UNIVERSITE HASSAN II DE CASABLANCA

FACULTE DES SCIENCES Ben M'sik

Département De mathématique et informatique

MODULE : SYSTÈME D'EXPLOITATION 1

FILIÈRE : SMI/S3

Pr Nawal SAEL

Année universitaire : 2018/2019

Descriptif du module

Structure du module:

- Introduction Aux Systèmes d'Exploitation
- Introduction aux système d'exploitation Linux
- Les Commandes de Base du Système
- La Programmation Shell

Charge horaire :

Cours	**12 Séances**
TD-TP	14 Séances

Introduction Aux Systèmes d'Exploitation

Avant propos

Nous devons répondre à des questions élémentaires :

C'est quoi un système d'exploitation ?
Quel est son rôle ?

Avant propos : Avant propos

La gestion d'un système informatique donné, se fait a priori en langage machine. Ceci est primaire et lourd à gérer pour la plupart des ordinateurs, en particulier en ce qui concerne les entrées-sorties.

Bien peu de programmes seraient développés si chaque programmeur devait connaître le fonctionnement, par exemple, de tel ou tel disque dur et toutes les erreurs qui peuvent apparaître lors de la lecture d'un bloc.

Il a donc fallu trouver un moyen de libérer les programmeurs de la complexité du matériel.

Solution : Enrober le matériel avec une couche de logiciel qui gère l'ensemble du système. Il faut présenter au programmeur une interface de programmation d'application, ce qui correspond à une machine virtuelle plus facile à comprendre et à programmer.

Avant propos : Avant propos

Un ordinateur moderne comprend:

- Un ou plusieurs processeurs
- Une Mémoire principale
- Disques
- Divers dispositifs d'entrée / sortie : Imprimantes….

La gestion de tous ces composants variés nécessite une couche logiciel appelé

Système d'exploitation (OS)

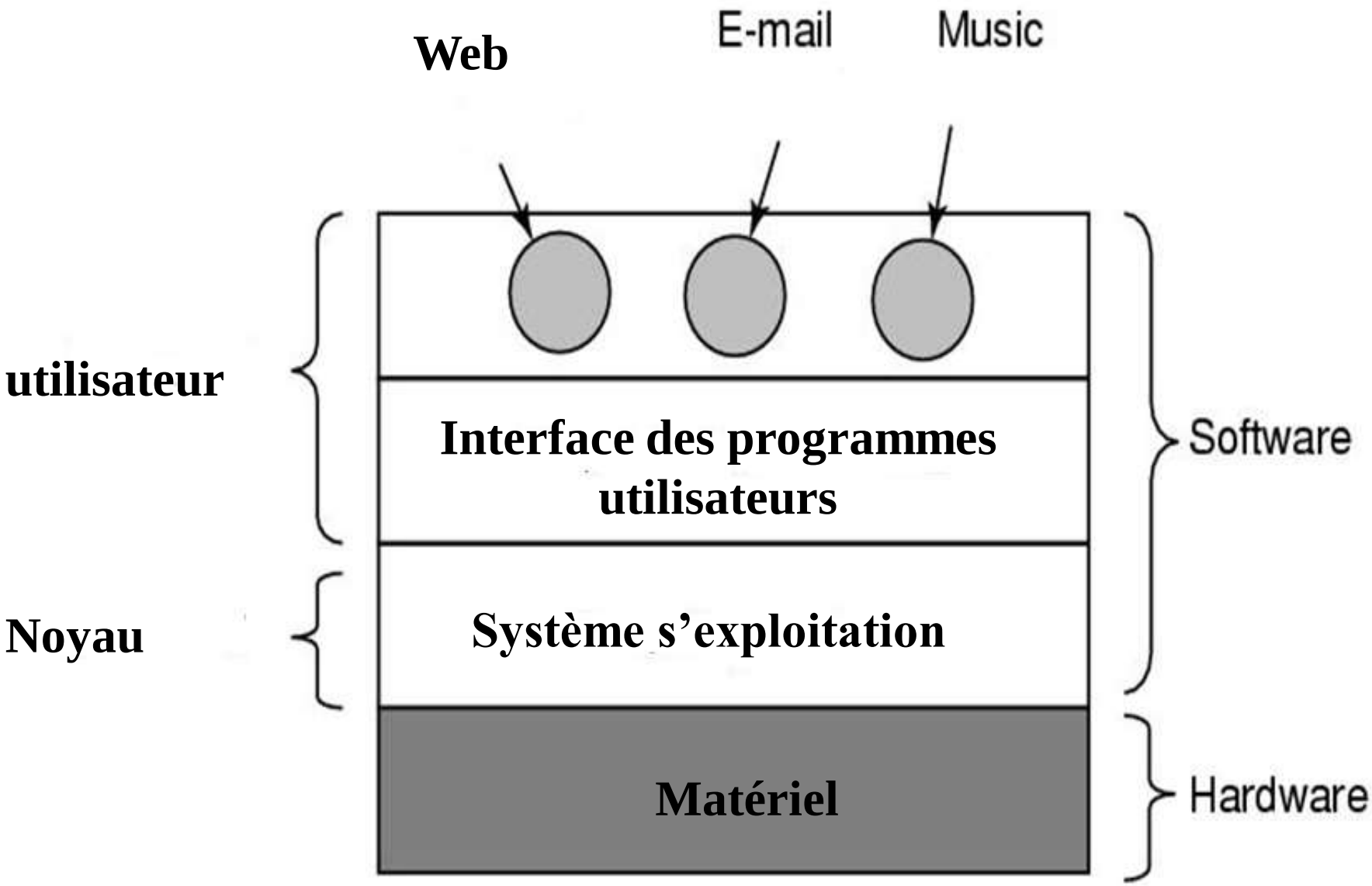
C'est quoi un système d'exploitations

Un système d'exploitation est un programme qui sert comme intermédiaire / interface entre un utilisateur d'un ordinateur et le matériel informatique.

L'objectifs du système d'exploitation:

- Contrôler / Exécuter des programmes / Application utilisateur.
- Rendre le système informatique facile à utiliser.
- Utilisez le matériel informatique de manière efficace.

Où se situe le système d'exploitation

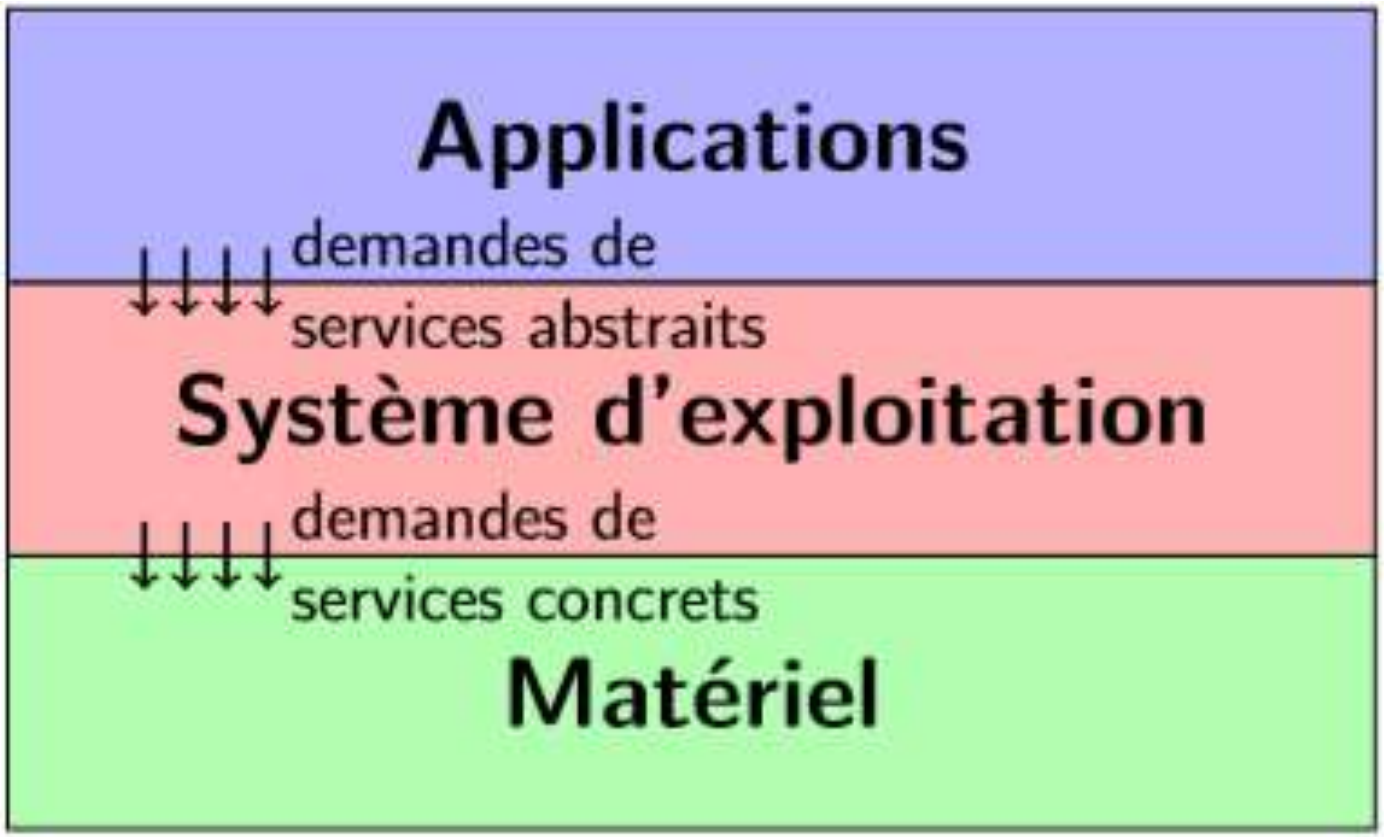


Où se situe le système d'exploitation

Abstrait la machine :

- Cache certains détails que l'utilisateur n'a pas à connaître pour exploiter la machine
- Présente à l'utilisateur une machine virtuelle facile à utiliser et à programmer
- Offre des services abstraits
- Connait les détails internes et garantie l'utilisation des services concrets de la machine

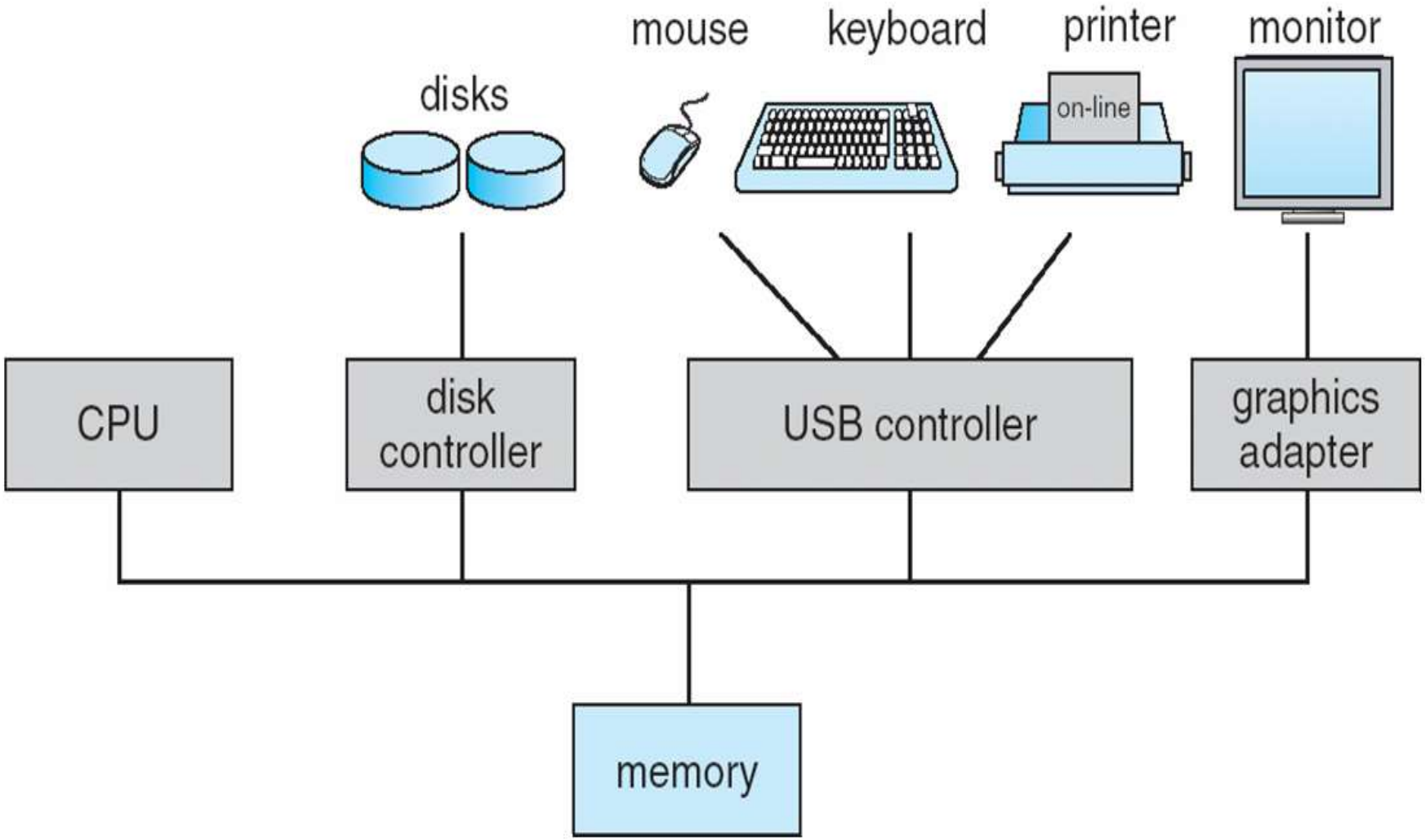
Où se situe le système d'exploitation



Services offert par un OS

1. **Facilité de la création de programme**
2. **Editeurs, compilateurs, débogueurs, etc.**
3. **Exécution du programme**
4. **Chargement en mémoire,**
5. **Gestion des E / S et initialisation de fichier.**
6. **Accès aux E / S et aux fichiers**
7. **Traitement des spécificités des E / S et des formats de fichiers.**
8. **Accès au système**
9. **Résolution des conflits de ressources.**
10. **Protection de l'accès aux ressources et aux données.**

Organisation matériel de l'ordinateur



Composants d'un système informatique

1. Matériel : fournit des ressources informatiques de base (CPU, mémoire, périphériques E / S, communication).
2. Système d'exploitation : contrôle et coordonne l'utilisation du matériel parmi divers programmes d'application pour différents utilisateurs.
3. Programmes système et application : utilisation des ressources système pour résoudre les problèmes informatiques des utilisateurs (traitements de texte, compilateurs, navigateurs Web, systèmes de base de données, jeux vidéo).
4. Utilisateurs - Personnes, Machines, autres ordinateurs.

Vues générales d'un Système d'exploitation 1

Il y a trois vues classiques (dans la littérature) des SE :

- Gestionnaire de ressource : gère et alloue des ressources.
- Programme de contrôle : contrôle l'exécution des programmes utilisateur et des opérations des périphériques d'E / S.
- Exécuteur de commande : Fournit un environnement pour l'exécution des commandes utilisateur.
- Une Nouvelle vision plus moderne : le système d'exploitation en tant que machine virtuelle.

Vues générales d'un Système d'exploitation 2

Gère et protège plusieurs ressources informatiques: CPU, processus, mémoire interne / externe, tâches, applications, utilisateurs, canaux de communication, etc.

Gère et alloue des ressources à plusieurs utilisateurs ou à plusieurs programmes exécutés en même temps et dans le même espace (par exemple, temps processeur, mémoire, périphériques d'E / S).

Décide entre des demandes contradictoires pour une utilisation efficace et équitable des ressources (par exemple, maximiser le débit, minimiser le temps de réponse).

.

Vues générales d'un Système d'exploitation 3

- Contrôle l'exécution des programmes utilisateur et des périphériques d'E / S pour éviter les erreurs et l'utilisation incorrecte des ressources informatiques.

- Surveille et protège l'ordinateur: moniteur, superviseur, cadre, contrôleur, maître, coordinateur….

Vues générales d'un Système d'exploitation 4

Exécuteur de Commande :

Interfaces entre les utilisateurs et la machine.

Fournit des services / utilitaires aux utilisateurs.

Fournit aux utilisateurs une interface de ligne de commande (CLI) pratique, également appelée shell (sous UNIX), pour saisir les commandes utilisateur

Vues générales d'un Système d'exploitation

Machine virtuelle : Une interface entre l'utilisateur et le matériel qui masque les détails du matériel (par exemple, les E / S).

Construit des ressources (virtuelles) de niveau supérieur à partir de ressources (physiques) de niveau inférieur (par exemple, des fichiers).

Définition: OS est un ensemble d'améliorations logicielles exécutées sur le matériel, aboutissant à une machine virtuelle de haut niveau qui sert d'environnement de programmation avancé.

Caractéristiques d'un Système d'exploitation

•**Systèmes multi-tâches :** permettent l'exécution de plusieurs tâches à la fois, tel que : exécuter le programme d'un utilisateur, lire les données d'un disque ou afficher des résultats sur une imprimante.

•**Systèmes Multi-utilisateurs** : capable d'exécuter de façon concurrente et indépendante des applications appartenant à plusieurs utilisateurs.

Remarque : Un système multi-utilisateurs est nécessairement multi-tâches mais la réciproque est fausse (le système d'exploitation MS-DOS est mono-utilisateur et mono-tâche ; Windows 7 est mono-utilisateurs mais multi-tâches ; Unix et Windows NT sont multi- utilisateurs.

Structure externe d'un Système d'exploitation 1

Point de vue de l'interface présentée à l'utilisateur et au programmeur :

- Noyau et utilitaires
- Gestionnaire des tâches
- Gestionnaire de la mémoire
- Gestionnaire des fichiers
- Gestionnaire de périphérique
- Chargeur du système d'exploitation
- Interpréteur de commande

Structure externe d'un Système d'exploitation 2 : Le noyau

- Le système d'exploitation comporte un certain nombre de routines (sous-programmes). Les plus importantes constituent le noyau (kernel en anglais).
- Il est chargé en mémoire vive à l'initialisation du système et contient de nombreuses procédures nécessaires au bon fonctionnement du système. Les autres routines, moins critiques, sont appelées des utilitaires.
- Ce noyau d'un système d'exploitation se compose de quatre parties principales : les gestionnaire de tâches (ou des processus), de mémoire, de fichiers et de périphériques d'entrée-sortie.
- Il possède également deux parties auxiliaires : le chargeur du système d'exploitation et l'interpréteur de commandes.

Structure Externe d'un SE : Le gestionnaire de tâches

Sur un système à temps partagé, l'une des parties les plus importantes du système d'exploitation est le gestionnaire de tâches ou ordonnanceur (Sur un système à un seul processeur, il divise le temps en laps de tps).

Périodiquement, le gestionnaire de tâches décide d'interrompre le processus en cours et de démarrer (ou reprendre) l'exécution d'un autre, soit parce que le premier a épuisé son temps d'allocation du processus soit qu'il est bloqué (en attente d'une donnée d'un des périphériques).

Le contrôle de plusieurs activités parallèles est un travail difficile. C'est pourquoi les concepteurs des systèmes d'exploitation ont constamment, au fil des ans, amélioré le modèle de parallélisme pour le rendre plus simple d'emploi.

Structure Externe d'un SE : Le gestionnaire de la mémoire

La mémoire est une ressource importante qui doit être gérée avec prudence. Le moindre micro-ordinateur a, dès la fin des années 1980, dix fois plus de mémoire que l'IBM 7094, l'ordinateur le plus puissant du début des années soixante. Mais la taille des programmes augmente tout aussi vite que celle des mémoires.

La gestion de la mémoire est la tâche du gestionnaire de mémoire. Celui-ci doit connaître les parties libres et les parties occupées de la mémoire, allouer de la mémoire aux processus qui en ont besoin, récupérer la mémoire utilisée par un processus lorsque celui-ci se termine et traiter le va-et-vient (swapping en anglais) entre le disque et la mémoire principale lorsque cette dernière ne peut pas contenir tous les processus.

Structure Externe d'un SE : Le gestionnaire des fichiers

Une des tâches fondamentales du système d'exploitation est de masquer les spécificités des disques et des autres périphériques d'entrée-sortie et d'offrir au programmeur un modèle agréable et facile d'emploi. Ceci se fait à travers la notion de fichier.

Structure Externe d'un SE : Le gestionnaire des périphériques

Le contrôle des périphériques d'entrée-sortie (E/S) de l'ordinateur est l'une des fonctions primordiales d'un système d'exploitation. Ce dernier doit :

Envoyer les commandes aux périphériques, intercepter les interruptions, et traiter les erreurs.

Fournir une interface simple et facile d'emploi entre les périphériques et le reste du système qui doit être, dans la mesure du possible, la même pour tous les périphériques, c'est-à-dire indépendante du périphérique utilisé.

Le code des entrées-sorties représente une part importante de l'ensemble d'un système d'exploitation.

Structure Externe d'un SE : Le chargeur du SE

En général, de nos jours, lorsque l'ordinateur (compatible PC ou Mac) est mis sous tension, il exécute un logiciel appelé BIOS (pour Basic Input Output System) placé à une adresse bien déterminée et contenu en mémoire ROM. Ce logiciel initialise les périphériques, charge un secteur d'un disque, et exécute ce qui y est placé.

Lors de la conception d'un système d'exploitation, on place sur ce secteur le chargeur du système d'exploitation ou, plus exactement, le chargeur du chargeur du système d'exploitation (ou pré-chargeur) puisque le contenu d'un secteur est insuffisant pour le chargeur lui-même.

La conception du chargeur et du pré-chargeur est indispensable, même si ceux-ci ne font pas explicitement partie du système d'exploitation.

Structure Externe d'un SE : L'interpraiteur de commande

Le système d'exploitation proprement dit est le code qui permet de définir les appels système.

Les programmes système tels que les éditeurs de texte, les compilateurs, les assembleurs, les éditeurs de liens et les interpréteurs de commandes ne font pas partie du système d'exploitation. Cependant l'interpréteur de commandes (shell en anglais) est souvent considéré comme en faisant partie.

Sous sa forme la plus rudimentaire, l'interpréteur de commandes exécute une boucle infinie qui affiche une invite (montrant par là que l'on attend quelque chose), lit le nom du programme saisi par l'utilisateur à ce moment-là et l'exécute.

Exemples de SE

Microsoft Windows (dernière version : Windows 10) : les systèmes d'exploitation de Microsoft sont actuellement préinstallés sur plus de 91 % des ordinateurs personnels

Dérivés d'Unix (sous différentes déclinaisons : BSD, System V, etc.) dont :

macOS et iOS (ex-iPhone OS) : systèmes pré-installés sur la majorité des ordinateurs et appareils mobiles vendus par Apple.

GNU/Linux : que nous allons voir dans la suite de ce cours

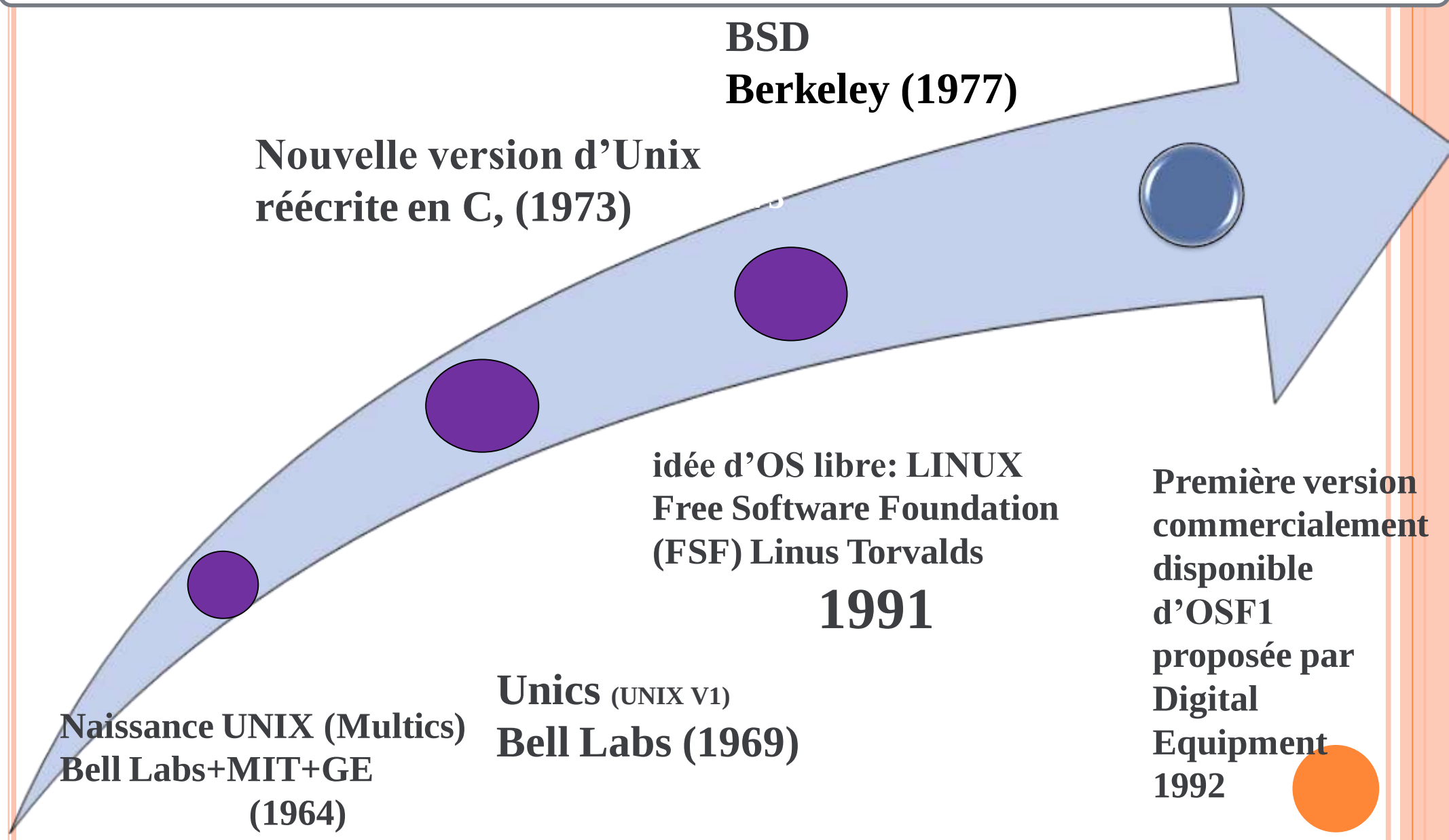
Les Unix dits « propriétaires » : de nombreux systèmes d'exploitations propres à des environnement machine,

Exemple : A/UX (Apple, systemV), HP-UX (Hewlett-Packard, SystemV), Sinix (Siemens), Solaris (Sun, SystemV), SunOS (Sun, BSD).

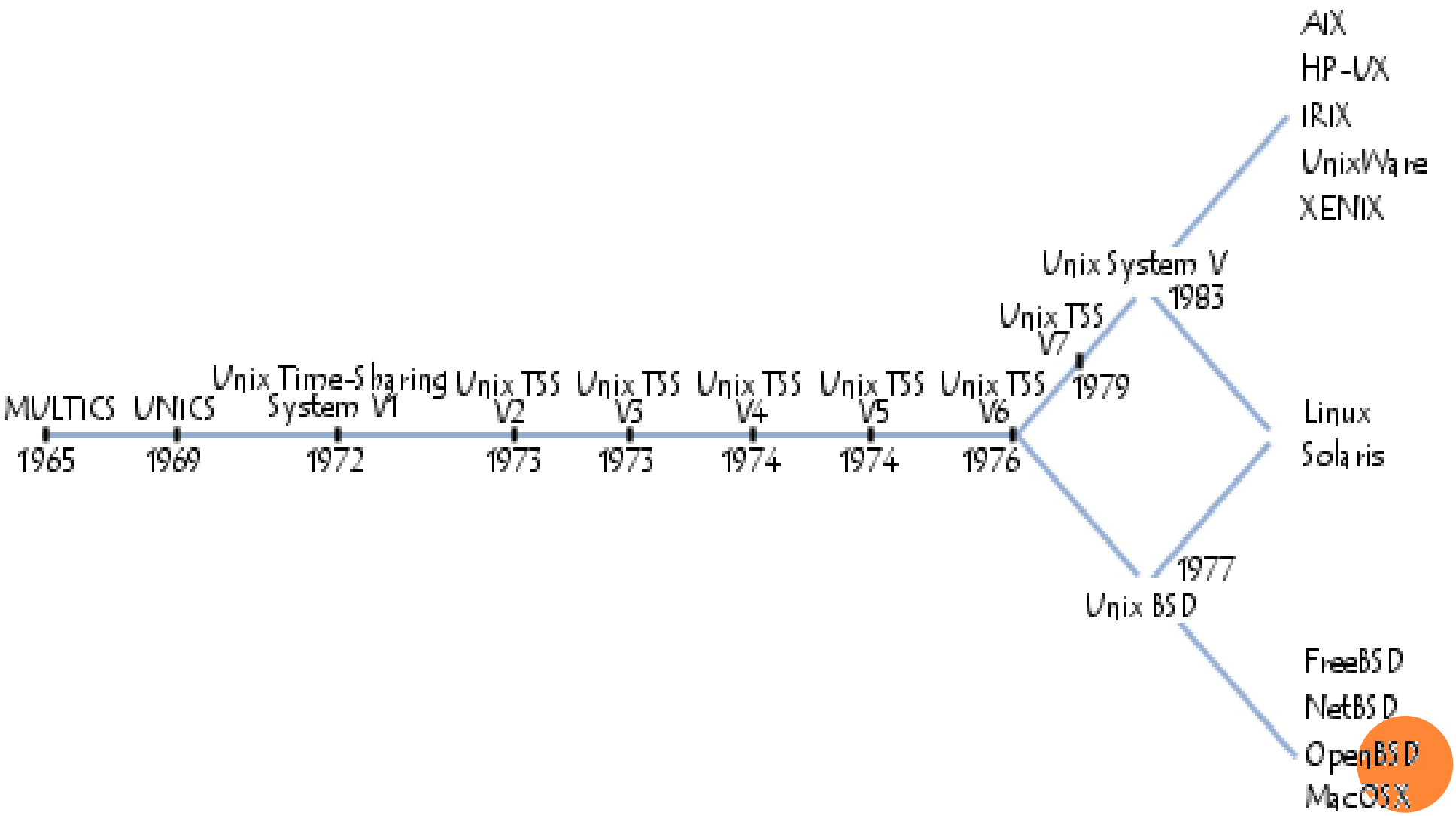
LE SYSTÈME LINUX (UNIX)

Chapitre 2
Présentation générale de LINUX

Un peu d'histoire



Evolution des SE



Points forts

logiciel libre :

- ✓ Linux n'appartient à personne
- ✓ Un projet communautaire
- ✓ Système extrêmement économique.

système sécuritaire

système hautement configurable

fiabilité et sa robustesse

Alternative au monopole Microsoft

Logiciel libre

Un logiciel libre est un logiciel dont l'utilisation, l'étude, la modification et la duplication en vue de sa diffusion sont permises, techniquement et légalement. Ceci afin de garantir certaines libertés induites, dont le contrôle du programme par l'utilisateur et la possibilité de partage entre individus.

Liberté 0 : La liberté d'exécuter le programme, pour tous les usages.

Liberté 1 : La liberté d'étudier le fonctionnement du programme, et de l'adapter à des besoins. Pour cela, l'accès au code source est une condition requise.

Liberté 2 : La liberté de redistribuer des copies, donc d'aider son voisin.

Liberté 3 : La liberté d'améliorer le programme et de publier ces améliorations

Licence GPL

La GPL met en œuvre la notion de *copyleft*, un jeu de mots anglais faisant référence à la notion de copyright

copyleft **copyright**

En français en parlant de « *Gauche d'auteur* » par référence au Droit d'auteur.

Libre n'est pas gratuit

- **Le logiciel libre est souvent confondu à tort avec les freewares (gratuiciels) ;**
- **les logiciels gratuits ne sont pas nécessairement libres, car leur code source n'est pas systématiquement accessible, et leur licence peut ne pas correspondre à la définition du logiciel libre ;**
- **l'open source : le logiciel libre, selon son initiateur, est un mouvement social qui repose sur les principes de Liberté, Égalité, Fraternité.**

Le mouvement GNU

- Au début des années 1980, les systèmes Unix étaient encore, propriétaires
- Cette situation était frustrante pour les étudiants et techniciens qui ne pouvaient s'offrir une licence
- Le besoin a suscité des initiatives alternatives, dont la première fut en 1983 avec le lancement du projet GNU par Richard Stallman (dit « RMS »).
- En 1984, ce dernier a créé la Free Software Foundation (FSF, fondation du logiciel libre), cadre juridique au projet GNU.
- **L'objectif était** : d'écrire un système Unix complet en repartant de zéro, de manière compatible avec les systèmes existants, et sous forme de logiciel libre
- RMS a rapidement été rejoint par des collaborateurs et volontaires du monde entier.
- La concrétisation en est la publication en 1989 de la première version de la licence **GPL** qui sera alors le fondement éthique, juridique et politique du mouvement du Libre.

GNU / LINUX

- Le projet GNU arrive en 1991 avec de très nombreux outils libres, mais il lui manque un élément central : le **noyau**.

Cet élément est essentiel car il gère la mémoire, le microprocesseur, les périphériques comme le clavier, la souris, les disques durs.

- à cette époque un étudiant finlandais, *Linus Torvalds*, commence à développer un noyau et demande aux personnes intéressées d'y contribuer.
- La licence GPL a été publiée à la même époque et Linus Torvalds s'est laissé persuader de placer son noyau sous cette dernière.
- Le système d'exploitation est donc un assemblage des outils GNU fonctionnant sur un noyau Linux, on parle donc de GNU/Linux
- GNU/Linux est un système d'exploitation complètement Libre, performant, hautement configurable et ne dépend d'aucune multinationale.
- Dès 1993, les premières solutions complètes intégrant le noyau Linux et le système GNU, ainsi que quelques applicatifs, ont vu le jour.

Les distribution LINUX

Linux est le plus souvent diffusé sous forme d'une distribution, un ensemble de programmes (noyau, sources des utilitaires, commandes, applications) formant après installation un système complet.

POURQUOI AUTANT DE DISTRIBUTIONS ?

Chaque distribution a sa cible, certaines sont orientées :

- sur la facilité d'utilisation,
- pour l'utilisation dans le domaine scolaire ou musical,
- fonctionnalités optimisées pour les netbooks et leur écran réduit,
- d'autres encore se veulent très légères et fonctionnent sur des PC.

Il y a autant de distributions que de cas d'utilisation !

UBUNTU

- Ubuntu Linux est une distribution GNU/Linux non commerciale basée sur Debian et lancée en 2004.

- Son nom provient d'un ancien mot bantou (langue d'Afrique), Ubuntu, signifiant «humanite aux autres» ou encore «je suis ce que je suis grâce a ce que nous sommes tous».

- Initiée par le milliardaire sud-africain Mark Shuttleworth, et sponsorisée par sa société Canonical Ltd., Ubuntu Linux est conçue principalement pour les ordinateurs de bureau (PC et Macintosh) avec un objectif de convivialité et d'ergonomie.

UBUNTU

- Edition standard
- Pour ultraportable

variante pour l'éducation

un bureau KDE paramétrable à souhait

La création multimédia

variante légère (<384Mo RAM)

l'usage interne des employés Google

centre multimédia et enregistreur TV

UBUNTU

- La numérotation des versions de Ubuntu est basée sur l'année et le mois de sa sortie [A.MM].
- La première version de Ubuntu, sortie en octobre 2004, portait le numéro de version 4.10.
- La version suivante, sortie en avril 2005, portait le numéro 5.04.
- La suivante, la 5.10, était sortie en octobre 2005.
- Chaque version de Ubuntu a une combinaison unique de ses composantes - le noyau, le serveur graphique X11, l'environnement de bureau GNOME, GCC, libc... - qui ont toutes des numéros de version différents et n'ayant pas tous la même signification.

UBUNTU

Contrairement à d'autres distributions Linux, lorsqu'une version de Ubuntu est stabilisée, les versions des logiciels qu'elle inclut sont gelées.

Ainsi, si une nouvelle version stable d'un logiciel ou d'une bibliothèque quelconque sort après la stabilisation de Ubuntu, l'intégration de cette nouvelle version a Ubuntu se produira dans la prochaine mouture de l'OS.

Cette manière de procéder assure une meilleure homogénéité des versions pour du support technique de la part de Canonical Ltd (est une société fondée par l'entrepreneur sud-africain Mark Shuttleworth dont l'objet est la promotion de projets à source libre). et ses partenaires;

Cette caractéristique assure que le système, dans sa version actuelle, reste stable et fonctionnel.

Les seules mises a jour publiées pour les versions stables sont des mises a jour de sécurité, corrigeant bogues, failles et autres problèmes de fonctionnement de l'actuelle version.

UBUNTU

Des versions stables de Ubuntu sortent deux fois par année, aux mois d'avril et d'octobre.

Le développement de Ubuntu est lié au développement de l'environnement de bureau GNOME: la version finale de Ubuntu sort environ un mois après la publication d'une nouvelle version stable de GNOME.

Ubuntu suit donc un cycle de développement de six mois.

Pré-requis d'installation

Avant de commencer à utiliser Ubuntu, vous aurez besoin de vous procurer une copie de l'image d'installation d'Ubuntu au format DVD ou USB.

Configuration minimale

- **Processeur 1GHz x86 (Pentium 4 ou mieux) ;**
- **1Go de mémoire (RAM) ; de préférence 2 Go**
- **5 Go d'espace disque (au moins 15 Go est recommandé) ;**
- **Vidéo ayant une capacité de résolution de 1024×768 ;**
- **Support audio ;**
- **Une connexion internet (fortement recommandée, mais pas obligatoire)**

Installation du système

Choisissez la manière dont vous voulez installer Ubuntu :

- Version « live » sans installation pour essayer Linux
- Installer un seul système d'exploitation sur la partition
- Installer Linux comme un simple programme sur une Partition Windows
- Faire cohabiter les deux systèmes sue une machine et choisir durant le boot

NB: 32 bits versus 64 bits

Installation d'UBUNTU

Installer Linux en version live
Créer une clé USB Linux ou un CD linux

1. Télécharger l'image du CD d'installation d'ubuntu à partir du site web :

   http://www.ubuntu-fr.org/telechargement

2. Graver l'image du CD d'installation sur une clé USB ou un CD :

   - Télécharger LinuxLive USB Creator (depuis http://www.linuxliveusb.com/fr/download) si ce n'est pas déjà fait et l'installer.
   - Lancer LinuxLive USB Creator depuis votre menu Démarrer -> Tous les programmes -> LinuxLive USB Creator.

Installation d'UBUNTU

Installer Linux en version live
Créer une clé USB Linux ou un CD linux

- étape 1 : choisir une clé USB dans la liste.
- étape 2 : sélectionner un fichier ISO ou un CD.
- étape 3 : choisir la taille des données persistentes (habituellement entre 250 Mo et 2 Go).
- étape 4 : cocher les options que vous désirez.
- étape 5 : cliquer sur l'éclair pour démarrer la création.

INSTALLER UBUNTU EN DUAL BOOT

installer Ubuntu tout en **conservant Windows**.

Au démarrage de l'ordinateur on aura le **choix** entre Windows (XP, Vista ou Seven …) ou Ubuntu.

C'est ce que l'on appelle un **Dual Boot** et c'est une technique qui permet de faire **cohabiter 2 systèmes d'exploitation** sur un même ordinateur.

Installation d'UBUNTU

PRÉPARER UNE PARTITION

1ère étape

Après avoir un CD d'installation prêt.

2ème étape

Il faut préparer le disque dur pour libérer un peu d'espace pour ubuntu (au moins 10 Go) :

⇒**Défragmenter** le disque dur

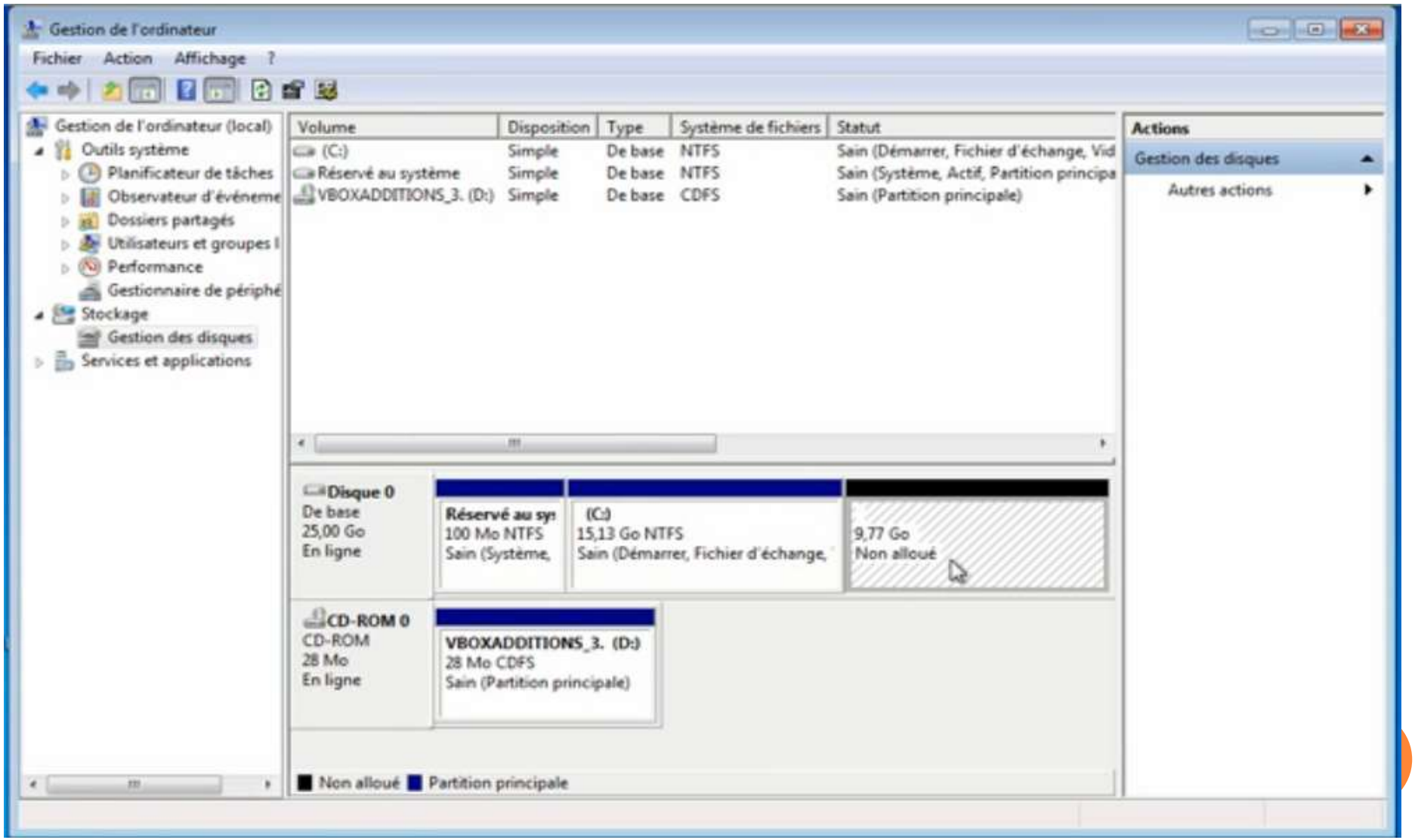
clique droit sur C:/Propriété/outils/défragmenter maintenant).

⇒**Réduire** la partition de Windows afin de laisser de l'espace pour ubuntu.

1. Aller dans : démarrer, clique droit sur « ordinateur » ou « poste de travail » selon la version de Windows puis cliquer sur « gérer ».
2. Aller dans **stockage/gestion des disques** puis réduire la partition windows en faisant un clic droit dessus puis « réduire ».
3. Prévoir une dizaine de gigas.

Installation d'UBUNTU

Préparer la partition



Installation d'UBUNTU

CHANGER L'ORDRE DU BOOT

3ème étape

Insérer le CD dans le lecteur puis redémarrer l'ordinateur.

Lorsqu'il démarre faites bien attention aux inscriptions qui se situe en bas de l'écran lorsque le bios se lance.

Il faut repérer sur quelle touche appuyer, généralement c'est « suppr » ou « F12 » mais ça change d'un bios à l'autre et il en existe plusieurs (ca dépend de la marque de la carte mère).

Une fois dans le bios, il faut changer l'ordre de démarrage (ordre de boot) afin de placer le lecteur CD en premier sur la liste.

Encore une fois, selon les types de bios la procédure change.

Installation d'UBUNTU

INSTALLATION DU SYSTÈME: BOOTER SUR USB OU CD

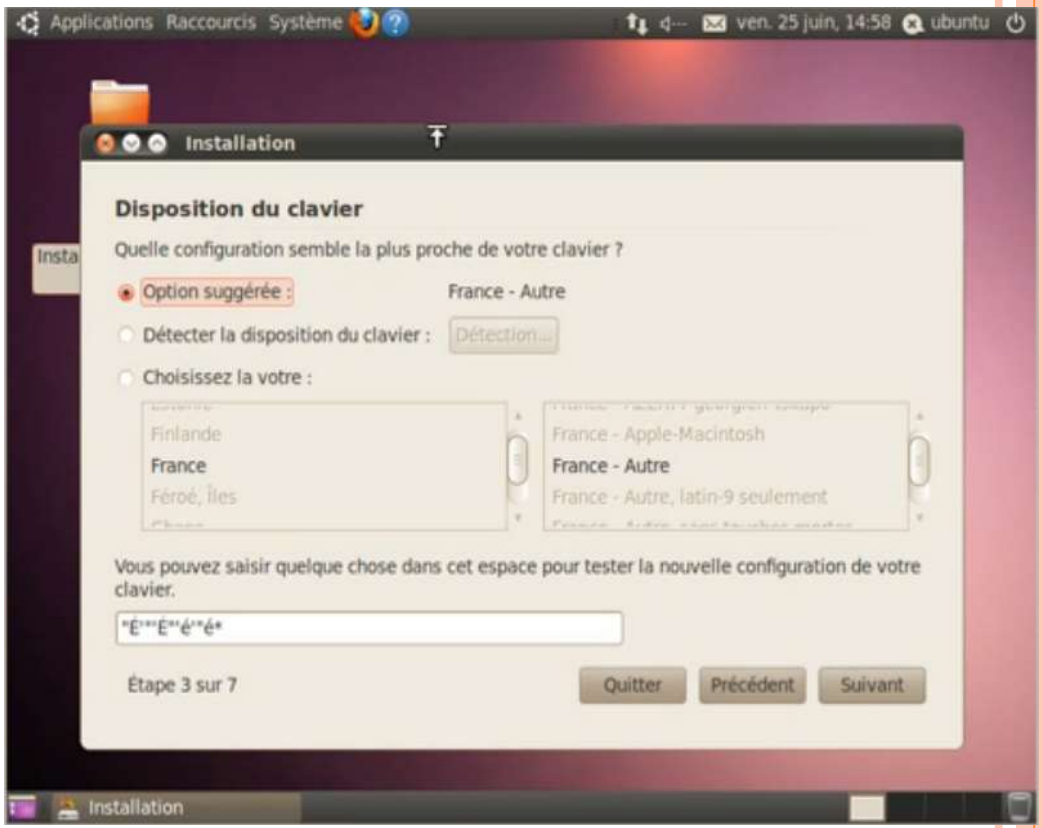
4ème étape

L'écran de « Bienvenue » pour choisir la langue

Installation d'UBUNTU

CHOIX DU FUSEAU HORAIRE ET CLAVIER

5



6

Installation d'UBUNTU

PARTITIONNEMENT MANUEL

5ème étape :

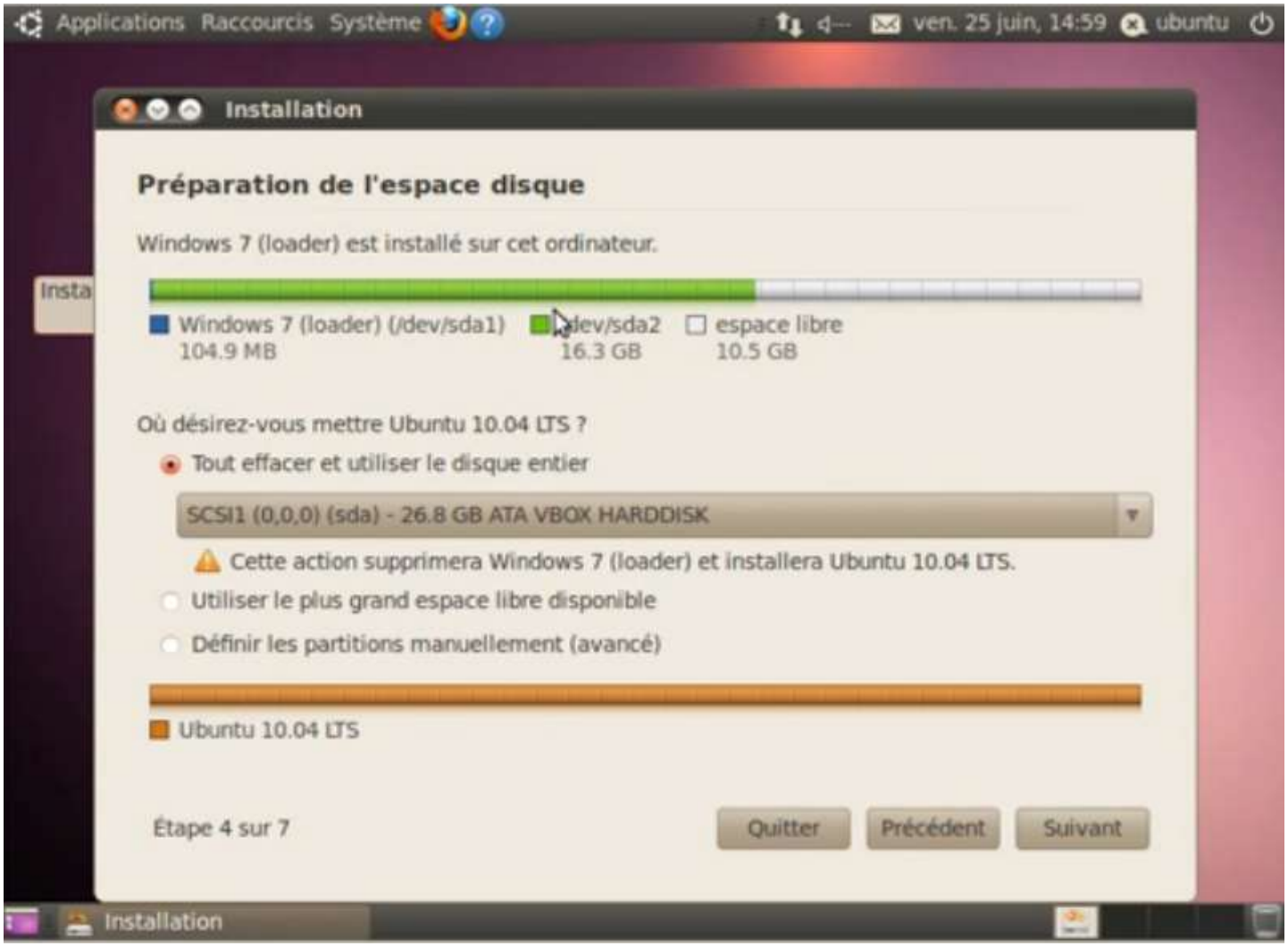
Le partitionnement manuel permet de choisir quelles partitions doivent être créées, réutilisées, formatées et supprimées.

Ubuntu peut techniquement être installe avec une seule partition (la partition racine [/]); pour un meilleur fonctionnement et pour plus de sécurité, nous il est conseillé de créer au moins les trois partitions suivantes:

- Une **partition racine** [point de montage: /] dans laquelle s'installeront les utilitaires et services du système d'exploitation, ainsi que tous les programmes;
- Une **partition swap** [aucun point de montage], qui sert d'extension a la mémoire vive de l'ordinateur. On suggère que cette partition ait une taille de : 1,5 a 2 fois celle de la capacité en RAM de l'ordinateur (ex: si l'ordinateur dispose de 512 Mo de RAM, alors la swap devrait avoir une taille entre 768 Mo et 1024 Mo);
- Une **partition utilisateurs** [point de montage: /home] dans laquelle seront contenus les fichiers des utilisateurs (documents texte, films, fichiers audio, etc.) ainsi que les paramètres personnels des utilisateurs.

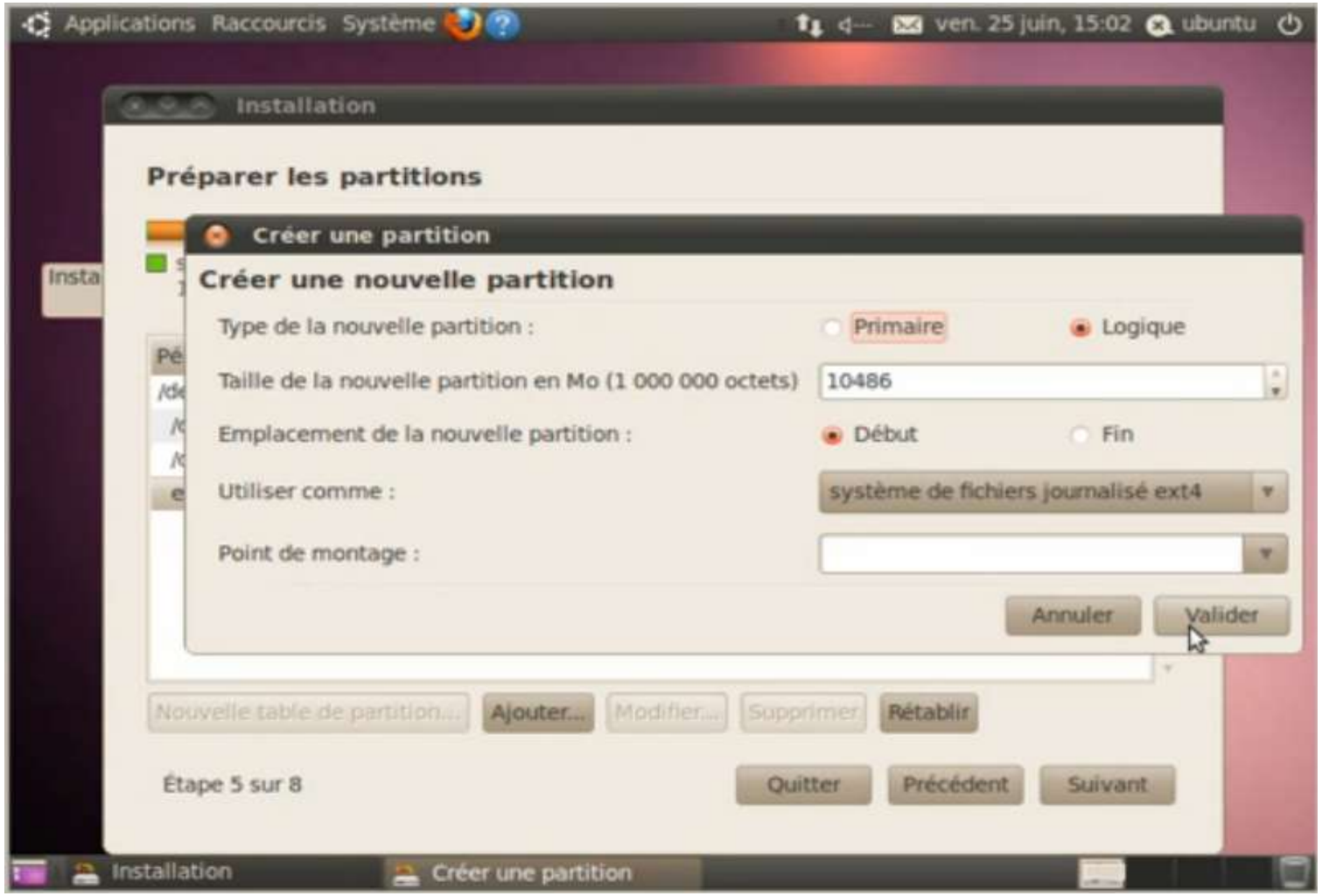
Installation d'UBUNTU

CRÉER DES PARTITIONS



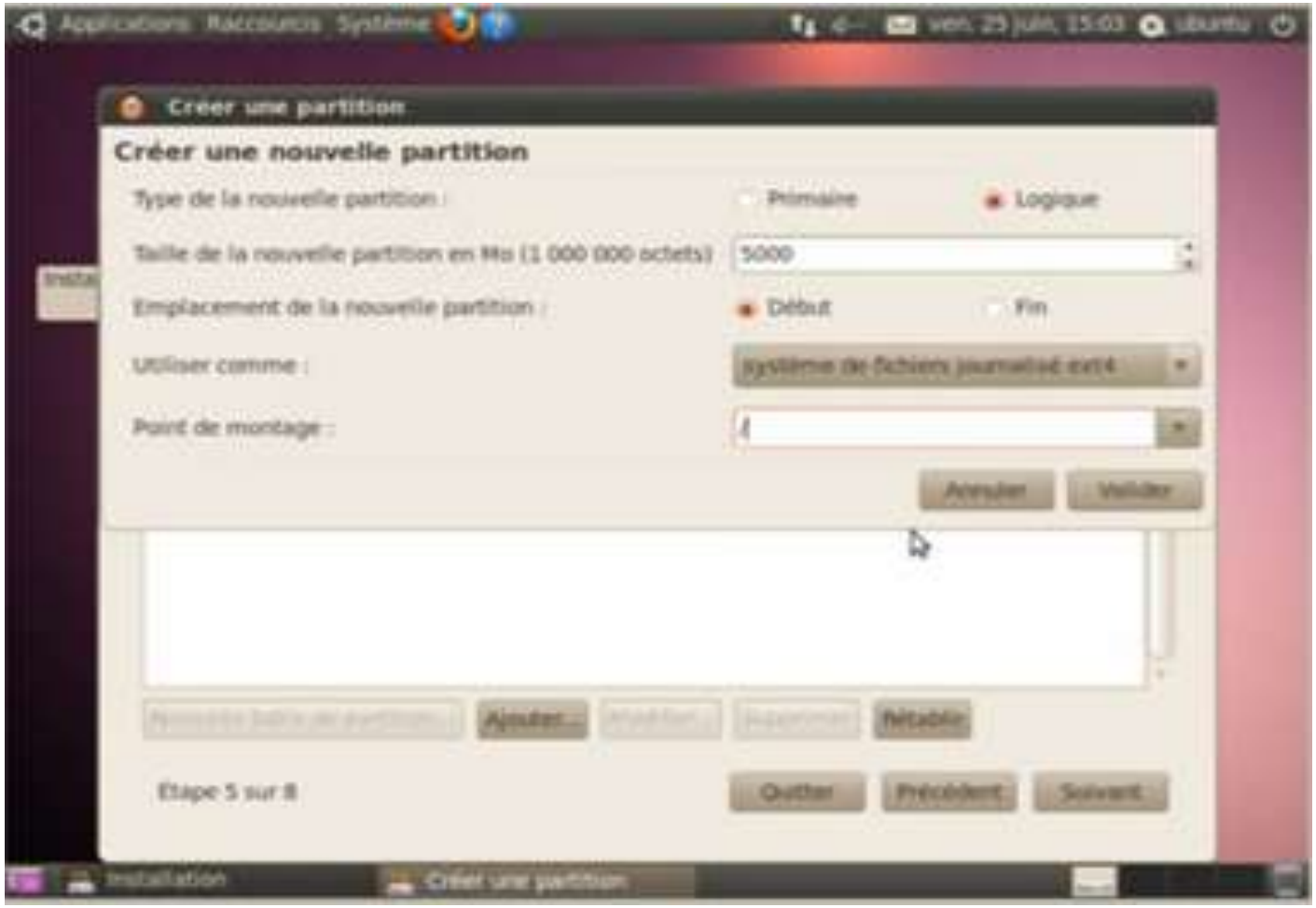
Installation d'UBUNTU

Créer une Partition noyau



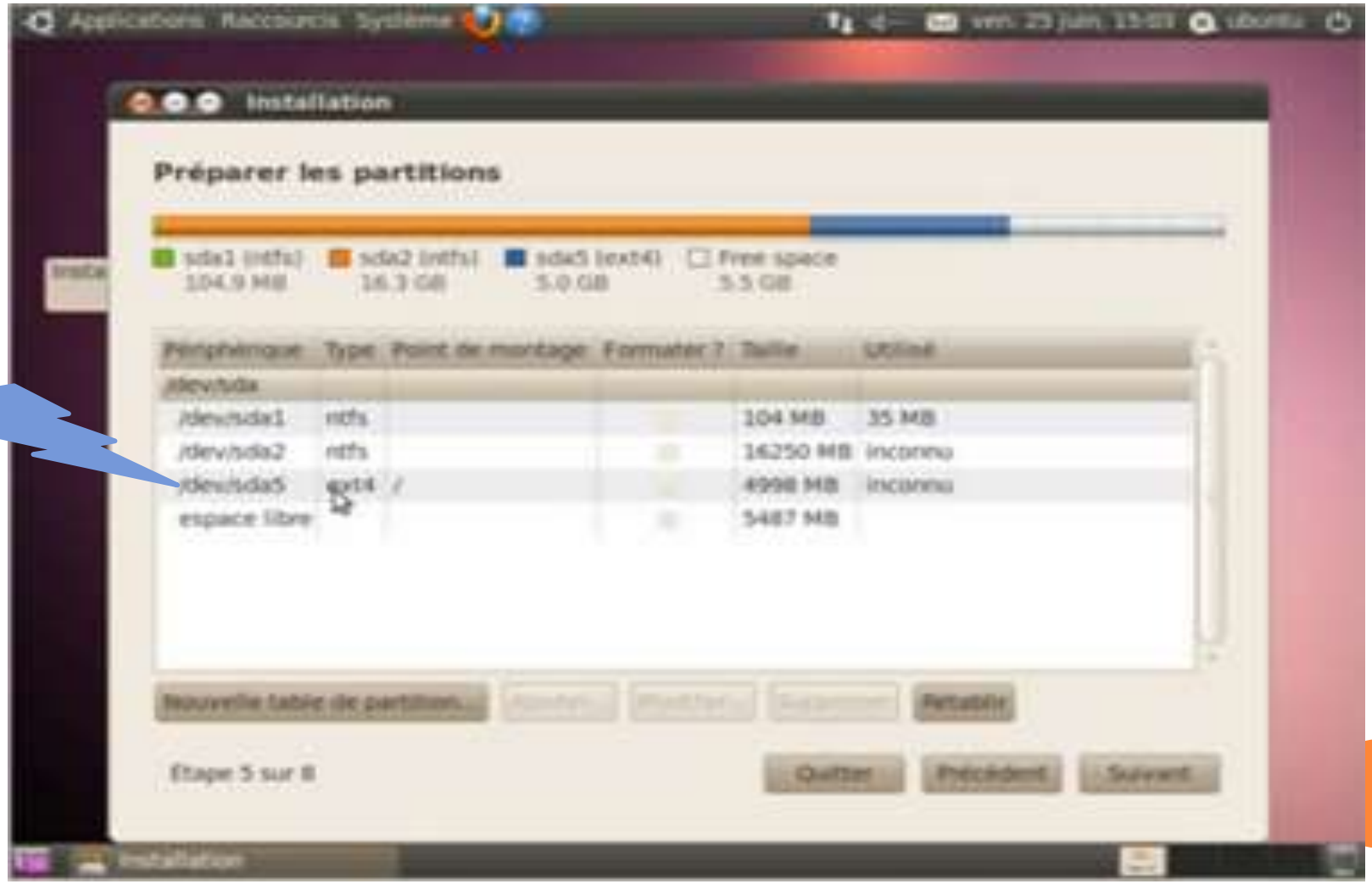
Installation d'UBUNTU

CRÉER UNE PARTITION NOYAU



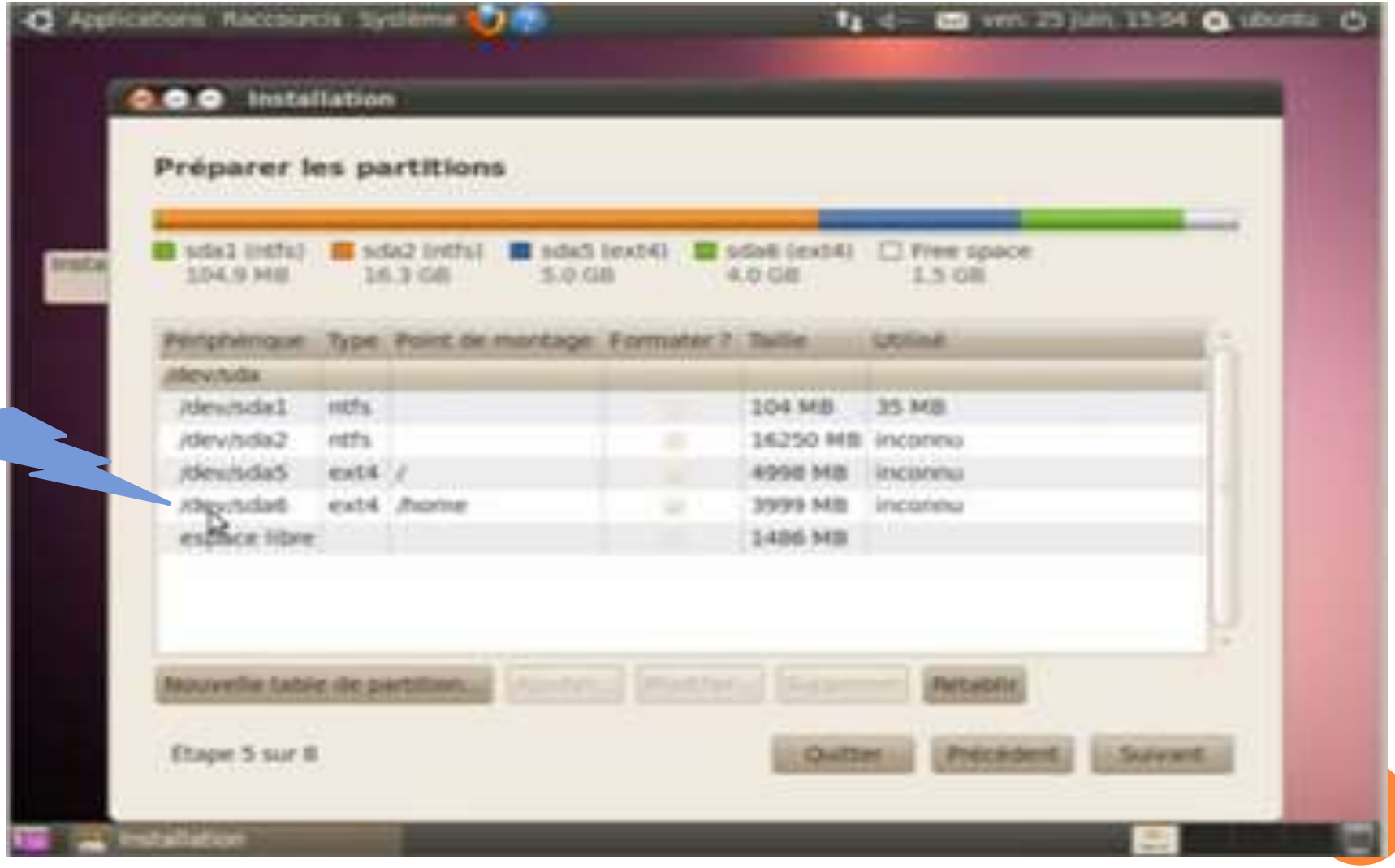
Installation d'UBUNTU

Créer une Partition noyau



CRÉER UNE PARTITION ESPACE PERSONNELLE

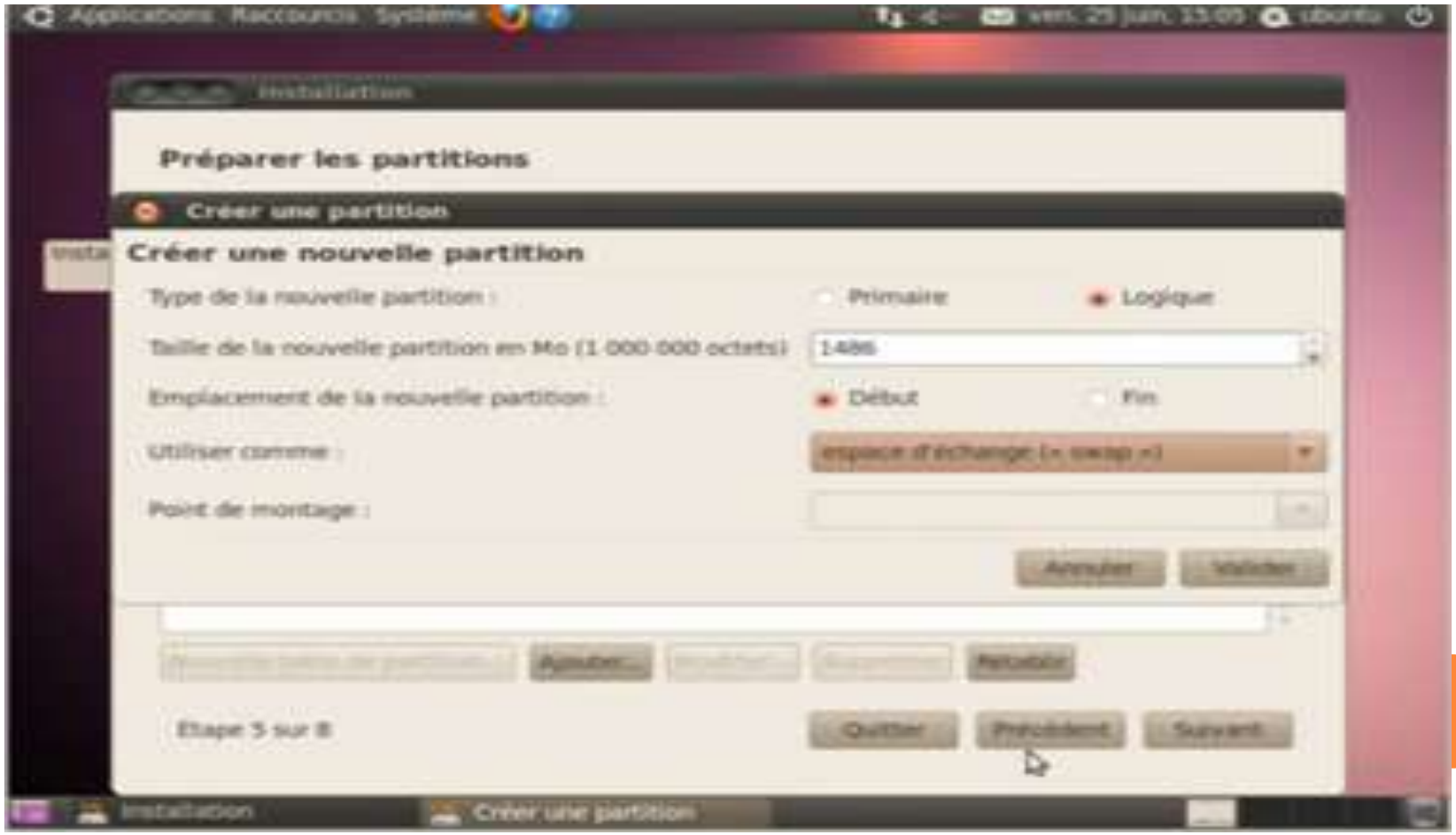
Installation d'UBUNTU



Installation d'UBUNTU

CRÉATION D'UN ESPACE SWAP

pas plus de 4 Gbit, la moitié de la ram, utiliser le disque dur quand il n'y a pas d'espace ram.



Installation d'UBUNTU

CRÉATION D'UN MOT DE PASSE

Installation d'UBUNTU

SUITE DE L'INSTALLATION

Installation d'UBUNTU

INSTALLATION DU SYSTÈME: VIRTUAL MACHINE

1. Télécharger l'image du CD d'installation d'ubuntu à partir du site web :

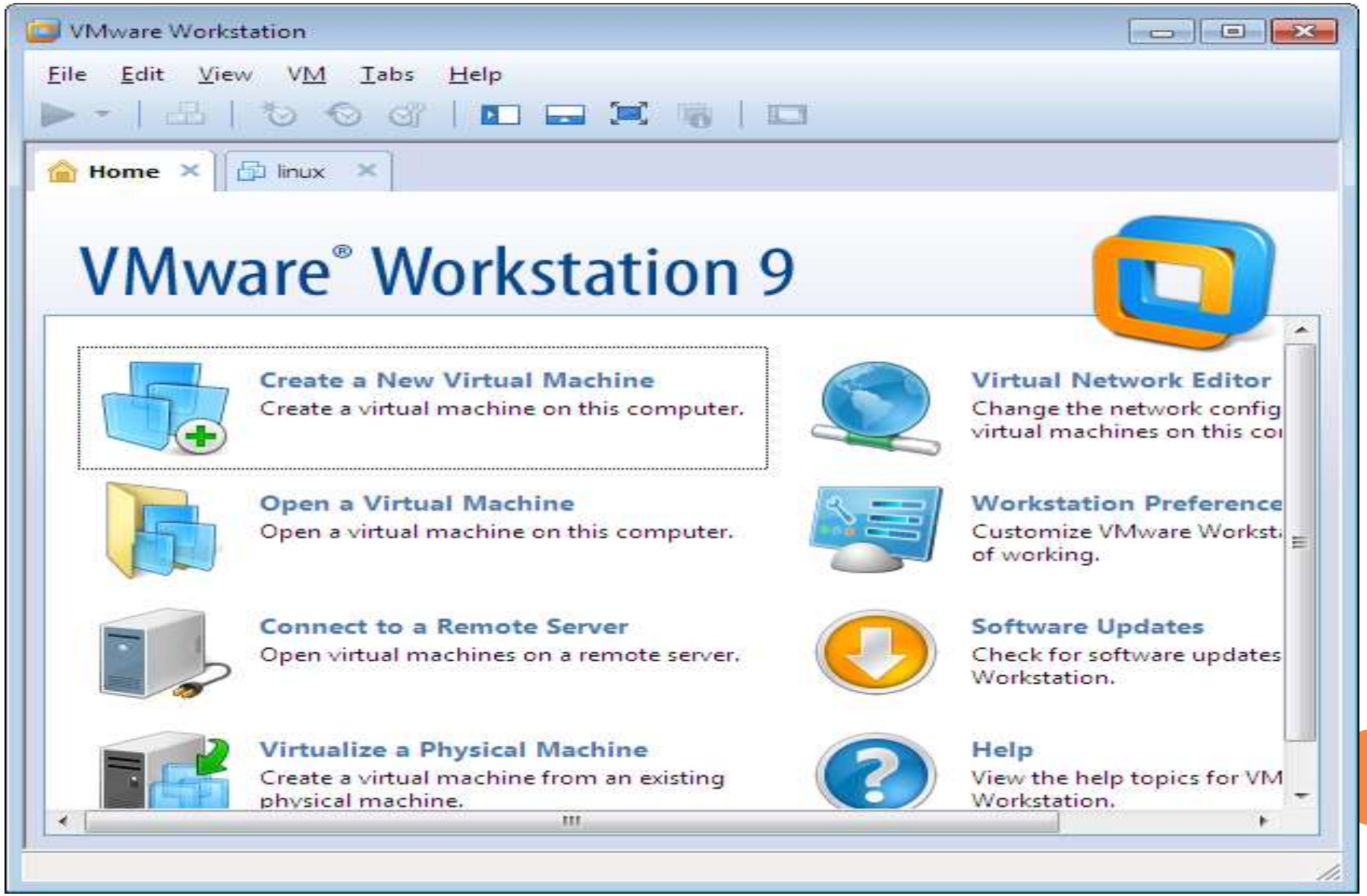
   http://www.ubuntu-fr.org/telechargement

2. Télécharger VMware Workstation: permet de créer une machine virtuelle

3. Installer le logiciel

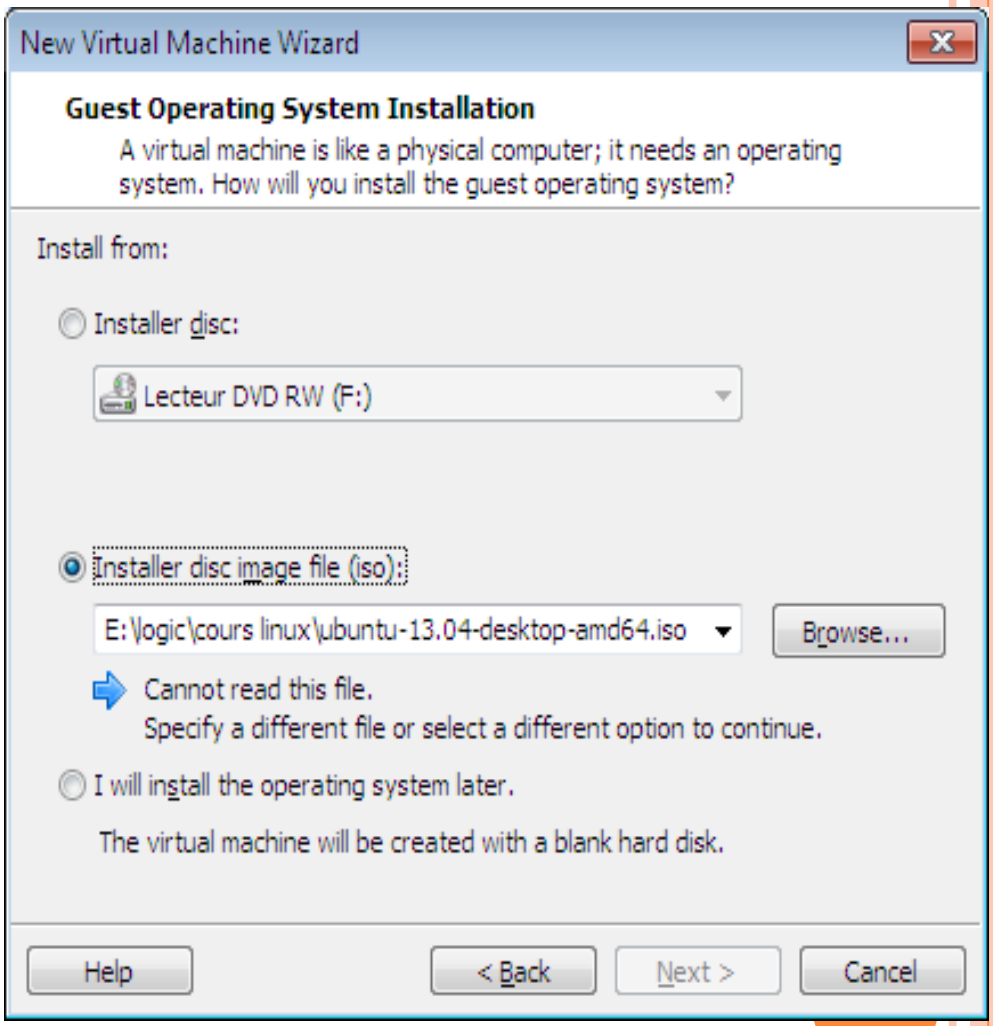
Installation d'UBUNTU

INSTALLATION DU SYSTÈME: VIRTUAL MACHINE



Installation d'UBUNTU

INSTALLATION DU SYSTÈME: VIRTUAL MACHINE



Suivre les étapes

Récapitulatif

Quelle est la différence entre Linux et les systèmes d'exploitation UNIX?

• UNIX est protégé par un copyright, et Seules les grandes entreprises sont autorisées à utiliser le nom UNIX (IBM AIX, Sun Solaris et HP-UX sont tous des systèmes d'exploitation UNIX)

• **Linux est un clone d'UNIX :** Linux est un clone d'Unix écrite à partir de zéro par Linus Torvalds avec l'aide d'une équipe de collaborateurs à travers le monde.

• **Linux est un noyau :** Linux est juste un noyau entouré par plusieurs applications et systèmes de gestion, (GNU / LINUX)

• Les systèmes d'exploitation UNIX sont considérés comme des systèmes d'exploitation complets.

Récapitulatif

Quelle est la différence entre Linux et les systèmes d'exploitation UNIX?

- Linux est gratuit .
- On peut le télécharger à partir d'Internet ou de le redistribuer sous licences GNU.
- La plupart des systèmes UNIX d'exploitation ne sont pas libres .
- Linux est considéré comme le plus **convivial** des UNIX.

RÉCAPITULATIF : INSTALLATION

- En version live
- En dual boot
- Sur une machine virtuelle
- Linux uniquement

Organisation générale du système

Le système Unix est organisé en couches :

• Noyau : la couche de plus haut niveau, elle assure la communication
avec le matériel. Le noyau s'occupe de :

- la gestion de la mémoire, la gestion des tâches
- l'accès aux périphériques (disque dur, lecteur de CD-Rom, clavier, souris, …),
- la gestion des fichiers, …

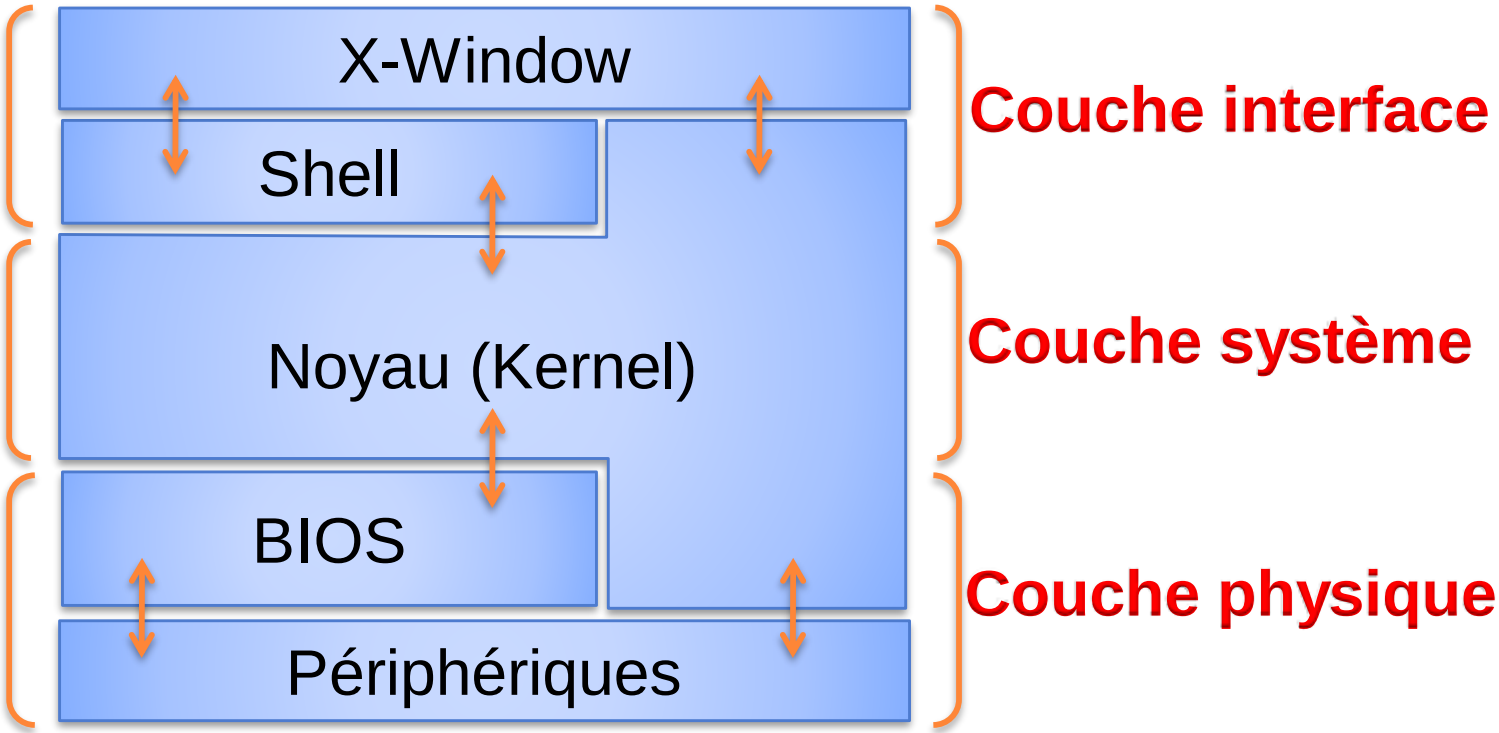
• Shell : interprète les ordres de l'utilisateur et les fait exécuter par le noyau. Les ordres peuvent être passés soit directement au clavier, soit en utilisant des outils graphiques de plus haut niveau

• Applications : interagissent avec l'utilisateur ou avec d'autres applications et communiquent avec le shell ou avec le noyau

Architecture de LINUX

Divisée en 3 couches distinctes

- ✓ La couche physique : Périphériques et BIOS
- ✓ La couche système : Gérée par le noyau
- ✓ La couche interface : le Shell et/ou le système X-Window



SE LOGGER

- Linux possède un mécanisme d'identification connu sous le nom de login
- Pour utiliser un système Linux sur une machine, il faut avoir un compte sur cette machine et rentrer au clavier :
  - son nom d'utilisateur : **login**
  - son mot de passe : **password**
- Le système vérifie la correspondance entre le login et le mot de passe
  - si échec, il refuse l'accès
  - si correct, il lance la procédure de login (analyse différents fichiers de configuration et met en place l'environnement de l'utilisateur)
- L'utilisateur est alors placé dans son répertoire d'accueil : c-à-d

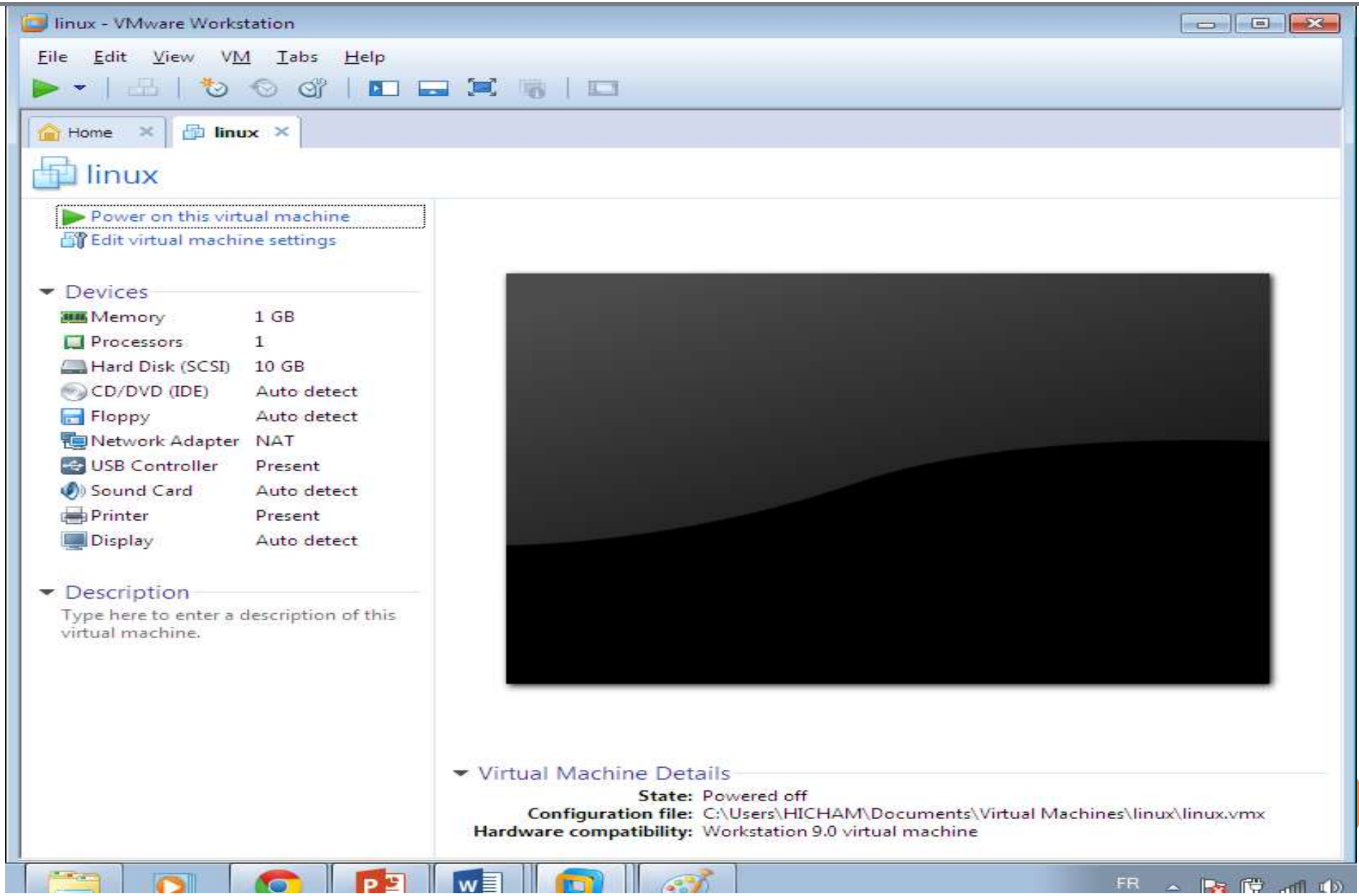
  /home/user1

CHANGER SON MOT DE PASSE

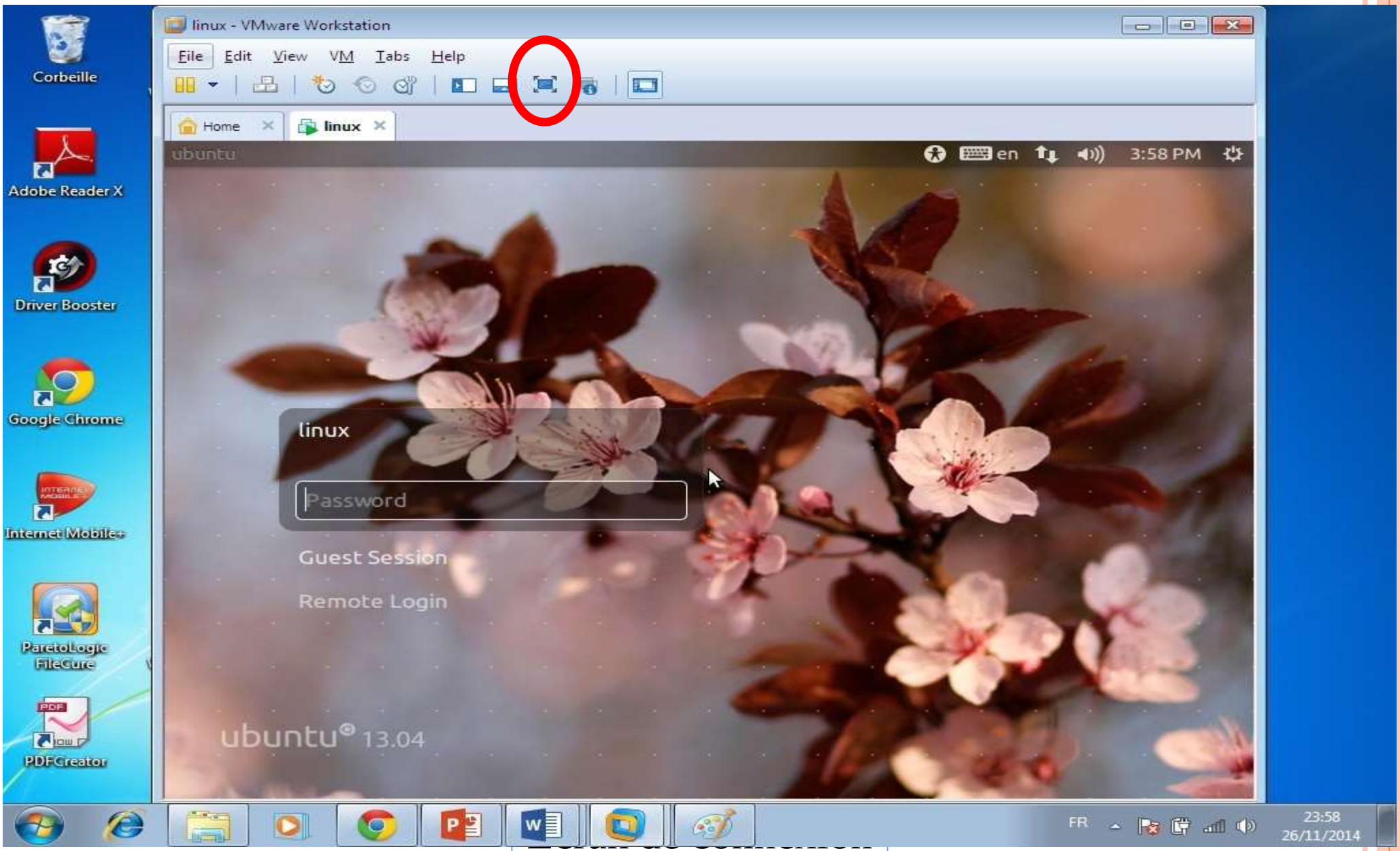
- Si vous souhaitez changer votre mot de passe, la commande pour réaliser cette opération est : **passwd** ou **yppasswd**

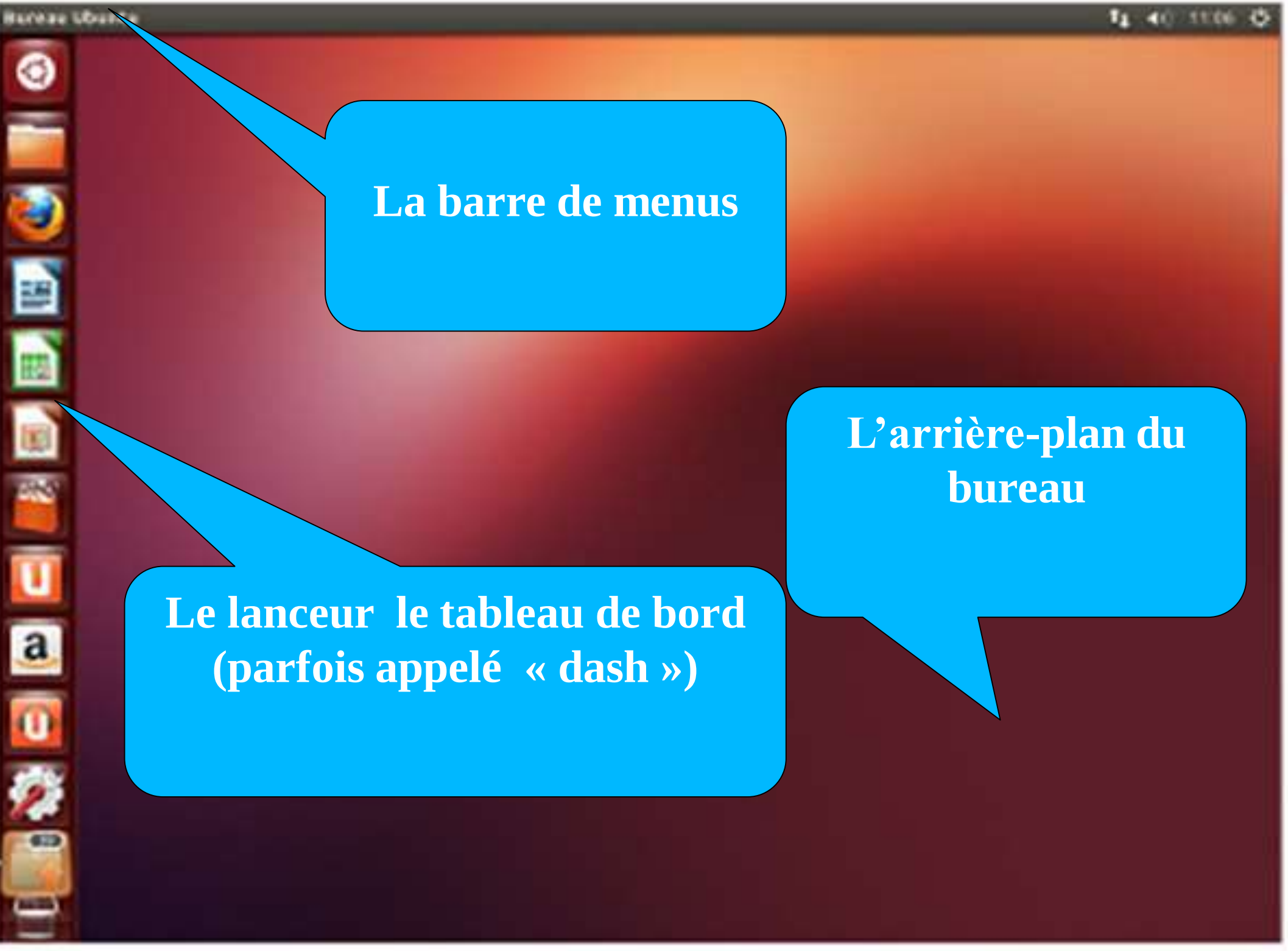
```
% passwd
Changing NIS password for USER on MACHINE
Old password:                    --entrez votre mot de passe courant
New password:                    --entrez votre nouveau mot de passe
Retype new password:             --rentrez votre mot de passe
NIS entry has changed on filemon
```

TRAVAILLER AVEC UBUNTU



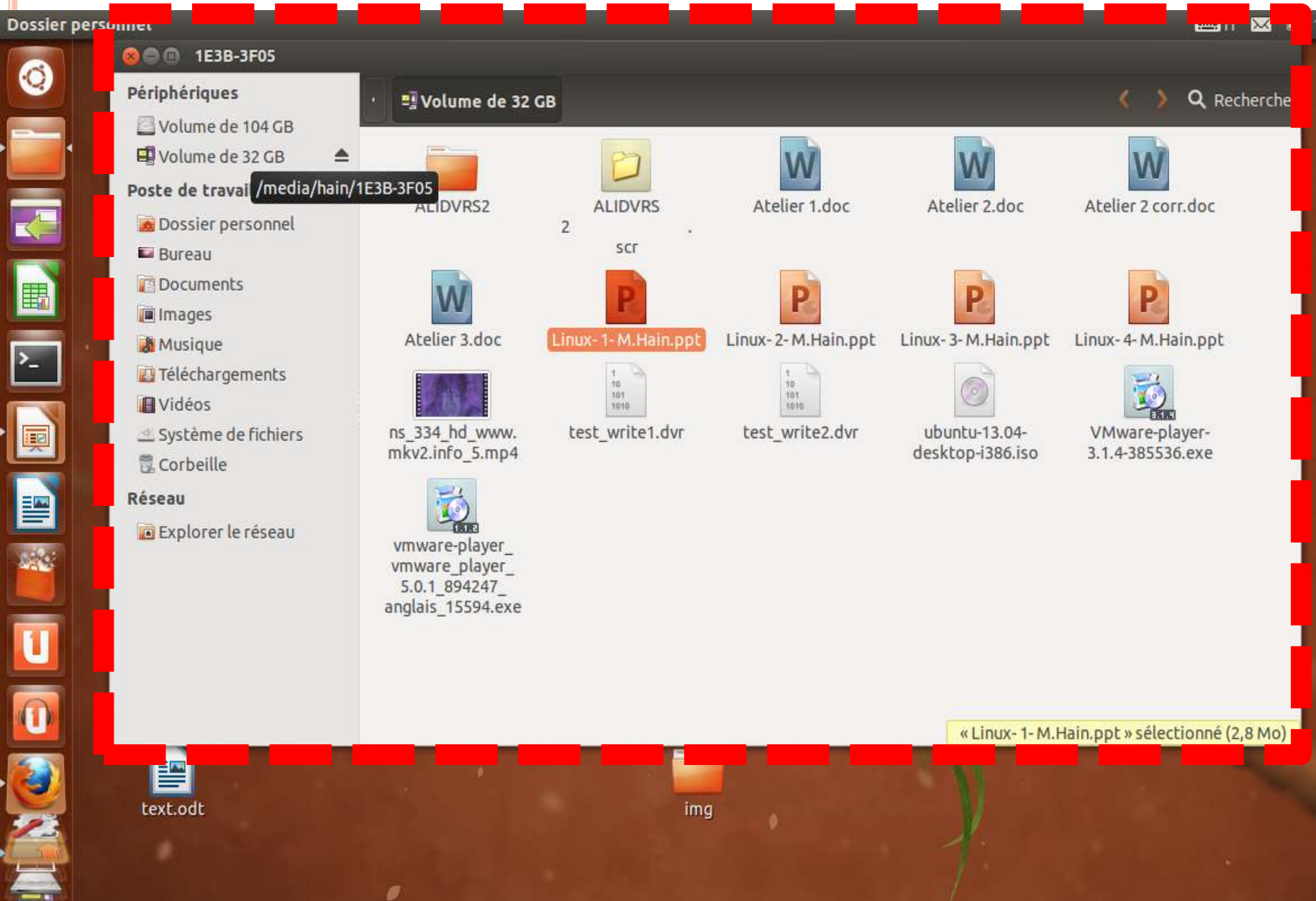
TRAVAILLER AVEC UBUNTU




La barre de menus
L’arrière-plan du bureau
Le lanceur le tableau de bord (parfois appelé « dash »)

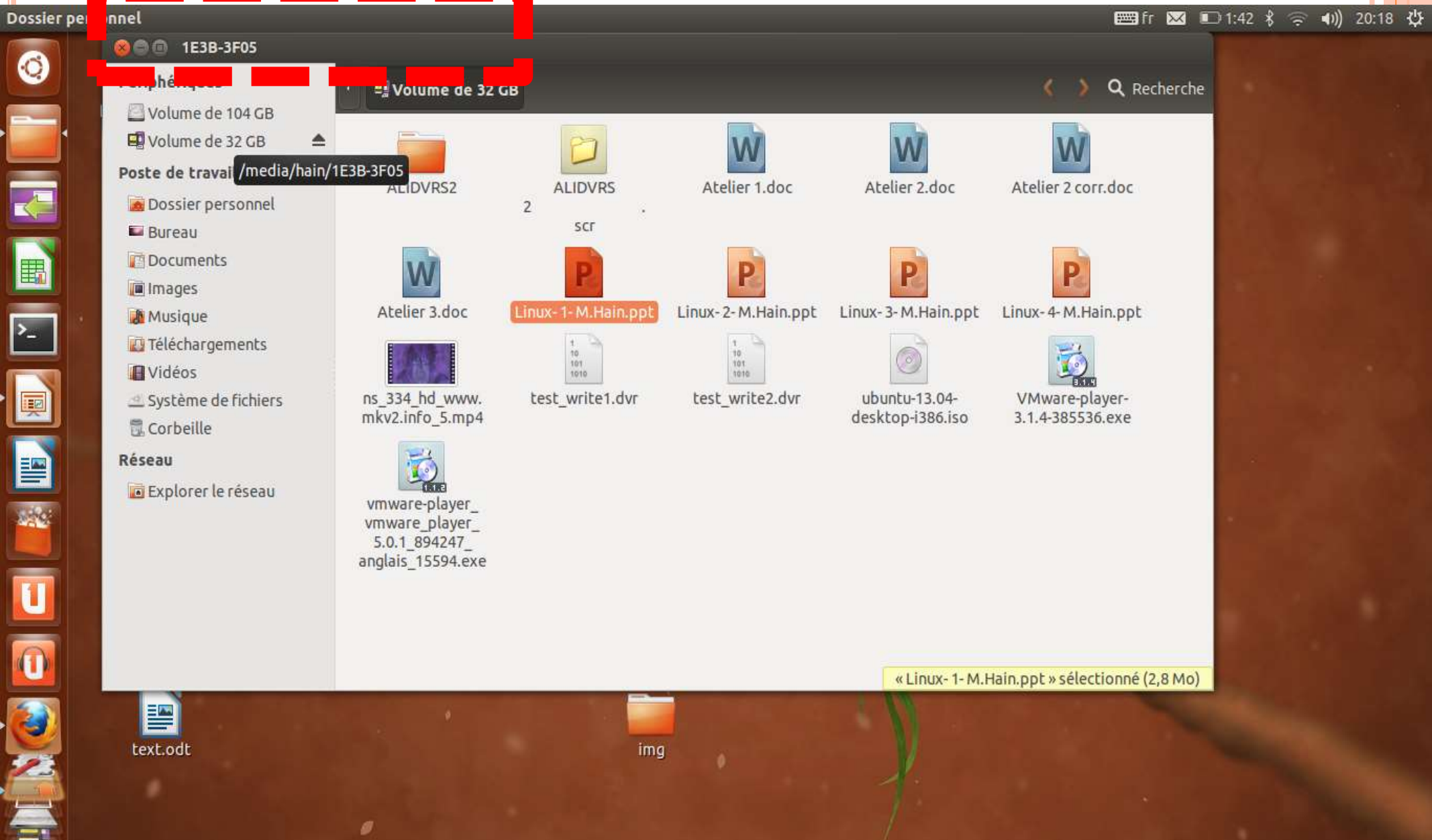
TRAVAILLER AVEC UBUNTU

Gestionnaire de fichiers Nautilus

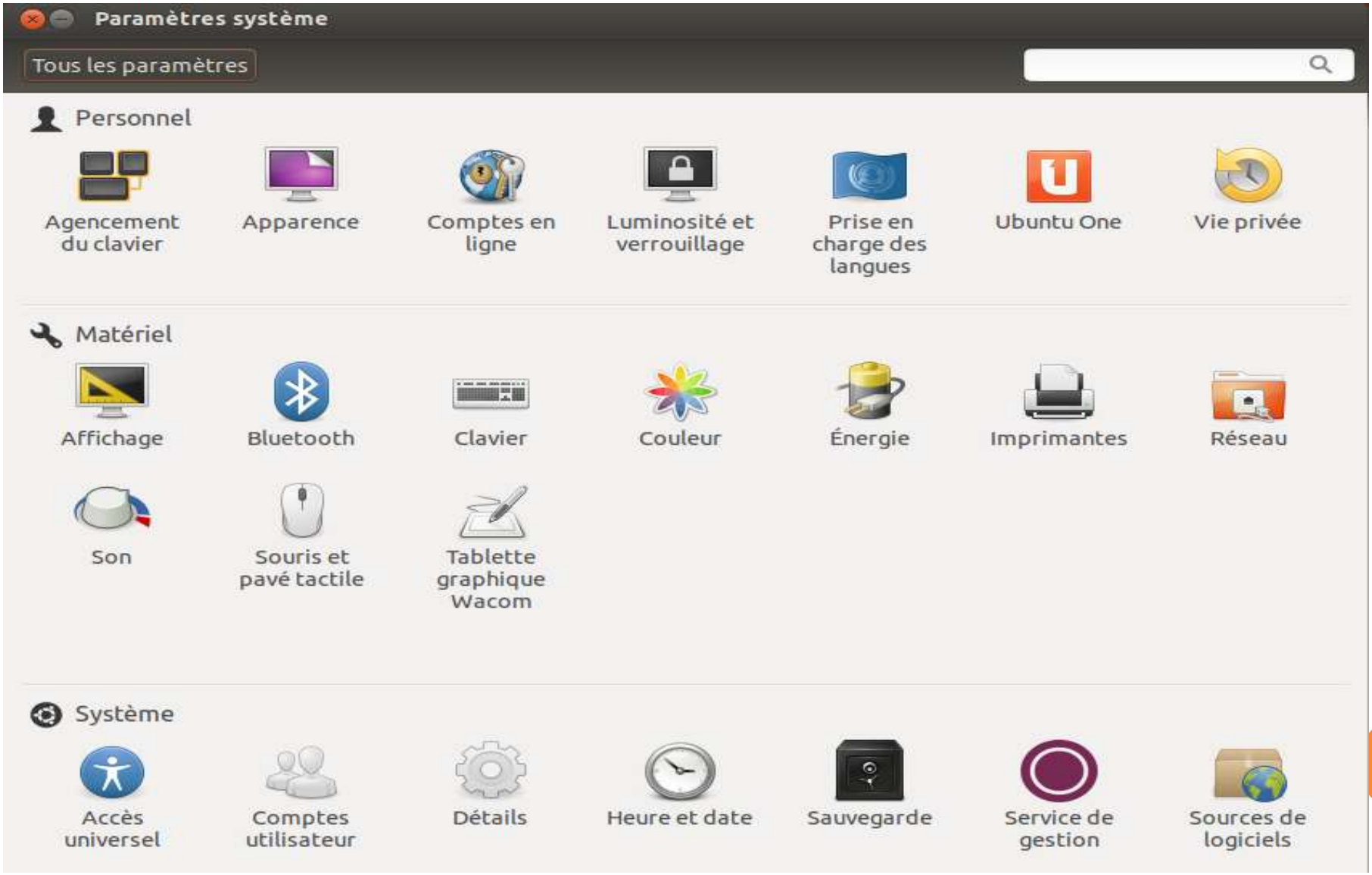


TRAVAILLER AVEC UBUNTU

Fermer, maximiser, restaurer et réduire



Personnalisation du Bureau



Thème


Apparence
Tous les paramètres Apparence
Aspect
Comportement
Arrière-plan
Papiers peints
Zoom
Thème
Taille des icônes du lanceur

Suites bureautiques

- **Windows : Microsoft Office, LibreOffice;**
- **Mac OSx : iWork, Microsoft Office, LibreOffice;**
- **Linux : LibreOffice, KOffice, GNOME Office, Kexi (app BD)**

Dans Ubuntu, vous pouvez choisir parmi de nombreuses suites bureautiques. La suite la plus populaire est l'application LibreOffice (anciennement OpenOffice). Inclus dans la suite :

- **Writer — traitement de texte**
- **Calc — tableur**
- **Impress — gestionnaire de présentations**
- **Draw — programme de dessin**
- **Base — base de données**
- **Math — éditeur d'équation**

Navigateurs web

- **Windows : Microsoft Internet Explorer, Mozilla Firefox, Opera, Chromium, Google Chrome**
- **Mac OS x: Safari, Mozilla Firefox, Opera, Chromium, Google Chrome**
- **Linux : Mozilla Firefox, Opera, Chromium, Google Chrome, Epiphany**

Lecteurs PDF

- **Windows : Adobe Reader ;**
- **Mac OS x: Adobe Reader ;**
- **Linux : Evince, Adobe Reader, Okular.**

NB: Evince est un lecteur minimaliste et convivial, c'est le lecteur de PDF par défaut. Si Evince ne couvre pas vos besoins, le lecteur Adobe Readerest aussi disponible pour Ubuntu.

Lecteurs multimédia

- **Windows : Windows Media Player, VLC;**
- **Mac OS x: Quicktime, VLC;**
- **Linux : Totem, VLC, MPlayer, Kaffeine.**

Gestion de Photos

- **Windows : Gestionnaire d'images Microsoft Office, Picasa ;**
- **Mac OS x: Aperture, Picasa ;**
- **Linux : Shotwell, gThumb, Gwenview, F-Spot.**

Édition & Retouche graphique

- **Windows : Adobe Photoshop, GIMP ;**
- **Mac OS x: Adobe Photoshop, GIMP ;**
- **Linux : GIMP, Inkscape.**

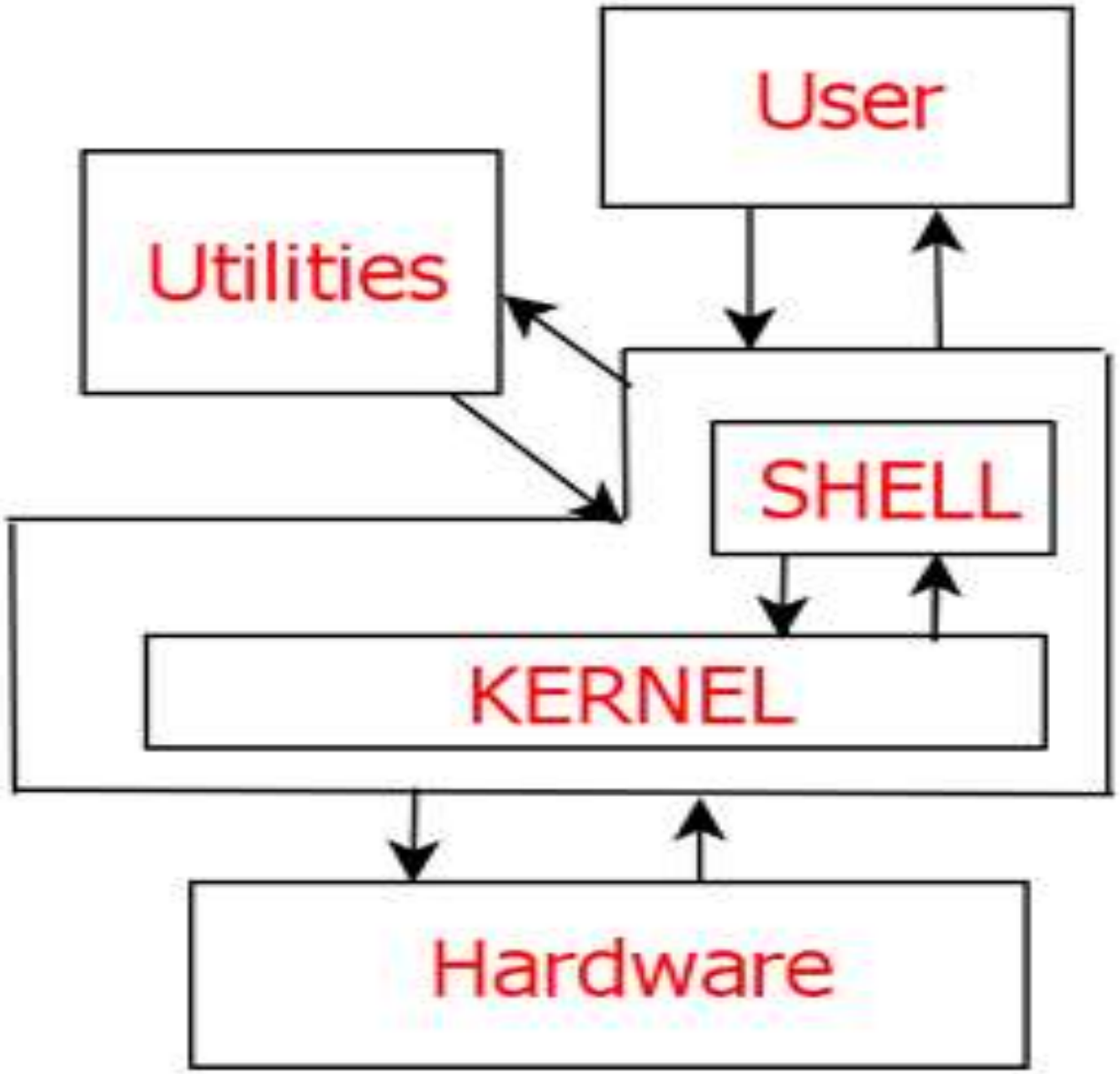
Messagerie instantanée

- **Windows : Windows Live Messenger, Yahoo ! Messenger, Google Talk ;**
- **Mac OS x: Windows Live Messenger, Yahoo ! Messenger, Adium, iChat;**
- **Linux : Empathy, Pidgin, Kopete, aMSN.**

Clients BitTorrent

- **Windows : µTorrent, Azureus ;**
- **Mac OS x: Transmission, Azureus ;**
- **Linux : Transmission, Deluge, Azureus, KTorrent, Flush, Vuze, BitStorm Lite.**

LES SHELLS



LES SHELLS

Qu'appelle t'on un shell ?

Un shell est la liaison la plus élémentaire entre l'utilisateur et le système d'exploitation, c'est à dire le programme de gestion de la ligne de commande.

Les commandes saisites sont interprétées par le shell et transmises au système d'exploitation.

QUEL SHELL

- Après le login, l'utilisateur accède à un interpréteur de commandes ou **shell**
- Le shell affiche un «prompt» et attend les commandes de l'utilisateur
- Il en existe plusieurs avec des fonctionnalités et des interfaces différentes les uns des autres
  - **sh : Bourne Shell (shell standard)**
  - **ksh : Korn Shell**
  - **csh : C Shell**
  - **bash : GNU (Bourne Again Shell)**
- Pour savoir quel shell est utilisé, tapez :

```
% echo $SHELL
/bin/bash
```

- Le liste des shells autorisés : **/etc/shells**

SHELL

- Ouverture du shell (sous X Window) :
  - cliquer sur l'icône représentant le shell, ou sélectionner «ouvrir un terminal» dans le menu droit de la souris
- A ce point le shell peut recevoir des commandes :
  - exemples :
    - date : affiche la date
    - ls : liste les fichiers du répertoire courant
- Fermeture du shell :
  - commande exit
  - commande logout
  - Ctrl-D

LE TERMINAL


en
4:06 PM
terminal
Applications
Terminal
UXTerm
XTerm
More suggestions
See 21 more results
Terminal 4
Locura Terminal
Terminal Four
The Terminator Theme (Terminator)
Legal notice
VM, move the mouse pointer inside or press Ctrl+G.

LE TERMINAL


Untitled 1.odp
osons Aimer
m
hain@hain: ~
hain@hain:~$
prompt
Clt+Alt+T

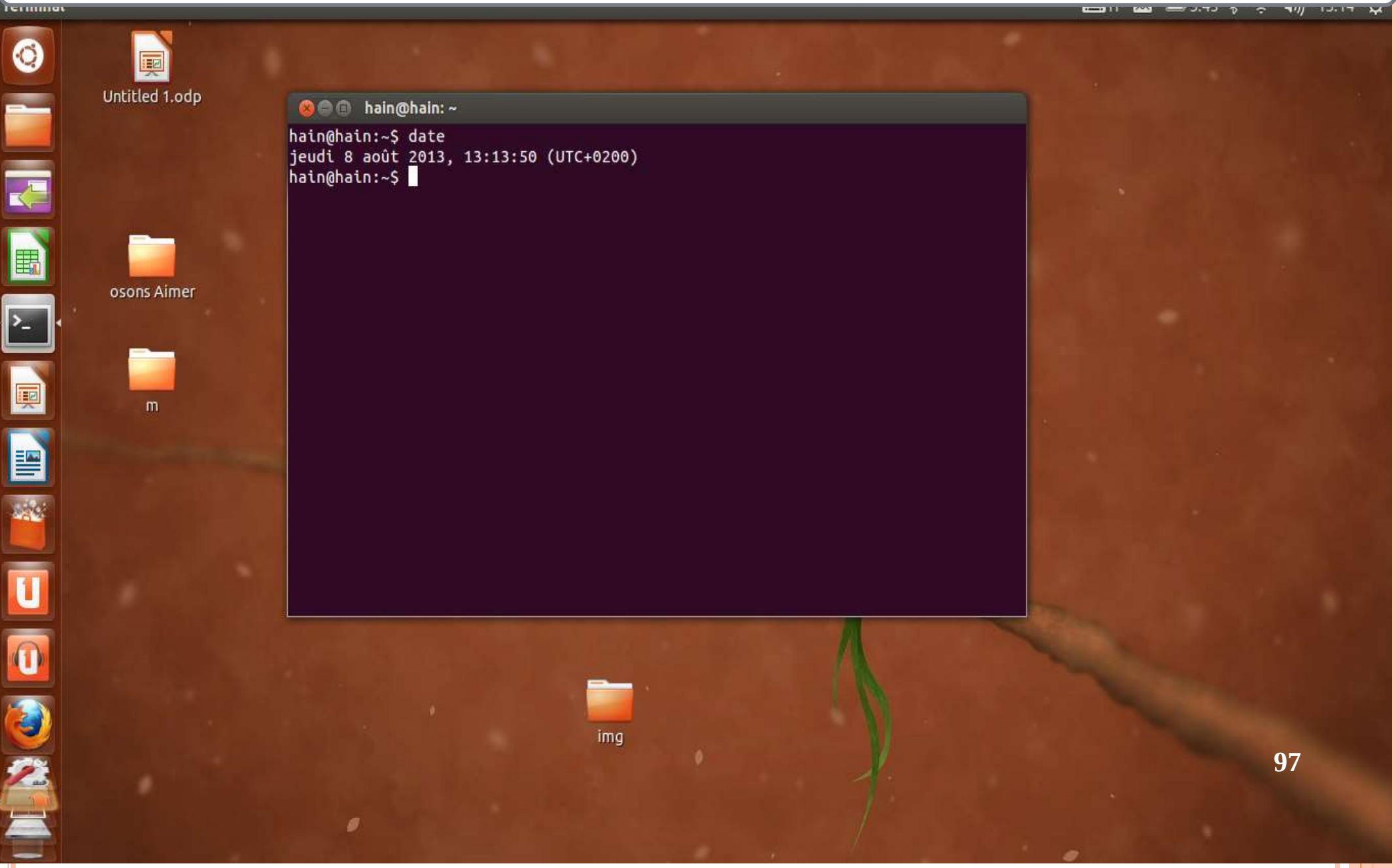
LE TERMINAL

Il est possible sous Linux d'activer simultanément plusieurs consoles de connexion.

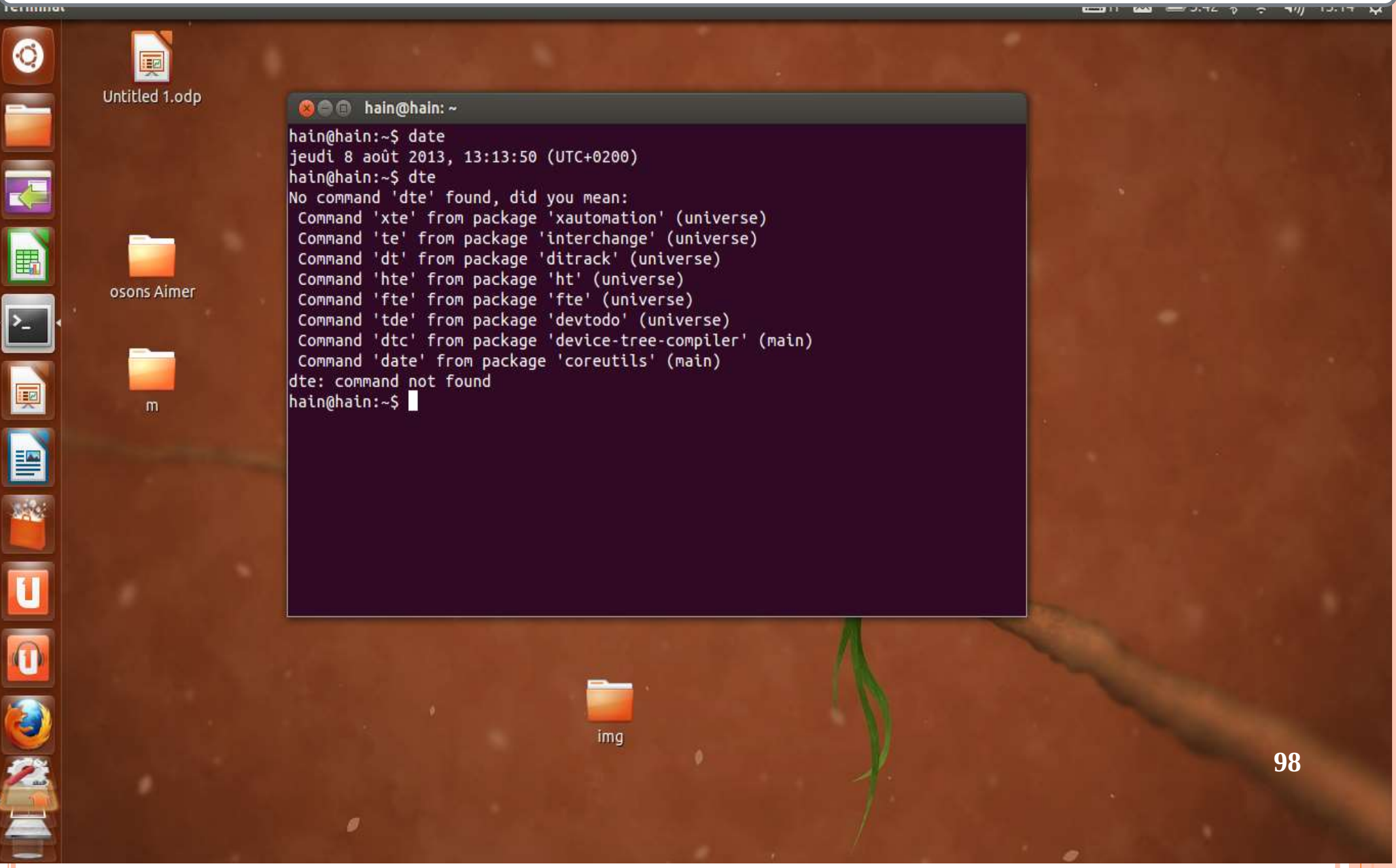
Le basculement entre les consoles est obtenu par la combinaison de touches <ctrl–alt–Fx>;

où Fx représente les touches de fonction F1 à F8 .

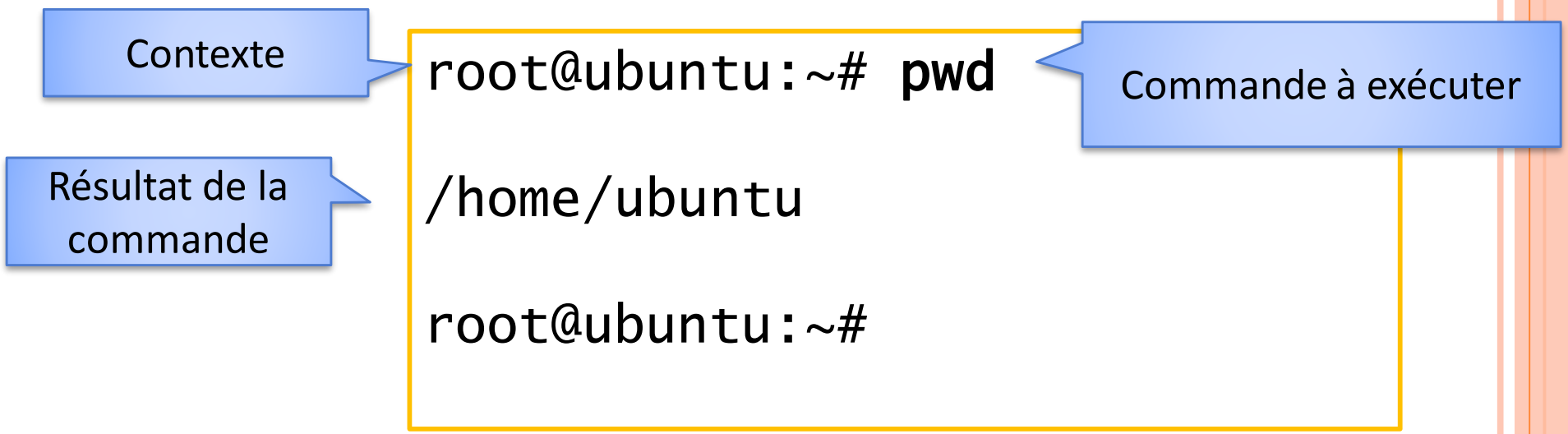
LE TERMINAL


Untitled 1.odp
osons Aimer
m
hain@hain: ~
hain@hain:~$ date
jeudi 8 août 2013, 13:13:50 (UTC+0200)
hain@hain:~$
img

LE TERMINAL


Untitled 1.odp
osons Aimer
m
img
hain@hain: ~
hain@hain:~$ date
jeudi 8 août 2013, 13:13:50 (UTC+0200)
hain@hain:~$ dte
No command 'dte' found, did you mean:
 Command 'xte' from package 'xautomation' (universe)
 Command 'te' from package 'interchange' (universe)
 Command 'dt' from package 'ditrack' (universe)
 Command 'hte' from package 'ht' (universe)
 Command 'fte' from package 'fte' (universe)
 Command 'tde' from package 'devtodo' (universe)
 Command 'dtc' from package 'device-tree-compiler' (main)
 Command 'date' from package 'coreutils' (main)
dte: command not found
hain@hain:~$

SHELL


Contexte
root@ubuntu:~# pwd
Commande à exécuter
Résultat de la commande
/home/ubuntu
root@ubuntu:~#

QUELLE EST MON IDENTITÉ ?

- Pour Linux, l'identité d'un utilisateur est celle sous laquelle il se logge
- La commande **whoami** vous donne votre identité

```
% whoami
root
```

- L'utilisateur appartient également à un ou plusieurs groupes
- La commande **id** vous donne votre identité et votre groupe

```
% id
uid=5230(ubuntu)  gid=64(profs)  groups=64(profs)
```

n° de l'utilisateur — utilisateur — n° du groupe — groupe

COMMANDES LINUX

Une commande est l'exécution d'un programme dans l'interprète (Shell). Elle prend en entrée des options et/ou des paramètres.

Elle peut renvoyer de l'information à l'écran ou dans un fichier, modifier un fichier, ou produire un message d'erreur.

Une commande est composée en premier d'un code mnémonique, suivi parfois d'options ou de paramètres.

```
$ cmd –option argument
```

Pour obtenir toutes les options d'une commande, il faut appeler l'option --help.

COMMANDES LINUX

$ commande options arguments

- les options (souvent très nombreuses) permettent de modifier le comportement de la commande
- en général elles sont précédées du signe '-' (e.g. ls -l)
- Certaines commandes utilisent des arguments (e.g. nom de fichier) Il y a un manuel en ligne: man ls

COMMANDES LINUX

IMPORTANT :

- Unix est sensible à la casse
  - **a != A**
  - **ls != LS** ou de **Ls**
- Unix utilise l'espace comme séparateur de commandes

  Exemple :

  man date et non mandate

CHAÎNER LES COMMANDES

```
$ date; pwd; cal
lun fév 25 22:29:09 CET 2013
/usr/share/man/man9
février 2013
di lu ma me je ve sa
1  2
3  4  5  6  7  8  9
10 11 12 13 14 15 16
```

QUELQUES ASTUCES

Quelques séquences de raccourcis à connaître :

- [Ctrl] C : interruption du programme : il se termine.
- [Ctrl] Z : stoppe le programme (voir les processus).
- [Ctrl] D : interrompt une saisie sur un prompt >

Chapitre 3
Gestion des fichiers et des répertoires

SYSTÈME DE FICHIERS

Fichiers de stockage

Mémoire vivre

Répertoires d'arrangement

SYSTÈME DE FICHIERS

Un système de fichier = Un système de fichiers est l'agencement logique et structuré des données sur un média.

- **un disque dur,**
- **une clé USB ,**
- **un DVD,**
- **...**

Les informations ne sont pas écrites n'importe comment sur les disques. Une organisation est nécessaire pour y placer tant les informations sur les fichiers qui y sont stockés que les données. C'est le système de fichiers (et les pilotes associés) qui définit cette organisation.

SYSTÈME DE FICHIERS

Fichier = Ensemble de données numériques réunies sous un même nom, enregistrées sur différents supports et dont la méthode de stockage implique un format.

Techniquement un fichier est une information numérique constituée d'une séquence d'octets, permettant des usages divers, L'ensemble des fichiers est architecturé autour d'une unique arborescence dont la base, appelée racine, est notée «/».

SYSTÈME DE FICHIERS

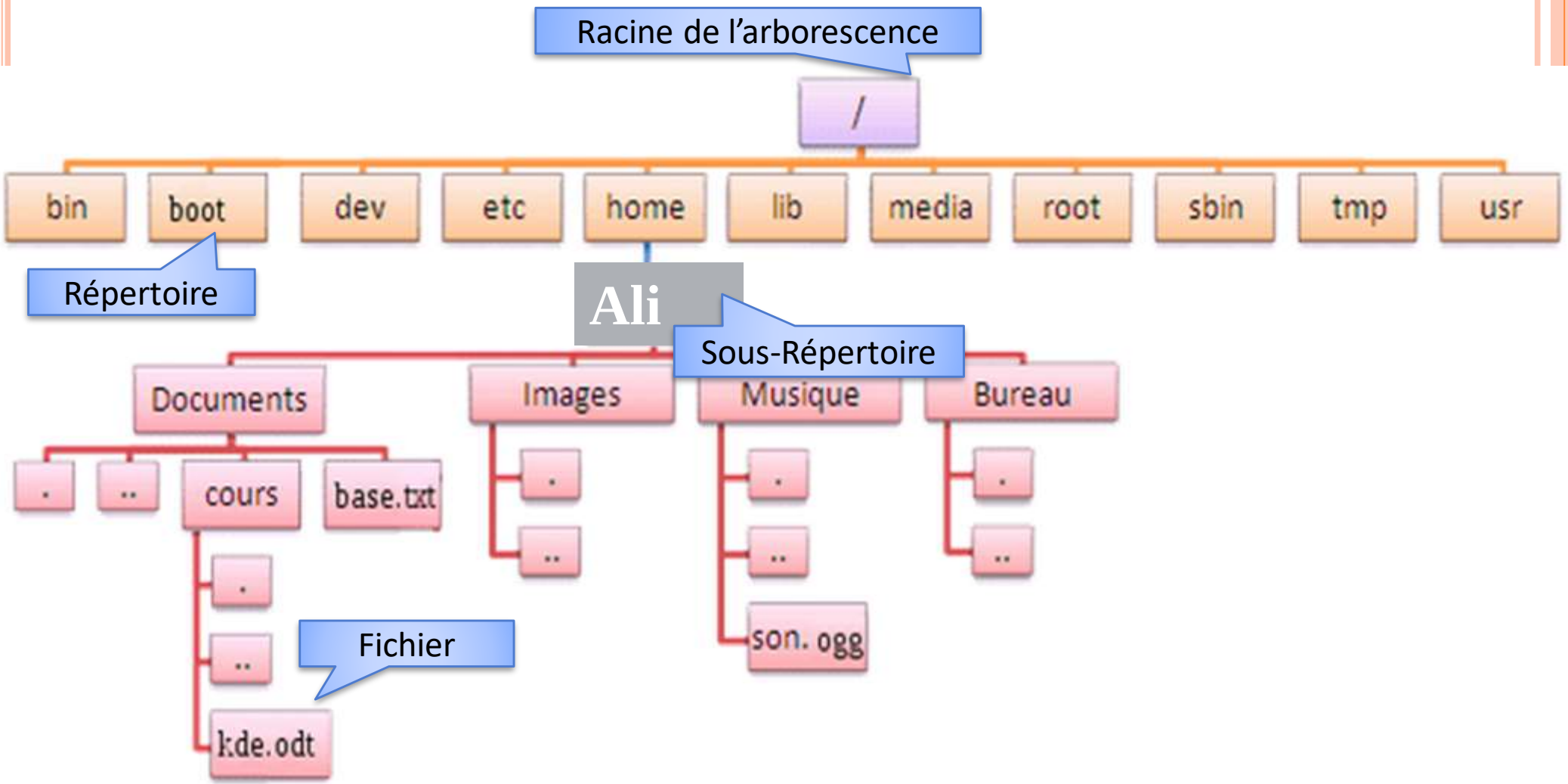
- Qu'est-ce qu'une arborescence ?
  - Organisation logique des fichiers sur un ou plusieurs systèmes de fichiers
  - Il s'agit d'une structure de données hiérarchique de type arbre

SYSTÈME DE FICHIERS

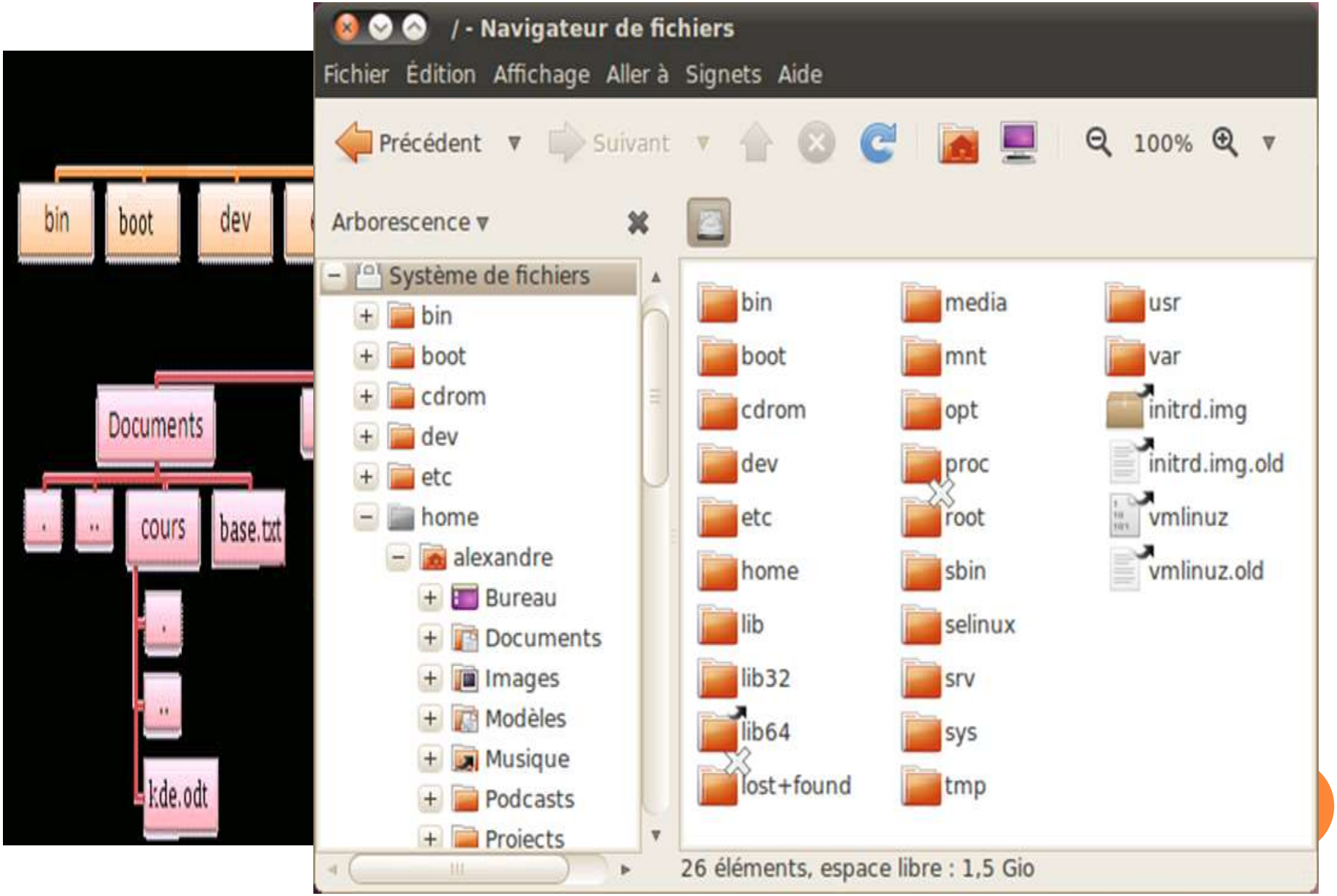
RÉPERTOIRE : permet de ranger les fichiers. Il est possible d'en créer plusieurs. Il est l'équivalent du classeur (Folder, Collection)

SYSTÈME DE FICHIERS

le système de fichiers est organisé en une structure arborescente dont les nœuds sont des répertoires et les feuilles des fichiers ordinaires.

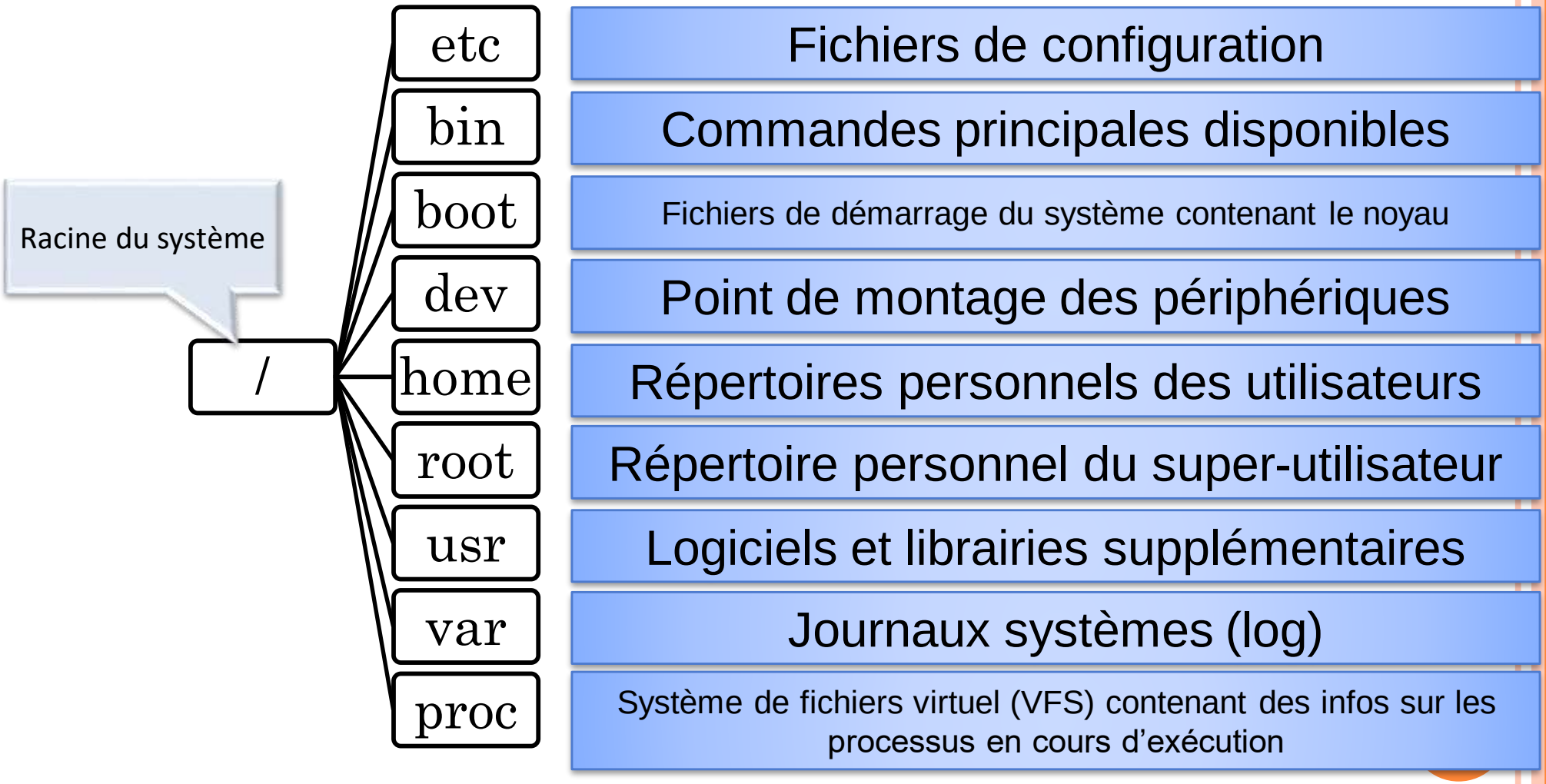


ARBORESCENCE LINUX



ARBORESCENCE LINUX

l'arborescence typique d'un système Linux



ARBORESCENCE LINUX

SYMBOLES ASSOCIÉS À L'ARBORESCENCE:

Différents symboles sont utilisés pour désigner des répertoires :

- **Le « . » → Répertoire courant**
- **Le « .. » → Répertoire parent**
- **Le « ~ » → Répertoire personnel de l'utilisateur courant**
- **Le « / » → Répertoire racine de l'arbre**

LES TYPES DE FICHIERS LINUX

Le système Linux offre trois types de fichiers principaux :

a. **Les fichiers ordinaires (regular files).**

b. **Les fichiers répertoires ou répertoires (directories).**

c. **Les fichiers spéciaux. Ils désignent les périphériques, les tubes ou autres supports de communication .**

Les fichiers spéciaux sont de plusieurs types, à savoir :

- l : un lien symbolique
- b : un fichier spécial de type bloc (périphériques …)
- c : un fichier spécial de type caractère (périphériques …)
- s : socket
- …

LE RÉPERTOIRE PERSONNEL- LE TILDE

Le terminal interprète le caractère tilde ~ comme un alias du répertoire personnel, mais le tilde ne doit être précédé d'aucun caractère.

Exemple: Pour vous déplacer dans le répertoire tmp de votre dossier personnel d'où que vous soyez :

```
$ cd  ~/tmp
```

Mais la commande ci- dessous, génère une erreur :

```
$ cd  /~
```

Commande de manipulation des répertoires

La commande permettant d'afficher le chemin d'accès du répertoire courant (**C'est-à-dire votre position dans l'arborescence)** est :

pwd **Print Working Directory**

Commande de manipulation des répertoires

- La commande permettant de **se déplacer dans une arborescence** est :

  cd répertoire (change directory)

```
% pwd
/home/profs/yassine
% cd Enseignement
% pwd
/home/profs/yassine/Enseignement
```

- Chaque répertoire contient 2 entrées supplémentaires :
  - « . » : désigne le répertoire courant
  - « .. » : désigne le répertoire parent

- On peut se déplacer en utilisant un chemin : **cd chemin**

- Deux types de chemins : **absolu ou relatif**

Commande de manipulation des répertoires

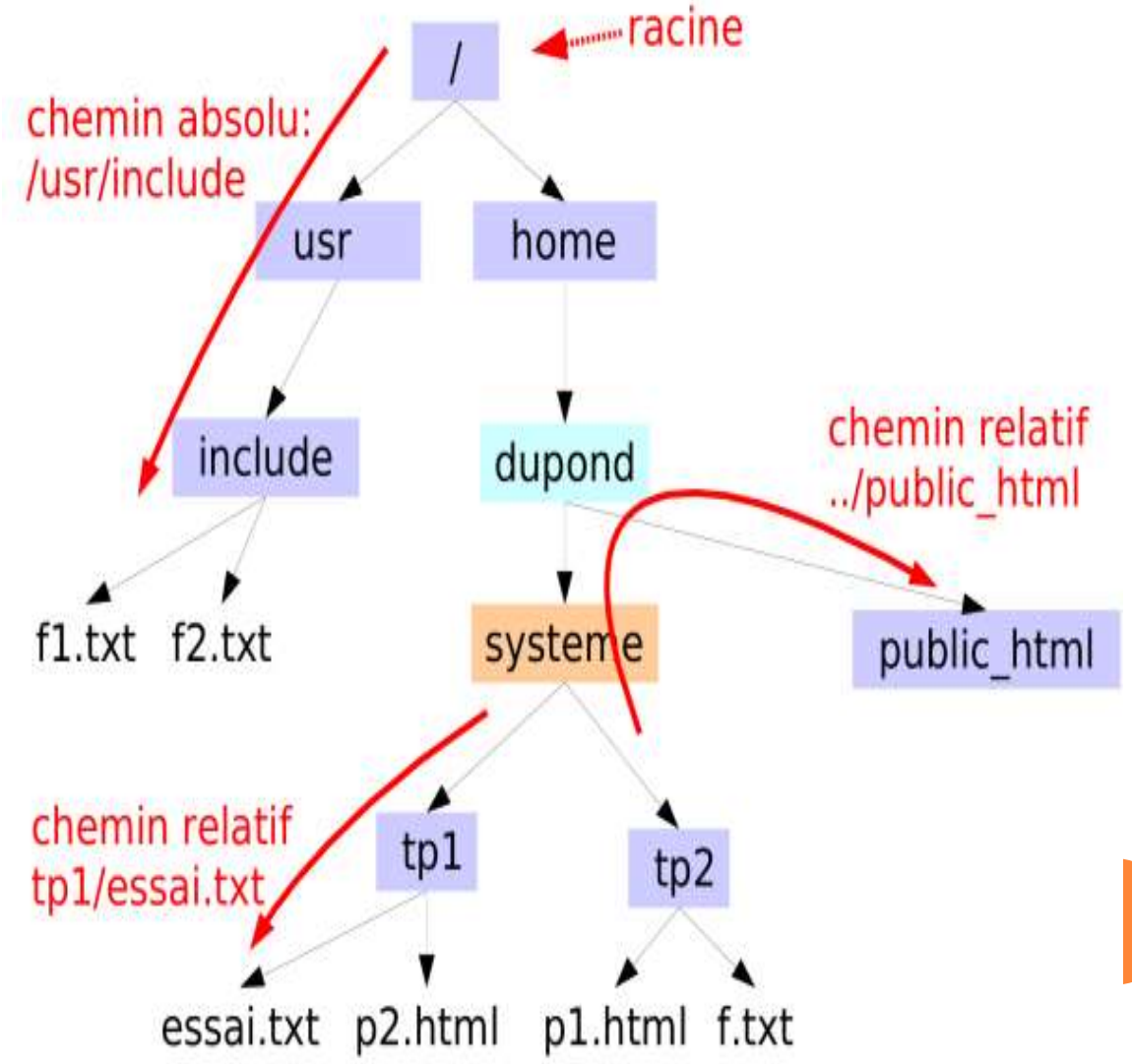
LES CHEMINS D'ACCÈS

- chemin relatif :

part de l'endroit ou vous êtes (répertoire courant)

- chemin absolu :

commence de la racine "/" ou bien par un raccourci "~"



Commande de manipulation des répertoires

LES CHEMINS RELATIFS ET ABSOLUS ?

Voici par exemple l'utilisation de cd pour aller dans le bureau à l'aide d'un chemin absolu:

$ cd /home/ali/Bureau

Voici la même commande mais en utilisant un chemin relatif. Le dossier courant est /home/ali.

$ pwd
/home/ali
$ cd Bureau

Commande de manipulation des répertoires

Le répertoire de travail est :/home/ali.

Chemin relatif	Chemin absolu
	/home/ali/liste.txt
.	/home/ali/exo
	/home/ali/Rep/trace
	/home/ali
..	/home
	/home/amal/prog.c
	/temp

Commande de manipulation des répertoires

- La commande servant à créer des répertoires est :

mkdir [options] répertoires… (make directory)

- Il suffit d'avoir le droit d'écrire (w) dans le répertoire père
- Pour créer une arborescence entière, on utilise l'option –p

Exemple : créer l'arborescence ~/TP_Linux/TP_Groupe1

mkdir –p TP_Linux/TP_Groupe1

- La commande servant à supprimer un répertoire est :

rmdir répertoire **(remove directory)**

Le répertoire doit être vide

- **rmdir –p** repertoire pour supprimer les sous répertoires qui peuvent figurer dans le chemin d'accès à un répertoire s'ils sont vides
- Si le répertoire est non vide : **rmdir –r** repertoire

Commande de manipulation des répertoires

La commande du

permet d'afficher la taille d'une arborescence.

- Syntaxe : du [options] [répertoire]

```
karima@karima-HP-EliteBook-8440p:~$ du
4       ./Musique
4       ./.cache/gnome-screenshot
768     ./.cache/wallpaper
196     ./.cache/upstart
4       ./.cache/unity
20      ./.cache/ibus/bus
24      ./.cache/ibus
20      ./.cache/software-center/software-center-agent.db.tmp
4       ./.cache/software-center/piston-helper
4       ./.cache/software-center/rnrclient
108     ./.cache/software-center
116     ./.cache/compizconfig-1
8       ./.cache/sso
12      ./.cache/thumbnails/normal
84      ./.cache/thumbnails/fail/gnome-thumbnail-factory
```

Commande de manipulation des répertoires

OPTIONS DE LA COMMANDE DU

Option	description
-a	**Affiche les statistiques pour tous les fichiers, pas seulement les répertoires.**
-b :	**Affiche les tailles en octets.**
-c :	**Affiche la totalité de l'espace utilisé par tous les arguments.**
-k :	**Affiche la taille en kilo-octets.**
-S :	**Affiche séparément la taille de chaque répertoire, sans inclure la taille des sous-répertoires.**

Commande de manipulation des Fichiers

ls LiSt files

Permet d'obtenir la liste et les caractéristiques des fichiers contenus dans un répertoire.

Exemples

```
ubuntu@pc :~$ pwd
/home/ubuntu
ubuntu@pc :~$ ls
Bureau    examples.desktop  Modèles  Public          Ubuntu One
Documents  Images           Musique  Téléchargements  Vidéos
ubuntu@pc :~$ cd Bureau
ubuntu@pc :~/Bureau$ ls
img  m  osons Aimer  presentation.odp
```

Commande de manipulation des répertoires

OPTIONS DE LA COMMANDE LS

Option	description
-l	**liste les attributs de fichiers**
-lu	**affiche les fichiers par date de dernier accès et indique cette date**
-a	**liste tous les fichiers du répertoire, y compris les fichiers cachés.**
-m	**affiche les fichiers en les séparant par une virgule au lieu de les présenter en colonnes.**
-d	**affiche les répertoires et non leur contenu,**
-b	**affiche les caractères non imprimables.**
-R	**affiche le contenu d'une arborescence.**
-t	**trie par date, c'est-à-dire en les classant du récent au plus ancien.**
-F	**trie par type.**
-S	**trie par ordre de taille décroissante.**
-X	**trie par extension.**
-r	**trie par ordre alphabétique inverse.**
-tr	**affiche les fichiers par date en commençant par les plus anciens.**

Commande de manipulation des répertoires

LES CARACTÈRES SPÉCIAUX

Les caractères spéciaux et leur signification

- *** désigne toute chaîne de 0 à n caractères;**
- **? désigne un caractère quelconque;**
- **[...] désigne un caractère quelconque appartenant à l'ensemble des caractères entre crochets.**

Commande de manipulation des Fichiers

Exemples

```
ubuntu@pc :~$  ls   cou*
coursUnix              cours_1

ubuntu@pc :~$  ls   *cou*
coursUnix              cours_1    lecours        _cours

ubuntu@pc :~$ ls   exo_?
exo_2                  exo_7                  exo_9

ubuntu@pc :~$ ls   exo_??
exo_12
```

Commande de manipulation des Fichiers

Exemples

ubuntu@pc :~$ ls exo_ [1-4]

exo_2 exo_2 exo_4

ubuntu@pc :~$ ls exo_ [!1-4]

exo_5 exo_8

Commande de manipulation des Fichiers

CoPy

Cette commande permet la copie de fichiers.
Elle s'utilise sous quatre formes :

1) La copie d'un fichier source dans un fichier destination.
Exemple

Dans le répertoire p2, copie du fichier text1 dans text2.

```
ubuntu@pc :~/Bureau/p2$ cp text1 text2
```

Commande de manipulation des Fichiers

2) La copie d'un fichier dans un répertoire.

Exemple

```
ubuntu@pc :~/Bureau/p2$ Pwd
 /home/yasine/Bureau/p2
ubuntu@pc :~/Bureau/p2 $ cp text1 home/ubuntu/Bureau/p1
cp: cannot create regular file `home/ubuntu/Bureau/p1': No such file or
directory
ubuntu@pc :~/Bureau/p2 $ cp text1 /home/ubuntu/Bureau/p1
```

Commande de manipulation des Fichiers

3) La copie d'un répertoire dans un autre (seuls les fichiers sont copiés : on obtient un message d'erreur pour la copie des répertoires).

Exemple

Copie du contenu du répertoire xstra dans /home/xstra/projet2.

```
ubuntu@pc :~$  cd /home/xstra
ubuntu@pc :~$  mkdir projet2
ubuntu@pc :~$  cp *  /home/xstra/projet2
```

Commande de manipulation des Fichiers

4) La copie récursive permet de copier une arborescence.

Exemple

Copie de l'arborescence de xstra/projet1 sous xstra/projet2.

```
ubuntu@pc :~$ cd /home/xstra/projet1
ubuntu@pc :~$ cp –r * /home/xstra/projet2
```

Commande de manipulation des Fichiers

MoVe

En première analyse, cette commande est équivalente à une copie, suivie d'une suppression. Elle s'utilise sous deux formes :

1) Transfert de text1 dans text2 et suppression de

text1. Si text2 existe, il est effacé :

Exemple

```
ubuntu@pc :~$  ls
text1  text1~  text2
ubuntu@pc :~$  mv text1 text2
```

Commande de manipulation des Fichiers

2) Transfert de(s) fichier(s) cité(s) dans le répertoire avec le(s) même(s) nom(s) : mv fichier(s) répertoire

Exemple

```
ubuntu@pc :~$  ls
text1  text1~  text2
ubuntu@pc :~$  mv text2  /Bureau/p2
mv: cannot move `text2' to `/Bureau/p2': No such file or directory
ubuntu@pc :~$  pwd
/home/hain/Bureau/p1
ubuntu@pc :~$ mv text2 / home/ubuntuBureau/p2
```

Commande de manipulation des Fichiers

ReMove
Supprime un (ou plusieurs) fichier(s) d'un répertoire.

Exemple

Suppression du fichier toto du répertoire projet1.

```
ubuntu@pc :~$ cd /home/xstra/projet1
ubuntu@pc :~$ rm toto
```

Commande de manipulation des Fichiers

touch TOUCH

Cette commande permet (entre autres) de créer un fichier vide.

Exemple

```
ubuntu@pc :~$  touch demo.txt
```

Commande de recherche des Fichiers

FIND

Recherche un fichier à partir du répertoire donné.

Options les plus fréquentes :

- **name : Recherche d'un fichier par son nom**
- **iname : Même chose que -name mais insensible à la casse**
- **type : Recherche de fichier d'un certain type**
- **atime : Recherche par date de dernier accès**
- **mtime : Recherche par date de dernière modification**
- **link : Recherche du nombre de liens au fichier**
- **user : Recherche de fichiers appartenant à l'utilisateur donné**
- **group : Recherche de fichiers appartenant au groupe donné**

Commande de recherche des Fichiers

Exemples

$ find /home/ -name monfichier

Recherche le fichier *monfichier* dans toute la descendance de /home/

$ find . -name "*.c"

Recherche tous les fichiers ayant une extension .c

$ find . -mtime -5

Recherche les fichiers du répertoire courant qui ont été modifiés entre maintenant et il y a 5 jours

$ find . ! -user root

Affiche tous les fichiers n'appartenant pas à l'utilisateur root

Commande de recherche des Fichiers

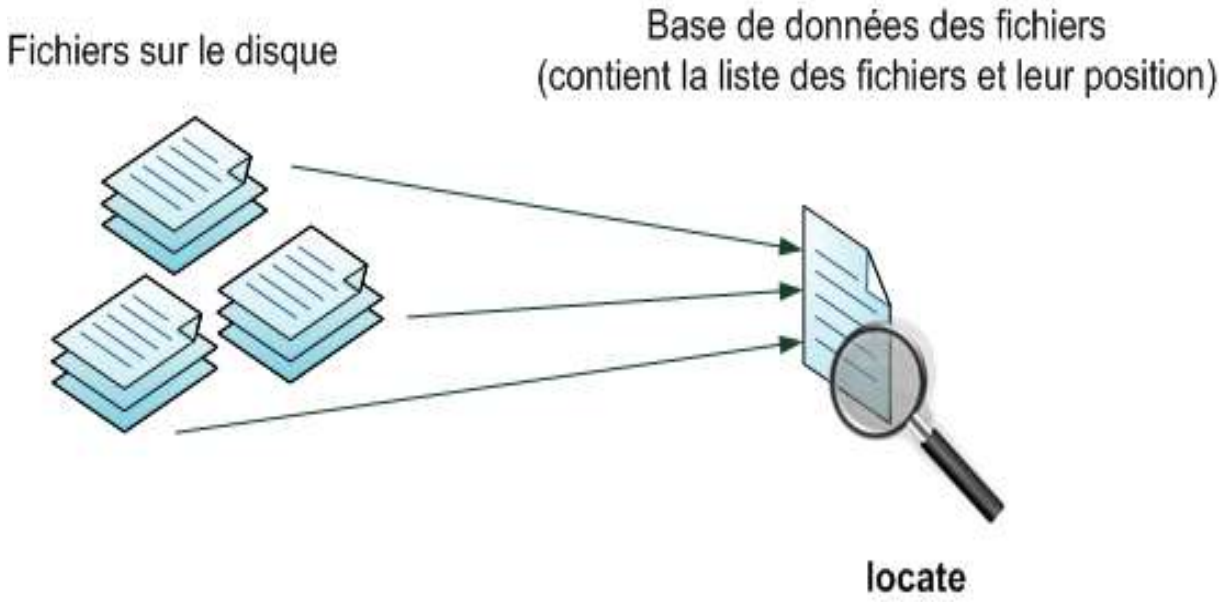
LA RECHERCHE DE FICHIER : COMMANDE LOCATE

- Permet de chercher un fichier donnée à travers son nom des fichiers
  - Syntaxe : locate « nom du fichier »

```
manchot@ubuntu:~$ locate projet
/host/photos/mini projet salma.docx
/host/photos/mini projet.docx
/host/photos/recherches/projet biodiversité.docx
/host/photos/salmaa valide/mini projet salma.docx
/host/photos/salmaa valide/mini projet.docx
manchot@ubuntu:~$
```

Commande de recherche des Fichiers

LA RECHERCHE DE FICHIER : COMMANDE LOCATE



LOCATE RECHERCHE DANS TOUS LES RÉPERTOIRES DU DISQUE DUR

Commande de recherche des Fichiers

LA RECHERCHE DE FICHIER : COMMANDE LOCATE

LOCATE → Affiche le résultat même si l'utilisateur n'a pas le droit de le lire.

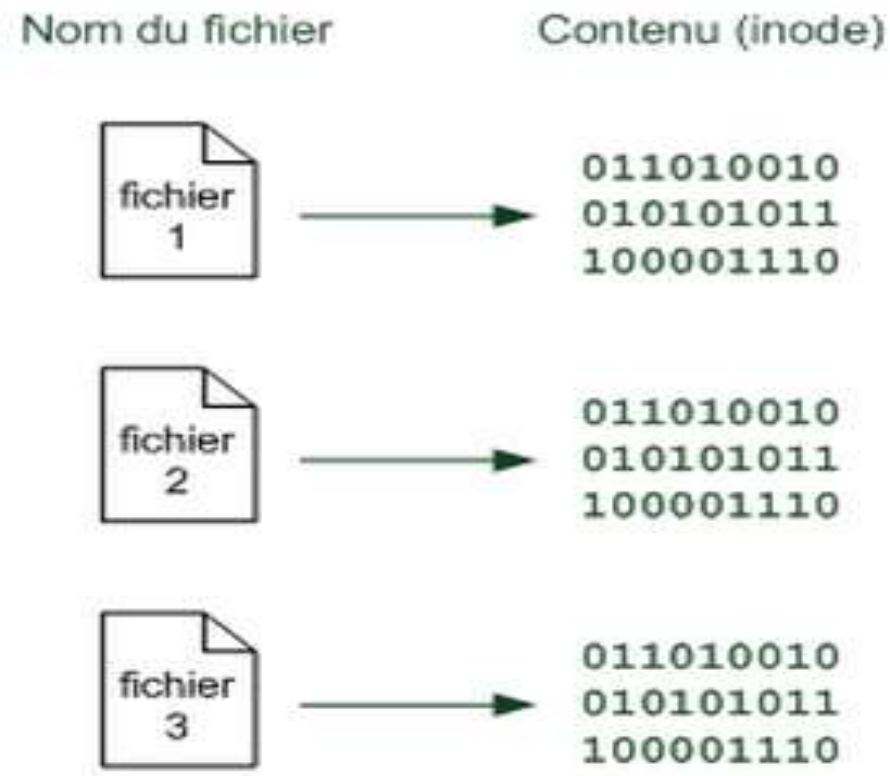
slocate → Vérifie les droits d'accès des fichiers avant l'affichage du résultat.

Commande de manipulation des Fichiers

LE STOCKAGE DES FICHIERS

sur le disque dur, chaque fichier est séparé en deux parties :

- **Son nom**
- **Son contenu**



Commande de manipulation des Fichiers

LES LIENS ENTRE DES FICHIERS

LA COMMANDE LN

Permet de créer des liens entre les fichiers

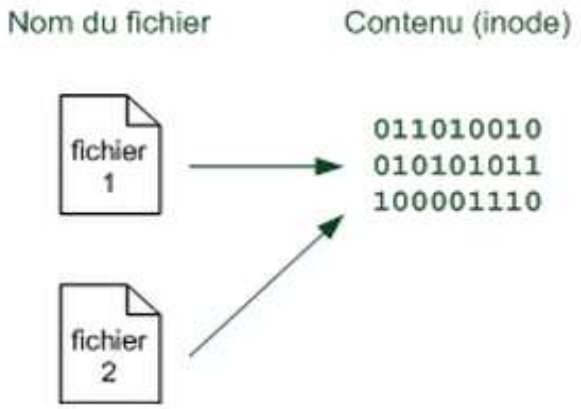
- **Liens physiques**
- **Liens symboliques**

- **Syntaxe :** ln « nom_fichier1 » « nom_fichier2 »

Commande de manipulation des Fichiers

LIEN PHYSIQUE

permet d'avoir deux noms de fichiers qui partagent exactement le même contenu (inode)



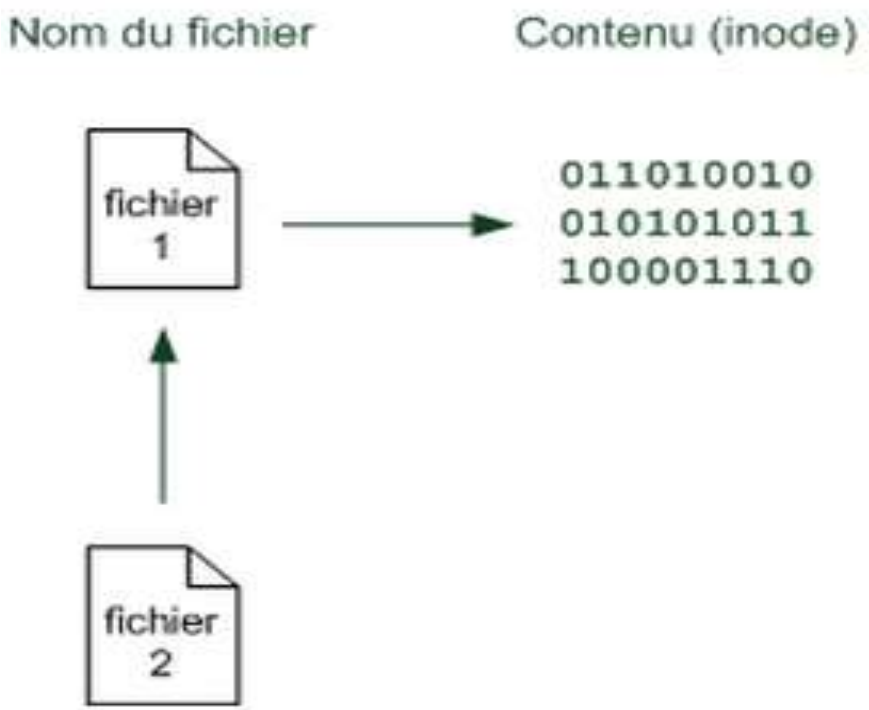
LA COMMANDE LS -LINKS

Permet d'afficher les numéros d'inode correspondants
et de vérifier que ces deux fichiers sont associés au même inode

Commande de manipulation des Fichiers

LIEN SYMBOLIQUE

Permet de créer un lien vers un autre nom de fichier



ln –s créer un lien symbolique au lieu d'un lien physique

Commande de manipulation de contenu des Fichiers

LA COMMANDE DIFF

- Permet de comparer deux fichiers textes
  - Syntaxe: diff [options] fichier1 fichier2

```
karima@karima-HP-EliteBook-8440p:~$ cat > fich1
Linux est un système d'exploitation
multitaches
multiutilisateurs
un système très structuré
^Z
[9]+  Stopped                 cat > fich1
karima@karima-HP-EliteBook-8440p:~$ cat > fich2
Linux est un système d'exploitation
multitaches
multiutilisateurs
^Z
[10]+  Stopped                 cat > fich2
karima@karima-HP-EliteBook-8440p:~$ diff fich1 fich2
4d3
< un système très structuré
```

Commande de manipulation du contenu des Fichiers

Afficher le contenu d'un fichier

```
ubuntu@pc :~$ cat  liste.txt
Bonjour tout le monde, je suis très …;

ubuntu@pc :~$ cat  liste.txt   liste2.txt
Bonjour tout le monde, je suis très …;
Voila le contenu de dexime fichier

ubuntu@pc :~$ more   liste.txt
```

Affichage page par page

Commande de manipulation de contenu des Fichiers

CONVERSION DE CHAÎNE DE CARACTÈRE :TR

- Permet de transposer ou éliminer des caractères dans un fichier ou un flux de données
  Autrement dit, un caractère appartenant à chaine1 est remplacé par le caractère de même position dans chaine2
- Syntaxe : **tr [options] chaine1 chaine2**
- Options:

Option	Signification
-d	Suppression des caractères sélectionnés
-s	Suppression des caractères dupliqués
-c	Remplacer une chaîne par son complément Tous les caractères qui ne sont pas spécifiés dans la première chaîne sont convertis selon les caractères de la seconde.

Commande de manipulation de contenu des Fichiers

CONVERSION DE CHAÎNE DE CARACTÈRE :TR

Pour faire la même chose on peut aussi bien éditer le fichier avec cat et rediriger par pipe vers tr, en tapant :

```
cat carnet-adresse | tr " : " " # "
```

On peut utiliser des métacaractères :

```
cat carnet-adresse | tr " [a-f] " " [A-F] "
```

Nous allons remplacer les caractères de a à f de minuscule en majuscule:

Commande de manipulation de contenu des Fichiers

EDITER UN FICHIER PAR LA FIN : TAIL

Soit un fichier très long, et on ne veut visualiser que la fin, on dispose de la commande tail.

La syntaxe est la suivante :

```
$tail  +10 mon-fichier
```

On obtient toutes les lignes du fichier de la 10eme jusqu'à la fin.

```
$tail  -10 mon-fichier
```

On obtient les 10 dernières lignes à partir de la fin. On peut indiquer si notre unité est la ligne (par défaut), le bloc avec l'option –t ou le caractère :

```
$tail  -c -10  mon-fichier
```

On obtient les 10 derniers caractères du fichier.

Commande de manipulation de contenu des Fichiers

Editer un fichier par le début : head

Si on a un fichier très long, et qu'on ne ve

uille visualiser que le début, on dispose de la commande head.
La syntaxe est la suivante, si on tape :

```
$head  -n +10 mon-fichier
```

On obtient toutes les lignes du fichier de la 10eme jusqu'au début.

```
$head  -n -10   mon-fichier
```

On obtient tous le fichier sauf les 10 dernier

si l'unité est la ligne (par défaut), le bloc ou le caractère avec l'option -t

```
$head  -n 10 -c   mon-fichier
```

On obtient les 10 premiers caractères du fichier.

Commande de manipulation de contenu des Fichiers

LA COMMANDE CUT

La commande cut permet d'afficher des zones spécifiques d'un fichier.

OPTIONS DE LA COMMANDE CUT

Option	Description
-c	**Permet de sélectionner les colonnes.**
-d	**Permet de spécifier un séparateur de champs**
-f	**Indique le numéro du ou des champs à couper**
-b	**Sélection sur le numéro d'octet**
-s (avec –f)	**Supprime les lignes vides .**

Commande de manipulation de contenu des Fichiers

La commande cut

Exemple :

```
cut -c1 /etc/passwd
```

affichera le premier caractère (colonne) du fichier /etc/passwd. Il existe d'autres spécifications :

```
cut -d: -f6 /etc/passwd
```
(avec séparateur de champs)

affichera le 6eme champ du fichier /etc/passwd, dont le séparateur de champs est le caractére double point (``:'').

Commande de manipulation de contenu des Fichiers

LA COMMANDE SORT

Permet de trier les lignes

- **Syntaxe : sort [-o|-n|-r|-R|-u] nom_fichier [nom_fichier]**

```
guest-yE7s3q@baqloul-Aspire-E1-531:~$ sort -r file
Vidéos
unix2
unix
Téléchargements
Public
Musique
Modèles
Images
file
examples.desktop
Documents
Bureau
```

Commande de manipulation de contenu des Fichiers

OPTIONS DE LA COMMANDE SORT

Option	Signification
-o	**Ecrire le résultat dans un fichier**
-u	**Elimine les lignes identiques**
-r	**Trier en ordre inverse**
-R	**Trier aléatoirement**
-n	**Trier des nombres**

Commande de manipulation de contenu des Fichiers

LA COMMANDE DIFF

- Permet de comparer deux fichiers textes
- Syntaxe: diff [options] fichier1 fichier2

```
karima@karima-HP-EliteBook-8440p:~$ cat > fich1
Linux est un système d'exploitation
multitaches
multiutilisateurs
un système très structuré
^Z
[9]+  Stopped                 cat > fich1
karima@karima-HP-EliteBook-8440p:~$ cat > fich2
Linux est un système d'exploitation
multitaches
multiutilisateurs
^Z
[10]+  Stopped                 cat > fich2
karima@karima-HP-EliteBook-8440p:~$ diff fich1 fich2
4d3
< un système très structuré
```

Commande de manipulation de contenu des Fichiers

OPTIONS DE LA COMMANDE DIFF

Option	Signification
-i, --ignore-case	Ne fait pas de différence entre les caractères minuscules et majuscules.
-w, --ignore-all-space	Ne tient pas compte du caractère espace.
-B	Ne tient pas compte des lignes blanches.

Commande de manipulation de contenu des Fichiers

wc permet de compter le nombre de ligne d'un fichier, mais aussi le nombre de mot ou de caractères.

paste permet la fusion de lignes de fichiers, les options sont les suivantes :

-dx Le caractère x définit le séparateur de champ

-s Les lignes sont remplacées par des colonnes

QUELQUES AUTRES COMMANDES

echo ECHO
Affiche à l'écran le texte qui suit la commande echo.

Exemple

ubuntu@pc :~$ echo bonjour

bonjour

QUELQUES AUTRES COMMANDES

Uname

elle affiche les informations systèmes sur la machine sur laquelle elle est exécutée. Elle est apparue dans PWB/UNIX

Options principales :

-m : affiche le type de la machine.

-n : affiche le nom de la machine.

-r : affiche le numéro de version du système.

-s : affiche le nom du système.

-a : affiche toutes les informations ci-dessus.

QUELQUES AUTRES COMMANDES

su ***switch user***

Permet de changer l' identité de l'utilisateur

Exemple

ubuntu@pc :~$ su

Login: root

QUELQUES AUTRES COMMANDES

Sudo : exécuter en se substituant à l'utilisateur

Cette commande permet à l'administrateur système d'accorder à certains utilisateurs (ou groupes d'utilisateurs) la possibilité de lancer une commande en tant qu'administrateur, ou comme autre utilisateur.

```
$ sudo reboot
```

Lance la commande reboot avec les droits de l'utilisateur root

QUELQUES AUTRES COMMANDES

Reboot : est une commande permettant de redémarrer le système.

Shutdown : est une commande permettant l'extinction de la machine à partir du terminal.

Exemple

ubuntu@pc :~$ Shutdown

QUELQUES AUTRES COMMANDES

LE MANUEL DE LINUX

La commande man permet de rechercher des informations sur les commandes. Man est le manuel Unix en ligne. Cette commande recherche les informations, le cas échéant, dans deux répertoires et leurs sous–répertoires :
/usr/man
/usr/local/man

ubuntu@pc :~$ man

What manual page do you want?

ubuntu@pc :~$ man ls

Les filtres de recherche

Les expressions régulières

Définition – Formule qui représente une chaîne de caractères. Composée de caractères et d'opérateurs.

Utilisation – On recherche alors non pas un mot ou une simple chaîne de caractères mais une suite de caractères qui correspondent au critères énoncés par la formule

– Certains opérateurs doivent être précédés d'un \ pour ne pas entrer en conflit avec le shell, ainsi : {, }, , (,) et | seront écrits \{, \}, \, \(, \) et \|. C'est aussi le cas de l'espace

LES EXPRESSIONS RÉGULIÈRES

Rappel :

- * N'importe quelle séquence de caractères
- ? N'importe quel caractère
- [] N'importe quel caractère choisi dans les caractères donnés entre crochets
- [^] n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
- [–] n'importe quel caractère dans la plage de caractères donnés entre crochets

Exemples :

```
$ ls *
$ ls ?ateau
$ ls *ateau
$ ls [gr]ate*
$ ls [ ∧br]ateau
$ ls [a-c]*
```

LES MÉTA CARACTÈRES

Les expressions régulières sont aussi des suites de caractères permettant de faire des sélections. Elles sont utilisé avec certaines commandes comme *grep*.

Les différentes expressions régulières sont :

Caractère spécial	Signification
|	Ou logique, l'expression située avant ou après doit apparaître.
(...)	Groupement de caractères.
[...]	Un caractère à cette position parmi ceux indiqués.
. (point)	Un caractère quelconque.
+	Répétition, le caractère placé avant doit apparaître au moins une fois.
*	Répétition, le caractère situé avant doit apparaître de zéro à n fois.
?	Le caractère situé avant doit apparaître une fois au plus.
{n}	Le caractère situé avant doit apparaître exactement n fois.
{n,}	Il apparaît n fois ou plus.
{n,m}	Il apparaît entre n et m fois.
^	En début de chaîne.
$	En fin de chaîne.

LES EXPRESSIONS RÉGULIÈRES

Représentation :

- Un caractère compris entre 'a' et 'z' : [a-z]
- Un caractère compris entre '0' et '9' : [0-9]
- Le début d'une ligne : ^
- La fin d'une ligne : $
- Un choix entre deux chaînes : \|
- Et en combinant avec le choix : Caractère compris entre 'a' et 'z' ou 'A' et 'Z' : [a-zA-Z]
- Répétitions d'une occurrence
  - Exactement 2 répétitions de 'x' : x\{2\}
  - Entre 2 et 5 répétitions de 'x' : x\{2, 5\}
  - Au moins 2 répétitions de 'x' : x\{2, \}

LES EXPRESSIONS RÉGULIÈRES

EXEMPLE :

- l'expression [a-z][a-z] * cherche les lignes contenant au minimum un caractère en minuscule. [a-z] caractère permis, [a-z]* recherche d'occurrence des lettres permises.
- L'expression **^[0-9]\ {4\}$** a pour signification, du début à la fin du fichier $, recherche les nombres [0-9] de 4 chiffres \ {4\}.
- L'expression [a-z][a-z] * cherche les lignes contenant au minimum un caractère en minuscule. [a-z] caractère permis, [a-z]* recherche d'occurrence des lettres permises.
- L'expression **^[0-9]\ {4\}$** a pour signification, du début à la fin du fichier $, recherche les nombres [0-9] de 4 chiffres \ {4\}.
- Remarque : Le caractère ^ n'a la signification de non qu'entre crochets.

Quelle est la différence entre [^abc] et ^[abc] ?

LES EXPRESSIONS RÉGULIÈRES

EXEMPLE :

- \[02-57]\ 0, 2, 3, 4, 5 ou 7
- \[a-d5-8X-Z]\ a, b, c, d, 5, 6, 7, 8, X, Y ou Z
- \[0-5-]\ 0, 1, 2, 3, 4, 5 ou -
- \[^0-9]\ pas un chiffre
- Un code permanent contient 4 majuscules suivies de 8 chiffres : [A-Z]\{4\}[0-9]\{8\}
- Un code permanent contient 12 caractères lettres ou chiffres : [A-Z0-9]\{12\}

EXEMPLE PLUS AVANCÉ:

Repérer les lignes contenant une date qui est exprimée par JJ/MM/AA ou bien JJ/MM/AAAA

Jours 01 à 09 : 0[1-9]

Jours 10 à 29 : [1-2][0-9]

Jours 30 et 31: 3[0-1]

On obtient alors : \(0[1-9]\|[1-2][0-9]\|3[0-1]\)

Mois 01 à 09 : 0[1-9]

Mois 10 à 12 : 1[0-2]

on obtient alors : \(0[1-9]\|1[0-2]\)

Une année contient simplement 2 ou 4 chiffres :

on obtient alors : \([0-9]\{2\}\|[0-9]\{4\}\)

Concaténation des expressions formalisant les jours, mois et années avec un '/' entre elles : \(0[1-9]\|[1-2][0-9]\|3[0-1]\)/\(0[1-9]\|1[0-2]\)/\([0-9]\{2\}\|[0-9]\{4\}\)

LA COMMANDE GREP

La commande grep permet de rechercher une chaîne de caractères dans un fichier. Les options sont les suivantes :

-v affiche les lignes ne contenant pas la chaîne
-c compte le nombre de lignes contenant la chaîne
-n chaque ligne contenant la chaîne est numérotée
-x ligne correspondant exactement à la chaîne
-l affiche le nom des fichiers qui contiennent la chaîne

Exemple avec le fichier carnet-adresse :

```
olivier:29:0298333242:Brest
marcel:13:0466342233:Gardagnes
myriam:30:0434214452:Nimes
yvonne:92:013344433:Palaiseau
```

La commande :

$grep ^[g-n] carnet-adresse

Permet d'obtenir toutes les lignes commençant par les caractères compris entre g et n.

$grep Brest carnet-adresse

Permet d'obtenir les lignes contenant la chaîne de

LA COMMANDE GREP

- Les options de la commande grp

Option	Signification
-v	Afficher les lignes qui ne correspondent pas au critère
-c	Retourner le nombre de lignes qui correspond au critère
-i	Ne pas tenir en compte la casse
-n	Chaque ligne trouvée est retournée avec son numéro de ligne
-l	Lister uniquement les noms de fichiers qui correspondent au critère
-a	Traiter un fichier binaire comme s'il s'agissait de texte
-R, -r, -- recursive	Lire tous les fichiers à l'intérieur de chaque répertoire, récursivement

LA COMMANDE GREP

Remarque : Différence entre grep et find :

Find : sert à parcourir l'arborescence pour la recherche d'un nom de fichier

Grep : sert à chercher la chaîne de caractère qui lui est spécifiée dans les fichiers donnés en argument

EXERCICE

Vérification très sommaire de la validité d'une adresse IP, « 4 décimaux de 3 chiffres qui sont séparés par des . ».

LA COMMANDE GREP

SOLUTION

```
echo $IP | grep '([0-9]{1,3}\.){3}([0-9]{1,3}')
```

Voici comment cette ligne est décomposée :

- ✓ ([0-9]{1,3}\.){3} :www.xxx.yyy.
- ✓ [0-9] : un caractère entre 0 et 9
- ✓ {1,3} : répété entre une et trois fois, donc x, xx ou xxx
- ✓ \. : suivi d'un point
- ✓ {3} : le tout trois fois
- ✓ Puis [09]{1,3} : .zzz
- ✓ [0-9] : un caractère entre 0 et 9
- ✓ {1,3} : répété entre une et trois fois

Exemples de Recherche avec find

Opérateurs logiques

!	Non logique
-a	Liaison par et logique de deux critères
-o	Liaison par ou logique de deux critères

Exemple avec find

Les fichiers n'appartenant pas à l'utilisateur olivier:

```
find . ! -user olivier -print
```

Recherche des fichiers qui ont pour nom a.out et des fichiers se terminant par .c:

```
find . \(-name a.out -o -name "*.c"\) -print
```

Recherche des fichiers qui ont pour nom core et une taille supérieure à 1Mo.

```
find .\ (-name core -a -size +2000 \) -print
```

find utilisé avec d'autres commandes UNIX:

```
find . -type f -print| grep "test"
```

Exemples de Recherche avec find

- Remarque

Les opérateurs and et or n'ont par la même priorité, donc, il faut positionner les parenthèses pour interpréter le ou logique avant le et logique. Les parenthèses étant à leur tour des caractères spéciaux, on doit les désactiver.

LA COMMANDE FIND

EXERCICES: FIND

- 1) Afficher tous les fichiers qui commencent par 'a'`majuscule ou minuscule.
- 2) Afficher tous les fichiers qui se composent de 3 caractères dont le dernier est 'M'.
- 3) Sauvegarder tous les noms de fichiers se terminant soit par a, b ou c dans un fichier F1.
- 4) Créer les fichiers suivants dans un répertoire :a1, a2, a3……a9 et afficher les 5 premiers
- 5) Afficher tous les noms de fichiers répertoires commençant par une lettre majuscule.
- 6) Afficher tous les noms de fichiers de l'utilisateur « webmaster1 » qui appartient au groupe « webos ».
- 7) Afficher tous les fichiers normaux ayant pour taille plus de 150 Ko.
- 8) Afficher tous les fichiers dont les dernières modifications remontent à moins de trois jours.
- 9) Afficher tous les répertoires ayant les autorisations d'accès 'rwxr-xr-x '.
- 10) Supprimer tous les fichiers auxquels vous n'avez pas accédés depuis plus de deux semaines.
- 11) Afficher tous les fichiers répertoires ou les fichiers dont le nom se termine par le caractère 'a'.
- 12) Afficher tous les fichiers qui n'appartiennent pas à l'utilisateur « userxx1 »

SOLUTION: FIND

- $find / -name "[a-A] *" -print
- $find / -name "??M" -print
- $find / -name "*[a b c] " > F1
- $touch a1 a2 a3 a4 a5 a6 a7 a8 a9|find -name "a[1-5]"
- $find / -type d - name "[A-Z] *"
- $find / -user "webmasters1" -a -group "webos"
- $find /-type f -size "+150K"
- $find /-type f -mtime -3
- $find /-type d -perm 755
- $find /-type f -atime +15 -exec rm
- $find /(-type d -o type f) -name "*a"
- $find ! -user "user xg"

EXERCICES: CUT, GREP, TAIL, HEAD, TR

A partir du fichier etc/passwd

- Affichez uniquement la ligne concernant l'utilisateur root
- Affichez uniquement les login de tous les utilisateurs triés (1èr champ).
- Affichez les correspondances login (1) numéro d'utilisateur (3), nom réel(5)
- affichez le nom des utilisateurs du groupe(4) 1000 et leur groupe.
- Combien y'en a-t-il ?
- Quels sont les fichiers de /etc/ contenant des nombres de plus de trois chiffres
- Afficher la 7ième ligne de ce fichier
- Afficher le fichier /etc/passwd en remplaçant les caractères / par des X.

EXERCICE

- Lister le contenu du répertoire /dev.
- Lister le contenu du fichier /etc/passwd.
- Afficher par ordre alphabétique les utilisateurs définis dans le fichier /etc/passwd.
- Rechercher tous les fichiers du répertoire /etc contenant la chaine de caractères "root".
- Rechercher la localisation du fichier "stdio.h" dans le système de fichier de votre installation.
- Utiliser les commandes de comparaison de fichiers de Linux pour tester le contenu d'un même fichier texte en version Unix et Dos.
- Combien de lignes, de mots et de caractères comportent les fichiers "montexte.unix", "montexte.dos" et "montexte.mac"?

Les alias

Il permet de donner un synonyme à une commande ou à un groupement de commandes.

Cela permet de gagner du temps et de simplifier les commandes.

On peut mettre un certain nombre d'alias dans le fichier .profile

La syntaxe est la suivante : alias nom_alias = ' commande '

Exemple :

$ alias l='ls-al | more' # est une commande très #pratique.
$alias goappli='cd /home/compte_appli/appli'

Pour obtenir la liste des alias définis : alias

REDIRECTION DES ENTRÉES/SORTIES

LES SÉPARATEURS CONDITIONNELS DE COMMANDES

Il est possible de contrôler la séquence d'exécution de commandes en utilisant des séparateurs conditionnels.

- Le séparateur && permet d'exécuter la commande qui le suit, si et seulement si la commande qui le précède a été exécutée sans erreur.

- Le séparateur || permet d'exécuter la commande qui le suit si et seulement si la commande qui le précède a été exécutée avec erreur.

LES SÉPARATEURS CONDITIONNELS DE COMMANDES

Exemple
Suppression des fichiers si la commande cd projet1 a été correctement exécutée.

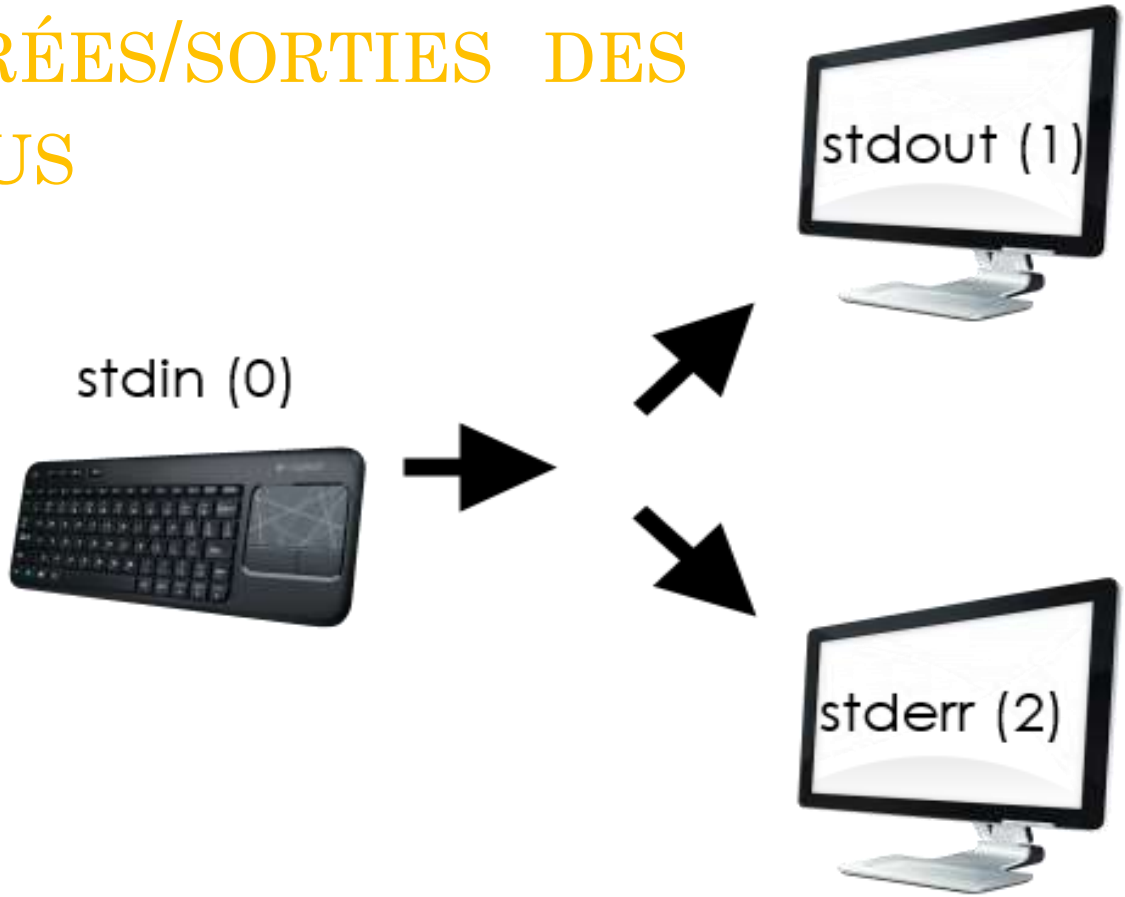
```
$ cd projet1 && rm *
```

Exemple
Si le répertoire projet1 n'existe pas, alors il sera créé par la commande mkdir.

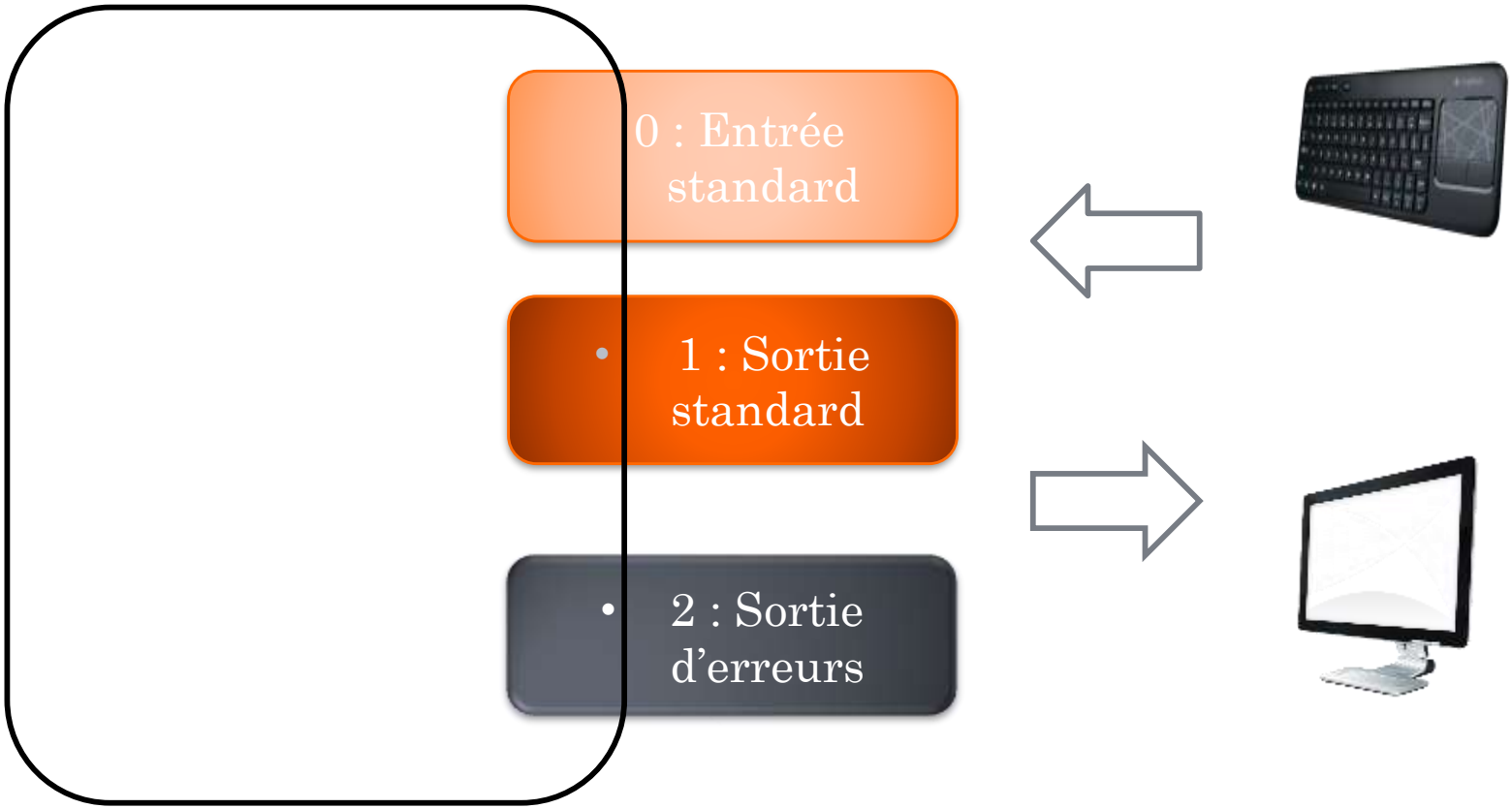
```
$ cd projet1 || mkdir projet1
bash: cd: projet1: No such file or directory
$ ls
projet1
```

Les Entrées sorties d'un processus

Les entrées/sorties des processus



LES ENTRÉES SORTIES D'UN PROCESSUS



Les touches de redirection

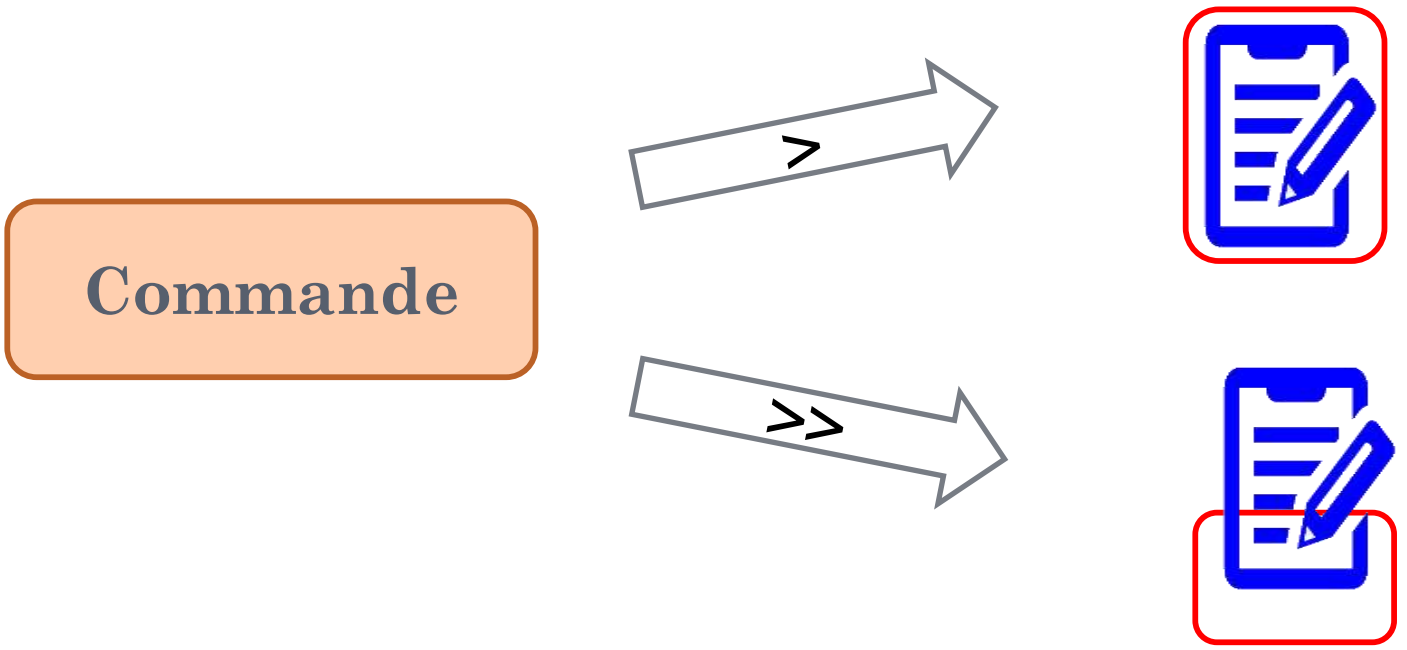
La touche « > » redirige la sortie standard vers un fichier et l'écrase s'il existe déjà.

Exemple:

```
guest-D7S7vy@baqloul-Aspire-E1-531:~$ touch fichier1
guest-D7S7vy@baqloul-Aspire-E1-531:~$ ls >fichier1
guest-D7S7vy@baqloul-Aspire-E1-531:~$ cat fichier1
Bureau
Documents
examples.desktop
Fichier
fichier1
Images
Modèles
Musique
Public
Téléchargements
unix
Vidéos
```

Les touches de redirection

- La touche « **>>** » redirige la sortie standard à la fin d'un fichier et le crée s'il n'existe pas.



Les touches de redirection

La touche « < » envoie le contenu d'un fichier à une commande.

<

Commande

La sortie du fichier devient l'entrée de la commande

Les touches de redirection

La touche « << » passe la console en mode saisie au clavier, ligne par ligne.

Toutes ces lignes seront envoyées à la commande lorsque le mot-clé de fin aura été écrit.

Exemple:

```
guest-D7S7vy@baqloul-Aspire-E1-531:~$ touch document
guest-D7S7vy@baqloul-Aspire-E1-531:~$ sort -n << document
> 5
> 3
> 8
> 2
> 0
> 7
> document
0
2
3
5
7
8
```

Les touches de redirection

La touche « 2> » redirige les erreurs dans un fichier (s'il existe déjà, il sera écrasé).

```
khawla@KHAWLA-UBUNTU:~$ cat toto
cat: toto: Aucun fichier ou dossier de ce type
khawla@KHAWLA-UBUNTU:~$ cat toto 2>erreur.log
khawla@KHAWLA-UBUNTU:~$ ls
Bureau     erreur.log  examples.desktop  les     Musique  SMI   touches  Vidéos
Documents  essai-grep  Images           Modèles  Public   smi4  unix
khawla@KHAWLA-UBUNTU:~$ cat erreur.log
cat: toto: Aucun fichier ou dossier de ce type
khawla@KHAWLA-UBUNTU:~$ cat toto 2>touches
khawla@KHAWLA-UBUNTU:~$ cat touches
cat: toto: Aucun fichier ou dossier de ce type
```

Les touches de redirection

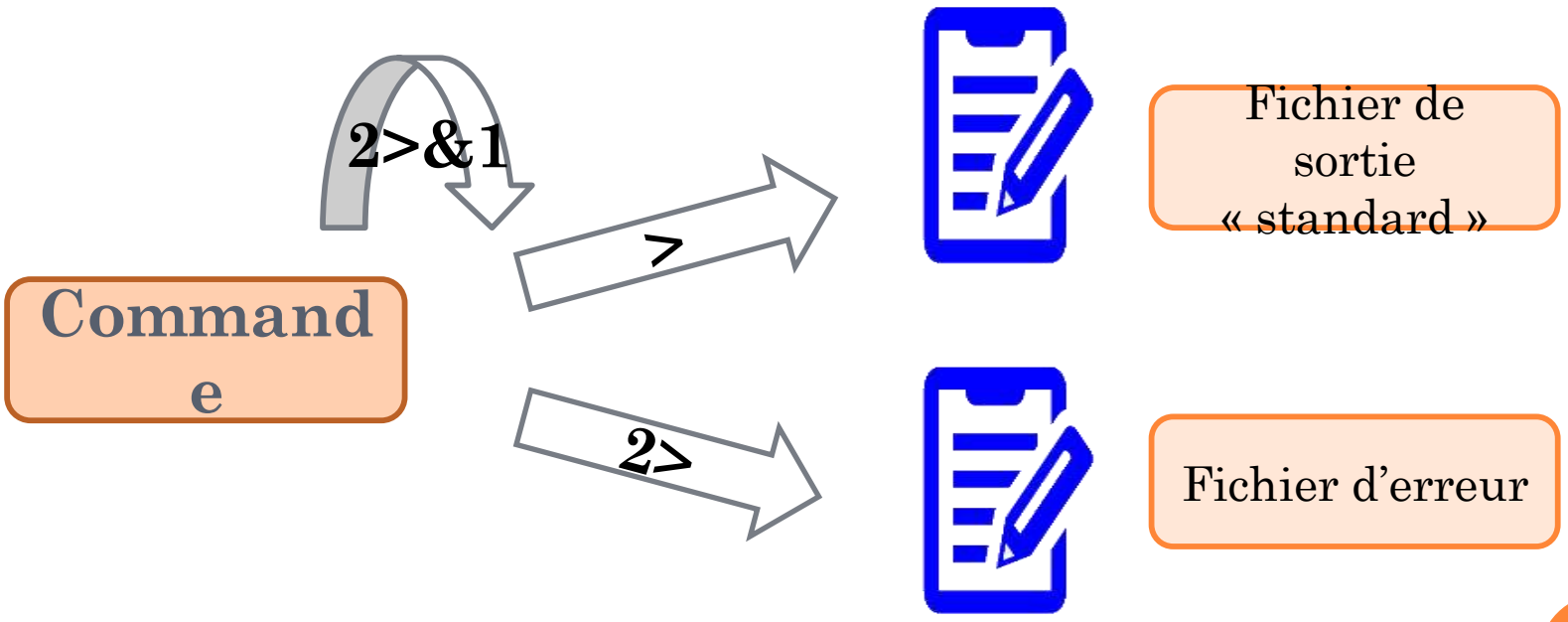
La touche « 2>> » redirige les erreurs à la fin d'un fichier (s'il n'existe pas il sera créé) .

Exemple:

```
khawla@KHAWLA-UBUNTU:~$ cat toto 2>> unix
khawla@KHAWLA-UBUNTU:~$ cat unix
Bureau
Documents
essai-grep
examples.desktop
Images
Modèles
Musique
Public
SMI
smi4
unix
Vidéos
Bureau
Documents
essai-grep
examples.desktop
Images
Modèles
Musique
Public
SMI
smi4
unix
Vidéos
cat: toto: Aucun fichier ou dossier de ce type
```

Les touches de redirection

La touche « 2>&1 » redirige les erreurs au même endroit et de la même façon que la sortie standard.

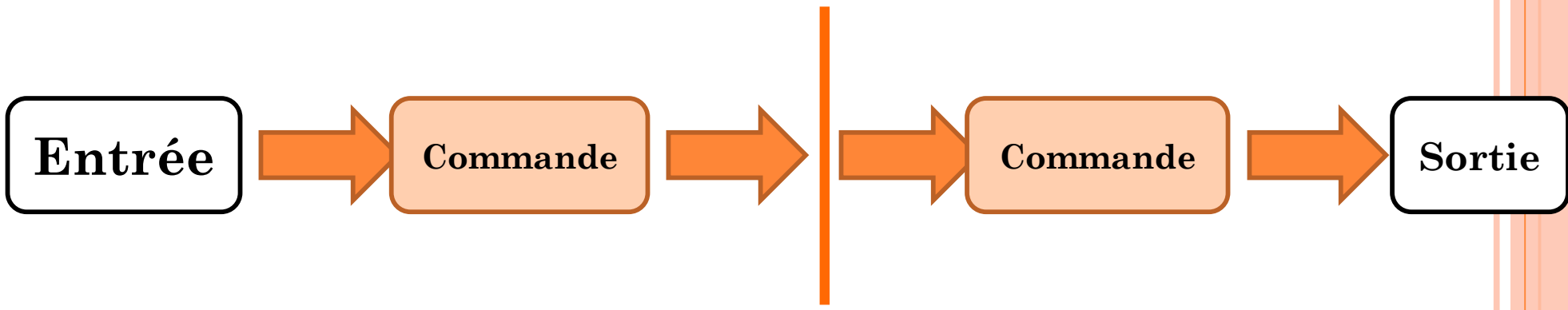


Les touches de redirection

La touche « **2>/dev/null** » supprime l'ancienne redirection de la sortie d'erreur.

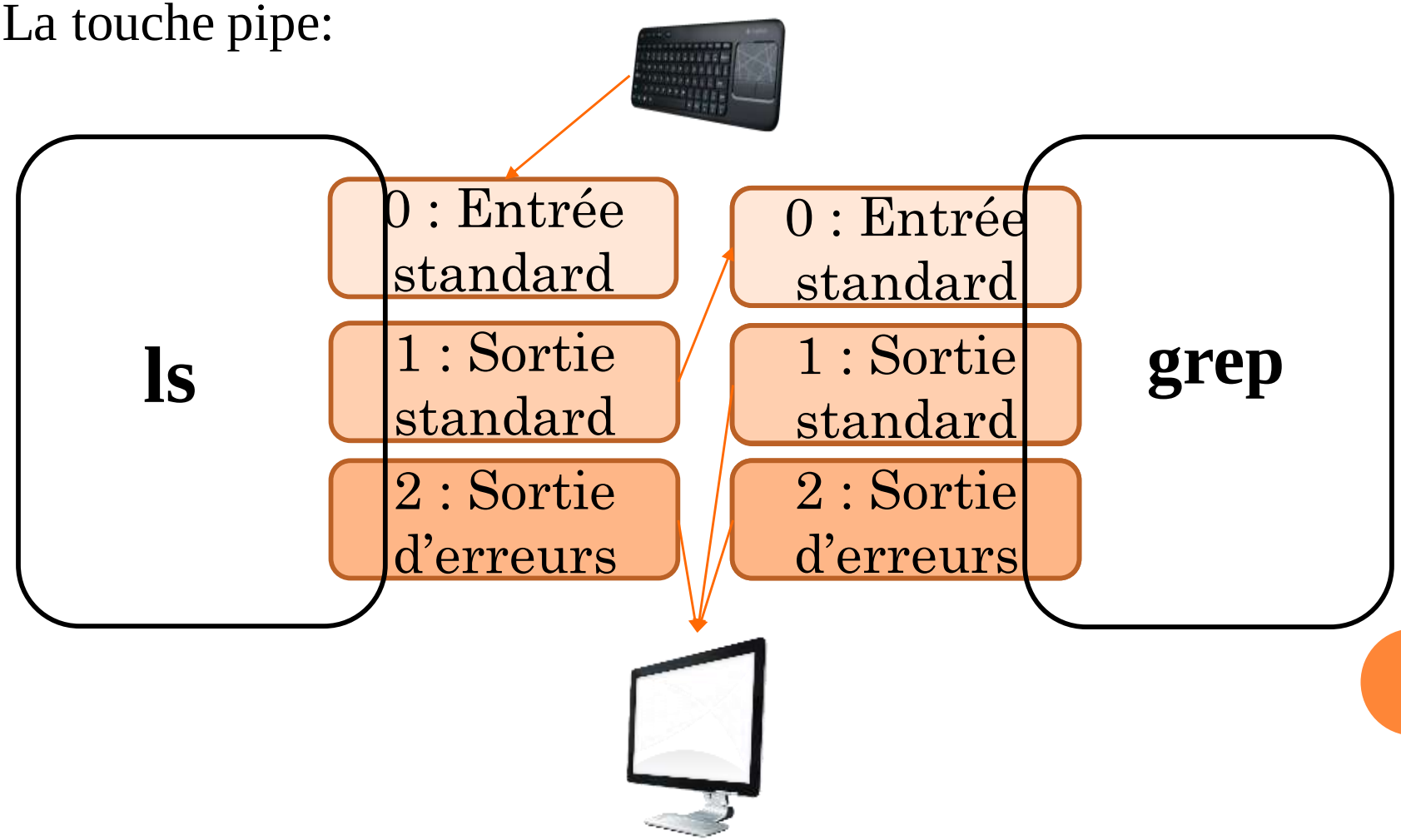
Les touches de redirection, les pipes

Pipe « | »



Les touches de redirection, les pipes

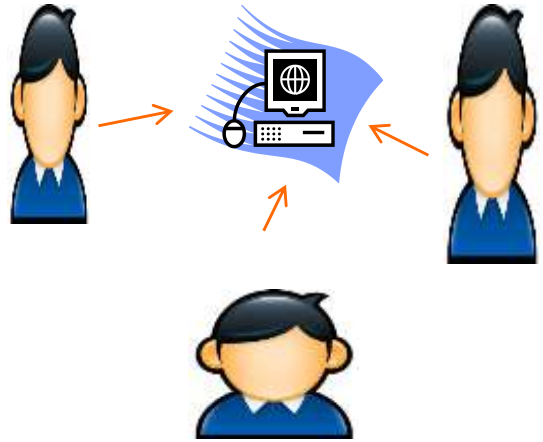
La touche pipe:



Chapitre 3
Gestion des utilisateurs

Les utilisateurs

Linux est un système multiutilisateurs :



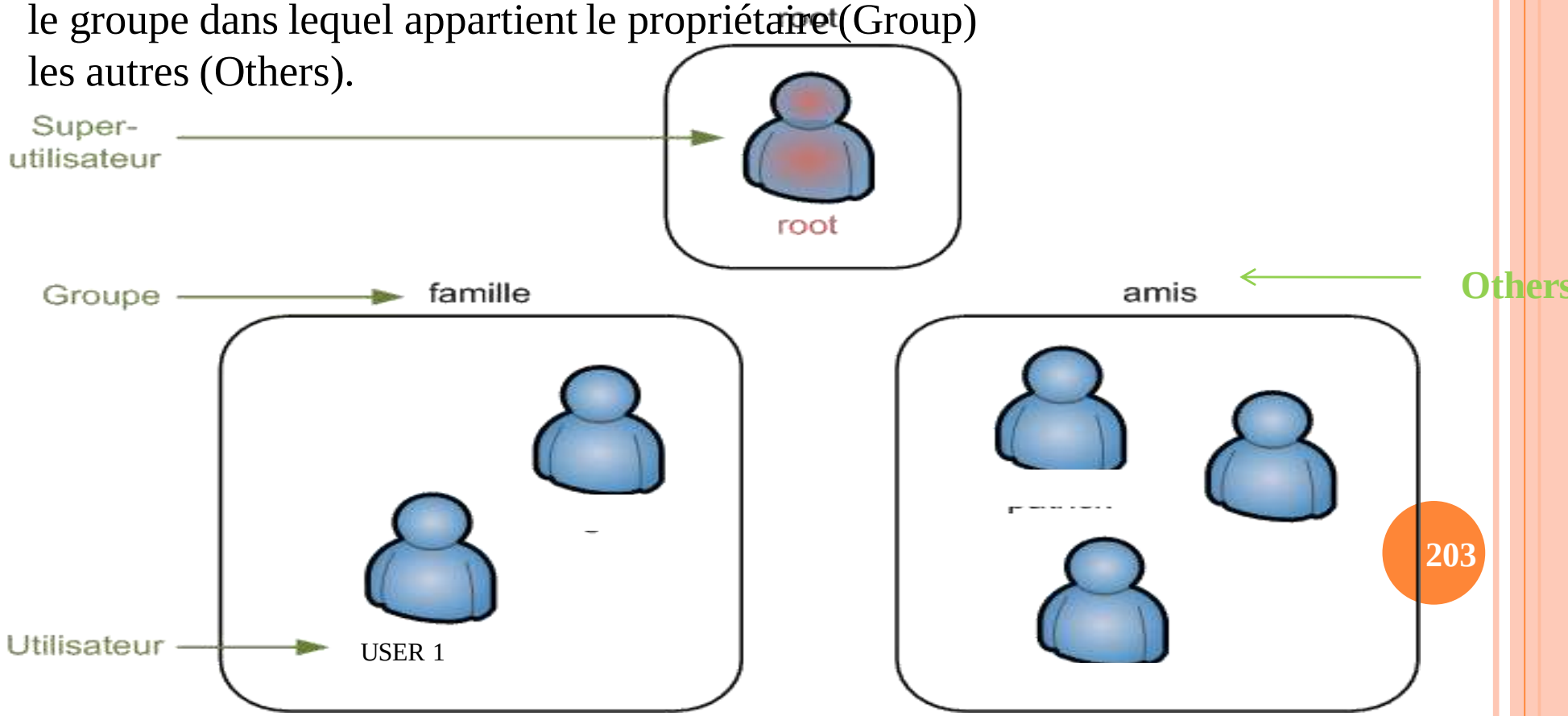
Pour préserver la sécurité et la confidentialité des donnée, Linux assure un système d'identification et de gestion des utilisateurs

Un utilisateur est donc celui qui s'est logué en donnant un nom de login. Ce login permet de définir l'utilisateur et tracer l'activité menée en son nom.

Les utilisateurs

- Types d'utilisateurs : Il existe deux types d'utilisateurs
  - Le root ou le super utilisateurs a l'ensemble des droits sur le système
  - Utilisateur ordinaire ayant des droits bien définies sur le système

- Classes d'utilisateurs : Il existe 3 classes d'utilisateurs :
  - le propriétaire du fichier (User)
  - le groupe dans lequel appartient le propriétaire (Group)
  - les autres (Others).



Les utilisateurs

Utilisateur : personne autorisée a se connecter sur le système.

- son accès est autorisé après identification :
  - par son nom d'utilisateur (username ou login).
  - par son mot de passe associe (password).
- il dispose d'une zone privée sur le disque,
  - généralement située dans /home/username ou dans /users/username
  - sur laquelle il a tous les droits.
- la taille de cette zone privée (i.e. le nombre d'octets qu'il est possible d'y stocker) peut être restreinte a un quota.
- il est identité par un numéro unique ou UID (User IDentication number).
- il appartient a un ou plusieurs groupes (mais un groupe dans lequel il est par défaut).

Les groupes

groupe : ensemble de personnes ou de groupes.

- les groupes sont utilisés pour contrôler les accès au sein du système.
- le changement de groupe est (éventuellement) contrôlé par un mot de passe.
- il est géré par un administrateur spécifique au groupe (ou par défaut le super-utilisateur).
- il est identité par un numéro unique ou GID (Group IDentication number).

Les utilisateurs et les groupes

Fichiers de description des utilisateurs et des groupes

- /etc/passwd pour les utilisateurs
- /etc/group pour les groupes

Les droits d'accés

Sous Linux chaque fichier (ou répertoire) possède un ensemble d'attributs définissant les droits d'accès à ce fichier pour tous les utilisateurs du système.

- A sa création, un fichier appartient à son auteur.
- Le propriétaire du fichier peut ensuite distribuer ou restreindre les droits d'accès sur ce fichier

- Pour chaque classe d'utilisateurs, il y a 3 types d'accès à un fichier donné :
  - r : en lecture (Read).
  - w : en écriture (Write).
  - x : en exécution (eXecute).

Les droits d'accés

Propriétaire

Membres du groupe propriétaire

Autres utilisateurs

Les modes d'autorisation:

- ✓ **autorisation d'écriture (w)**
- ✓ **autorisation de lecture (r)**
- ✓ **autorisation d'exécution (x)**

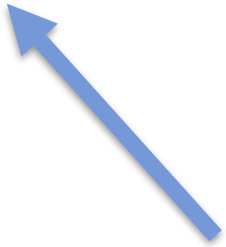
Les Droits d'accès des fichiers et des répertoires

- Un droit d'accès est un modèle qui décrit pour chaque fichier ou chaque répertoire, qui a le droit de lire, d'exécuter ou de modifier.

Droits du Propriétaire	Droits du Groupe	Droits du Reste Du Monde
r w x	r w x	r w x

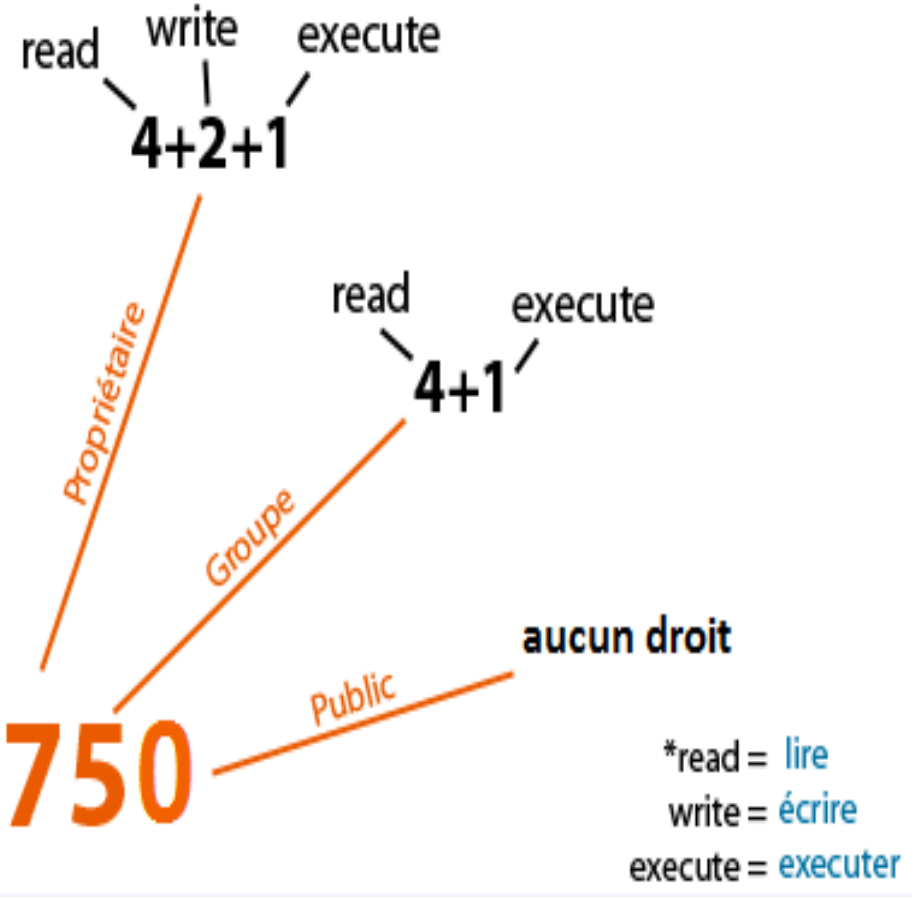
Afficher les droits d'accès d'un fichier ou d'un répertoire

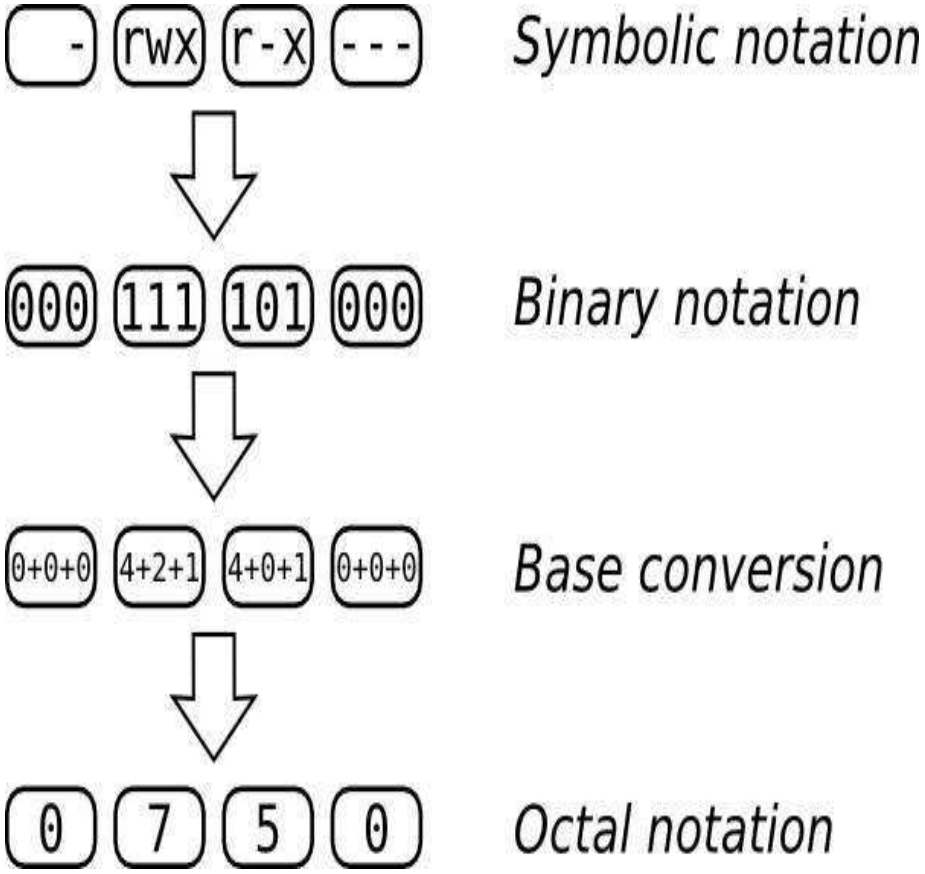
d rwx r-x r-x



```
karima@karima-HP-EliteBook-8440p:~$ ls -l
total 72
drwxr-xr-x 2 karima karima 4096 déc.   3 19:29 Bureau
drwxrwxr-x 2 karima karima 4096 déc.  13 22:17 docs
drwxr-xr-x 2 karima karima 4096 déc.   3 19:29 Documents
-rw-r--r-- 1 karima karima 8980 déc.   3 19:19 examples.desktop
-rw-rw-r-- 1 karima karima    0 déc.  13 21:23 exercice
-rw-rw-r-- 1 karima karima   96 déc.  17 12:58 fich1
-rw-rw-r-- 1 karima karima   67 déc.  17 12:59 fich2
-rw-rw-r-- 1 karima karima    0 déc.  17 13:04 fichier1
-rw-rw-r-- 1 karima karima    0 déc.  17 13:04 fichier2
drwxrwxr-x 3 karima karima 4096 déc.  13 22:25 images
drwxr-xr-x 2 karima karima 4096 déc.  17 15:42 Images
-rw-rw-r-- 1 karima karima    0 déc.  13 22:43 karima
-rw-rw-r-- 1 karima karima   14 déc.  17 13:08 mehdi
drwxr-xr-x 2 karima karima 4096 déc.   3 19:29 Modèles
drwxr-xr-x 2 karima karima 4096 déc.   3 19:29 Musique
```

Droits d'accés





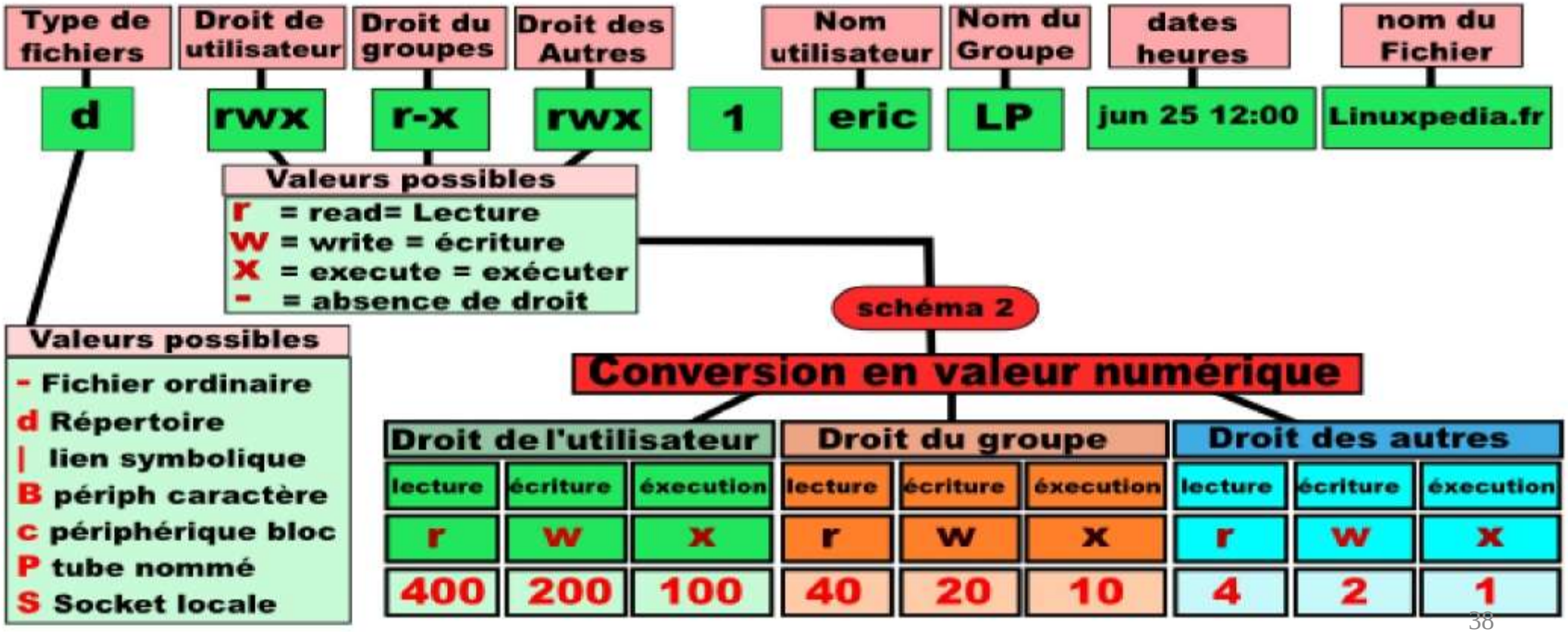
Droit d'accés à un répertoire

Accès	Fichier	Répertoire
R	**lire le contenu du fichier**	**afficher contenu du répertoire**
W	**modifier le contenu du fichier**	**ajouter et supprimer fichier**
X	**lancer un fichier exécutable**	**rendre le répertoire courant**

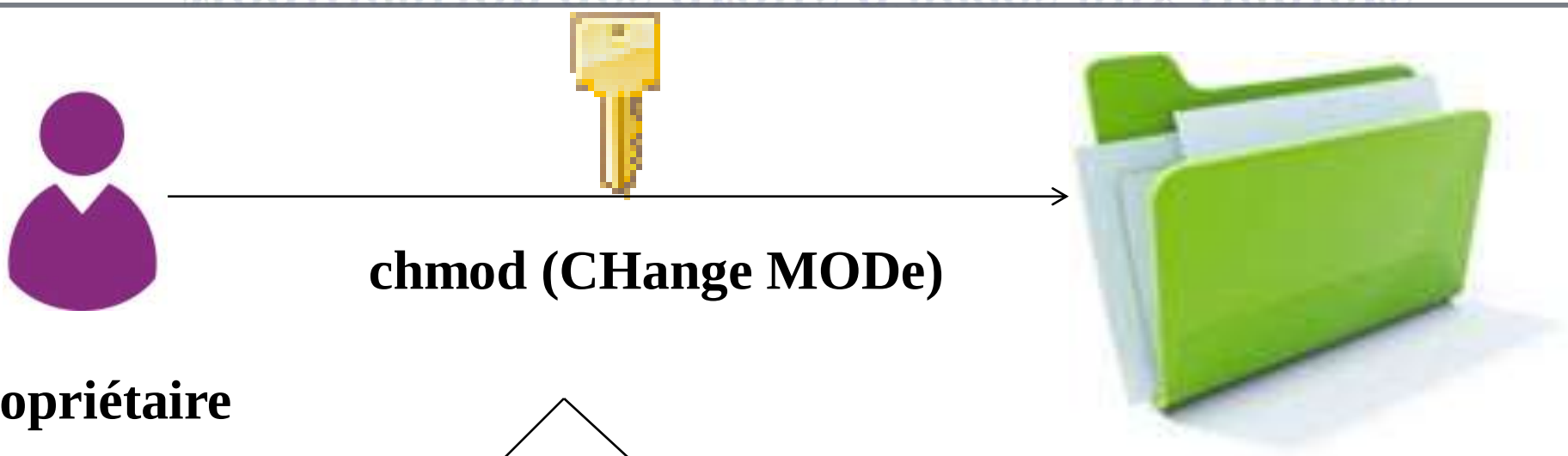
D'une façon générale

La commande ls -l en console permet d'afficher les détails d'un répertoire ou d'un fichier. On y découvre alors une somme importante d'information.

Exemple d'affichage après la commande ls-l



Modification des droits d'accés aux fichiers



chmod (CHange MODe)

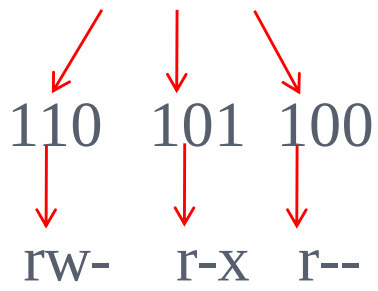
Propriétaire

Description des protections par un nombre octal

Mode symbolique

Modification des droits d'accés aux fichiers : le mode octal

- Les droits sont représentés par un nombre octal (Base 8)
  - De 1 à 7
- La représentation binaire (base 2) donne le détail des droits
  - Exemple : 6 5 4



- Ce mode permet de modifier tous les droits en même temps
  - A utiliser avec précaution
  - Très efficace pour s'assurer que tous les fichiers ont les mêmes droits
  - Utilisé pour sécuriser les accès des utilisateurs aux fichiers

Modification des droits d'accés aux fichiers : le mode octal

Exemple:

On veut modifier la protection du fichier toto de la manière suivante :

rx rw rwx

1. autorisant un accès lecture et exécution au propriétaire,
2. lecture et écriture au groupe
3. un accès sans restriction aux autres (cas très rare).

$ chmod 567 toto

$ ls -l toto

rx rw rwx Jul 20 17:01 toto

Modification des droits d'accés aux fichiers : mode symbolique

Le deuxième mode:

le mode symbolique, permet une description absolue ou relative des droits d'accès, comme suit :

chmod [qui]op[permission] fich/rep

combinaison de lettre u (user), g (groupe), o (other=autre) ou a (all=tous) pour ugo

+ permet d'ajouter un droit d'accès,
- de supprimer un droit d'accès
= d'affecter un droit de manière absolue

permission :
r (read=lecture),
w (write=écriture),
x (exécution).

Modification des droits d'accés aux fichiers : mode symbolique

$ chmod u -w file

Supprime le droit d'écriture au propriétaire.

$ chmod g+r file

Ajoute le droit de lecture pour le groupe.

$ chmod ug=x file

Accès uniquement en exécution pour le propriétaire et le groupe, pas de modification pour les autres.

Modification des droits d'accés aux fichiers : mode symbolique

Exemple :

chmod ugo+x monRep

Ajoute l'exécution (ouverture) du répertoire *monRep* à tous (propriétaire, groupe, autres)

chmod go-wx monRep

Supprime l'autorisation d'écriture et l'exécution de *monRep* au groupe et aux autres

chmod u=rw, go=r MonFichier

Fixe l'autorisation de lecture et d'écriture au propriétaire et une autorisation de lecture au groupe et aux autres.

Modification des droits d'accés aux fichiers : mode octal

Exemple:

chmod 644 MonFichier

Exactement la même chose , mais en utilisant les valeurs octales (Nota : 6 = 4+2 = lecture + écriture)

chmod u=rw, g=r, o= MonFichier

Fixe l'autorisation d'ouverture et de lecture au propriétaire, uniquement la lecture au groupe et interdit tout accès aux autres.

chmod 640 MonFichier

Exactement la même chose que ci-dessus mais en utilisant les valeurs octales

Modification des propriétaire et du groupe

- **chown permet de changer le propriétaire d'un fichier, seul l'administrateur peut modifier le propriétaire d'un fichier ou d'un répertoire.**

- **chgrp permet le changement de groupe, à condition que l'utilisateur fasse partie du nouveau groupe et soit propriétaire de ces fichiers ou répertoires**

Exemple

$ chgrp info f1

§ possible si j'appartiens au groupe info

$ chown soline f1

chown: f1: Operation not permitted

§ possible seulement pour root

$ sudo chown soline fichier1

Modification des droits par défauts : umask

Quand vous créer un fichier, par exemple avec la commande touch, ce fichier par défaut possède certains droits.

Ce sont 666 pour un fichier (-rw-rw-rw-) et 777 pour un répertoire (-rwxrwxrwx), ce sont les droits maximum.

Vous pouvez faire en sorte de changer ces paramètres par défaut. La commande umask est là pour ça, elle définit les permissions par défaut d'un répertoire ou d'un fichier créé.

Syntaxe : umask x

Où x est un nombre exprimé sous forme octale qui déterminera les permissions

Modification des droits par défauts : umask

Pour un fichier :

- Si vous tapez umask 022
  - vous partez des droits maximum 666 et vous retranchez 022
    - on obtient donc 644
    - par défaut les fichiers auront comme droit (rw-r--r--).
- Si vous tapez umask 244
  - vous partez des droits maximum 666 et vous retranchez 244
    - on obtient donc 422
    - par défaut les fichiers auront comme droit (r---w--w-).

Modification des droits par défauts : umask

Pour un répertoire :

- Si vous tapez umask 022
  - vous partez des droits maximum 777 et vous retranchez 022
    - on obtient donc 755
    - par défaut ils auront comme droit (-rwxr-xr-x).

- Si vous tapez umask 244
  - vous partez des droits maximum 777 et vous retranchez 244
    - on obtient donc 533
    - par défaut ils auront comme droit (-rwx-wx-wx).

Autres commande de gestion des utilisateurs et du système

Autres commandes de gestions des utilisateurs

- **who** (affiche la liste des utilisateurs connectés)
- **whoami** (indique le « login » de l'utilisateur)
- **id** (identité de l'utilisateur actif, UID, GID)
- **finger** (affiche des informations sur les utilisateurs)
- **chsh** : changer le shell ;
- **id** : affiche l'identifiant de l'utilisateur ;
- **last** : afficher la liste des connexions utilisateurs;
- **chfn** (change les informations de finger dans « /etc/passwd »: nom + mot de passe (ou « x » si « /etc/shadow ») + UID + GID + commentaire + répertoire de base + shell)
- **Passwd** (changer de mot de passe)
- **uptime** (temps de connexion de l'utilisateur)
- **su** (switch user ou substitute user, ouvrir un shell avec un autre compte utilisateur)
- Sudo : obtenir des droits étendus du root

Administration des utilisateurs

Création
Suppression
Modification
Consultation

Il faut évidemment être connecté en root

Administration des utilisateurs

Principales commandes

Gestion des comptes utilisateurs :

- **adduser : ajouter un utilisateur ;**
- **usermod : modifier un utilisateur.;**
- **deluser : supprimer un utilisateur.**
- **Users : liste des utilisateurs**

Gestion des groupes :

- **addgroup : ajouter un groupe ;**
- **groupmod : modifier la définition d'un groupe;**
- **delgroup : supprimer un groupe ;**
- **groups : afficher le groupe de l 'utilisateur.**

Administration des utilisateurs

$ adduser login_utilisateur

Notons :

- **la création du répertoire personnel dans /home/login_utilisateur**
- **la gestion et l 'authentification des utilisateurs est inscrit dans un seul fichier /etc/passwd**
- **la gestion des groupes est assurée par /etc/group**
- **les mots de passe cryptés sont placé s dans /etc/shadow**

Administration des utilisateurs

Les options de adduser :

-c : saisie du commentaire.
-d : le répertoire personnel
-g : le groupe de base
-G : groupes supplémentaires pour utilisation avec la commande newgrp
-m : recopie du /etc/skel dans le répertoire personnel.
-s : le shell
-u : choix d'un UID spécifique > 99
-o : l'UID n'est pas unique
-r : UID privilégié système <= 99
-e : date d'expiration du login
-f : délai avant verrouillage si mot de passe non changé
-p : mot de passe déjà crypté avec crypt

Exemples :

1. useradd -m user
2. useradd -c "login de test" -d /home/logtest -s /bin/bash -g users -u 123

Administration des utilisateurs

La commande de modification d'un compte utilisateur s'applique en fonction des options désirées.

Exemples :

$ usermod -G nom_groupe login_utilisateur

ajouter utilisateur dans un groupe (existant bien sûr !).

$ usermod -L login_utilisateur

bloquer le compte de l'utilisateur

$ usermod -e MM/JJ/AA login_utilisateur

changer la date d'expiration du compte

Administration des utilisateurs

La commande usermod modifie les paramètres d'un compte et permet aussi de le verrouiller. Les options sont:

[-c commentaire] [-d rép_perso [-m]]
[-e date_expiration] [-f inactivité]
[-g groupe_initial] [-G groupe[,...]]
[-l nom_connexion] [-p mot_de_passe]
[-s shell] [-u uid [-o]] [-L|-U] login

Remarque :
usermod ne permet pas de modifier le nom d'un utilisateur qui est actuellement connecté. (s'assurer avant d'exécuter cette commande que l'utilisateur nommé n'est pas en train d'exécuter un quelconque programme).
il faut modifier le nom du propriétaire de tous les fichiers crontab ainsi le nom du propriétaire de n'importe quel travail at manuellement

Pour changer les informations personnels de l'utilistaeur il faut utiliser la commande :

```
chfn [-f full_name] [-r room_no] [-w work_ph] [-h home_ph] [-o other] [user]
```

Administration des utilisateurs

Supprimer le compte d'un utilisateur comporte l'obligation que celui-ci ne soit pas connecté :

```
userdel [ - r] login_utilisateur
```

- **L'option -r supprime aussi le répertoire personnel (non effacé par dé faut).**

Administration des utilisateurs

```
$ passwd  login_utilisateur
```

À noter :

- **l'option -d pour supprimer le mot de passe**
- **l'option –l pour le verrouiller**
- **l'option -u pour le déverrouiller**

Administration des utilisateurs

passwd permet de changer le mot de passe, et dispose de plusieurs options.

passwd [-k] [-l] [-u [-f]] [-d] [-S] [*nom_utilisateur*]

-f : Force le changement de mot de passe à la prochaine connexion

-d : Désactiver un mot de passe en supprimant la saisie du mot de passe pour un compte donné. Disponible uniquement pour root.

-k : Indique que la mise à jour ne devrait être effectuée que pour les mots de passe expirés; un utilisateur conserve son mot de passe non expirés.

-l : Verrouille le compte spécifié, elle n'est disponible que pour root. Le verrouillage est effectué en rendant le mot de passe crypté invalide (en le préfixant par un !).

-u : L'inverse de l'option précédente - il déverrouillera le mot de passe du compte. disponible uniquement pour root.

Administration des utilisateurs

Changement régulié du mot de passe

Cette fonctionnalité vous permet de gérer le nombre maximum de jours pendant lesquels un utilisateur peut utiliser le même mot de passe.

La commande **passwd** a quatre options pour gérer les expirations :

-n min : (rarement utilisée) est appliquée pour définir le nombre minimum de jours requis pour qu'un utilisateur change son mot de passe.

-x max: Définit le nombre maximal de jours pendant lesquels l'utilisateur peut utiliser son mot de passe sans le changer.

-c garde: pour envoyer un avertissement à l'utilisateur lorsque son mot de passe est sur le point d'expirer. L'argument de cette option spécifie combien de jours l'utilisateur sera avant que son mot de passe expire.

–i inact: Cette option permet la gestion de l'expiration du mot de

Administration des groupes

Gestion des groupes

- **Pour créer un nouveau groupe :**

$ addgroup nom_du_groupe

- **Pour lister tous les groupes d'un utilisateur :**

$ groups nom_du_groupe

- **Pour supprimer un groupe :**

$ delgroup nom_du_groupe

- **Pour ajouter un utilisateur à un groupe :**

$ usermod user nom_du_groupe
(ou groupmod avec l'option -n).

Programmation shell

Le Shell- Interpréteur de commandes

- Nous allons étudier dans cette partie le script bash
  - Mécanismes de base de l'interprétation des commandes par le shell.
  - Programmation shell.

Présentation

Le shell n'est pas qu'un simple interpréteur de commandes, mais dispose d'un véritable langage de programmation avec notamment une gestion des variables, des tests et des boucles, des opérations sur variables, des fonctions...

Shell utilisé

La commabde ps: Nom du SHELL

Ex : $ **ps**

PID TTY TIME CMD

6908 pts/4 00:00:00 bash => l'interpréteur utilisé est *bash*

6918 pts/4 00:00:00 ps

$

La commande bash –version : Donne la version du SHELL

Ex : $ **bash --version**

GNU bash, version **4.2**.24(1)-release (i686-pc-linux-gnu) Copyright (C) 2011 Free Software Foundation, Inc.

Licence GPLv3+ : GNU GPL version 3 ou ultérieure <http://gnu.org/licenses/gpl.html>

...

$

Commandes internes et externes

- Le shell distingue deux sortes de commandes :
  - ***les commandes internes***
    le code est implanté au sein de l'interpréteur de commande (**cd** , **echo** , **for** , **pwd,…)**
  - ***les commandes externes.***
    le code se trouve dans un fichier ordinaire (**ls**, **mkdir**, **cat**, **sleep,…)**
- ***type –t*** Cmde:

  builtin : Cde Interne
  file : Cde externe

Ex : $ **type -t sleep**

file ⇒ *sleep* est une commande externe

$ **type -t echo**

builtin ⇒ *echo* est une commande interne du shell

$

Modes d'exécution d'une commande

- ***l'exécution séquentielle (Par défaut)***

  le shell lit la commande entrée par l'utilisateur, l'analyse, la prétraite et si elle est syntaxiquement correcte, l'exécute.

- ***Exécution en arrière-plan***

  L'exécution en arrière-plan permet à un utilisateur de lancer une commande et de récupérer immédiatement la main pour lancer « en parallèle » la commande suivante (parallélisme logique). On utilise le caractère **&** pour lancer une commande en arrière-plan

  $ sleep 5 &

Commentaires

- Un commentaire débute avec le caractère **#** et se termine avec la fin de la ligne. Un commentaire est ignoré par le shell.

Ex :

```
$ echo  bonjour   # l'ami
bonjour
$ echo  coucou   #l'   ami
coucou
$ #  echo  coucou
$
```

Fichiers shell

Lorsqu'un traitement nécessite l'exécution de plusieurs commandes, il est préférable de les sauvegarder dans un fichier plutôt que de les retaper au clavier chaque fois que le traitement doit être lancé.

Ce type de fichier est appelé *fichier de commandes ou fichier **shell*** ou encore ***script shell***.

Structure et exécution d'un script

- Par convention les **shell scripts** se terminent généralement (pas obligatoirement) par « **.sh** » pour le ***Bourne Shell*** et le ***Bourne Again Shell***, par « **.ksh** » pour le ***Korn Shell*** et par « **.csh** » pour le ***C Shell***.
- Pour lancer l'exécution d'un ***fichier shell***, on peut utiliser la commande : **bash**
  - $ **bash** *monscript*
- Il est plus simple de lancer l'exécution d'un programme shell en tapant directement son nom, comme on le ferait pour une commande unix ou une commande interne.
- Pour que cela soit réalisable, deux conditions doivent être remplies :
  - l'utilisateur doit posséder les permissions **r** (*lecture*) et **x** (*exécution*) sur le fichier shell :
    - ***$ chmod u+x monscript***
    - Pour l'exécuter : ***$ ./monscript***
  - Pour éviter le **./** le répertoire dans lequel se trouve le fichier shell doit être présent dans la liste des chemins contenue dans ***PATH***. :
    - ***$ PATH=$PATH:.***
    - ***$ monscript***

Structure et exécution d'un script

- Une ligne de commentaire commence toujours par le caractère « **#** ».
  Un commentaire peut être placé en fin d'une ligne comportant déjà des commandes.

  # La ligne suivante effectue un ***ls***
  ls *# La ligne en question*

- La première ligne à une importance particulière car elle permet de préciser quel **shell** va exécuter le **script**

  #!/bin/sh
  #!/bin/ksh

Dans le premier cas c'est un script *Bourne*, dans l'autre un script *Korn*

Dans notre cas on va utiliser le ***Bourne Again Shell***

#!/bin/bash

Exercice 1 :

1.) A l'aide d'un éditeur de texte, créer un fichier ***premier*** *contenant les lignes* suivantes :

```
#!/bin/bash
# premier
echo -n "La date du jour est: "
date
```

La notation #! en première ligne d'un fichier interprété précise au shell courant quel interpréteur doit être utilisé pour exécuter le programme (dans cet exemple, il s'agit de /bin/bash).

La deuxième ligne est un commentaire.

2.) Vérifier le contenu de ce fichier.

3.) Pour lancer l'exécution d'un fichier shell ***fich****, on peut utiliser la commande :* **bash** ***fich*** ou ***./fich*** ou directement ***fich***

Ex : $ **bash premier**

la date du jour est : jeudi 27 février 2014, 17:34:25 (UTC+0100)

$

Structure et exécution d'un script

Exercice 1

Écrire un programme shell ***monscript1*** qui affiche le nom de connexion de l'utilisateur et le chemin absolu de son répertoire courant de la manière suivante :

Ex : $ ***monscript1***

mon nom de connexion est : stagiaire
mon repertoire courant est : /home/stagiaire
$

Les types de parameters

Le shell distingue trois types de paramètres :

- les ***variables***, identifiées par un *nom*
  Ex : *a* , **PATH**
- les ***paramètres de position***, identifiés par un *numéro*
  Ex : **0** , **1** , **12**
- les ***paramètres spéciaux***, identifiés par un *caractère spécial*
  Ex : **#** , **?** , $

1. Variables

Une variable est identifiée par un ***nom,***
Une suite de lettres, de chiffres ou de caractères
espace souligné ne ***commençant pas par un chiffre***.
Les lettres majuscules et minuscules sont différenciées.

1. VARIABLES

Les variables peuvent être classées en trois groupes :

A- les variables **utilisateur** (ex : *a, valeur)*

B- les variables **prédéfinies du shell** (ex : **PS1, PATH, REPLY, IFS, HOME)**

C- les variables **prédéfinies de commandes unix** (ex : **TERM).**

En général, les noms des **variables utilisateur** sont en lettres **minuscules** tandis que les noms des **variables prédéfinies** (du shell ou de commandes unix) sont en **majuscules**.

1. VARIABLES

- Les principales commandes de manipulation des variables sont :
  - Echo
  - Print
  - Export
  - read
  - Set, unset
  - Shift

Les variables

- **Variables simples:**
  - Nous appellerons variable simple toute variable définie par l'utilisateur pour ses propres besoins.

Les variables simples

- ***Désignation de la variable***
  - Pour désigner une variable, il suffit d'utiliser la syntaxe suivante:

    $variable
  - Si la variable doit être suivie d'une valeur autorisée (exemple de nom de variable auquel on ajoute séquentiellement un nombre), la notation est alors la suivante:

    ${variable}nombre

- ***Affichage d'une variable***
  - La commande:

    echo [-n] [texte] $variable
  - permet d'afficher la valeur de la variable.
  - On peut aussi l'intégrer dans du texte.
  - L'option –n permet d'éviter le passage à la ligne après l'affichage.

Types d'affectation des Variables

1. Affectation directe

Syntaxe : *nom*=***[valeur]*** ***[nom=[valeur] ...]***

le symbole = et la valeur à affecter forment une **seule chaîne de caract**ères.

Plusieurs affectations peuvent être présentes dans la même ligne de commande

Ex : $ x=coucou y=bonjour

=> la variable *x contient la chaîne de caractères coucou*

=> la variable *y contient la chaîne bonjour*

Ex : $ **echo x est $x**

x est coucou

$

le *nom d'un paramètre* ***« x »*** *et la valeur de ce paramètre* ***« $x »*** *ne se* désignent pas de la même manière.

Ex : $ **x=xy** ***=> x : contenant, $x : contenu***
$ **echo $x**
coucoubonjour
$

Lorsque l'on souhaite effectuer une concaténation, il est possible d'utiliser l'opérateur **+=** de **bash.**

Ex :

```
$ x=coucou y=bonjour
$ x+=$y
$ echo $x
coucoubonjour
$
```

2. Affectation par lecture : saisie au clavier

Syntaxe : **read [*var1 ...]***

Ex : $ **read a b**
bonjour Monsieur => chaînes saisies par l'utilisateur et enregistrées respectivement dans les variables *a et b*

$ **echo $b**
Monsieur
$

Lorsque la commande interne **read est utilisée sans argument, la ligne lue est enregistrée dans la** variable prédéfinie du shell **REPLY.**

Ex :

```
$ read
bonjour tout le monde
$
$ echo $REPLY
bonjour tout le monde
$
```

L’ ’option « **–p »** de read affiche une chaîne d’appel avant d’effectuer la lecture

read –p *chaîne_d_appel [var …]*

Ex :

```
$ read -p "Entrez votre prenom : " prenom
Entrez votre prenom : Eric
$
$ echo $prenom
Eric
$
```

Remarque:

R1. s'il y a **moins** de variables que de mots dans la ligne lue, le shell affecte le 1er mot à la première variable, le 2ème mot à la 2ème variable, etc., la dernière variable reçoit tous les mots restants.

Ex : $ **read a b c**
un bon jour coucou
$
$ **echo $a**
un
$ **echo $c**
jour coucou
$

R2. s'il y a Plus de variables que de mots dans la ligne lue, chaque variable reçoit un mot et après épuisement de ces derniers, les variables excédentaires sont vides

Ex : $ **read a b**
un
$
$ **echo $a**
un
$
$ **echo $b**
=> valeur *null*
$

La commande **read** propose plusieurs options intéressantes.

-p : afficher un message de prompt

-n : limiter le nombre de caractères

-s : ne pas afficher le texte saisi

Exercice 1 : Ecrire un programme shell ***deuxfois*** *qui affiche le message "Entrez un mot : ", lit le* mot saisi par l'utilisateur puis affiche ce mot deux fois sur la même ligne.

Ex : $ deuxfois
Entrez un mot : toto
toto toto
$

Solution deuxfois

```
#!/bin/bash
read -p "Entrez le mot:" a
echo $a $a
```

3. Variable en « lecture seule » ou Constante

declare –r ***nom=valeur [nom=valeur …]***

Ex : $ **declare -r mess=bonjour**

$

On dit que la variable ***mess*** *possède* ***l'attribut r.***

Une tentative de **modification** de la valeur d'une constante provoque une **erreur**.

Ex : $ **mess=salut**
bash: mess : variable en lecture seule
$

Pour connaître la liste des constantes définies : **declare –r**

Ex : $ **declare -r**

```
declare -r
BASHOPTS="checkwinsize:cmdhist:expand_aliases:extglob:extquote:force_fign
                                comments:progcomp:promptvars:sourcepath"
ore:histappend:interactive_comments:progcomp:promptvars:sourcepath
declare -ir BASHPID
declare -r BASH_COMPLETION="/etc/bash_completion"
declare -r BASH_COMPLETION_COMPAT_DIR="/etc/bash_completion.d"
declare -r BASH_COMPLETION_DIR="/etc/bash_completion.d"
declare -ar BASH_VERSINFO='([0]="4" [1]="2" [2]="24" [3]="1"
[4]="release" [5]="i686-pc-linux-gnu")'
declare -ir EUID="1000"
declare -ir PPID="1864"
declare -r
SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:interactivecomments:
monitor"
declare -ir UID="1000"
declare -r mess="bonjour"
$
```

Quotes, apostrophe, guillemets et apostrophe inversée

- Les simples quotes " délimitent une chaîne de caractères. Même si cette chaîne contient des commandes ou des variables shell, celles-ci ne seront pas interprétées

  $ variable="secret"

  $ echo 'Mon mot de passe est $variable.‘

  Mon mot de passe est $variable.

- Les doubles quotes " " délimitent une chaîne de caractères, mais les noms de variable sont interprétés par le shell

  $ variable="secret"

  $ echo "Mon mot de passe est $variable."

  Mon mot de passe est secret.

- Bash considère que les Anti-quotes `` délimitent une commande à exécuter. Les noms de variable et les commandes sont donc interprétés

  $ echo `variable="connu"; echo "Mon mot de passe est $variable."`

  Mon mot de passe est connu.

  echo `ls` **cette commande affiche le contenu du répertoire courant à l'écran**

Les attributs de variables : typeset

- La commande positionne, réinitialise ou affecte les variables selon différentes options :
  - **-L cadrage à gauche**
    var=MAVARIABLE; echo $var
    MAVARIABLE
    typeset -L4 var ; echo $var
    MAVA
  - **-R cadrage à droite**
    var= MAVARIABLE ; echo $var
    MAVARIABLE
    typeset –R5 var ; echo $var
    IABLE
  - **-Z cadrage à droite et remplissage par des 0 à gauche**
    typeset –Z5 var=$(ls | wc -l)
    00005

Les attributs de variables

- **conversion en majuscule -u ou en minuscule –l**

  var=MAVARIABLE ; print $var # Affiche MAVARIABLE
  typeset -u var ; print $var # Affiche MAVARIABLE
  typeset -l var ; print $var # Affiche mavariable

- **protection d'une variable en lecture uniquement -r**

  typeset -r var
  var=bonjour # Affiche message : bash: var is read only

- **+r annuler l'option (-r)**

  typeset -r var
  var=bonjour # Affiche message : bash: var is read only
  typeset +r var
  var=bonjour; echo $var # Affiche bonjour

- *De façon générale + annule l'option ou l'attribut affecté à une variable.*

EXPORTATION DES VARIABLES

- D'ordinaire, une variable n'est utilisée que dans le shell-script où elle reçoit son affectation, si on souhaite l'utiliser pour les programmes appelés ultérieurement, on doit l'exporter.

```
VARIABLE=valeur
export VARIABLE
```

- Permet de rendre global (visible pour des shells secondaires, et non les autre shells) des *variables* qui, par défaut, sont locales au shell courant.

```
ma_variable=toto
export ma_variable
bash
echo $ma_variable          #affiche toto
```


Declare et export

- Pour rendre une variable disponible pour une exportation en dehors de l'environnement du script lui-même on utilise:

```
declare -x var3
```

```
declare -x var3=373
```

- La variable var3 peut être appelée par un autre script dans le même shell
- Mais reste inconnue pour les autres shell (nouvelle console)

REMARQUES :

Les variables Bash ne sont pas typées : À l'inverse de nombreux langages de programmation. Essentiellement, les variables bash sont des chaînes de caractères mais, suivant le contexte, Bash autorise des opérations entières et des comparaisons sur ces variables, le facteur décisif étant la seule présence de chiffres dans la variable.

declare –i permet de déclarer une variable de type entier par exemple

Les commandes intégrées declare ou typeset, qui sont des synonymes exacts, permettent de modifier les propriétés des variables. Ceci est une forme très faible du typage disponible dans certains langages de programmation. La commande declare est spécifique à la version 2 ou ultérieure de Bash. La commande typeset fonctionne également dans les scripts ksh.

VARIABLES D'ENVIRONNEMENT

Les variables accessibles à un interpréteur (ou plus généralement à un processus) peuvent se décomposer en deux classes :

- ***les variables locales*** *(comme les variables de l'exemple précédent) : elles disparaissent* lorsque l'interpréteur (ou le processus) se termine

- ***les variables d'environnement*** *: celles-ci ont la particularité d'être automatiquement* transmises à sa descendance, c.-à-d. copiées, à tous les shells fils (ou processus fils) qu'il créera dans le futur.

Des exemples de *variables d'environnement usuelles sont* ***PATH et HOME.***

VARIABLES D'ENVIRONNEMENT

Pour créer une *variable d'environnement ou faire devenir variable d'environnement une variable* déjà existante, on utilise la commande interne **export.**

Ex : $ **a=coucou**
$ **b=bonjour => définition de la variable locale *b***
$ **export b => la variable *b devient une variable d d'environnement***
$ **export c=bonsoir => définition de la *variable d'environnement c***
$ **bash => création d'un shell fils**
$ **echo $a**
$ => la variable *a est inconnue*
$ **echo $b**
bonjour
$ **echo $c**
bonsoir => les variables *b et c sont connues du shell fils*
$ **exit => fin du shell fils**
exit
$

VARIABLES D'ENVIRONNEMENT

La transmission des *variables d'environnement est* ***unidirectionnelle*** *: du processus* ***père vers le(s)*** **processus fils**.

Si un processus fils modifie la valeur d'une *variable* d'environnement*, seule sa* copie locale en sera affectée.

Ex : $ export a=coucou
$ bash
$ echo $a
coucou
$ a=bonsoir => modification de la variable d'environnement *a dans le shell fils*
$ echo $a
bonsoir
$
$ exit
exit
$ echo $a
coucou => la valeur n'a pas été modifiée dans le shell père
$

VARIABLES D'ENVIRONNEMENT

- ***Visualisation des variables disponibles***
  - La commande ***printenv :*** affiche les valeurs des **variables de l'environnement** du processus en cours
- ***Supprimer une variable***
  - La commande ***unset variable :*** supprime la variable de la liste des variables disponibles.

Variables du shell - personnalisation de l'environnement

- ***Les principales variables***
  - Le shell permet le stockage de variables d'environnement (en majuscules)
  - Définies pour l'ensemble du système
    - Exemples : HOME, PATH, SHELL, USER
  - On accède à leur valeur en les faisant précéder d'un «$»
  - Pour visualiser toutes les variables d'environnement: «env»

Variables du shell - personnalisation de l'environnement

- CDPATH: liste des répertoires recherchés quand cd est exécuté avec un nom relatif comme argument,
- DISPLAY: l'écran sur lequel on travaille (par défaut :0.0), ENV: nom d'un fichier exécuté à chaque appel de bash,
- GID: le numéro du groupe actuel de l'utilisateur,
- HOME: répertoire de login,
- HOST: le nom de la machine sur laquelle on se trouve,
- LOGNAME: le nom de l'utilisateur,
- MANPATH: liste des répertoires où se trouvent les pages de manuel,
- PATH: liste des répertoire de recherche des commandes, le séparateur est «:»,
- PWD: le répertoire courant,
- SHELL: le shell par défaut,
- TERM: nom du type de terminal utilisé,

Substitutions des variables

- Effectuer un certain nombre de substitutions lors de l'affichage:
  - $paramètre → affiche la valeur courante de paramètre
  - ${paramètre} → affiche la valeur courante de paramètre et autorise la concaténation
  - ${paramètre:-valeur} → affiche la valeur courante de paramètre si elle est non nulle, sinon affiche valeur
  - ${paramètre:=valeur} → affiche la valeur courante de paramètre si elle est non nulle, sinon affiche valeur et affecte à paramètre valeur,
  - ${paramètre:?valeur} → affiche la valeur courante de paramètre si elle est non nulle, sinon on écrit valeur sur la sortie standard et on sort.

Paramètres de position

Un *paramètre de position est référencé par un ou plusieurs chiffres : 8 , 0 , 23*

L'assignation de valeurs à des paramètres de position s'effectue :

- soit à laide de la commande interne **set**
- soit lors de **l'appel** d'un fichier **shell** ou d'une **fonction** shell.

on ne peut utiliser ni le symbole **=,** ni la commande interne **read** pour affecter directement une valeur à un paramètre de position.

Ex : $ **23=bonjour**
23=bonjour : commande introuvable
$
$ **read 4**
aa
bash: read: « 4 » : identifiant non valable
$

Commande interne set :

La commande interne **set affecte une valeur à un ou plusieurs paramètres de position** en numérotant ses arguments suivant leur position. La numérotation commence à **1.**

Syntaxe : **set *arg1 ...***

Ex : $ **set alpha beta gamma**

⇒*alpha est la valeur du paramètre de position 1,*
⇒*beta la* valeur du 2ème paramètre de position
⇒*gamma* la valeur du paramètre de position 3

Pour obtenir la valeur d'un paramètre de position, il suffit de placer le caractère **$** devant son numéro. Ainsi, $1 permet d'obtenir la valeur du premier paramètre de position, $2 la valeur du deuxième et ainsi de suite.

Ex : $ **set ab be ga => numérotation des mots *ab, be et ga***

```
$
$ echo $3 $2
ga be
$
$ echo $4
$
```

➢Tous les paramètres de position sont ***réinitialisés*** dès que la commande interne ***set est utilisée avec au moins un argument***.

Ex : $ set coucou
$ echo $1
coucou
$ echo $2
=> les valeurs *be et ga précédentes ont disparues*
$

➢La commande interne **set --** rend indéfinie la valeur des paramètres de position préalablement initialisés.

Ex : $ set alpha beta
$ echo $1
alpha
$ set --
$ echo $1
=> les valeurs *alpha et beta sont perdues*
$

Remarques

- **Variable prédéfinie IFS et commande interne set** :

  La variable prédéfinie IFS du shell contient les caractères séparateur de mots. (Par défaut, ce sont les caractères espace, tabulation et interligne).

  L'initialisation de IFS est effectuée par bash. (Ex : $ set Jean:12:Rodez (3 arguments si IFS=: et un seul sinon))

- Utilisée sans argument, **set** a un comportement différent : elle affiche, entre autres, la liste des noms et valeurs des variables définies. (Ex : **$ set | more**)

- Si la valeur du premier **argument** de set commence par un caractère - ou +, une erreur se produit. En effet, les options de cette commande interne commencent par un de ces deux caractères.

  Pour éviter que ce type d'erreur ne se produise, on utilise la syntaxe : **set –-arg…**

Ex : **$ a=+qui**

$ set $a

bash: set: +q : option non valable

set : utilisation : set [-abefhkmnptuvxBCHP]

[-o option-name] [--] [arg ...]

$ set -- $a

$

$ echo $1

+qui

Paramètres spéciaux

- Un *paramètre spécial est référencé par un caractère spécial. L'affectation d'une valeur à un* paramètre spécial est effectuée par le shell (implicitement).

- Pour obtenir la valeur d'un paramètre spécial, il suffit de placer le caractère **$ devant le caractère spécial qui le représente.**

PARAMÈTRES SPÉCIAUX

- $# donne le nombre d'arguments passés au script
- ${#var} *longueur de la variable "var"*
- $* liste complète des valeurs des paramètres
  - représente seul mot constitué de la valeur de tous les paramètres positionnels séparées par le premier caractère de la variable spéciale IFS.
  - Ceci signifie que $* est équivalent à "$1c$2c...", dans laquelle c est le premier caractère de la valeur de la variable IFS.
  - Si IFS est nulle ou inexistante, les paramètres sont séparés par des espaces.
- $0 le nom du script

PARAMÈTRES SPÉCIAUX

- $n, ${n} valeur du nième argument du script,
- $@ équivalent à $*, mais $@ signifie: "$1" "$2" … "${n}"
  - chaque paramètre se transforme en un mot distinct
- $$ le numéro de processus du script,

Exercice 3 : Ecrire un programme shell ***nbmots*** *qui demande à l'utilisateur de saisir une suite* quelconque non vide de mots puis ***affiche le nombre de mots saisis***.

Ex : $ nbmots
Entrez une suite de mots : un deux trois quatre cinq
5 => 5 mots ont été saisis
$

Solution 1

```
#!/bin/bash
read -p "Entrez une suite de mots:" a
set $a
echo $#
```

Solution 2

```
#!/bin/bash
echo "Entrez une suite de mots"
read a
set $a
echo $#
```

Commande interne shift

La commande interne shift décale la numérotation des paramètres de position ayant une valeur.
Syntaxe : **shift [n]**

$ **set a b c d e** => *1 2 3 4 5*
$ **echo $1 $2 $#**
a b 5
$ **shift 2**
=> *a b c d e les mots a et b sont devenus inaccessibles*
=> *1 2 3*
$ **echo $1 $3**
c e
$ **echo $#**
3
$

Remarques :
La commande shift **sans argument** est équivalente à **shift 1**
shift ne modifie pas la valeur du paramètre de position 0 qui possède une signification particulière.

Commande interne set : cas particulier

Lorsque la commande interne **set** est utilisée à l'intérieur d'un programme shell, la syntaxe **$1** possède deux significations différentes : **$1** comme *premier argument du programme shell,*
*et **$1*** comme *premier paramètre de position initialisé par **set** au sein du fichier shell.*

Exemple : programme shell ***ecrase_arg***

```
---------------
# !/bin/bash
echo '$1' est $1 => la chaîne $1 est remplacée par le premier argument
set hello              => set écrase la valeur précédente de $1
echo '$1' est $1
---------------
```

Ex : $ **ecrase_arg** bonjour coucou salut
$1 est bonjour
$1 est hello
$

Exercice

Ecrire un script SHELL « **copie** » qui réalise la copie d'un fichier source vers un fichier destination

Solution copie

Soit le programme shell *copie :*

```
#!/bin/bash
# copie
echo "Nom du programme : $0"
echo "Nb d'arguments : $#"
echo "Source : $1"
echo "Destination : $2"
cp $1 $2
```

Paramètres spéciaux * et @

Les *paramètres spéciaux* ***@ et * contiennent tous deux la liste des valeurs des paramètres de*** position initialisés.

Ex : $ **set un deux trois quatre**
$
$ **echo $***
un deux trois quatre
$
$ **echo $@**
un deux trois quatre
$

"$*" est remplacée par "$1 $2 ... "
"$@" est remplacée par "$1" "$2" ...

Ex : $ set bonjour "deux coucou" salut

=> trois paramètres de position sont initialisés

$

$ set "$*" => est équivalent à : *set “bonjour deux coucou salut”*

$

$ echo $#

1

$

$ echo $1

bonjour deux coucou salut

$

"$@" produit autant de chaînes que de paramètres de positions initialisés.

"$*" traite lensemble des paramètres de position initialisés comme une unique chaîne de caractères.

Inversement, ,

Ex : $ set bonjour "deux coucou" salut => trois paramètres de position initialisés

$

$

$ set "$@" => est équivalent à : *set “bonjour” “deux coucou” “salut”*

$

$ echo $#

3

$ echo $2

deux coucou => l’intégrité de la chaîne a été préservée

$

Substitution de commandes

Syntaxe : **$(*cmd*)**

Une commande **cmd** entourée par une paire de parenthèses ( ) précédées d'un caractère $ est exécutée par le shell puis la chaîne $(cmd) est remplacée par les résultats de la commande cmd écrits sur la sortie standard, c'est à dire l'écran.

Ces résultats peuvent alors être affectés à une variable ou bien servir à initialiser des paramètres de position.

Ex : $ pwd
/home/ubuntu => résultat écrit par pwd sur sa sortie standard
$ repert=$(pwd) => la chaîne /home/ubuntu remplace la chaîne $(pwd)
$
$ echo mon repertoire est $repert
mon repertoire est /home/ubuntu
$

Mettre une commande dans une var

```
rep=`pwd`
echo $rep
var=$(echo `pwd`)
Var=$(echo `ls –l|wc -l`)
Var =$(` echo $mavariable | grep [a-z] | wc -l` )
```


Exercice : En utilisant la substitution de commande, écrire un fichier shell ***mach*** *affichant :*
"Ma machine courante est *nomdelamachine*"

Ex : $ mach
Ma machine courante est ubuntu
$

Solution

```
#!/bin/bash
mamachine=$(hostname)
Echo ’’Ma machine courante est $mamachine’’
```

Pour l'utiliser avec la commande set il faut utiliser le double « - »

Set - - $(cmd)

$ **set $(ls -l .bashrc)**
bash: set: -w : option non valable
set : utilisation : set [-abefhkmnptuvxBCHP] [-o option-name] [--] [arg...]
$ **ls -l .bashrc**
-rw-r--r-- 1 sanchis sanchis 3486 mai 18 2013 .bashrc
$

$ **set -- $(ls -l .bashrc)**
$
$ **echo $1**
-rw-r--r--
$

Exercice :

Ecrire un programme shell ***taille*** *qui prend un nom de fichier en argument et* affiche sa taille. On ne considère aucun cas d'erreur.

Ex : $ ls -l .bashrc

-rw-r--r-- 1 ubuntu ubuntu 3486 mai 18 2013 .bashrc

$

$ taille .bashrc

3486

GROUPEMENT DE COMMANDES

Le shell fournit deux mécanismes pour grouper l'exécution d'un ensemble de commandes sans pour cela affecter un nom à ce groupement :

- l'insertion de la suite de commandes entre une paire d'accolades { suite_cmds ; }

```
Ex : $ pwd /home/sanchis => répertoire initial
$
$ { cd /bin ; pwd ; } => changement du répertoire courant /bin
$
$ pwd /bin => répertoire final (le changement a été préservé !)
$
```

- l'insertion de cette suite de commandes entre une paire de parenthèses (suite_cmds ;)

```
Ex : $ ( pwd ; date ; echo FIN ) > fin
$
$ cat fin
 /home/sanchis
 lundi 31 mars 2014, 10:55:31 (UTC+0200)
 FIN
$
```

Exercice : A l'aide de la commande unix date, écrire un programme shell *jour qui affiche le* jour courant du mois.

Ex : $ date
lundi 24 mars 2014, 09:08:02 (UTC+0100)
$
$ jour
24
$

Set - - $(date)
Echo $2

Exercice :

a) Ecrire un programme shell *heure1 qui affiche l'heure sous la forme : heures:minutes:secondes*

Ex : $ heure1
09:11:40
$

b) Ecrire un programme shell *heure qui n'affiche que les heures et minutes.* On pourra utiliser la variable prédéfinie **IFS** du shell

Ex : $ heure
09:11
$

Exercice :

Ecrire un programme shell ***uid*** *qui affiche l'uid de l'utilisateur. On utilisera la* commande unix id, la commande interne set et la variable prédéfinie IFS.

```
$id
uid=1016(toto) gid=100(users) groups=100(users)
```

Suppression des ambiguïtés

Pour éviter les ambiguïtés dans l'interprétation des références de paramètres, on utilise la syntaxe ${*paramètre*}.

Ex : $ x=bon
$ x1=jour
$ echo $x1 => valeur de la variable ***x1***
jour
$ echo ${x}1 => pour concaténer la valeur de la variable ***x*** *à la chaîne* ***"1"***
bon1
$

Ex : $ set un deux trois quatre cinq six sept huit neuf dix onze douze
$ echo **$11**
un1
$
$ echo **${11}** => pour obtenir la valeur du onzième paramètre de position
onze
$

Paramètres non définis

Trois cas peuvent se présenter lorsque le shell doit évaluer un paramètre :
- le paramètre n'est pas défini,
- le paramètre est défini mais ne possède aucune valeur (valeur *vide),*
- le paramètre possède une valeur non vide.

Lorsque l'option **nounset** de la commande interne **set** est positionnée à l'état ***on*** *(commande* **set -o nounset**), bash affiche un message d'erreur quand il doit évaluer un paramètre non défini.

```
Ex : $ set -o nounset                  => option nounset à l'état on
$ echo $e                              => variable utilisateur e non définie,
bash: e : variable sans liaison        => message d'erreur !
$ set --                               => paramètres de position sont
réinitialisés
$ echo $1
bash: $1 : variable sans liaison
$ d=                                   => d : variable définie mais vide
$ echo $d
                                       => > valeur null, pas d d'erreur
$
```

Pour associer au paramètre une valeur ponctuelle lorsqu' il est non défini ou bien défini vide.
La syntaxe à utiliser est la suivante :

${paramètre:-chaîne}

Ex : $ set -o nounset
$ ut1=root => *ut1 définie et non vide*
$ **ut2**= **=>** *ut2 définie et vide*
$ echo $ut1
root
$ echo $ut2
$ echo $1
bash: $1 : variable sans liaison => paramètre de position *1 non défini*
$

Ex : $ echo **${ut1:-Remplace}**
root => *ut1 étant définie non vide, sa valeur est utilisée*
$ echo **${ut2:- Remplace}** => *ut2 est définie et vide, la valeur de remplacement est*
Remplace => utilisée
$
$ echo **${1:- Remplace}**
Remplace => le premier paramètre de position est non défini, la
$ => valeur de remplacement est utilisée

Cette association ne dure que le temps d'exécution de la commande.

Ex : $ **echo $1**
bash: $1 : variable sans liaison => le paramètre est redevenu indéfini
$

La commande « **set +o nounset** » positionne l'option nounset à l'état ***off*** *: le shell traite les* paramètres non définis comme des paramètres vides.

```
Ex : $ set +o nounset        => option nounset à l'état off
$
$ echo $e                    => variable e non définie,
                             => aucune erreur signalée
$
$ echo $2
                             => aucune erreur signalée
$
$ f=                         => f : variable définie vide
$ echo $f
$
```

Suppression de variables

Pour rendre indéfinies une ou plusieurs variables, on utilise la commande interne **unset.**

Syntaxe : **unset *var [var1 . . .]***

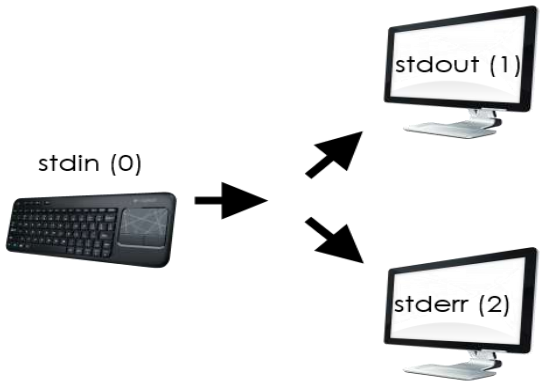
Ex : $ **a=coucou** **=> la variable *a est définie***
$ **echo $a**
coucou
$ **unset a** **=> la variable *a est supprimée***
$
$ **set -o nounset** **=> pour afficher le message d'erreur**
$ **echo $a**
bash: a : variable sans liaison
$

Une variable en « **lecture seule** » ne peut être supprimée par **unset.**

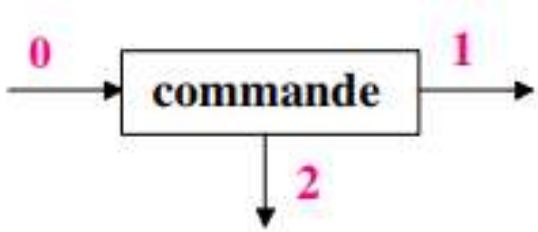
Ex : $ declare -r a=coucou => définition de la constante *a*
$ echo $a
coucou
$ unset a
bash: unset: a : « unset » impossible : variable est en lecture seule
$

Rappel sur les redirections d'entrée sortie

Un processus Unix possède par défaut trois voies d'interaction avec l'extérieur appelées entrées / sorties standard



Toute commande étant exécutée par un processus, possède trois entrées / sorties standard.



- **> fichier** Sortie standard vers un fichier (l'écrase si il existe)
- **>> fichier** sortie standard vers la fin d'un fichier
- **> fichier** entrée standard est un fichier
- **>> fichier** entrée standard en mode saisie jusqu'à mot de fin
- 2> **fichier** sortie d'erreur vers fichier
- **2>> fichier** sortie standard à la fin d'un fichier
- **2>&1** rediriger la sortie standard pour les messages d'erreur vers la sortie standard
- **cmd1 | cmd2** sortie de cmd1 est l'entrée de cmd2

Remarque :

Pour éviter que le contenu d'un fichier ne soit écrasé lors d'une redirection malheureuse, il est possible d'utiliser l'option **noclobber de la commande interne set.**

Ex : $ cat t
bonjour
$ **set -o noclobber** => activation de l'option *noclobber*
$ echo coucou > t
bash: t : impossible d'écraser le fichier existant
$ echo ciao >> t
$ => la concaténation reste possible
$ cat t
Bonjour
ciao

$ set +o noclobber => pour désactiver l'option
$

REDIRECTIONS ET SCRIPTS : SUBSTITUTION DE COMMANDE

Une substitution de commande associée à une redirection de l'entrée standard permet **d'affecter à une variable le contenu d'un fichier**.

En effet, la substitution de commande **$(cat *fichier)* *peut être*** avantageusement remplacée par la syntaxe **$(< *fichier)*. *Cette deuxième forme est plus rapide.***

Ex : $ **cat fic_noms**
pierre beteille
anne debazac
julie donet
$
$ **liste=$(<fic_noms)**
$
$ **echo $liste** **=> *liste est une variable et non un fichier !***
$ pierre beteille anne debazac julie donet => elle contient le contenu du fichier fic_noms

REDIRECTIONS ET SCRIPTS : SUBSTITUTION DE COMMANDE

Exercice :
Ecrire un programme shell *nblignes prenant un nom de fichier en argument et qui* affiche le nombre de lignes de ce fichier.

Ex : $ nblignes fich_noms
3
$

Soluton

```
$ nblignes=$( wc -l < fic_noms )
$
$ echo $nblignes
3                                   => nombre de lignes
$
```

REDIRECTIONS ET SCRIPTS

Redirections séparées des entrées / sorties standard

Les entrées / sorties peuvent être redirigées indépendamment les unes des autres.

Ex : $ **wc -l <fic_noms >fic_nblignes 2>err**
$ **cat fic_nblignes**
3
$ **cat err**
$

❖Il est possible de placer les redirections où l'on souhaite car le shell traite les redirections avant d'exécuter la commande.

Ex : $ **< fic_noms > fic_nblignes wc 2>err -l**
$
$ **cat fic_nblignes**
3
$
$ **cat err**
$

Texte joint

Il existe plusieurs syntaxes légèrement différentes pour passer un texte comme entrée à une commande.
La syntaxe présentée ci-dessous est la plus simple :

cmd **<<*mot***
texte
mot

Ex : $ **a=3.5 b=1.2**
$
$ **bc << Fin**
>$a + $b **=> la chaîne "> " indique que la commande >**
>Fin **=> n'est syntaxiquement pas terminée**
4.7
$

La commande unix bc est une calculatrice utilisable en ligne de commande.

Chaîne jointe

Syntaxe :

cmd ***<<< chaîne***

C'est une syntaxe plus compacte que le *texte joint :*
le contenu transmis à l'entrée standard de la commande *cmd se présente sous la forme d'une chaîne de caractères chaîne.*

Ex : $ **a=1.2 b=-5.3**
$
$ **bc <<< "$a + $b"**
-4.1
$

Tubes et chaînes jointes

Le shell crée un processus différent pour chaque commande d'un tube ***cmd1 | cmd2. Cela*** provoque l'effet suivant : si *cmd2 modifie la valeur d'une variable, cette modification disparaitra* avec la fin du processus qui exécute cette commande.

Ex : $ **a=bonjour**
$ **echo $a | read rep** **# (a)**
$ **echo $rep**

=>la variable *rep ne contient pas la chaîne bonjour ; elle n'est pas initialisée*

$

Pour éviter ça, on place les deux commandes à l'intérieur d'une paire de **parenthèses**

Ex : $ **echo $a | (read rep ; echo $rep)**
bonjour => l'affectation à *rep a bien été effectuée*
$ **echo $rep**

=> > variable *rep non initialisée*

$

Tubes et chaînes jointes

Cela provient du fait qu'il y a deux variables *rep différentes : une qui est créée et utilisée par le* processus qui exécute la commande *read rep et une qui est créée et utilisée par le processus qui* exécute la commande *echo $rep.* L'utilisation d'une *chaîne jointe permet de résoudre de manière élégante ce problème de* transmission de données entre processus.

Ex : $ **read rep <<< "$a"**
$
$ **echo $rep**
bonjour
$

Substitution de processus

La substitution de processus généralise la notion de tube.
La principale forme de substitution de processus est la suivante :

cmd <(suite_cmds)

Le shell crée un fichier temporaire, connecte la sortie de *suite_cmds à ce fichier temporaire puis* lance l'exécution de cette suite de commandes.

Ex : $ **ls -l**
total 8
-rw-rw-r-- 1 sanchis sanchis 140 mars 31 10:29 err
-rw-rw-r-- 1 sanchis sanchis 0 mars 31 10:26 f
-rw-r--r-- 1 sanchis sanchis 156 mars 31 10:30 trace
$ **wc -l <(ls -l)**
4 /dev/fd/63 => la commande **ls –l affiche 4 lignes**
$

Dans l'exemple ci-dessus, la sortie de la commande **ls –l a été enregistrée dans un fichier** temporaire */dev/fd/63.*
Puis, la commande wc –l /dev/fd/63 a été exécutée.

Le Shell – interprétation de commande test

- **La commande Test :**

  test condition && alors_instruction1 || sinon_instruction2

  ou

  [condition] && alors_instruction1 || sinon_instruction2

- est utilisée dans de nombreux cas par les structures de contrôle du shell.
- Cette commande renvoie 0 si la condition est vérifiée et une valeur différente de 0 sinon.
- Dans la deuxième syntaxe, il faut faire attention à mettre des espaces entre les crochets et la condition.
- **commande1 && commande2**: exécute commande1, si le code de retour est OK alors on exécute commande2,
- **commande1 || commande2** : exécute commande1, si le code de retour est mauvais alors on exécute commande2.

Tests de conditions

La commande **test** permet d'effectuer des tests de conditions. Le résultat est récupérable par la variable **$?** (***code retour***). Si ce résultat est **0** alors la **condition est réalisée**.

- ## Tests sur chaîne

  - **test -z "variable"** : ***zero***, retour **OK** si la variable est vide (ex test -z "$a")
  - **test -n "variable"** : ***non zero***, retour **OK** si la variable n'est pas vide (texte quelconque)
  - **test "variable" = chaîne** : **OK** si les deux chaînes sont identiques
  - **test "variable" != chaîne** : **OK** si les deux chaînes sont différentes

  Exemple :

```
$ a=
$ test -z "$a" ; echo $?
  0
$ test -n "$a" ; echo $?
  1
$ a=omar
$ test "$a" = omar ; echo $?
  0
```

Tests de conditions

- **Tests sur valeurs numériques**

Les chaînes à comparer sont converties en valeurs numériques. La syntaxe est :

test valeur1 option valeur2

et les options sont les suivantes :

Option	***Rôle***
-eq	***Equal*** : égal
-ne	***Not Equal*** : différent
-lt	***Less than*** : inférieur
-gt	***Greater than*** : supérieur
-le	***Less or equal*** : inférieur ou égal
-ge	***Greater or equal*** : supérieur ou égal

Exemple :

```
$ a=10
$ b=20
$ test "$a" -ne "$b" ; echo $?
   0
$ test "$a" -ge "$b" ; echo $?
   1
$ test "$a" -lt "$b" && echo "$a est inferieur a $b"
  10 est inferieur a 20
```

Tests de conditions

- ## Tests sur les fichiers

La syntaxe est : **test option nom_fichier**
et les options sont les suivantes :

Option	Rôle
-f	**Fichier normal**
-d	**Un répertoire**
-c	Fichier en mode caractère
-b	Fichier en mode bloc
-r	**Autorisation en lecture (pour le propriétaire)**
-w	**Autorisation en écriture (pour le propriétaire)**
-x	**Autorisation en exécution (pour le propriétaire)**
-s	**Fichier non vide (au moins un caractère)**
-e	**Le fichier existe**
-L	Le fichier est un lien symbolique
-u	Le fichier existe, SUID-Bit positionné
-g	Le fichier existe SGID-Bit positionné

Tests de conditions

- **Tests sur les fichiers**

Exemple :

```
$ ls -l
-rw-r--r-- 1 oracle system 1392 Aug 14 15:55 dump.log
lrwxrwxrwx 1 oracle system 4 Aug 14 15:21 lien_fic1  -> fic1
lrwxrwxrwx 1 oracle system 4 Aug 14 15:21 lien_fic2 -> fic2
-rw-r--r-- 1 oracle system 234 Aug 16 12:20 liste1
-rw-r--r-- 1 oracle system 234 Aug 13 10:06 liste2
-rwxr--r-- 1 oracle system 288 Aug 19 09:05 param.sh
-rwxr--r-- 1 oracle system 430 Aug 19 09:09 param2.sh
-rwxr--r-- 1 oracle system 292 Aug 19 10:57 param3.sh
drwxr-xr-x 2 oracle system 8192 Aug 19 12:09 rep1

$ test -f lien_fic1 ; echo $?
  1
$ test -x dump.log ; echo $?
  1
$ test -d rep1 ; echo $?
  0
```

Tests de conditions

- **Tests combinés par critères ET OU NON**

On peut effectuer plusieurs tests avec une seule instruction. Les options de combinaisons sont les mêmes que pour la commande **find**.

Critère	Action
-a	*AND*, ET logique
-o	*OR*, OU logique
!	*NOT*, NON logique

Exemple :

$ test -d "rep1" -a -w "rep1" && echo "rep1: repertoire, droit en ecriture"
rep1: repertoire, droit en ecriture

*$ test **!** -d "rep1" && echo "rep1 n'est pas un repertoire"*

Tests de conditions

- **Syntaxe allégée**

Le mot **test** peut être remplacé par les **crochets** ouverts et fermés « **[...]** ».
Il faut respecter un espace après et avant les crochets.

Exemple :

```
$ [ "$a" -lt "$b" ] && echo "$a est inferieur a $b"
    10 est inferieur a 20
```

Tests de conditions

- ## Tests étendus

  - Une nouvelle commande permet des **tests étendus** et généralement plus performants, par l'utilisation des doubles-crochets « **[[...]]** ».

    $ test –f fich ⇔ [-f fich] ⇔ [[-f fich]]

  - Il existe quelques différences entre la commande **test** (ou **[]**) et la nouvelle

    - Les caractères spéciaux (métacaractères) de recherche de fichiers ne sont pas interprétés.
    - **-a** et **-o** (*ET* et *OU*) sont remplacés par « **&&** » et « **||** ».

Exemple :

$ ***[[-d "rep1" && -r "rep1"]] && echo*** *"rep1 : repertoire avec droits write"*

Opérateurs && et || sur les codes de retour

- Les caractères « **&&** » et « **||** » permettent d'effectuer une exécution conditionnelle.

```
commande1  &&  commande2
commande1  ||  commande2
```

- La commande située après « **&&** » sera exécutée uniquement si la commande précédente a retourné **0** (**réussite**).
- La commande située après « **||** » ne sera exécutée que si la commande précédente a retourné autre chose que **0**.

Exemple :

$ rm toto || echo "toto non efface"

rm: ne peut enlever `toto': Aucun fichier ou répertoire de ce type toto non efface

Entiers et expressions arithmétiques

- Expressions arithmétiques habituelles évaluées à l'aide de
  - **expr** (révolue),
  - **let** (moderne et fait pour plusieurs expressions)
  - **((…))** pour une seule expression (Comme let)

Exemple

$ a=100
$ expr $a + 1
101 =>Affiche la valeur de **a** incrémenté de 1

$ let a=a+1 b=2*a/3
Incrémente **a** de 1 et affecte à **b** la valeur **2a/3**
Pas d'espace dans les expressions

$ ((y = 2 * x +3))
Affecte à **y** la valeur de l'expression **2x+3**

Entiers et expressions arithmétiques

- **Opérateurs identiques à ceux du langage C :**

  - **Arithmétiques :**

    () pour fixer l'ordre d'évaluation, **!** Négation, * multiplication,
    / quotient, **%** modulo, **+** addition, **-** soustraction,
    a++, **a--** post-incrémentation, post-décrémentation
    ++a, **--a** pré-incrémentation, pré-décrémentation

  - **Booléens :**

    <=, **>=**, **<**, **>**, **==** (égalité), **!=** (différent), **&&** (et logique),
    || (ou logique)

  - **Affectation :**

    = (affectation simple)

Entiers et expressions arithmétiques

- **Commande interne ((**
  - Cette commande interne est utilisée pour effectuer des opérations arithmétiques.

    Syntaxe : **((*expr_arith*))**
  - Son fonctionnement est le suivant :

expr_arith est évaluée ; si cette évaluation donne une valeur **différente de 0**, alors le **code de retour** de la commande interne **((** est égal à **0** ***sinon*** **le code de retour** est égal à **1**.

Exemple :

$ ***((1-5))*** *=> la valeur de l'expression arithmétique est égale à -4 donc le code de retour de* ***((*** *est égal à* ***0***

$ ***echo $?***

0

$ ***((5-5))*** *=> la valeur de l'expression arithmétique est 0, donc le code de retour de* ***((*** *est égal à* ***1***

$ ***echo $?***

1

Entiers et expressions arithmétiques

- ## Commande interne ((
  - Il est inutile d'utiliser le caractère de substitution **$** devant le nom d'une variable car il n'y a pas d'ambiguïté dans l'interprétation ; par contre, lorsqu'une expression contient des paramètres de position, le caractère **$** doit être utilisé.

Exemple :

```
$ declare -i a=2 2 b
$ (( b = a + 7 ))
$ echo $b
9
```

```
$ date
jeudi 21 décembre 2006, 19:41:42 (UTC+0100)
$
$ set $(date)
$
$ (( b = $2 +1 )) => incrémentation du jour courant
$
$ echo $b
22
```

Entiers et expressions arithmétiques

- ## Valeur d'une expression arithmétique

  - La commande interne **((** *expr_arith* **))** n'affiche pas sur la sortie standard la valeur de l'expression arithmétique *expr_arith.*
    Pour obtenir la valeur de l'expression arithmétique, on utilise la syntaxe : ***$((*** *expr_arith* ***))***

Exemple :

$echo ***$((*** *7 * 2* ***))***
14

$echo ***$((*** *a= 2*8* ***))***
16

$echo ***$((*** *7 < 10* ***))***
1

Entiers et expressions arithmétiques

Commande expr

- La commande **expr** permet d'effectuer des calculs sur des valeurs numériques, des comparaisons, et de la recherche dans des chaînes de texte.

Exemple :

```
$ expr 7 + 3
10

$ expr 7 \* 3
21

$ a a=`expr 13 – 10`
$ echo $a
3
```

Entiers et expressions arithmétiques

- **Commande let**

En bash, les var. sont toutes des chaînes de caractères
⇒Incapable de manipuler des nombres ⇒ pas opérations!!

On peut faire des calculs en utilisant l'instruction "**let**".

Syntaxe : **let *"expr_arith"***

Exemple :

```
$ let "b=12+3"
$ echo $b
15
```

Les opérations :

L'addition : +

La soustraction : −

La multiplication : ∗

La division : /

La puissance : ∗∗

Le modulo : **%**

Entiers et expressions arithmétiques

- **Variables de type entier**

  - Pour définir et initialiser une ou plusieurs variables de type entier, on utilise la syntaxe suivante : (même syntaxe avec la commande **typeset**)

    declare -i nom[=expr_arith] nom[=expr_arith] ...

  - L'avantage est qu'il devient possible d'effectuer des calculs et des comparaisons sans passer par **expr**. La commande **let** ou « **((...))** » permet des calcules sur variables.

  - Pour que la valeur d'une variable entière ne soit pas accidentellement modifiée après qu'elle ait été initialisée, il suffit d'ajouter l'attribut **r**.

  - pour connaître toutes les variables entières définies, il suffit d'utiliser la commande **declare -i**

Entiers et expressions arithmétiques

Variables de type entier

Exemples :

```
$ declare -i x=35     => définition et initialisation de la variable entière x
$ declare -i v w      => définition des variables entières v et w
$ v=12                => initialisation de v par affectation
$ read w
34                    => initialisation de w par lecture
$declare -ir a=-6     => seule la consultation est autorisée !

$ typeset -i resultat
$ resultat=6*7
$ echo $resultat
42
$ resultat=resultat*3
126
$ resultat=Erreur
Erreur: bad number
```

Délai d'attente

Commande sleep

- La commande **sleep** permet d'attendre le nombre de secondes indiqués. Le script est interrompu durant ce temps. Le nombre de secondes et un entier compris entre 0 et 4 milliards (136 ans).

Exemple :

$ sleep 10 ; date

Structures de contrôle

- **Structure de contrôle conditionnelle : if ... then ... else**

  - La structure **if then else fi** 'est une structure de contrôle conditionnelle.

    if *<commandes des_co condition>*
    then
        <commandes exécutées si condition réalisée>
    else
        <commandes exécutées si dernière condition pas réalisée>
    fi

  - On peut aussi préciser le **elif**, en fait un **else if**. Si la dernière condition n'est pas réalisée on en teste une nouvelle.

  - Le fonctionnement est le suivant : *commandes_condition* est exécutée ; si son code de retour est égal à **0,** alors la branche ***then*** est exécutée sinon c'est la branche ***elif*** ou la branche ***else*** qui est exécutée, si elle existe.

Autres écritures du IF

```
if condition ; then
commande
fi
```

```
if condition ; then command1; else commande2; fi
```

If Imbriquées

```
if    condition1
then
   Commandes
elif condition2
   then comandes
      elif ...
else
   commandes
fi
```

Structures de contrôle

1. Créer un script **rm1** pour supprimer un fichier Donnée en paramètre Si la suppression se passe bien afficher message « Fichier a été supprimé » Sinon message « fichier n a pas été supprimé »
2. Créer un script **affic** qui affiche la type de fichier donné en paramètre si fichier ordinaire afficher le contenu, si repértoire lister le contenu, sinon afficher message « type non traité »

programme ***rm1***

```
#!/bin/bash
if rm $1 2>/dev/null
then echo $1 a été supprimé
else echo $1n\'a pas ete supprime
fi
```

programme ***affic***

```
#!/bin/bash
if  [[ -f $1 ]]
then
echo $1 : fichier ordinaire
cat $1
elif [[ -d $1 ]]
  then
  echo $1 : repertoire
  ls $1
  else
  echo $1 : type non traité
fi
```

Structures de contrôle

- **Choix multiples case**

La commande **case esac** permet de vérifier le contenu d'une variable ou d'un résultat de manière multiple.

Syntaxe:

```
Case    valeur in
        Expr1) commandes ;;
        Expr2) commande ;;
        ....
Esac
```

- ***Expr*** est soit un simple texte, soit composé de caractères spéciaux. Chaque bloc de commandes lié à ***Expr*** doit se terminer par deux points-virgules.
- Dès que le ***Expr*** est vérifié, le bloc de commandes correspondant est exécuté.
- L'étoile en dernière position (chaîne variable) est l'action par défaut si aucun critère n'est vérifié.

Structures de contrôle

- **Choix multiplescase**

Exemples :

```
read langue
case $langue in
        Français)    echo "bonjour" ;;
        Anglais)     echo "hello" ;;
        Espagnol)    echo "buenos dias" ;;
        *)           echo "erreur de choix"
esac
```

```
for var in *
do
    echo "$var \c"
    type_fic=`ls -ld $var | cut -c1`
    case $type_fic in
;;      -) echo "Fichier normal" ;;
        d) echo "Repertoire" ;;
        b) echo "mode bloc" ;;
        l) echo "lien symbolique" ;;
        c) echo "mode caractere" ;;
        *) echo "autre" ;;
    esac
done
```

Structures de contrôle

- Choix multiplescase

Expr peut être construit à l'aide des caractères et expressions génériques de **bash.** Dans ce contexte, le symbole **|** signifie ***OU***.

Exemple2 :

```
read –p ”Entrez votre réponse : ” rep
case $rep in
  o|O ) echo OUI ;;
      *) echo Indefini
esac
```

Structures de contrôle

- **Choix multiples case**

Exemple3 :

```
#!/bin/bash
if  [ $# -ne 0 ]
then
echo  "$# parametres en ligne de commande"
else
echo  "Aucun parametre"
fi
case      $1 in
          a*) echo "Commence par a" ;;
          b*) echo "Commence par b" ;;
          fic[123]) echo "fic1 fic2 ou fic3" ;;
          *) echo "Commence par n'importe" ;;
esac
```

Structures de contrôle

Choix multiplescase

EXEMPLE

- Ecrire un script qui test chaque fichier dans le répertoire courant et pour chaque ressource indique si c'est un ficher, repertoire, lien , fichier ou autre et écrit cela dans un fichier détail.

```
echo -n "entrez les oprérondes x , y  et l'opération (+,-, *, /, %)?"
read  x y op
case $op in
+)  res=$((x+y));;
*)  res=$((x*y));;
/)  res=$((x/y));;
%)  res=$((x%y));;
-)  res=$((x-y));;
*) echo « erreur opérartion";;
esac
```

Structures de contrôle

Choix multiplescase

EXEMPLE

- Ecrire un script qui test si le caractère saisi est un nombre, majuscule, minuscule, alphabet, ou autre

```
#!/bin/bash
# Tester des suites de caractères.
echo "Appuyez sur une touche, puis faites ENTER."
read Touche
case "$Touche" in
  [[:lower:]]   ) echo "Lettre minuscule";;
  [[:upper:]]   ) echo "Lettre majuscule";;
  [0-9]   ) echo "Nombre";;
  *       ) echo "Ponctuation, espace blanc ou autre";;
esac
```

Structures de contrôle

- **Boucle for**

L'itération **for** possède plusieurs syntaxes dont les deux plus générales sont :

- **Première forme :**

```
for var
do
        suite_de_commandes
done
```

Lorsque cette syntaxe est utilisée, la variable ***var*** prend successivement la valeur de chaque **paramètre de position** initialisé.

Exemple : programme ***for_arg***

```
#!/bin/bash
for i
do
    echo $i
    echo "Passage a l'argument suivant ..."
done
```

$ ***for_arg un deux***

```
un
Passage a l'argument suivant ...
deux
Passage a l'argument suivant ...
```

Structures de contrôle

- **Boucle for**

- **Deuxième forme :**

```
for var in liste_mots
do
        suite_de_commandes
done
```

La variable ***var*** prend successivement la valeur de chaque mot de ***liste_mots***.

Exemple : programme ***for_liste***

```
#!/bin/bash
for a in toto tata
do
    echo $a
done
```

*$ **for_liste***

toto
tata

Structures de contrôle

- **Boucle for**

  - **Deuxième forme :**

    Si ***liste_mots*** contient des substitutions, elles sont préalablement traitées par **bash**.

Exemple : programme ***affich.ls***

```
#!/bin/bash
for a in tmp $(pwd)
do
    echo " --- $i ---"
    ls $i
done
```

Exemple : programme ***affich.user***

```
#!/bin/bash
echo "Liste des utilisateurs dans /etc/passwd"
for var in `cat /etc/passwd | cut -d: -f1`
do
    echo "$var"
done
```

Structures de contrôle

Structure for pour les expressions arithmétiques

Bash a introduit une nouvelle structure **for** adaptée aux traitements des expressions arithmétiques, itération issue du langage C. Elle fonctionne comme cette dernière.

Syntaxe :

```
for (( expr_arith1 ; expr_arith2 ; expr_arith3 ))
do
        suite_cmd
done
```

expr_arith1 : est l'expression arithmétique d'initialisation.
expr_arith2 : est la condition d'arrêt de l'itération.
expr_arith3 : est l'expression arithmétique qui fixe le pas d incrémentation ou de décrémentation.

Exemple For

- **Ecrire un** programme qui recopie tous les fichiers *.c du répertoire courant dans un répertoire fourni à par l'utilisateur

```
Read rep
mkdir $rep
for i  in '*.c'
do
cp $i  $rep/
Done
```

- *Afficher les variables* $1 $3 $4 $6 $7 $9 $10

```
for i in 1 3 4  6 7 9 10;do  echo $i; done
```

- Pour chaque fichier dans le répertoire courant afficher si c'est fichier, répertoire, lien ou autre et mettez le résultat dans un fichier détails

Structures de contrôle

- Structure for pour les expressions arithmétiques

Exemples :

```
#!/bin/bash
declare -i x
for (( x=0 ; x<5 ; x++ ))
do
        echo $(( x*2 ))
done
```

```
#!/bin/bash
declare -i x y
for  (( x=1,y=10 ; x<4 ; x++,y-- ))
do
        echo $(( x*y ))
done
```

Structures de contrôle

- **La boucle while**

La commande **while** permet une boucle conditionnelle « **tant que** ».
Tant que la condition est réalisée, le bloc de commande est exécuté.
On sort si la condition n'est plus valable

Syntaxe :

```
while condition
do
        commandes
done
```

Ou

```
while
        bloc d'instructions formant la condition
do
        commandes
done
```

Structures de contrôle

- **La boucle while**

Exemple1#!/bin/bash

```
while who | grep etudiant >/dev/null
do
    echo ”l’utilisateur etudiant est encore connecté”
    sleep 5
done
 utilisateur etudiant est encore connecte
```

Exemple2#!/bin/bash

```
while
 echo "Chaine ? \c"
 read nom
 [ -z "$nom" ]
do
 echo "ERREUR : pas de saisie"
done
echo   Vous avez saisi : $nom
```

Structures de contrôle

- ## La boucle until

La commande **until** permet une boucle conditionnelle « **jusqu'à** ».
Dès que la condition est réalisée, on sort de la boucle.

Syntaxe :

```
until condition
do
        commandes
done
```

Ou

```
until
        bloc d'instructions formant la condition
do
        commandes
done
```

Structures de contrôle

break et continue

- La commande **break** permet d'interrompre une boucle. Dans ce cas le script continue après la commande **done**. Elle peut prendre un argument numérique indiquant le nombre de boucles à sauter, dans le cadre de boucles imbriquées (rapidement illisible).
- La commande **continue** permet de relancer une boucle et d'effectuer un nouveau passage. Elle peut prendre un argument numérique indiquant le nombre de boucles à relancer (on remonte de n boucles). Le script redémarre à la commande **do**.

```
while true
do
    echo "Chaine ? \c"
    read a
    if [ -z "$a" ]
    then
        break
    fi
Done
```

Structures de contrôle

break et continue

```
#!/bin/bash
somme=0

#boucle infinie
while true
do
    echo "Entrez un nombre : \c"
    #saisie d'un nombre
    if read nombre
    then
    #si la saisie est incorrect, on remonte a la boucle
        if [[ "$nombre" != ?([+ -]) + ([0-9]) ]]
                then
        echo "la valeur saisie n'est pas un nombre ”
        continue
        fi
        somme='expr $somme + $nombre'
        else
        # l'utilisateur a saisie ^d : sortie de la boucle
        break
    fi
done
#Affichage du resultat
echo "\n la somme est : $somme\c"
```

SELECT

- Permet d'avoir un menu pour choisir la réponse

```
select var in selection1… selectionN
do instruction(s)
done
```

- Chaque sélection est affichée précédée par son numéro d'ordre
- La réponse est stockée dans une variable d'environnement REPLY
- $PS1 Le prompt principal du shell par défaut '$'.
- $PS2 Le prompt du shell utilisé quand une saisi va se faire par défaut '>'.
- L'utilisateur peut changer le prompt ou définir un nouveau avec $PS3

SELECT

- Ecrire un script qui selon que c'est une femme ou homme vous afficher « bonjour Madame » ou « bonjour Monsieur »

```
echo "Etes vous un homme ou une femme ?"
PS3="Femme ou Homme?"
select i in "homme" "femme"
Do
case $REPLY in
1)    echo "Bonjour monsieur "; break;;
2)   echo "Bonjour madame (ou mademoiselle) ";
  break;;
*) echo "mauvaise reponse";;
esac
done
```

Ce qui affiche

1) homme
2) femme

Femme ou Homme ?1

SELECT

```
#!\bin\bash
echo 'Select your terminal type:`
PS3="terminal? "
select term in  "Givalt VT100" "Tsoris VT220"  "Shande VT320"  "Vey
  VT5 20"do
case $REPLY in
1 ) TERM="gl35a";;
2 ) TERM="t2000" ;;
3 ) TERM="s531" ;;
4 ) TERM="vt99" ;;
* ) echo "invalid." ;;
esac
  if [[ -n $term ]]; then        echo " TERM is $TERM" ;  break
fi
done
```

INSTRUCTIONS LIÉES AUX BOUCLES

- A l'intérieur de ces différentes structures, il est important que l'utilisateur puisse fixer lui-même et selon ses propres critères la sortie de boucles.
  - La commande: continue [n] permet de sauter à la fin de la boucle et de recommencer une nouvelle boucle.
  - La commande: **break [n]** provoque, quand à elle la sortie de la boucle en cours et passe à l'instruction qui suit la boucle.
- Dans les deux cas, l'entier n permet d'indiquer que l'action se porte sur une boucle externe.
- Par exemple, **break 2** sortira de la boucle qui englobe la boucle dans laquelle cette instruction est écrite.

COMMANDES «BUILT-IN» DIVERSES

- Un certain nombre de commandes sont contenues dans le shell lui-même : Ce sont des mots réservés.
- En voici quelques exemples:
  - **eval chaînes**
  - **exit:** termine le shell courant,
  - **exec commande**
  - **shift**: décale d'un cran vers la gauche la liste des paramètres,

COMMANDES «BUILT-IN» DIVERSES

- Un certain nombre de commandes sont contenues dans le shell lui-même : Ce sont des mots réservés.
- **eval chaînes**...: Permet l'évaluation d'une ligne de commande.
  - Exemples:

```
CMD=´date | wc -c ´
$CMD            #erreur execution de la commande
date: invalid option -- 'c'   Try `date --help' for
  more information.
eval $CMD
43              #execution de la commande
```

Commande eval

- Exemples:

```
1. eval last='$'{$#};
   X=10; Y=X; echo –n '$'$Y;
   eval echo –n '$'$Y
   $X  10
2. wg=´eval who|grep´;
    $wg user
3. a=´echo Ceci est un exemple´;b=a;\$$b
    $a : commande not found #afdonne une erreur
    a=´echo Ceci est un exemple´;b=a;eval \$$b
  Ceci est un exemple     #affiche le contenu
4. cmd=´date´; eval $cmd #donne la date
```

Commande exec

- La commande exec exécute la commande à la place du processus (shell) courant

  ```
  exec [commande [arg ...]]
  ```

- Il n'y a pas de création de nouveau processu
- arg, . . . deviennent les arguments de commande
- si commande n'existe pas, le shell termine avec un code de retour
- Exemple:
  - `exec vi myfile`
  - `exec 1>/tmp/out`

COMMANDE EXPR

- ***expr arguments…***
  - Cette commande évalue les arguments comme une expression et le résultat est envoyé sur la sortie standard.
  - Ce n'est pas une commande interne au shell
  - « expr », comme toutes les commandes UNIX, doit avoir ses arguments séparés par des espaces.
  - La première utilisation de « expr » concerne les opérations arithmétiques simples.
    - Les opérateurs +, -, * et / correspondent respectivement à l'addition, à la soustraction, à la multiplication et à la division.
    - La seconde utilisation de la commande « expr » concerne la comparaison et la gestion des chaînes de caractères grâce à l'opérateur « : ».

EXPR ET TRAITEMENTS DES CHAÎNES DE CARACTÈRES

- expr exp1 \| exp2 renvoie exp1 si l'expression n'est pas nulle, sinon Exp2
- expr exp1 \& exp2 renvoie exp1 si l'expression n'est pas nulle, sinon exp2
- expr exp1 : exp2 comparaison des deux arguments (renvoie le nombre de caractères en commun)
- expr length exp retourne le nombre de caractères de exp
- expr substr exp n1 n2 retourne une sous chaîne de exp commençant a la place n1 et de n2 caractères
- expr index exp car retourne la position du caractère car dans la chaîne exp

EXEMPLES

```
X=3; Y=5
Z=`expr $X + 4 `
echo $Z                    #affiche 7
Z=`expr \(  $Z + $X  \)  \*  $Y`
echo $Z                              #affiche  50
X=abcdef
Z=`expr $X : '.*' `
echo $Z                              #affiche 6
 Z=`expr \( $X : '.*' \) + $Y`; echo $Z          #affiche 11
set a = 3; set b = `expr \( 1 + 3 \) \* $a`;  echo $b          #affiche 12
```


Calculs, traitements des chaînes de caractères

- **Exemple:** programme qui incrémente la valeur d'une variable de 1 et l'affiche tant que c'est inférieur à 10:

affichage des nombres entiers de 1 à 10:

```
a=1
 while [ $a -lt 10 ]
do
echo $a; a= ´expr $a +1 ´

done
```

Les fonctions

- **Définition d'une fonction**

  Le shell bash propose plusieurs syntaxes pour définir une fonction:

```
function nom_fct
{
suite_de_commandes
}
nom_fct () {
  commande...
}
function nom_fct
{ suite_de_commandes ;}
```

- Pour appeler une fonction, il suffit de mentionner son nom.

Définition de fonction en mode ligne

- Comme pour les autres commandes composées de bash, une fonction peut être définie directement à partir d'un shell interactif.

```
$ function f0
> {
> echo Bonjour tout le monde !
> }
$ f0     => appel de la fonction f0
Bonjour tout le monde !
$
```


L'EXÉCUTION D'UNE FONCTION

- L'exécution d'une fonction s'effectue dans l'environnement courant, autorisant ainsi le partage de variables.

- En mode commande

```
$ c=Coucou
$
$ function f1
> {
> echo $c    => utilisation dans la fonction d'une
  variable externe c
> }
$
$ f1
Coucou
$
```

Où définir une fonction

- Une définition de fonction peut se trouver en tout point d'un programme shell ; il n'est pas obligatoire de définir toutes les fonctions en début de programme.
- Il est uniquement nécessaire que la définition d'une fonction soit faite avant son appel effectif, c'est-à-dire avant son exécution :

```
function f1
{ ... ;}
suite_commandes1
function f2
{ ... ;}

suite_commandes2
```

- Dans le code ci-dessus, suite_commandes1 ne peut exécuter la fonction f2 (contrairement à suite_commandes2).
- Pour utiliser les fonctions dans tout le script il suffit de les déclarer au début

DECLARE -F

- Les noms de toutes les fonctions définies peuvent être listés à l'aide de la commande :

```
declare –F
#exemple de réponse
declare -f f0
declare -f f1
```

- Les noms et corps de toutes les fonctions définies sont affichés à l'aide de la commande : declare –f

- Pour afficher le nom et corps d'une ou plusieurs fonctions : declare -f nomfct ...
  - `declare -f f0`

Suppression d'une fonction

- Une fonction est rendue indéfinie par la commande interne : `unset -f nomfct ...`

```
$ declare -F
declare -f f0
declare -f f1
$ unset -f f1
$ declare -F
declare -f f0     => la fonction f1 n'existe
  plus !
```


Les arguments passés au script

- Les arguments d'une fonction sont référencés dans son corps de la même manière que les arguments d'un programme shell le sont : $1 référence le premier argument, $2 le deuxième, etc., $# le nombre d'arguments passés lors de l'appel de la fonction.

- Le paramètre spécial $0 n'est pas modifié : il contient le nom du programme shell.

- Ecrire un script qui affiche $0, $1 et $# avant la définition dela fonction, dans la fonction et à et après.

EXEMPLE

- Scriptargs.sh

```
#!/bin/bash
echo "Apres f : \$0 : $0"
echo " Apres f : \$# : $#"
echo " Apres f : \$1 : $1"

function farg
{
echo " --- Dans f : \$0 : $0"
echo " --- Dans f : \$# : $#"
echo " --- Dans f : \$1 : $1" ; }

echo "Avant f : \$0 : $0"
echo "Avant f : \$# : $#"
echo "Avant f : \$1 : $1"
```


LES VARIABLE : GLOBALE

- Par défaut, une variable définie à l'intérieur d'une fonction est globale ; cela signifie qu'elle est directement modifiable par les autres fonctions du programme shell.
- **Exemple : g**lob.sh

```
#!/bin/bash
function fun
{
var="Un"                    # creation de la variable var
}
function fdeux
{
var=${var}Deux          # premiere modification de var
}
fun
fdeux
var=${var}Princ         # deuxieme modification de var
echo $var
#résultat UnDeuxPrinc
```

LES VARIABLE : LOCALE

- Pour définir une variable locale à une fonction, on utilise la commande interne local.
- Sa syntaxe est :

```
local [option] [nom[=valeur] ...]
local prenom="zouhair"
```

- Les options utilisables avec local sont celles de la commande interne declare.
  - Par conséquent, on définira une ou plusieurs variables de type entier avec la syntaxe local –i (local –a pour un tableau local).
- La portée d'une variable locale inclut la fonction qui l'a définie ainsi que les fonctions qu'elle appelle (directement ou indirectement).

LES VARIABLE : LOCALE

- **Exemple:**

```
#!/bin/bash
  function f1
  {  local -i a=12                    => a est une
  variable locale à f1
  (( a+=1 ))
  echo " Dans f1 : a  => $a ";}
  f1
  echo "Dans main : a  => $a"  =>  erreur
```

Le return

- On peut définir une fonction qui retourne une valeur comme le montre l'exemple (min entre args):

```
function min()
{if [ -n $1 ] && [ -n $2 ]
 then
if [ $1 -lt $2 ]
   then       return $1
else       return $2
fi
 fi}
min 112 13
echo "$? est la plus petite valeur"
```

Restriction du return

- Il n'est pas possible de mettre une valeur supérieure à 256 comme argument et une fonction ne peut renvoyer une valeur supérieure à 256.
- Cette contrainte provient des codes de sortie.
- Pour palier au problème, il suffit d'utiliser une variable globale

```
VAL=
function min()
{  if [ -n $1 ] && [ -n $2 ]
 then
 if [ $1 -lt $2 ]   ; then      VAL=$1
else      VAL=$2
 fi
 fi
}
min 1132 1300
echo "$VAL est la plus petite valeur "
```

Exporter une fonction

- Pour qu'une fonction puisse être exécutée par un programme shell différent de celui où elle a été définie, il est nécessaire d'exporter cette fonction.

- On utilise la commande interne **export**: `export -f nomfct`

- Pour que l'export fonctionne, le sous-shell qui exécute la fonction doit avoir une relation de descendance avec le programme shell qui exporte la fonction.

EXEMPLE

- Progdef.sh

```
#!/bin/bash
function bonj
{
echo "bonj : Bonjour $1 "
}
bonj Madame          #appel normal à la fonction
```

- ProgShell.sh

```
#!/bin/bash
echo "appel a la fonction externe : bonj "
bonj Monsieur              #appel erroné à la
  fonction
```

Visibilité d'une Fonction

- Le programme devient:

```
#!/bin/bash
function bonj {
echo bonj : Bonjour $1
}
export -f bonj
```


Visibilité d'une Fonction

- Après son export, la fonction bonj sera connue dans les sous-shells créés lors de l'exécution de ProgShell.

- La visibilité d'une fonction exportée est similaire à celle d'une variable locale, c'est-à-dire une visibilité arborescente dont la racine est le point d'export de la fonction.

Substitution de fonction

- La commande interne return ne peut retourner qu'un code de retour.
- Pour récupérer la valeur modifiée par une fonction, on peut :
  - enregistrer la nouvelle valeur dans une variable globale,
  - ou faire écrire la valeur modifiée sur la sortie standard, ce qui permet à la fonction ou programme appelant de capter cette valeur grâce à une substitution de fonction : `$( fct [ arg … ] )`
- Exemple:

```
#!/bin/bash
function ajout
{echo " $1 coucou "
}
echo "la chaine est : $(ajout bonjour)"
```

Exemples de Fonctions récursives

1ère version :calcul de Factoriel de $1

```
#!/bin/bash
recursion=$1   # Nombre fournit en argument.
count=1
recurse ()
{
var=$1
while [ $var -gt 1 ]
  do
  ((count=count * var));(( var-- ))
    recurse $var
  done
}
recurse $recursion
echo "le factoriel est $count"
```

FONCTIONS RÉCURSIVES

2ème version :

```
#!\bin\bash
fact ()
{
local nombre=$1
 if [ $nombre -eq 0 ] ; then    factoriel=1
  else
let "decrnum = nombre - 1"
fact $decrnum
let "factoriel = $nombre * $?"
fi
return $factoriel;}
fact $1
echo "Le factoriel de $1 est $?."
```

APPEL DE FONCTION

- Lorsque la fonction dans un programme shell est définie dans un autre on l'exécute dans l'environnement du fichier shell « principal ».

- Dans l'exemple suivant:
  - soit f une fonction
  - f définie dans le fichier *def_f.sh*
  - et nous voulons l'appeler depuis le fichier shell *appel.sh*

- Il suffit d'utiliser la commande interne `source` ou `'.'` devans le nom du script contenant la fonction

- Seule la permission lecture est nécessaire pour *def_f*.

Exemple

- **Soit le script Appel.sh :**

```
#!/bin/bash
source def_f    # ou plus court : . def_f
                # Permissions suff de def_f : r--r--r--
x=2
f               # appel de la fonction f contenue dans def_f
```

- **Définition def dans le script def_f.sh**

```
#!/bin/bash
function f()
{echo $((x+2));}
```

APERÇU SUR LES TABLEAUX

- **Instanciation**

  tab[1]=1
- appel de l'ensemble du tableau:

  ${tab[*]}
- appel d'un élément :

  ${tab[i]}
- Declare -a tab # l'index de tab est un nombre
- **Exemple :**

```
#!/bin/bash
nom[0]='Bonjour'
nom[1]='Monsieur'
echo ${nom[0]}    #affiche 'Bonjour'
echo ${nom[*]}    #affiche 'Bonjour Monsieur'
```

EXEMPLE

```
tab=(red green blue yellow magenta)
len=${#tab[*]}
echo “Le tableau a $len éléments.  Ce sont:“
i=0
while [ $i -lt $len ]
 do  echo "$i: ${array[$i]}“
  let i++
done
```


Tableau associatif

- **Version bash 4**
- **Declare** -A ARRAY
- **Initialisation**
- Le tableau peut être initialisé directement :
  - tab=**([**" nom"**]**="berbère" **[**"age"**]**=26**)** ;
- ou élément par élément :
  - ARRAY**[**"nom"**]**="berber"
  - ARRAY**[**"C"**]**="26"
  - ARRAY**[**"B"**]**=10 ;
  - ARRAY**[**"FHH"**]**='c'

Tableau associatif

- **Accès aux éléments**
- Les différents modes d'accès aux éléments des tableaux sont ensuite les mêmes que pour un tableau indexé :

```
echo ${ARRAY["A"]} ;
echo ${ARRAY["FHH"]}
```

- **Informations sur le tableau**
  - Le nombre d'élément du tableaux :

```
echo "Nombre d'éléments : "${#ARRAY[*]} "ou"
  ${#ARRAY[@]}
```

  - Les indexes sont retourné par "${!ARRAY[@]}" ou "${!ARRAY[*]}" :

```
echo "Indexes du tableau" ${!ARRAY[@]}
```

- **Parcours du tableau**

```
for elem in ${!ARRAY[*]}
 do echo "Key \"${elem}\" : Value :
  "${ARRAY[${elem}]}
 done
```