Support du cours

Système d'exploitation UNIX/Linux/Ubuntu

-

Première partie : commandes et outils

-

(la seconde partie traite du scripting en Bash, Perl et Python)

Pierre Pompidor

-

LIRMM, 161 rue Ada,
34 392 Montpellier Cedex 5
téléphone : 04 67 41 85 36
(ce numéro de téléphone apparaît ici à titre décoratif)
adresse électronique : **pompidor@lirmm.fr**
URL : http://www.lirmm.fr/~pompidor (le caractère mystérieux avant pompidor est un tilda)
mais les supports de cours sont sur le Moodle de la FDS de l'UM !

Ce polycopié accompagne le cours **Système d'exploitation/Unix/Linux** dispensé en Licence et Master d'Informatique. Un second polycopié vous sera donné sur la partie Scripting.

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, de démons, de pilotes...) qui hantent votre machine), ces programmes permettant d'utiliser un ordinateur pour une myriade de tâches (pour développer de nouveaux programmes, faire de la bureautique, utiliser des logiciels, 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 (et son avatar Linux), Windows, MacOs ...).

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 ordinateurs personnels), 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. Par ailleurs, ce système est gratuit ;).

A savoir pourquoi apprendre des commandes et non pas s'entraîner à une utilisation virtuose 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, que connaître quelques centaines de commandes ;) permettra ensuite de les insérer dans des scripts (programmes) systèmes, et qu'enfin de cliquer un peu partout fait perdre beaucoup trop de temps...

Voici quelques remarques sur les chapitres du polycopié (première et seconde parties) :

- Chapitre 1 : **Introduction**
  Présentation générale d'*Unix*, de *Linux* et d'*Ubuntu*.

- 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 les macro-commandes permettant de se connecter sur une machine distante (principalement avec *ssh*, *ftp (ou ncftp, sftp...)* et *cURL*, les outils de gestion de versions (en fait GIT) et les éditeurs de textes (pour la programmation et pour la bureautique) les plus importants intégrés à Linux.

- Chapitre 6 : **Interpréteurs 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.

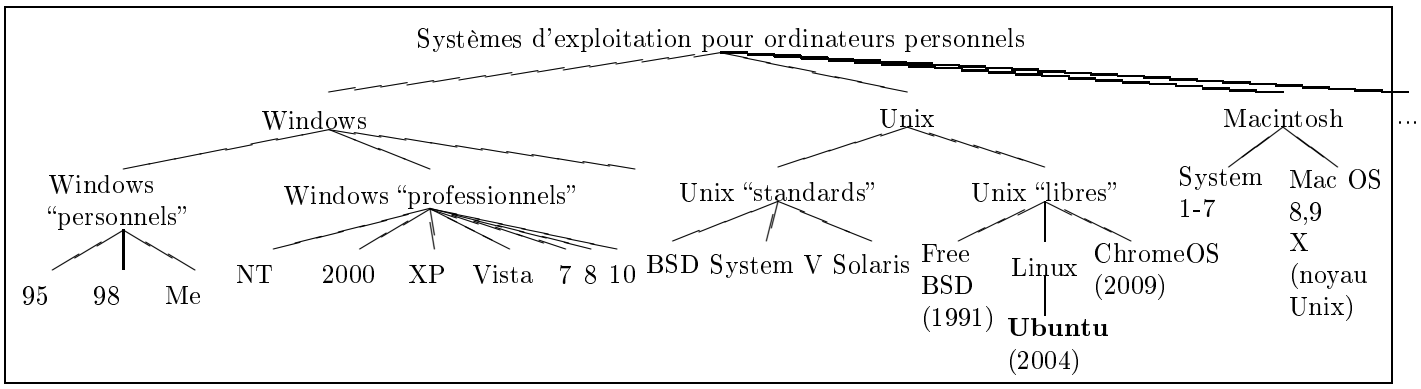
Les parties importantes sont encadrées.

Les touches du clavier sur lesquelles appuyer sont entourées par des "<" et ">"; exemple : $<Tab>$.

Contents

1 Introduction :

1.1 Panorama des systèmes d'exploitation pour ordinateurs personnels :



Parts de marché estimées pour (Wikipedia : Usage share of operating systems) :

- les postes clients : Windows → 82%, MacOS → 13%, **Linux → à peu près 2%, ...**
- les postes serveurs : **Linux → 37%**, Windows → 33%, Unix (hors Linux) → 30%
- La consécration, les super-calculateurs : **Linux → 97%** et avec les autres systèmes Unix 99%
- mais attention : **22%** des informaticiens développent sous Linux !

Par ailleurs, il ne faut pas oublier qu'avec **Android**, c'est Linux qui équipe la majorité des smartphones !

1.2 Bref historique d'Unix et de Linux et vue générale :

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 **Unity**, **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, la plus populaire étant **Ubuntu**, mais d'autres sont à citer :

- La distribution Linux de Debian ;
- Red Hat Linux, Fedora Core, CentOS.

Comme il le sera expliqué, une "partie" d'Ubuntu est intégré à Windows 10 (*WSL*).

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.

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 subtilement 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 (comme les autres distributions) exécutable via une machien virtuelle (ceci est expliqué ci-après) ;
- 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 Pratique de Linux/Ubuntu sur votre ordinateur personnel où Windows est installé

Pour utiliser Linux/Ubuntu sur votre ordinateur personnel, tout en gardant Windows pré-installé, vous avez trois possibilités :

- mettre en œuvre un sous-sytème de Windows (10 exclusivement) qui permet d'exécuter certains binaires de la distribution Ubuntu ;
- l'utiliser dans une machine virtuelle (sans qu'il soit réellement installé sur votre machine) ;
- l'installer à côté de Windows sur votre disque dur (installation en *dual boot*).

Je vous recommande de prévilégier la première ou la seconde solution qui nécessitent beaucoup moins d'opérations techniques qu'une installation en *dual boot*.

2.1.1 Windows Subsystem for Linux (WSL)

Sur les versions 64 bits de Windows 10, un sous-sytème est installable qui permet d'exécuter certains binaires de la distribution Ubuntu de Linux (depuis sa version 16.04).

C'est assez intrigant et voici les deux liens que je vous recommande :
https://doc.ubuntu-fr.org/wsl
http://www.numerama.com/tech/158150-le-shell-bash-sous-windows-10-ce-quil-faut-savoir.html

Une fois l'application *bash* installée sous Windows :

- sous bash les systèmes de fichiers Windows sont accessibles à partir de **/mnt** et les comptes utilisateurs à partir de **/mnt/c/Users** ;
- sous Windows les systèmes de fichiers Linux sont accessibles à partir de `C:\Users\<NomDeLUtilisateur>\AppData\Local\lxss`.

Attention, il est possible que l'antivirus/pare-feu de Windows s'oppose aux installations de paquets, dans ce cas il faut le désactiver temporairement.

2.1.2 Virtualisation (*) de Ubuntu sous Windows grâce à VirtualBox

VirtualBox, développé maintenant par **Oracle**, permet de virtualiser des systèmes d'exploitation ***invités*** sur un système d'exploitation ***hôte***. A peu près toutes les combinaisons sont possibles (par exemple héberger Linux sous du Windows ou le contraire).

Schématiquement, pour héberger Ubuntu sous Windows, vous devez :

- récupérer l'image ISO du CD d'installation d'Ubuntu :
  - se connecter à **http://www.ubuntu.com/download/desktop**
  - downloader la LTS (Long Term Support) **ubuntu-18.04-desktop-amd64.iso** *Bionic Beaver*

- installer VirtualBox :
  - se connecter à **http://www.virtualbox.org/**
  - downloader le **binaire** VirtualBox pour la version de Windows intallée

- lancer VirtualBox et créer une **machine virtuelle** → icône *Nouvelle* :
  - en la pré-configurant pour Ubuntu (pré-configuration un peu mystérieuse)
  - en lui associant :
    * un lecteur CD (a priori c'est fait automatiquement), mais qui s'en sert encore ?
    * un disque dur virtuel (redimmensionnable, 8G de taille max).

- installer le système invité (Ubuntu) sur le système hôte (Windows) : → icône *Démarrer* :
  - browser l'image ISO : → pour le faire apparaître dans la liste associée à "Choix du média d'installation" ;
  - puis rajouter les additions relatives au système Ubuntu
    (notamment pour pouvoir mieux gérer la fenêtre relative à ce système).

(*) Le mot *émulation* serait un peu impropre car il désigne la traduction des instructions d'un micro-processeur vers un autre, mais bon on peut l'employer...

2.1.3 Installation en dual-boot d'Ubuntu à côté de Windows (si vous aimez les mauvaises fréquentations)

Si vous avez un ordinateur assez récent ;), il est probable que le BIOS (un premier logiciel stocké sur une mémoire non volatile et qui fait le lien entre les micrologiciels (firmwares) installés sur les matériels (cartes, disques...) et le premier système d'exploitation, ait été remplacé par une interface UEFI (Unified Extensible Firmware Interface). Cette nouvelle interface complexifie passablement l'installation d'Ubuntu.
Si c'est le cas allez à cette adresse : **http://doc.ubuntu-fr.org/uefi**
Sinon lisez ce qui suit.

Vérification des partitions existantes sous Windows (étape 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 **Gparted-Live**.

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** ou **ext4**;

- **/home** : partition utilisateur (également associée à ext3/ext4) ;
- **swap** : partition d'échange pour gérer la mémoire virtuelle ;
- *vous pourriez également créer une partition supplémentaire d'échange avec Windows en FAT32.*

2.2 Prise en main d'Ubuntu

2.2.1 Connexion et opération système avec les droits de l'administrateur :

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]** (**sudo -s** pour créer une session) ;
- pour lancer une application interfacée par : **gksudo [application]** .

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

2.2.2 Découverte du bureau dans les versions classiques:

Ce paragraphe ne traite pas de l'interface utilisateur par défaut des versions récentes d'Ubuntu : ***Unity*** *qui doit être totalement intuitive !*

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), chargeur d'application : Clic-droit sur le bureau → Créer un lanceur...
Par exemple, lancez le navigateur **Chromium** ou **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. Essayez 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 ;

(Mémorisez les URLs qui vous intéressent comme **"signets"**/**"bookmarks"**)

2.2.3 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.

2.2.4 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
  (essayez la **flèche vers la droite** 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.

2.3 Installation de logiciels :

2.3.1 Installation de paquets (ou paquetages), généralement des applications déjà compilées :

Pour installer de nouvelles applications, de nouvelles bibliothèques, plusieurs possibilités vous sont offertes (dans ce qui suit <*paquet*> désigne un nom de paquet...) :

- par l'application **Synaptic** si elle existe ;
- par l'outil **apt** (*Advanced Packaging Tool*), et plus précisément la commande **sudo apt install** <paquet> qui installe un paquet en essayant de résoudre les dépendances (càd en important également les bibliothèques nécessaires à l'installation)
- par la commande **sudo dpkg -i** <paquet.deb> :
  pour des paquets de bas niveau (mais sans que les dépendances ne soient résolues).

La commande *apt* rend obsolète la commande *apt-get*.

Le fichier de configuration **/etc/apt/sources.list** répertorie l'ensemble des **serveurs = dépôts** connus contenant des paquets. A partir de ces dépôts, *apt* connaît la liste des paquets disponibles : cette liste peut être mise à jour par la commande **sudo apt update**.
La commande **sudo apt upgrade** met à jour tous les paquets.
La commande **sudo do-release-upgrade** permet d'installer une nouvelle version du système.

2.3.2 Installation de sources archivées (codes à compiler) :

Quelquefois l'application qui vous intéresse n'est pas livrée dans un paquet mais dans une archive (généralement au format **.tar**) : dans ce cas cette archive contient les codes sources de l'application qui doit être recompilée.
*Attention : la commande **tar** sera décrite un peu plus loin.*

- Téléchargement d'une archive **.tar**, **.tar.gz** ou **.tgz**, l'archive est recopiée par défaut dans le répertoire **Téléchargements / Downloads**
- **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

2.3.3 Installation et configuration d'Apache2 sur votre machine :

Apache2 est le **serveur HTTP** qui permet de servir des ressources web (pages HTML, script PHP...).
*(Ce n'est pas le seul serveur HTTP qui peut être utilisé sous Linux, **nginx** ou la plateforme JavaScript **Node.js** peuvent également être utilisés.)*

- Installer Apache → à télécharger via le gestionnaire de paquets (*sudo apt install apache2*)
- 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 en gérant le service qui le contrôle :
  - Pour l'arrêter : **sudo service apache2 stop** (anciennes versions d'Ubuntu : */etc/init.d/apache2 -k stop*)
  - Pour le lancer : **sudo service apache2 start** (anciennes versions d'Ubuntu : */etc/init.d/apache2 -k start*)
  - Pour le relancer : **sudo service apache2 restart** (anciennes versions d'Ubuntu : */etc/init.d/apache2 -k restart*)
- Déposez les ressources web communes dans /**var**/**www** (pour les script CGI voir ci-après)
- 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
    la ligne : `ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/` peut localiser par défaut 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>
```

2.4 Gestion des utilisateurs et des groupes :

- Créez un nouvel utilisateur avec le gestionnaire des utilisateurs et groupes
- Accordez-lui le privilège 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 différents environnements graphiques (nommés **bureaux**) comme **Unity**, **GNOME** ou **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** ou l'effrayant éditeur **emacs**).
- pour explorer les répertoires et ouvrir les fichiers, vous pouvez utiliser l'explorateur de répertoires (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 :

Lors de l'ouverture d'un compte permettant l'utilisation des salles informatiques à la FDS, sont attribués :

- 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,

A la Faculté des Sciences de l'UM, l'ouverture d'un compte est subordonnée à l'inscription administrative :
suivre ce lien → https://sif.info-ufr.univ-montp2.fr/ puis cliquer sur "procédure informations de mon compte".

3.1.2 Connexion :

La connexion s'effectue soit sur une machine isolée ou sur un réseau de machines (anciennement gérées par **NIS** : Network Information Service) ou par un système d'annuaire de type **LDAP**.
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 :
  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 est occulté en mode "shadow" càd recopié dans un autre fichier.
- le fichier *passwd* géré par le serveur NIS dans le cas d'une connexion sur un réseau de machines ;
- via un annuaire LDAP.

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** (cela-dit c'est super mauvais signe). 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 MacOS), 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 Informations sur votre machine et la version du système Linux installé :

uname -a	informations sur la machine locale *nom_système nom_machine révision version Nom_modèle_machine*
lsb_release -a	affichage de la version de Linux
hostname	nom de la machine
lscpu	architecture de votre machine
lspci	périphériques connectés à la carte mère via des bus PCI
date	affichage de la date et de l'heure *(je sais que cette commande n'a rien à faire là)*

3.1.6 Informations sur les connexions réseaux :

Un ordinateur reçoit les messages qui proviennent du réseau via une ou plus plusieurs **cartes réseau**.
Chaque carte réseau possède une **adresse** MAC (*Media Access Control*) unique.

Le message réseau est accompagné d'un **port** (une sorte de numéro de boîte à lettres) qui indique quel est le processus qui doit le prendre en charge (un processus est un programme en cours d'exécution).

Un processus (gérant des messages réseau) et s'exécutant en permanence s'appelle un **service**.

ifconfig	cartes réseaux et adresses IP rattachées : *eth0* : première carte ethernet (connexion filaire) *wlan0* : première carte wifi (*Wlan = Wireless Local Area Network*)
service –status-all	état des services (à partir de Ubuntu 16.04)
netstat -antup	connexions réseaux

3.1.7 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	permet de connaître les utilisateurs des machines du réseau local *attention : cela ne fonctionne que si un serveur rwho est en activité sur chaque machien du réseau.*
ou **rusers**	(interrogation de chaque machine)
last	permet d'avoir l'historique des dernières connexions sur la machine

3.2 Le système de gestion des processus :

Un **processus** est :

- un programme en train d'être exécuté
  (en fait plus exactement, un programme qui a été chargé en mémoire centrale) ;
- 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.

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
top :	affichage dynamique (càd se raffraîchissant toutes les secondes) des processus cette commande est interfacée avec le gestionnaire de pages **more**
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).

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 peuvent être rédéfinis par l'administrateur, mais quelques conventions sont respectées :

/bin	commandes non internes du shell (de *bash*)
/dev	noms des fichiers spéciaux associés aux périphériques
/etc	fichiers de configurations, scripts de contrôles de certaines applications
/home	répertoires de travail des utilisateurs
/mnt	les répertoires utilisés pour monter temporairement un système de fichiers (clef USB...)
/tmp	fichiers temporaires
/usr	les applications
/usr/bin	applications installées avec Linux
/usr/local/bin	applications spécifiques
/var	données variables (et notamment sites web)

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
h : affichage human-friendly
t : tri chronologique

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* → renommage *mv nom_fichier répertoire* → déplacement *mv nom_fichier répertoirenouveau_nom_du_ficher* → déplacement et renommage
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 751 fichier* → droits : rwxr-x–x

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.
Exemples : `find . -name "*.mp4" | wc -l` : compte tous les fichiers mp4 à partir du répertoire courant
`find / -name "*.txt" | xargs grep ...` recherche d'un motif dans tous les ".txt" du système de fichiers
→ **xargs** commande permettant de récupérer les arguments passés par la commande précédente.

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 {} \;`
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 Autres commandes à fréquenter :

4.1 Relatives aux interpréteurs de commandes :

Commandes externes : l'exécution de cha-
cune 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.

4.2 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++

4.3 Relatives aux places mémoires occupées :

df -h	état des disques logiques (de la partition) notamment nb de blocs libres
du -h	espace alloué aux différents fichiers
free	état de la mémoire vive
pstat -s	statistiques sur la mémoire centrale (pas sous Linux)
quota	affichage de vos quotas et de la place mémoire que vous occupez

4.4 Relatives aux fichiers :

4.4.1 Sur leur recherche :

find : recherche récursive de fichiers dans une arborescence, par rapport à un nom :
find répertoire -name nom_du_fichier_recherché
exemple : *find . -name "*.txt"* recherche à partir du rép. courant des fichiers d'extension "txt"
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
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 commande {} \;
{} correspondant au nom du fichier

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 :

Pour ce qui suit, un fichier départements est pris comme exemple, chaque département étant décrit sur une ligne par des informations séparées par des ":" :
01:50:5756:Ain:Bourg-en-Bresse:Rhône/Alpes
...

[début de la définition d'un ensemble de caractères
] fin de la définition d'un ensemble de caractères
- définition d'intervalles
[abc] : soit a, soit b, soit c
[0-9] : un chiffre quelconque
. un caractère quelconque
***** indicateur d'itérations (de 0 à n fois le caractère précédent)
+ indicateur d'itérations (de 1 à n fois le caractère précédent)
. ou .+ : une chaîne de caractères quelconque*
? facultativité (de 0 à 1 fois le caractère précédent)
^ début de ligne en début d'expression ou complément assembliste après un [
[^0-9] : n'importe quel caractère sauf un chiffre
$ fin de ligne en fin d'expression
**** déspécialisation d'un caractère
| alternative entre deux chaînes `(Pierre|Paul)`

Exemples :
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 Bourgogne :
`grep -c -v  'Bourgogne$' départements`
(ici ' pour éviter que $ soit interprété par bash)

4.4.2 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

xxd -b : visualisation d'un fichier en binaire

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

4.4.3 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`

4.4.4 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 ...`

4.4.5 Sur des cryptages, compressions et 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 (obsolète) → suffixe *.Z* associé au fichier compressé
uncompress	décompression (obsolète)
gzip	compression → suffixe *.z* ou *.gz* associé au fichier compressé
gzip -d	décompression
gunzip ou **unzip**	idem

La compression de données est surtout associée à la création d'**archives** (une archive permet la création d'un seul fichier correspondant généralement au contenu d'une arborescence de répertoires).
Cette compression est par défaut effectuée par la commande **gzip** lors de l'utilisation de la commande d'archivage **tar** avec l'option **z** ou avec la commande **tgz**.

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*
	désarchivage :	*tar -xvf archive* (fichiers d'une archive) → option *z* pour décompresser une archive compressée avec *gzip* *tar -xzvf archive* → option *k* pour conserver des fichiers préexistants *tar -xkvf archive*
tgz		même commande que *tar* mais compresse le résultat (similaire à l'utilisation de l'option **z**)

4.4.6 Sur des impressions :

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

4.4.7 Montage/démontage de systèmes de fichiers :

Généralement, les périphériques (disques durs externes, clefs... sont automatiquement montés, càd intégrés au système de fichier, quand ils sont connectés par exemple aux slots USB, mais bon quelques fois il faut les monter à la main, ou par exemple les démonter pour les remonter en lecture seule par précaution.
Ce sont les commandes **mount** et **umount** (pour le démontage) qui permettent de faire cela.

mount:	montage d'une arborescence de fichiers	
	forme générale :	*mount device répertoire*
	montage d'une clef :	*mount /dev/sdb /media/login/nomDeLaClef*
	montage en lecture seule d'un disque externe:	*mount -r /dev/sdb /monDisque*

Sans paramètres, la commande **mount** liste tous les points de montage, par exemple :
`/dev/sda5 on / type ext4 (rw,errors=remount-ro)`
→ la cinquième partition (logique) du premier disque dur (sda5) est montée sur le répertoire /

Le fichier **/etc/fstab** donne également des informations sur les montages effectués.

5 Outils :

5.1 Outils de communication :

Les principaux outils de communication sécurisés (ceux à utiliser) sont les suivants :

- **ssh** : ouverture d'une session de travail
  `ssh login@adressseIP`

- **scp** : copie de fichiers, exemples d'utilisation :
  `scp fichier login@adressseIP:cheminAPartirDurepertoireDAccueil`
  `scp login@adressseIP:cheminAPartirDurepertoireDAccueil/fichier cheminSurLaMachineLocale`

- **sftp** : transfert de fichiers en **gestion interactive** `sftp login@adressseIP`
  ncftp et **ftp** qui est la commande générique sont expliquées plus en détail ci-après.

Pour éviter de s'authentifier systématiquement avec un mot de passe, il est possible de générer une paire de clefs publique et privée.
Par exemple pour créer cette paire de clef en utilisant le chiffrement RSA, utilisez la commande suivante :
ssh-keygen -t rsa (Il vous sera aussi demandé une passphrase (un mot de passe) pour sécuriser la clef privée).
Les clefs seront stockées dans le dossier **/.ssh** sous le nom **id_rsa** (la clef privée) et **id_ras.pub** (la clef publique). La clé publique doit alors être copiée sur le serveur distant dans le dossier **/.ssh/authorized_keys**.

ssh a rendu obsolète et dangereux (mais cela-dit quelquefois utile pour faire des test) **telnet**

ncftp :	Transfert de fichiers entre la machine locale et 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 ?)

cURL (abréviation de *client URL request library*)
est une commande qui permet de récupérer le contenu d'une ressource accessible sur le web.

Exemples :
curl www.lirmm.fr
curl www.lirmm.fr > pageAccueilLirmm

5.2 Outils de gestion de versions (en fait GIT et plus précisément via un dépôt GitLab) :

Deux principaux outils :

- **Subversion** (**SVN** en abrégé)
- et surtout **GIT**

permettent de gérer les différents fichiers des différentes versions d'une application informatique (ou plus globalement de tout corpus de textes).
Ces fichiers pourront/devront être répartis sur plusieurs machines (au moins celles des différents contributeurs à l'application et sans doute d'autres serveurs).

GIT est à privilégier sur SVN car ce dernier outil centralise l'intégralité des fichiers sur un serveur unique, ce qui rend l'application vulnérable en cas de défaillance de celui-ci.
Nous ne présenterons (brièvement) donc que le fonctionnement de GIT (attention cette présentation est vraiment minimaliste).

GIT permet de synchroniser des fichiers qui se trouvent ches vous dans un dépôt "local" vers un dépôt distant et réciproquement.
(A proprement parler il n'y a donc pas de serveur et de client.)

Un fichier géré dans un dépôt GIT passe par plusieurs états :

- de votre dépôt vers un dépôt distant : sélectionné (commande **add**) → déposé (commande **commit**) → copié sur un autre dépôt (commande **pull**
- du dépôt distant vers votre dépôt : copié (commande **push**)

Git permet aussi de créer des branches de développement (qui ne sont pas abordées ici) mais a priori vous n'en aurez pas tout de suite besoin avant de commencer un projet conséquent comprenant plusieurs options de développement !

Voici les commandes de base à utiliser pour :

- créer un **dépôt GitLab** (géré par la Faculté des Sciences) distant qui centralisera les fichiers de votre projet ;
- le cloner sur vos machines locales ;
- et ensuite synchroniser le dépôt Gitlab avec vos dépôts locaux.

(Je considère que vous êtes sous Linux mais cela fonctionne sous n'importe quelle plateforme).

5.2.1 Créer une clef de connexion sécurisée sur votre machine locale et installer git :

ssh-keygen (sauver la clef dans `/home/<login>/.ssh/id_rsa`
→ voir la page d'explication : `https://gitlab.info-ufr.univ-montp2.fr/help/ssh/README`
→ deux clefs sont créés :
id_rsa.pub : la clef publique qui pourra être publiée
id_rsa : la clef privée
installer git (sous Linux/Ubuntu : **sudo apt-get install git**)

5.2.2 Création du dépôt Gitlab :

Pour utiliser ce service, il suffit de se rendre à l'adresse suivante puis de saisir son adresse institutionnelle UM (prenom.nom@umontpellier.fr) et son mot de passe ENT :
https://gitlab.info-ufr.univ-montp2.fr
→ cliquer sur le lien créer une clef ssh et copier la clef publique dans le formulaire

5.2.3 Création du dépôt local et synchronisation avec le dépôt distant :

SUR VOTRE MACHINE et dans le répertoire de travail :
cloner votre dépôt : commande à recopier sur le site Gitlab du type :
`git clone git@gitlab.info-ufr.univ-montp2.fr:<identifiant de votre projet>`

Puis vous pourrez utiliser les commandes suivantes :
git add <fichier à rajouter dans le dépôt> → par exemple : git add *

git commit -m "commentaire" → mise à jour du dépôt local
git push → mettre à jour le dépôt distant
get pull → mettre à jour le dépôt local

Il est essentiel avant de travailler sur vos codes de réaliser un "git pull" pour récupérer les nouvelles versions de codes.
Nous n'épiloguerons pas sur le cas où lors d'un "git push", GIT vous signale qu'il y a un conflit avec une autre version de code déposé entretemps...

Un fichier **.gitignore** permet de recenser les fichiers à ne pas prendre en compte (par exemple quand vous faites un git add *)
git status permet d'avoir à un moment donné l'état de votre dépôt.

Remarque, pour créer ex nihilo un dépôt (sans le cloner), vous devez utiliser la commande **git init** : **git init** → création d'un sous-répertoire **.git**
git config –global user.name "..."
git config –global user.email "...@umontpellier.fr"

5.3 Editeurs de texte :

Il ne faut pas confondre les éditeurs de texte pour la programmation qui ne rajoutent aucune information aux données enregistrées, aux éditeurs de texte de bureautique.

5.3.1 Liste des principaux éditeurs disponibles :

Visual Studio Code (qui est libre) : à lancer avec la commande `code`

ed	le plus ancien, cité ici pour des raisons sentimentales
vi	éditeur pleine page
nano	idem, plus moderne
atom	éditeur graphique
Sublime Text	la star du moment
emacs	ancien, indémodable, complexe mais tellement intéressant (macros ...)
LibreOffice	suite bureautique avec *lowriter* comme éditeur) **A ne jamais utiliser pour programmer !**

5.3.2 Aperçu de vi :

vi est un éditeur *plein texte*, c'est à dire qui se passe d'interface graphique. Quelques fois il peut vous sauver la vie.

Cet éditeur possède deux modes :

- le **mode commande** (positionné au lancement) ;
- le **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
:w nom_fichier

Quitter :

:q

5.3.3 Aperçu d'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 Introduction au scripting système :

A partir d'un terminal Linux (un *shell*), il est possible de créer et d'exécuter des **programmes système**, c'est à dire des programmes qui utilisent des commandes ou des informations du système d'exploitation. Ces programmes qui sont aussi appelés des **scripts** car ils sont interprétés, peuvent permettre :

- de mémoriser des lignes de commandes compliquées et/ou paramétrables (par exemple je veux rechercher à partir d'un répertoire dont le nom est passé en paramètre au script tous les fichiers qui ont une certaine extension (autre paramètre du script) et les supprimer) ;
- de créer de nouvelles commandes (par exemple je veux créer une commande nommée *bashConfig* qui m'indique suivant différentes options comment est configuré bash) ;
- de réaliser des sauvegardes ;
- d'effectuer des analyses particulières de l'état du système, du comportement des utilisateurs ;
- ...

Pour ce faire, plusieurs langages de haut niveau et interprétés sont à votre disposition :

- le langage de programmation associé à **bash**
- le langage de programmation associé à **(t)csh**
- le langage **Python**
- le langage **Perl**
- le langage **Ruby**

Les langages bash et (t)csh ont le gros avantage d'interfacer directement les commandes et les gros inconvenients de ne pas être modulaires et d'avoir des syntaxes vieillottes et piégeuses (surtout bash) : de ces deux langages, seul bash sera abordé en cours.
Des langages de plus haut niveau et universels (car ils peuvent être utilisés pour d'autres buts que la programmation système) Python, Perl et Ruby, seul Python sera abordé en cours (et sans utiliser le paradigme de la programmation par objets).

Remarques :
Il est bien sûr aussi possible d'utiliser le langage C pour créer des programmes système mais celui-ci non interprété et de bas niveau est plus difficile à maîtriser et la création des programmes en sera beaucoup plus longue. Cela-dit, quand le programme doit interfacer des appels systèmes (les fonctions proposées par le noyau du système), le langage C est incontournable (par exemple vous avez en tête de réécrire bash...).
Sous Windows, outre Python, Perl et Ruby, le langage phare dans la programmation système est **PowerShell**.

7 Scripts exécutés par l'interpréteur de commandes ((t)csh ou bash) :

bash et (t)csh sont des **interpréteurs de commandes** : leur rôle est d'interpréter les lignes de commandes (une ligne de commande pouvant contenir plusieurs commandes séparées par exemple des points-virgules ou des pipes) saisies par l'utilisateur dans un terminal Linux.
Il sont aussi capables d'interpréter des instructions (manipulation de variables, tests, boucles...) et possèdent donc **leurs propres langages de programmation** (mais beaucoup moins puissants que ne peuvent l'être les langages de programmation **Python** ou **Perl**).

Un programme qu'il soit écrit pour (t)csh ou bash peut être exécuté de différentes manières :

- **interactivement** : les instructions du programme sont directement saisies dans le terminal
  → un nouveau prompt apparaît si le bloc d'instructions n'est pas fini (par exemple les instructions comprises dans une boucle).

- sous la forme d'**un programme exécutable** (généralement suffixé par **.sh**) :
  par exemple, le script nommé *bonjour.sh* contenant la ligne

  ```
  echo bonjour
  ```

  sera exécuté par l'une des commandes suivantes :

- `./bonjour.sh` si le script est exécutable (le point fait référence au répertoire courant) ;
- `bonjour.sh` si le script est exécutable et si la variable d'environnement **PATH** contient le nom du répertoire où le script se trouve ;
- `source bonjour.sh` ou `. bonjour.sh` si le script n'est pas forcément exécutable ;
- `bash bonjour.sh` pour (bizarrement) exécuter ce script par un autre processus bash que celui qui gère le terminal.

Un programme interprétable par l'interpréteur de commandes bash ou (t)csh est appelé un **script shell**, Par défaut il est exécuté par l'interpréteur de commandes actif (celui dont le nom apparaît dans la variable d'environnement SHELL).

Si vous voulez explicitement préciser dans un script shell, quel interpréteur de commandes doit l'exécuter, vous pouvez faire apparaître un shebang ;) sur la première ligne du script :

- `#!/bin/csh` pour **csh** (ou tcsh)
- `#!/bin/bash` pour **bash**

7.1 La configuration de bash ou de (t)csh :

bash ou (t)csh sont configurables à différents niveaux, en pratique seule la configuration au niveau de l'utilisateur nous intéressera.

Voici les principaux fichiers de configuration exécutés :

- Sous **csh** ou **tcsh** :
  - */etc/csh.cshrc*
  - */etc/csh.login*
  - **.cshrc** (dans votre répertoire d'accueil) → ce fichier va contenir vos configurations
- Sous **bash** :
  - */etc/profile*
  - *.bash_profile* (dans votre répertoire d'accueil)
  - **.bashrc** (dans votre répertoire d'accueil) → ce fichier va contenir vos configurations

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 :
  par exemple, voici comment la compléter avec le répertoire courant

  Sous csh/tcsh : *setenv PATH "${PATH}:."*
  Sous bash : *export PATH=$PATH:.*

- de définir des alias qui sont des synomymes à une commande :

  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 :
  exemple : `alias chm='function \_chm () \{ chmod \$1 \$2; \}; \_chm'`

- changer la valeur du prompt :
  - Sous csh/tcsh, par exemple avec **set prompt='%~ $ '**
    - %~ nom du répertoire courant
    - %t : heure

– Sous bash, il faut insérer des codes spéciaux dans la valeur de la variable **PS1** :
  * \d : date
  * \h : nom de la machine
  * \W : nom du répertoire courant
  * \w : chemin du répertoire courant
  * \u : login de l'utilisateur

  Exemple avec *PS1='\W \d $ '*

7.2 Variables et paramètres :

Les variables ne sont pas typées.
L'accès à la valeur d'une variable est signifié en faisant précéder le nom de la variable par $.

7.2.1 Les variables système :

Des variables prédéfinies permmettent d'accéder (et de modifier) des informations relatives au système d'exploitation :

- les **variables système d'environnement** dont les valeurs seront accessibles par les programmes lancés sous bash ou tcsh ;
- les variables système (simples) dont les valeurs n'ont une importance que pour l'interpréteur de commandes.

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
LOGIN :	login
PATH :	liste des répertoires où le shell cherche les exécutables à exécuter
PWD :	répertoire courant
SHELL :	shell (par exemple /bin/bash)
TERM :	type de terminal (par exemple xterm)

La principale variable système "simple" est celle qui donne au valeur au prompt dans le terminal :
PS1 sous *bash*) et **prompt** (sous *(t)csh*).

7.2.2 Déclaration / initialisation des variables :

Avec bash :

```
nom_variable=valeur
```

Exemple :

```
prenom=Pierre
```

(Attention : n'insérez pas d'espace avant ou après le signe =)

Une fois définie, une variable doit être préfixée par **$** pour être utilisée !

```
echo $prenom
```

Une variable peut être instanciée par le résultat d'une commande :

```
variable=‘commande‘
variable=$(commande)
```

Exemple :

```
ll=‘ls -l‘
```

Dans le cas de l'instanciation d'une variable d'environnement, il faut utiliser la commande **export**.
Exemple (ici pour rajouter le répertoire courant dans le PATH) :

```
export PATH=$PATH:.
```

Enfin pour récupérer une valeur au clavier, il faut utiliser la commande **read** :

```
read entree
echo $entree
```

Avec (t)csh :

Déclaration / initialisation d'une variable :
`set nom_variable = valeur`
Une variable peut être instanciée par le résultat d'une commande : `set variable = ‘commande‘`
Dans le cas de l'instanciation d'une variable d'environnement, il faut utiliser la commande **setenv**.

7.2.3 Calcul arithmétique :

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

- avec la syntaxe `$((<opération>))` : `i=$((3+4)); echo $i`
- avec la commande interne `let` : `let i=3+4; echo $i`

7.2.4 Les paramètres de la ligne de commande :

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

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

7.3 Structures de contrôles des scripts :

7.3.1 Structure conditionnelle avec if :

Sous *bash* et comme dans la quasi totalité des langages de programmation, le mot réservé **if** introduit une expression conditionnelle dans une structure de programmation qui permet de conditionner l'exécution d'un bloc d'instructions.
Mais à la différence des autres langages de programmation, l'expression conditionnelle peut être encadrée par différents délimitateurs (crochets, doubles crochets, parenthèses) ou même ne pas en utiliser : cette diversité est affreuse.
Vraiment.

Voici en apéritif, quelques exemples d'expressions conditionnelles :

```
if ls -l | grep -qc Cours -> l'expression conditionnelle est une ligne de commande :
                             pas de délimitateurs (l'option q de grep l'empêche d'afficher son résultat)
if test $i -gt 0          -> comparaison arithmétique avec la commande test et les opérateurs classiques
                             ici -gt teste que le premier opérande est plus grand que le second
if [ $i -gt 0 ]           -> les crochets remplacent la commande test
                             il faut mettre une espace après le premier crochet et avant le dernier
if [[ $i > 0 ]]           -> Les doubles crochets permettent d'utiliser tous les opérateurs arithmétiques
                             mais attention, la comparaison est lexicographique (voir exemple suivant)
if [[ $a > $b ]]          -> si $a vaut 20 et $b vaut 100, ce test renvoie vrai car 2 est supérieur à 1 !
if (( $a > $b ))          -> avec les doubles parenthèses, la comparaison arithmétique fonctionne !
```

Revenons au schéma d'utilisation de la structure conditionelle avec if (et ici en encadrant l'expression conditionnelle avec des doubles crochets) :

```
if (( expression conditionnelle ))
then
        bloc d'instructions
else                              (facultatif)
        bloc d'instructions
fi
```

Attention si vous ne passez pas à la ligne avant *then*, *else* ou *fi*, vous devez rajouter un point-virgule pour le signaler à l'interpréteur bash (cela aussi c'est affreux) !

```
if (( expression conditionnelle )); then bloc d'instructions; fi
```

Les doubles crochets autour de l'expression conditionnelle permettent d'utiliser les opérateurs de comparaisons avancés.

Il est donc aussi possible :

- d'utiliser la commande *test* ou des simples crochets pour :
  - des tests via des opérateurs unaires, exemple : `if [ -f $fichier ]`
  - des tests via des opérateurs binaires mais "vintage" (*-eq*, *-ne*, *-gt*, *-ge*, *-lt*, *-le*)
    exemple : `if [ $i -eq 1 ]`
- d'utiliser des doubles parenthèses pour des comparaisons arithmétiques
- de ne pas encadrer l'expression conditionnelle dans le cas de l'évaluation du résultat d'une commande directement évaluée dans celle-ci :
  `if <commande>`

Des exemples de diverses structures conditionnelles sont donnés dans le paragraphe *Exemples de scripts bash*).

Sous *(t)csh* les expressions conditionnelles sont sagement circonscrites par des parenthèses :

```
if (expression conditionnelle) then
        bloc d'instructions
else                            (facultatif)
        bloc d'instructions
endif
```

7.3.2 case et switch :

Sous *bash* :

```
case expression in
    pattern1 )
         bloc d'instructions ;;
    pattern2 )
         bloc d'instructions ;;
    ...
esac
```

Sous *(t)csh* :

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

7.3.3 Tant que (while) :

Sous *bash* (avec l'expression conditionnelle circonscrite avec des doubles crochets) :

```
while [[ expression conditionnelle ]]
do
        bloc d'instructions
done
```

Sous *(t)csh* :

```
while (expression conditionnelle)
        bloc d'instructions
end
```

7.3.4 Itération sur les valeurs d'une liste (foreach, for)

Sous *bash* :

```
for variable in liste
do
        bloc d'instructions
done
```

La liste peut être :

- une suite de valeur → `for i in 1 2 3; do echo $i; done`

- **produite par l'exécution d'une commande** → `for i in $(cat fichier); do echo $i; done`

Sous *(t)csh* :

```
foreach variable (liste de valeurs)
        bloc d'instructions
end
```

7.4 Opérateurs :

Les opérateurs du C peuvent être utilisés à la fois pour bash et (t)csh (mais attention bash possède aussi d'autres "vieux" opérateurs) :

- ==, !=, <, <=, >, >=
- !
- +, -, *, /
- ++, –

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
...	...

Bash permet aussi d'utiliser des opérateurs "old-school" du type (ici pour un test d'égalité) `if [ $a -eq $b ]`.

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**
$? pour bash et $status pour (t)csh :	valeur retournée par le dernier processus

7.5 Exemples de scripts bash :

7.5.1 Lister tous les répertoires du répertoire courant :

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

7.5.2 Afficher les fichiers réguliers qui contiennent une des chaînes passées en paramètres :

Première solution en utilisant une variable intermédiaire et via un comptage :

```
for f in *
do
  for p in $*
  do
    if [ -f "$f" ]
    then
       r=‘grep -c $p "$f" 2> /dev/null‘
       if (( $r > 0 ))
       then
          echo "$f contient $p $r fois"
       fi
    fi
  done
done
```

Vous remarquerez l'utilisation des doubles quotes autour de $f pour se prémunir des noms de fichiers avec espaces.

Seconde solution en évaluant directement grep dans l'expression conditionnelle :

```
for f in *
do
  for p in $*
  do
       if grep -q $p "$f" 2> /dev/null
       then
          echo "$f contient $p"
       fi
  done
done
```

Vous remarquerez l'utilisation de l'option q (*quiet*) de grep qui permet de lui faire renvoyer soit faux ou vrai.

7.5.3 Affichage numéroté des lignes d'un fichier dont le nom est passé en paramètre :

```
IFS=$’\n’
for ligne in $(cat $1)
do
    i=$((i+1))
    echo $i $ligne
done
```

Vous remarquerez avec gourmandise :

- la modification du "Internal Field Separator" pour éviter que echo ne réalise des passages à la ligne sur les espaces
- l'incrémentation de la variable i via l'évaluation arithmétique opérée par les doubles parenthèses

Bash n'est-il pas un langage de programmation merveilleux ?

Webographie / Bibliographie :

J'ai décidé de ne plus présenter que des ressources accessibles sur le web (hormis "le petit guide à l'usage du développeur agile" et le livre sur les expressions régulières (attention, celui-ci n'est à mettre dans n'importe quelles mains).

- **Pratique de Unix/Linux (commandes) :**
  - **http://juliend.github.io/linux-cheatsheet/** → c'est très raffraîchissant
  - **http://doc.ubuntu-fr.org/tutoriel/console_commandes_de_base**

- **Installation de Linux/Ubuntu :**
  - **http://doc.ubuntu-fr.org/**
  - **http://linux.developpez.com/tutoriels/debuter-installation/guide-linux-distribution/**

- **Programmation en Python : http://inforef.be/swi/download/apprendre_python3_5.pdf**

- **Programmation en Perl : http://lhullier.developpez.com/tutoriels/perl/intro/**

- **Virtuosité en expressions régulières :**
  Mastering Regular Expressions 3e (en anglais) - Jeffrey E F Friedl

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

Par ailleurs, voici un site de mémos fantastique sur à peu près tout (enfin en informatique) :
http://www.cheat-sheets.org/

Index