Introduction aux systèmes informatiques

Rushed Kanawati

Département R & T, IUT de Villetaneuse
Université Paris 13
rushed.kanawati@lipn.univ-paris13.fr

September 28, 2012

Objectifs & organisation du cours

- Objectifs : Comprendre le fonctionnement d'un système informatique
- Utilisation (avancée) d'une machine Linux (Unix)
- Organisation : 5 cours, 6 TP, un contrôle court et un contrôle long.

Plan du cours

- Introduction
- Système de fichiers
- Les processus
- Commandes utilitaires
- Environnement utilisateur et scripts

Plan du cours

- Introduction
  - Composants d'un système informatique
  - Système d'exploitation : concepts de base
  - Le système Linux
- Système de fichiers
- Les processus
- Commandes utilitaires
- Environnement utilisateur et scripts

Système d'information

Système informatique = Matériels + Logiciels

Système informatique : Matériels

- **Traitement de l'information** : ordinateurs, serveurs, terminaux mobiles, etc.
- **Entrée /Sortie** : Moniteurs, scanners, imprimantes, etc.
- **Stockage de l'information** : disques et mémoires de stockage et d'archivage.
- **Communication** : cartes réseaux, commutateurs, routeurs, support de transmission.

Système informatique : Logiciels

- Logiciels de base : Applications pour le traitement de problèmes usuels
  - **Système d'exploitation** : Gestion d'un ordinateur : lancement de programmes, gestion du processeur et des périphériques, etc.
  - **Services** : Systèmes graphiques, communication réseaux (protocoles), *base de données*, . . . , etc.
  - **Outils** : Traitement de texte, Impression, Compilateurs, Interpréteurs, Outils réseaux (mail, navigateur web, etc).
- Applications : Logiciels achetés ou développés pour résoudre des problèmes spécifiques : Jeux, éducation, publication, dessins, calcul scientifique, . . . , etc.

Système d'exploitation

- Objectif : offrir une interface de gestion d'un ordinateur
- **Système d'exploitation = Machine Virtuelle**
- Cacher l'hétérogénéité des matériels (processeurs, disques, cartes réseaux, etc.)
- Plusieurs classes de systèmes :
  - Mono/Multi-utilisateurs
  - Mono/Multi-tâches
  - Monoprocesseur, Multiprocesseur, Réparti, Parallèl
  - Temps réel, embarquée, ...
- Examples : Windows, **Linux/Unix**, MAC OS X, IOS 6, Android, ...

Système d'exploitation : Principaux concepts

- **Utilisateur** (user): réel ou logiciel, possède des droits et nécessite d'être authentifié lors de la connexion.
- **Fichier** (file) : une structure logique qui délimite une zone de stockage de données sur disque. Il est caractérisé par un type qui dépend de la nature des données qu'il contient et par des attributs (comme les droits, date de création, date de modification,. . . , etc.)
- **Processus** (process) : Un programme en cours d'exécution.

Le système Unix/Linux: Historique

- Unix : Création aux laboratoires Bell (USA) en 1969.
- But : gestion d'un mini-ordinateur
- Système développé en langage C.
- Intéresse rapidement universités puis constructeurs.
- Linux : Version d'Unix pour micro-ordinateurs, 1991.
- Différentes distributions : Slackwre, Red Hat, Debian, SuSE, Ubuntu, . . . , etc.

Le système Unix/Linux : Caractéristiques

- Multi-plateforme
- Multi-tâches et multi-utilisateur
- **Un système ouvert**
- Différents environnements de commandes : les shells
- Systèmes de fichiers hiérarchique
- Gestion hiérarchisée de processus

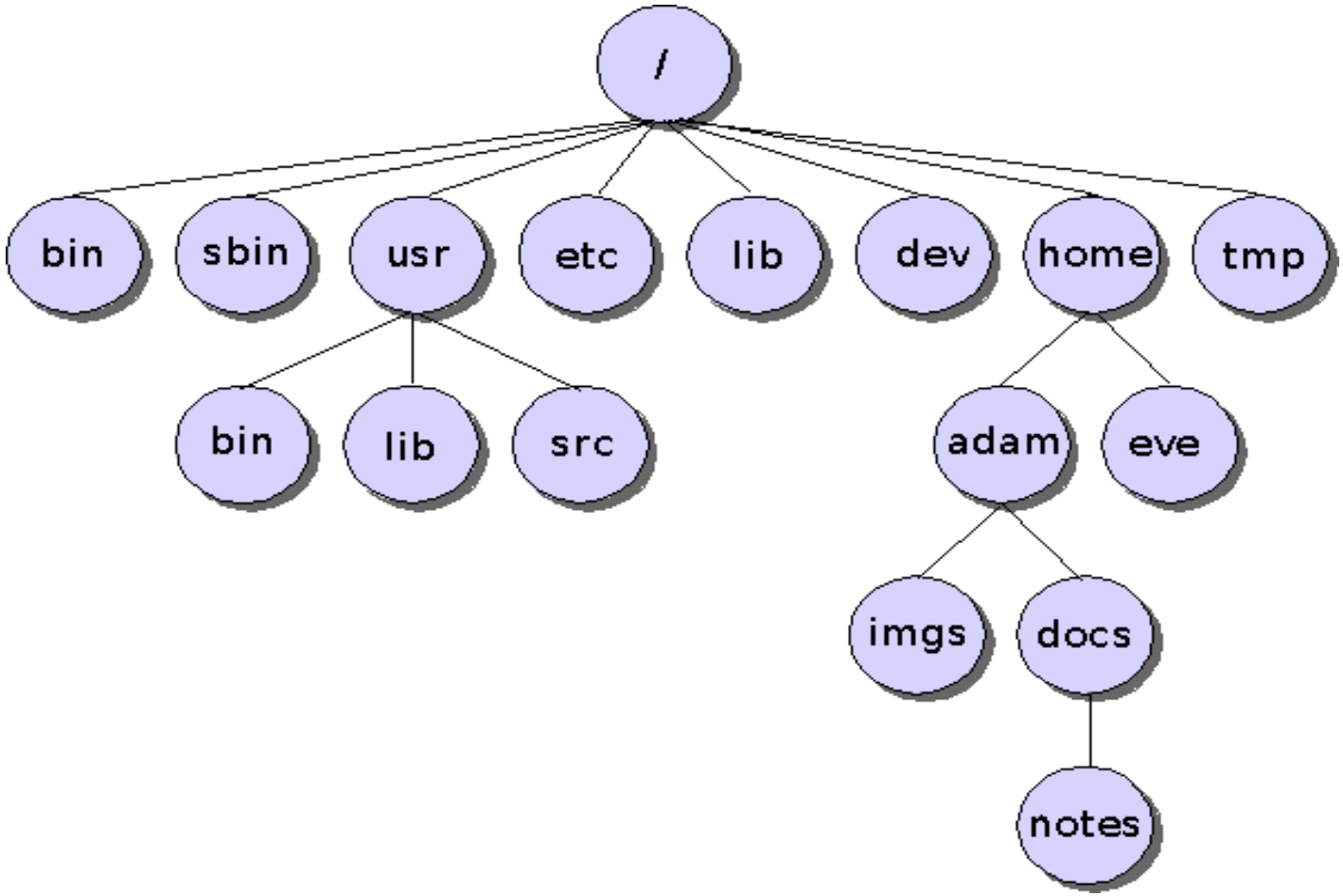
Plan du cours

- Introduction
- Système de fichiers
  - Structure
  - Navigation
  - Manipulation
  - Gestion de droits
- Les processus
- Commandes utilitaires
- Environnement utilisateur et scripts

Système de fichiers

- Un fichier est une suite de données sauvegardée sur un support de stockage permanent (disque dur, mémoire flash, CD).
- Le codage utilisé détermine la nature du fichier : texte, image, son, vidéo, documents, programme exécutable, etc.
- Les fichiers sont regroupés dans des dossiers nommés répertoires.
- Dans un même répertoire deux fichiers ne peuvent pas avoir le même nom.
- Un répertoire peut contenir des fichiers et des sous-répertoires $\rightarrow$ Système de fichiers sous forme d'arbre.

Structure arborescente du système de fichiers



Règles de nommage

- Nom d'un fichier : **nom.extension**
- (en théorie) Linux autorise tous les caractères pour nommer les fichiers.
- **(En pratique) Limitez-vous aux caractères alphabétiques, aux chiffres, ainsi qu'au point (.) et au caractère de soulignement (_).**
- Linux fait la différence entre les caractères majuscules et les minuscules.
  `Essai.txt` $\neq$ `essai.txt`
- L'extension permet à l'utilisateur de se souvenir du type du fichier : `txt`, `pdf`, `png`, `jpeg`, `avi`

Répertoires

Répertoire courant désigné par .

Répertoire de connexion d'un utilisateur C'est le répertoire courant par défaut lors de la connexion au système.
Exemple: pour l'utilisateur `eve` le répertoire de connexion est `/home/eve`

Répertoire père Le répertoire qui englobe le répertoire courant est désigné par ..

Répertoire racine désigne par / est le seul répertoire qui n'a pas de répertoire père

Fichiers et répertoires : Références

- Chaque fichier ou répertoire est désigné d'une **manière unique** par le chemin depuis la racine. On appelle ce chemin référence absolue
- On peut aussi désigner un fichier par des références relatives
- Exemples : pour l'utilisateur `adam` et à partir de son répertoire de connexion, le fichier `notes` peut être désigné par les références :
  - `./docs/notes`
  - `~/docs/notes`
  - `./imags/../docs/notes`

Commandes de navigation

- `pwd` : (Print Working Directory) affiche le nom du répertoire courant
- `cd` : (Change Directory)
  `cd référence_répertoire`, Le répertoire courant devient le répertoire `référence_répertoire`
  `cd ~nom_utilisateur`, Le répertoire courant devient le répertoire de connexion de l'utilisateur *nom_utilisateur*
  `cd`, Le répertoire courant devient le répertoire de connexion de l'utilisateur qui exécute la commande.
  `cd ..`, Aller vers le répertoire père du répertoire courant.

Commandes de navigation

- `ls` : (list)
  `ls` affiche le contenu du répertoire courant
  `ls référence_répertoire` affiche le contenu du répertoire désigné (s'il est accessible).
  `ls -l référence_répertoire` affiche le contenu avec des informations détaillées : noms des fichiers/sous-répertoires, tailles, droits, propriétaires, dernière date de modification

Commandes de manipulation

- `cp` : (copy)
  `cp ancien nouveau` : copier le fichier *ancien* dans *nouveau*
  `cp -R répAncien répNouveau` : copie d'une manière récursive le répertorie *répAncien* dans *répNouveau*
- `mv` : (move)
  `mv ancien nouveau` : renommer ou dépalcer le fichier *ancien* par *nouveau*
- `rm` : (remove)
  `rm fichier` : effacer le fichier *fichier*
- `mkdir` : (make directory)
  `mkdir répertoire` : créer le répertoire *répertoire*
- `rmdir` : (remove directory)
  `rmdir répertoire` : effacer le répertoire *répertoire*
- `touch` :
  `touch référence` : si le fichier désigné existe alors mettre à jour la date de dernière modification, sinon créer un fichier vide avec le nom

Méta-caractères

- `*` : remplace toute chaîne de caractères ne commençant pas par .
- `?` : remplace n'importe quel caractère.
- `[abc]` : un caractère parmi a,b ou c
- `[a-d]` : un caractère dans l'intervalle : `a` à `d`.

Méta-caractères : Exemples d'utilisation

- `ls -l *.txt` : affiche seulement les fichiers ayant l'extension `txt`
-
- `rm td[1-3].pdf` : effacer les fichiers `td1.pdf`, `td2.pdf` et `td3.pdf`
-
- `cp/tmp/?2011.data .` : copier tous les fichiers dans le répertoire `/tmp` dont le nom est composé d'une chaîne de cinq caractères qui se termine par 2011 et qui ont l'extension `data` dans le répertoire courant

Chercher un fichier/répertoire : la commande `find`

- `find répertoire critère action` : chercher dans le répertoire `répertoire` les fichiers satisfaisant `critère` et exécuter `action` sur chaque fichier retrouvé.
- Exampale : `find ~ -name "*.pdf" -print` : Afficher tous les fichiers dont le nom se termine par l'extension `pdf` et qui sont dans la sous arborescence enracinée dans le répertoire de connexion de l'utilisateur.

La commande `find` : actions

- `-print` : afficher le résultat de la recherche sous la forme d'une liste de noms.
- `-exec cmd {} \;` : exécuter la commande `cmd` sur chaque résultat retrouvé
- `-ok cmd {} \;` : même effet que `exec` mais en demandant la confirmation de l'utilisateur pour chaque exécution de `cmd`
- Exemple : `find ~ -name "*.pdf" -exec ls -l {} \;` Afficher les détails sur tout les fichiers ayant l'extension `.pdf` dans le répertoire de l'utilisateur.

La commande `find` : Critères

- `-name nom` : rechercher selon le nom de l'objet.
- `-size [+|-]n[c|k|b]` : rechercher selon la taille de l'objet. + : plus grand, - : plus petit, rien : exactement, c : octet, k : kilo octet, b : bloc de 512 o
- `-mtime [+|-]j` : rechercher selon la date de modification de l'objet. + : plus grand, - : plus petit, rien : exactement, j = nombre de jours
- `-atime [+|-]j` : rechercher selon la dernière date d'accès à l'objet.
- `-ctime [+|-]j` : rechercher selon la date de création de l'objet.
- `-type [f|d|l]` : rechercher selon le type de l'objet (f : fichier, d : répertoire et l : lien)
- `-newer objet` : rechercher les objets plus récents que l'objet donné.

La commande `find` : Critères

- `-perm droit` : rechercher selon les droits associés à l'objet.
- `-user uid` : rechercher selon l'identité du propriétaire.
- `-group gid` : rechercher selon l'identité du groupe.
- `-nouser` : rechercher les objets sans utilisateur
- `-nogroup` : rechercher les objets sans groupe.

La commande `find` : Critères composés

- `-a` : ET logique
- `-o` : OU logique
- `!` : NON logique
- Examples :
  - `find /tmp -size 0 -a -user dupont -print` : Afficher les noms des fichiers sous l'arborescence `/tmp`, de taille 0 octets appartenant à l'utilisateur `dupont`
  - `find / -user dupont -o -user martin -print` : Afficher les fichiers appartenant à l'utilisateur `dupont` ou à l'utilisateur `martin`

Liens

- Un lien est une référence supplémentaire qui peut désigner une ressource existante (i.e. fichier) à partir de n'importe quel nœud de l'arborescence du système de fichier.
- C'est une sort de deuxième étiquette qui implante un raccourcis dans l'arborescence.
- Chaque ressource compte le nombre de références qui lui désignent : compteur de références.
- Le compteur de références est affiché par la commande (`ls -l`.
- Exemple :
  `-rw-r--r-- 2 rushedkanawati staff 0 21 sep 04:55 ex`
- Une ressource est complètement supprimée si le compteur de références est égale à 0.

Liens : commandes

- Deux types de liens : Physique, Symbolique
- `ln ressource nom_lien_physique`
- `ln -s ressource nom_lien_symbolique`
- Example :

```
touch example.txt
ln example.txt example.physique
ln -s example.txt example.symbolique
ls -l example.*
Résultat :
-rw-r--r-- 2 rushedkanawati staff 0 21 sep 05:06 example.phy
lrwxr-xr-x 1 rushedkanawati staff 11 21 sep 05:07 example.sym
-> example.txt
-rw-r--r-- 2 rushedkanawati staff 0 21 sep 05:06 example.txt
```

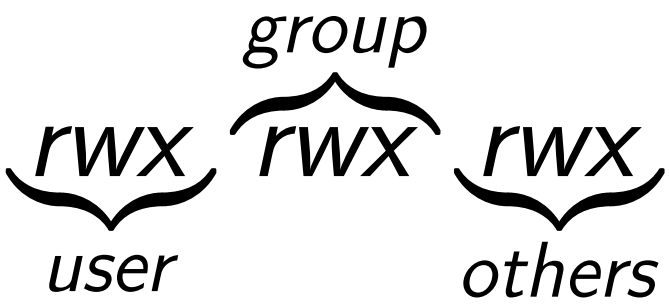
La commande `file`

- `file référence` : permet de se renseigner sur le type de référence donné et la nature des données contenues dans le fichier désigné.
- Type de références : Répertoire (Directory), Fichier ou Lien.
- Nature de données : texte, PDF, image (PNG, GIF, ?), . . . , etc.

Gestion des droits d'accès

- Chaque utilisateur du système est :
  - désigné par un identificateur unique : `uid`
  - peut appartenir à un ensemble de groupes. Chaque groupe est à son tour désigné par un identificateur `gid`
- La commande `id user_login` donne l'uid de l'utilisateur et la liste des groupes auxquels il appartient.
- Chaque ressource (i.e. fichier) a un seul propriétaire.
- On spécifie pour chaque ressource 3 groupes de droits d'accès :
  - Les droits du propriétaire : `user`
  - Les droits des utilisateurs appartenant au même groupe que le propriétaire : `group`
  - Les autres : `others`

Gestion de droits d'accès

- Trois types de droits d'accès :
  - Droit de lecteur (read) désigné par : r
  - Droit de écriture (write) désigné par : w
  - Droit d'exécution (exécute) désigné par : x
- Le droit d'exécution sur un répertoire donne l'autorisation de le traverser.
- Les droits d'accès sont codés alors sur 9 bits :
- 

- Expression des droits en code octal :
  - `rw-r----` → 640
  - `rwxrwxr--` → 774

Manipulation des droits d'accès

- Les droits par défaut sur les ressources crées par un utilisateur peuvent être fixés par la commande umask
- `umask code_octal` : le code est l'inverse des droits à donner.
- La commande `umask` n'agit pas sur le droit de l'exécution (`x`)
- umask 003 donne les droits 664

Manipulation de droits d'accès

- `chmod droits référence` : fixer les droits d'accès de la référence donnée.
- Les droits peuvent être exprimés de deux manières :
  - Code octal: chmod 640 exemple.txt
  - Code symbolique : `cible opérateur droit`
    - Cible : `u` (user), `g` (group), `o` (others)
    - Opérateur : `+` (donner) , `-` (ôter), `=` laisser inchangé)
    - Droit : `r` (read), `w` (write), `x` (execute)
    - chmod ug+rw,u+x toto : le propriétaire et le groupe ont le droit de lire et modifier, le propriétaire a le droit d'exécuter. Attention les autres droits ressent inchangés.
    - chmod o=g,o-x toto : Les autres ont les mêmes droits que le groupe mais sans le droit d'exécution.
    - chmod ugo+r,ug+w,go-x,u+x toto : que fait cette commande ?

Changement de propriétaire/groupe

- La commande `chown` (change owner)
  - chown user file : Le proprietarie de *file* devient *user*.
  - chown -R user dir : appliquer le changement sur toutes les ressources dans dir.
  - chown user.group dir : changer le groupe aussi.
- La commande `chgrp` (change group)
  - chgrp group file : Le group de *file* devient *group*.

Plan du cours

- Introduction
- Système de fichiers
- Les processus
  - Généralités
  - Gestion et interaction
  - Signaux
- Commandes utilitaires
- Environnement utilisateurs et scripts

Les Processus

- Un processus est un programme en exécution
- Un processus est identifié par un numéro unique `PID` (Process IDentifier).
- Chaque processus linux est créé par un autre processus, appelé **processus père** et désigné par un identificateur `PPID` (Parent Process Identifier). Le processus créé est appelé **processus fils**.
- Un processus primitif (`init`, `PID` $\leftarrow 1$) est crée au démarrage du système.

Les Processus

- Un processus de gestion de swap (`pid` $\leftarrow 0$) est créé avant afin de se charger de migration de processus entre la mémoire centrale et la mémoire secondaire (disque).
- La création d'un processus fils se fait par **clonage** du processus père grâce à l'appel système **fork()**.
- Conséquence : Les processus sont organisés dans une structure arborescente
- **L'arrêt d'un processus père entraîne l'arrêt de tous les descendants**
- La commande nohup cmd permet au processus fils exécutant la commande `cmd` de survivre à l'arrêt de son processus père.

Exécution de processus

- Plusieurs processus s'exécutent en même temps sur le même processeur en mode : **temps partagé**
- Etats d'un processus :
  - **prêt** si toutes les ressources requises sont disponibles sauf le processeur.
  - **bloqué** si il manque une ressource autre que le processeur.
  - **en exécution** si le processeur est alloué au processus
- Le système d'exploitation se charge de choisir un processus parmi les processus **prêts** pour lui allouer le processeur pendant une courte période du temps appelée : **quantum**.
- Différentes politiques de sélection de du processus prêt à exécuter (ex. système de priorité).
- A l'éxpiration du quantum le processus en exécution bascule dans l'état prêt.

Enchaînement de processus

- **comm1 ; comm2** : Exécution séquentielle. La commande `comm2` est exécutée après la fin de l'exécution de `comm1`
- **comm1 | comm2** : Exécution en parallèle des deux commandes. La sortie de `comm1` est l'entrée de `comm2`
- Exemple :
  - `find ~ -name "*.pdf" ; ls -a`
  - `find ~ -name "*.pdf" | ls -a`
  - La première commande trouve tous les fichiers ayant l'extension `.pdf` dans la sous-arborescence de l'utilisateur puis affiche le contenu du répertoire courant.
  - La deuxième commande affiche les informations détaillés sur les fichiers `pdf` qui se trouvent dans l'arborescence de l'utilisateur.

Tâche de fond

- **appli &** : l'application `appli` est exécutée en parallèle avec l'interpréteur de commandes. Ainsi l'interpréter n'attends pas la fin de l'application et peut exécuter d'autres commandes.
- L'interpréteur affiche le numéro du processus crée.
- Exemple :
  - `xeyes &`
  - `[1] 2021`
- **Il est préférable de lancer toutes les applications dotées d'interface graphique en tâche de fond.**

Processus : Commandes de gestion

- **ps (process status)** : afficher des informations sur les processus exécutés dans le système.
- Principales options :
  - -u uid : afficher les processus appartenant à l'utilisateur `uid`
  - -a : afficher des informations sur les processus de tous les utilisateurs `uid`
  - -f : afficher des informations détaillées sur chaque processus y compris le `PPID`
  - -x : afficher tous les processus y compris ceux qui ne sont pas contrôlés par un terminal.

Processus : Commandes de gestion

- **top** : affiche l'état des processus en temps réel
- **jobs** : affiche la liste de tâches lancées par l'interpréteur de commandes.
- **fg** : permet de ramener une tâche à l'exécution en premier plan.
- **bg** : permet de ramener une tâche à l'exécution comme tâche de fond.

Interaction avec les processus : Les signaux

- Un signal est un événement asynchrone destiné à un ou plusieurs processus.
- Les deux commandes **trap -l** et **kill -l** peuvent donner la liste des signaux systèmes.
- Chaque signal est identifié par un entier et par un nom symbolique.
- Un signal peut être envoyé par le système ou par un autre processus.
- A la réception d'un signal, un processus interrompt son exécution et exécute une fonction de traitement du signal avant de reprendre son exécution (si c'est encore possible).
- Chaque signal est associé à un traitement par défaut que le processus peut changer ou même ignorer.

Exemples de signaux

Signal	Num.	Fonction	Raccourcis clavier
SIGINT	2	Arrêter un processus	CTRL+c
SIGQUIT	3	Quitter après un *core lump*	CTRL+\
SIGSTOP	20	Suspendre un processus	CTRL+z
SIGKILL	9	*Tuer* un processus	-
SIGCONT	18	Reprendre un processus	-

Envoi & traitement de signaux

- **kill -signum pid** : envoyer le signal ayant le numéro `signum` au processus identifié par `pid`.
- `kill -9 2323` : arrêter immédiatement le processus 2323.
- **trap 'cmd' numsig** : `cmd` est le nouveau traitant à exécuter à réception du signal `signum`.
- **trap 'écho hello' 2** : On affiche `hello` chaque fois on envoie le signal `SIGCONT` (taper `CTRL+c`)
- **trap ' ' numsig** : inhiber le traitement du signal `numsig`.
- **trap numsig** : rétablir le traitant par défaut du signal `numsig`.

Plan du cours

- Introduction
- Système de fichiers
- Les processus
- Commandes utilitaires
  - Manipulation de fichiers textes
  - Outils réseaux
- Environnement utilisateurs et scripts

Manipulation de fichiers textes

- **echo** `chaîne` : afficher `chaîne` sur la sortie standard (l'écran).
  - `echo hello world` : affiche `hello world`.
- **more** `fichier` : afficher le contenu de `fichier` page par page. Le passage à la page suivante se fait en appuyant sur `espace`.
- **cat** `fichier` : afficher le contenu de `fichier` sur la sortie standard.
- **wc** `fichier` : afficher le nombre de caractères, de mots ou de lignes dans `fichier`
  - `wc -l toto` : affiche le nombre de lignes dans `toto`.
- **head -n** `fichier` : afficher les premiers `n` lignes de `fichier`.
- **tail -n** `fichier` : afficher les dernières `n` lignes de `fichier`.
  - `tail -2 toto | head -1` : affiche l'avant-dernière ligne du fichier `toto`

Les filtres textes

- **sort** `fichier` : trier les lignes de `fichier`.
- **diff** `fich1 fich2` : afficher les lignes de `fich1` et `fich2` qui sont différentes.
- **uniq** `fichier` : remplace un bloc de lignes identiques par une seule ligne (supprimer la duplication).
- **tr** `str1 str2` : remplacer chaque occurrence de `str1` par `str2`.
- **cut options** `fichier` : découper chaque ligne de `fichier` en liste de valeurs et renvoyer un des champs selon les options.
  - `cut -d:  -f2 toto` : Les lignes du fichier `toto` seront découpées en champs de valeurs séparés par : (l'option `-d` pour délimiter ou séparateur). On affiche pour chaque ligne ainsi découpée le deuxième champ (option `-f2`).

Filtre : La commande grep

- **grep** `motif fichier` : afficher les lignes dans `fichier` qui contiennent `motif`.
- Quelques options (voir `man grep` pour une liste complète) :
  1. `-`*num* : afficher `num` lignes avant et après la ligne où figure le motif.
  2. `-A` *num* : afficher `num` lignes après la ligne où figure le motif.
  3. `-B` *num* : afficher `num` lignes avant la ligne où figure le motif.
  4. `-n` : affiche le numéro de ligne.
  5. `-c` : affiche **le nombre** d'occurrence du motif dans le fichier.
  6. `-i` : ignorer les différences entre minuscules et majuscules.
  7. `-v` : chercher les lignes qui **ne contiennent pas** le motif.
- Variations sur le motif :
  1. `grep '^[aA]' fich` : chercher les lignes qui commencent par `a` ou `A` dans le fichier `fich`.
  2. `grep '[aA]$' fich` : chercher les lignes qui se terminent par `a` ou `A` dans le fichier `fich`.

Filtres textes : Exemples

- Soit le fichier `annuaire.txt` dont le contenu est :
  Ivan:le terrible:ivan.leterrible@kremlin.ru
  Attila:le hunnique:attila.hunnique@pekin.cn
  Alexandre:le grand:alexis@athene.gr
  Attila:le hunnique:attila@google.com
- head -1 annuaire.txt | cut-d: -f3 affiche `ivan.leterrible@kremlin.ru`
- more annuaire.txt | grep -i '^attila' | cut -d: -f3 | wc -l : affiche 2, le nombre d'adresses mail de Atilla.
- Que font les deux commandes suivantes ? :
  - more annuaire.txt | cut -d: -f1-2 | sort | uniq | wc -l
  - tail -1 annuaire.txt | tr : '\n' | wc -l

Redirection des entrées/Sorties

- Les entrées sont les paramètres fournies à une commande et les entrées sont les affichages faites par une commande.
- Entrée Standard : le clavier.
- Sortie Standard : l'écran.
- Redirections :
  - `commande < fichier` : l'entrée de `commande` est le contenu de `fichier`
  - `commande > fichier` : l'affichage fait par `commande` sera écrit dans `fichier` et non à l'écran. Si `fichier` existe son ancien contenu est effacé.
  - `commande >> fichier` : l'affichage fait par `commande` sera écrit à la fin `fichier` et non à l'écran.
  - Exemple :
    - `echo "première ligne" > toto`
    - `echo "deuxième ligne" >> toto`

Outils réseaux : les adresses

- La communication entre machines nécessite l'utilisation d'interfaces réseaux.
- Exemple : **cartes réseaux Ethernet**, port série, USB, . . . , etc.
- Une interface réseau a une **adresse physique**. Dans le cas d'une carte Ethernet on parle d'adresse `MAC`[1]
- Une adresse MAC est une suite de 6 octets qui identifie d'une manière unique une carte. Souvent noté en Hexa.
- Une interface réseau relié à l'Internet doit avoir `une adresse logique` dite `adresse IP`[2].
- Une adresse IP (version 4) est une suite de 32 bits notée sous forme de 4 entiers séparés par .
- `10.0.0.1` est l'adresse `00001010 00000000 00000000 00000001`

[1] à étudier dans le module R2

[2] à étudier dans le module R4

Outils réseaux : les adresses

- Les machines connectées à Internet peuvent avoir aussi des `adresses symboliques` plus faciles à manipuler
- Exemple : `www.iutv.univ-paris13.fr` correspond à l'adresse `194.254.164.240`
- Des systèmes de mise en correspondances entre adresses symboliques et adresses IP. Et entre adresse IP et des adresses physiques sont en place (à étudier dans les modules R4, TR2).

Commandes réseaux de base

- `hostname` : afficher le nom symbolique de la machine.
- `ifconfig` : outil de configuration des interface réseaux (pour l'administrateur). Ca permet aussi de se renseigner sur les caractéristiques d'une interface réseau donné da la machine.
- `/sbin/ifconfig eth0` : affiche les information de configuration de l'interface `eth0` (première carte réseau Ethernet de la machine).
- `nslookup adrsym` : chercher à trouver l'adresse IP correspondant à l'adresse symbolique donnée.

Commandes réseaux de base

- `traceroute adr` : donner la route entre la machine locale et la machine désignée par *adr* si la route existe.
- `ping adr` : permet de tester si la machine ayant l'adresse *adr* existe et si oui donner des informations sur la qualité de la route vers cette machine.
- `ping www.univ-paris13.fr`
  `PING www.univ-paris13.fr (194.254.164.240): 56 data bytes`
  `64 bytes from 194.254.164.240:` $\texttt{icmp}_s eq = 0 ttl = 62 time = 0.780 ms$
- `ssh login adr` : se connecter à distance à la machine ayant l'adresse *adr* et en tant que l'utilisateur identifié par `login`.

Environnement utilisateurs

- A la connexion d'un utilisateur le système exécute un ensemble de commandes pour configurer l'environnement de travail :
  1. Spécifier les commandes accessibles,
  2. Renommer certaines commandes (*définition des aliases*)
  3. Définir les droits par défaut
  4. Options de présentation
- Les commandes de configuration sont spécifiées dans différents fichiers : `.login`, `.profile` , `.bashrc`
- La configuration de l'environnement est décrite par un ensemble de variables d'environnement

Variables d'environnement

- Le nom d'une variable est une chaîne alpha-numérique avec la possibilité d'utiliser le caractère _.
- Le nom ne doit pas commencer par un chiffre
  - Noms valables : `var1, _var`
  - Noms invalides : `1var, -var`.
- `env` : affiche l'ensemble des variables définies dans un *shell*

Manipulation de variables

- Déclaration & initialisation
  - `nom_de_variable=valeur` : création ou affectation d'une variable.
  - Exemple : var1="R&T"
  - **pas d'espaces autour de =**
- Saisie clavier
  - `read nom_de_variable`
  - `read var1`
- Affichage :
  - `echo $nom_de_variable`
  - `echo $var1` affiche `R&T`
  - `echo var1` affiche `var1`
- Destruction : `unset nom_de_variable`
- Protection : `readonly nom_de_variable`

Portée d'une variable

- Par défaut, une variable est accessible dans le shell où elle est crée.
- export nom_de_variable permet au shell de propager la variable aux processus fils créés **après** l'exécution de la commande `export`.
- Exemple :
  - `var1=1`
  - `var2=100`
  - `export var1`
  - `bash /* exécution d'un shell fils*/`
  - `echo $var1 /* affiche 1 */`
  - `echo $var2 /* n'affiche rien */`

Types de variables

- Type par défaut : chaîne de caractères.
- La notation `$(($nom_de_variable))` permet de traiter une variable comme un entier
- Exemple :
  - `var1=4`
  - `var2=$((var1*2))`
  - `echo $var2 /* affiche 8*/`

Variables d'environnements : Exemples

- PATH : Liste de répertoires dans lesquels le shell chercher à localiser des exécutables.
  - Lorsqu'on tape une commande (externe) le shell exécute le *premier* exécutable ayant le nom de la commande demandée en explorant les répertoires listés dans la variable PATH.
  - la command which nom_commande retourne la référence absolue (si il existe) de la commande à exécuter.
- PS1 : la chaîne de prompte
- SHELL : le type d'interpréteur de commande utilisé (cash, csh, tsh, sh, . . . , etc.)
- USER : Le login de l'utilisateur courant.
- HOME: le répertoire de connexion de l'utilisateur courant.
- PWD: le répertoire courant.
- RANDOM: un entier aléatoire.

Substitutions

- $VAR : est le valeur de la variable VAR
- " $VAR ¨ : est la chaîne de caractères obtenue après avoir remplacé $VAR par sa valeur
- '$VAR' : est la chaîne littérale $VAR
- `$VAR` : est la chaîne qui résulte de l'exécution de la commande donnée dans $VAR

les aliases

- Une *aliase* permet de renommer une commande.
- Déclaration : `alias nom_alias= cmd`
- Suppression : `unalias nom_alias`
- Lister les aliases : `alias`

Programmation de scripts shell

- Pourquoi ? :
  - Enchainement complexe de commandes de base.
  - Automatiser des tâches d'administration.
- Avantages :
  - Code compact, rapide à écrire.
  - Peu d'interaction avec l'utilisateur.
  - Accès à toutes les commandes de base du système.
- Inconvénients :
  - Code compact, alors difficile à lire et comprendre.
  - Différences syntaxiques entre les différents shells.
- Les shells existant : **bash**, csh, zsh ksh, tcsh, . . . , etc.

Variables particulières

- `$i` : Le paramètre *i*. Le paramètre 0 étant le nom du script.
- `$#` : Le nombre d'arguments passés sur la ligne de commandes.
- `$*` : La liste d'arguments passés sur la ligne de commandes.
- `$$` : Numéro du processus exécutant la commande.

Exemple : Manipulation de paramètres

- `script param.bash`
  - `echo "Le programme $0 a $# paramètres"`
  - `echo "Le premier paramètre a la valeur $1"`
- L'apple `param.bash Hello there` affiche :
  - `Le programme param.bash a 2 paramètres`
  - `Le premier paramètre a la valeur Hello`

Modification de la liste de paramètres

- `set liste_param` : redéfinition de la liste de paramètres
- `shift` *n* : décaler la liste de paramètres de *n* paramètres à gauche
- Exemple :
  - set alpha beta gamma
  - shift 2
  - echo $1 /* affiche gamma */

Manipulation de chaînes de caractères

- `${#var}` : retourne la longueur de la chaîne valuer de la variable `var`
- Exemple :
  - `var="IUT de Villetaneuse"`
  - `echo ${#var}` /* affiche 19*/
- `${var:i:n}` : extraire n caractères à partir de la position i.
- `${var:i}` : supprimer les caractères de 0 à i.
  - `var="IUT de Villetaneuse"`
  - `echo ${var:4:2}` /* affiche de */
  - `echo ${var:7}` /* affiche Villetaneuse */

Manipulation de chaînes de caractères

- `${var#motif}` : Suppression de la plus courte occurrence du `motif` à gauche.
- `${var##motif}` : Suppression de la plus longue occurrence du `motif` à gauche.
- `${var%motif}` : Suppression de la plus courte occurrence du `motif` à droite.
- `${var%%motif}` : Suppression de la plus longue occurrence du `motif` à droite.
- `${var/motif/}` : Suppression de la première occurrence du `motif`.
- `${var//motif/}` : Suppression toutes les occurrences du `motif`.
- `${var/motif1/motif2/}` : Remplacer la première occurrence de `motif1` par `motif2`.
- `${var//motif1/motif2/}` : Remplacer toutes les occurrences de `motif1` par `motif2`.

Tests et expressions logiques

- Notations :
  - `test expr`
  - `[ expr ]`
- Opérateurs logiques :
  - Négation : `! expr`
  - ET : `expr1 -a expr2`
  - OU : `expr1 -o expr2`

Tests sur les chaînes de caractères

- Egalité : `s1 = s2`
- Différence : `s1 != s2`
- Chaîne vide : `-z s`
- Chaîne non vide : `-n s`

Tests sur les nombres

- `n1 -eq n2` : $=$
- `n1 -ne n2` : $\neq$
- `n1 -lt n2` : $<$
- `n1 -le n2` : $\leq$
- `n1 -gt n2` : $>$
- `n1 -ge n2` : $\geq$

Tests sur les fichiers

- -e f : existence du fichier
- -s f : existence d'un fichier non vide
- -f f : existence du fichier ordinaire
- -d dir : existence du répertoire
- -r : existence du droit de lecture
- -w f : existence du droit d'écriture
- -x f : existence du droit d'exécution

Structures de contrôle : instructions conditionnelle

```
if cond
then
...
fi
if cond
then
...
else ...
fi
```

Choix

```
case var in
case1) ...
;;
case2) ...
;;
*) ...
;;
esac
```

Boucle for

```
for var in liste
do
...
...
done
```

Boucle while

```
while expr
do
...
...
done
```

Fonction

```
toto() {
...
...
}
```