ECUE 3R-IN3
Systèmes d'exploitation

Présentation de l'ECUE
Année 2024-2025

Xavier HILAIRE
Département IT

Objectifs et organisation

- Introduction à Linux et la ligne de commande
  - Grands principes de fonctionnement du système
  - Le langage du shell
  - Écriture de scripts
- L'ECUE à vocation à vous préparer à l'unité E5 de cybersécurité, au cours de laquelle vous devrez utiliser de nombreuses lignes de commande
- Séquencement :
  - 4 cours seulement (de 2h chacun) pour 3 chapitres. Les cours sont entièrement distanciels, enregistrés, les enregistrements mis à disposition (depuis https://perso.esiee.fr/~hilairex/3R-IN3)
  - 3 TD de 2h chacun. Mise en pratique immédiate du cours sur des exercices simples. Peuvent être faits sur machine. Corrigés fournis.
  - 3 TP de 3h chacun. Travail plutôt en autonomie. Niveau plus élevé que les TD.

Objectifs et organisation

- Page de l'ECUE : https://perso.esiee.fr/~hilairex/3R-IN3
- Elle contient tous les documents et ressources dont vous aurez besoin
- Les corrigés des TD et TP y seront ajoutés au fil de l'eau, dès que le dernier groupe aura fini sa séance, sans avertissement par mail.

Post-assistance

- Je suis généralement disponible les jeudis, en début d'après-midis (16h dernière limite) à mon bureau (5352) pour toute question relative à l'ECUE
- Possibilité de session distancielle si rendez-vous convenu à l'avance par mail

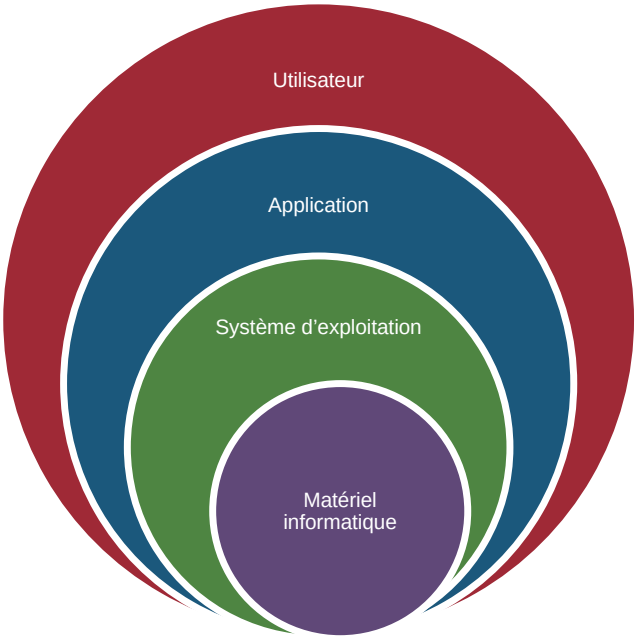
Introduction à Linux

Cours n°1

Systèmes d'exploitation – ESIEE Paris - 3R-IN3

Document originel : Clément BOIN
Adaptation : Xavier HILAIRE

Qu'est-ce qu'un système d'exploitation ?



Définition

- Système d'exploitation (SE)
  - Operating System (OS), en anglais
- Le système d'exploitation est un intermédiaire (interface) entre les applications et le matériel.
- Chaque SE fonctionne avec une gamme particulière de machines.

Typologie des systèmes d'exploitation (1/2)

- ***multi-tâches*** : permet l'exécution simultanée de plusieurs programmes.
  - Début dans les années 1960,
  - tous les systèmes d'exploitation contemporains le sont
- ***multi-utilisateurs*** *:* conçu pour être utilisé simultanément par plusieurs utilisateurs
  - Via un réseau le plus souvent
  - Utilisés pour des serveurs notamment
  - Sécurisation des connexions des utilisateurs

Typologie des systèmes d'exploitation (2/2)

- ***multi-processeurs*** *:* conçu pour exploiter un ordinateur équipé de plusieurs processeurs.
  - plusieurs programmes sont exécutés simultanément par les différents processeurs.
- ***temps réel*** : garantit que les opérations seront effectuées en respectant des délais très stricts, et ce quelles que soient les conditions d'utilisation.
  - Utilisés notamment dans l'industrie, l'aéronautique

Composition d'un SE (1/4)

- **Interface de programmation**
  - API (Application Programming Interface)
  - Mise à disposition de fonctions (dans des bibliothèques)
  - Norme POSIX (famille UNIX) - IEEE 1003
- **Ordonnanceur**
  - Contrôle de déroulement des programmes
  - Exécution simultanée (multi-tâche)
- **Communication inter-processus**
  - échanges d'informations entre les processus

Composition d'un SE (2/4)

- **Gestion de la mémoire**
  - allocation : réservation de la mémoire
  - protection : la mémoire est utilisé uniquement par le programme qu'il l'a réservé
  - pagination : décomposition en zones de taille fixe (pages)
- **Mémoire virtuelle (swap)**
  - Simuler la présence de mémoire centrale en utilisant un autre type de mémoire (un disque dur par exemple)
  - Exécuter plus de programmes que ce que la mémoire centrale peur contenir.
- **Pilotes**
  - contiennent les instructions permettant d'utiliser certains périphériques

Composition d'un SE (3/4)

- **Système de fichiers**
  - file system en anglais
  - structure arborescente dans laquelle sont stockés les données (fichiers)
  - Exemples de dispositions : ext2fs, ext4fs (Linux), NTFS, FAT32 (Windows), HFS (Mac)
- **Réseau**
  - niveaux 1 à 4 du modèle OSI pris en charge par le SE
- **Contrôle d’accès**
  - identification des utilisateurs
  - droits d'accès sur les fichiers

Composition d'un SE (4/4)

- **Interface utilisateur (graphique ou texte)**
  - permet à l'utilisateur de dialoguer avec la machine via une image envoyée au matériel
  - Famille UNIX : X Window System
- **Logiciels utilitaires**
  - fournis avec un SE
  - applications
  - interpréteur de commande (scripting)
  - environnement de bureau
  - installation et mise à jour des logiciels
  - surveillance de l'utilisation
  - configuration, droits d'accès
  - ...

Le noyau d'un OS

- Espace mémoire protégé

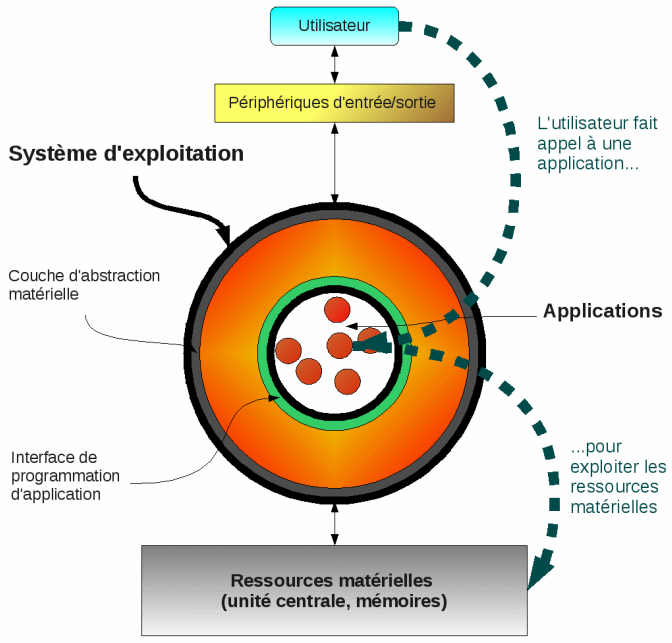
  +
- Ensemble de programmes qui forment la base minimale de l'OS
- Tout ce qui n'est pas un appel système fonctionnera dans l'espace utilisateur
- Des choix de conception sont fait :
  - Exemple: L'interface graphique
    - Sous Windows : intégré dans le système
    - Sous Linux : programme utilisateur (serveur X)

Les types de noyau

- Monolithique:
  - tout est dans le noyau (système de fichiers, pilotes, etc)
  - *Exemples : Linux, FreeBSD*
- Micro-noyau:
  - seulement le strict minimum (ordonnanceur+mémoire virtuelle)
  - *Exemples : Minix, Mac OS X*
- Hybride
  - *Windows NT (noyau de Windows 10)*
- Exo-noyau: rien n'est protégé

Les relations dans un SE moderne

(source: Wikipedia)



Les relations entre utilisateur, applications, système d'exploitation et matériel

Les grandes familles de systèmes d'exploitations

Famille UNIX

- Distributions GNU/Linux
  - **Debian**
  - Ubuntu Server
  - Red Hat Enterprise Linux
  - CentOS
  - Fedora
  - ...

- Android

- Mac OS X / iOS

- FreeBSD, NetBSD, OpenBSD
  - Unix "libre"

Famille Windows

- Windows 10
- Windows Server
  - Version actuelle : 2022

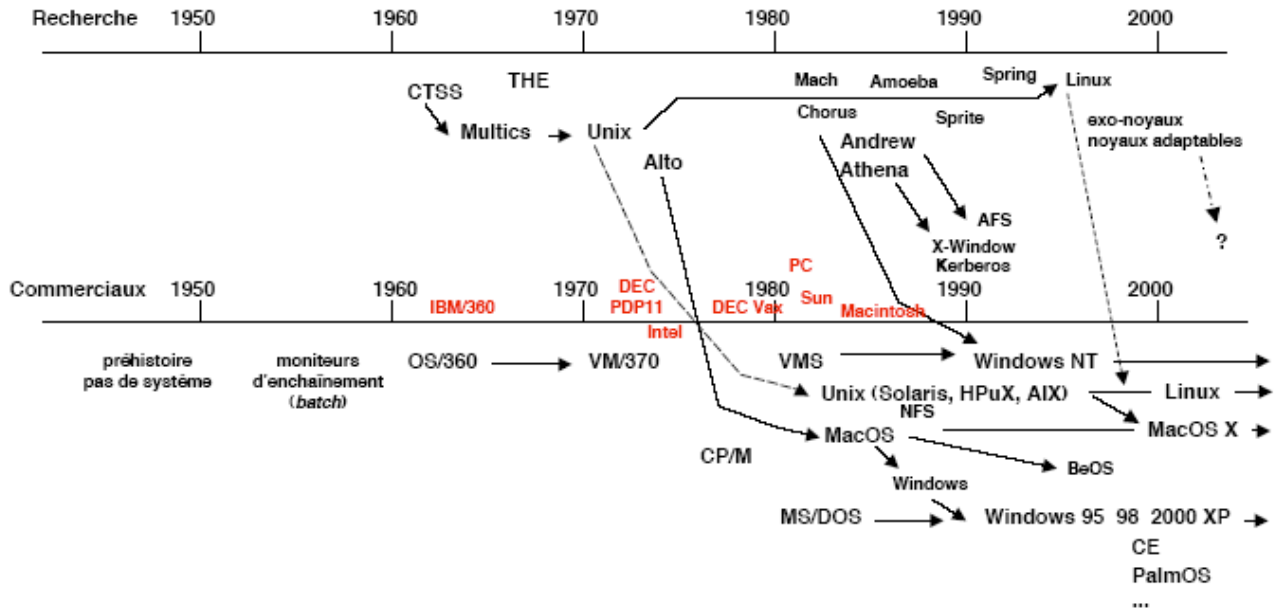
Principaux systèmes d'exploitations

(source: Wikipedia)

Apple	Mac OS Classic	Système 5 · Système 6 · Système 7 · Mac OS 8 · Mac OS 9
	Dérivés de NeXTSTEP	NeXTSTEP · Rhapsody · Darwin · OS X · iOS
Dérivés de BeOS		BlueEyedOS · Haiku · ZETA
DOS		DR-DOS · FreeDOS · MS-DOS · PC-DOS
IBM		AIX · MVS · OS/2 · OS/360 · OS/390 · z/OS · OS/400
Microsoft Windows	Basés sur MS-DOS	1.0 · 2.x · 3.x · 95 · 98 · Me
	Branche NT	NT 3.x · NT 4.0 · 2000 · XP · 2003 · Vista · 2008 · 7 · 2012 · 8 · 10
POSIX / Unix	BSD	FreeBSD · PC-BSD · GhostBSD · DragonFly BSD · FreeNAS · OpenBSD · NetBSD
	GNU/Hurd	Debian GNU/Hurd · Arch Hurd
	GNU/Linux (liste)	Arch · CentOS · Debian · Fedora · Mageia · openSUSE · PCLinuxOS · Puppy · Red Hat · Slackware · Ubuntu
	Autres dérivés	AIX · HP-UX · IRIX · LynxOS · Minix · QNX · Oracle Solaris · System V · Tru64 · UnixWare · ChorusOS · UNICOS · Raspbian
Dérivés d'AmigaOS		MorphOS · AROS
D'importance historique		CP/M · CTSS · GCOS · Genera · ITS · Multics · Plan 9 · QDOS · RSTS · TENEX · TOPS-20 · TOS · VMS
Mobile		Android · Bada · BlackBerry OS · Firefox OS · Open webOS · iOS · Sailfish OS · Tizen · Ubuntu Touch · Windows Phone
Embarqués	pour capteur en réseau	Contiki · TinyOS
	pour carte à puce	Java Card · Multos
	Temps réel	eCos · FreeRTOS · Linux embarqué · LynxOS · MenuetOS · OS-9 · PikeOS · QNX · RTEMS · RTLinux · RTX · µC/OS-II · VxWorks
Autres systèmes		eyeOS · IOS (Cisco) · Inferno · MenuetOS · ReactOS

Historique sommaire et ancien !

Les débuts des SE



Installation d'un système

Quel système d'exploitation choisir ?

- Qui sont les utilisateurs ?
- Coûts directs (et indirects)
- Matériel (compatibilité)
- Fonctionnalités/Logiciels
- Objectifs à atteindre
- Logiciel libre ou propriétaire

Installation d'un système

Quelle distribution choisir ?

- De nombreuses distributions Linux
  - https://fr.wikipedia.org/wiki/Liste_des_distributions_GNU/Linux
- Qualité de la documentation en ligne
  - Communauté active
- Fréquence des mises à jours
- Debian ?

Installation d'un système

Méthode d'installation

- Quel support d'installation?
  - CD/DVD
  - Installation par internet avec une image minimale (netinst)
  - Clé USB
  - Live CD ou Live USB
  - PXE (par le réseau)
- Partitionnement des disques
  - Partitionnement classique (avec limitations)
  - LVM (Logical Volume Manager) : plus souple
  - RAID : sécurité des données
- Multi-boot avec d'autres systèmes
  - MBR/UEFI
- Choix des paquets
  - Choix de l'environnement de bureau (GNOME, LXDE, KDE, Cinnamon...) ?
- Configurer le réseau et les services

Linux

GNU/Linux pour les puristes

Présentation de Linux

- Linux, appellation courante du système d'exploitation libre GNU/Linux,
  - implémentation libre d'un plus vieux système d’exploitation → UNIX
- Rencontre entre le mouvement du logiciel libre et le modèle de développement collaboratif et décentralisé via Internet.
- Alternative aux systèmes propriétaires Microsoft Windows et Apple Mac OS.
- Autres alternatives libres: FreeBSD, NetBSD et OpenBSD.

Applications disponibles

- Tout type d'applications libres
- Certains logiciels propriétaires fonctionnent également sous Linux.
- Possibilité de faire fonctionner des applications Windows (programme wine)

Historique de Linux

- **1969** Naissance du SE Unix dans les laboratoires Bell (AT&T).
- **1974** Diffusion de Unix dans les universités américaines.
- **1983** Versions commerciales d'Unix.

- **1991** Un étudiant finlandais, **Linus Torvalds**, écrit son propre noyau de système d'exploitation pour mieux comprendre le fonctionnement de son ordinateur.
  Il se base sur Unix mais réécrit tout depuis le début, publie son travail sur Internet (Licence GNU GPL en 1992)
- **1993** Première version de la distribution Debian.
- **1996** Le manchot *Tux* est utilisé comme symbole pour Linux.
  Lancement de l'environnement graphique KDE.
- **1997** Lancement de l'interface graphique Gnome.
- **1998** Début de la commercialisation massive de distributions payantes et de services payants autour de Linux.
- **2004**: Début du projet Ubuntu (basé sur Debian) par le millionaire Sud-africain Mark Shuttleworth (société Canonical).
- **2009** : Microsoft contribue au noyau Linux

Particularités/ Noyau

- Linux est un système d'exploitation :
  - Multi-tâches exécute plusieurs programmes à la fois.
  - Multi-utilisateurs plusieurs utilisateurs peuvent être actifs sur une même machine en même temps.
  - Multi-plateformes

- Versions du noyau Linux : Un numéro de version de Linux est noté ainsi : x.y.z
  - x → version de kernel
  - y → révision majeur
  - z → révison mineur

- Noyau développé par Linus Torvald (et la communauté) depuis 1991

- *Dernière version: version 5.8.10 (17 septembre 2020)*
- *Version utilisée par Debian 10.5 : 4.19.0 (octobre 2018)*
- Pour connaitre la version utilisée: **uname -r**

Notion de distribution

- Une distribution Linux est un ensemble cohérent de logiciels formant un système d'exploitation composé :
  - d'un noyau Linux
  - et d'applications diverses destinées aux utilisateurs (navigateur, suite bureautique, lecteur vidéo...).

- Une distribution peut aussi être associée à des prestations de service (parfois payantes) et de la documentation.

- Distributions grand public:
  - **Ubuntu** (commerciale orientée grand public, dynamique)
  - Fedora (logiciels fréquemment mis à jours)
  - openSUSE (grand public et professionnels)
  - Mageia (issue de Mandriva, editée par un association française)
  - Linux mint (grand public)

- Distributions larges
  - **Debian** (très utilisée sur les serveurs)
  - ArchLinux (dernières versions des logiciels)
  - Gentoo (pour les experts)
  - RedHat (distribution commerciale)
  - Slackware (la plus ancienne)

Le système d'exploitation Debian

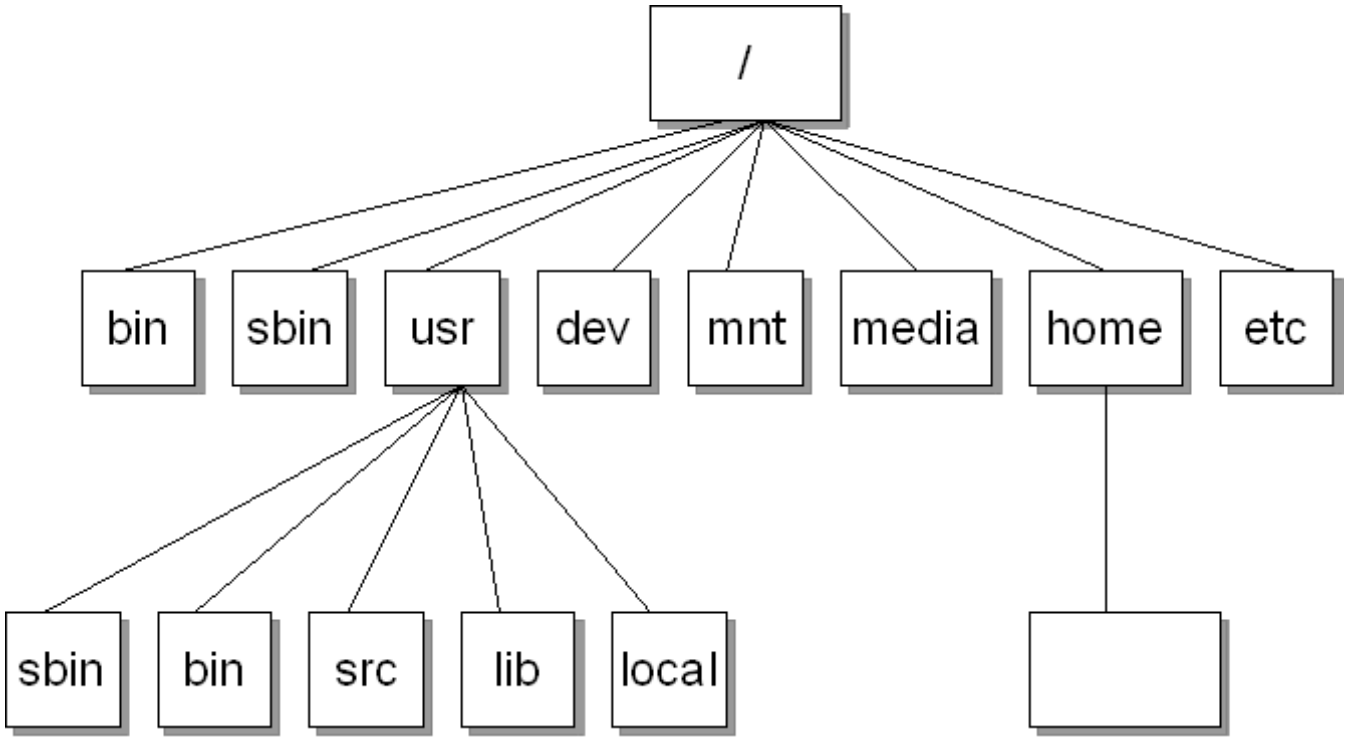
Généralités sur Debian

- Système d'exploitation "universel"
  - depuis 1997
- Disponible sur de nombreuses architectures
- Noyau Linux
  - Mais des projets avec d'autres noyaux (Hurd, BSD)
- Entièrement composé de logiciels libres par défaut
- Version utilisée à l'ESIEE en 2023 : 11 (Bullseyes)
- La commande **lsb_release -a** donne la version de la distribution.

Le système de fichiers

- Utilise ext4fs comme système de fichier
  - Extended file system
  - Depuis 2008, successeur de ext3
- Système de fichiers journalisé
  - Enregistrement des modifications dans un journal avant de les faire sur les fichiers.
  - Possibilité de récupération
  - *Autre exemple de SF journalisé : NTFS*
  - *Non journalisé : FAT32, exFAT*

Arborescence du système de fichiers



Arborescence du système de fichiers

Répertoire	description
/	Répertoire "racine", point d'entrée du système de fichiers
/boot	Répertoire contenant le noyau Linux et l'amorceur
/bin	Répertoire contenant les exécutables de base, comme par exemple cp, mv, ls, etc.
/dev	Répertoire contenant des fichiers spéciaux nommés *devices* qui permettent le lien avec les périphériques de la machine
/etc	Répertoire contenant les fichiers de configuration du système
/home	Répertoire contenant les fichiers personnels des utilisateurs (un sous-répertoire par utilisateur)
/lib	Répertoire contenant les librairies et les modules du noyau (/lib/modules)
/media	Répertoire contenant les « points de montage » des médias usuels : CD, DVD, disquette, clef USB
/root	Répertoire personnel de l'administrateur
/sbin	Répertoire contenant les exécutables destinés à l'administration du système
/tmp	Répertoire contenant des fichiers temporaires utilisés par certains programmes
/usr	Répertoire contenant les exécutables des programmes (`/usr/bin` et `/usr/sbin`), la documentation (`/usr/doc`), et les programmes pour le serveur graphique (`/usr/X11R6`).
/var	Répertoire contenant les fichiers qui servent à la maintenance du système (les fichiers de journaux notamment dans `/var/log`)

Le shell

- Donne une interface en ligne de commande
- Disponible même sur un système minimal
- De nombreuses variantes et versions :
  - UNIX : sh, csh, tcsh, **bash** ,**dash**...
  - Windows : cmd, powershell
- *bash* est le shell interactif par défaut pour les utilisateurs Debian
  - *dash* est comme interpréteur de commande pour exécuter les scripts (plus rapide)

Les commandes de gestion des fichiers

- Le dossier $HOME
- cd : changer de répertoire courant
- pwd : afficher le répertoire courant
- ls : lister les fichiers
  - caractères joker * et ?
  - options -aliR
- Chemin relatif et absolu
- Dossiers biens connus : / . .. ~
- Fichiers cachés
- cp : copier des fichiers
  - cp -R pour un répertoire
- mkdir : créer des répertoires
- rm : supprimer
  - rmdir
  - rm -r (répertoires)
  - rm -i (demande de validation)
- mv : renommer
- ln et ln -s : créer un lien

Autres commandes

- cat file , more file , less file : voir le contenu
- vi file , nano file , emacs file : éditer les fichiers textes
  - vi toujours disponible mais nécessite un apprentissage :q
- file *file* : connaitre le type du fichier
- man : aide sur la commande
- locate : chercher un fichier
- Flèche du haut et du bas : historique
  - commande : history

La gestion des paquets
Commande apt

- La commande apt (apt-get / aptitude)
  - `apt install nom_du_paquet`
  - `apt purge nom_du_paquet`
  - `apt remove nom_du_paquet`
  - `update` : mise à jour des dépôts
  - `upgrade` : mise à jour des paquets
  - `dist-upgrade` : mise à jour des paquets et des dépendances

Les dépôts

- **main** -> paquets libres
- **non-free** -> paquets non libres
- **contrib** -> paquets libres mais ayant des dépendances en dehors de main
- **backports** -> dernières versions encore en développement
- Fichier contenant les sources des paquets : `/etc/apt/sources.list`
  - dépôts distants
  - éventue

```
deb http://deb.debian.org/debian/ buster main
deb-src http://deb.debian.org/debian/ buster main

deb http://security.debian.org/debian-security buster/updates main
deb-src http://security.debian.org/debian-security buster/updates main

# buster-updates, previously known as 'volatile'
deb http://deb.debian.org/debian/ buster-updates main
deb-src http://deb.debian.org/debian/ buster-updates main
```

Manipulation de texte

Une grande partie des commandes shell consiste en de la manipulation basique de chaîne de caractères.

- **`echo`** : Afficher un message
- **`less, more`** : afficher le contenu d'un fichier texte
- **`wc`** : statistique et comptage
- **`cut`** : découpage et extraction
- **`grep`** : recherche dans une chaîne
- **`sort`** : tri
- **`uniq`** : suppression des doublons consécutifs
- **`tee`**: sauvegarder un log dans un fichier
- **`sed et awk`** : multi-fonctions (plein)
- **`cat`** : concaténation de fichiers
- Et bien d'autres : `expr, head, tail, join, uniq, tr, nl, strings`…

Connaître le manuel

Utilisation du manuel incontournable !

- `man man` : Le manuel de man
- `man <commande>` : Le manuel de la commande
- `man -k <mot clé>` : Chercher un mot clé dans tous les manuels
- `man -f <commande>` : Manuel simplifié (équivalent à `whatis`)

Gestion des utilisateurs et des droits

L'organisation des utilisateurs sous linux

- Système multi-utilisateur.
- Tout le monde ne peut pas tout faire :
  - Excepté l’administrateur (root)
  - Faire partie des « sudoers » (groupe sudo) permet a un utilisateur de lancer des commandes en tant qu’administrateur
- Il faut donc donner des privilèges à certains utilisateurs et en priver d’autres.

L'organisation des utilisateurs sous linux

- Un utilisateur n'est rien d'autre qu'un entier appelé UID (User ID)
- Un groupe est aussi un entier GID (Group ID)
- Tout utilisateur appartient au moins à un groupe, appelé groupe primaire
- Il peut aussi appartenir à d'autres groupes, dits groupes secondaires
- Possibilité de basculer d'un groupe secondaire en lançant un nouveau shell grâce à la commande newgrp

L'organisation des utilisateurs sous linux

- Dès ses débuts, UNIX stockait les informations des utilisateurs dans le fichier `/etc/passwd`
- Et celles concernant les groupes dans `/etc/group`
- Solution par défaut toujours utilisée !
- Il existe aussi des fichiers `/etc/shadow` et `/etc/gshadow` qui reprennent ces mêmes informations et les complètent par les mots de passe chiffrés
- A la différence de `/etc/passwd` et `/etc/group`, ils ne sont lisibles par personne
- Solution par défaut toujours utilisée !
- Mais il en existe d'autres, notamment via PAM (Pluggable Authentication Modules, permet l'authentification LDAP, Kerberos, SSH, ...

Extrait de fichier /etc/passwd

```
sshd:x:74:74:Privilege-separated
SSH:/usr/share/empty.sshd:/sbin/nologin
chrony:x:994:992::/var/lib/chrony:/sbin/nologin
dnsmasq:x:993:991:Dnsmasq DHCP and DNS
server:/var/lib/dnsmasq:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
systemd-coredump:x:989:989:systemd Core
Dumper:/:/usr/sbin/nologin
systemd-timesync:x:988:988:systemd Time
Synchronization:/:/usr/sbin/nologin
hilairex:x:1000:1000:Xavier HILAIRE:/home/hilairex:/bin/bash
```

login

Mot de passe

UID

GID

groupe primaire

Nom complet

Répertoire “home”

Shell par défaut

Extrait de fichier /etc/group

```
wheel:x:10:hilairex
gluster:x:973:
qemu:x:107:hilairex
saslauth:x:76:
libvirt:x:972:
postgres:x:26:
hilairex:x:1000:
```

L’utilisateur hilairex est aussi membre du groupe wheel (utilisateurs avec privilèges)

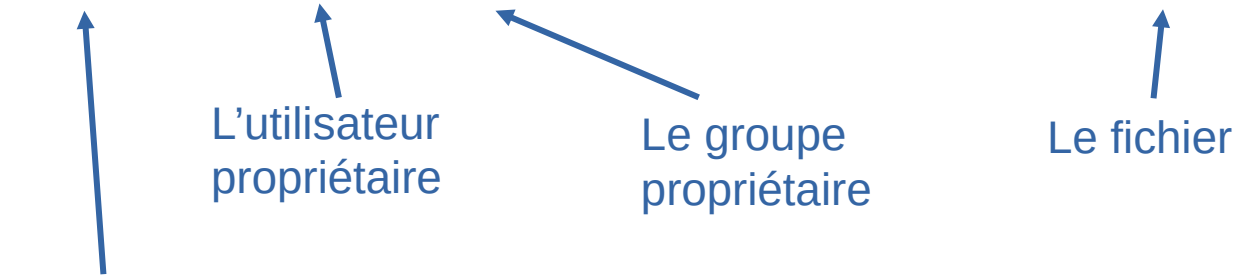
L’utilisateur hilairex est aussi membre de qemu

Le groupe hilairex a pour GID 1000. Ne déclare pas que l’utilisateur hilairex est dans ce groupe (fait indépendamment dans /etc/passwd, cf. diapo précédente; ne pas confondre les deux)

Les droits

- Tout fichier possède obligatoirement un propriétaire (UID) , un groupe (GID), et des droits dès sa création :

```
[hilairex@pc5352a ~]$ ls -l visu*
-rw-r-xr--. 1 hilairex vboxusers 55039 30 nov.  18:26 visupca.sh
```



Les droits :
- utilisateur (U) = rw- (lecture r + écriture w)
- groupe (G) = r-x (lecture r, et exécution)
- autres (O) = r-- (lecture r)

Les droits

- Un seul des ensembles de droits UGO (User, Group, Others) est utilisé pour déterminer les droits d'accès au fichier
- Le système les examine dans l'ordre UGO et une seule fois :
  - S'il y a concordance entre l'UID du fichier et l'UID du programme qui demande l'accès → on applique les droits U
  - Sinon, s'il y a concordance de GID → on applique les droits G
  - Sinon, on applique les droits O
- Les droits sont non cumulatifs. Dans l'exemple précédent, même si hilairex est propriétaire du fichier et appartient à vboxusers, il n'aura pas le droit d'exécution. A l'inverse quelqu'un qui est du group vboxusers sans être hilairex aura ce droit.

Autre exemple de droits

		droits concernant le propriétaire du fichier			droits concernant les autres membres du même groupe que le groupe propriétaire du fichier			droits concernant tous les autres utilisateurs		
$ ls -l	type de fichier	u(ser)			g(roup)			o(ther)		
-rwxrw-r-- unfichier	-	r	w	x	r	w	-	r	-	-
	indique un fichier ordinaire	Le propriétaire peut lire, écrire et exécuter ce fichier			Les membres du même groupe propriétaire peuvent lire et écrire, mais pas exécuter ce fichier			Les autres utilisateurs peuvent seulement lire ce fichier		

Les bits spéciaux

- En plus des 3 ensembles de bits UGO, il existe un ensemble supplémentaire de bits spéciaux :
  - SetUID : utilisé avec le droit d'exécution, aura pour effet de lancer l'exécutable sous l'UID de son propriétaire
  - SetGID : même chose avec le GID du propriétaire
  - Sticky : utilisé avec un répertoire sur lequel le droit d'écriture est accordé à tout le monde, interdit les utilisateurs non propriétaires d'un fichier de supprimer ce fichier.

Les bits spéciaux : SetUID + SetGID

- Exemple d'utilisation de SetUID : la commande passwd → utilisée pour changer de mot de passe
- Doit modifier le fichier `/etc/passwd`
- Mais l'utilisateur n'a pas de droit d'écriture sur ce fichier (heureusement !)

```
[hilairex@pc5352a ~]$ ls -l /etc/passwd
-rw-r--r--. 1 root root 2889 15 févr. 17:44
/etc/passwd
[hilairex@pc5352a ~]$
```

Les bits spéciaux : SetUID + SetGID

- Solution : root, propriétaire de /usr/bin/passwd, positionne le bit SetUID :

```
[hilairex@pc5352a ~]$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 32744 21 janv.  2022
/usr/bin/passwd
[hilairex@pc5352a ~]$
```

- La commande passwd s'exécutera en tant que root et pourra ainsi modifier /etc/passwd

Les bits spéciaux : sticky

- Exemple typique d'utilisation de sticky : /tmp

```
[hilairex@pc5352a ~]$ ls -l / | grep tmp
drwxrwxrwt.  26 root root  880  3 avril 11:47 tmp
[hilairex@pc5352a ~]$
```

- Tout le monde peut créer ou supprimer des fichiers dans ce répertoire
- Mais personne ne peut supprimer les fichiers des autres

Les droits sur les fichiers

- Pour voir les droits affectés à un fichier on peut utilise les commandes ls –l ou stat
- chmod : changer les droits d'un fichier
- Deux façons d'exprimer les droits : littérale et octale
- Littérale : rwsr_xr_
  - r : lecture
  - w : écriture
  - x : exécution (pour un fichier) ou accès (pour un répertoire) = 1
  - s et S : SetUID et SetGID
  - t : sticky
  - Exemple (ci-dessus) : u=rws, g=rx, o=r

Les droits sur les fichiers

- Octale : rwsr_xr__
  - r=4, w=2, x= 1 → s'applique pour chacun des ensembles de bits de UGO, et donne 3 chiffres en base octale
  - Setuid= 4, setgid=2, sticky=1 → forme un chiffre supplémentaire qui doit être placé devant les chiffres UGO
- Exemple : rwsr-xr-- = 4754 car :
  - SetUID positionné → 4 frontal
  - rwx = 4+2+1 = 7 ; r-x = 4+1 = 5 ; r-- = 4
- Autre exemple : rwxrwxrwt = 1777 car :
  - Sticky positionné → 1 frontal
  - rwx = 7, trois fois de suite

Commandes utilisateur

- who : qui est connecté à la machine
- passwd : changement de mot de passe
- adduser, useradd, deluser et userdel
  - En mode administrateur
  - adduser et deluser sont des versions améliorées de useradd et userdel

Gestion des utilisateurs et des groupes

- addgroup, usermod : gestion des groupes
- chown, chgrp : Gestion des propriétaires d'un fichier
- Les utilisateurs et les groupes du système sont stockés dans le fichier /etc/group
- Les informations sur le dossier home de l'utilisateur et son shell par défaut sont dans /etc/passwd

Le shell
Principales commandes

Xavier HILAIRE
x.hilaire@esiee.fr

Qu’est-ce que le shell ?

- Le shell est un programme exécutable, dont la principale fonction est de permettre à l’utilisateur d’interagir avec le système via un terminal.
- Il est aussi appelé *interpréteur de commandes*.
- Deux modes d’utilisation :
  - Interactif : l’utilisateur saisit et exécute ses lignes de commandes une par une dans un terminal ;
  - Non interactif : le shell lit un ensemble de commandes à partir d’un fichier appelé *script*.
- Il existe aujourd’hui plus d’une trentaine de shells différents, mais deux grandes familles dominent :
  - Les csh : shells orientés administration, avec une syntaxe inspirée du langage C

Qu’est-ce que le shell ?

- La famille des sh (à l’origine: ash), bsh (Bourne shell), bash (Bourne again shell) : shells orientés utilisateur, majoritaires aujourd’hui. La plupart des scripts shell sont écrits en sh, ou au moins compatibles sh.
- Le shell UNIX standard est sh.
- Mais Bash supplante de plus en plus souvent sh (c’est le cas sur Linux) :
  - Il consiste en un mélange de sh, de quelques fonctions du csh, et d’autres du Korn shell (ksh)
  - Il est 100% compatible sh.

Principe d'exécution

Pour simplifier, le shell adopte toujours le même plan d'exécution :

1) Lit une **ligne de commande** soit à partir du terminal, soit à partir d'un fichier script
2) Effectue une première analyse qui détermine quels en sont les **opérateurs**, et quels en sont les **mots**
3) Décompose la ligne de commande en **commandes simples**, en appliquant un jeu de priorités fixé par les opérateurs identifiés
4) Pour chaque commande simple :
   - Réécrit les mots de cette commande lorsque c'est possible
   - Lance la commande en question

Grammaire

- Les mots sont soit des suites ininterrompues de caractères, soit des chaînes de caractères entourées de quotes simples ' ' ou doubles " "
- Les quotes simples interdisent de modifier le contenu qu'elles délimitent
- Les quotes doubles l'autorisent, mais seulement pour le métacaractère $
- Deux types d'opérateurs :
  - ***Contrôle*** : ils servent à séparer deux commandes ou deux listes. Ce sont: `&` `&&` `( )` `{}` `;` `;;` `|` `||`
  - ***Redirection*** : ils servent à rediriger les entrées/sorties et portent sur une seule commande. Ce sont: `<` `>` `>|` `<<` `>>` `<&` `>&` `<<-` `<>`

Grammaire

Une ligne de commande est :

- **simple** si elle ne comporte aucun opérateur de contrôle ;
- **composée** dans le cas contraire.

Priorités des opérateurs de contrôle :

Priorités égales (*)

( )
{ }
|
&& ||
& ;
;;

Priorités décroissantes (priment sur les priorités horizonales)

(*) L'opérateur rencontré le premier de gauche à droite remporte la priorité

Grammaire

Exemple : analyse de la ligne de commande

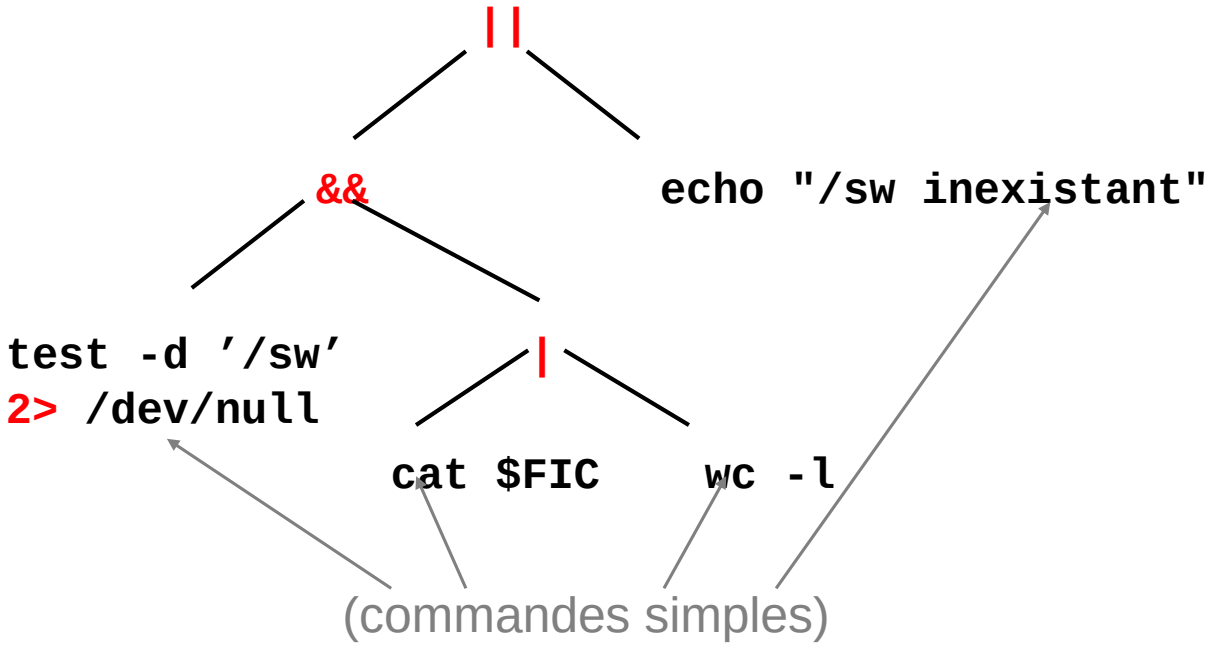
```
test -d   '/sw'  2>/dev/null && cat $FIC |
wc -l || echo  "/sw inexistant"
```

Etapes 2 et 3 : distinguer les mots des opérateurs.
On obtient :

```
test -d '/sw' 2> /dev/null && cat $FIC | wc
-l || echo  "/sw inexistant"
```

Grammaire

Etape 4 : résultat de l'analyse



Commandes simples

Une commande simple a la forme suivante (les [ ] marquent le caractère optionnel):

[env] [com] [args…] [redir]

- **env** consiste en aucune, une ou plusieurs affectations de *variables d'environnement* portant sur la commande qui suit (et seulement elle)
- **com** est la commande elle-même. Il s'agit soit d'un nom de commande interne, soit d'un nom de fichier exécutable accessible par la *variable d'environnement* PATH
- **args** sont les arguments optionnels de la commande
- **redir** consiste en aucune, une, ou plusieurs redirections de fichiers

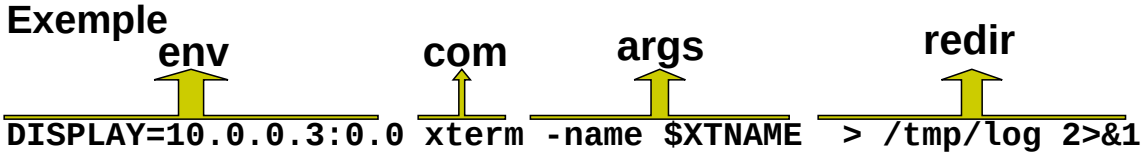
Commandes simples

Evaluation d'une commande simple : ce qui se passe (dans l'ordre)

1. Chaque mot de la commande est **évalué**, c'est à dire soit laissé tel quel s'il est interdit de l'interpréter (quotes simples), soit réécrit sur la ligne de commande dans le cas contraire (quotes doubles ou pas de quotes)
2. Les redirections **[redir]** sont appliquées, puis retirées de la ligne de commande
3. Le bloc **[env]** est évalué, appliqué, puis retiré de la ligne de commande.
4. La commande **[com]** est lancée, si elle existe (*oui, il est possible d'avoir un bloc [com] vide!*)

Commandes simples

Exemple



1. Variables d'environnement : une seule ici , `DISPLAY=10.0.0.3:0.0` qui est exécutée, puis retirée. Il reste :

`xterm -name $XTNAME > /tmp/log 2>&1`

2. Résolution : une seule substitution à faire ici, `$XTNAME` qui est une variable. En supposant que XTNAME=myterm, il reste :

`xterm -name myterm > /tmp/log 2>&1`

3. Redirections : `> /tmp/log` indique que la sortie standard doit être redirigée vers le fichier `/tmp/log`, `2>&1` que la sortie d'erreur standard et la sortie standard doivent être unifiées. Le shell créé /tmp/log, fait la redirection, puis retire la fin de ligne. Il reste :

`xterm -name myterm`

4. Le programme xterm est lancé, avec deux arguments seulement :

`xterm -name myterm`

Les variables

- Le shell connaît deux types de variables : **ordinaires** et **d'environnement**.

- Une **variable ordinaire** n'est connue que par le shell : aucun des programmes que le shell lancera ne pourra la consulter/modifier.

- A l'inverse, une **variable d'environnement** est une variable ordinaire dont les processus lancés par le shell recoivent une copie.

- On peut affecter une valeur à une variable avec l'opérateur '=' (attention: **pas d'espace !**)

```
$ mavariable=3
$ echo $mavariable
3
```

- Pour faire d'une variable ordinaire une variable d'environnement, on utilise la commande **export** :

```
$ sh -c 'echo $mavariable' # lance la com. dans un sous-shell
$ export mavariable
$ sh -c 'echo $mavariable'
3
```


Evaluation

- C'est une technique de réécriture très puissante durant laquelle les variables, les noms de fichiers, et certains caractères sont substitués dans chaque mot qui les contient soit par leur propre valeur, soit par autre chose.

- Cela n'est toutefois pas toujours possible :

  - Si le mot concerné est entouré par des quotes simples : ' ' le shell n'est pas autorisé à interpréter quoi que ce soit, il laissera le mot tel quel, mais retirera les quotes délimitantes
  - Si le mot est entouré par des quotes doubles " " : le shell est autorisé à évaluer l'opérateur $, à substituer le résultat sur la ligne de commande, et à retirer les quotes délimitantes
  - Si le mot n'est entouré par rien, le shell est autorisé à évaluer l'opérateur $ et les noms de fichiers, et à substituer le résultat sur la ligne de commande

Evaluation

Exercice 1

- Nous allons vérifier ce qui vient d'être dit grâce à une première manipulation.
- Ouvrir un terminal, puis taper la commande suivante dedans :

```
curl -o args1.c
   https://perso.esiee.fr/~hilairex/3R-IN3/args1.c
```

- Examiner le programme args1.c : il se contente d'afficher les arguments qu'il a reçus
- Compiler ce programme :

```
gcc -o args1 args1.c
```

- Exécuter et expliquer le résultat des commandes suivantes :
- `./args1`
- `echo $PWD`
- `$PWD/args1`

Evaluation

Exercice 1 (suite)

- `./args1 un    deux      trois  $PWD`
- `./args1 "un    deux      trois" $PWD`
- `./args1 'un    deux      trois' $PWD`
- `salut="Bonjour  tout le  monde !!"`
- `./args1 $salut`
- `./args1 "$salut"`
- `./args1 '$salut'`
- `./args1 "/???"`
- `./args1 '/???'`
- `./args1 /???`

Evaluation

Exercice 2

- Nous allons à présent vérifier l'effet des variables d'environnement sur les programmes
- Ouvrir un terminal, et taper la commande suivante dedans :

```
curl -o args2 https://perso.esiee.fr/~hilairex/3R-IN3/args2.c
```

- Ce deuxième programme fait la même chose que le premier, mais affiche en plus les variables d'environnement qu'il a reçues dans son envp
- Examiner et expliquer l'effet des commandes suivantes (dans l'ordre indiqué) :
  - `./args2 un deux`
  - `ZZZZZZ=essai ./args2 un deux`
  - `ZZZZZZ=essai`
  - `./args2 un deux`
  - `export ZZZZZZ`
  - `./args2 un deux`
  - `man ls`
  - `LANG=C man ls`

Evaluation des variables

Evaluation des variables

- Il est possible de faire remplacer une variable par autre chose que son contenu.
- Le tableau ci-dessous donne un aperçu de quelques substitutions possibles.

Forme	var définie	var indéfinie
${var} ou $var	Substitue var	Substitue la chaîne vide
${var:-mot}	Substitue var	Substitue mot
${var:+mot}	Substitue mot	Substitue la chaîne vide
${var:?[mot]}	Substitue var	Erreur + substitue mot si spécifié
${var:=mot}	Substitue var	Affecte mot à var; substitue mot

Evaluation des variables

Forme	Substitution
${#var}	Longueur de la chaîne var
${var%motif}	var privée de la plus courte occurrence droite de motif
${var%%motif}	var privée de la plus longue occurrence droite de motif
${var#motif}	var privée de la plus courte occurrence gauche de motif
${var##motif}	var privée de la plus longue occurrence gauche de motif

```
$ var=aaabccc
$ echo $var
aaabccc
$ echo ${va:-mot}
mot
$ echo ${var:-mot}
aaabccc
$ echo ${var:+mot}
mot
$ echo ${va:?test}
```

```
va: test
$ echo ${va:?}
va: parameter null or
not set
$ echo ${va:=mot}
mot
$ echo $va
mot
$ echo ${#var}
7
```

```
$ echo ${var%e}
aaabccc
$ echo ${var%c}
aaabcc
$ echo ${var%%c}
aaabcc
$ echo ${var%c*}
aaabcc
$ echo ${var%%c*}
aaab
```


Evaluation des variables

Variables spéciales (utilisables seulement dans un script) :

Nom	Substitué par
*	L'ensemble des paramètres concaténés en une seule chaîne
@	L'ensemble des paramètres en autant de chaînes que nécessaire
#	Nombre de paramètres passés
-	Option courante
0,1,2…	Chaque paramètre, y compris le nom de la commande (0)

Variables spéciales toujours utilisables :

Nom	Substitué par
$	Numéro de processus (PID) du shell courant
!	PID de la dernière commande d'arrière-plan terminée
?	Code de retour du dernier processus terminé (exit de la dernière commande)

Evaluation arithmétique

Evaluation arithmétique entière

Forme générale: $((expression)) où expression est une expression arithmétique entière

```
$ echo $((3+1))
4
$ echo $((3+2*6))
15
$ echo $((3-(1+1)*6))
-9
$ i=2
$ i=$((i+1))
$ echo $i
3
$ echo $((3+(1+1.5)*6))
sh: 3+(1+1.5)*6: missing `)' (error token is ".5)*6")
```


Evaluation des noms de fichiers

Evaluation des noms de fichiers

- Lorsqu'il rencontre les métacaractères '*', '?', '~', '[', et ']' dans un mot non protégé (pas de quotes), le shell cherche à les remplacer pour les faire concorder avec des noms de fichiers existants sur disque
- Le shell substitue alors autant de chaînes qu'il est possible de former de fichiers concordants. S'il n'en trouve pas, le mot est laissé tel quel.
- Le tilde '~' seul, suivi de / ou d'un nom de login est toujours substitué, et en premier.

?	Remplace un caractère exactement
*	Remplace un nombre quelconque de caractères (y compris aucun)
[p]	Remplace un seul caractère parmis ceux indiqués dans p (même syntaxe que pour les expressions régulières)
~nom	Substitué par la variable $HOME de l'utilisateur *nom*

Evaluation des noms de fichiers

```
$ echo ~
/Users/xavier
$ echo $HOME
/Users/xavier
$ ls
cap1.tiff       cap2.tiff       test
$ echo cap?.tiff capi*.tiff
cap1.tiff cap2.tiff capi*.tiff
$ echo *es*
test
$ echo /???
/bin /dev /etc /lib /tmp /usr /var
$ echo /???/??
/bin/cp /bin/dd /bin/df /bin/ed /bin/ln /bin/ls /bin/mv /bin/ps
   /bin/rm /bin/sh /dev/fd /etc/rc /var/at /var/db /var/vm
   /var/yp
$ echo /???/[a-c]??
/bin/cat /bin/csh /usr/bin
```


Evaluation

En résumé :

- Tous les mots non délimités par des " et des ' sont évalués : variables, arithmétique, et de noms de fichiers
- Les variables contenues dans les mots encadrés par des " sont évaluées, mais pas les noms de fichiers
- Les chaînes de caractères encadrées par des ' ne sont jamais évaluées
- Un métacaractère précédé de ‘\’ se comporte comme un caractère ordinaire
- Les chaînes encadrées par des " ou par des ' sont toujours considérées comme un seul mot

Exemples

```
$ i=1
$ ls
cap1.tiff       cap2.tiff
   test
$ echo  $i $((i+1)) cap*
1 2 cap1.tiff cap2.tiff
```

```
$ echo  "$i" "$((i+1))" "cap*"
1 2 cap*
$ echo  '$i' '$((i+1))' 'cap*'
$i $((i+1)) cap*
$ echo  \$i \$\(\(i+1\)\) cap\*
$i $((i+1)) cap*
```

Evaluation de commande

L'évaluation de commande demande à ce que le contenu produit par `commande` sur sa sortie standard soit substitué sur la ligne de commande courante. Les délimiteurs autres que l'espace (tabulations, retours chariots) sont remplacés par des espaces.

Forme générale: `$(commande)` ou `` `commande` ``

Exemple

```
$ ls
cap1.tiff       cap2.tiff       test
$ ls | wc -l
       3
$ echo Il y a $(ls | wc -l) fichiers
Il y a 3 fichiers
$
```


Les redirections

A son lancement, tout processus dispose de trois canaux ouverts par le système :

- l'**entrée standard** = stdin, canal **0**
- la **sortie standard** = stdout, canal **1**
- la **sortie d'erreur standard** = stderr, canal **2**

A ces canaux sont associés des fichiers par défaut :

- les caractères tapés par l'utilisateur dans le terminal d'exécution alimentent stdin
- le terminal est alimenté par stdout et stderr

Les redirections permettent d'associer d'autres fichiers aux canaux que ceux par défaut.

```
$ ls
cap1.tiff       cap2.tiff       test
$ ls > /tmp/toto # redirige stdout vers /tmp/toto
$ ls -l /tmp/toto
-rw-r--r--   1 xavier  wheel  25 Feb 25 23:24 /tmp/toto
```

Les redirections

```
$ cat /tmp/toto
cap1.tiff
cap2.tiff
test
$ wc -l < /tmp/toto # lit stdin a partir de /tmp/toto
       3
```

[n]> fichier	Redirige stdout (ou n) vers le fichier
[n]>> fichier	Ajoute stdout (ou n) en fin de fichier
[n]< fichier	Alimente stdin (ou n) à partir du fichier
[n1]>&n2	Unifie stdout (ou n1) au canal n2 en sortie
[n1]<&n2	Unifie n2 et stdin (ou n1) en entrée
[n]>&-	Fermer la sortie standard (ou n)
[n]<> fichier	Associer le fichier en lecture/écriture à stdin (ou n)
[n]<<chaine	Lit sur stdin (ou n) jusqu'à l'apparition de la chaîne

Commandes composées

Une commande composée est :

- soit une liste de commandes → on parle de liste (tout court),
- soit une exécution groupée (en sous-shell ou shell courant),
- soit une instruction de contrôle,
- soit une fonction

Listes

Une liste est une suite de commandes, éventuellement composées (opérateurs '&&', '||', ';', '&', '|'), et terminée par un ';', un '&' ou un retour chariot.

Si C1 et C2 sont deux *commandes simples,* alors:

- C1 && C2 : lance C1 puis C2 seulement si C1 a réussi
- C1 || C2 : lance C1 puis C2 seulement si C1 a échoué
- C1 ; C2 : lance C1 puis C2
- C1 & : lance C1 en arrière-plan → **exécution parallèle**
- C1 | C2 : lance C1 et C2 en parallèle et en redirigeant la sortie standard de C1 vers l'entrée standard de C2 (*tube)*

Commandes composées

Les tubes

Forme générale: C1 | C2 où C1 et C2 sont 2 commandes simples

Le tube (‘|’) permet de lancer C1 et C2 en parallèle après avoir connecté la sortie standard de C1 vers l’entrée standard de C2 - donc C2 lit ce que C1 a produit

Illustration

- ls -l produit des fichiers (1 fichier / ligne) sur sa sortie standard
- wc (word count) compte combien de caractères, mots, et lignes elle trouve sur son entrée standard
- Pour compter combien de lignes ls a produit, on pourrait opérer en deux temps :
  - faire rediriger la sortie de ls vers un fichier
  - faire relire ce fichier par word count

```
[hilairex@pc5352a ~]$ ls -l > /tmp/sortie
[hilairex@pc5352a ~]$ wc < /tmp/sortie
  29  259 1856
[hilairex@pc5352a ~]$
```


Commandes composées

Illustration (suite)

Le tube fait la même chose, sauf qu’il évite de créer */tmp/sortie,* et qu’en outre, ls et wc sont parallélisées :

```
[hilairex@pc5352a ~]$ ls -l | wc
     29     259    1856
[hilairex@pc5352a ~]$
```

A cause de ce mécanisme, la quasi totalité des commandes UNIX qui lisent un fichier et en produisent un autre par défaut, le font systématiquement à partir de leur entrée standard et vers leur sortie standard.

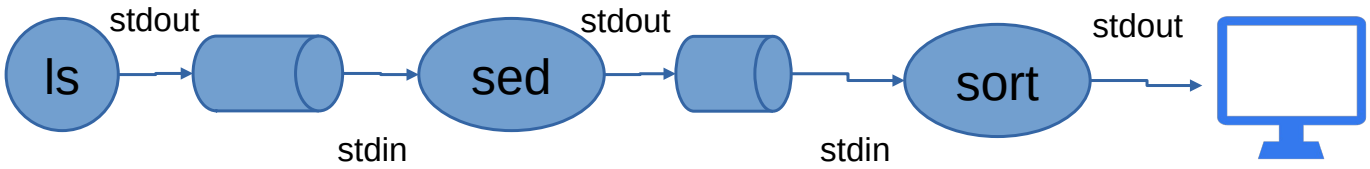
Consulter le manuel ! (**man** + commande)

Commandes composées

Autre exemple avec sed (stream editor), et sort (tri)

```
$ ls -1
cap1.tiff
cap2.tiff
test
$ ls -1 | sed "s/tiff/TIF/"
cap1.TIF
cap2.TIF
test
```

```
$ ls -1 | sed "s/tiff/TIF/" |
      sort -r
test
cap2.TIF
cap1.TIF
$
```



Commandes composées

Exécution groupée

Formes générales:

{ *liste*; } pour une exécution dans le shell courant

(*liste*) pour une exécution dans un sous-shell

Modifie la priorité d'analyse (lexicale) de liste. Pour l'exécution dans un sous-shell, tout se passe comme si liste constituait un processus à part entière (y compris pour les entrées/sorties) et toutes les variables (ordinaires et d'environnement) du shell courant sont préservées.

Illustration

```
{ c1 && c2; } || c3
si c1 alors
      si non c2 alors c3 fsi
sinon
      c3
fsi
```

stdin

≠

```
c1 && { c2 || c3; }
si c1 alors
      si non c2 alors
            c3
      fsi
fsi
```

Commandes composées

Instructions de contrôle

Le shell admet les instructions de contrôle: if, while, for, break, continue, case. La sémantique est la même qu'en langage C.

Syntaxe de if

```
if liste; # si le code de retour de liste est zéro
then liste       # alors exécuter cette liste
[elif liste
then liste ] …
[ else liste ]
fi
```

Exemple

```
$ if test "$SHELL"
> then echo SHELL="$SHELL"
> else echo "Variable indefinie"
> fi
SHELL=/bin/bash
$
```


Commandes composées

Syntaxe de while

```
while liste; # tant que code de retour de liste = 0
do liste # exécuter cette liste
done
```

Exemple:

```
$ i=0
$ while test $i -le 2  # tant que i <= 2
> do echo i=$i ; i=$((i+1))
> done
i=0
i=1
i=2
$ echo $i
3
$
```

Commandes composées

Syntaxe de for

for variable in *mot1 [mot2…]; do*
** *liste* # exécuter cette liste**
done

Exemple:

```
$ for i in un 2 trois; do
> echo $i
> done
un
2
trois
```

Syntaxe de break et continue

break [n]
continue [n]

Sort du nième niveau (ou du courant) d'itération soit pour arrêter (break) ou poursuivre (continue) . Idem langage C.

Commandes composées

Syntaxe de case

case mot in
motif) liste ;;
…
esac

Motif peut comporter plusieurs motifs élémentaires séparés par des '|', chacun suivant les mêmes règles de développement que pour les noms de fichiers. Sémantique identique à celle du langage C, mais break n'est pas nécessaire.

Exemple:

```
$ case $OSNAME in
> Lin*) echo "motif 1";;
> tux|*nux) echo "motif 2";;
> *) echo "defaut";;
> esac
motif 1
```

```
$ case toto in
> Lin*) echo "motif 1";;
> tux|*nux) echo "motif 2";;
> *) echo "defaut";;
> esac
defaut
```

Commandes composées

Les fonctions

Forme générale: **`nomfonc () liste`**

où `nomfonc` est un mot et `liste` une liste (généralement entre {}).

- N'est commodément utilisable que dans un script.
- Après déclaration nomfonc est vu comme un nom de commande, qui peut recevoir des arguments
- Les arguments passés sont accessibles par $*, $@, $0, $1,…
- La fonction peut renvoyer une valeur entière via l'instruction return, récupérable par l'appelant à travers la variable $?
- Les variables qui ne sont pas déclarées `local` sont en fait celles du shell, et la fonction les altère (!)

Commandes composées

Fonctions (suite)

```
$ compte_fic ()
> {
>     local v
>
>     test -d "$1" && ls -l "$1" | wc -l && v=0 || v=1
>     return $v
> }
$
$ compte_fic /tmp/
       3
$ echo "Retour1=$?"
Retour1=0
$ compte_fic /inexistant/
$ echo "Retour2=$?"
Retour2=1
```

Scripts shell

- On peut stocker un ensemble de lignes de commandes dans un fichier texte pour en faire un script shell exécutable.
- Pour pouvoir être lancé, le fichier créé doit avoir le droit d'exécution (chmod +x)
- Si la première ligne du fichier débute par #!, elle doit être suivie du chemin du shell à lancer. Si ce n'est pas le cas, le shell courant l'exécutera dans un sous-shell.
- On peut alors lancer le script en l'appelant directement par son nom

```
$ ls monscript.sh
-rw-r--r--   1 xavier  xavier  165 Feb 26 16:44
   monscript.sh
$ chmod +x monscript.sh
$ cat monscript.sh
#!/bin/sh
#
```


Scripts shell

```
compte_fic ()
{
    local v

    test -d "$1" && ls -l "$1" | wc -l && v=0 || v=1
    return $v
}


compte_fic "$1"
if test $? -ge 1
then echo Erreur
fi
$ ./monscript.sh .
       7
$ ./monscript.sh /inexsitant
Erreur

$
```

Principales commandes
Principaux utilitaires

Commandes internes

Ce sont des commandes propres au shell - elles ne sont ni des scripts, ni des fichiers binaires, ni des fonctions. Commandes essentielles:

Nom	Fonction	Exemple
pwd	Donne le chemin courant	$ pwd /Users/xavier/tmp $ echo $PWD /Users/xavier/tmp

cd [dir]	Change le chemin courant à $HOME (ou dir)	$ cd /tmp
		$ pwd
		/tmp
		$ cd
		$ pwd
		/Users/xavier

Commandes internes

Nom	Fonction	Exemple
read	Lit une variable depuis l'entrée standard	$ read ligne Ceci est un test. echo $ligne Ceci est un test.
shift [n]	Décale tous les paramètres de 1 (ou n) vers la gauche	shift # $1 devient $0, $2 devient $1,…
jobs	Liste les processus lancés en arrière-plan	$ jobs [1]- Running xterm & (wd: ~/tmp) [4]+ Running xterm & (wd: ~/tmp)
fg [n]	Ramène le dernier processus d'arrière-plan (ou n) au premier-plan	$ fg 4 xterm (wd: ~/tmp)

Commandes externes

- Au sens large, comprennent tous les fichiers binaires accessibles par la variable PATH.
- La plupart des utilitaires et commandes système se trouvent dans :

/bin,/sbin,/usr/bin,/usr/sbin,/usr/local/bin,/usr/X11R6/bin

Accessibles lors du boot

Accessibles après montage de /usr

Accessibles sur réseau local

Clients X11

Une commande très utile : man

`man -a commande`

affiche les pages manuel décrivant `commande`

`man -k mot-clef`

recherche les commandes qui concordent avec le mot-clé fourni.

Commandes externes

Commandes diverses

Nom	Description
`test ou [`	**Test l'existence de fichiers/répertoires, l'égalité, la supériorité, l'infériorité sur des entiers et des chaînes de caractères**
`clear`	Efface le terminal
`id`	Affiche les informations d'identité de l'utilisateur
`uname`	Nom du système d'exploitation
`who`	Liste les utilisateur connectés
`passwd`	Modification du mot de passe utilisateur
`su`	Changer d'identité utilisateur (substitute user)
`hostname`	Nom de la machine exécutant le shell

Commandes externes

Manipulation de fichiers

Nom	Description
`cp`	Copie de fichiers
`mv`	Renomme/déplace des fichiers
`mkdir`	Créer des répertoires
`rmdir`	Supprimer des répertoires
`ls`	Lister les fichiers
`find`	Chercher des fichiers dans l'arborescence
`chmod`	Modifier les droits d'accès
`chown`	Changer le propriétaire d'un fichiers
`mktemp`	Créer un fichier temporaire

Commandes externes

Commandes sur fichiers texte

Nom	Description
`cat`	Concatène des fichiers
`sort`	Trier les lignes d'un fichier
`cut`	Sélectionner des champs/colonnes
`paste`	Fusionner des champs entre eux
`tr`	Substituer des caractères à d'autres
`grep`	Sélectionner des lignes
`sed`	Editeur de flux (stream editor)
`vi, vim, ed`	Editeurs en mode texte
`less`	Consulter un fichier interactivement

Commandes externes

Commandes sur processus

Nom	Description
`ps`	Afficher les processus chargés
`top`	Afficher les processus chargés en temps réel
`kill`	Envoyer un signal à un processus
`renice`	Changer la priorité d'exécution d'un processus

Commandes externes

Commandes réseau

Nom	Description
`ifconfig`	Afficher les informations sur les interfaces réseau
`netstat`	Afficher l'état des ressources réseau (trafic, sockets, ports, tables de routage)
`ping`	Envoyer des paquets ICMP
`arp`	Résolution d'adresse Internet->Ethernet
`traceroute`	Afficher l'itinéraire hôte-hôte par ICMP
`nslookup`	Résolution nom d'hôte -> Adresse IP
`ypbind`	Maintenance de la liaison avec un serveur NIS

Les processus

Cours n°3

Systèmes d'exploitation – ESIEE Paris - 3R-IN3

Document originel : Clément BOIN
Adaptation : Xavier HILAIRE

Plan

- Introduction sur les processus
- Gestion des processus sous Linux
  - principales commandes
  - les signaux
  - les modes d'exécution
  - attendre un processus enfant

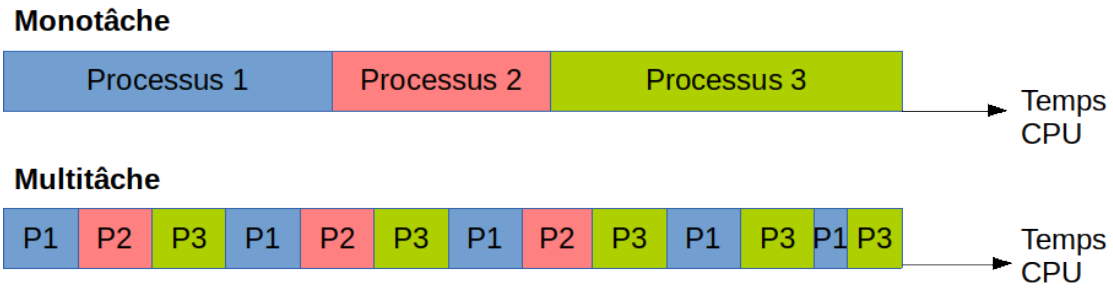
Introduction

Les processus : définitions

- Un processus est une unité de programme en exécution :
  - Un espace mémoire
  - Des ressources utilisées (fichiers, périphériques…)
- Plusieurs processus peuvent être exécutés en même temps.

Monotâche/Multitâche

- Monotâche :
  - un seul processus à la fois
  - Pas de problème de gestion de la mémoire
  - Pas de conflit de ressources
- Multitâche
  - Mise en commun, partage
  - Attention à la gestion, aux conflits et à la sécurité



Gestion de processus (sous Linux)

Arborescence des processus (sous Linux)

- Au démarrage du système un processus nommé systemd est lancé (init sur certains linux)
  - responsable du lancement des autres processus
  - hiérarchie de processus de racine systemd
- Chaque processus est identifié par :
  - un numéro unique (PID : process identifier),
  - l'identité du propriétaire
  - le numéro du processus père (PPID) qui l'a créé : un processus n'a qu'un seul parent
  - systemd a en principe un PID de 1, mais ce nombre peut changer (un systemd par utilisateur, en particulier)

Commande pstree

Hiérarchie des processus

```
clement@boko:~$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager─┬─2*[dhclient]
        │                └─2*[{NetworkManager}]
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─acpid
        ├─apache2───5*[apache2]
        ├─avahi-daemon───avahi-daemon
        ├─boltd───2*[{boltd}]
        ├─colord───2*[{colord}]
        ├─cron
        ├─cups-browsed───2*[{cups-browsed}]
        ├─cupsd───dbus
        ├─dbus-daemon
        ├─dropbox───86*[{dropbox}]
        ├─firefox─┬─Web Content───48*[{Web Content}]
        │         ├─Web Content───37*[{Web Content}]
        │         ├─Web Content───24*[{Web Content}]
        │         ├─WebExtensions───32*[{WebExtensions}]
        │         └─80*[{firefox}]
        ├─fwupd───4*[{fwupd}]
        ├─gdm3─┬─gdm-session-wor─┬─gdm-wayland-ses─┬─gnome-session-b─┬─gnome-sh+
        │      │                 │                 │                 ├─gsd-a11y+
        │      │                 │                 │                 └─gsd-clip+
```

Sur une machine cliente (extrait)

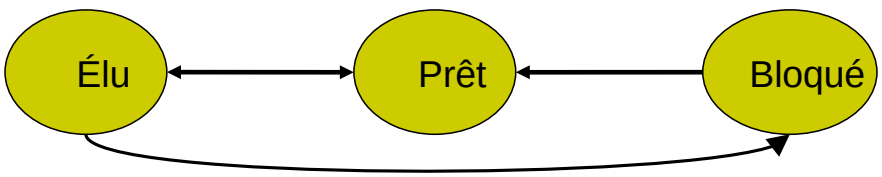
```
test@ubuntu:~$ pstree
systemd─┬─accounts-daemon─┬─{gdbus}
        │                 └─{gmain}
        ├─acpid
        ├─apache2───5*[apache2]
        ├─atd
        ├─cron
        ├─dbus-daemon
        ├─dhclient
        ├─2*[iscsid]
        ├─login───bash───pstree
        ├─lvmetad
        ├─lxcfs───2*[{lxcfs}]
        ├─mdadm
        ├─mysqld───26*[{mysqld}]
        ├─polkitd─┬─{gdbus}
        │         └─{gmain}
        ├─rsyslogd─┬─{in:imklog}
        │          ├─{in:imuxsock}
        │          └─{rs:main Q:Reg}
        ├─snapd───6*[{snapd}]
        ├─systemd───(sd-pam)
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-timesyn───{sd-resolve}
        └─systemd-udevd
test@ubuntu:~$ _
```

Sur un serveur
sans interface graphique
(tous les processus)

Etat d'un processus

- Chaque processus possède son propre environnement d'exécution.

- 3 états possibles :
  - élu : le processus est en cours d'exécution
  - prêt : tout sauf le processeur
  - bloqué : attend une ressource non encore disponible (saisie d'une valeur par exemple)



Quelques commandes système liées à la gestion des processus

- `top`
- `ps`
- `pstree`

- `free`: état de la mémoire vive

- `iostat –h` : état du processeur

- `nohup` : exécuter une commande résistante aux déconnexions

- `nice`: modifier la priorité d'une commande

La commande **top**

- Permet d'afficher des informations en continu sur l'activité du système.
  - ressources que les processus utilisent
    - quantité de RAM
    - pourcentage de CPU
    - la durée de ce processus depuis son démarrage

- On peut par exemple stopper un processus
  - taper **k**.
  - Saisir le PID du processus
  - Saisir le signal de fin de processus : 15 (SIGTERM)
  - 9 (SIGKILL) est plus brutal !

- Pour quitter top, appuyer sur la touche "**q**".

La commande **ps**

- Permet de connaître les processus actifs à un moment donné
  - Chaque processus est identifié par un nombre unique (PID).
  - TTY : à quel port de terminal est associé le processus.
  - STAT : état du processus
    - S comme "sleep" : endormi
    - R comme "run" : en cours d'exécution
  - TIME : depuis combien de temps le processus utilise les ressources du microprocesseur.
  - COMMAND précise la commande

```
test@ubuntu:~$ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1034  0.0  0.3  65832  3400 tty1     Ss   22:45   0:00 /bin/login --
test      1294  0.0  0.5  22916  5508 tty1     S    22:46   0:00 -bash
test      1427  0.0  0.3  37472  3412 tty1     R+   23:40   0:00 ps au
```

- `ps aux` (syntaxe BSD)
  - permet de connaître les utilisateurs associés à chaque processus (en incluant les applications du serveur graphique X)
- `ps axms`
  - permet d'avoir des informations sur les threads

Communication inter-processus

- Les processus peuvent communiquer entre eux par l'envoi de messages appelés signaux.
  - Asynchrone
  - Nombre limité
  - Réactions prédéfinies
- Arrêter un processus (interrompre)
  - signal SIGINT
  - signal numéro 2
  - généré depuis le clavier en appuyant simultanément sur les touches CTRL+C.
- Le signal de terminaison
  - SIGKILL
  - signal numéro 9
  - ne peut pas être inhibé

La commande **kill**

- Permet d'expédier un signal à un processus en cours.
  `kill [options] PID`

- Si vous n'arrivez pas à tuer un processus, vous devez utiliser la commande :
  `kill -9 PID`

- **`killall`** permet aussi de tuer un processus en indiquant son nom.

Commande kill -l

Liste des signaux

```
clement@boko:~$ trap -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- 64 messages différents
- Interruption du flot de contrôle d’un processus
- Proviennent d'un événement matériel ou logiciel

Les modes d'exécution

Mode d'exécution séquentiel

On tape les commandes les unes après les autres

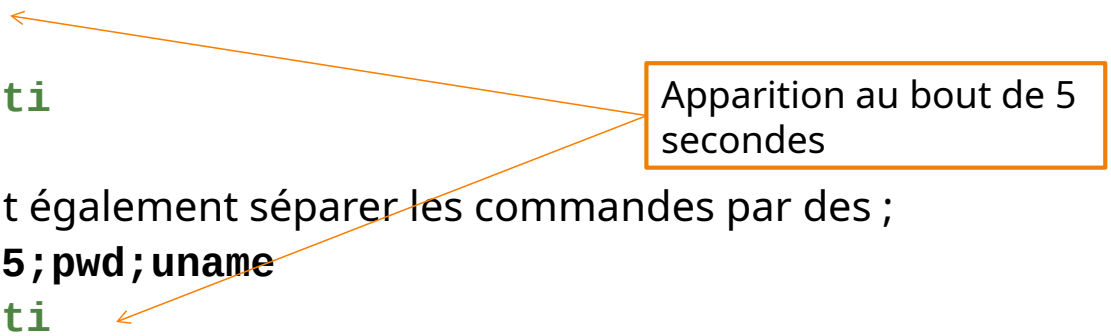
- Entrée entre chaque commande

```
$sleep 5
pwd
$pwd
/home/uti
```

- On peut également séparer les commandes par des ;

```
$sleep 5;pwd;uname
/home/uti
Linux
```

- Pour arrêter une exécution on tape Ctrl-C



Exécution en arrière plan

- Opérateur &
- Permet de lancer une commande et récupérer la main immédiatement

```
cboin@clement-boin:~$ sleep 5&
[1] 1085
cboin@clement-boin:~$ ps
  PID TTY          TIME CMD
  986 tty1     00:00:00 bash
 1085 tty1     00:00:00 sleep
 1086 tty1     00:00:00 ps
cboin@clement-boin:~$ ps
  PID TTY          TIME CMD
  986 tty1     00:00:00 bash
 1085 tty1     00:00:00 sleep
 1087 tty1     00:00:00 ps
cboin@clement-boin:~$ ps
  PID TTY          TIME CMD
  986 tty1     00:00:00 bash
 1088 tty1     00:00:00 ps
[1]+  Done                    sleep 5
cboin@clement-boin:~$ ps_
```


La commande **wait**

- Commande interne du shell, qui permet d'attendre la fin de ses processus enfants

- Deux formes possibles :
  - wait PID : attend ce PID précisément. Le code de sortie (= valeur renvoyée par exit) du fils décédé est récupérable dans la variable spéciale '?'
  - wait : attend tous les fils. '?' vaut toujours 0.

La commande **wait**

```
[hilairex@pc5352a ~]$ gedit &
[3] 10789
[hilairex@pc5352a ~]$ echo $!
10789
[hilairex@pc5352a ~]$ wait 10789
[3]+  Fini                    gedit
[hilairex@pc5352a ~]$ echo $?
0
[hilairex@pc5352a ~]$ gedit & emacs &
[1] 11117
[2] 11118
[hilairex@pc5352a ~]$ wait
[1]-  Fini                    gedit
[2]+  Fini                    emacs
[hilairex@pc5352a ~]$  echo $?
0
[hilairex@pc5352a ~]$
```

Validation au terminal (entrée) **puis** fermeture de la fenêtre gedit

Fermeture de gedit et emacs (peu importe l'ordre)

La commande **wait**

- Remarque : le fils peut déjà être terminé lorsque l'appel à wait est fait, sans que cela soit un problème

```
[hilairex@pc5352a ~]$ (exit 2) &
[1] 11444
[1]+  Termine 2               ( exit 2 )
[hilairex@pc5352a ~]$ wait 11444
[hilairex@pc5352a ~]$ echo $?
2
[hilairex@pc5352a ~]$
```

Exécution dans un sous-shell

non bloquant

Expressions régulières et utilitaires

Xavier HILAIRE
x.hilaire@esiee.fr

Rappels ou révélations

- Une *expression régulière* (ou *expression rationnelle*, ou *ER*, ou encore «*regex*») est une expression qui décrit un langage rationnel
- Un *langage rationnel* sur un alphabet Σ est:
  - Soit le langage vide $\oslash$
  - Soit un langage de la forme {a}, où $a \in \Sigma$
  - Soit un langage de la forme L*, L + R, ou L·R, où L et R sont des langages rationnels, et L* dénote la fermeture de Kleene de L.
- Par définition:
  - L+R est le langage formé des mots de L et R (l'union)
  - $L \cdot R = \{ xy : x \in L, \in R \}$ est le langage formé par concaténation de tous les mots de L et de R
  - L* est le langage obtenu en concaténant entre eux, et autant de fois que l'on veut (y compris infini) l'ensemble de tous les mots de L

Rappels ou révélations

Exemple: soit Σ = { a,b,c }, L = { a,ab }, et R = { ε,c }.
Alors:

- L + R = { ε, a, ab, c }
- L·R = { a, ac, ab, abc }
- L* = { ε, a, ab, aab, aba, aaab, … }
- L et R sont réguliers sur Σ

Autres exemples:

- L'ensemble des numéros minéralogiques français forme un langage rationnel sur Σ = { 0,1,2,3,4,5,6,7,8,9, A, …, Z }
- L'ensemble des entiers naturels forme un langage rationnel sur l'alphabet Σ = { 0,1,2,3,4,5,6,7,8,9 }

Rappels ou révélations

Contre-exemples:

- L'ensemble des expressions algébriques bien parenthésées de deux variables a et b ne forme pas un langage rationnel sur Σ = { a,b,+,-,(,) }
- L'ensemble des palindromes ne forme pas un langage rationnel sur l'alphabet usuel Σ = { a, b, …, z }

Théorème (Kleene) : tout langage régulier est reconnaissable par un automate à états finis (AEF).

Donc il suffit de trouver un moyen de représenter un AEF - et la reconnaissance du langage régulier suivra… c'est précisément le but des expressions régulières.

Rappels ou révélations

Un automate à états finis est un quintuplet (Q,Σ, δ, q0, T) dans lequel :

- Q est un ensemble dénombrable et fini appelé ensemble des états
- Σ est l'alphabet (fini)
- $\delta : Q \times \Sigma \rightarrow Q$ est une fonction appelée fonction de transition
- $q0 \in Q$ est un état appelé état initial
- T est l'ensemble des états terminaux

On représente généralement un AEF par un graphe orienté dans lequel les arcs représentent la fonction de transition, et les nœuds les états. **Chaque transition «consomme» les symboles qui la définissent**

Rappels ou révélations

Exemple: automate reconnaissant un numéro minéralogique (ancien format: au moins un chiffre qui n'est pas 0, suivi d'au moins une lettre, suivie de deux chiffres exactement):

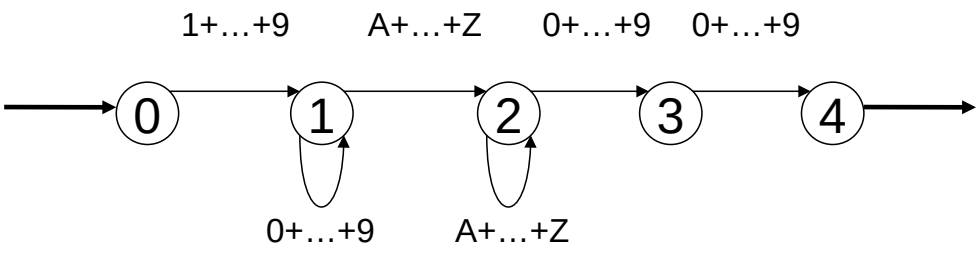
Q={0,1,2,3,4}, q0=0, Σ = { 0,…,9,A,…,Z }, T = {4}

δ(0,1)= δ(0,2)=…= δ(0,9)=1 ; δ(0,0) indéfini

δ(1,A)=…= δ(1,Z)= 2

δ(1,0)= δ(1,1)=…= δ(1,9)=1

(exercice: compléter la définition de δ)

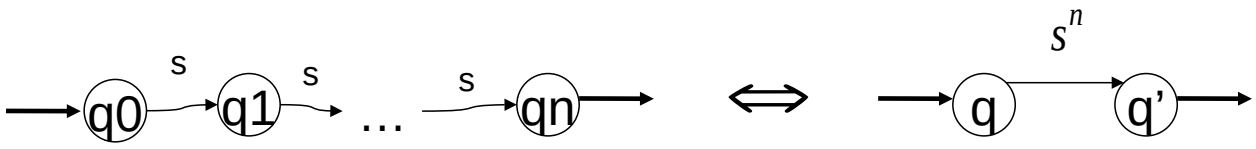


Rappels ou révélations

Remarques:

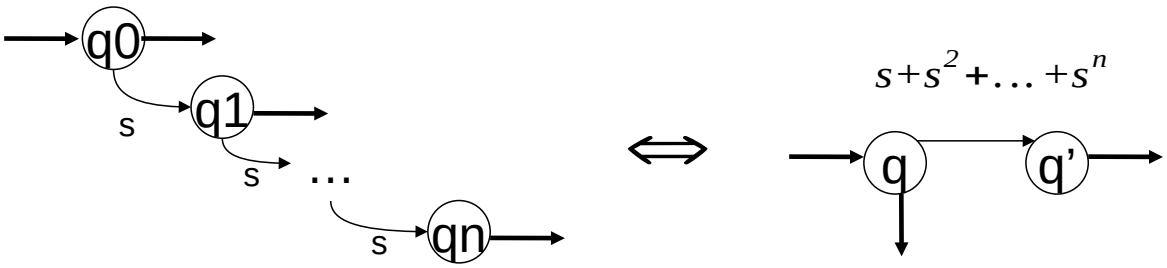
- En toute rigueur, on aurait du utiliser 9 flèches pour la transition état 0 -> état 1 (une par chiffre). Pour alléger les notations, on a recours à l'opérateur '+' pour dénoter le OU logique entre ***mots*** ; de même, l'opérateur '.' (ou produit) dénote le ET logique
- On peut alors étendre δ aux mots sur Σ, et on obtient les représentations suivantes :

1. Répétition n fois (n $\geq$1) d'un même symbole s

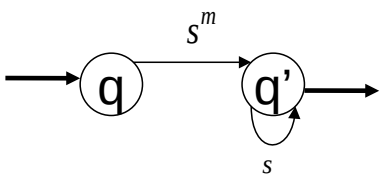


Rappels ou révélations

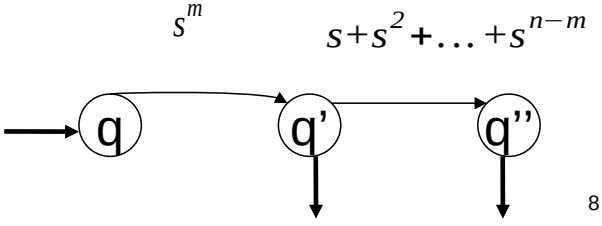
2. Répétition au plus n fois (n $\geq$1) d'un même symbole s



3. Répétition au moins m fois (m $\geq$1) d'un même symbole s



4. Répétition au moins m et au plus n fois (n > m $\geq$1) d'un même symbole s

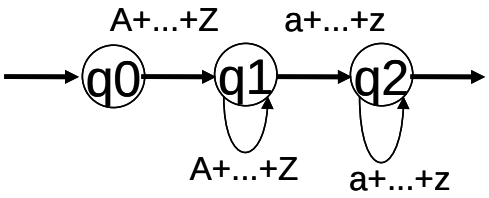


Rappels ou révélations

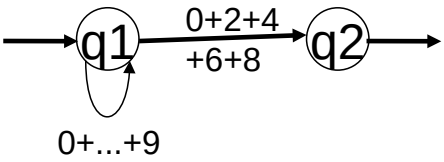
Automate déterministe et non déterministe

La fonction étant à image dans Q, elle n'est censé renvoyer qu'un seul état possible à partir d'un état donné et d'un symbole : on parle alors d'**automate déterministe**

Si ce n'est pas le cas (un même symbole peut conduire à plusieurs états possible), ou si le symbole qui conditionne la transition est le mot vide, on parle d'**automate non déterministe**



Automate déterministe reconnaissant un prénom



Automate non déterministe reconnaissant un entier naturel pair

Rappels ou révélations

Remarques :

- Il est possible de transformer un automate non déterministe en sa version déterministe, grâce à l'algorithme de Rabin-Scott
- Le nombre d'états est toutefois en $O(2^{|\Sigma|})$
- Le temps d'exécution suit la même complexité dans le pire cas

Expressions régulières

- Les expressions régulières (ER) UNIX sont des chaînes de caractères qui permettent de coder l'AEF reconnaissant un LR
- Elles servent généralement à compiler (fonction système C recomp()) puis à exécuter (regexec()) l'AEF de reconnaissance associé
- Deux formes usuelles d'ER définies par le standard IEEE 1003.2:
  - Forme **de base** : les méta-caractères '{','}','(',')' doivent être précédés d'un '\', sinon ils se comportent comme des caractères ordinaires. Les caractères '+', '?', et '|' sont toujours des caractères ordinaires. Les caractères '^' et '$' n'ont de sens qu'en début et en fin d'expression.
  - Forme **étendue** : celle décrite ci-après

ER : Forme étendue

- Une ER étendue est un ensemble de sous-expressions régulières (SER) délimitées par des '|'. L'ER est reconnues ssi l'une des SER est reconnue.
- Une SER est une succession d'atomes, chacun pouvant être suivi de '*', '+', '?', ou de bornes {inf,sup}. Une SER est reconnue ssi tous les atomes sont reconnus dans leur ordre (analyse gauche-droite) :
- Les bornes {inf,sup}, où inf et sup sont des entiers, indiquent que l'atome qui précède doit être reconnu au moins inf fois et au plus sup fois durant la reconnaissance. Il est possible d'omettre l'une ou l'autre des bornes, mais pas les deux.
- '*' est équivalent à '{0,}' - l'atome peut être reconnu un nombre arbitraire de fois, y compris zéro

ER : Forme étendue

- ‘+’ est équivalent à ‘{1,}’ - l’atome doit être reconnu au moins une fois.
- ‘?’ est équivalent à {,1} - l’atome qui précède peut être reconnu au plus une fois

Un atome est soit :

- une ER délimitée par ‘(‘ et)’ (y compris la chaîne vide). Si ‘(‘ et ‘)’ sont changées en ‘\(‘,’\)’, l’ER représente la portion de chaîne à extraire de la chaîne candidate lors de la reconnaissance, si celle-ci réussit
- le caractère ‘.’, qui autorise la reconnaissance de n’importe quel caractère

ER : Forme étendue

- une expression crochetée, de la forme ‘[p…p]’, qui est reconnue ssi l’un des ‘p’ est reconnu. Chaque ‘p’ est:
  - soit une plage de la forme *x-y*, où *x* et *y* sont des caractères, et qui signifie « tous les caractères dans la plage x:y »
  - soit un caractère simple
  - soit une plage ou un caractère simple précédé par ‘^’, et qui signifie « tout sauf » la plage ou le caractère qui suit
- le caractère ‘^’, qui force la reconnaissance d’un début de ligne
- le caractère ‘$’, qui force la reconnaissance du mot vide (fin de la chaîne)
- une chaîne de la forme ‘\c’, où c est un caractère qui devra être reconnu comme tel - n’a d’intérêt que pour \^, \$, \[, etc.

ER : Forme étendue

Exemples

1. Expression régulière étendue reconnaissant les chaînes de la forme «Prénom Nom»:

   `[A-Z][a-z]+ [A-Z][a-z]+`

2. ER étendue reconnaissant un prénom éventuellement composé: «Prénom1[-Prénom2]»

   `[A-Z][a-z]+(-[A-Z][a-z]+)?`

3. ER étendue reconnaissant un prénom arbitrairement composé: «Prénom1[-Prénom2[-Prénom3]…]»

   `[A-Z][a-z]+(-[A-Z][a-z]+)*`

4. ER étendue reconnaissant les chaînes qui ne débutent pas par un ‘A’

   `^[^A]`

5. ER étendue reconnaissant les chaînes ne débutant pas par une lettre majuscule ou se terminant par une lettre minuscule, et comportant 3 caractères exactement:

   `^[^A-Z]..$|^..[a-z]$` ou bien: `^([^A-Z]..|..[a-z])$`

ER : Forme étendue

6. ER étendue reconnaissant une plaque minéralogique française (modèle simplifié: entier sur au plus 4 chiffres, espaces arbitraires, au moins 2 et au plus 3 lettres, espaces arbitraires, 2 chiffres):

   `[1-9][0-9]{,3} *[A-Za-z]{2,3} *[0-9]{2,2}`

Exercice:

- ER reconnaissant un entier naturel:
- ER reconnaissant un entier relatif:
- ER reconnaissant un nombre flottant (sans notation scientifique ‘E’)
- ER reconnaissant un nombre flottant (avec notation scientifique ‘E’)

ER : Forme de base

- Forme moins puissante que la forme étendue, puisque '+', '?', et '|' sont des caractères ordinaires
- Les fonctions associées à '+' et '?' disparaissent, cependant c+ équivaut à c\{0,\} et c? à c\{0,1\} → transformation toujours possible
- Les métacaractères (,), {, } de la forme étendue conservent leur sens s'ils sont précédés d'un \
- Les méta caractères ^ et $ gardent leur sens en début et fin d'expression sans besoin d'être précédés de \
- En revanche, le choix multiple '|' est perdu, et le caractère ^ perd son sens négatif en milieu d'expression.

ER : Forme de base

Exemples :

Chaîne	ER de base	Concordance ?
Alain	**[A-Za-z]+**	non
Alain	**[A-Za-z]\{1,\}**	oui
Pierre-Alain	**[A-Za-z]\{1,\}\(-[A-Za-z]\{1,\}\)?**	non
Pierre-Alain	**[A-Za-z]\{1,\}\(-[A-Za-z]\{1,\}\)\{0,1\}**	oui
Alain	**^Alai$**	non
Alain	**^[A-Za-z]*n$**	oui
Alain	**^Alain$\|^Eric$**	non

La commande expr

Elle permet de tester la concordance d'une chaîne avec une expression régulière de base, et optionnellement d'extraire certaines parties reconnues

Syntaxe: `expr e1 op e2`

- e1 et e2 sont deux expressions, et op un opérateur (peut être : ou |, &, +, -, …)
- Si op est ':', alors e2 doit être une ER **de base**, et e1 une chaîne candidate. Expr écrit soit le résultat de la reconnaissance de e1 par e2 si e2 comporte des '\(\)', soit la longueur du résultat .

La commande expr

Exemple : reconnaissance d'une chaîne «Nom Prénom»

```
$ expr "Xavier Hilaire" : "[A-Z][a-z]+ [A-Z][a-z]+"
0
```

Ne marche pas : e2 doit être une ER **de base**. Donc:

```
$ expr "Xavier Hilaire" : "[A-Z][a-z]\{1,\} [A-Z][a-z]\{1,\}"
14
```

On peut aussi extraire une partie de l'ER reconnue, à condition de la délimiter par '\(' et '\)'; par exemple, pour le nom seul:

```
$ expr "Xavier Hilaire" : "[A-Z][a-z]\{1,\} \([A-Z][a-z]\{1,\}\)"
Hilaire
$
```

La commande expr

Remarques:

1. Du fait de l'utilisation des ER de base, expr ne peut pas utiliser de parenthésage sans réaliser l'extraction !
2. On utilise plus fréquemment le code de retour de expr que le résultat de la reconnaissance proprement dit:
   - 0 → sortie avec succès
   - 1 → chaîne candidate non reconnue
   - 2 → ER invalide

Utilisation de expr (avec extraction) pour les exemples précédents (à vérifier): voir TD

La commande expr

3. La commande expr ne supporte pas les expressions régulières étendues
4. Mais le Bash offre les mêmes services avec sa commande de comparaison interne [[=~]] :

```
[hilairex@pc5352a]$ [[ Jean-Paul =~ [A-Z][a-z]+(-[A-Z][a-z]+)* ]]
[hilairex@pc5352a]$ echo $?
0
[hilairex@pc5352a]$ [[ Jean-Paul2 =~ [A-Z][a-z]+(-[A-Z][a-z]+)* ]]
[hilairex@pc5352a]$ echo $?
0
[hilairex@pc5352a]$ [[ Jean-Paul2 =~ ^[A-Z][a-z]+(-[A-Z][a-z]+)*$ ]]
[hilairex@pc5352a]$ echo $?
1
[hilairex@pc5352a]$
```

L'expression régulière **ne doit pas être entourée de quotes**

On peut récupérer la partie de chaîne qui concorde dans la variable spéciale BASH_REMATCH

La commande grep

- Elle lit le(s) fichier(s) passés en paramètre, ou à défaut l'entrée standard, et écrit les lignes du fichier pour lesquelles la reconnaissance de l'ER exp réussit.
- Syntaxe: `grep [options] exp [fichiers…]`
- Options utiles:
  - -E : interpréter exp comme une ER étendue (par défaut, c'est une ER de base)
  - -l : écrit les numéros des lignes qui concordent plutôt que les lignes elles-mêmes
  - -L : écrit les noms des fichiers qui concordent (au moins une occurrences) plutôt que les lignes
  - -v : inverse le comportement de grep - écrit les lignes qui ne concordent pas

La commande grep

Exemples

```
$ cat /etc/passwd
##
# User Database
##
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```

- Elimination des lignes de commentaires (commencent par '#'):

```
$ grep "^[^#]" /etc/passwd Ou:  grep -v "^#" passwd
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```

La commande grep

- Utilisateurs n'ayant pas de mot de passe (2ème champ vide - situation anormale!):

```
$ grep -E "^[^:]*::" /etc/passwd
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
```

- Utilisateurs ayant un compte « étoilé » (2ème champ est '*'):

```
$ grep -E "^[^:]*:\*:" /etc/passwd
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
```

- Utilisateurs dont le chemin d'accès au shell (7ème champ) comporte au moins 3 répertoires:

```
$ grep -E "^([^:]*:){6,6}(/[^/]+){4,}" /etc/passwd
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```


La commande sed

- Syntaxe (simplifiée):
  - `sed [-E] com [fichiers …]`
  - `sed [-E] -e com [-e com [-e com …]] [fichiers …]`
  - `sed [-E] -f fcom [fichiers …]`
- La commande sed (*stream editor*) effectue dans l'ordre et cycliquement les opérations suivantes:
  - Lit une ligne à partir du premier fichier spécifié, sinon l'entrée standard
  - Copie cette ligne dans un espace tampon (ET)
  - Applique la fonction com sur l'ET si c'est la seule spécifiée; ou l'ensemble des fonctions com si l'option -e est utilisée, dans l'ordre; ou l'ensemble des fonctions stockées dans le fichier fcom

La commande sed

- Ecrit le contenu de l'ET sur la sortie standard
- Libère l'ET
- Passe à la ligne suivante s'il en reste
- Passe au fichier suivant s'il y en a

- Il existe plus d'une vingtaine de fonctions différentes, mais nous ne verrons que la plus utile d'entre elles: 's' (substitute) :

  `s/regex/rempl/[flags]`

- La fonction 's' teste si l'ET concorde avec l'ER regex spécifiée.
- S'il y a concordance, l'ET est substitué par l'expression rempl, la substitution étant paramétrée par la ou les valeurs données dans les drapeaux flags, optionnels:

La commande sed

- N : ne remplacer que la Nième occurrence de regex dans l'ET
- g : remplacer toutes les occurrences dans l'ET - par défaut, seule la première occurrence est remplacée
- p : écrit l'ET vers la sortie standard
- w fichier : ajoute l'ET à fichier

- S'il n'y a pas concordance, l'ET est laissé intact.
- L'expression rempl peut contenir:
  - Le caractère &, qui sera remplacé par toute la chaîne consommée lors de la reconnaissance de regex
  - Des séquences de la forme \# ,où # est un chiffre (1-9) désignant la nième des portions d'ER reconnues et délimitées par '\(\)' si regex en comporte
- regex est supposée « de base », sauf si l'option -E a été spécifiée lors de l'appel à sed

La commande sed

Exemples

- Supprimer toutes les occurrences de ‘root’ dans le fichier passwd précédent:

```
$ sed "s/root//g" passwd
##
# User Database
##
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
:*:0:0:System Administrator:/var/:/bin/sh
daemon:*:1:1:System Services:/var/:/usr/bin/false
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```

- Ne conserver que le premier champ:

```
$ sed  "s/\([^:]*\):.*/\1/" passwd
##
# User Database
##
nobody
root
daemon
xavier
```


La commande sed

- Supprimer les deux premiers champs:

```
$ sed -E -e "s/[^:]*://1" -e "s/[^:]*://1" passwd
##
# User Database
##
-2:-2:Unprivileged User:/:/usr/bin/false
0:0:System Administrator:/var/root:/bin/sh
1:1:System Services:/var/root:/usr/bin/false
1001:1001:Xavier Hilaire:/home/xavier:/usr/local/bin/bash
```

- Permuter les champs 1 et 3:

```
$ sed "s/\([^:]*\):\([^:]*\):\([^:]*\)/\3:\2:\1/" passwd
##
# User Database
##
-2::nobody:-2:Unprivileged User:/:/usr/bin/false
0:*:root:0:System Administrator:/var/root:/bin/sh
1:*:daemon:1:System Services:/var/root:/usr/bin/false
1001:a293deFZ=/K52:xavier:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```