : Recouvrement de processus
→ retour : -1 en cas d'erreur
→ exec.p : recherche du fichier dans les répertoires dénotés par PATH
→ exec.e : passage d'un nouvel environnement

execl : `int execl (const char *chemin_ref, const char *arg, ..., NULL)`
Le fichier de nom ref est chargé et la fonction main correspondante est appelée avec la liste des paramètres suivants

execv : `int execv (const char *chemin\_ref, const char *argv [])`

8.2.6 Les primitives de communication :

Sous Unix différents types de communications existent :
– (par fichiers),
– par signaux,
– par tubes,

– par tubes nommés,
– par envoi de messages,
– par sémaphores,
– ou par mémoires partagées.

pipe : Communication par tube non nommé
```
#include <unistd.h>
int pipe (int p[2])
```
Création de deux descripteurs dans la table des processus
p[0] : descripteur en lecture, p[1] : descripteur en écriture
gestion en mode fifo
retour : 0 réussi, -1 échec

8.3 Algorithme de réécriture d'un shell simplifié :

Un shell simplifié doit pouvoir traiter les lignes de commandes suivantes :
`commande_1 < entrée1 | commande ... | commande_n > sortie`

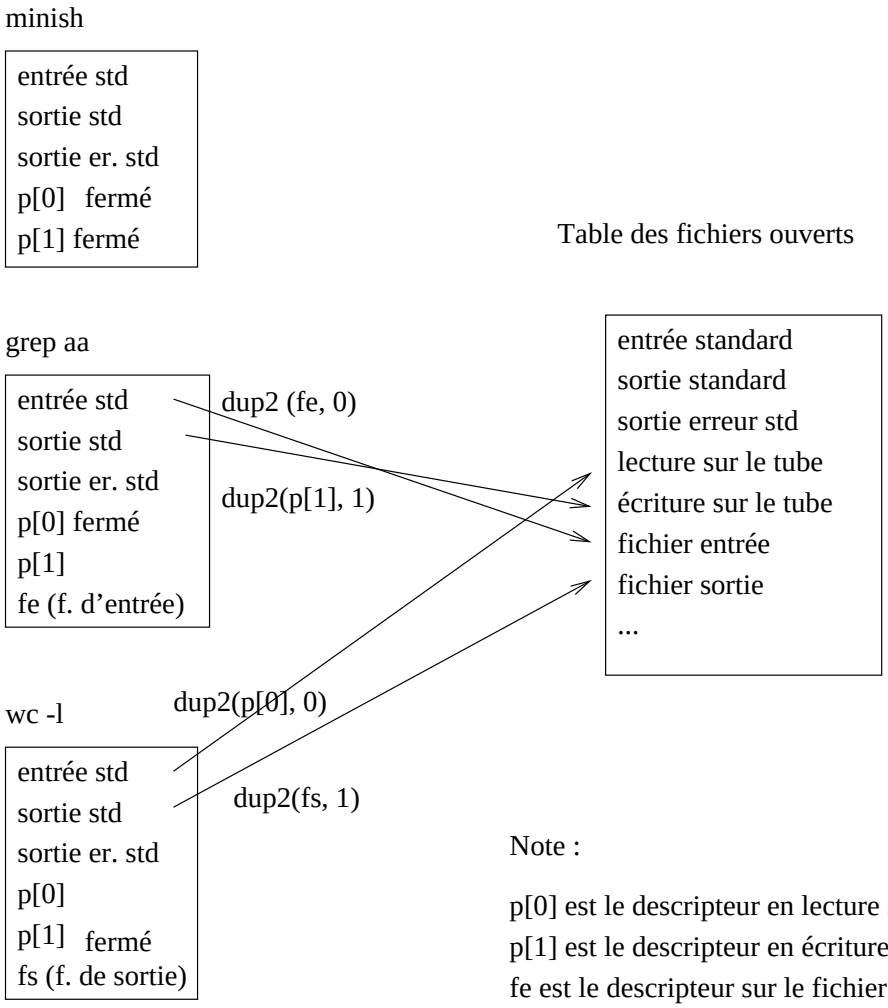
Exemple de l'algorithme traitant simplement deux commandes comme suit :
`grep aa < fichier_entree | wc -l > fichier_sortie`

Lecture d'une ligne de commandes au clavier (**read**)
Création d'un tube (**pipe**)
Création du premier fils (**fork**)
- père : attente (**wait**)
- fils 1 : ouverture du fichier d'entrée (**open**)
- fils 1 : redirection de l'entrée standard sur le fichier d'entrée (**dup2**)
- fils 1 : redirection de la sortie standard sur le tube (**dup2**)
- fils 1 : fermeture du descripteur en lecture du tube (**close**)
- fils 1 : recouvrement (**execvp ou execlp**)

Création du second fils (**fork**)
- père : fermeture des descripteurs du tube (**close**)
- père : attente (**wait**)
- fils 2 : création du fichier de sortie (**creat**)
- fils 2 : redirection de la sortie standard sur le fichier de sortie (**dup2**)
- fils 2 : redirection de l'entrée standard sur le tube (**dup2**)
- fils 2 : fermeture du descripteur en écriture du tube (**close**)
- fils 2 : recouvrement (**execvp ou execlp**)

Illustration de l'effet des appels systèmes **dup2** sur l'exemple précédent :

Tables des descripteurs
des processus



Note :

p[0] est le descripteur en lecture sur le tube
p[1] est le descripteur en écriture sur le tube
fe est le descripteur sur le fichier d'entrée
fs est le descripteur sur le fichier de sortie

9 X Window - Utilisation :

9.1 Historique :

Projet de recherche DEC - MIT
Commercialisation de la version 11 en 1988
Actuellement, 6ème distribution : X11R6

9.2 Le système X Window :

9.2.1 Présentation générale :

X Window est un système de gestion de fenêtres basé sur une architecture client-serveur.
Le serveur reçoit les requêtes graphiques des clients (applications des utilisateurs), et génère les affichages. Il peut également renvoyer des informations aux clients (touches enfonçées du clavier, mouvements de la souris).
Le découplage entre les applications et les contraintes liées aux matériels utilisés permet une grande portabilité des applications. Les clients et les serveurs peuvent être distants et communiquer par réseau (TCP/IP ou DECnet). Un client peut ainsi s'exécuter sur une machine donnée et s'afficher sur les écrans (screens) d'autres machines.

9.2.2 Lancement de X et la variable d'environnement DISPLAY :

- A partir d'une station de travail avec un ou plusieurs écrans bitmaps :
  lancement du script x11start, startx, xstart ou x11
  - ajout de /usr/bin/X11 à la variable PATH
  - Chargement du fichier .Xdefaults via xrdb (ou /usr/lib/X11/sys.Xdefaults)
  - xinit appelant .x11start ou /usr/lib/X11/sys.x11start
    - lancement du gestionnaire de fenêtres (par ex. mwm)
    - lancement d'un xterm
- A partir d'un terminal X :
  - Connection à la charge de xdm
  - Lancement du script /usr/lib/X11/xdm/Xsession
    - .xsession
    - Chargement du fichier .Xdefaults via xrdb (ou /usr/lib/X11/sys.Xdefaults)
      - lancement du gestionnaire de fenêtres (par exemple mwm)
      - lancement d'un xterm

Utilisation de la commande **xhost +** / **xhost -**
pour autoriser / interdire la communication des autres machines avec la sienne.
Chaque serveur est identifié par sa variable d'environnement **DISPLAY** :
`machine:numéro_de_serveur_X.numéro_d_écran`
Exemple : `setenv DISPLAY ncda6:0.0`

9.2.3 Architecture et gestionnaire de fenêtres :

L'architecture du système X Window s'organise comme suit, chaque couche pouvant directement se lier à chacune des couches antérieures :
Applications
Générateurs d'interfaces
Librairie Motif Xm ...)
Toolkit standard (Xt ...)
Librairie Xlib (X ...)

Un programme prédéfini X contrôle l'apparence, le positionnement et les interactions avec l'utilisateur des fenêtres, c'est le gestionnaire de fenêtres.
Différents gestionnaires existent : **twm** : assez ancien
olwm : celui développé par Sun (OpenLook)
mwm : celui de Motif (le plus utilisé, standard de fait)
fmwm : celui de Motif pour Linux

Fonction **Refresh** dans un menu de la souris : réaffichage complet de l'écran
Fonction **Restart** dans un menu de la souris : relance le gestionnaire de fenêtres

9.3 Quelques clients standards et leur paramétrage :

xterm : terminal (fenêtre)
xtalk : utilitaire de communication
xedit : éditeur de texte
xemacs : éditeur de texte
xfig : éditeur de graphiques
xman : documentation
xclock : horloge
xcalc : calculette
xload : charge de la machine
xeyes : scrutation du pointeur de souris
xlsfonts : liste des fontes disponibles
xfontsel : visualisation des fontes
xcolors : visualisation des couleurs disponibles
xmag : zoom sur une partie d'une fenêtre
xdpr : impression d'une fenêtre

Les ressources permettent la modification des paramètres d'une application (tels que son apparence ou son comportement) sans pour autant avoir à la recompiler. Elles sont très utilisées dans les applications construites avec des toolkits permettant de nommer les classes d'objets graphiques (initiales en majuscules) et leurs instances (initiales en minuscules).

Paramétrage d'une ressource : spécification : valeur La spécification décrit la hiérarchie de classes et débute en général par le nom de la classe. Elle utilise le séparateur . et un caractère spécial *.

Il existe trois façons de configurer les ressources (du plus prioritaire au moins prioritaire) :
- Spécification sur la ligne de commande des clients :
  - -display : indication du serveur
  - -geometry : dimensions et positionnement de la fenêtre
  - -background : couleur d'arrière plan
  - -foreground : couleur d'avant plan
  - -reverse : inversion vidéo
  - -iconic : démarrage sous forme d'icône

  Exemple : `xterm -display ncda6:0.0 -geometry largeur*hauteur+abscisse+ordonnee`
- Utilisation des fichiers de configuration lus au lancement de chaque client :
  Dans les fichiers de ressources propres aux applications (noms de classes)
  $APPLRESDIR ou /usr/lib/X11/app-defaults
  comme par exemple le fichier **.Xdefaults**
  Commentaires : lignes précédées de !
- Par téléchargement par xrdb.

Exemples de ressources (par exemple dans le fichier .Xdefaults) :

Mwm*keyboardFocusPolicy :	pointer	! Activation sur entrée
Mwm*keyboardFocusPolicy :	explicit	! Activation en cliquant
Mwm*useIconBox :	True	! Création d'une boîte à icones
Mwm*iconBoxGeometry :	10x1+0-0	! Taille et position de la boîte
XTerm*Font :	fonte	! Sélection d'une fonte

9.4 Paramétrage des boutons de la souris et des touches du clavier :

Ce paramétrage est effectué par le fichier de configuration .mwmrc
(ou .twmrc, .olwmrc, fmwmrc ... suivant les shells utilisés).

9.4.1 Paramétrage des boutons de la souris :

Déclarations des menus par la syntaxe suivante :

```
Menu    nom_du_menu
{
        Options Raccourci clavier       fonction
        ...
}
```

Remarque :
Le raccourci clavier débute par un modifieur : Ctrl, Meta, Alt, Shift ou Lock
Exemple : `"Restart ..."   Alt<Key>F12   f.start`

Fonctions principales pouvant être utilisées :

f.exec : exécute une commande externe
f.menu : appelle un menu
f.kill : tue un client
f.quit_mwm : quitte mwm
f.refresh : réaffiche l'écran
f.restart : relance mwm
...

Remarque : on peut utiliser un bitmap à la place d'un titre de menu (@ au début).

La liaison avec les boutons de la souris suit la syntaxe suivante :

```
Buttons DefaultButtonBindings
{
        Bouton          contexte        f.menu menu_à_appeler
}

Buttons DefaultButtonBindings
{
        <Btn1Down>              root            f.menu nom_menu_1
        <Btn2Down>              root            f.menu nom_menu_2
        <Btn3Down>              root            f.menu nom_menu_3
}
```

9.4.2 Paramétrage des touches du clavier :

```
Key     DefaultKeyBindings
{
        Action  Contexte        fonction
}
Ex. : <Key>F1   root|icon|window        f.quit_mwm
```

10 Comparatif des principaux systèmes (à remettre à jour, mais j'attends l'évaluation de "7")

La stabilité : Unix (et ses avatars)
Les systèmes les plus stables sont FreeBSD, NetBSD et OpenBSD suivis de près par Linux. Windows NT/2000/XP/Vista est également relativement stable en poste client mais présente encore quelques lacunes dans ses versions serveur (XP et Vista étant quand même bien meilleurs que les moutures précédentes). Globalement Windows reste le système le plus instable. MacOS (et le regretté BeOS) ont une stabilité correcte pour une utilisation personnelle.

Le domaine ludique : Windows
Pour un ordinateur familial, le monde du jeu est certainement le domaine qui demande le plus de puissance. Les passionnés n'hésiteront pas à changer leurs cartes graphiques plusieurs fois par an et à mettre leur configuration à jour avec les dernières nouveautés technologiques. Window9x est alors le système qui convient le mieux. Les constructeurs de matériel informatique sortent toujours les pilotes de leur dernier matériel d'abord sur Windows afin de toucher la majorité des utilisateurs. Les autres systèmes d'exploitation doivent alors attendre que le constucteur se décide à développer le pilote pour leur système.

Réseau : Unix (et ses avatars)
Les systèmes UNIX (xBSD, Linux, Solaris, etc...) sont les systèmes les plus aptes à une utilisation en Réseau. Pour des petits réseaux locaux, Windows ou Mac OS est cependant très satisfaisant grace à sa maintenance facile. Pour des gros réseaux et des systèmes ouverts à Internet, mieux vaut prendre un système UNIX qui est plus stable, plus performant et plus sécurisé.

La bureautique : Windows ou MacOS
Windows est le système le plus apte à une utilisation bureautique. La suite Ms-Office est la suite la plus utilisée du monde. Cependant le monde UNIX devient compétitif avec plusieurs suites très performantes et compatibles avec les formats Microsoft Office (**OpenOffice**, WordPerfect, Applixware...). Linux est donc le principal concurrent de Windows dans ce domaine grâce à sa simplicité d'utilisation (xBSD est réservé aux utilisateurs expérimentés).

Rapidité : Unix (plus particulièrement xBSD) (et feu BeOS)
FreeBSD, OpenBSD et NetBSD sont les systèmes les plus rapides et se contentent d'une très petite configuration. Linux est un peu moins rapide mais son utilisation est plus simple. Linux se contente également d'une petite configuration sauf si vous adoptez l'interface graphique. Cependant même sous X, le système ne sera pas trop exigeant et se comportera admirablement avec un 486 et 16Mo de Ram (voire 8Mo) (attention, je ne parle pas ici d'Ubuntu). Avec au moins un Pentium, BeOS était néanmoins beaucoup plus rapide qu'un Unix sous XWindow. Windows est également assez rapide à condition d'avoir une bonne configuration avec beaucoup de mémoire vive.

Graphisme : MacOS et dans une moindre mesure Linux et Windows
Windows semble être un bon système pour le graphisme mais le prix des logiciels et quelques limitations le rendent inadapté à cet tâche. A partir de Windows NT, Windows est un peu plus performant grâce à son support multiprocesseur. Linux possède des outils très performants et gratuits et excelle en multithreading. Donc, il semble que Linux prenne içi le dessus sur Windows. Si vous avez les moyens de vous acheter des logiciels à plusieurs milliers de francs, Windows reste un bon choix. Mais pourquoi payer lorsqu'on peut avoir des outils performants gratuitement ? MacOS fut et reste l'OS préféré des graphistes grâce à sa commodité d'utilisation et aux nombres d'outils disponibles.

Logithèque : Windows
Windows est sans conteste l'OS avec la plus grande logithèque. Cependant, Linux a maintenant une logithèque très confortable et recouvrant tous les domaines (ou presque). De plus, la plupart des logiciels linux sont gratuits et très performants. MacOS a également une logithèque polyvalente mais bien moins grande que celle de Windows.

Programmation : Unix (et ses avatars)
Les systèmes UNIX sont les systèmes les plus adaptés aux programmeurs avec de nombreux outils gratuits et la disponibilité des sources de nombreux programmes. Windows n'est cependant pas nécessairement un mauvais choix. En effet, la programmation sous Windows est très facilitée grâce à des outils performants et simples d'utilisation (Visual Studio, Builder C++, Delphi...). La programmation est même ouverte aux utilisateurs peu expérimentés avec des outils comme Visual Studio qui permet de programmer d'une manière intuitive et visuelle avec peu de connaissances en programmation. Ces considérations n'ont cependant plus lieu d'être car les systèmes Unix ont maintenant des outils comparables à Visual C++ et sous licenseGPL. C'est par exemple le cas de KDevelop, Glade...

11 Bibliographie :

– La programmation sous Unix, Jean-Marie Rifflet, Ediscience
– La communication sous Unix, Jean-Marie Rifflet, troisième édition, McGraw-Hill
– **Unix et les systèmes d'exploitation, Michel Divay, Dunod**

– **Le système Linux, Matt Welsh, troisième édition, O'Reilly**
– Linux in a nutshell, Jessica P. Hekman, édition française, O'Reilly

– Introduction à Python, Mark Lutz & David Ascher, O'Reilly
– **Programming Python, 2nd Edition, Mark Lutz, O'Reilly**

– **Programmation en PERL, troisième édition en français, Larry Wall, O'Reilly**
– Le langage C, B.W. Kernighan, D.M. Ritchie, deuxième édition, Masson

– **Petit guide à l'usage du développeur agile, Tarek Zadié, Dunod**

Index