Introduction à l'informatique

Cours complet

G. Santini, J.-C. Dubacq

IUT de Villetaneuse

S1 2016

Organisation du module

Remerciements

- Les cours et exercices de ce module sont directement inspirés des documents de **M. Bosc**, **J.-C. Dubacq** et **G. Santini**.
- D'autres intervenants ont participé à l'élaboration des supports.

Les enseignements

- 12 sessions de 4h et du travail personnel …
- 6 sessions pour la présentation générale du système d'exploitation Linux,
- 6 sessions pour la théorie de base du codage informatique

Votre présence est obligatoire

- Contrôle des présences.
- Rapport des absences.

L'évaluation

- Une composition après la sixième session (sur papier ou sur ordinateur).
- Une composition à la fin du module (sur papier ou sur ordinateur).

Généralités

Qu'est-ce qu'un ordinateur ?
Les composants principaux et les principes de fonctionnement d'un ordinateur

Définition

Définition (Ordinateur)

Machine électronique programmable capable de réaliser des calculs logiques sur des nombres binaires.

C'est une machine *Hardware*

Le fonctionnement d'un ordinateur est basé sur une architecture matérielle (processeur, support de stockage, interfaces utilisateurs, connexion, …) dont le fonctionnement est soumis aux lois de la physique.

C'est une machine programmable *Software*

Cette machine est capable de remplir des tâches différentes selon les instructions qui lui sont adressées. Ces instructions, rédigées sous forme de programmes par les informaticiens, sont traitées en fin de course par le matériel de l'ordinateur.

Interaction Hardware/Software

La plupart du temps, l'informaticien n'a pas a interagir directement avec le matériel. Pour traiter avec les composants, tous les ordinateurs disposent d'une couche logicielle appelée *système d'exploitation*. Cette couche est en charge de faire la passerelle entre l'informaticien, ses outils, les programmes qu'il développe et, les composants et leur fonctionnement.

Les interfaces

La forme classique

- Un ordinateur est classiquement composé d'une unité centrale et de périphériques matériels (écran, clavier, souris, disques durs, imprimantes/scaner, …).
- Les interfaces permettent l'interaction avec l'environnement (utilisateurs ou autres).

Des formes très variées

- Les ordinateurs modernes sont multiformes,
- Ils remplissent des tâches très variées.

Points communs et différences

Matériel commun

- Des capacités de calcul : CPU et/ou GPU
- De la mémoire : RAM, Disque dur, …

Logiciels similaires

- Pour dialoguer avec le matériel : Système d'exploitation, Firmware
- Pour accomplir ses tâches : logiciels, programmes, …

Périphériques différents

- Interfaces : Connexions réseau, écrans, claviers, …

La mémoire : une bibliothèque plus ou moins grande

Le guichet et les fiches numérotées

- Permet de stocker des informations comme nombre entiers

Toute information *d'un ordinateur* peut être vue comme des nombres entiers

- Fiches numérotées par des adresses entières. Exemple : la fiche numéro 221 contient la valeur 18.

L'interprétation de l'information n'est pas incluse ⟶ notion de codage

Les performances

- Guichet unique d'accès : une requête à la fois.
- On peut *écrire* une valeur dans une fiche ou *lire* une fiche, rien d'autre

On peut aussi demander un paquet de fiches contiguës ⟶ plus rapide !

- Notion de *mémoire cache hiérarchique* : copie de Grande Bibliothèque dans une bibliothèque plus rapide et plus petite
- Performance : de l'ordre de 20 Go/s

Le processeur : un moteur à quatre temps

Un assemblage hétéroclite

- Une unité de calcul qui sait faire... des calculs (simples)
- Des registres qui retiennent chacun une valeur
- Des circuits de transmission contrôlables électriquement, qui relient les composants entre eux et aussi le processeur à la mémoire.
- Une unité de contrôle qui découpe une *instruction* en morceaux et contrôle les transmissions des circuits en fonction des résultats.

Un cycle vital immuable

Le processeur effectue des opérations très rapidement, en suivant toujours la même procédure générale :

1. *Récupération* de l'instruction : on demande à la mémoire le contenu d'une adresse, dont la valeur est trouvée dans le registre PC.
2. *Décodage* de l'instruction : la valeur est analysée, les circuits de transmission sont mis en route
3. *Exécution* de l'instruction : l'unité de calcul est mobilisée
4. *Écriture des résultats* : un registre sauvegarde le résultat, le PC est augmenté de 1

Des instructions spécifiques, au lieu de calculs, permettent d'accéder à la mémoire en lecture (étape 2) ou écriture (étape 4) au lieu des registres.

L'étonnante efficacité

- Les instructions données doivent être simples (opérations arithémtiques entre deux valeurs, tests élémentaires uniquement).
  - Les registres sont très rapides ; la durée d'un cycle est de l'ordre de la nanoseconde.
  - Toute opération complexe est divisée par un humain en opérations élémentaires ⟶ *programmation*.
  - Les instructions forment un code compact appelé *code machine*.
- Analogie : pour faire une multiplication, on peut faire plein d'additions et tester si on arrive à 0.

Les grands défauts

- Aucune intelligence
- Aucune compréhension réelle des valeurs manipulées
- On ne peut pas tout surveiller ⟶ *bugs*

L'horizon matériel

Interaction avec le matériel

- Heureusement le programmeur ou l'utilisateur n'interagit pas directement avec le matériel (sauf pour remplacer une pièce défectueuse ou connecter un nouveau matériel …). Le dialogue avec l'architecture matériel est l'affaire de programmes dédiés.
- Plusieurs couches logicielles existent entre le matériel et l'utilisateur : les *firmwares*, le noyau du système et les outils et programmes du système d'exploitation.
- La plupart des logiciels que vous serez amené à développer n'interagiront qu'indirectement avec le matériel par le filtre des librairies système.

Haut Niveau ⟶

- Logiciel,langages de programmation, …

C'est le domaine de l'informatique et des informaticiens

Une interface : Le système d'exploitation

Bas niveau

- *Firmwares*, exécution des instructions machine, …
- C'est le domaine de la physique et des électroniciens.

Le système d'exploitation

Le système d'exploitation

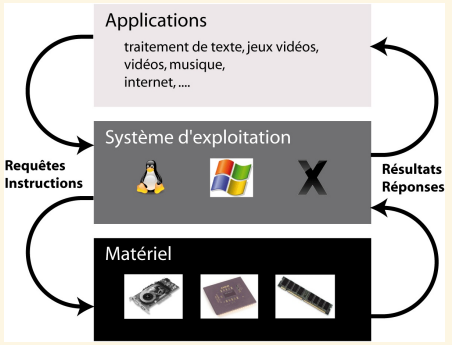
Le système d'exploitation permet de développer des programmes sans tenir compte de la complexité physique de la machine. Les programmes utilisent des fonctionnalités standardisées d'accès aux ressources matérielles.

Côté Système, l'O.S.

- coordonne l'utilisation des ressources (par exemple quel « programme » utilise le processeur à un moment donné, allocation de la mémoire, …),
- assure la maintenance et la fiabilité du système (par exemple gestion des fichiers, de la sécurité informatique, …)
- fournit des services commun à tous les programmes

Côté utilisateur, l'O.S.

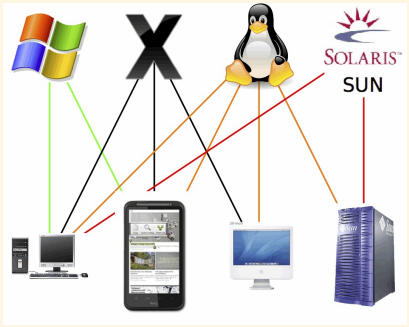
- facilite l'accès et l'utilisation des ressources matérielles,
- propose une interface de programmation permettant d'utiliser ces matériels



Les différents systèmes d'exploitation

Beaucoup d'OS différents existent :

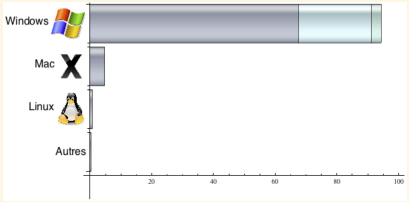
Chaque architecture matérielle demande un système d'exploitation adapté. Certain systèmes d'exploitation sont plus souples et prennent en charge des architectures matérielles multiples.



Trois OS se distinguent :

Windows est le système d'exploitation le plus utilisé, OS X est réputé le plus simple et Linux est le système d'exploitation le plus souple. Statistiques au 5 janvier 2011 : `http://gs.statcounter.com/`

- 90% des ordinateurs utilisent Windows,
- il existe plus de 600 distributions Linux…



Les différents systèmes d'exploitation

Linux

- Non propriétaire : Gratuit le plus souvent
- Ouvert : sources disponibles
- Flexible : sources modifiables
- Puissant : Programmable
- Communauté active : entraide des utilisateurs
- Plus complexe : plutôt pour les informaticiens (interfaces de programmation optimisées)

Windows

- Propriétaire : Payant
- Sources non disponibles
- Sources non modifiables
- Communauté active : nombreux utilisateurs, services payants
- Plus ergonomique : pour les utilisateurs (interfaces d'utilisation optimisées)

Les systèmes, en constante évolution

Depuis une dizaine d'année, Linux et Windows ont beaucoup évolué. La plupart des distributions Linux proposent des systèmes d'installation automatisés, des outils de bureautique ressemblant aux suites commerciales. Il bénéficie en outre d'une sécurité accrue à l'heure des virus et autres failles de sécurité. Windows propose de plus en plus de fonctionnalités empruntées à Linux.

Le système Linux

Un peu d'histoire
Debian : La distribution utilisée à l'IUT
Un système multi-utilisateurs
Une interface graphique
Les logiciels disponibles
Distribution et accès aux logiciels
La ligne de commande
De l'aide sur Linux et les commandes Shell

Un peu d'histoire

GNU-Linux

- Le système GNU-Linux est la rencontre d'une technologie, le noyau Linux et d'une philosophie de développement et de diffusion. C'est un système au développement collaboratif (par une communauté) qui est distribué librement et permet l'utilisation de tous les logiciels libres développés pour son architecture.
- Le noyau Linux est historiquement une version libre du système UNIX développé initialement par le Finlandais Linus Torvalds à partir du début des années 1990.
- Le projet GNU est celui du développement collaboratif et libre d'un système d'exploitation libre initié par Richard Stallman en 1983.

Aujourd'hui

- C'est un système très largement diffusé et utilisé sur lequel ont été développées plusieurs distributions (qui sont des suites logicielles qui accompagnent le noyau).
- Initialement confidentiel et réservé à des spécialistes avec des interfaces rudimentaires, il est aujourd'hui toujours plus ergonomique et automatisé pour les non spécialistes, mais laisse les outils et interfaces de bas niveau disponibles au plus grand nombre.
- On notera par exemple l'existence de nombreuses interfaces graphiques *Bureaux* (GNOME, KDE, …) de nombreux paquetages pré-compilées, de nombreux outils d'administration et de services (protocoles, …)

À l'IUT : Debian

Une distribution téléchargeable

`http://www.debian.org/`

Pour ce cours

- Les concepts abordés dans ce module sont généraux.
- Il pourront être testés sur tous les systèmes Linux (avec de très faibles variantes).
- Il vous est possible d'installer une version de Linux sur votre ordinateur personnel (installation ou version Live) pour votre pratique personnelle et la préparation de l'examen.
- Une pratique régulière devrait vous assurer une bonne note à peu de frais…

Pour vous préparer à l'examen

Il vous est possible :

- d'utiliser Linux dans les salles machines,
- d'installer une version de Linux sur votre ordinateur personnel (installation ou version Live).

Un système avec plusieurs utilisateurs

Des utilisateurs et des droits

- Chaque personne accédant au système est identifiée par un **nom d'utilisateur** (dit *login*) et un mot de passe (dit *password*).
- Chaque utilisateur bénéficie de permissions : exécution de certains programmes, lecture de certaines données, écriture de fichiers seulement dans certains répertoires.
- Chaque utilisateur bénéficie d'un *espace de travail* réservé sur le disque. C'est un répertoire de l'arborescence dans lequel l'utilisateur a tous les droits : il peut y créer des sous-répertoires, y écrire des fichiers, y installer des programmes et applications. Toutes ses données et préférences personnelles y sont regroupées.
- Ce répertoire est appelé "Répertoire Personnel" ou *"Home Directory"*. Il est en général placé dans un répertoire qui s'appelle `/home/` et porte le nom de l'utilisateur.

Superutilisateur - Root

- certains utilisateurs ont des permissions étendues pour administrer le système et effectuer des opérations interdites à l'utilisateur normal.
- l'utilisateur `root` a tous les droits dans le système (par exemple il peut changer les permissions de n'importe quel fichier, il fixe les noms d'utilisateur et les mots de passe, il peut installer des programmes et librairies dans les répertoires système, …)

Identification en 2 étapes

Étape #1

S'identifier en donnant au système son nom d'utilisateur

Identification en 2 étapes

Étape #2

Valider son identité avec le mot de passe

Exercices

Ce TP est un premier contact avec le système d'exploitation Linux. Il vous permettra d'appréhender les différences entre cet OS et ceux que vous pouvez avoir l'habitude d'utiliser (Windows, MacOS-X). Nous présenterons au cours du TP les grandes lignes de l'environnement de travail XFCE, la façon dont on peut interagir avec le système d'exploitation au moyen de l'outil "Terminal" ainsi que les outils de base pour envoyer des mails (configuration de votre compte mail à l'IUT) et pour obtenir de l'information sur internet (notamment sur Linux). Il existe de nombreuses versions gratuites ou payantes de Linux. La distribution installée à l'IUT se nomme Debian et est téléchargeable depuis `http://www.debian.org/`.

Connexion initiale

Q1 Lorsqu'on allume l'ordinateur un laps de temps est nécessaire pour charger le système d'exploitation. Au terme de ce chargement, une interface graphique propose à l'utilisateur de s'identifier. Linux est un système d'exploitation multi-utilisateur. Chaque utilisateur doit systématiquement s'identifier ("login") auprès du système pour avoir le droit de l'utiliser. Une fois identifié, l'utilisateur à accès a ses fichiers et son espace de travail personnel. Une fois qu'il a fini d'utliser le système, l'utilisateur se déconnecte ("logout"). La période entre l'identification et la connexion est appellée "session d'utilisation". Démarrez votre ordinateur.

Q2 Connectez-vous ! Votre identifiant est votre numéro d'étudiant, votre mot de passe est votre numéro INE. Attention : les identifiants et les mots de passe sont sensibles à la casse. Cela veut dire que les caractères majuscules et minuscules sont distingués.

Exercices

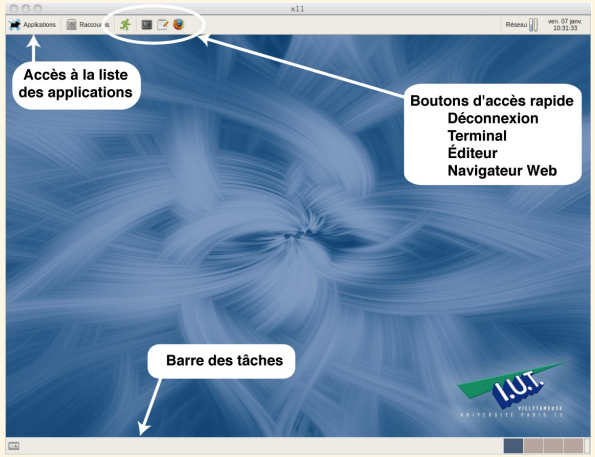
Métaphore du bureau

Contrairement aux systèmes d'exploitation propriétaires, l'environnement de travail (bureau) n'est pas directement lié au système d'exploitation. Les deux environnements de travail les plus utilisés sous Linux sont GNOME (`http://www.gnomefr.org/`) et KDE (`http://fr.kde.org/`). L'environnement choisi à l'IUT est XFCE (lié à GNOME mais plus léger). Une fois la session lancée et l'environnement chargé, vous arrivez dans un espace de travail appelé *bureau*. Cet environnement de travail est assez proche de celui qui peut être proposé par les systèmes d'exploitation propriétaires. Au moyen de la souris, vous pouvez intéragir avec le système. En cliquant sur les éléments graphiques, vous pouvez ouvrir des menus, lancer des programmes, quitter le système...

Q3 Identifier la barre de menu, la barre de tâches et le bureau.

Q4 Dans cet environnement, identifiez deux façons de lancer le navigateur internet (Firefox), et l'application terminal ().

Accès au système



Le bureau XFCE

Parmi les différents environnements graphiques existants, vous utiliserez l'environnement XFCE (`https://www.xfce.org/`).

Exercices

Lancement d'applications

Comme la plupart des systèmes d'exploitation modernes, la distribution de Linux mise à votre disposition est un système multi-tâches. Cela signifie, que vous pouvez exécuter en parallèle plusieurs applications. Il n'est pas rare que lors d'une session vous lanciez plusieurs programmes où chaque programme est associé à une fenêtre. À la suite des exercices précédents, vous devez avoir au moins 4 fenêtres ouvertes (même si elles ne sont pas toutes visibles à l'écran). Les fenêtres ouvertes apparaissent dans la barre des tâches située dans la partie basse de l'écran qui doit alors ressembler à ça :

Q5 Donnez différentes façons de passer d'un programme à l'autre, d'une fenêtre à l'autre, (au moyen de la souris ou du clavier) ? Qu'observez-vous au niveau de la barre des tâches lorsque vous passez d'une application à l'autre ?

Q6 Identifiez l'outil permettant de passer d'un bureau à l'autre. Décrivez dans quelles situations ces bureaux peuvent-être utiles. Trouvez comment on déplace une fenêtre depuis un bureau vers un autre.

Q7 Placez sur les bureaux 1 et 2, une fenêtre de terminal chacun et sur les 3 et 4, une fenêtre de navigateur.

Résultat attendu : 

Q8 Expliquez la fonction de chacun des boutons placés dans le coin supérieur droit des fenêtres. Fermez les fenêtres des programmes suivants : un terminal (bureau 1) et un navigateur (bureau 3).

Les logiciels disponibles

Les suites bureautiques

- Les suites bureautiques proposent les fonctionnalités grand public de traitement de texte, de tableur, de présentation, de dessin.
- Plusieurs suites gratuites existent en libre accès sous linux
  - CalligraSuite (`http://www.calligra-suite.org/`)
  - OpenOffice (`http://fr.openoffice.org/`)
  - …

Les programes dédiés

- Navigateur Web, Client de messagerie, comme sous d'autres OS, de nombreuses solutions existent.
  - Firefox, Opera, Konqueror, …
  - Thunderbird, KMail, …
- Des logiciels parmi les plus puissants :
  - Manipulation et création d'images : GIMP, ImageMagick, …
  - Modélisation 3D : Blender, …

De nombreuses micro-application ou programmes

- De nombreux programmes de conversion de format, de communication et de téléchargement existent en ligne de commande …

Exercices

Éditeur de texte

Nous allons créer un nouveau fichier. Pour cela nous allons utiliser un outil fondamental pour tout programmeur : un éditeur de texte. Plusieurs éditeurs de texte sont à votre disposition (vous pouvez explorer le menu Applications →Accessoires ou Applications →Développement dans la barre de menu de XFCE). À la différence de logiciels tels que Word, un éditeur de texte ne permet que de saisir du texte brut, sans mise en forme. Les programmes sont en général écrits dans un éditeur de texte. Pour lancer un éditeur de texte trois moyens sont à votre disposition :

- Lancer l'application depuis le menu application,
- Lancer l'application depuis une icône du bureau,
- Lancer l'application depuis la ligne de comande, par exemple en tapant :

```
login@host:~$ gedit ↩
```

Ceci aura pour effet d'ouvrir une fenêtre de l'éditeur.

Exercices

Éditeur de texte (suite)

Q9 Tapez du texte dans la fenêtre et enregistrez le fichier dans votre répertoire personnel, avec le nom `fichier_test_1.txt`.

Q10 Définissez ce qu'est un *raccourci clavier* et à quoi il sert (aidez-vous d'Internet si nécessaire). Donnez une liste d'au moins 8 *raccourcis clavier* standards les plus utilisés des éditeurs de texte.

Q11 Modifiez le fichier texte `fichier_test_1.txt` pour que le texte suivant y figure :

> *Ondoyons un poupon, dit Orgon, fils d'Ubu. Choux, bijoux, poux, puis du mou, du conflit, buvons non point un grog : un punch. Il but du vin itou, du rhum, du whisky, du coco, puis il dormit sur un roc.*

Q12 En utilisant les raccourcis clavier ou les menus et après les avoir testés, donnez les combinaisons ou procédures permettant de :

- Rechercher dans ce texte toutes les occurrences de la chaîne de caractères `oux`.
- Remplacer toutes les occurrences de la chaîne de caractères `oux`, par la chaîne de caractères `ou`.
- Supprimer toutes les occurrences de la chaîne de caractères `du`.

Q13 Enregistrez les modifications dans un nouveau fichier appelé `fichier_test_2.txt`.

Distribution et accès aux logiciels

Licences libres (open source)

Elles permettent de :

- d'utiliser le logiciel,
- d'étudier et de modifier les sources,
- de redistribuer les sources, modifiées ou non.

Licences Propriétaires

Elles restreignent un ou plusieurs des droits listés *supra*.

Gratuit ne signifie pas libre

Certains logiciels gratuits sont des logiciels propriétaires).

Copyright© contre Copyleft©

Le Copyleft©utilise le cadre légal du copyright pour inverser les rapports de force : le code distribué peut être modifié et redistribué, mais uniquement avec les mêmes droits →Les logiciels qui dérivent des sources Copyleft ne peuvent être distribués hors Copyleft.

Tout logiciel a un coût de développement

En général :

- Propriétaire est payant : On paie un coût de développement, un service de support, un service de mise à jour, ... Les sources sont protégées et seuls les propriétaires y ont accès.
- Libre est gratuit : Le coût est supporté par une communauté (utilisateurs, subventions publiques,

La ligne de commande

Interface de communication avec le système (IHM)

- Interface historique en mode texte,
- Interface privilégiée sous Linux : de nombreux programmes ne peuvent être appelés qu'à partir de la ligne de commande,
- Interface puissante et programmable.

Principes de fonctionnement

1. L'utilisateur tape des commandes sous forme de texte
2. Le texte est évalué par un interpréteur,
3. L'interpréteur lance l'exécution des commandes.

Utilité

- Permet de lancer des programmes ou des applications,
- Permet d'interroger le système et d'interagir avec lui.
- Basé sur un interpréteur, un langage de programmation permet de construire des scripts pour effectuer des tâches complexes de gestion ou d'administration.

La ligne de commande

```
login@host:~$ ■
```

La fenêtre de terminal ou Shell

La ligne de commande est un programme fenêtré simple qui permet de taper du texte.

- La ligne de commande comporte une partie non interprétée [user@localhost ~] appelée le *prompt*. Ici le prompt est configuré pour afficher le nom de l'utilisateur, le nom de la machine, et le nom du répertoire courant.
- Le caractère ■ marque la position du curseur. C'est là qu'est inséré le texte frappé par l'utilisateur.
- Le texte tapé par l'utilisateur sera évalué comme une (ou plusieurs) *commande(s)* par un interpréteur.

L'interpréteur

- L'interpréteur parcourt le texte tapé par l'utilisateur, identifie les commandes et les paramètres, et si la syntaxe est correcte, lance un processus.
- Plusieurs interpréteurs existent : csh, tcsh, bash. Dans ce cours nous utiliserons le **bash**.
- Bash est l'interpréteur du projet GNU. Il est le plus utilisé sous linux.

La ligne de commande

```
login@host:~$ ls
public_html/
login@host:~$ ■
```

Exécution d'une commande

- La commande (ici `ls`) est évaluée (lancée, interprétée) dès que l'utilisateur presse la touche ↵ (Entrée). L'ensemble du texte partant du prompt jusqu'à la fin de la ligne est interprété comme une commande.
- Si la commande est valide, un programme est lancé.
- Durant l'exécution du programme, la ligne de commande est indisponible. L'utilisateur doit attendre la fin de l'exécution du programme avant de pouvoir taper une nouvelle commande.
- Si le programme produit un affichage (ici `ls` affiche le nom des fichiers et répertoires), celui-ci est affiché par défaut dans la fenêtre du Shell.
- Une fois la commande exécutée, le Shell propose une nouvelle ligne de commande où l'utilisateur peut taper une nouvelle instruction.

La ligne de commande

```
login@host:~$ nom_commande options
paramètres ↵
affichage
...
login@host:~$ ■
```

Interpretation de la commande

nom_commande	Le premier mot doit correspondre au nom d'une commande connue du système,
options	Comme le nom l'indique les options ne sont pas obligatoires. Si il n'y en a pas la commande s'exécute selon un mode « par défaut ». L'ajout d'une option pourra modifier ce comportement par défaut. Attention à la différence entre `-` et `--`
paramètres	Certaines commandes peuvent fonctionner sans paramètre.

Se documenter sur le fonctionnement de Linux

Ressource sur le Web

- Les forums d'utilisateurs :
  - `https://wiki.debian.org/fr/FrenchLists`
  - `http://www.lea-linux.org/`
  - `http://www.linux-france.org/`
- Les pages Wikipedia pour les commandes, les concepts.
  - `http://fr.wikipedia.org/`
- De nombreux sites de description du système Linux
  - `http://www.linux-france.org/article/man-fr/`

Les pages de `man`

- La ligne de commande intègre une aide pour les commandes les plus courantes. La consultation des pages de `man` est essentielle pour avancer dans la maîtrise des commandes bash. Cela doit devenir un reflexe.
- Les pages de `man` détaillent les syntaxes, options et arguments des commandes. Ces options peuvent être très nombreuses.
- Les pages de `man` sont rédigées en anglais (une version française en ligne est disponible pour certaines commandes). Mais l'anglais est omniprésent en informatique, alors il faut vous faire une raison ...

Syntaxe pour man

```
man nom_de_la_commande
```

Description

- permet d'accéder à la documentation d'utilisation d'une commande (les pages de `man`).
- Les pages de `man` décrivent les syntaxes, les options, les arguments des commandes.
- Elles décrivent les résultats des évaluations et le format de ces résultats.

Exemple d'utilisation:

```
login@host:~$ man ls
```

affiche :

```
LS(1) BSD General Commands Manual LS(1)

NAME
ls - list directory contents

SYNOPSIS
ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1] [file ...]
```


Exercices

Usage du terminal

Une fenêtre de terminal est un outil de base fondamental à toute personne travaillant sous Linux. Cette fenêtre propose ce que l'on appelle une ligne de commande. C'est un moyen d'adresser directement des commandes au système, sans avoir à passer par une interface graphique. C'est un outil très puissant qui est de plus programmable. De ce fait, la ligne de commande permet de faire des choses qu'aucun programme graphique n'est capable de faire facilement. Cependant pour l'utiliser efficacement un apprentissage est nécessaire. Ce module est là pour vous en donner un aperçu.

Q14 Rappelez la structure de la ligne de commande telle qu'elle s'affiche dans le terminal (décrivez les différents éléments et leur rôle).

Q15 Évaluez la commande suivante et commentez l'affichage produit : `man ls`

Q16 Quelle est la fonction de la commande `ls` ?

Q17 Testez la commande ls avec plusieurs options parmi celles que vous avez identifié. Vérifiez que le comportement de la commande est modifié par l'utilisation d'options différentes.

Exercices

Usage du navigateur internet

Un navigateur internet tel que le logiciel Firefox (lancé plus tôt), est un outil de base dans tout travail informatique. Ces logiciels permettent de « naviguer » sur les pages internet. Les pages internet sont regroupées en sites internet, qui sont identifiés par une adresse. Certains proposent de l'information, des applications, le contenu d'autres est plus incertain. Le principe de base pour naviguer d'une page à l'autre sont les *liens hypertextes*. Précisés par le langage HTML, un *lien hypertexte* est une mise en forme qui associe un texte ou un élément graphique de la page à l'adresse d'une page internet. En cliquant sur le *lien hypertexte*, la page correspondant à l'adresse s'affiche dans le navigateur.
Dans la plupart des cas, il est simple d'identifier le texte supportant un lien hypertexte. Celui-ci est coloré ou souligné de façon à le distinguer des autres éléments de la page. La fenêtre d'un navigateur se structure en plusieurs parties que vous devez apprendre à identifier et à utiliser :

Q18 Identifiez et nommez les différents éléments qui composent la fenêtre d'un navigateur internet.

Q19 Donnez au moins 2 adresses correspondant à des moteurs de recherche

Q20 Avec un moteur de recherche, trouvez l'origine du nom de la distribution linux *Debian* ?

Exercices

Usage du client de messagerie électronique (e-mail)

Si votre inscription à l'IUT est finalisée, un compte mail personnel à été créé à votre nom. Son adresse est de la forme : `Prenom.Nom@edu.univ-paris13.fr`
Grâce à un logiciel appelé *client mail*, vous pouvez envoyer et recevoir du courrier électronique.
Consultez-le très régulièrement (au moins une fois par jour) !
Un moyen d'accéder à vos mails est d'utiliser le client web-mail de l'université : une application accessible depuis n'importe quel navigateur internet (connecté). L'adresse du web-mail de l'IUT est :
`http://ent.univ-paris13.fr`
Pour accéder à votre courrier vous devez fournir votre identifiant et votre mot de passe.

Q21 Après votre connexion au web-mail et après avoir identifié et cliqué sur le service de messagerie électronique, identifiez les différents boutons et champs de l'interface.

Q22 Après avoir sélectionné le service de rédaction d'un message, identifiez les différents champs de la fenêtre de rédaction. Décrivez à quoi servent les champs "À", "Cc", "Cci", "Sujet" et "Texte".

Q23 Renseignez les champs nécessaires et envoyez un mail à votre voisin de table.

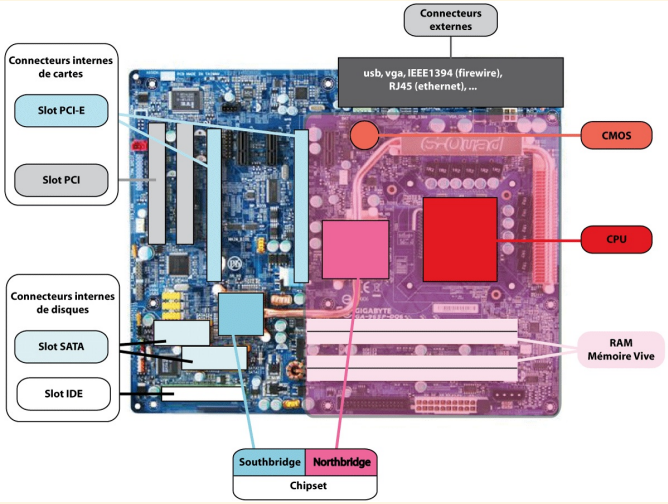
Q24 Ouvrez le mail que votre voisin vous a envoyé et répondez-lui dans le corps du message reçu.

Q25 Donnez la procédure pour ajouter l'adresse du web-mail de l'université dans les racourcis (onglets et favoris) de votre navigateur internet.

L'ordinateur de bas en haut

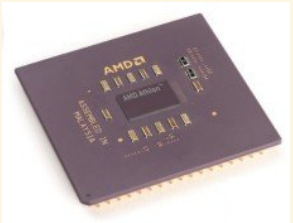
Le matériel

La carte mère



La carte mère est l'élément central de l'ordinateur sur lequel sont assemblés et mis en relation tous les composants matériels. Elle permet à tous ses composants de fonctionner ensemble efficacement.

Les unités de calcul



CPU - Central Processing Unit

- C'est une puce qui traite des instructions élémentaires en réalisant des calculs binaires,
- Fréquence de l'ordre de 3 GHz.

GPU - Graphics Processing Unit

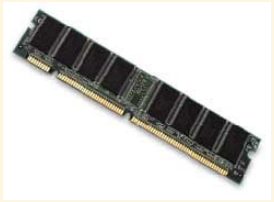
C'est une puce placée sur les cartes graphiques

- Elle prend en charge les nombreux calculs de rafraichissement des images 3D
- Une carte graphique moderne peut compter une grande quantité de ces puces.

Des mémoires différentes pour des usages différents

ROM : Read Only Memory

- Mémoire non-volatile maintenue par une conception physique,
- Taille limitée car très chère, très rapide,
- Contient instructions d'amorçage, routines…



RAM : Random Access Memory

- Mémoire volatile : maintenue par une tension électrique,
- Accès rapide,
- Taille limitée car assez chère.

Disque Dur, clef-usb, …

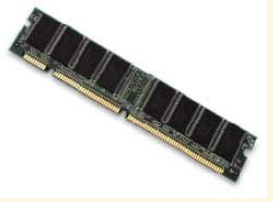
- Mémoire non-volatile (enregistrement magnétique le plus souvent),
- Accès lent,
- Taille très grande (support de stockage de masse), beaucoup moins chère.

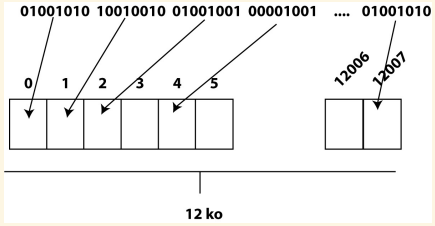
Des mémoires différentes pour des usages différents

Organisation de la mémoire

Les ordinateurs réalisent des calculs logiques sur des données binaires

- Les données et les instructions sont stockées sous forme de blocs repérés par une adresse,
- Les blocs contiennent une information binaire organisée en octet. Chaque octet contient 8 bits d'information qui sont lus comme une suite ordonnée de 0 ou de 1 ou de Vrai et de Faux.
- Un octet peut prendre $2^8 = 256$ valeurs différentes.





Les périphériques

Des composants externes

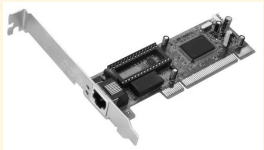
En fonction de leur tâche, de nombreux composants *ad hoc* peuvent être *greffés* sur la structure de base précédemment décrite. Par exemple :

- Ordinateur de Maison : Écran, souris, imprimante, scanner, joystick, modem, …
- Ordinateurs de bord : Sondes, actioneurs, …
- Télephone : Antenne, récepteurs, …
- Robot médical : Interface haptique, bras mécaniques, …

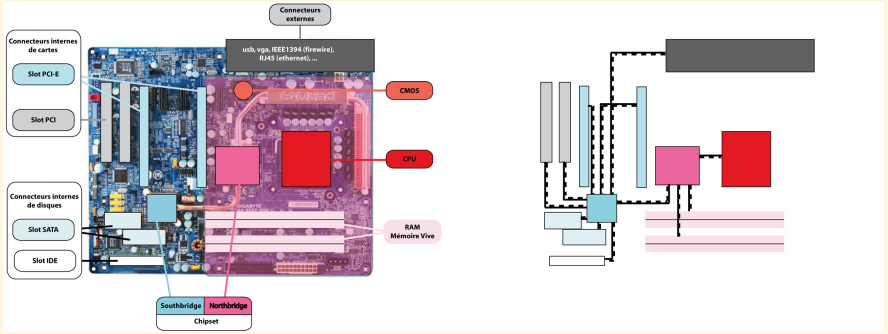
Des composants internes

En fonction des possibilités des cartes mères plusieurs types de composants peuvent être ajoutés :

- Cartes vidéo, Cartes son, disques durs internes, lecteurs, …
- Cartes d'acquisition ou de pilotage de périphériques, …



Les bus



La carte mère intègre les bus.

- Les bus sont des unités physiques qui assurent le transport efficace de l'information entre les différents composants connectés à la carte mère,
- La largeur (8, 16, 32 64 bits), série ou parallèle et la fréquence ($10^2 - 10^3$ MHz) des bus règlent le débit d'information entre les composants. Cela conditionne donc fortement l'efficacité d'une configuration matérielle.

Organiser ses données

Un fichier

De l'information au stockage

Les informations utilisées dans un ordinateur sont stockées dans la *mémoire de masse*, qui se distingue de la *mémoire vive* par sa résistance à l'extinction et de la *mémoire morte* (et plus tard, du *firmware*) par sa mutabilité.
Les performances des systèmes de stockage de masse sont meilleures chaques années, mais l'ordre de grandeur reste la ms ou 100 µs.

De l'information au fichier

L'information est découpée en petites unités qui s'appellent des fichiers, sémantiquement cohérentes — ce sont des informations qui « vont ensemble ». Ces éléments de base du stockage informatique peuvent ne contenir que très peu d'information ou représenter plusieurs Go de données par fichier.
Un fichier est lié à la façon dont on y accède (son *nom* et son *chemin*), mais nous verrons que ce n'est pas un identifiant : il peut y avoir plusieurs accès différents à un même fichier (*liens*).

Noms et contenu des fichiers

La décomposition traditionnelle d'un nom de fichier

Deux parties séparées par un point :

- La 1ère partie informe sur la nature du contenu du fichier,
- La 2ème partie informe sur le format ou la finalité des données.

nom.extension
prefix.suffix
description.format

Selon les systèmes, certains caractères sont interdits. Par exemple ∗ sous Windows, / sous Linux.

Exemples de noms de fichiers

Extension	Contenu
.c	Sources C
.html	Document Web
.pdf	Document Mis en page
.txt	Texte brut

Enigmatique	Informatif
e3.c	teste_boucle_for.c
New.pdf	2011_IntroSys_cours_1.pdf
toto.sh	test_boucle_for.sh

Choix des noms

Ils doivent être choisis minutieusement pour être informatifs.

Choisir un nom : réfléchir pour un gain de temps pour retrouver le fichier ou le répertoire concerné.

Importance de la casse (Linux), tolérance ailleurs (OS X, Windows).

Des fichiers et des répertoires

Les fichiers... en vrac ?

Les fichiers sont regroupés dans des répertoires (en anglais *directory* ou *folders*). Les répertoires peuvent contenir des fichiers ou d'autres répertoires. L'organisation des fichiers est réglée par le *système de fichiers* (ang. *filesystem*).

- Cette organisation arborescente permet de faciliter la recherche d'un fichier,
- Les fichiers sont regroupés par application, par thème, par format, par fonction, …
- Organisation *hiérarchique* qui permet d'organiser les données et de faciliter leur accès.

De très nombreux fichiers et répertoires

Le nombre de fichiers enregistrés sur un disque dur peut aisément dépasser 100.000 fichiers,

- Dans un même répertoire le nom est un identifiant.
- Les répertoires et les fichiers partagent les mêmes noms.

Sous Windows, pas d'extension pour les répertoires.

Remarque

Avec tous les fichiers au même *endroit*, il est très difficile de les lister (trop à lire).

Exemple d'arborescence Linux

```
/ ..... Répertoire racine (Root Directory)
├── bin/
│   └── (…)
├── home/
│   ├── moi/ ..... Répertoire personnel (User directory)
│   │   ├── Mes Documents/
│   │   │   ├── ListeDesCourses.txt
│   │   │   └── Exercice_1.sh
│   │   │   └── (…)
│   │   └── anonymous/
│   │       ├── LisezMoi.txt
│   │       └── Telechargements/
│   │           └── (…)
│   └── (…)
```

Les répertoires importants

- La **racine** (*Root directory*) contient tous les répertoires et fichiers accessibles depuis le système.
- Le **répertoire personnel** (*User Directory* ou *Home Directory*) est le répertoire dans lequel l'utilisateur peut faire ce qu'il veut (écrire, modifier, supprimer, installer …).

Répertoire courant et chemins relatifs

Le répertoire courant

- Le répertoire courant est un répertoire de référence d'où sont lancées les commandes du shell.
- Par défaut, le répertoire courant est le répertoire personnel de l'utilisateur,
- Naviguer dans l'arborescence équivaut à modifier le répertoire courant.

Exemples de chemins relatifs

```
home/ ..... ../..
└── moi/ ..... ../
    ├── Etoiles/ ..... Répertoire Courant ./
    │   ├── SOLEIL.jpg ..... SOLEIL.jpg ou ./SOLEIL.jpg
    │   └── Antares.jpg ..... Antares.jpg ou ./Antares.jpg
    └── Systeme_Solaire/ ..... ../Systeme_Solaire/
        └── terre.gif ..... ../Systeme_Solaire/terre.gif
```

Syntaxe d'un chemin relatif

- Le chemin *relatif* d'un fichier ou d'un répertoire donne la liste des répertoires et sous-répertoires en partant du répertoire courant (la référence *relative* dans l'arborescence) jusqu'à la cible.
- Il est relatif, car lorsque le répertoire courant change, le chemin relatif change.

La notion de chemin

Le chemin définit un accès unique à partir de la racine

- Deux fichiers ou répertoires ne peuvent pas porter le même nom si ils sont dans un même répertoire.
- Sous Linux, les noms des fichiers et répertoires différencient les caractères MAJUSCULES et minuscule. Les fichiers Essai.txt et essai.txt peuvent donc être dans le même répertoire.

Exemples de chemins absolus

```
/ ..... Un chemin absolu part de la racine /
└── home/ ..... /home/
    └── moi/ ..... /home/moi/
        ├── Etoiles/ ..... /home/moi/Etoiles/
        │   ├── SOLEIL.jpg ..... /home/moi/Etoiles/SOLEIL.jpg
        │   └── Soleil.jpg ..... /home/moi/Etoiles/Soleil.jpg
        └── Systeme_Solaire/ ..... /home/moi/Systeme_Solaire/
            └── SOLEIL.jpg ..... /home/moi/Systeme_Solaire/SOLEIL.jpg
```

Syntaxe d'un chemin absolu

Le chemin *absolu* d'un élément du système de fichier est unique (sauf avec un *lien*). Il donne la liste des répertoires et sous-répertoires en partant de la racine / (la référence de l'arborescence) jusqu'à la cible.

Notation spéciales

Les chemins des répertoires de référence

Répertoire	Notation
Répertoire racine	/
Répertoire personnel	~

Répertoire	Notation
Répertoire courant	.
Répertoire parent	..

⚠ La notation ~ est un chemin absolu, remplacée par le vrai chemin avant l'exécution des commandes. C'est un raccourci *au niveau du shell, pas au niveau du système d'exploitation*.

Exemple de chemins valides pointant le fichier cible

```
/ ..... Répertoire Racine
└── home/
    └── moi/ ..... Répertoire Personnel
        └── Etoiles/ ..... Répertoire Courant
            └── Soleil.jpg ..... Fichier cible
```

Chemins Absolus
`/home/moi/Etoiles/Soleil.jpg`
`~/Etoiles/Soleil.jpg`
`/home/moi/../moi/Etoiles/Soleil.jpg`
`/home/moi/../../home/moi/Etoiles/Soleil.jpg`
Chemins Relatifs
`Soleil.jpg`
`./Soleil.jpg`
`../Etoiles/Soleil.jpg`
`../../moi/Etoiles/./Soleil.jpg`

L'archivage

D'une arborescence à un fichier

Une technique souvent utilisée consiste à transformer une partie de l'arborescence en un fichier qui n'est pas utilisable directement. Ce fichier peut ensuite être retransformé en une arborescence.

Le format tar

Utilisé depuis les années 80, le format tar est un pilier du monde Unix. Il est parfaitement libre. Il servait initialement aux sauvegardes sur bande magnétique (*tape archive*).
Le format tar ne permet pas la compression, mais la commande `tar` donne accès à des programmes de compression qui permettent de réduire la taille de l'archive. Une archive au format tar est appelée un(e) *tarball*.
Le compresseur le plus connu est `gzip` dont les fichiers compressés ont un suffixe `.gz`.
Souvent on combine les deux suffixes : une archive compressée peut ainsi s'appeler `textes2015.tar.gz` ou `textes2015.tgz`.

Le format zip

Principalement utilisé pour son universalité depuis 1986, le format zip est plus ou moins libre (il y a des doutes sur la possibilité de brevet sur les techniques employées). Le format zip n'est pas uniquement caractérisé par son extension : plusieurs autres formats de fichier sont en fait une archive ZIP qui contient divers documents (par exemple, un fichier `docx` pour Microsoft Word est en fait un ZIP qui contient divers fichiers XML et images).
Le format zip, en plus de l'archivage permet aussi la compression. La commande `zip`/`unzip` doit donc permettre la décompression.

Conventions

Noms et chemins

- Un chemin peut être absolu ou relatif. Il peut utiliser les notations spéciales.
- Par convention la notion de fichier sera comprise dans son sens large. Par exemple, le chemin d'un fichier devra être interprété sans distinction comme le chemin vers un fichier ordinaire ou comme le chemin vers un répertoire (sauf mention contraire explicite).

Commandes, options, paramètres

Commande c'est le nom d'un programme qui exécute une action.

Options ce sont des paramètres optionnels. Ils peuvent être omis. L'ajout d'options modifie le comportement de la commande (le résultat). Les options sont montrées encadrées par les caractères `[ ... ]` (qu'il ne faut pas mettre).

Paramètres ce sont des arguments que la commande évalue.

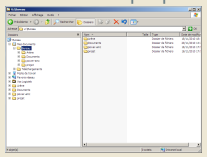
Sources et destination

Les commandes de déplacement acceptent une ou des *sources* qui sont des fichiers ou répertoires d'origine, et une *destination* qui est un fichier ou un répertoire.

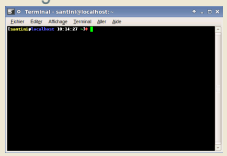
Manipulation de l'arborescence en ligne de commande

Alternatives pour naviguer dans l'arborescence et manipuler les fichiers

Interface Graphique



Ligne de Commande



Boîte à outils : manipuler l'arborescence

Commande	Fonction principale
`pwd`	Afficher le nom du répertoire courant
`cd`	Changer de répertoire courant
`ls`	Afficher le contenu d'un répertoire
`cat`	Afficher le contenu d'un fichier
`touch`	Créer un fichier
`mkdir`	Créer un répertoire
`rm`	Supprimer fichier(s) ou répertoire(s)
`cp`	Copier fichier(s) ou répertoire(s)
`mv`	Déplacer/Renommer fichier(s) ou répertoire(s)

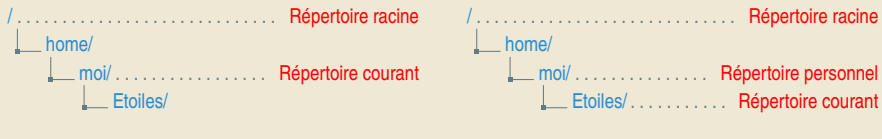
Syntaxe pour pwd

```
pwd
```

Description

- Affiche le nom du répertoire courant.

Exemple d'utilisation:




```
login@host:~$ pwd
/home/moi
```

```
login@host:~/Etoiles$ pwd
/home/moi/Etoiles
```

Syntaxe pour cd

```
cd  <cible>
```

Description

- Change le répertoire courant (permet de naviguer dans l'arborescence).
- Si le chemin du répertoire cible est omis, le répertoire courant redevient par défaut le *répertoire personnel*.

⚠ Ce n'est pas une commande, mais une fonctionnalité du shell.

Exemple d'utilisation:

```
/ ..... Répertoire Racine
└── home/ ..... Répertoire courant initial
    └── moi/ ..... Répertoire courant cas #1
        └── Etoiles/ ..... Répertoire courant cas #2
```

Commande cas #1 :

```
login@host:/home$ cd
login@host:~$ ■
```

Commande cas #2 :

```
login@host:/home$ cd moi/Etoiles
login@host:~/Etoile$ ■
```


Syntaxe pour ls

```
ls  <source>
```

Description

- Affiche le contenu d'un répertoire.
- Par défaut si aucune source n'est indiquée, la commande affiche le contenu du répertoire courant.

Exemple d'utilisation:

```
/ ..... Répertoire racine
└── home/ ..... Répertoire courant
    └── moi/
        ├── astronomie.txt
        └── Etoiles/
```

```
login@host:/home/$ ls
moi/
```

```
login@host:/home/$ ls moi/
Etoiles/ astronomie.txt
```


Syntaxe pour ls(bis)

```
ls -a <source>
```

Description

- Affiche le contenu d'un répertoire y compris les fichiers et répertoires cachés.
- Les fichiers et répertoires cachés ont un nom dont le premier caractère est un point.
- Les fichiers et répertoires cachés sont utilisés par le système ou certaines applications.

Exemple d'utilisation:

```
moi/ ..... Répertoire courant
├── ./ssh/
│   ├── id_rsa
│   ├── id_rsa.pub
│   └── known_hosts
├── .bashrc
├── astronomie.txt
└── Etoiles/
    └── soleil.jpg
```

Sans option *-a*

```
login@host:~$ ls
astronomie.txt  Etoiles/
```

Avec option *-a*

```
login@host:~$ ls -a
.        ..      .bashrc .ssh/
astronomie.txt  Etoiles/
```


Syntaxe pour cat

```
cat  fichier [fichier_2 ...]
```

Description

- Affiche le contenu des fichiers les uns à la suite des autres.
- Les fichiers sont concaténés dans l'ordre des paramètres.

Exemple d'utilisation:

Cette commande est en générale utilisée pour concaténer des fichiers textes. On l'utilise avec une commande de redirection (cf. Partie Redirections) pour enregistrer le résultat de la concaténation dans un nouveau fichier.

Soient les deux fichiers suivants :

tellur.txt

```
Mercure, Venus
Terre, Mars
```

jov.txt

```
Jupiter, Saturne
Uranus, Neptune
```

La commande :

```
login@host:~$ cat tellur.txt jov.txt
Mercure, Venus
Terre, Mars
Jupiter, Saturne
Uranus, Neptune
login@host:~$ ■
```

Syntaxe pour touch

```
touch  chemin [chemin_2 ...]
```

Description

- Si le chemin est occupé par un fichier ou un répertoire, mise à jour de la date de dernière modification.
- Sinon, création d'un ou de plusieurs fichiers vides à l'endroit spécifié par le chemin.

Exemple d'utilisation:

```
moi/ ..... Répertoire Courant
├── astronomie.txt
├── lisezmoi.txt ..... Création Commande #1
└── Stars/
    ├── TCeti.txt ..... Création Commande #2
    └── ACentauri.txt ..... Création Commande #2
```

```
login@host:~$ touch lisezmoi.txt
login@host:~$ touch Stars/TCeti.txt Stars/ACentauri.txt
```


Syntaxe pour mkdir

```
mkdir  chemin [chemin_2 ...]
```

Description

- Création d'un ou de plusieurs répertoires aux endroits spécifiés par les chemins.
- Si le chemin est occupé par un fichier ou un répertoire, il y a un message d'erreur.
- Si le chemin n'est pas déjà créé à part le dernier élément, il y a un message d'erreur.

Exemple d'utilisation:

```
moi/ ..... Répertoire courant
├── Systeme_Solaire/ ..... Création commande #1
└── Etoiles/
    ├── Rouges/ ..... Création commande #2
    └── Bleues/ ..... Création commande #2
```

```
login@host:~$ mkdir Systeme_Solaire
login@host:~$ mkdir Etoiles/Rouges Etoiles/Bleues
login@host:~$ mkdir Galaxies/M91
mkdir: impossible de créer le répertoire
« Galaxies/M91 »: Aucun fichier ou dossier de ce type
```


Syntaxe pour mkdir(bis)

```
mkdir -p chemin <chemin_2 ...>
```

Description

- Création d'un ou de plusieurs répertoires aux endroits spécifiés par les chemins.
- Si depuis la racine en suivant un chemin, on rencontre un fichier, il y a un message d'erreur.
- Si depuis la racine en suivant un chemin, il n'y pas de répertoire, il est créé.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt
├── Etoiles/
└── Galaxies/ ..... Création Commande #1
    └── M91/ ..... Création Commande #1
        └── highres/ ..... Création Commande #1
```

```
login@host:~$ mkdir -p Galaxies/M91/highres
```


Syntaxe pour rm

```
rm  chemin [chemin_2 ...]
```

Description

- La commande supprime le fichier pointé par le(s) chemin(s).
- Si le chemin pointe sur un répertoire, la commande affiche un message d'erreur.

Exemple d'utilisation:

```
moi/ ..... Répertoire Courant
├── astronomie.txt ..... Supprimé par la commande #1
├── Etoiles/
│   └── soleil.jpg ..... Supprimé par la commande #2
└── aldebaran.gif ..... Supprimé par la commande #2
```

```
login@host:~$ rm astronomie.txt
login@host:~$ rm aldebaran.gif Etoiles/soleil.jpg
```

Syntaxe pour rm(bis)

```
rm -r chemin <chemin_2 ...>
```

Description

- L'option `-r` (comme **r**écursif) permet de supprimer un répertoire et tout son contenu.

L'option `-f` (comme **f**orce) permet d'ignorer certaines questions.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt
├── Etoiles/ ..... Supprimé par la commande #1
│   ├── soleil.jpg ..... Supprimé par la commande #1
│   └── Galaxie/ ..... Supprimé par la commande #1
│       └── Andromede.pdf ..... Supprimé par la commande #1
└── aldebaran.gif
```

```
login@host:~$ rm -r Etoiles
```


Syntaxe pour cp

```
cp source cible
```

Description

- Copie le fichier source vers la cible.
- La source doit être un fichier ordinaire (pas un répertoire),
- Si la source est un répertoire la commande produit un message d'erreur.
- Si la cible :
  - est le chemin d'un répertoire existant, le fichier sera copié dans ce répertoire et conservera son nom,
  - ne correspond pas à un répertoire existant, le fichier sera copié avec le nom (chemin) `cible`.

Exemple d'utilisation:

```
moi/ ..... Répertoire courant
├── astronomie.txt ..... Fichier source commandes #1 et #2
├── Etoiles/ ..... Répertoire cible commande #1
│   ├── astronomie.txt ..... Copié/Créé par la commande #1
│   └── info.txt ..... copié/créé par la commande #2
└── cv.pdf
```

```
login@host:~$ cp astronomie.txt Etoiles
login@host:~$ cp astronomie.txt Etoiles/info.txt
```


Syntaxe pour cp(bis)

```
cp source <source_2 ...> cible
```

Description

- Copie plusieurs fichiers sources vers la cible.
- Les sources doivent être des fichiers ordinaires, et la cible un répertoire.

Exemple d'utilisation:

```
moi/ ..... Répertoire courant
├── cv.pdf ..... Fichier source
├── motivations.pdf ..... Fichier source
└── Candidature/ ..... Répertoire cible
    ├── cv.pdf ..... Copié/créé par la commande
    └── motivations.pdf ..... Copié/créé par la commande
```

```
login@host:~$ cp cv.pdf motivations.pdf Candidature
login@host:~$ cp cv.pdf motivations.pdf Candidature/ #
Moins ambigu
```


Syntaxe pour cp(ter)

```
cp -r source <source_2 ...> cible
```

Description

- L'option `-r` (**R**écursif) permet de copier un répertoire et son contenu si il apparait dans le(s) source(s).
- Attention : si le répertoire n'existe pas et qu'on copie un répertoire, il y a renommage

Exemple d'utilisation:

```
moi/ ..... Répertoire Courant
├── Galaxie/ ..... Source commandes
│   └── Andromede.pdf
├── Etoiles/ ..... Répertoire cible #1
│   ├── Galaxie/ ..... Copié/créé par la commande #1
│   │   └── Andromede.pdf ..... Copié/créé par la commande #1
└── Top10/ ..... Copié/créé par la commande #2 (renommage)
    └── Andromede.pdf ..... Copié/créé par la commande #2
```

```
login@host:~$ cp -r Galaxie Etoiles
login@host:~$ cp -r Galaxie Top10
```

Syntaxe pour mv

```
mv  source cible
```

Description

Déplace/renomme un fichier ou répertoire.

- modifie le chemin d'accès à la `source` qui devient le chemin `cible`.
- Le chemin `source` disparait et le chemin `cible` est créé.
- Le fichier ou répertoire pointé reste le même.
- La cible doit être un chemin non occupé ou un répertoire.

Exemple d'utilisation: Renommer un fichier

État Initial de l'arborescence :

```
moi/ ..... Répertoire courant
└── AstroNomIe.TXT ..... Fichier Source
```

État Final de l'arborescence :

```
moi/ ..... Répertoire courant
└── astronomie.txt ..... Fichier Renommé
```

```
login@host:~$ mv AstroNomIe.TXT astronomie.txt
```


Exemple d'utilisation: Déplacer un Répertoire

État Initial de l'arborescence :

```
moi/ ..... Répertoire courant
├── astronomie.txt ..... Fichier source
└── Etoiles/ ..... Répertoire cible
```

État Final de l'arborescence :

```
moi/ ..... Répertoire courant
└── Etoiles/ ..... Répertoire cible
    └── astronomie.txt ..... Fichier déplacé
```

```
login@host:~$ mv astronomie.txt Etoiles
```

Exemple d'utilisation: Renommer un répertoire

État Initial de l'arborescence :

```
moi/ ..... Répertoire courant
└── Etoiles/ ..... Répertoire Source
    └── astronomie.txt
```

État final de l'arborescence :

```
moi/ ..... Répertoire courant
└── Relativite/ ..... Répertoire Renommé
    └── astronomie.txt
```

```
login@host:~$ mv Etoiles Relativite
```


Exemple d'utilisation:

État Initial de l'arborescence :

```
moi/ ..... Répertoire courant
├── astronomie.txt ..... Fichier Source
├── relativite.pdf ..... Fichier Source
└── Etoiles/ ..... Répertoire Cible
```

État Final de l'arborescence :

```
moi/ ..... Répertoire courant
└── Etoiles/ ..... Répertoire Cible
    ├── astronomie.txt ..... Fichier Déplacé
    └── relativite.pdf ..... Fichier Déplacé
```

```
login@host:~$ mv astronomie.txt relativité.pdf Etoiles
```

Exemple d'utilisation:

État Initial de l'arborescence :

```
moi/ ..... Répertoire courant
├── relativite.pdf ..... Fichier Source
├── Etoiles/ ..... Répertoire Source
│   └── astronomie.txt
└── Espace/ ..... Répertoire Cible
```

État Final de l'arborescence :

```
moi/ ..... Répertoire courant
└── Espace/ ..... Répertoire Cible
    ├── relativite.pdf ..... Fichier Déplacé
    └── Etoiles/ ..... Répertoire Déplacé
        └── astronomie.txt ..... Fichier Déplacé
```

```
login@host:~$ mv relativité.pdf Etoiles Espace
```


Syntaxe pour tar

```
tar cvf nom_archive fichier_ou_repertoire [autres_sources]
```

Description

- Crée un fichier archive dont le nom (chemin) est donné en premier argument et porte classiquement l'extension `.tar`.
- Les fichiers sources qui servent a créer l'archive sont préservés par la commande `tar`.
- L'option `c` (comme **c**reate), indique que la commande `tar` doit utiliser un algorithme d'archivage.
- L'option `v` (**v**erbose), permet d'afficher le déroulement de l'archivage.
- L'option `f` (**f**ile), permet de préciser juste derrière un fichier d'archivage.

Exemple d'utilisation:

```
moi/ ..... Répertoire courant
├── astronomie.txt
├── Images/
│   ├── soleil2.jpg
│   └── Terre1.jpg
└── espace.tar ..... Créé par la commande #1
```

Regroupe dans la même archive `espace.tar` le fichier `astronomie.txt` et le répertoire `Images/` et son contenu :

```
login@host:~$ tar cvf espace.tar astronomie.txt Images
astronomie.txt
Images/
Images/soleil2.jpg
Images/Terre1.jpg
```

Syntaxe pour tar(bis)

```
tar tvf nom_archive
tar xvf nom_archive [chemin(s) dans l'archive]
```

Description

- Examine une archive ou crée des fichiers à partir de l'archive.
- Le fichier archive est préservé par la commande `tar`.
- L'option `x` (e**x**tract), permet de désarchiver.
- L'option `t` (lis**t**), permet de lister le contenu d'une archive

Exemple d'utilisation:

```
moi/ ..... Répertoire courant
├── espace.tar
├── astronomie.txt ... créé par la commande #2
└── Images ..... créé par la commande #3
    ├── soleil2.jpg .. créé par la commande #3
    └── Terre1.jpg .. créé par la commande #3
```

```
login@host:~$ tar tvf espace.tar
astronomie.txt
Images/
Images/soleil2.jpg
Images/Terre1.jpg
login@host:~$ tar xvf espace.tar
astronomie.txt
astronomie.txt
login@host:~$ tar xf espace.tar
login@host:~$
```


Exercices

Préparation

Q26 Ouvrez un terminal. Vérifiez que le répertoire dans lequel vous êtes est bien `/home/usager/123456789/`. Quelle est la commande qui permet de le faire ? (123456789 = votre identifiant)

Q27 Vérifiez le contenu du répertoire `Documents` qui est dans votre répertoire personnel. Quelle est la commande qui permet de le faire ? Est-ce qu'il y a quelque chose ?

Q28 Faites la vérification de trois façons différentes : chemin absolu, utilisation du raccourci `~`, utilisation d'un chemin relatif.

Q29 Changez le répertoire courant pour aller dans `Documents`. Quelle est la commande pour le faire ?

Q30 Créez en ligne de commande un répertoire `m1101` dans `~/Documents`. À partir de maintenant, assurez-vous que le répertoire courant est ce répertoire `m1101`.

Q31 Téléchargez l'archive contenant les données pour ce TP : Allez sur la page `http://lipn.fr/~dubacq/m1101.html`. Téléchargez le fichier `photos.tar`. Recherchez où le fichier a été écrit dans l'arborescence de votre répertoire personnel.

Q32 Donnez la (suite de) commande(s) permettant de déplacer le fichier d'archive dans le répertoire `m1101` que vous venez de créer. À la fin des commandes, le répertoire `m1101` sera toujours votre répertoire courant et ne contiendra que le fichiers photos.tar.

Q33 Quelle commande permet de vérifier que l'archive est bien dans le répertoire

Exercices

Examen de fichiers

Q34 Quelles sont les informations données par le nom du fichier ?

Q35 Les commandes `less`, `cat` et `hexdump` permettent d'afficher le contenu d'un fichier. Analysez la différence de comportement entre ces deux commandes sur le fichier `photos.tar`. Qu'en concluez-vous ? Quel est le programme le plus adapté pour voir le contenu de ce fichier ?

Q36 Relisez le manuel de la commande `tar`. Vérifiez la liste des fichiers contenus dans l'archive. Combien y en a-t-il ?

Q37 Sortez les fichiers de l'archive.

Q38 Avec les commandes de la question 35, regardez le fichier contenu dans un répertoire. Analysez la différence de comportement entre ces commandes. Qu'en concluez-vous ?

Remarques : si un affichage prend trop de temps, utilisez le raccourci clavier adéquat pour suspendre l'exécution de la commande courante. Si l'affichage de votre terminal est durablement perturbé, dans le menu Terminal →Réinitialiser le terminal.

Le métacaractère ∗

Le caractère ∗

Le shell traduit la ligne de commande en `commande argument1 argument2 ...`. Avant l'exécution, il traduit certains caractères selon des règles précisées ici.

- Le cataractère ∗ est utilisé comme un *joker* pour remplacer une chaîne de caractères,
- Il est utilisé dans un chemin pour pointer plusieurs fichiers ou répertoires existants dont le chemin partage un motif commun.
- Le caractère ∗ peut être n'importe où dans le chemin, plusieurs fois si nécessaire.

Exemple de manipulation avec la commande `mv`

```
login@host:~$ mv *.jpg Images/
```

Ici, le chemin `*.jpg` pointe tous les fichiers du répertoire courant dont le nom se fini par l'extension `.jpg`. Il pointe donc les fichiers `etacentauri.jpg` et `aldebaran.jpg` et exclue les autres fichiers (ici le fichier `alphacentauri.gif`).

```
moi/ ..... Répertoire Courant
├── aldebaran.jpg ..... Fichier ciblé
├── alphacentauri.gif
├── etacentauri.jpg ..... Fichier ciblé
└── Images/ ..... Répertoire final
```

```
moi/ ..... Répertoire Courant
├── alphacentauri.gif
└── Images/ ..... Répertoire final
    ├── aldebaran.jpg ..... Fichier déplacé
    └── etacentauri.jpg ..... Fichier déplacé
```

Exemples d'utilisation de l'étoile

Utilisation simple avec la commande mv

```
login@host:~$ mv al* Images/
```

Ici, le chemin `al*` pointe tous les fichiers du répertoire courant dont le nom commence par les caractères `al`. Il pointe donc les fichiers `aldebaran.jpg` et `alphacentauri.gif` et exclue les autres fichiers (ici le fichier `etacentauri.jpg`).

moi/ ..... Répertoire Courant
- aldebaran.jpg ..... Fichier ciblé
- alphacentauri.gif ..... Fichier ciblé
- etacentauri.jpg
- Images/ ..... Répertoire final

moi/ ..... Répertoire Courant
- etacentauri.jpg
- Images/ ..... Répertoire final
  - aldebaran.jpg ..... Fichier déplacé
  - alphacentauri.gif ..... Fichier déplacé

Utilisation double avec la commande mv

```
login@host:~$ mv *centauri* JPG/
```

Ici, le chemin `*centauri*` pointe tous les fichiers du répertoire courant dont le nom contient la chaîne de caractères `centauri`. Il pointe donc les fichiers `alphacentauri.gif` et `etacentauri.jpg` et exclue les autres fichiers (ici le fichier `aldebaran.jpg`).

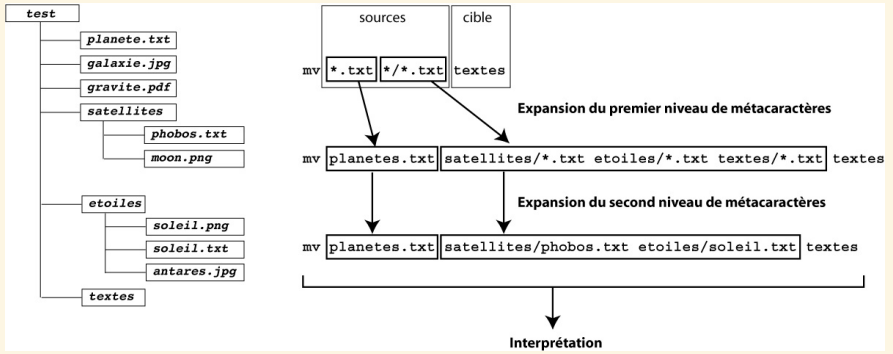
moi/ ..... Répertoire Courant
- aldebaran.jpg
- alphacentauri.gif ..... Fichier ciblé
- etacentauri.jpg ..... Fichier ciblé
- Images/ ..... Répertoire Final

moi/ ..... Répertoire Courant
- aldebaran.jpg
- Images/ ..... Répertoire final
  - alphacentauri.gif ..... Fichier déplacé
  - etcentauri.jpg ..... Fichier déplacé

Métacaractère et chemins ciblés

Exemple plus complexe et détails de l'interprétation

- Le cararctère ∗ est développé lors de l'interprétation.



Autres métacaractères

Du shell aux programmes

Il faut bien se souvenir que les métacaractères sont interprétés par le shell. Cela a deux conséquences :

- Le programme appelé ne sait pas si les noms ont été tapés en entier ou si des métacaractères ont été utilisés. Il n'a que le résultat final.
- Dans un programme, on ne peut pas utiliser les métacaractères.

Les jokers

Ce sont des motifs simples. Lorsqu'ils ne peuvent pas être instanciés, ils ne sont pas supprimés, mais passés tels quels. Exemple : `mkdir -p toto/*` selon que `toto` est un répertoire non-vide ou autre chose.
On y trouve ? qui remplace une lettre, `[a-c][0-2]` qui remplace a0 b0 c0 a1 b1 c1 a2 b2 c2,

Les raccourcis

Le motif `{fourch,brou}ette` est remplacé par `fourchette` et `brouette` indépendamment de l'existence ou nom de chemins correspondants.
Le motif ~ a déjà été vu et est remplacé par le chemin absolu du répertoire personnel de l'utilisateur courant. ~`user` est remplacé de la même façon mais pour l'utilisateur *user*.

Exercices

Copie et déplacement

Q39 Quelle commande permet la création "simultanée" de trois répertoires `GIF` et `Photos/Portugal`, `Photos/Marseille` et `Photos/Montagne` ?

Q40 Quelle commande permet de *déplacer* depuis le répertoire `images` tous les fichiers présentant l'extension `gif` dans le répertoire `GIF` nouvellement créé ?

Q41 Quelle commande permet de *copier* depuis le répertoire `images` tous les fichiers présentant l'extension `jpg` dans le répertoire `Photos` nouvellement créés ?

Q42 Définissez le répertoire `Photos/Montagne` comme votre répertoire courant. Quelle commande permet de déplacer la photo de chalet dans ce répertoire ?

Q43 En vous mettant dans `Photos`, déplacez les photos restantes dans le bon répertoire (Marseille est supérieure à 2000). Si possible, faites usages de jokers.

Exercices

Suppressions

Q44 Quel est le résultat de la séquence de commandes suivante :

```
cd ..
rm images
```

Q45 Comment modifier la dernière commande pour supprimer le répertoire `images/` ? Comment modifier la commande pour éviter les invites de confirmation ?

Q46 Quelle commande permet de copier le répertoire `GIF` et son contenu dans un répertoire nommé `images_GIF` ?

Q47 Quelle est la différence entre les deux commandes suivantes :

```
cd ~
cd /home/usager/votre_identifiant/
```

Q48 Fabriquez une archive qui contient le répertoire Photos (et uniquement celui-ci). Vérifiez son contenu.

Les partitions sous Linux

L'arbre unique

- Sous Linux, les partitions sont toutes dans regroupés dans une seule arborescence.

Les partitions qui ne sont pas la racine sont accrochées dans la partition racine (ou une autre déjà accrochée) au niveau d'un répertoire qui sert de *point de montage*.

Le contenu du point de montage est alors inaccessible et remplacé par le contenu du système de fichier qui a été monté

- Le chemin absolu d'un élément du système monté est le chemin du point de montage suivi du chemin dans le système de fichiers monté.

Exemple : fichier `moi/toto.txt` dans un système monté sur `/home`, le chemin absolu est `/home/moi/toto.txt`.

Le pseudo-système `/dev`

Sous Linux, les périphériques sont accessibles par une interface de type fichier. Leur chemin est `/dev/codeperipherique`.
Le sous-arbre à partir de `/dev` est un système de fichiers indépendant d'un périphérique physique. On parle de *système de fichiers virtuel*.

Le partitionnement

Du disque aux partitions

- Un disque est souvent divisé en plusieurs zones d'usage distinct (par exemple, système et données utilisateurs).
- Chacun de ces zones est appelée une *partition*. Elle est un système de fichiers indépendant des autres, et peut être combinée avec d'autres.
- Sous Windows, chaque partition est désignée par une lettre en fonction de son ordre de découverte par le système. Cette lettre fait partie du chemin.

L'ordre des partitions peut changer et donc la lettre ; ça pose problème pour les mises à jour.

Montage et démontage

- Un système d'exploitation peut rendre accessible une partition : c'est le *montage* de la partition.
- Inversement : c'est le *démontage* de la partition.

Une partition montée peut être utilisée normalement par les programmes.

Une partition démontée doit utiliser une interface spéciale plus compliquée qui contourne le *système de fichiers* et permet d'accéder directement au disque.

Syntaxe pour mount

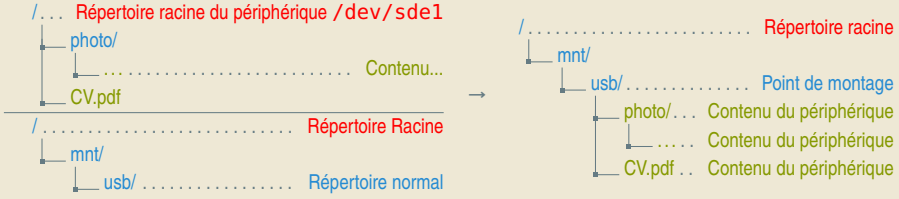
```
mount [[périphérique] point_de_montage]
```

Description

- `périphérique` correspond à périphérique (`/dev/xxx`). Il y a plusieurs syntaxes possibles.
- `point_de_montage` correspond à un nom de répertoire valide dans l'arborescence principale donnant accès au contenu de l'arborescence du périphérique.
- Sans argument, la commande liste tous les montages en cours

Exemple d'utilisation:

```
login@host:/home$ mount /dev/sde1 /mnt/usb
```



Exercices

Analyse de périphériques

Q49 Le périphérique `zero` est un périphérique virtuel. La commande `dd if=/dev/zero count=1 | hexdump -v` permet de voir les 512 premiers octets de ce périphérique virtuel (en hexadécimal). Regardez-le. Qu'est-ce qu'il a de particulier ?

Q50 Le périphérique `urandom` est un périphérique virtuel. La commande `dd if=/dev/urandom count=1 | hexdump -v` permet de voir les 512 premiers octets de ce périphérique virtuel (en hexadécimal). Regardez-le. Recommencez. Qu'est-ce qu'il a de particulier ?

Q51 Le périphérique `sda1` est une des partitions du disque dur. La commande `dd if=/dev/sda1 count=1 | hexdump -v` permet de voir les 512 premiers octets de ce périphérique virtuel (en hexadécimal). Regardez-le. Que se passe-t-il ? Un programme normal comme `dd` peut-il examiner le disque dur en outrepassant le système de fichiers ?

Q52 En utilisant la commande `mount`, analysez les différentes partitions présentes dans votre système. Identifiez celles qui correspondent à un vrai périphérique et les systèmes de fichier virtuel.

Espace libre

Une partition occupe une taille fixe. La plupart des systèmes de fichiers sont **de taille fixe**. Elles peuvent accueillir uniquement une certaine quantité de données.

L'espace de travail

Comme dans un parking, la quantité de données que l'on peut mettre dans un disque ne doit pas être égale à la quantité de données qu'il peut accueillir, sinon, on ne peut pas faire un certain nombre d'opérations. En plus de l'espace réservé à la signalisation (index et tables divers), on réserve aussi un peu d'espace pour les programmes importants du système.

La fragmentation

Les fichiers sont posés par petits blocs dans la partition (qui est elle-même un gros bloc dans l'ensemble du disque). Parce qu'un fichier est plus rapide à lire si les blocs sont les uns à côté des autres, les systèmes de fichiers essayent de maintenir cet état. Sous Windows, on peut aider le système en procédant à une opération de rangement : la *défragmentation*.
Les systèmes de fichiers utilisés sous Linux n'ont quasiment pas de fragmentation si on utilise moins de 95% de leur espace.

Syntaxe pour df

`df [-h] [emplacement]`

Description

- Affiche les disques montés et leur capacité de mémoire (celui de la partition qui correspond à l'emplacement, tous s'il n'y en a pas).
- L'option `-h` (human readable) convertie l'affichage des tailles mémoires en unités conventionnelles binaires (en nombre de blocs par défaut). Avec `-H` c'est en unités décimales.

Exemple d'utilisation:

```
login@host:~$ df -h
 Sys. de fichiers     Taille   Uti.   Disp.   Uti%   Monté sur
 /dev/sda1            56G      16G    37G     31%    /
 myserver:/home/moi   1,8T     1,6T   192G    90%    /users/moi
 ...                  ...      ...    ...     ...    ...
login@host:~$
```


Exercices

Partitions et espace disque

Q53 Analysez l'espace disque disponible et utilisé sur toutes les partitions de votre ordinateur. Comparez les unités décimales et binaires.

Q54 En utilisant `df /` et `df /etc`, vérifiez que ces deux répertoires sont bien sur la même partition. Comparez avec `df ~` et `df /boot`.

Q55 Si un utilisateur remplit complètement la partition qui abritent ses données, qu'est-ce qui cesse de fonctionner ? Qu'est-ce qui peut continuer à fonctionner ?

Q56 L'administrateur `root` a pour répertoire personnel `/root`. En regardant sur quelle partition c'est, expliquez pourquoi ce n'est pas dans `/home`.

Les nœuds d'index

Les nœuds d'index ou inodes

- Les données des fichiers sont stockées dans des blocs numérotés.

L'organisation des fichiers et répertoires est elle stockée dans des blocs spéciaux appelés nœuds d'index (ou *inodes*). Un chemin est associé à un inode unique qui va contenir la liste des numéros de blocs de données qu'il utilise.

- Un répertoire est un inode qui pointe vers un bloc de données qui contient un tableau de type nom → inode
- Un fichier est un inode qui pointe vers un ou plusieurs blocs de données qui contiennent... les données.

L'inode, c'est le représentant du fichier. Il contient aussi les *métadonnées* associées.

Lire la structure des inodes

Les inodes sont simplement désignés par un numéro. La commande `ls -i` ou `stat` permet d'accéder à cette information.

Syntaxe pour stat

```
stat  chemin [chemin_2 ...]
```

Description

- Si le chemin est occupé par un fichier ou un répertoire, affiche les métadonnées relatives au chemin.

Exemple d'utilisation:

```
login@host:~$ stat /etc/fstab
Fichier : « /etc/fstab » Taille : 672 Blocs : 8
Blocs d'E/S : 4096 fichier Périphérique : 801h/2049d
Inœud : 5244539 Liens : 1 Accès : (0644/-rw-r-r-)
UID : ( 0/ root) GID : ( 0/ root) Accès : 2015-09-22
22:04:47.768711256 +0200 Modif. : 2013-11-13
09:12:51.763972183 +0100 Changt : 2015-08-09
10:57:36.553313285 +0200 Créé : -
login@host:~$ ■
```


Exercices

Découvrir les métadonnées

Q57 En utilisant la commande `stat ~`, trouvez le numéro d'inode de votre répertoire personnel.

Q58 Quelles autres sortes de métadonnées arrivez-vous à comprendre ?

Q59 Avec la commande `ls -ial ~`, regardez les numéros d'inodes de vos répertoires. Sont-ils différents ? Comment l'expliquez-vous ?

Q60 Maintenant, regardez le numéro d'inode avec `stat` du répertoire `/home/usager`. Est-ce que vous le reconnaissez ?

Q61 Regardez avec ces commandes les valeurs des numéros d'inode de `/`, `/.` et `/..`. Expliquez.

Q62 Recommencez avec `stat /home` et `stat /`. Que remarquez-vous à propos de ces numéros d'inodes ? Est-ce que ces répertoires sont identiques ? Comme l'expliquer ?

Q63 Dans le répertoire `~/Documents/m1101/textes`, il y a un fichier texte. Regardez ses métadonnées. Dans une autre fenêtre, lisez-le. Puis regardez encore. Est-ce que quelque chose a changé ?

Q64 Faites une copie de ce fichier. Modifiez la copie avec `gedit copie.txt`. Vérifiez que les métadonnées changent encore. Lesquelles ? Est-ce qu'il y a une commande qui permet de faire ce changement de métadonnées sans vraiment changer le fichier ? (après, supprimez la copie).

Les liens durs

Un fichier, deux chemins (et plus si affinités)

Vu la structure utilisée, il est possible de mettre le même numéro d'inode dans deux répertoires différents (même nom ou pas) ou sous deux noms dans le même répertoire. On obtient ainsi deux chemins qui pointent vers le même fichier. C'est ce qu'on appelle un *lien* ou *lien dur* (hardlink).
Lorsqu'on édite le premier « fichier », le deuxième est aussitôt modifié (normal, c'est le même).
Lorsqu'on supprime l'un des deux fichiers, l'autre reste. Pour savoir quand effacer vraiment le fichier, il utiliser le compteur de liens (lorsqu'il est à zéro, on peut effacer).
On ne peut pas faire de liens durs entre partitions différentes.q

Boîte à outils : les partitions

Commande	Fonction principale
`mount`	Manipuler les partitions
`df`	Afficher l'espace restant
`ls`	Afficher le contenu d'un répertoire
`stat`	Afficher les métadonnées d'un chemin
`ln`	Créer un lien (dur ou symbolique)

Syntaxe pour ln

```
ln  source [cible]
```

Description

- Crée un lien dur entre la référence source et le chemin cible.

Exemple d'utilisation:

```
login@host:~$ ln astronomie.txt Etoiles/lisezmoi.txt
```

Le lien sur un fichier crée une deuxième entrée pointant vers le même inode.

```
moi/ ..... Répertoire courant
├── astronomie.txt ..... Référence source
├── Etoiles/
│   └── soleil.jpg
└── aldebaran.gif
```

```
moi/ ..... Répertoire courant
├── astronomie.txt
├── Etoiles/
│   ├── soleil.jpg
│   └── lisezmoi.txt ..... Nouveau chemin
└── aldebaran.gif
```


Exercices

Liens durs

Q65 Dans votre répertoire `~/Documents/m1101/textes`, créez un petit fichier `source.txt`. Insérez quelques lignes de texte dedans (avec `gedit source.txt`), puis quittez l'éditeur de texte.

Q66 Vérifiez dans la console son numéro d'inode, et son contenu. Quelles sont les commandes pour cela ?

Q67 Faites une copie de `source.txt` vers `copie.txt`, et un lien de `source.txt` vers `lien.txt`. Vérifiez le contenu de la copie et du lien. Vérifiez que les numéros d'inode et les compteurs de liens sont comme ce à quoi vous vous attendiez.

Q68 Avec l'éditeur de texte, comme plus haut, modifiez le fichier source. Regardez les trois fichiers dans le terminal. Est-ce conforme à vos attentes ? Essayez ensuite en changeant la copie.

Q69 Faites un lien de `lien.txt` vers `lienlien.txt`. Vérifiez le compteur de liens. Effacez ensuite `lien.txt`, et vérifiez encore.

Q70 Modifiez `lienlien.txt`, puis regardez tous les contenus. Effacez le fichier `source.txt`. Le fichier `lienlien.txt` est-il toujours là ?

Q71 Essayez de faire un lien entre `copie.txt` et `/tmp/copie.txt`. Que se passe-t-il ? Pourquoi ? Expliquez.

Les liens symboliques

Une redirection

Sous Windows ou sous Unix, on peut créer des « raccourcis » qui lient un chemin spécifique à un autre endroit dans l'arborescence.

- Unix les appelle des liens symboliques.
- Windows les appelle des raccourcis.
- Un lien symbolique est un chemin (relatif ou absolu) qui indique un autre point de l'arbre. Il fonctionne au niveau chemin et pas au niveau inode.
- Un lien symbolique peut traverser les partitions.

Un lien symbolique peut pointer sur un chemin qui ne correspond pas à un fichier ou un répertoire. On dit qu'il est brisé.

Exercices

Liens symboliques

Q72 Dans votre répertoire `~/Documents/m1101/textes`, recréez un petit fichier `source.txt`, effacez `lienlien.txt` et `copie.txt`.

Q73 Faites un lien symbolique de `source.txt` vers `lien.txt`. Vérifiez le contenu du lien. Regardez les métadonnées associées.

Q74 Avec l'éditeur de texte, comme plus haut, modifiez le fichier source. Regardez les deux fichiers dans le terminal. Est-ce conforme à vos attentes ?

Q75 Faites un lien de `lien.txt` vers `lienlien.txt`. Vérifiez le contenu des trois fichiers (en modifiant).

Q76 Modifiez `lienlien.txt`, puis regardez tous les contenus. Effacez le fichier `source.txt`. Le fichier `lienlien.txt` est-il toujours là ?

Q77 Essayez de faire un lien entre `commande.txt` et `/tmp/story.txt`. Que se passe-t-il ? Pourquoi ? Expliquez.

Q78 Faites un lien symbolique vers un répertoire. Que se passe-t-il ? Quel est le danger ?

L'archivage et les liens

Les formats zip et tar

- Le format `tar` permet d'archiver autant les liens durs que les liens symboliques
- Le format `zip` ne permet que d'archiver les liens symboliques
- Le programme `zip` remplace, par défaut, les liens symboliques par des copies.
- L'option `-symlinks` permet de conserver les liens symboliques dans les archives `zip`.
- Le programme `tar` a une option `-dereference` qui transforme les liens symboliques en copies. Les liens durs ne sont archivés qu'une seule fois par défaut.

Un lien symbolique archivé en tant que tel pointant en dehors de l'archive peut être brisé !

Exemple (Création d'archives)

```
login@host:~$ zip --symlinks -r sel.zip sel/
adding: selection/ (stored 0%)
adding: selection/best.jpg (stored 0%)
adding: selection/img_1363.jpg (deflated 2%)
adding: selection/img_1221.jpg (deflated 1%)
login@host:~$ unzip sel.zip
```


Exercices

Liens et archives

Q79 Dans votre répertoire `~/Documents/m1101`, créez un répertoire `selection`.

Q80 Faites un lien symbolique dans `selection` d'une image de `images`, un lien dur et une copie de fichier. Rajoutez un lien dur dans `selection` vers la copie de fichier sous un autre nom (par exemple `lameilleure.jpg`.

Q81 Archivez le répertoire `selection` au format `tar`. Vérifiez avec `tar vvf selection.tar` que les fichiers sont tous présents, sauf le lien symbolique.

Q82 Archivez avec `zip` le même répertoire. Vérifiez (avec `stat sel.zip`) que la taille de l'archive est cohérente avec la présence de quatre images (et non pas trois).

Q83 Archivez dans un autre fichier `zip` le même répertoire avec la conservation des liens symboliques. Comparez les tailles et expliquez.

Q84 Créez trois répertoires `d1`, `d2`, `d3`. Dans chacun de ces répertoires, décompressez les archives crées précédemment. Regardez ce qui arrive aux liens symboliques et aux liens durs dans chacun des cas.

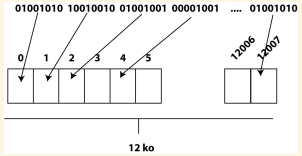
Fichiers exécutables et Processus

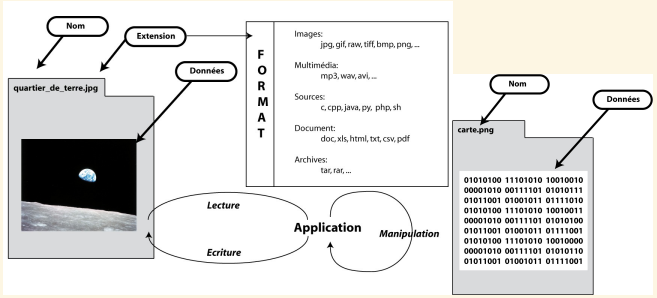
Fichier binaire et fichier texte
Processus dans un système multitâches et mutli-utilisateurs
Gestion de la mémoire vive
Gestion de l'accès au CPU
Processus en ligne de commande

Fichier binaire et fichier texte

Les données numériques

Tout fichier enregistré sur un support numérique est une suite d'octets.





Accès aux données

Lors de son utilisation un fichier est *lu* par un programme. Pour cela il doit décoder les informations binaires et les traiter.

Fichier binaire et fichier texte

Deux grands types de fichiers : Binaire Vs Numérique

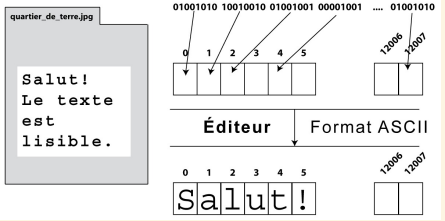
De façon générale un fichier binaire ne peut être *"lu"* que par un programme informatique, alors qu'un fichier texte peut être *"lu"* par être humain.

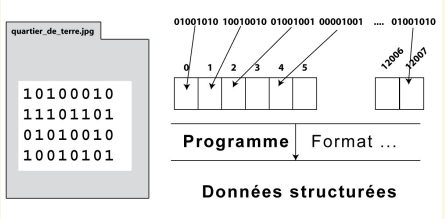
Les fichiers textes

C'est un fichier qui peut être *"lu"* par un éditeur de texte brut. Les données sont encodées comme une suite de caractères.

Les fichiers binaires

Ce n'est pas un fichier texte …Il peut contenir des instructions machines, des données compressées, des données binaires brutes nécessitant un programme pour être lues.





Fichiers sources →Exécutable →Processus

Les sources : Une *"recette de cuisine"*

- Exprime un ensemble de tâches à réaliser pour accomplir le programme (le plat cuisiné).
- Utilise un langage de programmation.
- C'est un fichier texte.

dessine.c

```
(...)
float r, x, y;
r=3.0;
x=0.0;
y=7.1;
cercle(0,0,,r)
segment(0,0,x,y)
```

L'exécutable

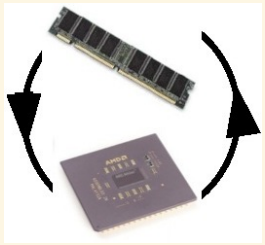
- Exprime les mêmes tâches dans un langage machine.
- Ce fichier ne fonctionne que sur des ordinateurs qui ont la même architecture.
- C'est un fichier binaire.

dessine

```
10100101 11101001
10001001 00100101
00101010 00100010
01111011 10110101
01000010 00110011
00101101 11010100
(...)
```

Les processus

- L'évaluation des instructions machines engendre des processus.
- Ces processus sont exécutés par le matériel.
- Les instructions machine doivent donc être adaptées au matériel.



Exercices

Préparation

85. Vérifiez que votre répertoire courant est bien `m1101`. Analysez l'affichage produit par la commande ls suivie des options `-lh`. Vous pourrez comparer les affichages obtenus par les commandes `ls -l` et `ls -lh` pour comprendre l'effet de l'option `-h`. Vous pourrez aussi rechercher cette information dans les pages de man.
86. Analysez l'arborescence créée lors de l'extraction des données de l'archive au moyen de la commande `ls`. Vous dessinerez cette arborescence.
87. Après vous être placé dans le répertoire créé lors de l'extraction de l'archive (donnees), quelle commande permet d'identifier le plus gros fichier (taille mémoire). Identifiez-le.
88. Quelles commandes vous permettent d'afficher le contenu des fichiers texte `command.txt` et README ? Quels sont leurs contenus ?
89. Analysez le résulat de l'évaluation des commandes suivantes :

```
file textes/README.txt
file textes/command.txt
file images/img_1175.jpg
```

90. Quelle est la fonction de la commande `file` ? Parcourez les pages de manuel de cette commande.

Identification des processus par le système d'exploitation

Système multi-utilisateur

- Plusieurs utilisateurs partagent les mêmes ressources matériel (RAM, CPU, disques, …),
- Chaque utilisateur lance des processus liés à ses activités sur la machine et il utilise les résultats de ces processus.

Système multi-tâches

- Plusieurs programmes en cours d'exécution partagent les mêmes ressources matériel (mémoire vive, CPU, disques, …). Ils peuvent provenir d'un seul ou de plusieurs utilisateurs,
- Chaque programmes lance des processus et il utilise les résultats de ces processus.

Il faut partager les ressources ! ! !

- Chaque programme doit être exécuté éventuellement *"en même temps"*. Il faut donc gérer le partage des ressources de calcul (accès à la mémoire vive, au CPU),
- Chaque programme ou utilisateur doit pouvoir retrouver les résultats de ses calculs. Il faut donc pouvoir identifier qui a lancé les processus et qui doit récupérer les résultats.

La gestion des processus est réalisée par le système d'exploitation. C'est une de ses tâches principales. Pour cela il a besoin de pouvoir identifier chaque processus.

PID et PPID

PID - **P**rocess **ID**entifier

- C'est un numéro unique attribué à chaque processus lors de son lancement.
- Il permet d'identifier de façon unique chaque processus.
- La liste des processus en cours d'exécution est accessible en ligne de commande par les commandes `ps` et `top`.

PPID - **P**arent **P**rocess **ID**entifier

- Le premier processus lancé porte le numéro de PID 1. Les processus suivants sont des processus issus de ce processus parent.
- Chaque processus est lancé par un processus parent *via* l'appel système `fork`.
- Le PPID est le PID du processus Parent.

Utilités

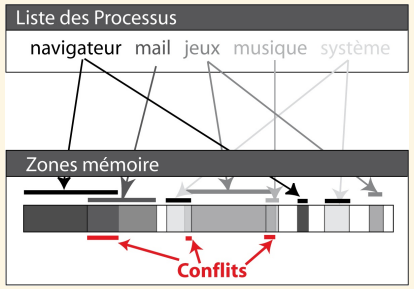
- L'utilisateur peut suivre un processus, le suspendre temporairement, le relancer ou le tuer (interruption définitive).
- Le système s'en sert pour lui affecter des ressources matériel.

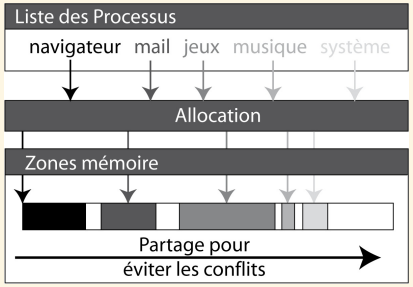
Exercices

Racourcis clavier et astuces en ligne de commande

91. Tapez les 2 caractères `sl` puis pressez la touche ⇆ (Tab). Que se passe-t-il ?
92. Tapez les 3 caractères `sle` puis pressez la touche ⇆. Que se passe-t-il ?
93. À la suite de l'affichage précédent tapez la combinaison de touches Ctrl A. Que se passe-t-il ?
94. Que fait la commande `man sleep` ? Que pouvez-vous dire de la commande `sleep` ?
95. Exécutez la commande `sleep 32000000`. Que se passe-t-il si vous tapez la combinaison de touches Ctrl C ?
96. Quelle action produit la pression de la flèche ↑ sur votre clavier ?
97. Quelle est l'action produite par la pression de la combinaison de touches Ctrl U après avoir tapé quelques lettres ? Par la combinaison de touche Ctrl L ?
98. Quelle est l'action produite en tapant ls ␣ ⇆ (le caractère ␣ signifie la présence d'un espace) ?

Gestion de la mémoire vive





Chaque processus a besoin de mémoire

Pour stocker et travailler sur :

- les données,
- les instructions,
- les résultats.

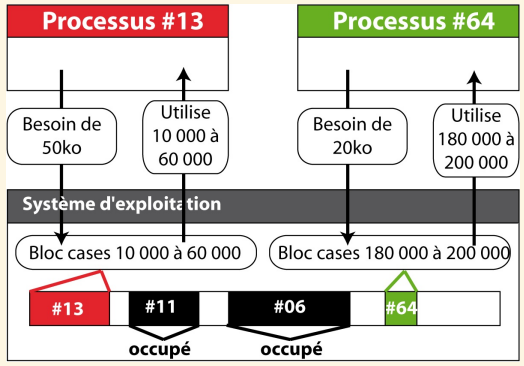
Il faut assurer l'intégrité des données !

Allocation de zone mémoire

L'allocation permet :

- d'attribuer à chaque processus un espace de travail en mémoire,
- le système contraint le programme à écrire dans sa zone mémoire et ainsi,
- évite qu'un programme modifie les données d'un autre programme.

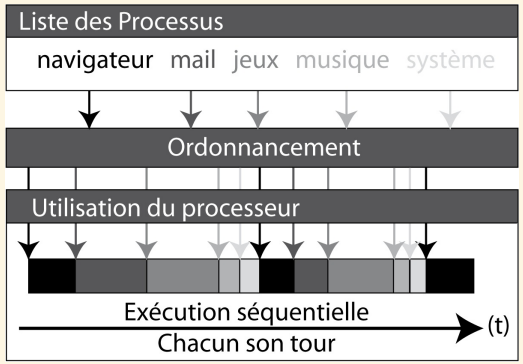
Gestion de la mémoire vive



Principes généraux de l'allocation

- L'OS maintient une table des zones mémoires allouées à chaque processus. Ces zones sont réservées et ne peuvent être utilisées que par le processus parent.
- Lorsqu'il a besoin de mémoire, un processus demande à l'OS quelle zone il peut utiliser,
- L'OS lui attribue, en fonction de l'espace libre, un certain nombre de blocs mémoire.
- Les blocs mémoire attribués sont alors réservés.

Gestion de l'accès au CPU



Le planificateur gère le temps CPU attribué à chaque processus

- Le CPU ne traite qu'un seul processus à la fois,
- Le planificateur permet l'alternance d'accès au CPU en attribuant une priorité à chaque processus.
- L'illusion d'exécution simultanée de plusieurs processus est donnée par une alternance rapide d'attribution de temps de calcul à chaque processus.

Syntaxe pour ps

`ps <-eu>`

Description

- Affiche les processus en cours d'exécution.
- L'option <-e> indique que tous les processus doivent être affichés,
- L'option <-u> restreint l'affichage aux processus de l'utilisateur.

Exemple d'utilisation:

```
login@host:~$ ps -eu
  Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net
    USER  PID %CPU %MEM  VSZ  RSS   TTY STAT START  TIME COMMAND
santini 5905  0.0  0.2 4824 1656 pts/1   Ss 09:27 0:00 -bash LC_ALL=fr_FR.UTF
santini 5962  0.0  0.1 3884  896 pts/1   R+ 09:48 0:00 ps -eu MANPATH=/etc/jav
login@host:~$
```


Syntaxe pour top

`top`

Description

- Permet de suivre dynamiquement (temps réel) les ressources matériel utilisées par chaque processus.
- Ouvre un interface dans la ligne de commande qui peut être quittée en pressant la touche Q
- Donne pour chaque processus en autres choses, le PID, le nom du propriétaire, la date de lancement du processus, les %CPU et %MEM utilisés.

Exemple d'utilisation:

```
  Tasks: 85 total, 1 running, 84 sleeping, 0 stopped, 0 zombie
  Cpu(s): 5.7%us, 0.0%sy, 0.0%ni, 93.6%id, 0.0%wa, 0.7%hi, 0.0%si, 0.0%st
  Mem: 772068k total, 231864k used, 540204k free, 2412k buffers
  Swap: 995992k total, 0k used, 995992k free, 161316k cached

 PID    USER PR NI  VIRT  RES  SHR S %CPU %MEM   TIME+             COMMAND
5116    root 20  0 33832  22m 6576 S  5.7  3.0 0:19.49                 X
5879 santini 20  0 16060 7344 6116 S  0.3  1.0 0:01.06     xfce4-netload-p
   1    root 20  0  1664  568  496 S  0.0  0.1 0:02.95               init
   2    root 20  0     0    0    0 S  0.0  0.0 0:00.00           kthreadd
   3    root RT  0     0    0    0 S  0.0  0.0 0:00.00        migration/0
```


Processus en ligne de commande

Occupation de la ligne de commande

- Lorsque l'on tape une commande, la ligne de commande est bloquée (plus de prompt) jusqu'à la fin de l'exécution.
- La ligne de commande est à nouveau disponible ensuite.

```
login@host:~$ sleep 20
(il faut attendre 20 secondes avant
l'apparition du nouveau prompt)
...
...
login@host:~$
```

```
login@host:~$ gedit
(Il faut quitter l'application ou tuer
le processus gedit pour avoir un nouveau
prompt)
...
...
```

Libération de la ligne de commande

Deux façons possibles de lancer une instruction en tâche de fond :

Lancement en tâche de fond

- Les commandes qui prennent beaucoup de temps peuvent être lancées en tâche de fond pour libérer la ligne de commande du shell.
- Pour lancer directement la commande en tâche de fond il suffit de faire suivre la commande du caractère &. On retrouve immédiatement un nouveau prompt.

```
login@host:~$ gedit &
login@host:~$ ■
```

Relégation en tâche de fond

- Si une tâche déjà lancée occupe la ligne de commande, il est possible de suspendre son exécution en pressant la combinaison de touches Ctrl Z. La tâche est alors interrompue et on retrouve un nouveau prompt.
- Il est possible de relancer le processus en tâche de fond au moyen de la commande `bg`.

```
login@host:~$ gedit
^Z
[1]$+ Stopped gedit
login@host:~$ bg
[1]$+ gedit &
login@host:~$ ■
```


Exercices

Gestion des processus

Afin d'illustrer la gestion des processus nous allons utiliser la commande `sleep` pour simuler l'exécution de programmes dont l'exécution n'est pas immédiate. Pour se rappeler de son fonctionnement vous pouvez utiliser la commande man.

91. Évaluez l'instruction `sleep 1000` puis tapez Ctrl C. Que se passe-t-il ?
92. Évaluez l'instruction `sleep 1000 &` (n'oubliez pas le caractère &). Que se passe-t-il ?
93. La commande ps permet d'afficher la liste de processus qui s'exécutent sur votre ordinateur. Un processus s'exécutant sous Linux est identifié par un numéro de processus, et par un propriétaire (celui qui a lancé le processus). Identifiez ces deux données lors de l'appel des commandes suivantes, donnez un explication à la différence des affichages (utilisez le man si nécessaire) :

    ```
    ps
    ps -ef
    ```

94. Quel est le numéro de processus associé à la commande `sleep 1000 &` ?

Exercices

Gestion des processus (suite)

95. La commande kill permet de « tuer » (supprimer) un processus. Sa syntaxe d'utilisation est la suivante : `kill PID` où PID (Process ID) doit être remplacé par le numéro du processus à supprimer.
96. Quelle commande permet de détruire le processus associé à la commande `sleep 1000 &` ?
97. Tapez la commande `gedit` dans le terminal. Quel est l'effet sur la ligne de commande ? Pouvez-vous saisir de nouvelle commandes ?
98. Après avoir lancé `gedit` (celui-ci étant en cours d'exécution), que se passe-t-il si on tape Ctrl Z dans le terminal qui a lancé `gedit` ? Quel est l'effet sur le programme gedit (utilisez `ps` pour suivre l'état des processus) ? Que se passe-t-il si vous tapez `bg` ?
99. Que fait la commande `top` ?
100. Exécutez la command `ps -ef f`. Examinez comment est construite la *forêt* de processus. Repérez comment sont agencés les processus qui gèrent vos terminaux entre eux.

Compléments sur l'arborescence

Droits sur les fichiers
Arborescence du système Linux
Interprétation ou Compilation
Exécution des commandes
Chemins par défaut et variable d'environnement
Configuration des variables d'environnement

Propriété des fichiers

Identifications des utilisateurs dans un environnement multi-utilisateurs

UID (**U**ser **ID**entifier) numéro unique associé à chaque utilisateur lors de la création de son compte.

GID (**G**roup **ID**entifier) numéro unique d'un groupe d'utilisateurs. Chaque utilisateur peut être associé à un ou plusieurs groupes.

Utilité

- Chaque fichier (ou répertoire) et chaque processus du système est associé à un utilisateur : cet utilisateur est le propriétaire du fichier (ou répertoire) ou celui qui a lancé le processus.
- Être propriétaire d'un fichier ou d'un processus confère des droits sur ceux-ci.

Connaitre l'identité du propriétaire d'un processus ou d'un fichier

- Les commandes `top` et `ps` affichent le nom du propriétaire des processus.
- La commande `ls` avec l'option `-l` affiche le nom et le groupe du propriétaire d'un fichier ou d'un répertoire.
- Les UID et GID sont enregistrés dans le fichier d'administration `/etc/passwd` ou d'autres mécanismes

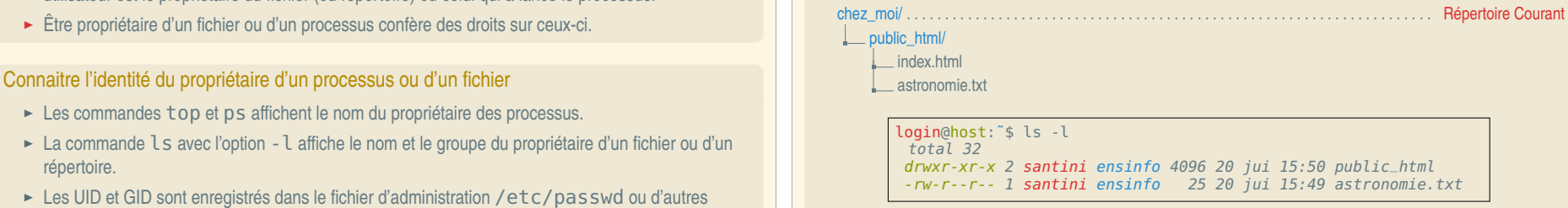
Syntaxe pour ls(ter)

```
ls -l <source>
```

Description

- Affiche le contenu d'un répertoire en format long.
- Le format long donne le nom du propriétaire et son groupe, ainsi que les droits des différentes classes d'utilisateurs sur les fichiers et répertoires.

Exemple d'utilisation:




```
login@host:~$ ls -l
 total 32
 drwxr-xr-x 2 santini ensinfo 4096 20 jui 15:50 public_html
 -rw-r--r-- 1 santini ensinfo   25 20 jui 15:49 astronomie.txt
```

Ici, le nom de l'utilisateur est `santini`, nom du groupe est `ensinfo` et les droits sont colorés en vert.

Les droits sur les fichiers et répertoires

3 catégories d'utilisateurs

-	r w x	r w x	r w x
Type de Fichier	Droits du propriétaire (**U**ser)	Droits du groupe (**G**roup)	Droits des autres (**O**ther)

Types de fichiers

	Types
-	Fichier ordinaire
d	Répertoire
l	lien symbolique

Droits/Permissions

		Fichier	Répertoire
r	(**R**ead)	lire	lister le contenu
w	(**W**rite)	écrire et modifier	modifier le contenu
x	(e**X**ecute)	exécution	traverser

Types d'utilisateurs

		Cible
u	(**U**)ser	Propriétaire du fichier/répertoire
g	(**G**)roup	Membre du même groupe que le propriétaire
o	(**O**)ther	Tous les autres
a	(**A**)ll	Tous les utilisateurs (réunion de `'u'` `'g'` et `'o'`)

Syntaxe pour chmod

```
chmod  droit fichier
```

Description

- Modifie les droits et permissions accordés par le propriétaire aux différents utilisateurs du système.

Exemple d'utilisation:

Retire au propriétaire le droit d'écriture sur le fichier `cv_2011.pdf`.

```
login@host:~$ chmod u-w cv_2011.pdf
```

Ajoute au propriétaire et aux membres de son groupe le droit d'exécution sur le fichier `listing.bash`.

```
login@host:~$ chmod ug+x
listing.bash
```

Retire aux utilisateurs qui ne sont ni le propriétaire ni membre de son groupe les droits de lecture, d'écriture et d'exécution.

```
login@host:~$ chmod o-rwx
listing.bash
```

Ajoute à tous les utilisateurs, tous les droits.

```
login@host:~$ chmod a+rwx
listing.bash
```

Description

Il existe plusieurs notations des droits.

- La notation alphanumérique : `(ugoa) (+/-) (rwx)`
- La notation octale
  - Calcul des droits pour un utilisateur (u, g ou o) :

Droit	`---`	`--x`	`-w-`	`-wx`	`r--`	`r-x`	`rw-`	`rwx`
Binaire	000	001	010	011	100	101	110	111
Octale	0	1	2	3	4	5	6	7

  - Exemple de notation octale des droits d'un fichier

	User			Group			Other		
Alphabétique	r	w	x	r	-	x	-	-	x
Binaire	1	1	1	1	0	1	0	0	1
Octale	7			5			1		

Exemple d'utilisation:

Alph.	Oct.
`--- --- ---`	000
`rw- --- ---`	600
`rw- r-- r--`	644
`rw- rw- rw-`	666

Alph.	Oct.
`rwx --- ---`	700
`rwx r-x r-x`	755
`rwx rwx rwx`	777

```
login@host:~$ chmod 700 dir_parano
login@host:~$ chmod 644 fichier_pub
```


Exercices

Identification et droits

Q101 Au moyen de la commande `id`, affichez votre UID et votre GID ? Comparez-le avec celui de votre voisin de table. Qu'en concluez-vous ? Comparez-les avec celui de l'utilisateur `root`. Qu'en concluez-vous ?

Q102 Quels sont vos droits sur le répertoire racine `/`, `root`, `/tmp`, sur votre répertoire `~/`, et celui de votre voisin de table `~/../login_voisin`.

Q103 Pouvez-vous lire les données contenue dans le répertoire de votre voisin. Quelle commande permettrait de le faire ? Qui doit lancer la commande ?

Q104 Donnez les commandes octale et alphanumérique de changement de droits permettant :

- d'autoriser aux membres de votre groupe et aux "autres" l'accès en lecture aux images du répertoire `donnees_tdtp2/images`.
- de donnez les droits d'écriture aux membres de votre groupe sur le fichier `donnees_tdtp2/command_line.txt`
- de vous (le propriétaire) retirer toute possibilité de supprimer le fichier `donnees_tdtp2/0readme`

Q105 Imaginez comment donner à votre voisin un accès sous votre répertoire personnel à un répertoire dans lequel il aurait les droits d'écriture sur un fichier spécifique, que vous ne pourriez vous que lire (mais pas modifier). Il ne doit pas pouvoir créer un autre fichier chez vous. Comment faites vous pour effacer ce fichier ?

Exercices

Remise en état

Après toutes les modifications pouvant impliquer votre répertoire personnel, n'oubliez pas `chmod 711` pour remettre les modes de votre répertoire à leur état d'origine.

Les principaux répertoires et leur contenu

Une structure plus ou moins normalisée

- Les fichiers nécessaires au fonctionnement du système sont organisés en arborescence,
- Cette arborescence est commune à presque toutes les distribution linux,
- Cette organisation rationalisée facilite l'installation de nouveaux programmes qui savent où trouver les fichiers dont ils peuvent avoir besoin.

Une organisation qui permet un cloisonnement

- Les fichiers et les répertoires systèmes sont protégés par des restrictions de droits,
- De nombreux fichiers ne peuvent être modifiés par un utilisateur « normal »,
- Seul l'utilisateur `root`, ou les utilisateur faisant partie du groupe `admin` peuvent avoir la permission de modifier certains fichiers.
- Il s'agit d'une protection. Pour réaliser une action susceptible d'affecter le comportement du système il faut montrer "patte blanche" et prendre conscience de ce que l'on fait. Entrer le mot de passe `root` doit être un signal d'alerte.

Les principaux répertoires et leur contenu

Répertoire	Contenu
/	Répertoire racine : Toutes les données accessibles par le système
/bin	Binaires exécutables des commandes de bases (cd, ls, mkdir, …)
/dev	Fichiers spéciaux correspondants aux périphériques
/etc	Fichiers de configuration (profile, passwd,fstab...)
/home	Les répertoires personnels des utilisateurs
/lib	Librairies partagées et modules du noyeau
/mnt	Points de montage des périphériques
/root	Répertoire personnel de l'administrateur
/tmp	Données temporaires
/usr	Ressources accessibles par les utilisateurs
/var	Fichiers de log ou fichiers changeant fréquemment

L'essentiel est synthétisé dans
`https://fr.wikipedia.org/wiki/Filesystem_Hierarchy_Standard`

Exercices

Hiérarchie du système

Astuce : si la sortie d'une commande est trop longue, on peut ajouter `|less` à la fin de la ligne pour l'afficher par morceaux. Ceci vous sera expliqué dans quelques séances...

Q106 Identifiez le propriétaire, le groupe et les différents droits des fichiers contenus dans le répertoire `/bin` ? Quels sont vos droits sur ces fichiers ?

Q107 Ces fichiers on le droit `x`. Que pouvez-vous en conclure ?

Q108 A votre avis, que se passe-t-il en fait lorsque vous saisissez une commande telle que `ls` ?

FHS

Q109 Identifiez, à l'aide de la FHS, la fonction de `/usr/include`. Confirmez votre hypothèse en regardant quelques fichiers.

Langages Compilés Vs Langages Interprétés

Caractéristiques des Langages Compilés

- L'ensemble du code source est compilé une seule fois avant l'exécution en instructions machine (contenues dans un fichier : exécutable).
- Le compilateur n'est pas nécessaire lors de l'exécution.
- Le compilateur est spécifique à la machine.
- L'exécutable (code compilé) est spécifique à la machine.

Inconvenients

- Il faut recompiler pour prendre en compte une modification du code.
- L'exécutable n'est pas portable sur d'autres machines.

Avantages

- Plus rapide (spécifique à la machine qui exécute les instructions).
- L'ensemble des instructions sont regroupées dans un seul fichier.

Exemples de langages CompilésInterprétés

- C, C++, ADA, Pascal, Fortran, Cobol,

Langages Compilés Vs Langages Interprétés

Caractéristiques des Langages Interprétes

- Les instructions du code source sont converties en instructions machine lors de l'exécution du programme
- L'interpréteur est nécessaire lors de l'exécution.
- L'interpréteur est spécifique à la machine,
- L'exécutable (le code source) n'est pas spécifique à la machine.

Inconvenients

- Moins rapide.
- Plusieurs fichiers (et librairies) servent à l'exécution.

Avantages

- Modifications du code source immédiatement prises en compte lors de la réexécution.
- Le code est portable sur d'autres machine

Exemples de langages CompilésInterprétés

- Java, Python, Bash, Lisp, PHP, Prolog, Perl, Javascript

Lancer un programme/une commande

Cas général

- Pour exécuter un programme il suffit saisir sur la ligne de commande le chemin menant au fichier contenant les instructions,
- Si le fichier présente la permission `"X"` pour exécutable, les instructions qu'il contient sont exécutées.

Script bash exécutable

- Un script `bash` est un fichier texte contenant des instructions `bash`
- La première ligne contient le chemin menant à l'exécutable de l'interpréter précédé des caractère `#!` (par exemple `#! /bin/bash`),
- La seconde ligne est souvent vide,
- Les lignes suivantes comportent des instructions.

test_bash.sh

```
#!/bin/bash

instruction 1;
instruction 2;
...
instruction N;
```


Exercices

Lancer un programme/une commande

Q110 Après avoir créé un répertoire `bin` dans votre répertoire personnel, définissez créez dans ce répertoire un script nommé `listintro.sh`. Ce script comporte une unique commande permettant de lister le contenu du répertoire de travail `Intro_Systeme` dans lequel vous avez l'habitude de travailler.

Q111 Attribuez les droits d'exécution sur ce fichier. Il est normalement devenu un exécutable.

Q112 Quelle commande permet d'exécuter ce script si le répertoire courant es le répertoire `~/bin` qui le contient ? Idem, si le répertoire courant est votre répertoire personnel. Vous vérifierez que le script se comporte comme attendu (il vous place dans une autre répertoire).

Q113 la commande `echo` permet d'afficher une message à l'écran. Modifiez le script pour qu'il avertisse l'utilisateur de la fin du script par un message explicite.

Syntaxe pour echo

```
echo  expression
```

Description

- Affiche sur la sortie standard l'expression après interprétation.

Exemple d'utilisation:

Affiche `'Bonjour'` :

```
login@host:~$ echo Bonjour
Bonjour
login@host:~$
```

Définie une variable puis affiche sa valeur :

```
login@host:~$ Astre=Terre
login@host:~$ echo $Astre
Terre
login@host:~$ echo La planete $Astre
La planete Terre
login@host:~$
```


Lancer un programme/une commande

Cas général

- Pour exécuter un programme il suffit saisir sur la ligne de commande le chemin menant au fichier contenant les instructions,
- Si le fichier présente la permission `"X"` pour exécutable, les instructions qu'il contient sont exécutées.

Cas particulier : les commandes

- Une commande (`ls`, `gedit`, `firefox`, …) est un programme comme un autre,
- Les instructions qui doivent être évaluées sont écrites dans un fichier (`/bin/ls`, `/usr/bin/python`, `/usr/share/bin/firefox`, …),
- Pourtant …

Des chemins qui mènent nulle part ! ! !

- les noms des commandes (`ls`, `gedit`, `firefox` …) sont toujours saisies comme des chemins relatifs (pas de `/bin/...` devant le nom du fichier), alors que le fichier de commande n'est pas dans le répertoire courant !…
- On donne donc un chemin vers un fichier qui n'existe pas …

Chemins par défaut et variable d'environnement

Lorsqu'on donne une commande au terminal, on ne spécifie pas le chemin vers le fichier qui contient l'exécutable, on donne juste le nom du fichier…

```
login@host:~$ ls
Mes_Documents/ Etoiles/ astronomie.txt
cv.pdf
login@host:~$ ■
```

…alors, comment le système trouve-t-il le fichier a exécuter correspondant à la commande ?…

Un mécanisme propre aux commandes

- Le premier mot tapé sur la ligne de commande est toujours interprétée comme le nom d'un fichier exécutable,
- Le système recherche donc dans une liste de répertoires contenant les exécutables si un fichier porte le nom de cette commande,
- Dès qu'il trouve dans ces répertoires un tel fichier, il l'exécute …

Chemins par défaut et variable d'environnement

Les variables d'environnement

- Comme les variables d'un script, les variables d'environnement sont associées à une valeur,
- De telles variables sont définies par le système d'exploitation pour son fonctionnement, ce sont les variables d'environnement,
- ces variables peuvent être utilisées par les programmes.

La variable d'environnement $PATH

- Sa valeur est une liste de répertoires séparés par le signe `':'`

```
PATH=repertoire1:repertoire2:...:RepertoireN
```

- Lors de chaque appel de commande, l'interpréteur parcourt cette liste dans l'ordre à la recherche d'un fichier portant le nom de la commande,
- Dès qu'il rencontre un tel fichier, il met fin à sa recherche et exécute le fichier.

Rôle de $PATH

→ Il s'agit d'une liste de répertoires que l'interpréteur parcours automatiquement et séquentiellement (par défaut) si aucun chemin n'est donné pour trouver le fichier exécutable.

Syntaxe pour which

```
which  nom_de_la_commande
```

Description

- Affiche le chemin du fichier correspondant à une commande.
- Parcours successivement les répertoires de la variable $PATH. Dès qu'il trouve un fichier correspondant au nom de la commande il renvoie son chemin.

Exemple d'utilisation:


```
login@host:/home/chez_mo$ echo $PATH
/bin:/usr/bin:/usr/local/bin:/home/chez_moi
login@host:/home/chez_moi$ which ls
/bin/ls
```


Chemins par défaut et variable d'environnement

La commande `export` pour modifer la variable $PATH

- Définir la variable $PATH

```
login@host:~/$ export PATH=monDir1:monDir2
```

- Ajouter un répertoire à $PATH

```
login@host:~/$ export PATH=$PATH:monDir2
```

Exercices

Environnement

Q114 Au moyen de la commande `env`, donnez la liste des répertoires contenus dans `$PATH`.

Q115 Au moyen de la commande `which`, afficher la localisation des exécutables correspondants aux commandes `mv`, `cd`, `man`, `cat`, `firefox`, `acroread`.

Q116 Vérifiez que ces répertoires font partie de la liste contenue dans la variable `$PATH` ? Que se passerait-il si ce n'était pas le cas ?

Q117 Ajouter le répertoire `~/bin` à la liste des répertoires `$PATH`.

Q118 Maintenant que `~/bin` est parcoure par défaut lors de l'appel d'une commande, comment invoque-t-on désormais l'exécution du script `listintro.sh` ? Vérifiez le comportement attendu.

Fichiers de configuration

Fichiers systèmes et utilisateurs

- Les variables d'environnement (et d'autres variables de configuration) sont définis dans divers fichiers.
- On distingue les fichiers système qui définissent des comportements pour tous les utilisateurs (stockés dans le répertoire `/etc/`) des fichiers de configuration propres à un utilisateur (stockés dans le répertoire personnel)

fichier	Propriétaire	Applicable à	Évalué lors
`/etc/profile`	`root`	Tous	Au début de chaque shell de login
`/home/chez_moi/.profile`	`utilisateur`	utilisateur	Au début de chaque shell de loginl
`/etc/bashrc`	`root`	Tous	Au début de chaque shell
`/home/chez_moi/.bashrc`	`utilisateur`	utilisateur	Au début de chaque shell

Configurer son environnement

- Chaque utilisateur peut redéfinir ses variables d'environnement,
- Pour cela il peut modifier le contenu des fichiers `.bashrc` et `.profile` dans son répertoire personnel,
- Ce sont des fichiers cachés (leur nom commence par un point : `.`). Pour voir si ils existent il faut utiliser l'option `-a` de la commande `ls`

Fichiers de configuration

Contenu d'un fichier `.bashrc`

- Redéfinition des variables d'environnement,
- Définition des alias,
- Définition des fonctions,
- et de façon générale toutes les instructions que l'on souhaite évaluer lors de l'ouverture d'un nouveau shell.

.bashrc

```
# Mes aliases
alias ll='ls -l'
alias df='df -h'
alias rm='rm -i'
# Mes variables
PATH=$PATH:$HOME/bin
```

Autres variables d'environnement

`$HOME` le chemin du répertoire personnel de l'utilisateur,

`$PWD` le chemin du répertoire courant.

Syntaxe pour alias

```
alias  nom_de_la_commande=expression
```

Description

- créet un alias entre un nom de commande et une expression.
- l'expression est donnée entre *quotes* : *'expression ...'*

Exemple d'utilisation:

chez_moi/ ..... Répertoire
- public_html/
  - index.html
- astronomie.txt

```
login@host:~$ ll
-bash: ll: command not found
login@host:~$ alias ll='ls -l'
login@host:~$ ls -l
 total 32
 drwxr-xr-x 2 santini ensinfo 4096 20 jui 15:50     public_html
   -rw-r-r- 1 santini ensinfo   25 20 jui 15:49 telluriques.txt
```

Exercices

Chemins par défaut et variable d'environnement

Q119 Copiez l'exécutable de la commande `ls` dans le répertoire `~/bin`. Deux versions de la même commande existe dans 2 répertoires différents listés sans $PATH. Quelle commande est exécutée ? Comment en être sur et pourquoi ?

Q120 Si vous modifiez la variable $PATH, de la façon suivante, quelle commande est alors exécutée ?

```
login@host:~/$ export PATH=monDir2:$PATH
```

Q121 Modifiez/créez un fichier `~/.bashrc` pour ajouter le répertoire `~/bin` de façon stable à votre variable $PATH.

Q122 ajoutez dans le même fichier les alias qui vous paraissent intéressants.

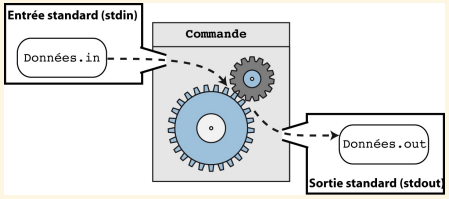
Flux de données

Entrée et sortie standard
Redirections
Tubes

Entrée et sortie standard

Rappel : Les programmes informatiques

- Un programme prend des données en entrée. Ces données peuvent être lues dans un fichier ou fournies par un flux du système.
- Le programme manipule ces données.
- Le programme fournit un résultat en sortie (des données). Ces données peuvent être écrites dans un fichier ou exportées comme un flux vers le système.



Les flux de données

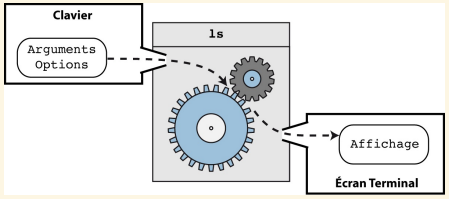
Pour fonctionner, un programme a donc besoin de lire des données (flux d'entrée : input) et d'écrire les résultats de ses évaluations (flux de sortie : output). On distingue 3 types de flux de données :

- **STDIN** : entrée standard (là où sont lues les données),
- **STDOUT** : sortie standard (là où sont écrits les résultats),
- **STDERR** : sortie erreur (là où sont écrit les messages d'erreur).

Entrée et sortie standard

Les commandes qui lisent sur l'entrée standard

- Certaines commandes Linux qui traitent les données d'un fichier (dont le chemin est passé en paramètre) peuvent alternativement, si aucun chemin fichier n'est spécifié, travailler directement avec les données lues sur l'entrée standard.
- Par exemple : `echo`, `cat`, `head`, `tail`, `grep`.
- **Par défaut, l'entrée standard est le clavier**.



Les commandes qui écrivent sur la sortie standard

- Les affichages produits par les commandes Linux sont le résultat de leur évaluation. Ce résultat est écrit sur la sortie standard.
- **Par défaut, la sortie standard est l'écran**.

Syntaxe pour cat

```
cat  fichier [fichier_2 ...]
```

Description

- Affiche le contenu des fichiers les uns à la suite des autres.
- Les fichiers sont concaténés dans l'ordre des paramètres.

Exemple d'utilisation:

Cette commande est en générale utilisée pour concaténer des fichiers textes. On l'utilise avec une commande de redirection (cf. Partie Redirections) pour enregistrer le résultat de la concaténation dans un nouveau fichier.

Soient les deux fichiers suivants :

tellur.txt

```
Mercure, Venus
Terre, Mars
```

jov.txt

```
Jupiter, Saturne
Uranus, Neptune
```

La commande :

```
login@host:~$ cat tellur.txt jov.txt
Mercure, Venus
Terre, Mars
Jupiter, Saturne
Uranus, Neptune
login@host:~$
```


Syntaxe pour head

```
head < -int > fichier
```

Description

- Affiche par défaut les 10 premières lignes d'un fichier.
- Si un entier n précède le nom du fichier, la commande affiche les n premières lignes du fichier.

Exemple d'utilisation:

Soit le fichier `planetes.txt` contenant les lignes suivantes :

planetes.txt

```
# Premier groupe
1  Mercure
Tellurique
2  Venus
Tellurique
3  Terre
Tellurique
4  Mars
Tellurique
# Deuxième groupe
1  Jupiter
Gazeuse
2  Saturne
```

La commande suivante affiche les 5 premières lignes du fichier :

```
login@host:~$ head -5 planetes.txt
# Premier groupe
1  Mercure  Tellurique
2  Venus    Tellurique
3  Terre    Tellurique
4  Mars     Tellurique
login@host:~$
```


Syntaxe pour tail

```
tail < -int > fichier
```

Description

- Affiche par défaut les 10 dernières lignes d'un fichier.
- Si un entier n précède le nom du fichier, la commande affiche les n dernières lignes du fichier.

Exemple d'utilisation:

Soit le fichier `planetes.txt` contenant les lignes suivantes :

planetes.txt

```
# Premier groupe
1  Mercure
Tellurique
2  Venus
Tellurique
3  Terre
Tellurique
4  Mars
Tellurique
# Deuxième groupe
1  Jupiter
Gazeuse
2  Saturne
3  Uranus
Gazeuse
4  Neptune
Gazeuse
```

La commande suivante affiche les 4 dernières lignes du fichier :

```
login@host:~$ tail -4 planetes.txt
1  Jupiter  Gazeuse
2  Saturne  Gazeuse
3  Uranus   Gazeuse
4  Neptune  Gazeuse
login@host:~$
```


Syntaxe pour grep

```
grep  "motif" fichier
```

Description

- Affiche les lignes du fichier qui comportent le `"motif"`.
- Les lignes sont affichées dans leur ordre d'apparition dans le fichier.

Exemple d'utilisation:

Soit le fichier `planetes.txt` contenant les lignes suivantes :

planetes.txt

```
# Premier groupe
1  Mercure
Tellurique
2  Venus
Tellurique
3  Terre
Tellurique
4  Mars
Tellurique
# Deuxième groupe
1  Jupiter
Gazeuse
2  Saturne
Gazeuse
3  Uranus
Gazeuse
4  Neptune
```

Commandes :

```
login@host:~$ grep 'Tellurique'
planetes.txt
1  Mercure  Tellurique
2  Venus    Tellurique
3  Terre    Tellurique
4  Mars     Tellurique
login@host:~$ grep '1' planetes.txt
1  Mercure  Tellurique
1  Jupiter  Gazeuse
```

Exercices

Manipulation du contenu d'un fichier texte

Q123 La commande suivante montre le contenu d'un fichier texte :

```
login@host:~/$ cat /proc/cpuinfo
```

Q124 Quelle sont les informations contenues dans ce fichier ?

Q125 À l'aide des commandes `cat` ou `less` identifiez dans le fichier /proc/cpuinfo le nombre de fois ou le mot 'cpu' apparait

Q126 La commande `grep 'cpu' /proc/cpuinfo` permet d'afficher les lignes du fichier `/proc/cpuinfo` où le mot 'cpu' apparait. Vérifiez qu'il y en le bon nombre ?

Q127 L'option -v permet d'inverser son comportement. Au lieu d'afficher les lignes qui présentent le motif, `grep` affiche alors les lignes qui ne présentent pas le motif. Affichez les lignes du fichier `/proc/cpuinfo` ne présentant pas le mot 'cpu'.

Q128 Proposez une commande permettant d'afficher les premières 5 lignes

Q129 Proposez une commande permettant d'afficher les dernières 5 lignes

Redirection des Entrée/Sorties

Commandes de Redirection

Il est possible de modifier le comportement par défaut des commandes et de donner une entrée et/ou une sortie standard différente des entrées/sorties standards.

```
command > fichier.out
```

- **Redirige la sortie standard** de la commande `command` vers le fichier `fichier.out`.
- Si le fichier `fichier.out` n'existe pas, il est créé avec comme contenu les affichages produits par la commande `command`.
- **Si le fichier `fichier.out` existe, son contenu est écrasé** et remplacé par les affichages produits par la commande `command`.

```
command » fichier.out
```

- **Redirige la sortie standard** de la commande `command` vers le fichier `fichier.out`.
- Si le fichier `fichier.out` n'existe pas, il est créé avec comme contenu les affichages produits par la commande `command`.
- Si le fichier `fichier.out` existe, les affichages produits par la commande `command` sont **ajoutés à la fin du contenu du fichier**.

```
command 2> fichier.err
```

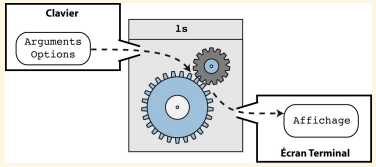
- **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec écrasement du contenu** si le fichier de sortie existe déjà.

```
command 2» fichier.err
```

- **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec préservation du contenu** si le fichier de sortie existe déjà.

Exemple de redirection

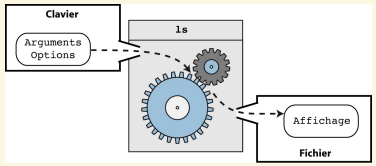
Comportement par défaut de la commande `ls`




```
login@host:~$ ls
aldenaran.jpg alphacentauri.gif
etacentauri.jpg
login@host:~$ ls
aldenaran.jpg alphacentauri.gif
etacentauri.jpg
login@host:~$ ■
```

La sortie standard de la première commande `ls` est l'écran. La liste du contenu du répertoire courant est affichée à l'écran.

Redirection de la sortie de la commande `ls`




```
login@host:~$ ls > 1.out
login@host:~$ ls
1.out aldenaran.jpg alphacentauri.gif
etacentauri.jpg
login@host:~$ ■
```

La sortie standard de la première commande `ls` est redirigée vers le fichier `1.out`. La liste du contenu du répertoire courant est écrite dans le fichier `1.out`.
La deuxième commande `ls`, montre qu'un fichier portant le nom `1.out` a été créé.

Syntaxe pour echo

```
echo expression
```

Description

- Affiche sur la sortie standard l'expression après interprétation.

Exemple d'utilisation:

Affiche `'Bonjour'` :

```
login@host:~$ echo Bonjour
Bonjour
login@host:~$ ■
```

Définie une variable puis affiche sa valeur :

```
login@host:~$ Astre=Terre
login@host:~$ echo $Astre
Terre
login@host:~$ echo La planete $Astre
La planete Terre
login@host:~$ ■
```

Exercices

Redirections

Q130 Que font les commandes suivantes ?

```
login@host:~$ echo “Bonjour"
login@host:~$ echo “Bonjour" > bonjour.out
login@host:~$ echo “Salut" > bonjour.out
login@host:~$ echo “Bonjour" » bonjour.out
```

Q131 Entrainez-vous avec les commandes suivantes. Profitez-en pour comprendre les affichages produits par les commandes `ps` et `file` :

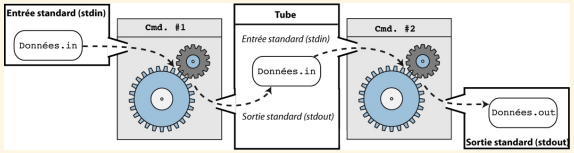
```
login@host:~$ ps > essai_ps.out
login@host:~$ file /usr/include/stdio.h > file.out
```

Q132 Proposez une commande pour copier le contenu de /proc/cpuinfo dans un fichier cpuinfo.out sans utiliser la commande `cp`

Tubes

Principes de fonctionnement des Tubes (Pipe en anglais)

- A la différence des redirections simples qui permettent de rediriger la sortie standard d'une commande vers un fichier,
- Un tube permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande.



Syntaxe

- Le tube est symbolisé par le caractère `|`.

```
cmd1 | cmd2
```

- - La sortie standard de la première commande (`cmd1`) est redirigée vers l'entrée standard de la deuxième commande (`cmd2`).
  - L'entrée standard de la commande `cmd1` et la sortie standard de la commande `cmd2` ne sont pas modifiées.

Exemple de Tubes avec les commande `ls` et `more`

Rappel des commandes :

- `ls` affiche à l'écran (stdout) la liste des fichiers contenus dans un répertoire.
- `more` affiche page par page le contenu des données passée sur son entrée standard.

Exemple #1

- Si de très nombreux fichiers sont contenus dans un répertoire, la commande `ls` peut produire un affichage qui ne tient pas dans l'écran, rendant impossible le parcours de la liste des fichiers (seuls les derniers sont visibles).

```
login@host:~$ ls
```

Défilement de tous les fichiers

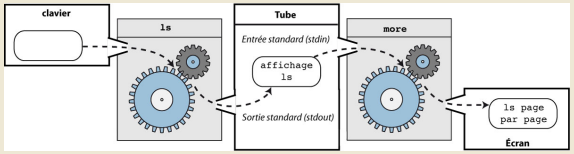
```
betelgeuse.jpg    etacentauri.jpg
soleil.jpg        syrius.gif
vega.png
login@host:~$ ■
```

Images/ . . . . . Répertoire courant
- aldebaran.jpg . . . . . Hors de la fenetre
- alphacentauri.gif . . . . . Hors de la fenetre
- betelgeuse.jpg . . . . . Dans la fenetre
- etacentauri.jpg . . . . . Dans la fenetre
- soleil.jpg . . . . . Dans la fenetre
- syrius.gif . . . . . Dans la fenetre
- vega.png . . . . . Dans la fenetre

Exemple de Tubes avec les commande `ls` et `more`

Exemple #1 (suite) :

- La redirection de la sortie standard de la commande `ls` vers l'entrée standard de la commande `more` permet de passer en revue l'affichage de la commande `ls` page par page.




```
login@host:~$ ls | more
aldebaran.jpg
alphacentauri.gif
betelgeuse.jpg
etacentauri.jpg
soleil.jpg        syrius.gif
```

Affichage d'une première page puis
Presser la touche Enter pour la page suivante

```
soleil.jpg        syrius.gif
vega.png
login@host:~$ ■
```

Images/ . . . . . Répertoire courant
- aldebaran.jpg . . . . . Page 1
- alphacentauri.gif . . . . . Page 1
- betelgeuse.jpg . . . . . Page 1
- etacentauri.jpg . . . . . Page 1
- soleil.jpg . . . . . Page 1&2
- syrius.gif . . . . . Page 1&2
- vega.png . . . . . Page 2

Exemple de Tubes avec les commande `ls` et `grep`

Rappel des commandes :

- `ls` affiche à l'écran (stdout) la liste des fichiers contenus dans un répertoire.
- `grep` affiche les lignes d'un texte qui comportent un certain motif.

Exemple #2 :

- Si de très nombreux fichiers sont contenus dans un répertoire, la commande `ls` peut produire un affichage qui ne tient pas dans l'écran, rendant compliqué l'identification de certain type de fichier (fichiers au format `gif` par exemple).

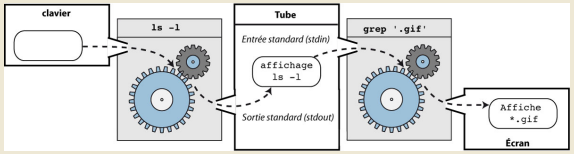
```
login@host:~$ ls
aldebaran.jpg
alphacentauri.gif
betelgeuse.jpg
etacentauri.jpg
soleil.jpg
syrius.gif
vega.png
login@host:~$ ■
```

Images/ . . . . . Répertoire courant
- aldebaran.jpg . . . . . Affiché
- alphacentauri.gif . . . . . Affiché
- betelgeuse.jpg . . . . . Affiché
- etacentauri.jpg . . . . . Affiché
- soleil.jpg . . . . . Affiché
- syrius.gif . . . . . Affiché
- vega.png . . . . . Affiché

Exemple de Tubes avec les commande `ls` et `more`

Exemple #2 (suite) :

- La redirection de la sortie standard de la commande `ls` vers l'entrée standard de la commande `grep` permet d'effectuer un filtrage des fichiers présents dans le répertoire sur la base d'un motif présent dans leur nom (par exemple l'extension `.gif`).




```
login@host:~$ ls | grep '.gif'
alphacentauri.gif
syrius.gif
login@host:~$ ■
```

Images/ . . . . . Répertoire courant
- aldebaran.jpg . . . . . Retenu par le filtre
- alphacentauri.gif . . . . . Affiché
- betelgeuse.jpg . . . . . Retenu par le filtre
- etacentauri.jpg . . . . . Retenu par le filtre
- soleil.jpg . . . . . Retenu par le filtre
- syrius.gif . . . . . Affiché
- vega.png . . . . . Retenu par le filtre

Syntaxe pour wc

```
wc  fichier <fichier_2 ...>
```

Description

- Affiche des statistiques sur le nombre de lignes, de mots et de caractères (comptés en nombre d'octets) contenus dans le fichier dont le chemin est donné en paramètre.

Exemple d'utilisation:

Soit le fichier suivant :

tellur.tsv

```
1 Mercure Venus
2 Terre Mars
```

Commande #1 :

```
login@host:~$ wc tellur.tsv
2     6     29  tellur.tsv
login@host:~$ ■
```

L'affichage produit indique que le fichier `tellur.tsv` comporte :

- 2 lignes,
- 6 mots et
- 29 caractères. La taille du fichier texte est donc de 29 octets …

Exercices

Tubes

Q133 Étudiez et comparez les commandes suivantes. Pour vous aider vous pouvez évaluer les commandes pas à pas en vous arrêtant avant chaque tube.

```
login@host:~$ cat /proc/cpuinfo | wc -l
login@host:~$ head /proc/cpuinfo | wc -l
login@host:~$ cat /proc/cpuinfo | grep 'cpu' | wc
-l
login@host:~$ head /proc/cpuinfo | grep 'cpu' | wc
-l
```

Q134 Proposez une commande pour afficher le nombre de fichiers dans votre répertoire home

Q135 Proposez une commande pour afficher le nombre des processus

Q136 Proposez une commande pour afficher les premières 5 lignes des dernières 10 lignes du fichier /proc/cpuinfo

Les scripts Bash

Introduction
Variables et Paramètres

Introduction

Structure d'un script Bash

- Un script Bash commence toujours par la ligne # !/bin/bash , suivi par une série d'instructions et commentaires (optionels)
- Un commentaire est une partie rédigée du script qui ne sera pas considérée comme une instruction lors de l'exécution du script. Pour commenter une portion du script on utilise le caractère #. L'ensemble du texte situé sur la même ligne et après le carcactère # sera considéré comme un commentaire et ne sera pas évalué.

Exemple

```
#!/bin/bash
echo Liste des Fichiers:
#affiche la liste
ls
```


Rappel

Les interpréteurs

- L'interpréteur parcourt le texte tapé par l'utilisateur, identifie les commandes et les paramètres, et si la syntaxe est correcte, lance un processus.
- Plusieurs interpréteurs existent : csh, tcsh, bash.
- Bash est l'interpréteur du projet GNU. Il est le plus utilisé sous linux. C'est Bash l'interpréteur qu'on utilise dans ce cours.
- L'interpréteur peut lire les commandes à partir d'un fichier, le *script* shell.

Introduction

Execution d'un script

- Un script est un simple fichier texte (habituellement, ils ont l'extension .sh) . Pour l'executer, il faut avant tout le rendre exécutable : `chmod u+x script.sh`
- Maintenant, on peut l'exécuter en faisant : `./script.sh`
- On peut aussi le lancer en appelant explicitement l'interpréteur : `bash script.sh`

Premier script Bash

Q137 Après avoir créé un repertoire nommé `/Intro_Systeme/TP_3/scripts/`, écrivez et exécutez un script `exo_0_script.sh` qui affiche à l'écran le nombre de fichiers contenus dans le repertoire courant, après un message de texte "Nombre de fichiers :"

Exercices

Introduction aux scripts Bash

Q138 Définissez et exécutez un script nommé `exo_1_script.sh` qui réalise la suite de commandes suivante : `echo "Debut"; sleep 2; echo "Apres 2 sec."; sleep 5; echo "Apres 5sec"`

Q139 Que se passe-t-il si vous commentez les lignes commencant par la commande sleep ?

Q140 Définissez un script `exo_2_script.sh` qui affiche "Bonjour", définit le répertoire `/Intro_Systeme/TP_3/scripts/` comme répertoire courant, puis crée dans celui-ci un répertoire `Test`, et finalement copie dans `Test` le fichier `/proc/cpuinfo`.

Q141 Définissez un script nommé `exo_3_script.sh` qui affiche le contenu du répertoire `Test`, puis supprime le fichier `cpuinfo` y contenu (`Test/cpuinfo`), et finalement crée dans `Test` un fichier `infoCPU.txt` composé par les lignes du fichier `/proc/cpuinfo` qui contiennent le mot `'cpu'`.

Les Variables

Les variables en Bash

- Pour affecter une valeur à une variable c'est très simple. Il suffit d'écrire `nom_variable=valeur`
- Pour accéder au contenu d'une variable, il faut utiliser le préfixe `$`
- On peut accéder aussi aux variables d'environnement, qui ont été définies ailleurs (par exemple `$PATH`)

Exemple

```
MSG=Bonjour
echo $MSG
echo $PATH
```

Les Variables

Q142 Définissez un script nommé `exo_4_script.sh` à partir du script `exo_2_script.sh`, et modifiez-le pour que le nom du répertoire `Test/` soit une variable dans le script.

Les Paramètres

Les paramètres

- Il s'agit d'unes variables spéciales qui contiennent les arguments fournis au script par la ligne de commandes
- **$0** : nom du script
- **$1 $2 ...** : paramètres en position 1, 2, ...
- **$#** : nombre de paramètres positionnels
- **$*** : ensemble des paramètres

Exemple

Soit `arg.sh` le script suivant :

```
#!/bin/bash
echo "Nombre d'argument "$#
echo "Les arguments sont "$*
echo "Le second argument est "$2
```

```
login@host:~$ ./arg.sh A B C
Nombre d'argument 3
Les arguments sont A B C
Le second argument est B
```


Exercices

Introduction aux scripts Bash

Q143 Définissez un script nommé `exo_5_script.sh` à partir du script `exo_2_script.sh`, et modifiez-le pour que le nom du répertoire `Test/` soit passé comme un paramètre du script.

Q144 Rédigez un script recevant 3 paramètres (nom, prénom et serveur) permettant l'affichage d'une adresse mail formatée (nom.prénom@serveur)

Structures de contrôle en BASH

Exercices

Les calculs arithmétiques

Q145 Proposez une suite de 2 commandes affectant à une variable `res` le résultat des opérations arithmétiques suivantes et affichant le résultat contenu dans cette variable : $5 + 7$ et $3 * 2$

Q146 Proposez une suite de 3 commandes permettant :

- d'affecter à une variable `res` la valeur 3,
- d'ajouter 13 à la variable `res`,
- d'afficher le résultat de l'addition stockée dans la variable `res`.

Les calculs arithmétiques

`Bash` un langage orienté sur le traitement des chaînes de caractères

Même si ce langage n'est pas fait pour effectuer des opérations de calcul arithmétique il propose des fonctionnalités de base permettant d'effectuer des calculs simples tels que les additions, soustractions, multiplications et divisions.

Syntaxe

`$((` *expression_arithmétique* **`))`**

Exemples

```
login@host:~$ total=$(( 5 + 3 ))
login@host:~$ echo $total
8
login@host:~$ echo $(( 5 - 3 ))
2
login@host:~$ echo $(( 5 * 3 ))
15
login@host:~$ echo $(( 5 / 3 ))
1
```


`for`

`for` Boucle itérative

- permet de répéter l'évaluation d'une ou plusieurs instructions,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur atteint une certaine valeurs.

Syntaxe #1

```
for (( init ; test ; incr )) ; do
    expr_1
    expr_2
    ...
done
```

Ici, la condition d'arrêt est sur la valeur numérique de l'itérateur.

Exemple #1

test_for_loop_1.bash

```
#!/bin/bash

echo "test #1"
for (( i = 0; i < 3; i++
));do
   echo '$i = '$i
done
```

```
login@host:~$
./test_for_loop_1.bash
test #1
$i = 0
$i = 1
$i = 2
```

for

for Boucle itérative

- permet de répéter l'évaluation d'une instruction,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur a parcouru toute la liste.

Syntaxe #2

```
for var in val_1 val_2 ... ; do
    expr_1
    expr_2
    ...
done
```

Ici, la boucle s'arrête lorsque toute la liste des valeurs a été parcourue.

Exemple #2

test_for_loop_2.bash

```
#!/bin/bash

echo "test #2"
for i in hello la terre;do
   echo '$i = '$i
done
```

```
login@host:~$
./test_for_loop_2.bash
test #2
$i = hello
$i = la
$i = terre
```


Exercices

La boucle for

Q147 Dans le cours nous avons vu plusieurs syntaxes possibles pour la boucle for. Soit le script suivant :

```
#!/bin/bash
# affiche les 10 premiers entiers pairs
for int in 2 4 6 8 10 12 14 16 18 20
do
echo $int
done
```

Q148 Modifiez ce script pour remplacer la liste de valeurs par une expression arithmétique

if

Branchements conditionnels

- Le `if` permet de mettre en place des alternatives.
- Un test (dont le résultat est Vrai ou Faux) permet de conditionner les expressions qui seront évaluées.

Syntaxe #1

```
if test
then
    expr_1
    expr_2
    ...
fi
```

Comportement

- Ici, les expressions ne sont évaluées que si le test renvoie la valeur Vrai.
- Aucune des expressions ne sont évaluées si le test renvoie la valeur Faux.

if

Syntaxe #2

```
if test
then
    expr_1
else
    expr_2
fi
```

Comportement

- Si le test renvoie la valeur Vrai l'expression `expr_1` est évaluée, et
- sinon le test renvoie la valeur Faux c'est l'expression `expr_2` qui est évaluée.

Syntaxe #3

```
if test_1
then
    expr_1
elif test_2
then
    expr_2
elif test_3
then
    expr_3
else
    ...
fi
```

Comportement

- Si `test_1` renvoie la valeur Vrai l'expression `expr_1` est évaluée,
- si `test_2` renvoie la valeur Vrai l'expression `expr_2` est évaluée,
- si `test_3` renvoie la valeur Vrai l'expression `expr_3` est évaluée, et
- si aucun des tests ne renvoie la valeur Vrai alors c'est l'expression

Les tests

Les tests peuvent prendre plusieurs formes

Il peuvent porter sur :

- l'arborescence (présence, absence, permission sur les répertoires et fichiers),
- les chaînes de caractères,
- les valeurs numériques.

Tests de l'arborescence

Syntaxe	Valeur
[-d fichier]	Vrai si fichier est un nom de répertoire valide (si il existe).
[-f fichier]	Vrai si fichier est un nom de fichier valide (si il existe).
[-r fichier]	Vrai si il y a le droit de lecture sur le fichier.
[-w fichier]	Vrai si il y a le droit d'écriture sur le fichier.
[-x fichier]	Vrai si il y a le droit d'exécution sur le fichier.

Exercices

Tests de l'arborescence

Q149 Créez un script `ico_existe.sh`, qui teste si un fichier `ico` est présent dans le répertoire courant. Si le fichier existe, le script affiche le message d'avertissement suivant ($PWD sera remplacé lors de l'exécution par la valeur de la variable d'environnement) :

```
Attention: le fichier $PWD/ico existe
```

Q150 Modifiez le script pour qu'il supprime le fichier `ico` si celui-ci existe et affiche un message d'avertissement indiquant que le fichier est supprimé. Les affichages seront alors les suivants :

```
Attention: le fichier $PWD/ico existe
Le Fichier $PWD/ico est supprime
```

Q151 Modifiez ce script pour qu'il teste en plus si le répertoire courant contient un répertoire nommé `ico/`. Si il ne contient pas de répertoire `ico/`, le script crée ce répertoire.

Les tests

Tests sur les chaînes de caractères

Syntaxe	Valeur
[chaine_1 = chaine_2]	Vrai si les 2 chaînes sont identiques.
[chaine_1 != chaine_2]	Vrai si les 2 chaînes sont différentes.
[-n chaine]	Vrai si la chaîne est non vide.
[-z chaine]	Vrai si la chaîne est vide.

Tests sur les chaînes

Q152 Définissez un script `testPWD.sh` qui prend en paramètre une chaîne de caractères et la compare avec la variable d'environnement `$PWD`, il doit afficher 'OK' si le paramètre correspond à la valeur de la variable, 'Non' en cas contraire.

Les tests

Tests sur les valeurs numériques

Syntaxe	Valeur
[nb_1 -eq nb_2]	Vrai si nb_1 = nb_2 (eq pour equal).
[nb_1 -ne nb_2]	Vrai si nb_1 $\neq$ nb_2 (ne pour not equal).
[nb_1 -gt nb_2]	Vrai si nb_1 $>$ nb_2 (gt pour greater than).
[nb_1 -ge nb_2]	Vrai si nb_1 $\geq$ nb_2 (ge pour greater or equal).
[nb_1 -lt nb_2]	Vrai si nb_1 $<$ nb_2 (ge pour lower than).
[nb_1 -le nb_2]	Vrai si nb_1 $\leq$ nb_2 (ge pour lower or equal).

Les tests

Opérateurs booléens

Syntaxe	Valeur
! [test]	NOT : Vrai si le test renvoie Faux (négation).
[test_1] | | [test_2]	OU logique.
[test_1] && [test_2]	ET logique.

Tables de vérité

ET (&&)	Vrai	Faux
Vrai	Vrai	Faux
Faux	Faux	Faux

OU (| |)	Vrai	Faux
Vrai	Vrai	Vrai
Faux	Vrai	Faux

NOT (!)	Vrai	Faux
	Faux	Vra

Substitution de commande

Un moyen de composer les instructions

La syntaxe `$(commande avec des arguments)` est remplacée à l'exécution par le résultat de l'exécution dans un sous-shell de la commande `commande avec des arguments`.
Cette fonctionnalité très puissante permet d'utiliser des commandes pour les affecter dans des variables et ensuite s'en servir dans le script.
C'est une substitution

Exemple

```
#!/bin/bash
TITLE="En ce jour du $(date -I)"
MOTS=$(grep cool /usr/share/dict/words)
for i in $MOTS; do
  echo "$TITLE, $i est un mot cool"
done
```


Exercices

Tests sur les valeurs numériques

Q153 Définissez un script `testTemp.sh` qui prend en paramètre une valeur numérique et une lettre ('C' ou 'F'). Si la lettre choisie est 'C', le script doit afficher 'chaud' si le paramètre numérique est plus grand que `25`, 'froid' si est moins que `10`, 'normal' dans les autres cas. Si la lettre choisie est 'F', il affiche 'chaud' si le paramètre numérique est plus grand que `78` et 'froid' si le paramètre numérique est inférieur à `50`, 'normal' dans les autres cas. Si la lettre n'est pas 'C' ou 'F', il affiche un message d'erreur.

Exercices

Archiveur

Faites un script qui a les actions suivantes si on lui donne en argument un répertoire (par exemple `~/M1101/TD6`) :

Q154 S'arrête si la cible n'est pas un répertoire

Q155 Définit une variable BACKUPDIR qui vaut le chemin du répertoire du dessus suivi du mot `sauvegarde` (ici `~/M1101/sauvegarde`) en utilisant la commande `dirname`

Q156 Crée le répertoire s'il n'existe pas

Q157 Définir une variable faite avec la date du jour et le nom du répertoire (par exemple 2014-10-31-TD6) en utilisante les commandes `basename` et `date`.

Q158 Crée une archive compressée du répertoire (ici en exécutant `tar czf ~/M1101/sauvegarde/2014-10-31-TD6.tgz ~/M1101/TD6`)

On pourra affiner en s'arrêtant si une archive existe déjà sous ce nom avant de la créer (ou proposer de l'effacer en utilisant la commande `read x` pour lire une variable depuis le terminal).