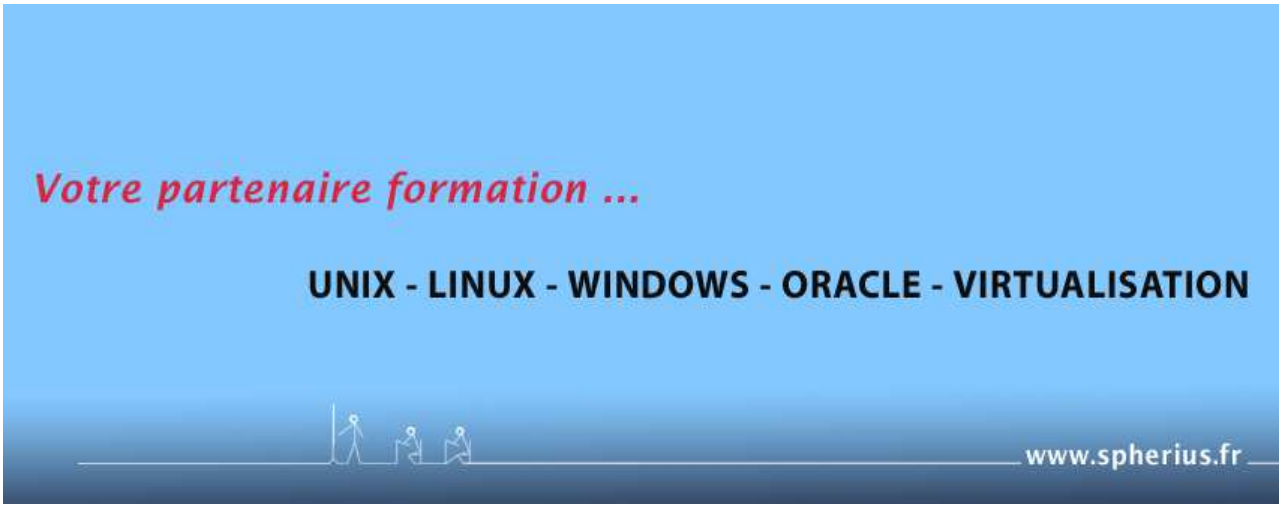
LINUX UTILISATEUR



SOMMAIRE

Les auteurs de ce document sont :

- Monsieur Baranger Jean-Marc,
- Monsieur Schomaker Theo.

La version du support de cours est:

utilisateur_Linux_version_1.0

La version de Linux utilisée pour les commandes de ce support de cours est :

CentOS 6, CentOS 7 et Debian 8

Les références sont : les documents disponible sur le site web de CentOS, de RedHat et de Debian.

Introduction

Dans ce chapitre nous allons découvrir les principes généraux d'un système d'exploitation Linux.

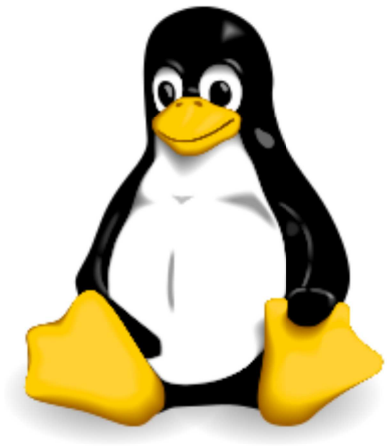
Introduction

- Présentation
- Historique
- Licence – Open source
- Distributions Linux
- Le Système d'exploitation

Introduction

Présentation

Bienvenue dans l'univers



LINUX

Présentation

Linux est un système d'exploitation. C'est à dire un logiciel qui permet de manipuler des fichiers, d'exécuter des programmes, ...etc via un ordinateur.

Pour utiliser ce système, nous disposons d'une interface graphique, ainsi qu'un terminal de commandes.

Linux appartient à la catégorie «**open source**», ce qui veut dire que son code source est disponible gratuitement par et pour les utilisateurs.

Nous retrouvons ce système d'exploitation principalement dans les entreprises, notamment pour gérer un serveur.

Introduction

Historique

1970 1991 1994 1996

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: Gcc-1.40 and a posix-question
Message-ID:
Date: 3 Jul 91 10:00:50 GMT

Hello netlanders,

Due to a project I'm working on (in minix), I'm interested in the posix
standard definition. Could somebody please point me to a (preferably)
machine-readable format of the latest posix rules? Ftp-sites would be
nice
```

Historique

Le système Linux vient du système UNIX.

Les dates importantes

1970 : Création d'Unics (UNIX) par Kenneth Thompson et Dennis Ritchie au sein des laboratoires Bell AT&T.
AT&T souhaite commercialiser son système.

1971 : 23 ordinateurs sont reliés à l'ARPANET. Ray Tomlinson envoi le premier courriel.

1972 : Dennis Ritchie crée le langage C (une évolution du langage B) rendant ainsi Unix portable sur différentes architectures physiques.

1973 : Définition du protocole TCP/IP.

1983 : Adoption du protocole TCP/IP. Premier serveur de noms (DNS).
L'université de Berkeley démarre le développement de Unix BSD.
AT&T prend le nom d'Unix System V.
Richard Stallman annonce le développement de GNU (Gnu is Not Unix) pour créer un système d'exploitation libre.

1985 : Richard Stallman créé la FSF (Free Software Foundation) pour s'assurer que tous les logiciels développés pour GNU restent libres.

1989 : Richard Stallman publie la première licence publique générale GNU.

1990 : Collaboration AT&T et SUN pour créer Unix AT&T System V.4.
Disparition d'ARPANET. Annonce du World Wide Web.

1991 : IBM, DEC et HP créent le groupement OSF (Open Software Foundation).
Démarrage de nombreux projets tel que FreeBSD.

Andrew Tanenbaum développe pour l'enseignement le système Minix. Il s'inspire d'Unix. Les sources sont disponibles mais ne sont pas libres.

Linus Thorvald décide de programmer un remplaçant à Minix qu'il appellera Linux. Le noyau a été publié sous licence GPL ce qui permet en le combinant aux outils GNU d'obtenir un système d'exploitation complet que l'on devrait appeler GNU/Linux au lieu de Linux.

1994 : Noyau Linux 1.0

1995 : Noyau Linux 1.2

1996 : Noyau Linux 2.0
Larry Ewing créé le symbole de linux le manchot **Tux.**
Matthias Ettrich créé le bureau **KDE.**

1997 : Miguel de Icaza créé le bureau **GNOME.**

1998 : Création de l'Open Source Initiative dédiée à la promotion de logiciels open source.

1999 : Entrée en Bourse de Redhat.
Noyau Linux 2.2.

2001 : Noyau Linux 2.4.

2003 : Acquisition de Suse par Novell.
Noyau Linux 2.6.0.

2012 : Noyaux Linux 3.2 LTS à 3.7.
Linus Thorvald optient le prix «Millennium Technology» remis par la Technology Academy Finland .

2015 : Noyaux Linux 3.19 à 4.3.

Premier Message envoyé par Linus Thorvald sur un système minix.

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)

Newsgroups: comp.os.minix
Subject: Gcc-1.40 and a posix-question
Message-ID:
Date: 3 Jul 91 10:00:50 GMT


Hello netlanders,


Due to a project I'm working on (in minix), I'm interested in the posix
standard definition. Could somebody please point me to a (preferably)
machine-readable format of the latest posix rules? Ftp-sites would be
nice
```

Message de Linus Thorvald annonçant l'inclusion de bash et de gcc dans son système.

```
From: torv...@klaava.Helsinki.FI (Linus Benedict Torvalds)

Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Keywords: 386, preferences
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT

Organization: University of Helsinki

Lines: 20


Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones.  This has been brewing
since april, and is starting to get ready.  I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want.  Any suggestions
are welcome, but I won't promise I'll implement them :-)

               Linus (torv...@kruuna.helsinki.fi)

PS.  Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```

Introduction

Licence – Open source

- Licence
- Open source
- GPL
- Copyleft

Licence – Open source

Licence

Une licence est un contrat permettant au titulaire des droits d'auteur, de définir les conditions d'accès à son programme (utilisation, modification et diffusion).

Open source

La désignation «**open source**» permet d'identifier un logiciel sur lequel s'applique une licence établie par l'**Open Source Initiative**.
C'est un logiciel qui à un code source et une distribution libre d'accès et sur lequel nous pouvons créer des travaux dérivés à partir de ce code.

Licence GPL

C'est une licence qui gère la législation ainsi que la distribution des logiciels libres provenant du projet **GNU**.
Elle fut créée par Richard Stallman, fondateur de la **Free Software Foundation**, qui est une organisation américaine pour la promotion du logiciel libre et la défense des utilisateurs.

La licence GPL s'appuie sur la notion de «**copyleft**», un clin d'œil au «**copyright**». Le «**copyleft**» est la liberté d'utiliser les codes sources et de les modifier. La contrainte est que toute adaptation réalisé est soumise à la même licence, donc l'obligation de mettre à disposition le code source.

Distributions Linux

Red Hat Entreprise Linux

Cette distribution commerciale à été développée par l'entreprise Red Hat.
Red Hat Entreprise Linux est, comme son nom l'indique, destinée aux entreprises.

Plusieurs distributions sont disponibles en fonction de leurs usage : versions serveurs d'entreprise (RHEL), version cloud, version poste de travail.

CentOS (Common ENTrepise Operating System)

Principalement destiné aux serveurs, cette distribution est un dérivé de Red Hat Entreprise Linux.
La première version de CentOS voit le jour en 2004 sur une base RHEL 2.1.

CentOS est une version gratuite de Linux Red Hat et le support est assuré par une communauté.

Debian

Debian est une distribution majeure dans le monde communautaire de Linux. Les distributions proposées sont non commerciales.

Ubuntu

Basée sur une distribution Linux Debian. Ubuntu est disponible sous une version commerciale, mais il existe également une distribution communautaire et grand public.

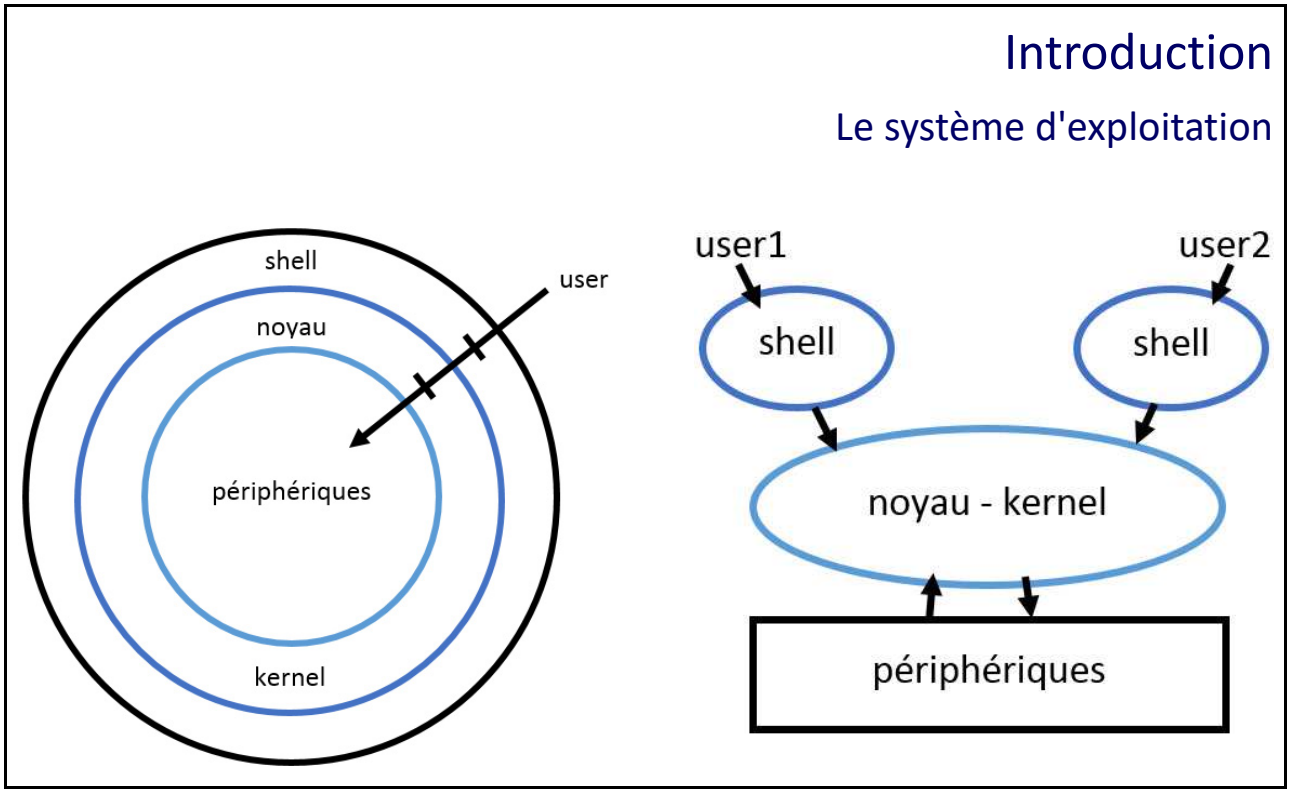
SUSE

Entreprise allemande du groupe Micro Focus International, elle a développé la distribution Linux «**SUSE Linux Entreprise**».
La première version est apparue en 1994, ce qui fait d'elle la plus ancienne distribution commerciale encore existante.

Compléments

Il existe un grand nombre de versions Linux ayant chacune leurs spécificités. Vous pourrez trouver facilement sur le web la liste complète et actualisée des différentes distributions Linux.

Certaines distributions sont publiées avec l'étiquette LTS (Long Time Support). Le distributeur assure ainsi que la distribution sera maintenue et supportée sur une certaine durée (5 ans usuellement).



Le système d'exploitation

Introduction

Un système d'exploitation (OS : Operating System) est ce qui démarre en premier sur un ordinateur. C'est un ensemble de programmes qui gèrent et sollicitent les capacités d'une machine. Il assure les interactions entre les ressources matérielles de l'ordinateur et un utilisateur via des applications.

Les périphériques

Un périphérique informatique est un composant de l'ordinateur.
Les flux de données sont nommés les flux d'Entrées/Sorties (ou I/O : Input/Output) vis à vis de l'unité centrale (UC ou processeur) .

Les périphériques d'entrées fournissent des flux d'entrés : clavier, souris, micro, scanner, …

Les périphériques de sorties fournissent des flux de sorties : écran, imprimante, hauts-parleurs, …

Les périphériques d'entrées/sorties fournissent les deux flux : disque, ram, ...

Le noyau

Le noyau ou kernel est le cœur du système. Il assure la gestion, l'administration et le contrôle de l'ensemble des fonctions d'un ordinateur.

Ses fonctions sont :

- L'administration des périphériques. Le noyau contient les drivers.
- La gestion des processus. Le noyau assure la communication inter processus via les appels systèmes.
- La gestion de l'exécution des commandes.

Le shell

Le shell est un programme, c'est un interpréteur de commandes. Cela correspond à une interface entre l'utilisateur et le système d'exploitation. Il y a au moins un shell par utilisateur connecté.

Son but est d'interpréter les lignes de commandes saisies par l'utilisateur, d'envoyer le résultat interprété au noyau pour exécution. Puis le shell récupérera le résultat de l'exécution de la commande du noyau pour l'affichage.

Il existe plusieurs type de shells :

- sh : Bourne Shell
- ksh : Korn Shell
- csh : C Shell
- bash : Bourne Again Shell

...etc

Un shell est unique à un utilisateur, il sera lancé lorsque l'utilisateur ouvrira une invite de commande (prompt).
Le prompt d'un utilisateur est caractérisé par le caractère «**$**». A l'exception de l'administrateur « root » qui est caractérisé par le caractère «**#**».

Un shell est paramétrable, ainsi l'environnement de l'utilisateur est personnalisable.

Notes

Premiers pas

Dans ce chapitre nous allons nous familiariser avec l'utilisation des commandes et obtenir de l'aide.

Premiers Pas

- Ouverture d'une session
- Commandes informatives
- Commande «man»

Premiers Pas

Ouverture d'une session

- Ouverture / fermeture de session
- Comptes utilisateur et root
- Login / Password
- Mode texte / mode graphique
- Mode locale / mode distant

Ouverture d'une session

Introduction

L'intérêt d'un système d'exploitation Linux est de pouvoir tout faire en ligne de commandes. Les interfaces graphiques ne sont pas nécessaires. Lorsque celles-ci sont présentes ce n'est que du confort pour l'utilisateur ou l'administrateur.

Pour des raisons de sécurité, il est installé en général que le strict minimum pour le bon fonctionnement d'un serveur Linux. Les interfaces graphiques ne sont donc pas installées, et nous avons uniquement les commandes à notre disposition.

Il est donc indispensable de maîtriser l'utilisation de la ligne de commandes.

Ouverture de session

L'ouverture de session c'est lorsque l'utilisateur se connecte sur son poste Linux.
On se connecte avec un compte utilisateur, c'est a dire que l'on diot s'authentifier avec un compte «**login**» et un mot de passe «**password**».

Fermeture de session

La fermeture de session en mode graphique s'effectue grâce à une interaction avec le bouton de fermeture de session.
Dans le mode texte, il faut saisir la commande «**exit**».

Comptes utilisateurs et root

Lors de la première connexion au système vous n'aurez qu'un seul compte disponible, celui de l'administrateur «**root**».
Ce compte vous donne tout les privilèges et droits pour le serveur Linux.
Le prompt de root est caractérisé avec le «#». Vous pourrez, par la suite, créer des comptes utilisateurs via le compte administrateur.

Login / Password

Le mot de passe est un système de sécurité pour vous connecter à votre session. Seules les personnes connaissant le mot de passe peuvent rentrer dans la session.

Login : c'est le champ à remplir pour indiquer votre nom d'utilisateur.
Exemple : utilisateur = user
administrateur = root

Mot de passe : c'est le champ à remplir pour vous authentifier avec votre «login».
Exemple : login = user
password = azerty

Mode texte / mode graphique

Le mode texte est un programme (shell) permettant d'ouvrir une session sur votre système Linux via des lignes de commandes.

Le mode graphique est un programme permettant d'ouvrir une session sur votre système Linux via une interface graphique.

Mode Local / Distant

Le mode local permet d'ouvrir une session via une interface texte (ligne de commande) ou une interface graphique.

Le mode distant permet de se connecter a votre machine à distance via le réseau. Il est recommander d'utiliser «**ssh**» qui est un protocole sécurisé.

Premiers Pas

Commandes informatives

- Informations sur les utilisateurs et le système

whoami	who am i	id	who	last
logname	finger	hostname	uname	
tty	date	cal		

Commandes informatives

- whoami : affiche le nom de l'utilisateur avec lequel vous êtes connecté à l'instant t.
- who am i : affiche le nom de l'utilisateur avec lequel vous vous êtes connecté.
- id : affiche les utilisateurs et groupes d'utilisateurs effectifs et réels.
- who : renseigne sur le nom des utilisateurs connectés, combien de temps ils sont restés connecté et le nom de l'hôte avec lequel ils se sont connectés.
- last : affiche la liste des dernières connexions des utilisateurs sur le poste.
- hostname : indique le nom du poste sur lequel vous êtes connecté.
- uname : affiche les informations du système.
- cal : affiche le calendrier.
- date : affiche la date.
- tty : visualise le nom du terminal dans lequel nous sommes.
- finger : permet d'avoir des informations sur un utilisateur.
- logname : affiche le nom d'utilisateur courant.

Exemples :

```
$ whoami
user1

$ who am i
user1    pts/2        2016-04-01 10:40 (spheriusform-pc.home)

$ tty
/dev/pts/2
```

Premier Pas

La commande «man»

- Manuel d'une commande

man commande

man section commande

Commande «man»

La commande «**man**» est une aide qui permet de visualiser le manuel d'une commande.

Syntaxe : man commande

Une fois le manuel de commande ouvert, voici ses principales sections :

- **Name :** nom de la commande et son descriptif court,
- **Synopsis :** la syntaxe de la commande,
- **Description :** la description complète de la commande,
- **Options :** la description complète de chaque options,
- **See Also :** «voir aussi» d'autres commandes en rapport avec celle qui est consultée.

Exemple :

```
$ man id

ID(1)                Manuel de l'utilisateur Linux             ID(1)
NOM
id - Afficher les UIDs et GIDs effectifs et réels
etc ...
```

Interactivité

L'interactivité dans le «**man**» est défini avec des touches du clavier, principalement pour se déplacer au sein de l'aide de la commande ou pour réaliser des recherches.

Raccourci	Action
flèches directionnelles	Permet de naviguer dans la page
espace	Afficher la page suivante
entrée	Afficher la ligne suivante
b	Remonter d'une page
q	Quitter

/	Rechercher en avant
?	Rechercher en arrière
n	Allez à l'occurrence suivante
N	Allez à l'occurrence précédente
h	Afficher l'aide

Sections

Il existe différentes sections pour agencer les pages de manuel.

1. Aide des commandes
2. Appels système
3. Les librairies
4. Fichiers spéciaux
5. Formats des fichiers
6. Jeux
7. Divers
8. Commandes d'administration du système
9. Les routines du noyau

Options utiles

-s (section) : permet d'indiquer la section où chercher les pages de manuel. Il est possible de chercher dans plusieurs sections en les séparant par des virgules.

Exemple :

```
$ man -s 5 passwd
$ man -s 1,5 passwd
```

Variante :

```
$ man 5 passwd
```

-L (locale) : permet de spécifier les paramètres régionaux pour l'affichage de la page de manuel. Pour afficher la page de manuel «**man**» en anglais taper cette commande :

Exemple :

```
$ man      -L  en     ls
```

Complément

apropos : permet de trouver une commande dont vous ne connaissez pas le nom. Il suffit d'entrer un mot clé à la suite de cette commande, puis celle ci cherchera toutes les commandes ayant ce mot clé dans leur description.

Exemple :

```
$ apropos      sound
esd (1)              - Le démon de son éclairé (Enlightened Sound Daemon)
alsactl (1)          - advanced controls for ALSA soundcard driver
alsaunmute (1)       - a simple script to initialize ALSA sound devices
amixer (1)           - command-line mixer for ALSA soundcard driver
... etc
```

Notes

Manipuler l'arborescence

Dans ce chapitre nous allons nous familiariser avec la manipulation de l'arborescence.

Manipuler l'arborescence

- Tout est Fichier
- L'arborescence
- Chemin Absolu
- Chemin Relatif
- Quelques Commandes
- Lister l'arborescence
- Création de fichiers et de répertoires
- Suppression de fichiers et de répertoires

Manipuler l'arborescence

Tout est fichier – Tout est numéro

- Fichier – Répertoire – Sous Répertoire
- Type de fichier : Répertoire
  Ordinaire
  Lien
- Attention à la casse
- Extension

Tout est Fichier

Un Système Fichier c'est une arborescence avec des fichiers, des répertoires et des sous répertoires.

Nous avons donc trois principaux types de fichiers :

- Fichier de type Répertoire : ce sont les répertoires et les sous répertoires
- Fichier de type Ordinaire : ce sont des fichiers «classiques» comme des fichier sons, des fichier vides, des images, des vidéos, etc …
- Fichier de type Lien : ce sont des raccourcis dans l'arborescence.

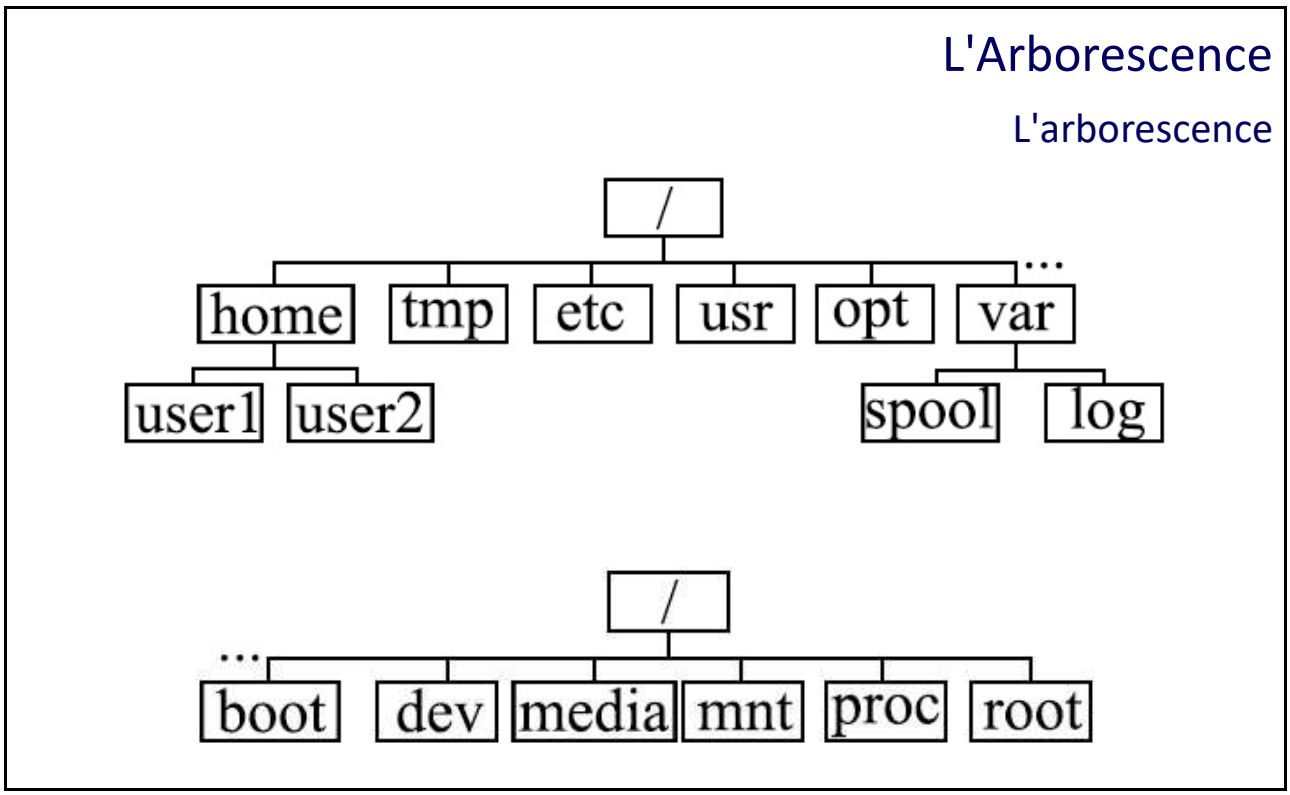
La casse

Dans un système d'exploitation comme «Windows» il est impossible d'avoir deux fichiers portant le même nom dans le même répertoire.
Dans le système Linux, il est possible d'avoir plusieurs fichiers de même nom du moment qu'un caractère est différencier par une majuscule ou une minuscule.

Par exemple : **fichier1** peut être dans le même répertoire que **Fichier1** et **FICHIER1**.

Extension

Dans le système Linux, il n'est pas utile d'avoir une extension pour le type de fichier.
Un fichier vidéo se nommera «**video1**» et non video1.avi (ou .mpeg, .mkv, etc.).



L'arborescence

Liste des répertoires spécifiques à un système d'exploitation Linux :

/home : contient les répertoires personnels des utilisateurs (home directory).

/tmp : contient les fichiers temporaires.

/etc : contient les fichiers de configuration du système et de certaines applications.

/usr : contient les commandes et les aides (man).

/opt : répertoire où l'on installe les logiciels commerciaux.

/var : contient les fichiers de taille variable.

/var/spool : répertoire de spool. Il contient les données en attente pour un traitement futur (impression, crontab, …).

/var/log : pour les fichiers de journalisations, de logs, des fichiers de suivi.

/boot : contient les fichiers nécessaires à la première phase de la séquence de démarrage.

/dev : contient les fichiers spéciaux des périphériques (devices).

/media: répertoire utilisé pour accéder aux médias amovibles (cd, usb, …).

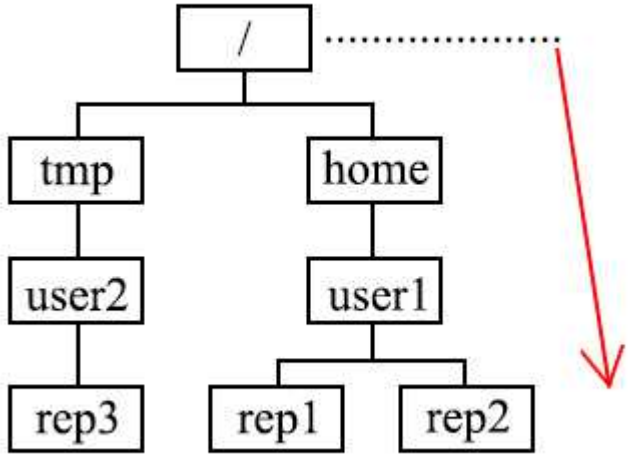
/mnt : répertoire vide par défaut, il est réservé à root.

/proc : contient les fichiers relatifs aux processus et à l'état du système.

/root : contient les fichiers personnels de l'administrateur (root).

Manipuler l'arborescence

Chemin Absolu



Chemin Absolu

Dans l'arborescence Linux, tout chemin absolu commence par la racine «**/**».
Le chemin absolu montre le chemin complet depuis la racine jusqu'à un fichier donné.

La commande «pwd»

pwd : affiche le chemin absolu du répertoire sur lequel on est positionné.

Exemple :

```
$ pwd
/home/user1/Bureau
```

La commande «cd»

cd : permet de ce déplacer dans l'arborescence.

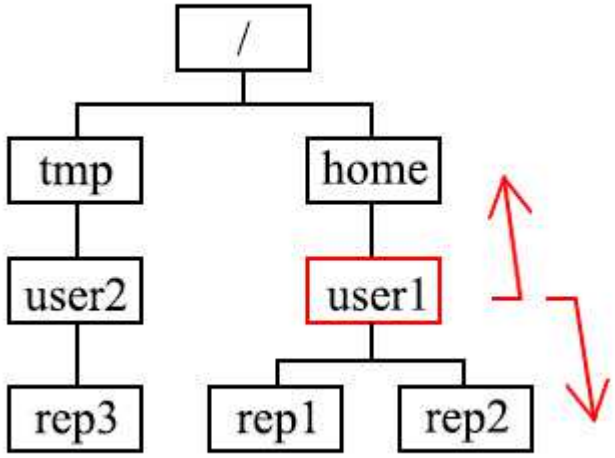
Exemple :

```
$ pwd
/home/user1/Bureau
$ cd /etc
$ pwd
/etc
```

Tous les chemins ci-dessus sont des chemins absolus car ils commencent tous par un «**/**».

Manipuler l'arborescence

Chemin Relatif



Chemin Relatif

Contrairement au chemin absolu, le chemin relatif ne commence pas par la **racine**.
Le chemin relatif montre le chemin du répertoire sur lequel on est positionné jusqu'à un fichier donné.

Cas particulier : Le «**..**» représente le répertoire père.
Le «**.**» représente le répertoire courant.

Quelques exemples :

1) A partir de **/home/user1** pour aller dans **rep1** : cd **rep1**

2) A partir de **/home** pour aller dans **rep1** : cd **user1/rep1**

3) A partir de **/home/user1** pour aller dans **/home** : cd **..**

4) A partir de **/home/user1** pour aller dans **/tmp** : cd **../../tmp**

Manipuler l'arborescence

Quelques commandes

- Répertoire de Connexion
- Commande «cd»
  - cd rep
  - cd ~
  - cd -

basename - dirname - whereis - which

Quelques Commandes

Répertoire de Connexion

C'est le répertoire où l'user est positionné lorsqu'il ouvre une session.
Ce répertoire lui appartient et c'est à partir de celui-ci qu'il peut créer sa propre arborescence, c'est son répertoire de travail (HOME DIRECTORY).

Commandes

cd : permet de ramener l'user dans son répertoire de connexion.
cd rep : utilisation usuelle, pour se déplacer sur le répertoire «rep».
cd ~ : permet de ramener l'user dans son répertoire de connexion.
cd ~Jean : pour se positionner sur le répertoire de connexion de l'utilisateur Jean.
cd - : pour se positionner sur le répertoire précédent.

basename : permet d'éliminer le chemin d'accès d'un fichier.

```
$ basename /rep1/rep2/rep3/fichier
fichier
```

dirname : permet de conserver la partie répertoire d'un chemin d'accès.

```
$ dirname  /rep1/rep2/rep3/fichier
/rep1/rep2/rep3
```

whereis : permet de rechercher les fichiers exécutables, les pages de manuel et les sources d'une commande.

```
$ whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
```

which : permet de déterminer le chemin absolu du positionnement du fichier exécutable d'un programme (en respectant la variable PATH).

```
$ which cd
/usr/bin/cd
```

Manipuler l'arborescence

Lister l'arborescence

- Commande «ls»
  - options : -l -a -d -R -r -F -i
  - ls -ld rep
  - ls -lrt rep
  - ll

Lister l'arborescence

La commande «ls»

La commande «**ls**» permet de lister les fichiers depuis votre emplacement dans l'arborescence.

Les options de «ls»

-l : permet un affichage détaillé (permissions, nom du propriétaire, ...etc).
-a : affiche tous les fichiers du répertoire.
-d : affiche un répertoire de la même façon qu'un fichier.
-R : affiche le contenu des sous-répertoires récursivement.
-r : inverse le tri du contenu.
-F : ajoute un caractère de distinction à la fin des fichiers.
«*» : fichiers exécutables.
«/» : les répertoires.
«@» : liens symboliques.
-i : affiche l'inode du fichier.

Exemple :

```
$ ls        /home/user1
Bureau  Documents  essai  fic1  fic2  fic3  folder1  Images  Modèles  Musique
Public  rep1  rep2  rep3  Téléchargements  Vidéos
```

```
$ ls        -l          /home/user1
total 0
drwxr-xr-x. 3 user1 user1 77 11 avril 13:48 Bureau
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Documents
drwxrwxr-x. 2 user1 user1 21 29 mars  15:59 essai
-rw-rw-r--. 1 user1 user1  0 29 mars  09:59 fic1
-rw-rw-r--. 1 user1 user1  0 29 mars  09:59 fic2
-rw-rw-r--. 1 user1 user1  0 29 mars  09:59 fic3
```

```
$ ls        -a          /home/user1
.   .bash_history  .bash_profile  Bureau  .config    .esd_auth  fic1  fic3
..  .bash_logout   .bashrc        .cache  Documents  essai      fic2  folder1
```

```
$ ls        -la
total 128
drwx------. 19 user1 user1    4096 28 avril 10:00 .
drwxr-xr-x.  5 root  root       42 30 mars  14:06 ..
-rw-------.  1 user1 user1    1894 25 avril 17:24 .bash_history
-rw-r--r--.  1 user1 user1      18 10 juin   2014 .bash_logout
-rw-r--r--.  1 user1 user1     193 10 juin   2014 .bash_profile
-rw-r--r--.  1 user1 user1     240 29 mars  09:25 .bashrc
drwxr-xr-x.  2 user1 user1      33 21 avril 11:05 Bureau
drwxr-xr-x.  2 user1 user1       6 25 mars  12:16 Documents
-rw-r--r--.  1 user1 groupe1   592 22 avril 10:55 fichier1
-rw-r--r--.  1 user1 groupe1    78 22 avril 13:28 fichier2
-rw-r--r--.  1 user1 groupe1    57 22 avril 15:19 fichier3
```

```
$ ls        -Rl
.:
total 68
drwxr-xr-x. 2 user1 user1    33 21 avril 11:05 Bureau
drwxr-xr-x. 2 user1 user1     6 25 mars  12:16 Documents
drwxrwxr-x. 2 user1 user1    21 29 mars  15:59 essai
-rw-r--r--. 1 user1 groupe1 592 22 avril 10:55 fichier1
-rw-r--r--. 1 user1 groupe1  78 22 avril 13:28 fichier2
-rw-r--r--. 1 user1 groupe1  57 22 avril 15:19 fichier3
drwxrwxr-t. 3 user1 user1    26 30 mars  14:51 folder1

./Bureau:
total 0
-rw-r--r--. 1 user1 groupe1 0 20 avril 11:16 fic

./Documents:
total 0

./essai:
total 0
-rwxrwxrwx. 1 user1 user1 0 29 mars  15:59 fichier1

./folder1:
total 0
-rwsr-sr-x. 1 user1 groupe1 0 30 mars  14:45 fic
drwxrwxrwt. 2 user1 groupe1 6 30 mars  14:51 rep

./folder1/rep:
total 0
```

```
$ ls        -rl
total 68
drwxrwxr-t. 3 user1 user1    26 30 mars  14:51 folder1
-rw-r--r--. 1 user1 groupe1  57 22 avril 15:19 fichier3
-rw-r--r--. 1 user1 groupe1  78 22 avril 13:28 fichier2
-rw-r--r--. 1 user1 groupe1 592 22 avril 10:55 fichier1
drwxrwxr-x. 2 user1 user1    21 29 mars  15:59 essai
drwxr-xr-x. 2 user1 user1     6 25 mars  12:16 Documents
drwxr-xr-x. 2 user1 user1    33 21 avril 11:05 Bureau
```

```
$ ls        -F
Documents/                fichier2            folder1/
essai/                    fichier3*           lien1@
fichier1*                 fichier4            Modèles/
```

```
$ ls        -li
total 68
 69770208 drwxr-xr-x. 2 user1 user1    33 21 avril 11:05 Bureau
 69770209 drwxr-xr-x. 2 user1 user1     6 25 mars  12:16 Documents
202870049 drwxrwxr-x. 2 user1 user1    21 29 mars  15:59 essai
 73289329 -rwxrwxrwx. 1 user1 groupe1 592 22 avril 10:55 fichier1
 73289328 -rw-r--r--. 1 user1 groupe1  78 22 avril 13:28 fichier2
 73289317 -rwxrwxrwx. 1 user1 groupe1  57 22 avril 15:19 fichier3
 73289330 -rw-r--r--. 1 user1 groupe1   0 28 avril 11:15 fichier4
 72927746 drwxrwxr-t. 3 user1 user1    26 30 mars  14:51 folder1
```

```
$ ls        -ld         /home/user1
drwx------. 19 user1 user1 4096 26 avril 12:41 /home/user1
```

```
$ ls        -lrt        /home/user1
total 0
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Téléchargements
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Modèles
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Vidéos
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Public
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Musique
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Images
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Documents
-rw-rw-r--. 1 user1 user1  0 29 mars  09:59 fic2
-rw-rw-r--. 1 user1 user1  0 29 mars  09:52 fic1
-rw-rw-r--. 1 user1 user1  0 29 mars  09:50 fic3
```

```
$ ll        /home/user1
total 0
drwxr-xr-x. 3 user1 user1 77 11 avril 13:48 Bureau
drwxr-xr-x. 2 user1 user1  6 25 mars  12:16 Documents
drwxrwxr-x. 2 user1 user1 21 29 mars  15:59 essai
-rw-rw-r--. 1 user1 user1  0 29 mars  09:59 fic1
-rw-rw-r--. 1 user1 user1  0 29 mars  09:59 fic2
-rw-rw-r--. 1 user1 user1  0 29 mars  09:59 fic3
```

Manipuler l'arborescence

Création de fichiers et répertoires

- touch fichier1 fichier2
- mkdir rep1 rep2
  - option -p

Création de fichiers et de répertoires

La commande «touch»

Plusieurs commandes permettent de créer des fichiers ordinaires, comme la commande «**touch**».

Exemples : touch fic1

touch fic1 fic2 fic3

La commande «mkdir»

Pour créer des répertoires, on utilise la commande «**mkdir**».
L'option -p permet de créer les répertoires intermédiaires s'ils n'existent pas.

Exemples : mkdir rep1

mkdir rep1 rep2 rep3

mkdir -p rep1/srep/autre

Manipuler l'arborescence

Supprimer des fichiers et répertoires

- rm fichier1 fichier2
  - Options -i
    -r
    -f
- rmdir rep1 rep2

Supprimer de fichiers et de répertoires

La commande «rm»

La commande « rm » supprime les fichiers, que ce soit des fichiers ordinaires où des répertoires.

Exemples : rm fic1
rm fic1 fic2 fic3

Les options de «rm»

-i : demande à l'utilisateur de confirmer la suppression du fichier.
Exemple : rm -i fic*

-r : supprime des répertoires. Attention, cette suppression est récursive.
Donc le répertoire est détruit même s'il n'est pas vide.
Exemple : rm -r rep1

-f : force la suppression du fichier.
Exemple : rm -f fic1 fic2 fic3

La commande «rmdir»

Cette commande supprime des répertoires s'ils sont vide.

Manipuler l'arborescence

Copier, déplacer et renommer des fichiers

- cp source destination
  - -i
  - -r
- mv source destination

Copier, déplacer et renommer des fichiers

Copie de fichiers

Pour copier des fichiers, il faut utiliser la commande «**cp**».

Les options de «cp»

-i : interroge l'utilisateur avant de copier le fichier.
-r : pour une copie récursive des répertoires.

Exemples :	cp	fic1	fic2			Copie le contenu de **fic1** dans **fic2**.
	cp	fic1	rep			Copie le fichier **fic1** dans **rep**.
	cp	fic1	rep/fic2			Copie le contenu de **fic1** dans le fichier **fic2** dans le répertoire **rep**.
	cp	fic1	fic2	fic3	rep	Copie les fichiers **fic1, fic2, fic3** dans le répertoire **rep**.
	cp	-i	fic1	fic2		
	cp	-r	rep1	rep2		

Déplacer et/ou renommer des fichiers

Pour déplacer et/ou renommer des fichiers, il faut utiliser la commande «**mv**».

Exemples :	mv	fic1	fic2			Renomme **fic1** en **fic2**.
	mv	fic1	rep			Déplace **fic1** dans **rep**.
	mv	fic1	rep/fic2			Déplace **fic1** dans **rep** et le renomme en **fic2**.
	mv	fic1	fic2	fic3	rep	Déplace **fic1**, **fic2**, **fic3** dans **rep**.
	mv	rep1	rep2			Déplace le répertoire **rep1** dans **rep2**.

Manipuler l'arborescence

Visualiser un fichier

- cat fichier
- more fichier
- less fichier
- nl fichier
- tac fichier

Visualiser un fichier

Plusieurs commandes permettent la visualisation du contenu d'un fichier.

La commande «cat»

La commande «**cat**» nous permet d'afficher le contenu d'un fichier.

Syntaxe : cat fic1

La commande «more»

La commande «**more**» affiche le contenu du fichier page par page.
Le déplacement au sein de ce fichier est effectué grâce aux sous commandes du tableau :

Raccourci	Action
flèches directionnelles	Permet de naviguer dans la page
espace	Afficher la page suivante
entrée	Afficher la ligne suivante
b	Remonter d'une page
q	Quitter

/	Rechercher en avant
?	Rechercher en arrière
n	Allez à l'occurrence suivante
N	Allez à l'occurrence précédente
h	Afficher l'aide

Syntaxe : more fic1

La commande «less»

Similaire à la commande «**more**», la commande «**less**» nous permet de visualiser un fichier page par page.
L'avantage de cette commande est sa rapidité d'exécution pour les gros fichiers. Elle ne charge pas entièrement le fichier en mémoire, ce qui implique une ouverture plus rapide.

Syntaxe : less fic1

La commande «nl»

Cette commande affiche le contenu du fichier avec la numérotation des lignes.

Syntaxe : cat fic1
première ligne.
deuxième ligne.

nl fic1
1 première ligne.
2 deuxième ligne.

La commande «tac»

La commande «**tac**» est l'inverse de la commande «**cat**». Elle affiche le contenu d'un fichier en commençant par la dernière ligne.
Intéressant pour les gros fichiers.

Syntaxe : cat fic1
première ligne.
deuxième ligne.

tac fic1
deuxième ligne.
première ligne.

Manipuler l'arborescence

Les liens

Lien = Raccourci dans l'arborescence

- Les inodes
- liens physiques ln fichier lien1
- liens symboliques ln -s fichier lien2

Les liens

Un lien est un fichier permettant d'accéder à un autre fichier.
Il est utilisé pour créer des raccourcis au sein de l'arborescence.
Il existe deux types de liens : les liens physiques et les liens symboliques.

Les inodes

Un disque est découpé en différentes zones appelées partitions. Un système de fichiers est créé au sein d'une partition, et permet d'utiliser la notion de fichiers, répertoires et sous répertoires.
Le système de fichiers est structurée avec une partie pour les données (fichiers et répertoires) et une partie pour les métadonnées (l'organisation interne du système de fichiers).
Au sein des métadonnées est localisée la table des inodes pour le système de fichiers.

Une inode est un numéro unique du système de fichiers pour caractériser un fichier linux.
L'inode contient toutes les informations sur un fichier (tout sauf le contenu du fichier).
Lorsque l'on manipule un fichier (par exemple par la commande cat), c'est l'inode qui est sollicitée en premier, afin de récupérer les informations sur le fichier et surtout les informations localisant ce fichier dans la partie des données du système de fichiers.
Pour afficher une partie des données d'une inode, on utilise la commande « ls -l».

contenu d'une inode :

```
$ ls        -li          /home/user1/fichier1
12256   -rw-rw-r--.   1   user1   groupe1   10     29 mars  09:59   fichier1
```

l'inode 12256 définie le fichier « fichier1 » du répertoire « /home/user1 », elle contient les informations suivantes :

- le type du fichier, les permissions, le nombre de liens physiques,
- le uid propriétaire, le gid propriétaire, la taille,
- 3 dates : dernière accès à l'inode, dernier accès au fichier, dernière modification du fichier,
- la liste des adresses des blocs de données du contenu du fichier.

Liens physiques

Un lien physique a la même inode que le fichier d'origine.

Un lien physique est un nom supplémentaire pour un fichier. Cela permet de créer un ficher accessible par deux noms différents.

Syntaxe : ln fichier lien1

Si l'on supprime le fichier «**fic**», «**lien1**» restera accessible et inversement. Le lien physique de dépend plus du fichier d'origine pour accéder aux données.
Ces fichiers doivent être au sein du même système de fichiers.

Liens symboliques

Un lien symbolique a une inode différente de celle du fichier d'origine.

Un lien symbolique est un «raccourci» d'un fichier dans l'arborescence. Ce type de lien contient la référence du chemin pour accéder au fichier d'origine, en fait l'inode du fichier d'origine. Ainsi si le fichier d'origine est supprimé, son lien ne fonctionnera plus. Dans ce cas là, on dit que le lien est cassé.

Syntaxe : ln -s fichier lien2

Le lien peut se trouver au sein d'un autre système de fichiers.

Notes

Variables et métacaractères

Dans ce chapitre nous allons nous familiariser avec les variables et les métacaractères.

Variables et métacaractères

- Les variables
- Les variables locales et globales
- Les métacaractères
- Les métacaractères - suite

Variables et métacaractères

Les variables

- echo $variable
- env
- Quelques variables
  - HOME LOGNAME
  - PATH HOSTNAME
  - PS1 LANG

Les variables

Les variables contiennent des chaînes de caractères qui permettent de stocker, d'une manière temporaire, des informations en mémoire.
Pour afficher le contenu d'une variable : echo $variable

Variables d'environnements

Les variables d'environnement permettent de configurer le système et un programme.
La commande «**env**» affiche la liste des variables situées dans l'environnement.

Quelques exemples de variables système

HOME	:	affiche le nom du répertoire de connexion.
PATH	:	liste les répertoires où le système va rechercher les commandes.
PS1	:	affiche le prompt principal.
LANG	:	affiche la langue du système.
LOGNAME	:	affiche le nom de l'utilisateur.
HOSTNAME	:	affiche le nom de l'hôte du système.

Variables et métacaractères

Les variables locales et globales

- set / unset
- Variables locales / Variables globales
  - export variable=valeur

Les variables locales et globales

set / unset

La commande «**set**» affiche l'ensemble des variables qu'elles soient dans environnement ou non.
La commande «**unset**» annule la définition d'une variable.

Exemple :

```
$  var=bonjour
$  echo  $var
bonjour

$  unset var
$  echo  $var
--- affiche rien ---
```

Variables locales / Variables globales

Les variables globales sont transmises (dupliquées) dans le «**sous-shell**» contrairement aux variables locales.

Ainsi, l'exploitation d'une variable locale ne peut être faite qu'au niveau du shell dans lequel elle a été créée.
La valeur d'une variable globale est exploitable au sein de sous-shells. La commande «**export**» définie une variable comme globale.

Aucune variable n'est transmise au shell père.

Par convention, les variables locales sont notées en minuscules et les variables globales en majuscules.

Exemple de variable locale : village=Salon

Exemple de variable globale : export VILLE=«Aix en Provence»
ou
VILLE=«Aix en Provence»
export VILLE

Exemple :

```
$  village=Salon                          variable locale
$  export  VILLE=«Aix en Provence»        variable globale
$  echo  $village
Salon
$  echo  $VILLE
Aix en Provence

$  bash                                   un sous-shell

$  echo  $village
--- affiche rien ---
$  echo  $VILLE
Aix en Provence

$  VILLE=Toulouse
$  echo  $VILLE
Toulouse

$  sante=Vacances                         variable locale au sous-shell
$  export  PAYS=France                    variable globale au sous-shell
$  echo  $sante
Vacances
$  echo  $PAYS
France

$  exit                                   retour au shell père

$  echo  $village
Salon
$  echo  $VILLE
Aix en Provence
$  echo  $sante
--- affiche rien ---
$  echo  $PAYS
--- affiche rien ---
```

Variables et métacaractères

Les métacaractères

- **Caractère spécial du shell**
- "texte"
- 'texte'
- `commande` ou $(commande)
- \x

Métacaractères

Les métacaractères sont des caractères qui ont une signification spéciale au sein du shell.

Métacaractère	Signification
" "	Cela permet de prendre l'ensemble de la chaîne de caractères qu'il y a entre guillemets comme un seul argument. Les caractères spéciaux au sein des guillemets sont interprétés.
' '	Cela permet de prendre l'ensemble de la chaîne de caractères qu'il y a entre simple côte comme un seul argument mais les caractères spéciaux sont neutralisés.
`cmd` **ou $(cmd)**	La chaîne de caractères entre côte inverse est exécutée comme une commande. On récupère donc le résultat de cette commande.
\x	Le backslash change la signification du caractère suivant (x). Cela signifie que si ce caractère est un caractère spécial, il est neutralisé. Par contre si c'est un caractère lambda, il peut devenir un caractère spécial. Exemple : \$ Le caractère dollar n'a plus sa signification spéciale. \t Représente une tabulation. \n Représente un saut de ligne.

Exemples :

```
$ echo " Répertoire $HOME=$HOME et date "
 Répertoire /home/user1=/home/user1 et date

$ echo ' Répertoire $HOME=$HOME et date '
 Répertoire $HOME=$HOME et date

$ echo " Répertoire $HOME=$HOME et `date` "
 Répertoire /home/user1=/home/user1 et mar. avril 26 16:52:38 CEST 2016

$ echo " Répertoire $HOME=$HOME et $(date) "
 Répertoire /home/user1=/home/user1 et mar. avril 26 16:53:12 CEST 2016

$ echo " Répertoire \$HOME=$HOME et `date` "
 Répertoire $HOME=/home/user1 et mar. avril 26 16:54:00 CEST 2016

$ echo -e " Répertoire \$HOME=$HOME \n et\t\t\t`date` "
 Répertoire $HOME=/home/user1
 et mar. avril 26 16:54:00 CEST 2016
```

Variables et métacaractères

Les métacaractères - suite

- **Caractère spécial du shell**

- * 0 à n caractères quelconques
- ? 1 caractère quelconque
- [abc] Liste de caractères
- [a-z] Intervalle de caractères
- [!abc] Exclusion d'une liste de caractères
- [!a-z] Exclusion d'un intervalle de caractères

Métacaractères - suite

Métacaractère	Signification
*****	Représente 0 à n caractères quelconques.
?	Représente 1 caractère quelconque.
[abc]	Représente un caractère parmi la liste entre [ ].
[a-z]	Représente un caractère parmi l'intervalle de caractères (utilisation du tiret). Remarques : [a-z] pour un caractère minuscule. [A-Z] pour un caractère majuscule. [a-zA-Z] pour un caractère minuscule ou majuscule. [0-9] pour un chiffre.
[!abc]	Représente un caractère quelconque sauf ceux de la liste.
[!a-z]	Représente un caractère quelconque sauf ceux de l'intervalle.

Exemples :

```
$ ls
fic1   fic2   fic3   fic4   fic5
fic6   fic7   fic8   fic9   fic10
fic11  fic12  fic20  ficAA  fic0z
fichier   file  foret  autre  nouveau  liste


$ ls fi*
fic1   fic2   fic3   fic4   fic5
fic6   fic7   fic8   fic9   fic10
fic11  fic12  fic20  ficAA  fic0z
fichier   file


$ ls fic?
fic1   fic2   fic3   fic4   fic5
fic6   fic7   fic8   fic9


$ ls ?i*
fic1   fic2   fic3   fic4   fic5
fic6   fic7   fic8   fic9   fic10
fic11  fic12  fic20  ficAA  fic0z
fichier   file  liste


$ ls fic??
fic10  fic11  fic12  fic20  ficAA  fic0z


$ ls fic[0-9][0-9]
fic10  fic11  fic12  fic20


$ ls fic[!0-9][!0-9]
ficAA


$ ls ?[!i]*
foret  autre  nouveau
```

Notes

Manipulation du contenu d'un fichier

Dans ce chapitre nous allons nous familiariser avec la manipulation du contenu d'un fichier.

Manipulation du contenu d'un fichier

- Les commandes «wc», «head» et «tail»
- Les commandes «file, «strings» et «od»
- Les commandes «cmp», «diff» et «paste»
- Les commandes «cut» et «awk»
- La commande «sort»
- Les commandes «uniq» et «tr»

Manipulation du contenu d'un fichier

Les commandes «wc», «head» et «tail»

- wc [-cwl] fichier
- head [-n] fichier
- tail [-n] fichier

Les commandes «wc», «head» et «tail»

La commande «wc»

Cette commande nous permet d'afficher des informations sur un fichier comme : le nombre d'octets, le nombre de mots et le nombre de lignes.

Plusieurs options sont disponibles pour cette commande.

-c : affiche le nombre d'octets.
-w : affiche le nombre de mots.
-l : affiche le nombre de lignes.

Exemples :

```
$ wc        /etc/passwd
40     68     1993   /etc/passwd

$ wc   -l   /etc/passwd
40     /etc/passwd
```

La commande «head»

Cette commande nous permet d'afficher les premières lignes d'un fichier, par défaut les 10 premières.
Nous pouvons personnaliser le nombre de lignes à afficher en positionnant l'option «**-n**» (n étant le nombre de lignes).

Exemple :

```
$ head     -5    /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

La commande «tail»

Cette commande permet d'afficher les dernières lignes d'un fichier, par défaut les 10 dernières.
Comme la commande «**head**» nous pouvons personnaliser le nombre de lignes à afficher grâce à l'option «**-n**».

Exemple :

```
$ tail     -5    /etc/passwd
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
user1:x:1000:1003:user1:/home/user1:/bin/bash
user2:x:1001:1004::/home/user2:/bin/bash
user3:x:1002:1005::/home/user3:/bin/bash
```

Manipulation du contenu d'un fichier

Les commandes «file», «strings» et «od»

- file fichier
- strings fichier
- od [-odxc] fichier

Les commandes «file», «strings» et «od»

La commande «file»

Cette commande permet de déterminer le type du contenu d'un fichier.

Syntaxe : file fic1

Exemple :

```
$ file /etc/*
/etc/abrt:          directory
/etc/adjtime:       ASCII text
/etc/aliases:       ASCII text
/etc/alsa:          directory
```

La commande «strings»

Cette commande permet d'afficher, sur la console, les caractères imprimables d'un fichier.

Syntaxe : strings fic1

La commande «od»

Cette commande permet d'afficher le contenu d'un fichier sous plusieurs formats (octal, hexadécimal, etc ...)

Exemple : Par défaut, l'affichage est en octal.

```
$ od        /etc/passwd
0000000 067562 072157 074072 030072 030072 071072 067557 035164
0000020 071057 067557 035164 061057 067151 061057 071541 005150
0000040 064542 035156 035170 035061 035061 064542 035156 061057
0000060 067151 027472 061163 067151 067057 066157 063557 067151
0000100 062012 062541 067555 035156 035170 035062 035062 060544
0000120 066545 067157 027472 061163 067151 027472 061163 067151
..etc
```

Il existe plusieurs options pour cette commande :

-o : permet d'afficher un contenu en octal.
-d : permet d'afficher un contenu en décimal.
-x : permet d'afficher un contenu en hexadécimal.
-c : permet d'afficher un contenu en ascii.

Exemple :

```
$ od        -xc         /etc/passwd
0000000    6f72    746f    783a    303a    303a    723a    6f6f    3a74
          r   o   o   t   :   x   :   0   :   0   :   r   o   o   t   :
0000020    722f    6f6f    3a74    622f    6e69    622f    7361    0a68
          /   r   o   o   t   :   /   b   i   n   /   b   a   s   h  \n
0000040    6962    3a6e    3a78    3a31    3a31    6962    3a6e    622f
          b   i   n   :   x   :   1   :   1   :   b   i   n   :   /   b
0000060    6e69    2f3a    6273    6e69    6e2f    6c6f    676f    6e69
          i   n   :   /   s   b   i   n   /   n   o   l   o   g   i   n
0000100    640a    6561    6f6d    3a6e    3a78    3a32    3a32    6164
         \n   d   a   e   m   o   n   :   x   :   2   :   2   :   d   a
0000120    6d65    6e6f    2f3a    6273    6e69    2f3a    6273    6e69
          e   m   o   n   :   /   s   b   i   n   :   /   s   b   i   n
0000140    6e2f    6c6f    676f    6e69    610a    6d64    783a    333a
          /   n   o   l   o   g   i   n  \n   a   d   m   :   x   :   3
0000160    343a    613a    6d64    2f3a    6176    2f72    6461    3a6d
          :   4   :   a   d   m   :   /   v   a   r   /   a   d   m   :
0000200    732f    6962    2f6e    6f6e    6f6c    6967    0a6e    706c
          /   s   b   i   n   /   n   o   l   o   g   i   n  \n   l   p
0000220    783a    343a    373a    6c3a    3a70    762f    7261    732f
          :   x   :   4   :   7   :   l   p   :   /   v   a   r   /   s
0000240    6f70    6c6f    6c2f    6470    2f3a    6273    6e69    6e2f
          p   o   o   l   /   l   p   d   :   /   s   b   i   n   /   n
0000260    6c6f    676f    6e69    730a    6e79    3a63    3a78    3a35
          o   l   o   g   i   n  \n   s   y   n   c   :   x   :   5   :
0000300    3a30    7973    636e    2f3a    6273    6e69    2f3a    6962
          0   :   s   y   n   c   :   /   s   b   i   n   :   /   b   i
0000320    2f6e    7973    636e    730a    7568    6474    776f    3a6e
          n   /   s   y   n   c  \n   s   h   u   t   d   o   w   n   :
...etc
```

Manipulation du contenu d'un fichier

Les commandes «cmp», «diff» et «paste»

- cmp fic1 fic2

```
$ cmp fic1 fic2
fic1 fic2 sont différents: octet27, ligne2
```

- diff fic1 fic2

```
1c1,5 < texte
6a5 ------
3d3 > texte
```

- paste fic1 fic2

```
$ paste fic1 fic2
1 Jean-Marc 5 Responsable
5 Robin 8 Stagiaire
```

Les commandes «cmp», «diff» et «paste»

La commande «cmp»

Cette commande permet de comparer deux fichiers, elle indique si les deux fichiers son identiques ou pas.

Syntaxe : cmp fichier1 fichier2

Exemple :

```
$ cmp fic1 fic2
$
```

Si les deux fichiers sont identiques, le **prompt** sera **vide** en retour.
Par contre si les deux fichiers sont différents, la commande «**cmp**» affichera le numéro de la ligne et le numéro du caractère à partir duquel les deux fichiers diffèrent.

Exemple :

```
$ cmp fic1 fic2
fic1 fic2 sont différents: octet27, ligne2
```

La commande «diff»

Cette commande nous informe sur les modifications à apporter au premier fichier afin qu'il ait le même contenu que le second.

Syntaxe du code : ligne(s)_du_fichier1 Action ligne(s)_du_fichier1
Action 'c' : pour changer des lignes.
Action 'd' : pour supprimer des lignes.
Action 'a' : pour ajouter des lignes.

Exemple :

```
$ cat      fic1
ceci est un exemple.
Il fait beau aujourd'hui.
La pluie c'est pour demain.
$
$ cat      fic2
Il ne fait pas beau aujourd'hui.
La pluie c'est pour demain.
Ceci est la fin de l'exemple.
$
$ diff     fic1       fic2
1,2c1                                  les lignes 1 à 2 de fic1 doivent être
< ceci est un exemple.                 changées par la ligne 1 de fic2
< Il fait beau aujourd'hui.
---
> Il ne fait pas beau aujourd'hui.
3a3                                    Après la ligne 3 de fic1 il faut
> Ceci est la fin de l'exemple.        Ajouter la ligne 3 de fic2
$
$ diff     fic2       fic1
1c1,2                                  la ligne 1 de fic1 doit être
< Il ne fait pas beau aujourd'hui.     Changée par les lignes 1 à 2 de fic1
---
> ceci est un exemple.
> Il fait beau aujourd'hui.
3d3                                    il faut supprimer la ligne 3 de fic2
< Ceci est la fin de l'exemple.
$
```

La commande «paste»

Cette commande permet de fusionner des fichiers ligne par ligne.

Exemple :

```
$ cat      fic1
1     Jean-Marc
5     Robin

$ cat      fic2
5     Responsable
8     Stagiaire

$ paste    fic1       fic2
1     Jean-Marc    5     Responsable
5     Robin        8     Stagiaire
```

Manipulation du contenu d'un fichier

Les commandes «cut» et «awk»

- cut

```
$ cut   -c  2-5   fic1
$ cut   -c  -3    fic1
$ cut   -c  3-    fic1
$ cut   -d:   -f  1     /etc/passwd
$ cut   -d:   -f  1,4   /etc/passwd
$ cut   -d:   -f  1-4   /etc/passwd
```

- awk

```
$ awk   '{print $1}'   /etc/hosts
$ awk   -F:   '{print $3, $1}'   /etc/passwd
$ awk   -F:   '{print "Uid = "$3" Login = " $1}'   /etc/passwd
```

Les commandes «cut» et «awk»

Commande «cut»

Cette commande sélectionne une partie de chaque ligne d'un fichier texte.
L'option «**-c**» permet de travailler avec la notion de caractère, alors que l'option «**-f**» permet de travailler avec la notion de champs.

Exemples :

Pour conserver les caractères 2 à 5 de chaque ligne :

```
$ cut      -c  2-5    fic1
```

Pour conserver les 3 premiers caractères :

```
$ cut      -c  -3     fic1
```

Pour conserver à partir du 3ème caractère jusqu'au dernier caractère de la ligne :

```
$ cut      -c  3-     fic1
```

Pour délimiter un champs à conserver, nous utiliserons l'option «**-d**» et l'option «**-f**».

-d : indique le caractère séparateur de champs (par défaut : la tabulation).
-f : indique le numéro du champs à conserver.

Exemples :

Pour récupérer le champs 1, avec le caractère séparateur de champs «:» :

```
$ cut -d: -f 1 /etc/passwd
```

Pour récupérer les champs 1 et 4, avec le caractère séparateur de champs «:» :

```
$ cut -d: -f 1,4 /etc/passwd
```

Pour récupérer les champs 1 à 4, avec le caractère séparateur de champs «:» :

```
$ cut -d: -f 1-4 /etc/passwd
```

Commande «awk»

La commande awk (nawk, gawk) permet de manipuler le contenu d'un fichier en exploitant les champs. Cette instruction peut avoir une syntaxe très complexe car elle intègre toutes les fonctionnalités des scripts.

Syntaxe : awk [-options] 'actions' fichier

```
$ awk '{print $1}' /etc/hosts
127.0.0.1
::1
```

La section entre **{ }** définit la liste des actions à réaliser sur chaque ligne du fichier.
La référence à un champ est indiquée par le caractère **$** suivi du numéro de champ, les caractères 'espace' et 'tabulation' sont les caractères séparateurs de champs par défaut.
Ainsi l'exemple ci-dessus affiche le premier champ du fichier /etc/hosts.

```
$ awk -F: '{print $3, $1}' /etc/passwd
0 root
1 bin
2 daemon
3 adm
4 lp
5 sync
Etc..
```

L'option **-F** redéfinie le caractère séparateur de champs (ici le «**:**»).
L'exemple affiche le champ 3 puis le champ 1 séparés par un caractère «**espace**». Ce dernier est présent à cause du caractère spécial «**,**».

```
$ awk -F: '{print "Uid = "$3" Login = " $1}' /etc/passwd
Uid = 0 Login = root
Uid = 1 Login = bin
Uid = 2 Login = daemon
Uid = 3 Login = adm
Uid = 4 Login = lp
Uid = 5 Login = sync
Etc..
```

Il est possible d'agrémenter l'affichage avec du texte, il suffit de l'écrire entre guillemets.

Manipulation du contenu d'un fichier

La commande «sort»

- sort = tri du contenu du fichier

```
$ sort    /etc/passwd
$ sort    -t:  -k3  /etc/passwd
$ sort    -t:  -k3n /etc/passwd
$ sort    -n   fichier
$ sort    -r   -n  fichier
```

La commande «sort»

Cette commande permet de trier un fichier par ligne dans l'ordre croissant alphabétique.
Plusieurs options sont disponibles pour cette commande comme :

-n : pour un tri numérique.
-k : pour trier à partir d'une colonne spécifique.
-t : défini le caractère séparateurs de champs.
Par défaut : espace, tabulation et retour chariot.
-r : pour trier dans l'ordre décroissant.

Syntaxes :	sort	fic1		Tri alphabétique des lignes du fichier.
	sort	-n	fic1	Le fichier correspond à une colonne numérique. Tri numérique du fichier.
	sort	-k3	fic1	Trier par rapport à la colonne 3 avec les caractères séparateurs de champs par défaut.
	sort	-t:	-k3 fic1	Tri alphabétique dans l'ordre croissant par rapport au champs numéro 3. Le caractère séparateur de champs étant le ':'.
	sort	-t:	-k3n fic1	Idem que la commande précédente mais le tri est un tri numérique sur le champs 3.

Exemples :

```
$ sort     /etc/passwd
abrt:x:173:173::/etc/abrt:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
chrony:x:994:993::/var/lib/chrony:/sbin/nologin
colord:x:997:995:User for colord:/var/lib/colord:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
… etc

$ sort     -t:   -k3   /etc/passwd
root:x:0:0:root:/root:/bin/bash
user1:x:1000:1003:user1:/home/user1:/bin/bash
user2:x:1001:1004::/home/user2:/bin/bash
user3:x:1002:1005::/home/user3:/bin/bash
qemu:x:107:107:qemu user:/:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
… etc

$ sort     -t:   -k3n  /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
… etc

$ cat  fichier
1002
25
1
34
20
2
999
56
$
$ sort  -r -n  fichier
1002
999
56
34
25
20
2
1
$
```

Manipulation du contenu d'un fichier

Les commandes «uniq» et «tr»

- uniq fichier — Lignes uniques

```
$ uniq    fic1
$ uniq    -c   fic1
$ uniq    -d   fic1
$ uniq    -u   fic1
```

- tr — Traduction

```
$ tr  'otu'  'say'    <  fic1
$ tr  "a-z"  "A-Z"    <  fic1
$ cat  fic2  |  tr  -s 'o'
$ who  |  tr  -s  ' '
```

Les commandes «uniq» et «tr»

La commande «uniq»

Cette commande détecte et supprime les lignes identiques successives dans un fichier.

Exemple :

```
$ cat      fic1
Jean
Jean
Marcel
Patrice
Patrice
Jean
```

```
$ uniq     fic1
Jean
Marcel
Patrice
Jean
```

Des options sont disponibles pour cette commande comme :

-c : affiche le nombre d'occurrences.
-d : affiche uniquement les lignes dupliquées dans le fichier.
-u : affiche que les lignes uniques du fichier.

Exemples :

```
$ uniq     -c    fic1
2 Jean
1 Marcel
2 Patrice
1 Jean

$ uniq     -d    fic1
Jean
Patrice

$ uniq     -u    fic1
Marcel
Jean
```

La commande «tr»

Cette commande permet de remplacer un caractère par un autre.
Dans l'exemple ci-dessous, nous remplaçons les caractères (o,t et u) respectivement par (s,a et y) :

```
$ cat      fic1
Bonjour tout le monde.

$ tr  'otu'  'say'    <  fic1
Bsnjsyr asya le msnde.
```

On peut utiliser des intervalles de caractères en utilisant le caractère «-». Ainsi il est possible de remplacer les caractères minuscules d'un fichier par des majuscules :

```
$ cat      fic1
Bonjour tout le monde.

$ tr  "a-z"  "A-Z"    <  fic1
BONJOUR TOUT LE MONDE.
```

Option « -s » : cette option remplace un succession d'un même caractère par un seul.

```
$ cat  fic2
Booooonjour nous avoooons
souvent oorganise des fooooormatiooons
$
$ cat  fic2  |  tr  -s 'o'
Bonjour nous avons
souvent organise des formations

$ who
user1    :0           2016-04-29 09:25 (:0)
user1    pts/0        2016-04-29 09:25 (:0)
user1    pts/1        2016-04-29 10:43 (spheriusform-pc.home)
$
$ who  |  tr  -s  ' '
user1 :0 2016-04-29 09:25 (:0)
user1 pts/0 2016-04-29 09:25 (:0)
user1 pts/1 2016-04-29 10:43 (spheriusform-pc.home)
```

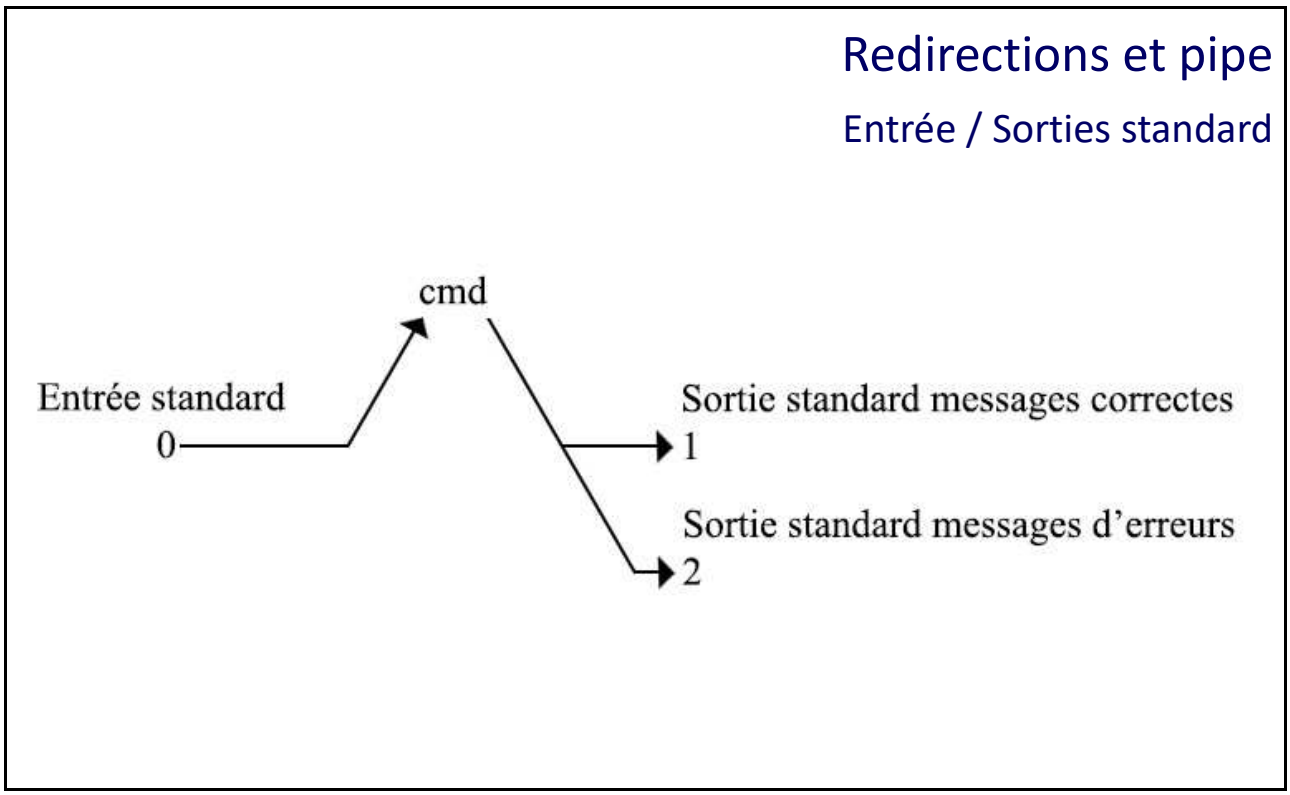
Notes

Redirections et pipe

Dans ce chapitre nous allons nous familiariser avec le mécanisme des redirections des entrées/sorties et du pipe.

Redirections et pipe

- Entrée / Sorties standard
- Redirection des Entrées / Sorties
- Présentation du Pipe



Entrée / Sorties standard

Lorsque l'on exécute une commande, il existe 3 flux de données par défaut :
L'entrée standard, la sortie standard, et la sortie standard des messages d'erreur.

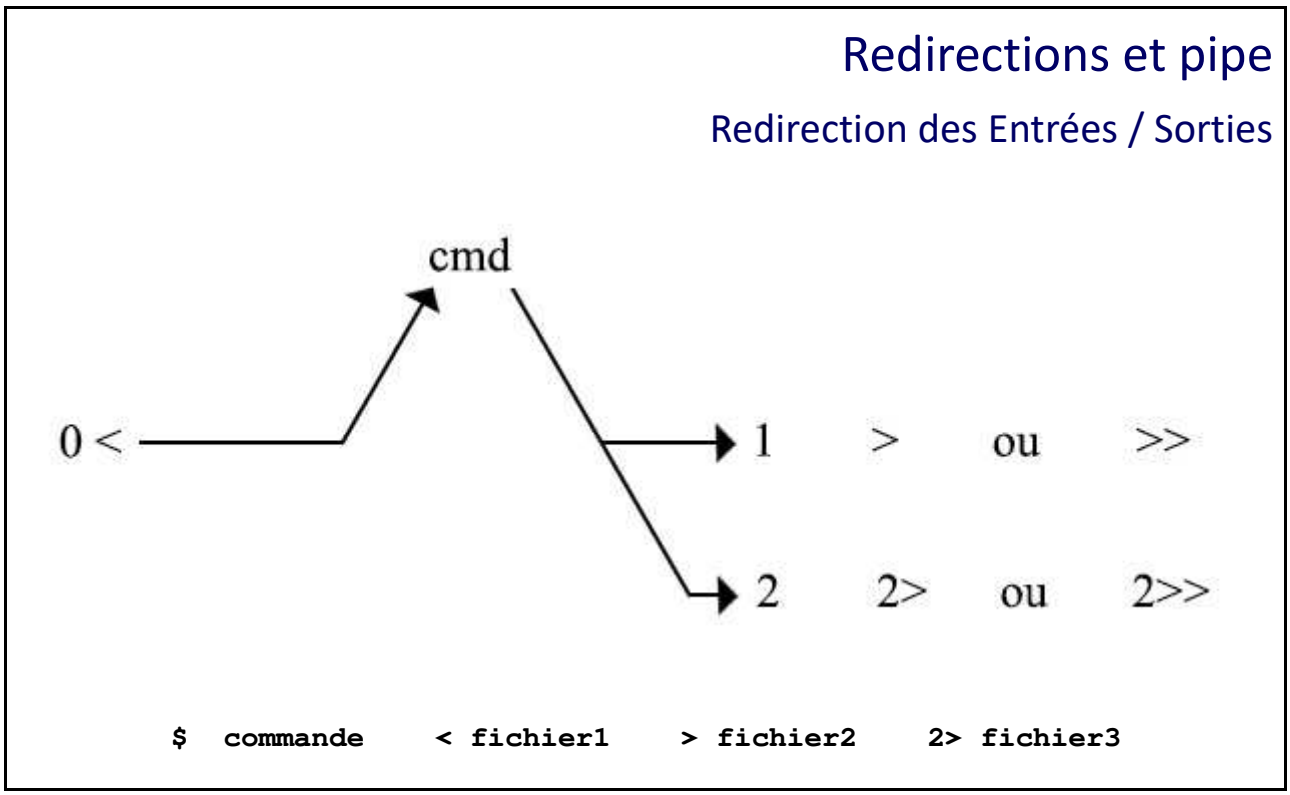
Entrée standard

C'est le flux qui permet de lire les données d'entrée. Appelé «**stdin**», il est identifié par le flux numéro **0**.
Par défaut ce flux correspond au clavier.

Sortie standard / Sortie d'erreur

Lors de l'exécution d'une commande ou d'un programme, les résultats sont envoyés dans deux flux distincts :

La sortie standard	:	pour les messages corrects. Appelé «**stdout**» et identifié par le flux numéro **1**. Par défaut cela correspond au pseudo-terminal (tty) auquel est rattaché la commande ou le programme.
La sortie d'erreur	:	pour les messages d'erreurs. Appelé «**stderr**» et identifié par le flux numéro **2**. Par défaut cela correspond à la même sortie que la sortie standard.



Redirection des Entrées / Sorties

Entrée standard - flux 0

Il est possible de changer la source de l'entrée standard par le contenu d'un fichier.
Ainsi lorsqu'une commande attend par défaut une saisie au clavier (flux 0 de l'entrée standard), on peut faire en sorte qu'à la place cette même commande lise le contenu d'un fichier. Il est donc nécessaire pour cela de changer l'entrée standard en la connectant non pas au clavier mais au fichier concerné. Ceci est possible avec la syntaxe suivante :

commande 0< fichier_d_entrée

Plus communément commande < fichier_d_entrée

Exemple :

```
$ mail user1@spherius.fr
_                                  attente de saisie au clavier
                                   du texte du corps du mail


$ mail user1@spherius.fr  <  fichier_du_corps_du_mail
$
```

Sortie standard des messages corrects - flux 1

Il est possible de changer la destination de la sortie standard des messages corrects pour alimenter le contenu d'un fichier.
Ainsi lorsqu'une commande affiche son résultat à l'écran, en fait le pseudo-terminal (flux 1 de la sortie standard), on peut faire en sorte qu'à la place le résultat de cette commande ne s'affiche pas à l'écran mais soit redirigé vers un fichier.

Première syntaxe – le 1> ou le >

commande 1> fichier_de_sortie

Plus communément commande > fichier_de_sortie

Si le fichier n'existe pas, il est créé. Si le fichier existe, le contenu est écrasé avec le résultat de la commande.

Deuxième syntaxe – le 1>> ou le >>

commande 1>> fichier_de_sortie

Plus communément commande >> fichier_de_sortie

Si le fichier n'existe pas, il est créé. Si le fichier existe, le résultat de la commande est ajouté à la fin du fichier.

Exemple :

```
$  cat fichier1
bonjour, ceci est le contenu de fichier1
$
$  date  >>  fichier1
$
$  cat fichier1
bonjour, ceci est le contenu de fichier1
mar. avril 26 16:54:00 CEST 2016
$
$  pwd  >>  fichier1
$  cat fichier1
bonjour, ceci est le contenu de fichier1
mar. avril 26 16:54:00 CEST 2016
/home/user1
$
$  date  >  fichier1
$
$  cat fichier1
mar. avril 26 16:56:10 CEST 2016
$
```

Sortie standard des messages d'erreurs - flux 2

Comme pour la sortie des messages corrects, il est possible de changer la destination de la sortie standard des messages d'erreurs pour alimenter le contenu d'un fichier.
Ainsi lorsqu'une commande affiche un message d'erreur à l'écran, en fait le pseudo-terminal (flux 2 de la sortie standard des messages d'erreurs), on peut faire en sorte qu'à la place le message d'erreurs ne s'affiche pas à l'écran mais soit redirigé vers un fichier.

Première syntaxe – le 2>

commande 2> fichier_de_sortie

Si le fichier n'existe pas, il est créé. Si le fichier existe, le contenu est écrasé avec le résultat de la commande.

Deuxième syntaxe – le 2>>

commande 2>> fichier_de_sortie

Si le fichier n'existe pas, il est créé. Si le fichier existe, le résultat de la commande est ajouté à la fin du fichier.

Exemple :

```
$  cat fichier1
bonjour, ceci est le contenu de fichier1
$
$  datex
Error : command 'datex' not found.
$
$
$  datex  2>>  fichier1
$
$  cat fichier1
bonjour, ceci est le contenu de fichier1
Error : command 'datex' not found.
$
$  datexyz  2>>  fichier1
$
$  cat fichier1
bonjour, ceci est le contenu de fichier1
Error : command 'datex' not found.
Error : command 'datexyz' not found.
$
$
$  cmdxxxx  2>  fichier1
$
$  cat fichier1
Error : command 'cmdxxxx' not found.
$
```

Syntaxe complète

nous pouvons cumuler sur la même ligne de commandes les redirections nécessaires :

Exemple 1 : pour rediriger les deux sorties standards :

```
$ commande > fichier1 2> fichier2
$

ou

$ commande 2> fichier2 > fichier1
$
```

Ainsi, les messages corrects résultant de la commande sont stockés au sein du fichier1 et les messages d'erreurs au sein du fichier2.

Exemple 2 : pour rediriger les trois flux standards :

```
$ commande < fichier1 > fichier2 2> fichier3
$
```

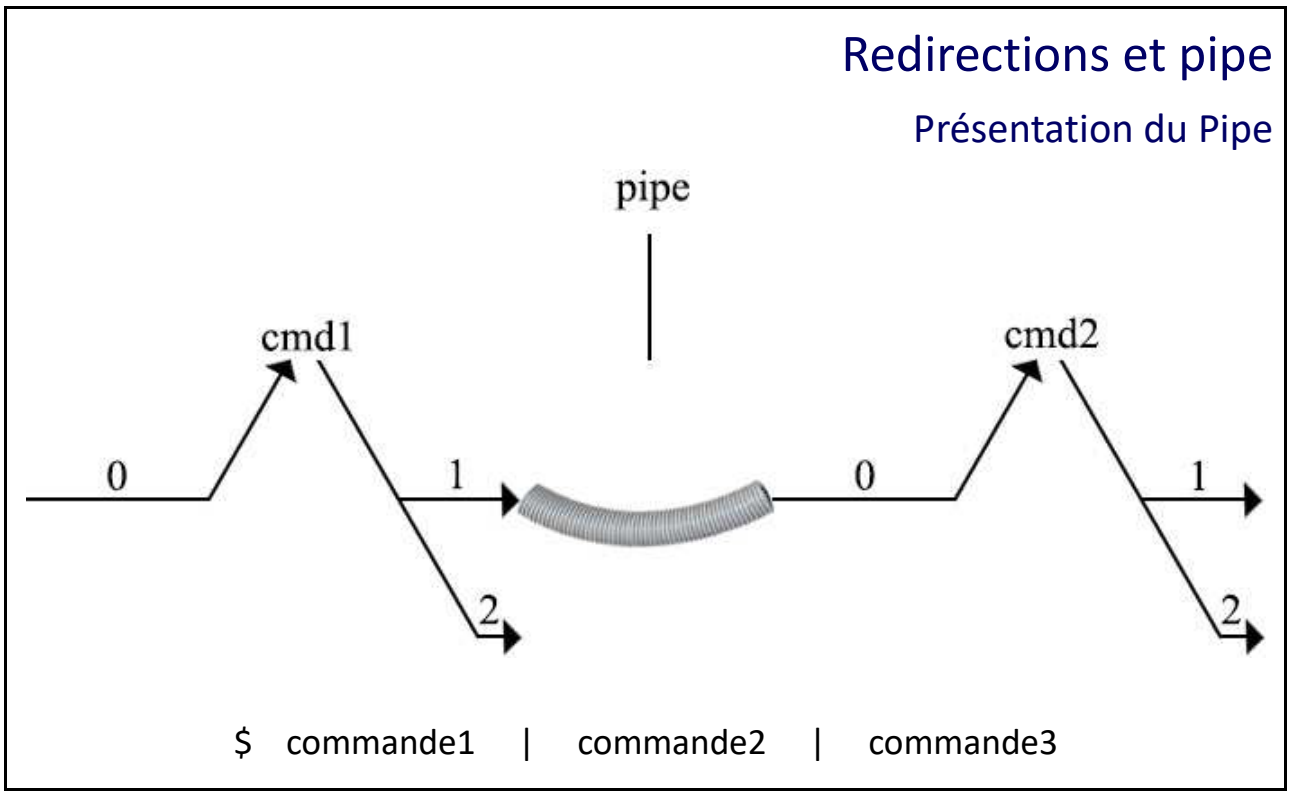
Ainsi, la commande utilise le contenu du fichier1 en entrée, les messages corrects résultant de la commande sont stockés au sein du fichier2 et les messages d'erreurs au sein du fichier3.

Remarque

Pour la redirection de la sortie, l'utilisation du fichier spécial /dev/null permet de ne pas conserver les messages reçus .

```
$ commande 2> /dev/null
$
```

Ainsi, la commande affiche les messages corrects, les messages d'erreurs n'apparaissent pas et ne sont pas conservés.



Présentation du Pipe

Le «**pipe**» est une passerelle qui permet de transférer le résultat d'une commande «**cmd1**» à une autre commande «**cmd2**».
Son symbole est la barre verticale «**|**».
Plus précisément, le flux numéro **1** (messages corrects) de la commande «**cmd1**» est envoyé en entrée du flux numéro **0** (flux d'entrée) de la commande «**cmd2**».

Premier exemple - ls -Rl / | more

La première commande est 'ls -Rl /'. Elle permet d'afficher de manière récursive toute l'arborescence du système au format long. Le résultat de cette commande fournissant énormément de lignes, il en résulte un défilement à l'écran qui n'est pas exploitable pour une personne.

L'idéal serait que cette affichage se fasse page par page. Or justement c'est la fonction de la commande more.

Ainsi en redirigeant le résultat de première commande en entrée de la commande more, l'affichage sera paginée. C'est que permet le pipe pour combiner ces deux commandes.

Deuxième exemple - grep root /etc/passwd | wc -l | lp

cette exemple permet d'imprimer le nombre de lignes contenant « root » du fichier /etc/passwd.

La première commande est 'grep root /etc/passwd'. Elle affiche du fichier /etc/passwd uniquement les lignes contenant la chaîne de caractères « root ».

'grep root /etc/passwd | wc -l' affiche le nombre de ligne résultant de la commande 'grep'.

L'affichage précédent est envoyé en entrée de la commande 'lp' qui a pour fonction d'imprimer ce qu'elle reçoit.

Exemple avec la commande «tee»

Grâce à la commande «**tee**» récupère les données de l'entrée standard, pour les rediriger simultanément vers un fichier (passé en argument) et vers l'écran (la sortie standard des messages corrects).
Elle est pratique pour le pipe car cela permet de faire une sauvegarde dans un fichier des données d'un état intermédiaire du pipe.

Exemple :

```
$ echo "Bonjour tout le monde." | tee fic3 | wc
1           4          22
$ cat fic3
Bonjour tout le monde.
```

Le résultat de la commande 'echo' a été stocké dans un fichier fic3 et redirigé vers la commande suivante. 'wc' affichera à l'écran le nombre de caractères, de mots et de lignes résultant du 'echo'.

Notes

Les permissions

Dans ce chapitre nous allons traiter les permissions possibles à affecter sur les fichiers ordinaires et les répertoires.

Les permissions

- Les droits
- Les types d'utilisateurs
- Mode des fichiers ou des répertoires
- Modification des droits d'accès
- Permissions supplémentaires
- Autres commandes

Les permissions

Les droits

- Le droit de lecture r
- Le droit d'écriture w
- Le droit d'exécution x

Les droits

Les droits permettent à un utilisateur de définir les permissions pour un fichier ou un répertoire. Ils permettent aussi de gérer les actions que les utilisateurs peuvent effectuer sur les fichiers (**lecture**, **écriture**, **exécution**) selon leur niveau de permissions.

Le droit de lecture

Le droit de lecture est symbolisé par la lettre (**r**). L'utilisateur ayant cette permission peut afficher le contenu des fichiers.
Pour un fichier de type fichier ordinaire, cela concerne les commandes telles que : **cat**, **more**, ...
Pour un fichier de type répertoire, cela concerne les commandes telles que : **ls**, …

Le droit d'écriture

Le droit d'écriture est symbolisé par la lettre (**w**). L'utilisateur ayant cette permission peut modifier le contenu des fichiers.
Pour un fichier ordinaire, cela concerne les commande telles que : **vi**, ...
Pour un répertoire, cela concerne les commandes telles que : **rm**, **mkdir**, **touch**, ...

Le droit d'exécution

Le droit d'exécution est symbolisé par le lettre (**x**). Ainsi, l'utilisateur peut exécuter le fichier.
Pour un fichier ordinaire, cela correspond à l'exécution, comme un script.
Pour un répertoire, cela correspond au droit de passage : **cd**, ...

Les permissions

Les types d'utilisateurs

- L'utilisateur propriétaire user
- Le groupe propriétaire group
- Les autres other

Les types d'utilisateurs

Il existe trois catégories d'utilisateurs pour un fichier :

- L'utilisateur propriétaire, caractérisé par son nom ou son numéro appelé **UID**.

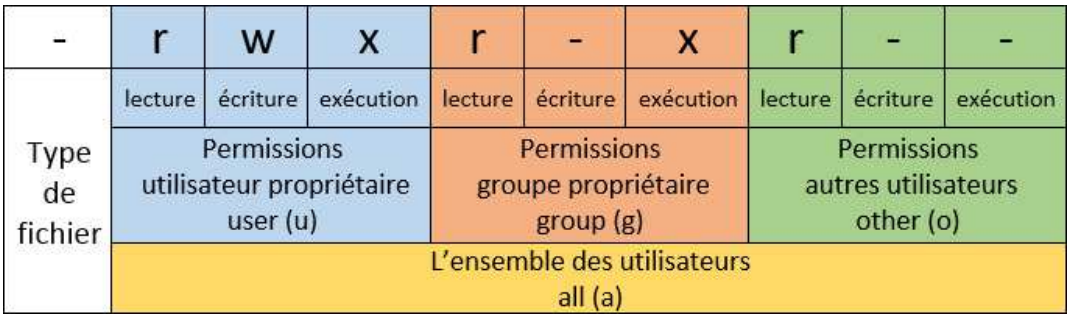
- Le groupe propriétaire, caractérisé par son nom ou son numéro appelé **GID**.

- Tous les autres utilisateurs.

Des permissions sur un fichier pourront être attribuées de manière spécifique à une catégorie d'utilisateurs.

Les permissions

Mode des fichiers ou des répertoires



Mode des fichiers ou des répertoires

u (user) : Désigne l'utilisateur propriétaire.
g (group) : Désigne le groupe propriétaire.
o (other) : Désigne les autres utilisateurs.
a (all): Désigne l'ensemble des utilisateurs.

Du résultat de la commande «**ls -l**» :
Le 1er caractère est le type du fichier (donc ne concerne pas les permissions).
Les 3 caractères suivants concernent l'attribution des permissions à l'utilisateur propriétaire (**u**).
Les 3 caractères suivants concernent l'attribution des permissions au groupe propriétaire (**g**).
Les 3 caractères suivants concernent l'attribution des permissions aux autres utilisateurs (**o**).

Nous utiliserons également la caractère «**a**» pour représenter tous les utilisateurs (a pour all).

Exemple :

-rwx r-x r-- user1 groupe1 fic1

Le 1er caractère est le type du fichier. Le tiret (**-**) indique un fichier ordinaire.
Le propriétaire est 'user1'. Il a tous les droits (**rwx**).
Le groupe propriétaire est 'groupe1'. Les utilisateurs de ce groupe ont les droits de lecture et d'exécution (**r-x**).
Les autres utilisateurs ont uniquement le droit de lecture (**r--**).

Les permissions

Modification des droits d'accès

- Notation symbolique

chmod permission fichier

«permission»		
Qui	Opérateur	Droits
u, g, o ou a	remplacer par le caractère +, - ou =	r, w ou x

```
$ chmod u+x fichier1
$ ls -l fichier1
-rwxrw-r--. 1 user1 user1 0 29 mars  15:59 fichier1
```

Modification des droits d'accès

Pour modifier les permissions d'un fichier, on utilise la commande « **chmod** » (**CH**ange **MOD**e).
On utilise soit la notation symbolique, soit la notation octale.

Pour utiliser cette commande, il faut être le propriétaire du fichier ou root.

Notation symbolique

La notation symbolique permet de modifier les droits d'accès via une description relative ou absolue.

Syntaxe : chmod [**Qui**][**Opérateur**][**Droits**] fichier

chmod [**ugoa**][**+ - =**][**rwx**] fichier

Opérateur :

+ : ajouter des droits.
- : supprimer des droits.
= : affecter des droits.

Exemples :

```
$ ls -l fichier1
-rw-rw-r--. 1 user1 user1 0 29 mars  15:59 fichier1

$ chmod u+x fichier1
$ ls -l fichier1
-rwxrw-r--. 1 user1 user1 0 29 mars  15:59 fichier1

$ chmod g+x,o+w fichier1
$ ls -l fichier1
-rwxrwxrw-. 1 user1 user1 0 29 mars  15:59 fichier1

$ chmod a-w,ug-x fichier1
$ ls -l fichier1
-r--r--r--. 1 user1 user1 0 29 mars  15:59 fichier1

$ chmod a=rwx fichier1
$ ls -l fichier1
-rwxrwxrwx. 1 user1 user1 0 29 mars  15:59 fichier1
```

Les permissions

Modification des droits d'accès

- Notation octale

r	=	4
w	=	2
x	=	1

```
$ chmod 754 fichier1
$ ls -l  fichier1
-rwx r-x r--   1  user1  user1  0  29 mars  14:37  fichier1
```

Notation octale

La notification octale est établie comme suit :

r	=	4
w	=	2
x	=	1

En prenant l'exemple d'un fichier ayant ce droit d'accès : **-rwx rw- r-x**

Sa représentation octale est le nombre 765 car :

rwx = 4+2+1	**rw-** = 4+2+0	**r-x** = 4+0+1
7	6	5

Exemple :
Si je souhaite modifier la protection du fichier «**fichier1**» avec les droits d'accès 754, j'utilise la syntaxe de la commande suivante :

```
$ chmod  754  fichier1
$ ls -l  fichier1
-rwx r-x r--   1   user1   user1   0   29 mars   14:37   fichier1
```

Les permissions

Permissions supplémentaires

- SetUID

chmod u+s fichier

chmod 4755 fichier

```
$ chmod  u+s  programme1
$ ls  -l  programme1
-rwsr-xr-x 1 user1  groupe1  27832  01  janvier  2016 programme1
```

- SetGID

chmod g+s fichier

chmod 2755 fichier

```
$ chmod  g+s  programme1
$ ls  -l   programme1
-rwxr-sr-x 1 user1  groupe1  27832  01  janvier  2016 progamme1
```

Permissions supplémentaires

Certains fichiers peuvent être associés à des permissions supplémentaires.

SetUID

De base lorsqu'on exécute une commande, c'est via l'identité du compte sous laquelle on s'est connecté. La permission **SetUID** permet de prendre l'identité du **propriétaire** de la commande juste le temps de son exécution.

Il est possible de visualiser cette permission par le caractère «**s**» qui remplace le caractère «**x**» dans la partie propriétaire.

Syntaxe : chmod u+s programme
chmod 4755 programme

Exemple :

```
$ chmod u+s programme1
$ ls -l programme1
-rwsr-xr-x 1 user1  groupe1  27832  01  janvier  2016 programme1
```

Grâce au SetUID, il est possible d'avoir les privilèges administrateur à partir d'une commande appartenant à root juste le temps de son exécution.

SetGID

Tout comme la permission **SetUID** qui agit sur l'identité du user propriétaire, la permission **SetGID** agit sur l'identité du groupe propriétaire juste le temps de l'exécution.

Il est possible de visualiser la présence de cette permission par le caractère «**s**» qui remplace le caractère «**x**» dans la partie groupe.

Syntaxe : chmod g+s programme
chmod 2755 programme

Exemple :

```
$ chmod  g+s  programme1
$ ls  -l  programme1
-rwxr-sr-x 1 user1  groupe1  27832  01  janvier  2016 progamme1
```

Compléments

ci-dessous un exemple détaillant l'intérêt des SetUID et SetGID.

Le 'programme1' contient une instruction exécutable que par 'user1' le propriétaire du script. Plus précisément, l'instruction nécessite l'identité UID du propriétaire du script.
Le 'programme2' contient une instruction exécutable qu'avec l'identité uid et gid du propriétaire du script, soit de 'user1' et 'groupe1'.

Un autre utilisateur 'user2' du 'groupe2' va tenter d'exécuter ces deux programmes.

```
user2:groupe2$ ls  -l  programme1  programme2
-rwxr-xr-x 1 user1  groupe1  27832  01  janvier  2016 progamme1
-rwxr-xr-x 1 user1  groupe1  23224  01  janvier  2016 progamme2
user2:groupe2$ ./programme1
Debut du traitement
Error : instruction line 24 : no permission
user2:groupe2$ ./programme2
Debut du traitement
Error : instruction line 127 : no permission

user1$ chmod  u+s  programme1
user1$ chmod  u+s,g+s  programme2

user2:groupe2$ ls  -l  programme1  programme2
-rwsr-xr-x 1 user1  groupe1  27832  01  janvier  2016 progamme1
-rwsr-sr-x 1 user1  groupe1  23224  01  janvier  2016 progamme2
user2:groupe2$ ./programme1     exécution en tant que user1:groupe2
Debut du traitement
Le traitement s est termine correctement.
user2:groupe2$ ./programme2     exécution en tant que user1:groupe1
Debut du traitement
Le traitement s est termine correctement.
user2:groupe2$
```

Les permissions

Permissions supplémentaires

- Sticky Bit

chmod o+t répertoire
chmod 1777 répertoire

```
$ chmod  1777  repertoire1

$ ls –l repertoire1
drwxrwxrwt   2   user1   user1   0   01 avril   11:25   repertoire1
```

Sticky Bit

Cette permission permet de créer un répertoire partagé. Tous les utilisateurs pourront créer des fichiers et des répertoires, mais le droit de suppression est accordé uniquement au propriétaire de ces fichiers.

Par défaut, le répertoire «**/tmp**» a un sticky bit (répertoire partagé).

Syntaxe : chmod o+t répertoire
chmod 1777 répertoire

Exemple :

```
$ chmod    1777  repertoire1
$ ls       –l    repertoire1
drwxrwxrwt   2   user1   user1   6   01 avril   11:25   repertoire1
```

Les permissions

Autres commandes

- umask

commande → umask
valeur umask → 0002

```
$ umask
0002

$ touch  fichier1
$ mkdir  repertoire1

$ ls -l
-rw-rw-r--. 1 user1 user1 0 01 avril  12:04 fichier1
drwxrwxr-x. 2 user1 user1 6 01 avril  12:04 repertoire1
```

Autres commandes

umask

La valeur du 'UMASK' permet de définir les permissions par défaut pour les fichiers et répertoires. Cette valeur peut être modifiée par la commande 'umask'.

```
$ umask                    pour afficher la valeur
0002

$ umask  044               pour modifier la valeur
$ umask
0044
```

Au moment de la création du fichier ou d'un répertoire, les permissions qui seront affectées suivent le mécanisme suivant :

Par défaut la valeur octale d'un répertoire est de 777 et la valeur d'un fichier est 666, à laquelle il faudra retirer la valeur du UMASK.

Exemple :

```
$ umask
0002
```

```
$ touch  fichier1
$ mkdir  repertoire1

$ ls -l
-rw-rw-r--. 1 user1 user1 0 01 avril  12:04 fichier1
drwxrwxr-x. 2 user1 user1 6 01 avril  12:04 repertoire1
```

Avec la valeur 0002 pour l'UMASK, le fichier et le répertoire auront respectivement la valeur octale 664 et 775.

Les permissions

Autres commandes

- chgrp nouveau_groupe fichier_ou_repertoire
- chown nouveau_propriétaire[:nouveau_groupe] fichier_ou_repertoire
- newgrp nouveau_groupe

La commande chgrp

Cette commande change le groupe propriétaire d'un fichier ou d'un dossier.
Il faut être le propriétaire du fichier ou root pour utiliser cette commande.

Syntaxe : **chgrp nouveau_groupe_propriétaire fichier_ou_repertoire**

Option : **-R** : modification récursive à partir d'un répertoire.

Exemple :

```
user1$ ls –l
-rw-rw-r--. 1 user1 groupe1 0 01 avril  12:59 fichier1

user1$ chgrp   groupe2   fichier1
change the group of fichier1 to «groupe2»

$ ls –l
-rw-rw-r--. 1 user1 groupe2 0 01 avril  12:59 fichier1
```

La commande chown

Cette commande 'chown' (Change OWNer) change le propriétaire d'un fichier ou d'un dossier, elle peut en plus modifier le groupe propriétaire.
Seul l'utilisateur root peut exécuter cette commande.

Syntaxe : **chown nouveau_propriétaire[:nouveau_groupe] fichier_ou_repertoire**

Option : **-R** : modification récursive à partir d'un répertoire.

Exemple :

```
# ls -l folder1
-rw-r--r--. 1 user1  groupe1    0 30 mars  14:13 folder1

# chown root folder1
change the owner of folder1 to «root»

# ls -l folder1
-rw-r--r--. 1 root  groupe1    0 30 mars  14:13 folder1

# ls -l fic1
-rw-r--r--. 1 user1 groupe1 0 30 mars  14:10 fic1

# chown user2:groupe3 fic1

# ls -l fic1
-rw-r--r--. 1 user2 groupe3 0 30 mars  14:10 fic1
```

La commande newgrp

Cette commande modifie l'identité du groupe principal associé à l'utilisateur.
L'utilisateur doit être au préalable invité au nouveau groupe pour que la commande fonctionne.

Syntaxe : **newgrp nouveau_groupe**

Notes

La commande find

commande find.

La commande find

- Présentation
- Critères de recherche
- Combinaison de critères de recherche
- Actions

La commande find

Présentation

- Recherche de fichiers dans l'arborescence

  find A_partir_d_où Critere_de_recherche Action

- A_partir_d_où = Un répertoire

Présentation

La commande «**find**» recherche des fichiers dans l'arborescence en fonction de différents types de critères. Elle permet également d'exécuter une action sur le résultat de la recherche.
Si le répertoire est omis la commande find recherche par rapport au répertoire courant.
L'action par défaut est '-print' pour un affichage des résultats à l'écran.

Syntaxe : find A_partir_d_où Critere_de_recherche action

A partir d'où

Le premier argument de la commande «find» défini le répertoire à partir duquel la recherche récursive commencera.
En utilisant un chemin relatif, le résultat de la recherche apparaîtra en chemin relatif (rep, ., .., ...).
Avec un chemin absolu, le résultat apparaîtra en chemin absolu (/home, $HOME, ...).

Exemples :

```
$ find .. -name fic2 -print
../Bureau/fic2
../Bureau/rep/fic2
../.local/share/Trash/files/fic2
```

```
$ find $HOME/rep -name fic2 -print
/home/user1/Bureau/rep/fic2
```

La commande find

Critères de recherche

- -name -iname
- -user -group
- -perm -inum -type
- -size -atime -mtime -ctime
- ...

Critères de recherche

-name : par le nom du fichier.
-iname : par le nom du fichier sans tenir compte de la casse.

```
$ find repertoire -name fichier -print

$ find repertoire -name «fic*» -print
```

-user : par le propriétaire du fichier.

```
$ find repertoire -user user1 -print
```

-group par l'appartenance du fichier à un groupe.

```
$ find repertoire -group groupe1 -print
```

-perm : par l'autorisation du fichier (permissions).

```
$ find repertoire -perm 777 -print
```

-inum : par le numéro d'inode du fichier.

```
$ find repertoire -inum 25345 -print
```

-type : par le type du fichier.
f = fichier régulier.
d = répertoire.
l = lien symbolique.

```
$ find repertoire -type d -print
```

-size : par la taille du fichier.
nc = par octets.
nk = kilo octets.
+n = supérieur à «n», -n = inférieur à «n»

```
$ find repertoire -size 500k -print
$ find repertoire -size +500k -print
$ find repertoire -size -500k -print
```

-atime : par la date de dernier accès, en nombre de jours.
-ctime : par la date de création, en nombre de jours.
-mtime : par la date de la dernière modification, en nombre de jours.
+n = plus ancien que n jours, -n = plus récent que n jours.

```
$ find repertoire -mtime 5 -print
$ find repertoire -atime +60 -print
$ find repertoire -atime -30 -print
```

Il existe de nombreux autres critères de recherche, qui peuvent varier en fonction de la version du Linux. Veuillez vous référer au man de la commande 'find'.

La commande find

Combinaison de critères de recherche

- OU Logique

```
$ find  /home/user1  \( -name  fic1  -o  -name  fic2 \)  -print
```

- ET Logique

```
$ find  $HOME  \( -type d  -a  -name  «rep*» \)  -print

$ find  /home  \( -user  user1  -a  -perm  777 \)  -print
```

- Pour exclure

```
$ find  /home/user1    ! -type d    -print

$ find  /home/user1  \( -type f  -a  ! -name  «fic*» \)  -print
```

Combinaison de critères de recherche

Il est possible de combiner plusieurs critères de recherche.

Pour réaliser un « OU Logique » - opérateur -o

Lorsqu'un seul critère correspond, cela suffit pour afficher le fichier.

Syntaxe : find rep \(critere_1 -o critere_2 \) action

Exemple :

```
$ find  /home/user1  \( -name  fic1  -o  -name  fic2 \)  -print
```

Pour réaliser un «ET Logique » - opérateur -a

Lorsque tous les critères doivent correspondre pour afficher le fichier.

Syntaxe : find rep \(critere_1 -a critere_2 \) action

Exemples :

```
$ find  $HOME  \( -type d  -a  -name  «rep*» \)  -print

$ find  /home  \( -user  user1  -a  -perm  777 \)  -print
```

Pour exclure - caractère spécial !

Lorsque que l'on souhaite exclure un critère, c'est à dire pour prendre l'inverse, on utile le «!».

Syntaxe : find rep ! Critere action

Exemples :

```
$ find  /home/user1    ! -type d    -print

$ find  /home/user1  \( -type f  -a  ! -name  «fic*» \)  -print
```

La commande find

Actions

- -print

```
$ find . -name «fic1» -print
```

- -exec

```
$ find . -name «fic1» -exec rm '{}' \;
```

- -ok

```
$ find . -name «fic1» -ok rm '{}' \;
```

Actions

Il existe différentes actions que vous pouvez appliquer au(x) fichier(s) résultant de la recherche.

-print : permet d'afficher le résultat sur la sortie standard (action par défaut).

Exemple :

```
$ find . -name «fic1» -print
./rep/fic1
```

-exec : permet d'exécuter une commande. La syntaxe «**'{}' \;**» positionne le résultat du find au sein de la commande que l'on souhaite exécuter.

Exemple :

```
$ find . -name «fic1» -exec rm '{}' \;
fic1 à été supprimé.
```

-ok : idem que l'action «**-exec**» mais demande l'autorisation à l'utilisateur avant d'exécuter la commande. Y pour oui, N pour non.

Exemple :

```
$ find . -name «fic1» -ok rm '{}' \;
< rm … ./rep/fic1 > ? y
fic1 à été supprimé.
```

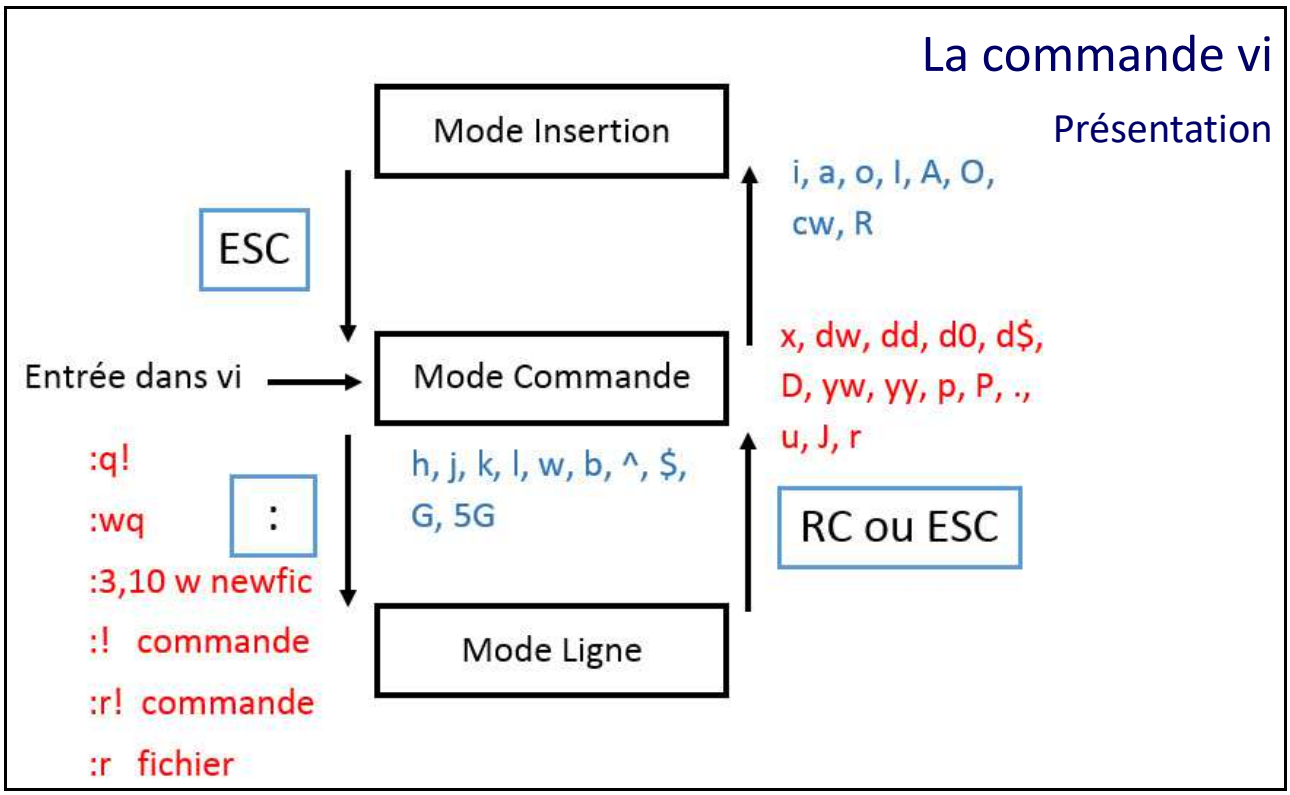
Notes

La commande vi

Dans ce chapitre nous allons traiter l'utilisation de l'éditeur de texte vi.

La commande vi

- Présentation
- Les déplacements du curseur
- Mode insertion
- Suppression – Mode commande
- Compléments – Mode commande
- Mode ligne
- Mode ligne – suite
- Fichier «.exrc»



Présentation

«**vi**» et «**vim**» sont des éditeurs de texte très puissant mais non conviviaux.
Toutes les fonctionnalités sont disponible via le clavier minimum (c'est à dire sans le pavé numérique, ni les flèches, ni les touches fonction).

Indispensable lorsqu'il n'y a pas d'interface graphique installé sur le serveur Linux.
Toute les fonctionnalités présentent sur ce slide sont détaillées par la suite dans ce module.

Les trois modes de «vi»

Mode commande	:	permet de ce déplacer et de modifier le texte dans l'éditeur.
Mode insertion	:	permet d'insérer des caractères dans le document.
Mode ligne	:	permet de réaliser des actions globales.

Syntaxe : vi fic

Édite le fichier fic, si le fichier n'existe pas il est créé.

vi +5 fic

Édite le fichier fic, et positionne le curseur sur la ligne 5.

La commande vi

Les déplacements du curseur

- h j k l + -
- w b 3w 3b
- ^ $
- G 1G 5G

Les déplacements du curseur

La plupart des actions ce font par rapport à la position du curseur. Les déplacements du curseur se font en mode commande.

Les touches de déplacement

Caractère	«**l**»	:	déplacement vers la droite.
Caractère	«**h**»	:	déplacement vers la gauche.
Caractère	«**k**»	:	déplacement sur la ligne du haut.
Caractère	«**j**»	:	déplacement sur la ligne du bas.
Caractère	«**+**»	:	déplacement au début de la ligne suivante.
Caractère	«**-**»	:	déplacement au début de la ligne précédente.

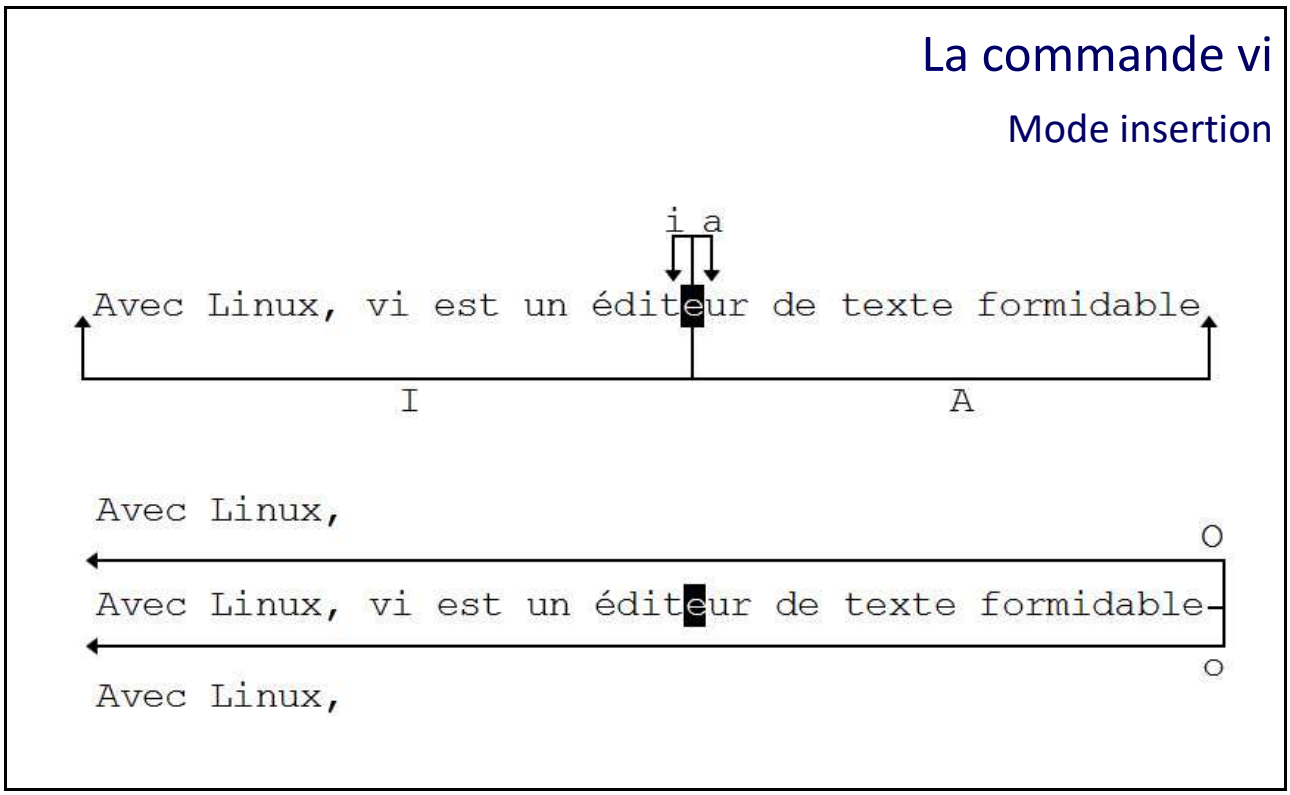
Nous pouvons aussi utiliser les flèches directionnelles.

Caractère	«**w**»	:	déplacement au début du mot suivant.
Caractère	«**b**»	:	déplacement au début du mot précédent.

En ajoutant une valeur devant le caractère, on spécifie le nombre de déplacements.

Caractère	«**^**»	:	déplacement sur le premier caractère de la ligne.
Caractère	«**$**»	:	déplacement sur le dernier caractère de la ligne.

Caractère	«**G**»	:	déplacement sur la dernière ligne du fichier.
	«**1G**»	:	déplacement sur la première ligne du fichier.
	«**5G**»	:	déplacement sur la cinquième ligne du fichier.



Mode insertion

Lorsque nous sommes dans le **mode insertion**, tout ce que nous tapons au clavier est inséré dans le document.

Pour basculer dans le mode insertion

Caractère	«**i**»	:	pour insérer du texte avant le curseur.
	«**I**»	:	pour insérer du texte au début de la ligne.
Caractère	«**a**»	:	pour insérer du texte après le curseur.
	«**A**»	:	pour insérer du texte à la fin de la ligne.
Caractère	«**o**»	:	pour insérer une ligne après la ligne où est localisé le curseur.
	«**O**»	:	pour insérer une ligne avant la ligne où est localisé le curseur.

Pour sortir du mode insertion

Pour quitter le **mode insertion** et retourner dans le **mode commande**, nous utilisons la touche «**ESC**».

La commande vi

Suppression - Mode commande

- x
- dd 5dd D
- dw 5dw
- db 5db
- d^ d$
- dG d1G d5G

Suppression – Mode commande

Ce mode nous permet de supprimer des paragraphes, des lignes, des mots ainsi que des caractères. Tout se fait par rapport à la position du curseur.

«**x**» : suppression du caractère sous le curseur.

«**dd**» : suppression de la ligne courante.
«**5dd**» : suppression de 5 lignes.
«**D**» : suppression du curseur à la fin de la ligne.

«**dw**» : supprime du curseur à la fin du mot.
«**5dw**» : supprime 5 mots après du curseur .
Donc si le curseur est positionné sur le premier caractère, cela revient à supprimer un mot.

«db» : supprime du curseur au début du mot.
«**5db**» : supprime 5 mots avant le curseur.

«**d^**» : suppression du curseur jusqu'au début de la ligne.
«**d$**» : suppression du curseur jusqu'à la fin de la ligne.

«**dG**» : supprime du curseur jusqu'à la fin du fichier.
«**d1G**» : supprime du curseur jusqu'au début du fichier.
«**d5G**» : supprime du curseur jusqu'à la 5ème ligne du fichier.

La commande vi

Compléments - Mode commande

• u	.			
• cw	5cw	r	R	
• J				
• yy	5yy	p	P	
• yw	5yw			
• /chaine	n	N		

Compléments - Mode commande

«**u**» : undo - annule le changement précédent.
«**.**» : répète la dernière action.

«**cw**» : change le texte du curseur à la fin du mot. On finalise la saisie par **ESC**.
«**5cw**» : change le texte du curseur jusqu'à la fin du 5ème mot.

«**r**» : remplacement du caractère sous le curseur.
Exemple : «**rA**» remplace le caractère sous le curseur par «**A**».
«**R**» : remplacement d'une suite de caractères.

«**J**» : joint la ligne suivante à la ligne courante.

Copier/coller
«**yy**» : sélection d'une ligne à copier.
«**5yy**» : sélection de 5 lignes à copier.
«**p**» : colle la sélection après le curseur.
«**P**» : colle la sélection avant le curseur.

«**yw**» : sélection d'un mot à copier. «**5yw**», pour 5 mots à copier.

Recherche
«**/xxx**» : pour rechercher un texte «**xxx**» dans le document.
«**n**» : pour passer à l'occurrence suivante.
«**N**» : pour passer à l'occurrence précédente.

La commande vi

Mode ligne

- :q!
- :w :wq
- :w newfic
- :3,10w newfic
- :! commande
- :r fichier :r! commande
- :6
- :4m6 :2,5m10

Mode ligne

Pour accéder au mode ligne, il faut saisir le caractère «**:**». Pour le quitter et revenir au **mode commande**, nous utiliserons la touche «**Entrée**» ou la touche «**Echap**».

Sauvegarder et quitter

:q!		:	quitte l'éditeur sans sauvegarder.
:wq		:	sauvegarde le fichier et quitte l'éditeur.
:w	newfic	:	sauvegarde le fichier dans un nouveau fichier nommé «**newfic**» sans quitter l'éditeur.
:3,10w	newfic	:	créer un nouveau fichier «**newfic**» avec les lignes 3 à 10 du fichier courant.

Interagir avec le Shell

:! commande	:	lance une commande puis revient à l'édition.
:r! commande	:	insère le résultat d'une commande après la ligne du curseur.
:r fichier	:	insère le contenu du fichier après la ligne du curseur.
:6	:	positionne le curseur sur la ligne 6.
:4m8	:	déplacement. La ligne 4 est déplacée après la ligne 8.
:2,5m10	:	déplacement. Les lignes 2 à 5 sont déplacées après la ligne 10.

La commande vi

Mode ligne - suite

- :set nu :set nonu
- :set list :set nolist

- map X 10dd unmap X
- ab rep repertoire unab rep

- 1,$s/ancien/nouveau/g
- %g/filtre/s/ancien/nouveau/g
- %g/filtre/d

Mode ligne - suite

Complément des commandes du mode ligne

:set	**nu**	:	numérote les lignes.
:set	**nonu**	:	supprime la numérotation.
:set	**list**	:	affiche les caractères invisibles.
:set	**nolist**	:	supprime l'affichage des caractères invisibles.
:**map**		:	permet de créer une macro remplaçant l'action d'une touche dans l'éditeur.
:**unmap**		:	pour supprimer une macro créée par map.

Exemple :

```
:map        X          10dd

:map
 X          10dd

:unmap      X
```

Dans l'exemple ci-dessus, nous avons attribué à la touche «**X**» l'action «**10dd**» qui correspond à la suppression de 10 lignes.
Puis nous avons supprimé cette macro grâce à la commande «**unmap**».

:ab : permet d'attribuer une abréviation à une chaîne de caractère.
:unab : permet d'annuler une abréviation.

Exemple :

```
:ab rep repertoire

:ab
! rep repertoire

:unab rep
```

Dans l'exemple ci-dessus, nous avons créé une abréviation pour la chaîne de caractères «**repertoire**».
Désormais, il suffira de rentrer «**rep**» suivi d'une touche de fin de mot («**,**», «**.**», «**Entrée**», etc...) pour que «**vi**» affiche «**repertoire**».
Puis nous avons supprimé l'abréviation grâce à la commande «**unab**».

:1,$s/ancien/nouveau/g :

Permet de modifier toutes les haines de caractères «**ancien**» par «**nouveau**» sur tout le document («**1,$**»).

Au lieu de «**1,$**», nous aurions pu préciser une autre zone de traitement telle que «**5,20**» pour un remplacement uniquement entre les lignes 5 à 20.

Le «**g**» final indique «**globale**» pour toutes les occurrences sur la ligne. En omettant le «**g**», seule la première occurrence sur chaque ligne sera substituée.

:%g/filtre/s/ancien/nouveau/g :

La substitution présentée précédemment sera réalisée sur l'ensemble du fichier uniquement que pour les lignes qui contiennent la chaîne de caractères du filtre.

:%g/filtre/d :

Toutes les lignes qui contiennent la chaîne de caractères du filtre seront supprimées.

La commande vi

Fichier «.exrc»

- Personnalisation du comportement de vi

- vi $HOME/.exrc

  set nu

  set list

  map X 10dd

Fichier « .exrc »

Le fichier «**.exrc**» est un fichier de configuration qui est chargé automatiquement au démarrage de l'éditeur «**vi**».

Ce fichier permet de définir un comportement spécifique à «**vi**» pour un utilisateur. Les actions possibles sont principalement les opérations «**set xxx**», «**map xxx**» et «**ab xxx**».

Exemple :

```
$  vi  $HOME/.exrc
set  nu
set  list
map  X  10dd
```

Notes

Sauvegarde et restauration

Dans ce chapitre nous allons nous familiariser avec l'utilisation des commandes relatives à la sauvegarde et à la restauration.

Sauvegarde et restauration

- Présentation
- La commande «tar»
- La compression

Sauvegarde et restauration

Présentation

- Sauvegardes des données - Restauration
- Importance - Rétention - Fréquence - Lieu
- Des Processus régulièrement testés
- Des outils Time Navigator Baccula

Présentation

La sauvegarde consiste à récupérer les données de l'utilisateur ou d'un projet, et de les stocker dans un lieu sûr. Il est fondamental d'avoir un processus de sauvegarde de ses données, ainsi qu'un processus de restauration.

Il est également à prévoir un niveau de rétention. C'est à dire fixer le nombre d'historique d'une sauvegarde, par exemple : sauvegarde de début du mois (n-1), sauvegarde de début du mois (n), sauvegarde de début de semaine et une sauvegarde de la veille.
Il faut déterminer la fréquence des sauvegardes, par exemple : sauvegardes journalières ou mensuelles, ...
Il faut prévoir la localisation où sont stockées les sauvegardes en prenant en compte les avantages et inconvénients, par exemple :
un jeu de sauvegardes sur son poste : facilement accessible mais indisponible si le poste est défectueux. Il est donc plus pratique que se soit sur un serveur accessible via le réseau.
Un jeu de sauvegardes sur un site distant : peut-être moins facilement accessible mais préserve les données en cas de problème sur le site sur lequel vous vous trouvez.

Les sauvegardes et les restaurations doivent avoir un processus de mise en œuvre détaillé, qui est régulièrement testé et validé.

Les sauvegardes systèmes sont à la charge de l'administrateur, ainsi que celles des données des utilisateurs. Mais pour des besoins ponctuels, l'utilisateur peut s'occuper de ses propres sauvegardes.
En plus des commandes, il existe des outils tels que Time Navigator (tina) ou Baccula.

Sauvegarde et restauration

La commande «tar»

- tar -cvf sauv.tar rep
- tar -tvf sauv.tar
- tar -xvf sauv.tar

La commande «tar»

Présentation

Cette commande nous permet de manipuler des archives dans le système Linux.
Elle permet de concaténer des fichiers dans une seule archive. C'est le principal système d'archivage pour les utilisateurs Linux.

Les options de la commande «tar»

Pour cette commande nous disposons de plusieurs options :

-c	:	pour créer l'archive.
-t	:	pour lister une archive.
-x	:	pour extraire une archive.
-v	:	mode verbeux.
-f	:	pour spécifier le nom de l'archive.
-z	:	compresse l'archive avec «**gzip**» ou -Z compresse l'archive avec **compress**.
-C	:	permet de spécifier le répertoire à partir duquel l'archive est restauré.

Création d'archive / Sauvegarde

Pour créer une archive, il faut utiliser la commande «**tar -cvf**». La sauvegarde sera faite en chemin relatif, la commande supprime automatiquement le caractère «**/**» de la racine si nécessaire.
Dans l'exemple ci-dessous, nous allons créer une archive contenant les fichiers «**fic1**» et «**fic2**»:

Exemple :

```
$ tar      -cvf  sauv.tar   fic1  fic2
```

Pour créer une archive contenant l'arborescence du répertoire «**rep**» on utilise tout simplement :

Exemple :

```
$ tar      -cvf  sauv.tar   rep
```

Lister une archive

Pour afficher la liste des fichiers d'une archive on utilise la commande «**tar -tvf**».
Dans l'exemple ci-dessous nous allons lister l'archive que nous venons de créer.

Exemple :

```
$ tar      -tvf       sauv.tar
-rw-r--r--   user1/groupe1        49      2016-04-06      14:07      fic1
-rw-r--r--   user1/groupe1        49      2016-04-06      14:08      fic2
```

Extraction d'une archive / Restauration

Pour extraire une archive, il faut utiliser la commande «**tar -xvf**».
Dans l'exemple ci-dessous nous allons restaurer tous les fichiers de l'archive «**sauv.tar**».

Exemple :

```
$ tar      -xvf       sauv.tar
```

La restauration est faite en chemin relatif.
Si on souhaite définir explicitement le répertoire à partir duquel la restauration sera fait il faut utiliser l'option «**-C**».

Exemple :

```
$ tar      -xvf       sauv.tar   -C    /tmp
```

Sauvegarde et restauration

La compression

- zip - unzip - zcat
- bzip2 - bunzip2 - bzcat
- gzip - gunzip - gzcat

La compression

Dans ce chapitre nous verrons la compression/décompression d'une archive au format «.**zip**». C'est un ancien format d'archivage. Principalement utilisé sous «**Windows**», il est supporté par le système Linux.

Utilisation de la commande «zip»

Pour créer une archive «.**zip**» nous utiliserons la commande «**zip**».

Exemple :

```
$ zip      sauv.zip   fic1
```

Le fichier «**fic1**» est compressé dans l'archive «**sauv.zip**».

La commande zip est tout particulièrement intéressante pour créer une archive compressé. On peut donc ramener toute une arborescence sous la forme d'un fichier, ce fichier étant en plus compressé au format **zip**.

C'est donc tout particulièrement intéressant lorsque l'on souhaite déplacer une arborescence d'une machine à une autre via le réseau.

Pour décompresser l'archive, nous utiliserons la commande «**unzip**».
Pour spécifier le répertoire ou le fichier sera décompressé, nous utiliserons l'option «**-d**» :

Exemple :

```
$ unzip          sauv.zip   -d    rep1
```

La commande «**zcat**» permet d'afficher le contenu du fichier zippé sans renflement décompresser le fichier. Ce n'est que le résultat qui est envoyé vers la sortie standard.

Utilisation de la commande «bz2»

Pour créer une archive «.**bz2**» nous utiliserons la commande «**bzip2**». Il s'agit d'un format de compression différent de la commande «**zip**».

Exemple :

```
$ bzip2          sauv.bz2   fic1
```

Pour décompresser l'archive, nous utiliserons la commande «**bunzip2**».

Exemple :

```
$ bunzip2        sauv.bz2
```

La commande «**bzcat**» permet d'afficher le contenu du fichier zippé sans renflement décompresser le fichier. Ce n'est que le résultat qui est envoyé vers la sortie standard.

Utilisation de la commande «gzip»

Pour créer une archive «.**gzip**» nous utiliserons la commande «**gzip**». Il s'agit d'un format de compression différent de la commande «**zip**».

Exemple :

```
$ gzip           sauv.gzip  fic1
```

Pour décompresser l'archive, nous utiliserons la commande «**gunzip**».

Exemple :

```
$ gunzip         sauv.gzip
```

La commande «**gzcat**» permet d'afficher le contenu du fichier zippé sans renflement décompresser le fichier. Ce n'est que le résultat qui est envoyé vers la sortie standard.

Notes

Configuration Environnement Utilisateur

Dans ce chapitre nous allons nous familiariser avec la configuration de l'environnement de l'utilisateur : les variables, alias, fichiers.

Configuration Environnement Utilisateur

- Les variables
- Les variables locales et globales
- Les alias
- Les fichiers de personnalisation
- La commande «su»

Configuration Environnement Utilisateur

Les variables

- echo $variable
- env
- Quelques variables
  - HOME LOGNAME
  - PATH HOSTNAME
  - PS1 LANG

Les variables

Les variables contiennent des chaînes de caractères qui permettent de stocker, d'une manière temporaire, des informations en mémoire.
Pour afficher le contenu d'une variable : echo $variable

Variables d'environnements

Les variables d'environnement permettent de configurer le système et un programme.
La commande «**env**» affiche la liste des variables situées dans l'environnement.

Quelques exemples de variables système

HOME	:	affiche le nom du répertoire de connexion.
PATH	:	liste les répertoires où le système va rechercher les commandes.
PS1	:	affiche le prompt principal.
LANG	:	affiche la langue du système.
LOGNAME	:	affiche le nom de l'utilisateur.
HOSTNAME	:	affiche le nom de l'hôte du système.

Configuration Environnement Utilisateur

Les variables locales et globales

- set / unset
- Variables locales / Variables globales
  - export variable=valeur

Les variables locales et globales

set / unset

La commande «**set**» affiche l'ensemble des variables qu'elles soient dans environnement ou non.
La commande «**unset**» annule la définition d'une variable.

Exemple :

```
$  var=bonjour
$  echo  $var
bonjour

$  unset var
$  echo  $var
--- affiche rien ---
```

Variables locales / Variables globales

Les variables globales sont transmises (dupliquées) dans le «**sous-shell**» contrairement aux variables locales.

Ainsi, l'exploitation d'une variable locale ne peut être faite qu'au niveau du shell dans lequel elle a été créée.
La valeur d'une variable globale est exploitable au sein de sous-shells. La commande «**export**» définie une variable comme globale.

Aucune variable n'est transmise au shell père.

Par convention, les variables locales sont notées en minuscules et les variables globales en majuscules.

Exemple de variable locale : village=Salon

Exemple de variable globale : export VILLE=«Aix en Provence»
ou
VILLE=«Aix en Provence»
export VILLE

Exemple :

```
$  village=Salon                          variable locale
$  export  VILLE=«Aix en Provence»        variable globale
$  echo  $village
Salon
$  echo  $VILLE
Aix en Provence

$  bash                                   un sous-shell

$  echo  $village
--- affiche rien ---
$  echo  $VILLE
Aix en Provence

$  VILLE=Toulouse
$  echo  $VILLE
Toulouse

$  sante=Vacances                         variable locale au sous-shell
$  export  PAYS=France                    variable globale au sous-shell
$  echo  $sante
Vacances
$  echo  $PAYS
France

$  exit                                   retour au shell père

$  echo  $village
Salon
$  echo  $VILLE
Aix en Provence
$  echo  $sante
--- affiche rien ---
$  echo  $PAYS
--- affiche rien ---
```

Configuration Environnement Utilisateur

Les alias

- alias nom_de_alias='commande'
- unalias nom_de_alias

Les alias

Un «**alias**» est un raccourci pour une commande ou une séquence de commandes.
Lorsque vous devez répéter une commande, ou que celle-ci est longue à taper, il est possible de définir un «**alias**» pour que ce soit plus pratique à utiliser.

Syntaxe : alias nom_de_alias='commande'

Dans l'exemple ci-dessous, nous allons **créer** un alias pour simplifier une commande :

Exemple :

```
$ alias    lr='ls -lrt'
```

Exemple d'utilisation :

```
$ lr       /var/log
```

C'est équivalent de :

```
$ ls       -lrt       /var/log.
```

Pour supprimer un «**alias**», nous utiliserons la commande «**unalias**».

Exemple :

```
$ unalias  lr
```

Remarque :
La commande alias sans argument liste tous les alias existants.

Configuration Environnement Utilisateur

Les fichiers de personnalisation

- Ouverture d'une session utilisateur
  - /etc/profile
  - $HOME/.bash_profile
  - La variable BASH_ENV
  - $HOME/.bashrc
- Fermeture d'une session utilisateur
  - $HOME/.logout

Les fichiers de personnalisation

Plusieurs fichiers peuvent être exploités afin de personnaliser l'environnement d'un utilisateur.
Au sein de ces fichiers, nous pouvons définir des variables, des alias et/ou avoir du scripting shell.

Ordre de traitement des fichiers de personnalisation lors de l'ouverture d'une session bash :

/etc/profile	fichier modifiable que par root, permet de centraliser tout ce qui sera commun aux utilisateurs bash.
$HOME/.bash_profile	s'il existe, modifiable par l'utilisateur.

Au sein de ce fichier peut être défini la variable BASH_ENV, en général positionnée à la valeur «$HOME/.bashrc»

$HOME/.bashrc	le fichier défini par la variable BASH_ENV, s'il existe. Fichier modifiable par l'utilisateur.

Fichier traité lors du lancement d'un sous-shell bash :

$HOME/.bashrc	le fichier défini par la variable BASH_ENV, s'il existe.

Fichier traité lors de la fermeture d'une session bash :

$HOME/.bash_logout	s'il existe, modifiable par l'utilisateur.

Configuration Environnement Utilisateur

La commande «su»

- Prendre l'identité d'un autre utilisateur
- su user2
- su - user2
- su ou su root
- su - ou su - root

La commande «su»

La commande «**su**» permet de prendre l'identité d'un autre utilisateur, avec ou sans son environnement de travail.

L'utilisation du tiret au sein de la commande su permet de traiter les fichiers de personnalisation de l'utilisateur. On récupère ainsi son environnement de travail en plus de son identité.

Exemple :

```
user1$  whoami
user1
user1$  pwd
/home/user1
user1$  echo  $village          variable définie par un fichier de
--- affiche rien ---            personnalisation de jeanmarc

user1$  su  jeanmarc
Mot de passe : xxx              Mot de passe de jeanmarc

jeanmarc$  whoami
jeanmarc
jeanmarc$  logname
user1
jeanmarc$  pwd
/home/user1
jeanmarc$  echo  $village
--- affiche rien ---
jeanmarc$  exit
```

```
user1$  whoami
user1
user1$  pwd
/home/user1
user1$  echo  $village
--- affiche rien ---

user1$  su  -  jeanmarc
Mot de passe : xxx                    Mot de passe de jeanmarc

jeanmarc$  whoami
jeanmarc
jeanmarc$  logname
user1
jeanmarc$  pwd
/home/jeanmarc
jeanmarc$  echo  $village
Salon
jeanmarc$  exit
user1$
```

Remarque

L'utilisation de la commande «**su**» sans arguments, c'est a dire sans spécifier le nom de l'utilisateur, permet de prendre l'identité du compte «**root**». Évidemment il vous sera demandé le mot de passe de «**root**».

```
user1$  su                            ou :  su  root
Mot de passe : xxx                    Mot de passe de root
#
…
#  exit
user1$


user1$  su  -                         ou :  su  -  root
Mot de passe : xxx                    Mot de passe de root
#
…
#  exit
user1$
```

Notes

Le service d'impression

Dans ce chapitre nous allons nous familiariser avec l'utilisation du service d'impression.

Le service d'impression

- Principe
- La commande «lp»
- Les commandes «lpstat» et «cancel»

Le service d'impression

Principe

- Imprimante locale distante réseau
- Répertoire de spool
- Client d'impression et serveur d'impression
- Classe d'imprimantes

Principe

Comme son nom l'indique, le service d'impression permet de gérer les requêtes d'impressions au sein du système Linux.
C'est un service complet qui permet de gérer les demandes d'impressions, les listes d'impressions, ainsi que l'administration des imprimantes vis à vis du système d'exploitation.
'root' peut tout administrer, mais un utilisateur pourra faire une demande d'impression et gérer uniquement que ses requêtes d'impressions.

Terminologie

L'imprimante locale : imprimante directement connectée à votre machine.
L'imprimante distante : imprimante connectée à une machine distante.
L'imprimante réseau : imprimante connectée au réseau.

Le répertoire de spool : répertoire où sont stockées les requêtes en attente d'impression.

Le client d'impression : service d'où est faite la demande d'impression.
Le serveur d'impression : service qui reçoit les demandes d'impression d'un client et qui va gérer cette requête.

La classe d'imprimantes : rassemble plusieurs imprimantes dans un seul groupe. La demande d'impression est faite à une classe d'imprimantes et non plus à une imprimante en particulier.

Le service d'impression

La commande «lp»

- lp fichier

- lp -d imprimante fichier
- lp -n2 fichier
- lp -m fichier
- lp -q 100 fichier

La commande «lp»

La commande «**lp**» soumet des fichiers pour une impression. Elle génère donc une requête d'impression qui sera localisée dans le répertoire de spool de l'imprimante avant impression.

Syntaxe de base: lp fichier

Quelques options de «lp»

-d : permet de sélectionner l'imprimante sur lequel le fichier sera imprimé.

```
$ lp -d imp1 fichier
request id is imp1-12 (1 file(s))
```

-n : permet de choisir le nombre d'exemplaires à imprimer.

```
$ lp -n 2 -d imp1 fichier
request id is imp1-13 (1 file(s))
```

-m : permet d'envoyer un email lorsque l'impression est terminée.

```
$ lp -m fichier
request id is imp1-14 (1 file(s))
```

-q : permet de définir la priorité pour la requête d'impression.
par défaut 50, 1 (minimale) à 100 (maximale).

```
$ lp -d imp1 fichier -q 100 fichier
request id is imp1-15 (1 file(s))
```

Le service d'impression

Les commandes «lpstat» et «cancel»

- lpstat -t -o -d -a

```
$ lpstat -t
scheduler is running
system default destination: imp1
device for imp1: usb://Brother/HL-4150CDN%20series
device for imp2: usb://Brother/HL-4150CDN%20series
imp1 accepting requests since Fri 29 Apr 2016 11:54:31 AM CEST
imp2 accepting requests since Fri 29 Apr 2016 11:54:35 PM CEST
printer imp1 is idle.  enabled since Fri 29 Apr 2016 11:54:31 AM CEST
      Printer is now online.
printer imp2 is idle.  enabled since Fri 29 Apr 2016 11:54:35 PM CEST
      Printer is now online.
imp2-8           user1            1024   Fri 29 Apr 2016 12:01:28 PM CEST
imp2-9           user1            3072   Fri 29 Apr 2016 12:01:32 PM CEST
imp2-10          user1            3072   Fri 29 Apr 2016 12:01:45 PM CEST
```

- cancel numero_requete

Les commandes «lpstat» et «cancel»

La commande «lpstat»

Cette commande affiche des informations sur le service d'impression, les différentes imprimantes déclarées et la liste des requêtes d'impressions en attente. Par défaut (sans arguments), elle affiche les requêtes de l'utilisateur qui sont en attente.

Les options de la commande «lpstat»

-t : affiche la configuration du système d'impression et la liste des requêtes.

```
$ lpstat -t
scheduler is running
system default destination: imp1
device for imp1: usb://Brother/HL-4150CDN%20series
device for imp2: usb://Brother/HL-4150CDN%20series
imp1 accepting requests since Fri 29 Apr 2016 11:54:31 AM CEST
imp2 accepting requests since Fri 29 Apr 2016 11:54:35 PM CEST
printer imp1 is idle.  enabled since Fri 29 Apr 2016 11:54:31 AM CEST
      Printer is now online.
printer imp2 is idle.  enabled since Fri 29 Apr 2016 11:54:35 PM CEST
      Printer is now online.
imp2-8           user1            1024   Fri 29 Apr 2016 12:01:28 PM CEST
imp2-9           user1            3072   Fri 29 Apr 2016 12:01:32 PM CEST
imp2-10          user1            3072   Fri 29 Apr 2016 12:01:45 PM CEST
```

-o : affiche la file d'attente des requêtes d'impression.

```
$ lpstat -o
imp2-8                  user1             1024   Fri 29 Apr 2016 12:01:28 PM CEST
imp2-9                  user1             3072   Fri 29 Apr 2016 12:01:32 PM CEST
imp2-10                 user1             3072   Fri 29 Apr 2016 12:01:45 PM CEST
```

-d : indique l'imprimante par défaut.

```
$ lpstat -d
system default destination: imp1
```

-a : liste des imprimantes installées.

```
$ lpstat -a
imp1 accepting requests since Fri 29 Apr 2016 11:54:31 AM CEST
imp2 accepting requests since Fri 29 Apr 2016 11:54:35 PM CEST
```

La commande «cancel»

Cette commande permet d'annuler des requêtes d'impressions dans la file d'attente.
L'utilisateur peut supprimer que les requêtes qui lui appartiennent.

Exemples :

```
$ cancel imp1-12
$
$ cancel imp1-13 imp1-14 imp1-15
$
```

Notes

Les processus

Dans ce chapitre nous allons traiter le mécanisme de fonctionnement et de gestion des processus ainsi que du traitement différé de commandes.

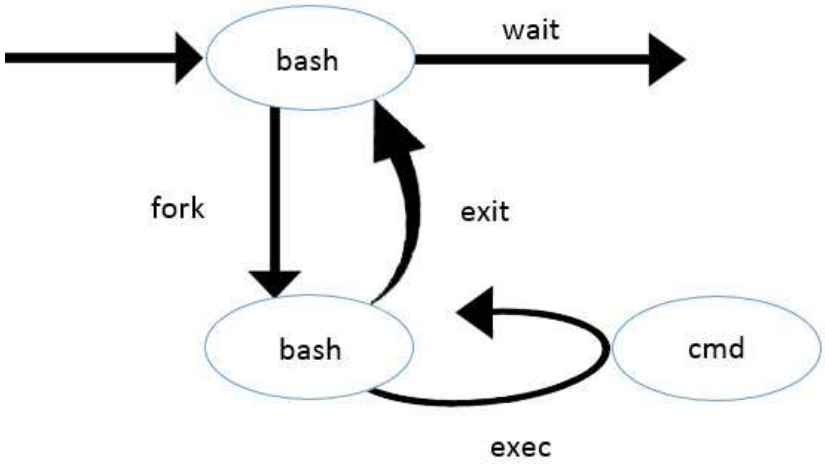
Les processus

- Définition
- Les états d'un processus
- Les commandes «ps» et «pgrep»
- Les commandes «kill» et «pkill»
- Commandes supplémentaires
- Présentation du «&» et du «;»
- Les jobs
- La commande «at»
- La fonctionnalité «crontab»

Les processus

Définition

- PID PPID UID
- Multitâche
- Scheduler



- Appels systèmes fork() exec() wait() exit()

Définition

Un processus est toute tâche exécutée par le système d'exploitation.
Un **thread** est un traitement spécifique au sein d'un processus.

Attributs des processus

Il existe plusieurs attributs pour identifier un processus :

PID	:	c'est un numéro unique permettant d'identifier le processus.
PPID	:	c'est le numéro du processus parent.
UID	:	c'est le numéro correspondant au propriétaire du processus.

Multitâche

On parle d'un système d'exploitation multitâche lorsque celui-ci peut exécuter plusieurs programmes de façon simultané.
C'est un partage équitable du temps unité centrale entre les différents processus, appelé «**time sharing**».

Scheduler

Le «**scheduler**» est un processus du noyau. Il permet de distribuer du temps CPU aux processus actifs du système d'exploitation.
Si un processus est inactif, le «**scheduler**» est averti et passe ce processus en état «**dormant**».

Appels systèmes

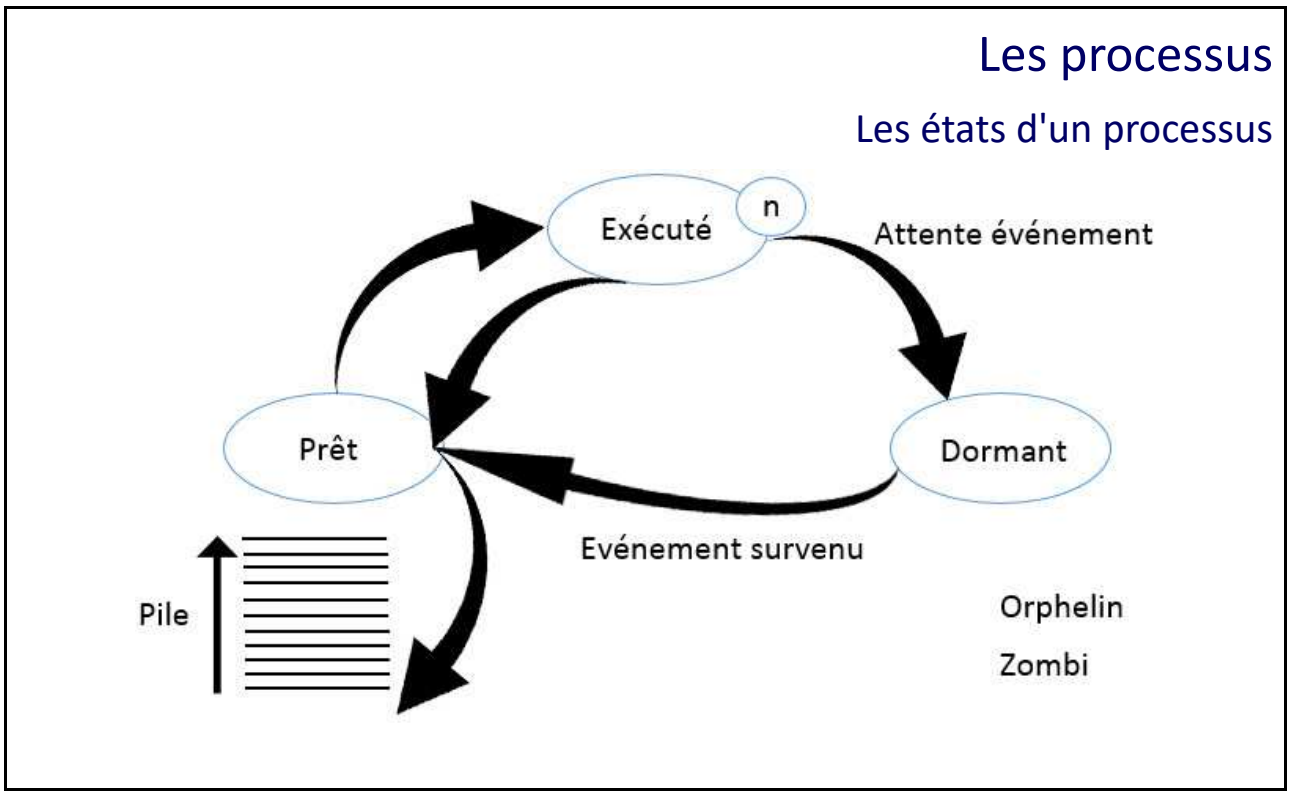
La gestion des processus est réalisée par le noyau via les appels systèmes :

- «**fork()**» : sert à créer un processus fils, identique au père, pour l'exécution et le traitement spécifique du processus dans son propre environnement. Le code de retour de cet appel système est exploité par le processus père à la fin (la mort) du processus fils.

- «**exec()**» : sert à récupérer le code du processus et à l'exécuter.

- «**wait()**» : sert au processus père à attendre la fin du processus fils.

- «**exit()**» : sert à terminer le processus (libère les ressources, le **PID** de la table des processus, etc), renvoie le code de retour du processus fils au processus père, et permet ainsi le déblocage de l'appel système **wait** du processus père.



Les états d'un processus

Les différents états d'un processus

- «**exécuté**» : le processus est en train de s'exécuter sur un cœur d'un processeur.
- «**prêt**» ou «**runnable**» : le processus est dans une pile, en attente d'exécution sur un cœur d'un processeur.
- «**dormant**» ou «**sleeping**» ou «**not runnable**» : le processus est hors de la pile «**runnable**» car il est en attente d'un événement.
- «**orphelin**» : le processus a son processus père qui est «**mort**». Il ne peut donc pas lui retourner son code de retour lors de sa propre fin d'exécution. Ainsi pour cette situation qui reste exceptionnelle, le processus est automatiquement rattaché au processus père de pid 1 (init).
- «**zombi**» : le processus est «**mort**» mais le processus père n'en a pas été informé. Le problème d'un tel état de processus est, en autres, que le numéro de **PID** n'a pas été libéré.

Les processus

Les commandes «ps» et «pgrep»

- Commande «ps»
  - Options : -l -ef -aux
- Commande «pgrep»
  - Options : -t -u -l

Les commandes «ps» et «pgrep»

La commande «ps»

Cette commande permet d'afficher tout les processus en cours d'exécution.

Les options de «**ps**» :

-l	:	permet un affichage long et détaillé des processus.		
-e	:	permet d'afficher tout les processus.		
-f	:	permet d'afficher toutes les informations disponibles.		
-aux	:	-a	:	liste l'ensemble des processus.
		-u	:	affiche l'utilisateur du processus.
		-x	:	affiche les processus sans terminal.

Exemples :

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 13:26 ?        00:00:01 /usr/lib/systemd/systemd --switc
root         2     0  0 13:26 ?        00:00:00 [kthreadd]
root       688     1  0 13:27 ?        00:00:00 /usr/sbin/crond -n
root       689     1  0 13:27 ?        00:00:00 /usr/sbin/atd -f
user1     2892  2621  2 13:29 ?        00:00:21 /usr/bin/gnome-shell
root      2897     1  0 13:29 ?        00:00:00 /usr/sbin/cupsd -f
user1     3294  3277  0 13:30 pts/0    00:00:00 /bin/bash
user1     3605  3362  0 13:45 pts/1    00:00:00 ps -ef
```

```
$  ps  -aux
USER     PID %CPU  %MEM     VSZ      RSS   TTY   STAT  START  TIME     COMMAND
root       1  0.1   0.4    53668     7596   ?      Ss   13:26  0:01     *
                  */usr/lib/systemd/systemd--switched-root --system --deserialize 24
root       2  0.0   0.0        0        0   ?      S    13:26  0:00     [kthreadd]
root     688  0.0   0.0   126292     1700   ?      Ss   13:27  0:00     /usr/sbin/crond -n
root     689  0.0   0.0    25928      968   ?      Ss   13:27  0:00     /usr/sbin/atd -f
user1   2892  1.8  11.7  1560600   222400   ?      Sl   13:29  0:21     /usr/bin/gnome-shell
root    2897  0.0   0.2   178032     3864   ?      Ss   13:29  0:00     /usr/sbin/cupsd -f
user1   3294  0.0   0.1   116264     2876  pts/0   Ss+  13:30  0:00     /bin/bash
user1   3631  0.0   0.0   125440     1420  pts/1   R+   13:48  0:00     ps -aux
```

La commande «pgrep»

Cette commande permet de rechercher les processus en cours d'exécution selon un critère de recherche.

Les options de «**pgrep**» :

-t : permet de sélectionner les processus dont le terminal est listé.
-u : permet de sélectionner les processus dont l'**UID** est listé.
-l : permet de lister le nom du processus avec son identifiant.
-x : permet de sélectionner les processus exact au nom du motif.

Exemples :

```
$ pgrep    -l    bash
3892 bash
3946 bash
7836 bash

$ pgrep    -l    -t pts/1   bash
3946 bash

$ pgrep    -l    -u user1   bash
3892 bash
3946 bash
$ pgrep    -l    -u root    bash
7836 bash

$ pgrep    -l    -x    bash
3892 bash
3946 bash
7836 bash
$ pgrep    -l    -x    ba
$
```

Les processus

Les commandes «kill» et «pkill»

- kill PID_du_programme
- Les signaux
  kill -15 PID_du_programme
  kill -9 PID_du_programme
- Commande «pkill»
  pkill nom_du_programme
  pkill -9 -x mail

Les commandes «kill» et «pkill»

Cette commande permet de communiquer avec un processus. Il faut être le propriétaire du processus pour communiquer avec, ou être 'root'.
Par défaut cela permet de tuer un ou plusieurs processus actifs. Il est nécessaire de connaître le **PID** du processus pour pouvoir le terminer avec cette commande.

Syntaxe : kill pid1
kill pid1 pid2 pid3

Les signaux

Un signal correspond à l'action à entreprendre sur un processus.
La commande «**kill -l**» permet de lister les signaux.
Pour avoir accès au manuel des signaux, il faut utiliser la commande «**man -s7 signal**».

Exemple :

```
$ kill -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
Etc…
```

Exemple :

```
$ man      -s7   signal

SIGNAL(7)                Manuel du programmeur Linux                SIGNAL(7)
…
  Signal      Valeur       Action       Commentaire
──────────────────────────────────────────────────────────────────────────
 SIGHUP       1            Term         Déconnexion détectée sur le terminal
                                        de contrôle ou mort du processus de
                                        contrôle.
 SIGINT       2            Term         Interruption depuis le clavier.
 SIGQUIT      3            Core         Demande « Quitter » depuis le clavier.
 SIGILL       4            Core         Instruction illégale.
 SIGABRT      6            Core         Signal d'arrêt depuis abort(3).
 SIGFPE       8            Core         Erreur mathématique virgule flottante.
 SIGKILL      9            Term         Signal « KILL ».
 SIGSEGV      11            Core        Référence mémoire invalide.
 SIGPIPE      13           Term         Écriture dans un tube sans lecteur.
 SIGALRM      14           Term         Temporisation alarm(2) écoulée.
 SIGTERM      15           Term         Signal de fin.
 SIGUSR1      30,10,16     Term         Signal utilisateur 1.
 SIGUSR2      31,12,17     Term         Signal utilisateur 2.
 SIGCHLD      20,17,18     Ign          Fils arrêté ou terminé.
 SIGCONT      19,18,25     Cont         Continuer si arrêté.
 SIGSTOP      17,19,23     Stop         Arrêt du processus.
 SIGTSTP      18,20,24     Stop         Stop invoqué depuis le terminal.
 SIGTTIN      21,21,26     Stop         Lecture sur le terminal en arrière-plan.
 SIGTTOU      22,22,27     Stop         Écriture dans le terminal en arrière-plan.
Etc…
```

Les signaux les plus couramment utilisés sont :

Le signal **TERM** (15) est un signal de terminaison classique. Il peut être ignoré par le processus.

Syntaxe : kill -15 PID_du_programme

Le signal **KILL** (9) qui correspond à l'arrêt immédiat du processus. Utilisé lorsque le signal **TERM** échoue.

Syntaxe : kill -9 PID_du_programme

La commande «pkill»

Cette commande a le même effet que «**kill**», mais simplifiée car il n'exige pas de connaître le **PID** du processus. La plupart des options de «**pgrep**» sont également disponible pour cette commande.

Syntaxe : pkill nom_du_programme
pkill -Signal nom_du_programme

Exemples : pkill mail

Dans l'exemple ci -dessus, «**pkill**» envoie le signal 15 à tous les processus qui contiennent «**mail**» dans leur nom.

pkill -9 -x mail

Dans l'exemple ci-dessus, «**pkill**» envoie le signal 9 à tous les processus qui s'appellent exactement «**mail**».

Les processus

Commandes supplémentaires

- pstree
- top
- uptime

Commandes supplémentaires

La commande «pstree»

Permet d'afficher l'arborescence des processus. Cette commande est pratique pour connaître de quel processus dépend un **PID**.

Exemple :

```
$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager─┬─dhclient
        │                └─3*[{NetworkManager}]
        ├─2*[abrt-watch-log]
        ├─abrtd
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─alsactl
        ├─at-spi-bus-laun─┬─dbus-daemon───{dbus-daemon}
        │                 └─3*[{at-spi-bus-laun}]
...etc
```

La commande «top»

Permet d'afficher les processus sur une page. Par défaut, ils sont triés dans l'ordre décroissant du taux d'utilisation CPU. L'affichage se rafraîchi régulièrement.

Quelques options :

-q : permet de quitter la commande «**top**».
-h : affiche l'aide.
-f : ajoute ou supprime des colonnes.
-u : filtre en fonction de l'utilisateur.
-k : tue un processus.
-s : change l'intervalle de temps de rafraîchissement de la liste (3 secondes par défaut).

Exemple :

```
$ top
top - 15:19:43 up  6:11,  3 users,  load average: 0,00, 0,01, 0,05
Tasks: 154 total,   3 running, 151 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:   1885264 total,  1778880 used,   106384 free,      148 buffers
KiB Swap:  2113532 total,        0 used,  2113532 free.  1175304 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0   53672   7616   2520 S  0,0  0,4   0:01.60 systemd
    2 root      20   0       0      0      0 S  0,0  0,0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S  0,0  0,0   0:00.20 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kworker/u2:0
    7 root      rt   0       0      0      0 S  0,0  0,0   0:00.00 migration/0
    8 root      20   0       0      0      0 S  0,0  0,0   0:00.00 rcu_bh
```

La commande «uptime»

Permet d'indiquer des informations sur le système, l'heure actuelle, depuis combien de temps le système est en marche, le nombre d'utilisateurs connectés et la charge du système.
La charge du système (load average) nous informe sur le nombre de processus en attente de ressources pour les 1, 5 et 15 dernière minutes.

```
$ uptime
11:11:17 up       1:32,      3 users,      load average:   0,00,    0,01,    0,05
```

La commande «time»

Permet de mesurer le temps d'exécution d'une commande. Trois résultats sont affichés :

real : affiche le temps total.
user : affiche le temps nécessaire au processeur pour exécuter les directives du programme.
sys : affiche le temps nécessaire au processeur pour traiter les directives du système.

```
$ time    sleep 5
real        0m5.004s
user        0m0.001s
sys         0m0.002s
```

Les processus

Présentation du «&» et du «;»

- & Arrière plan

  cp vidéo1 copie_vidéo1 **&**

- ; Exécution séquentielle

  cp video1 videoA **;** cp video2 videoB **;** cp son sonA

Présentation du «&» et du «;»

Arrière plan

Pour des raisons pratiques, il est possible de lancer des processus en arrière plan. Pour cela nous utiliserons le symbole «**&**» (et commercial).

Dans l'exemple ci-dessous, je souhaite lancer une copie d'un fichier (une vidéo) en arrière plan.

```
$ cp vidéo1 copie_vidéo1 &
[1]   16504
```

Nous pouvons voir l'apparition d'une ligne d'informations.
Le **[1]** nous indique que c'est le premier processus que nous envoyons en arrière plan (numéro de jobs). Le nombre **16504** est le **PID** de ce processus.

Exécution séquentielle

Le caractère «**;**» réalise des exécutions séquentielles de commandes.

```
$ cp video1 videoA ; cp video2 videoB ; cp son sonA
```

Dans l'exemple ci-dessus, une seule ligne de commande nous permet de copier plusieurs fichiers.

Les processus

Les jobs

- Commande «fg»
- Commande «bg»
- Le «Ctrl + Z»
- Le «Ctrl + C»

Les jobs

La commande «**jobs**» permet de connaître les processus qui sont exécutés en arrière plan.

```
$ jobs
[1]+      Fini            cp   fic1   copiefic1
```

Dans l'exemple ci-dessus, «**jobs**» nous indique que le fichier à été copié et nous informe sur la ligne de commande entrée pour cette copie.

```
$ sleep 1001&
[1] 3418
$ sleep 1002&
[2] 3419
$ sleep 1003&
[3] 3420

$ jobs
[1]   En cours d'exécution   sleep 1001 &
[2]-  En cours d'exécution   sleep 1002 &
[3]+  En cours d'exécution   sleep 1003 &
```

Tous ces jobs sont en cours d'exécution. Le caractère «**+**» indique le dernier processus lancé, et le caractère «**-**» l'avant dernier.

La commande «fg»

Permet de passer le processus en premier plan.
Si vous avez lancé une commande en arrière plan et que vous voulez la passer en premier plan, vous devrez utiliser la commande «**fg**» suivi du numéro du **jobs** à traiter.

```
$ sleep 1001&
[1] 3813

$ jobs
[1]+  En cours d'exécution   sleep 1001 &

$ fg %1
sleep 1001
```

Dans l'exemple ci-dessus, nous n'avons plus le prompt car la commande 'sleep', le job 1, est passée en premier plan.

La commande «bg»

Permet de passer le processus en arrière plan.
Si vous avez lancé une commande en premier plan et que vous voulez la passer en arrière plan, vous devrez la mettre en pause et récupérer l'invite de commande grâce à «**Ctrl + z**».
Puis exécuter «**bg**» pour que le processus soit relancée en arrière plan.

```
$ jobs                                      liste les jobs.
[1]+  En cours d'exécution   sleep 1001 &

$ fg  %1                                    bascule le job 1 en premier plan.
Sleep 1001
^z                                          freeze la commande en premier plan.
[1]+  Stoppé                 sleep 1001

$                                           on a «repris la main» sur le prompt.

$ jobs
[1]+  Stoppé                 sleep 1001

$ bg  %1                                    bascule le job 1 en arrière plan.
[1]+ sleep 1001 &

$ jobs
[1]+  En cours d'exécution   sleep 1001 &
```

Le «Ctrl + Z»

Matérialisé dans le terminal avec les caractères «**^Z**».
Cette commande permet de mettre en pause l'exécution du programme qui est en premier plan et de le basculer en arrière plan.
Cette commande est utile lorsque l'on veut passer un programme de premier plan en arrière plan.

```
$ programme
^Z
[1]+   Stoppé        programme
$
```

Après le «**Ctrl + Z**», nous avons récupéré le prompt car le programme est suspendu en arrière plan.

Le «Ctrl + C»

Matérialisé dans le terminal avec les caractère «**^C**».
Cette commande permet d'arrêter le programme en cours d'exécution en premier plan.

```
$ programme
^C
$
```

Après le «**Ctrl + C**», nous avons récupéré le prompt car le programme ne fonctionne plus.

Les processus

La commande «at»

- Présentation

```
[user1]$ at 18:30
at> echo bonsoir > /dev/pts/1
at> <EOT>
job 4 at Wed Oct 28 18:30:00 2015
```

- Commandes «atq» et «at -l»
- Commandes «atrm» et «at -d»
- Les fichiers at.deny / at.allow

La commande «at»

La commande «**at**» permet d'exécuter une commande de manière ponctuelle à un moment différé. Elle est usuellement suivie de l'heure à laquelle la tâche doit être programmée. Si l'heure est passée vous programmez la tâche pour le lendemain. Il est possible de spécifier une date en plus de l'heure.

Syntaxe : $ at [-m] moment_a_exécuter
at> commande1
at> commande2
at> ^D

«moment a exécuter» peut être :

11:15	pour indiquer la prochaine fois où il sera 11h15
11:15 June 16	pour indiquer le 16 juin à 11h15

On peut utiliser midnight, noon, teetime, ow, today, tomorrow ou les jours de la semaine.

now + 30 minutes	pour indiquer dans 30 minutes.
11:45 Saturday	

L'option «**m**» permet d'envoyer un émail à l'utilisateur lorsque la commande sera exécutée.

Exemples :

```
[user1]$ at 18:30
at> echo bonsoir > /dev/pts/1
at> <EOT>
job 4 at Wed Oct 28 18:30:00 2015
```

```
[user1]$ at teatime
at> echo bonjour > /dev/pts/1
at> <EOT>
job 5 at Thu Oct 29 16:00:00 2015
```

```
[user1]$ at now +5days
at> echo coucou
at> <EOT>
job 6 at Mon Nov  2 16:08:00 2015
```

```
[user1]$ at 5am tomorrow
at> echo hello
at> <EOT>
job 7 at Thu Oct 29 05:00:00 2015
```

```
[user1]$ at 16:30 122415
at> echo joyeux noel
at> <EOT>
job 8 at Thu Dec 24 16:30:00 2015
```

```
$ echo "tar -rvf archive.tar fic2" | at 1457
job 1 at Mon Apr 18 14:57:00 2016
```

Une archive nommé «**archive.tar**» contenant le fichier «**fic2**» sera créé à 14h57.

Les commandes «**atq**» et «**at -l**» listent les tâches «**at**» en attente d'exécution.

```
[user1]$ atq
6       Mon Nov  2 16:08:00 2015 a user1
4       Wed Oct 28 18:30:00 2015 a user1
5       Thu Oct 29 16:00:00 2015 a user1
8       Thu Dec 24 16:30:00 2015 a user1
7       Thu Oct 29 05:00:00 2015 a user1
```

```
[user1]$ at -l
6       Mon Nov  2 16:08:00 2015 a user1
4       Wed Oct 28 18:30:00 2015 a user1
5       Thu Oct 29 16:00:00 2015 a user1
8       Thu Dec 24 16:30:00 2015 a user1
7       Thu Oct 29 05:00:00 2015 a user1
```

La commande «**atrm**» ou «**at -d**» supprime une tâche.

```
[user1]$ atrm 4
[user1]$ atrm 6
[user1]$ atq
5       Thu Oct 29 16:00:00 2015 a user1
8       Thu Dec 24 16:30:00 2015 a user1
7       Thu Oct 29 05:00:00 2015 a user1
```

Les fichiers de configurations

Les tâches «**at**» en attente d'exécution sont stockées dans le répertoire **/var/spool/at**.

```
# ls /var/spool/at
a00003016fbffc  a00005016fc524  a00007016fc290  a0000801710042  spool
```

Par défaut, tous les utilisateurs ont le droit d'utiliser la commande «**at**» à l'exception de ceux qui sont listés dans le fichier **/etc/at.deny** (vide par défaut). Il n'y a que root qui peut mettre à jour ce fichier.

```
# more /etc/at.deny
user3
user1
```

```
[user3]$ at 15:30
You do not have permission to use at.
```

Il peut également exister le fichier **/etc/at.allow**. Dans ce cas, seuls les utilisateurs listés dans le fichier auront le droit d'utiliser la commande «**at**». Il n'y a que root qui peut manipuler ce fichier.

```
$ more /etc/at.allow
user3
```

```
[user3]$ at 13:40
at> echo coucou
at> <EOT>
job 9 at Thu Oct 29 13:40:00 2015
```

```
[user5]$ at 15:20
You do not have permission to use at.
```

Il n'y a que root qui a le droit d'utiliser la commande «**at**» si aucun des deux fichiers n'existent (at.deny et at.allow).

Les processus

La fonctionnalité «crontab»

- Présentation

```
[theo@formateur ~]$ crontab -l
20     10     *      *      *      /scripts/script1.bash
10     00     1      *      *      /scripts/script_sauve.bash
*/10   *      *      *      0      /scripts/script2.bash
20,40  10,20  *      *      *      /scripts/script3.bash
15,45  14-16  *      *      1-5    /scripts/script4.bash
```

- Sauvegarde
- Suppression
- Restauration
- Les fichiers cron.deny / cron.allow

La fonctionnalité «crontab»

Présentation

Une crontab permet d'exécuter des tâches de manière récurrente. Chaque utilisateur peut posséder une crontab.

Chaque fichier crontab possède 5 champs pour définir à quel moment la tâche doit être lancée.

```
[theo@formateur ~]$ crontab -l
20 10 * * * /scripts/script1.bash
10 00 1 * * /scripts/script_sauve.bash
*/10 * * * 0 /scripts/script2.bash
20,40 10,20 * * * /scripts/script3.bash
15,45 14-16 * * 1-5 /scripts/script4.bash
```

Le premier champ représente le champ **minutes**. Il peut prendre les valeurs de **0 à 59.**
Le deuxième champ représente le champ **heures**. Il peut prendre les valeurs de **0 à 23**.
Le troisième champ représente le champ **jour du mois**. Il peut prendre les valeurs **1 à 31.**
Le quatrième champ représente le champ **mois**. Il peut prendre les valeurs **1 à 12.**
Le cinquième champ représente le champ **jour de la semaine**. Il peut prendre les valeurs **0 à 7.** 0 étant le dimanche, 1 le lundi, …

script1.bash est exécuté tous les jours à 10h20
script_sauve.bash est exécuté tous les 1er du mois à 00h10
script2.bash est exécuté toutes les 10 minutes tous les dimanches.
script3.bash est exécuté tous les jours à 10h20, 10h40, 20h20, 20h40
script4.bash est exécuté tous les lundi à vendredi à 14h15, 14h45, 15h15, 15h45, 16h15, 16h45

Pour éditer sa crontab, il faut exécuter «**crontab -e**». La crontab est éditée avec «**vi**» par défaut. Pour paramétrer un autre utilitaire, il faut configurer la variable **EDITOR**.

```
[theo@formateur ~]$ export EDITOR=/usr/bin/nano
```

Sauvegarde d'une crontab

Sauvegarde de la crontab de theo dans le fichier **/tmp/crontab.txt** :

```
[theo@formateur ~]$ crontab -l  > /tmp/crontab.txt
[theo@formateur ~]$ more /tmp/crontab.txt
20 10 * * * /scripts/script1.bash
10 00 1 * * /scripts/script_sauve.bash
*/10 * * * 0 /scripts/script2.bash
20,40 10,20 * * * /scripts/script3.bash
15,45 14-16 * * 1-5 /scripts/script4.bash
```

Suppression d'une crontab – option -r

```
[theo@formateur ~]$ crontab -r
[theo@formateur ~]$ crontab -l
no crontab for theo
```

Restauration d'une crontab

```
[theo@formateur ~]$ crontab  /tmp/crontab.txt
[theo@formateur ~]$ crontab -l
20 10 * * * /scripts/script1.bash
10 00 1 * * /scripts/script_sauve.bash
*/10 * * * 0 /scripts/script2.bash
20,40 10,20 * * * /scripts/script3.bash
15,45 14-16 * * 1-5 /scripts/script4.bash
```

Les fichiers de configurations

Les crontabs sont stockées dans le répertoire **/var/spool/cron**. Un fichier portant le nom de l'utilisateur possédant une crontab est présent.

```
# ls /var/spool/cron
theo  user5
```

```
# more /var/spool/cron/theo
20 10 * * * /scripts/script1.bash
10 00 1 * * /scripts/script_sauve.bash
*/10 * * * 0 /scripts/script2.bash
20,40 10,20 * * * /scripts/script3.bash
15,45 14-18 * * 1-5 /scripts/script4.bash
```

Tous les utilisateurs ont le droit d'utiliser la crontab à l'exception de ceux spécifiés dans **/etc/cron.deny** (vide par défaut). Il n'y a que root qui peut manipuler ce fichier.

```
# more /etc/cron.deny
user1
user2
user3
```

```
[user3@formateur ~]$ crontab -e
You (user3) are not allowed to use this program (crontab)
See crontab(1) for more information
```

Le comportement peut être inversé en créant le fichier **/etc/cron.allow** (action faisable que par root). Alors, il n'y aura que les utilisateurs présents dans ce fichier qui pourront utiliser la commande crontab.

```
# more /etc/cron.allow
user2
user3
```

```
[user3@formateur ~]$ crontab -l
*/5 * * * * echo bonjour
```

```
[user1@formateur ~]$ crontab -l
You (user1) are not allowed to use this program (crontab)
See crontab(1) for more information
```

Si aucun des deux fichiers existe (/etc/cron.deny et /etc/cron.allow), seul root peut utiliser la commande crontab.

Le système possède une crontab qui est /etc/crontab. Il est actuellement vide. A l'origine, il était utilisé pour exécuter les commandes situés dans les répertoires /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly et /etc/cron.yearly (n'existent plus).

C'est «**anacron**» qui exécute le contenu de certains de ces répertoires comme on peut le constater dans le fichier /etc/anacrontab.

Notes

Expressions régulières et les commandes grep

caractères ainsi que les commandes «grep», «fgrep» et «egrep».

Les expressions régulières et les commandes grep

- Commande «grep»
- Expressions régulières
- Commandes «fgrep» et «egrep»

Les expressions régulières et les commandes grep

Commande «grep»

- Introduction
- grep 'chaîne_de_caractères' nom_du_fichier
- Options

  -i -v -l -c -q

Commande «grep»

Introduction

Cette commande permet de filtrer le contenu d'un fichier.
Elle permet donc d'afficher toute les lignes qui contiennent la chaîne de caractère au sein d'un fichier.

Syntaxe : grep 'chaîne_de_caractères' nom_du_fichier

Exemples :

```
$ grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

```
$ grep 'root' /etc/passwd /etc/group
/etc/passwd:root:x:0:0:root:/root:/bin/bash
/etc/passwd:operator:x:11:0:operator:/root:/sbin/nologin
/etc/group:root:x:0:
```

```
$ ps -ef | grep bash
root       706     1  0 09:38 ?        00:00:00 /bin/bash /usr/sbin/ksmtuned
user1     2761  2609  0 09:39 ?        00:00:00 /usr/bin/ssh-agent /bin/sh -cexec
-l /bin/bash -c "env GNOME_SHELL_SESSION_MODE=classic gnome-session --session
gnome-classic"
user1     3298  3285  0 09:39 pts/0    00:00:00 -bash
user1     3395  3298  0 09:42 pts/0    00:00:00 grep --color=auto bash
```

Les options de «grep»

-i : pour ignorer la casse.

Exemple :

```
$ grep -i 're' /etc/passwd
unbound:x:998:996:Unbound DNS resolver:/etc/unbound:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
```

-v : pour récupérer les lignes qui ne contiennent pas la chaîne de caractères.

Exemple :

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
```

```
$ grep -v 'nologin' /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
user1:x:1000:1003:user1:/home/user1:/bin/bash
user2:x:1001:1004::/home/user2:/bin/bash
```

-l : pour afficher que la liste des fichiers qui contiennent la chaîne de caractères.

Exemple :

```
$ grep -l 'mount' /etc/init.d/*
/etc/init.d/functions
/etc/init.d/network
```

-c : indique le nombre de lignes trouvées.

Exemple :

```
$ grep -c 'root' /etc/passwd
2
```

-q : mode silencieux, n'écrit rien sur la sortie standard.
Pratique pour la programmation (scripts shells).

Exemple :

```
$ grep -q 'root' /etc/passwd
$
```

Les expressions régulières et les commandes grep

Expressions régulières

- Liste de caractères [abc]
- Intervalle de caractères [a-z] [A-Z] [a-zA-Z] [0-9]
- Exclusion de liste [^abc] [^a-z]
- Un caractère quelconque .
- 0 à n fois le caractère précédent *
- marqueur de début de ligne ^
- marqueur de fin de ligne $
- marqueur de début de mot \<
- marqueur de fin de mot \>

Expressions régulières

Introduction

Les expressions régulières sont les caractères spéciaux pour les chaînes de caractères.

Liste de caractères [abc]

Représente un caractère parmi la liste de ceux spécifiés entre les crochets «**[**» et «**]**» .

Exemple :

```
$ grep      'r[aio]'      /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
libstoragemgmt:x:995:994:daemon account for
libstoragemgmt:/var/run/lsm:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
radvd:x:75:75:radvd user:/:/sbin/nologin
chrony:x:994:993::/var/lib/chrony:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
```

Dans l'exemple ci-dessus, nous recherchons les lignes contenant un caractère «**r**» suivi d'un caractère a ou i ou o.

Intervalle de caractères [a-z]

L'utilisation du «-» permet de définir une liste de caractères.

[a-z] : un caractère en minuscule, de «**a**» à «**z**».

Exemple :

```
$ grep '[a-z]' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

[A-Z] : un caractère en majuscule, de «**A**» à «**Z**».

Exemple :

```
$ grep '[A-Z]' /etc/passwd
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
```

[a-zA-Z]: un caractère en minuscule ou majuscule.

Exemple :

```
$ grep '[a-zA-Z]' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

[0-9] : un chiffre de **0** à **9**.

Exemple :

```
$ grep '[0-9]' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

[^abc] : un caractère exclu de la liste.

Exemple :

```
$ grep '[^abc]' /etc/passwd
```

[^a-z] : un caractère exclu de l'intervalle de caractères.

Exemple :

```
$ grep '[^a-z]' /etc/passwd
```

Un caractère quelconque .

La caractère «**.**» représente un caractère quelconque.

Exemple :

```
$ grep 'r..t' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
```

L'exemple ci-dessus filtre les lignes contenant la chaîne de caractères «**r**», suivi de 2 caractères quelconques, suivi de «**t**».

**La closure *

Le caractère «*****» est la closure, il représente 0 à n fois le caractère précédent.

Exemple :

```
$ grep 'ro*t' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
```

L'exemple ci-dessus filtre les lignes contenant la chaîne de caractères «**r**», suivi de 0 à n fois le caractère «**o**», suivi de «**t**». C'est à dire rt, rot, root, roooooot, ...

Le marqueur de début de ligne ^

Le caractère «**^**» est le marqueur de début de ligne, défini «les lignes qui commencent par».

Exemple :

```
$ grep '^u' /etc/passwd
unbound:x:998:996:Unbound DNS resolver:/etc/unbound:/sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
user1:x:1000:1003:user1:/home/user1:/bin/bash
user2:x:1001:1004::/home/user2:/bin/bash
user3:x:1002:1005::/home/user3:/bin/bash
```

Le marqueur de fin de ligne $

Le caractère «**$**» est le marqueur de fin de ligne, défini «les lignes qui finissent par».

Exemple :

```
$ grep 'nologin$' /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

Le marqueur de début de mot \<

L'ensemble de caractères «**\<**» représentent les lignes dans lesquelles des mots commencent par la chaîne de caractères.

Exemple :

```
$ grep '\<bin' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
user1:x:1000:1003:user1:/home/user1:/bin/bash
user2:x:1001:1004::/home/user2:/bin/bash
```

Le marqueur de fin de mot \>

L'ensemble de caractères «**\>**» représentent les lignes dans lesquelles des mots se finissent par la chaîne de caractères.

Exemple :

```
$ grep 'mu\>' /etc/passwd
qemu:x:107:107:qemu user:/:/sbin/nologin
```

Les expressions régulières et les commandes grep

Les commandes «fgrep» et «egrep»

- Commande «fgrep»

```
$ fgrep    '.gif'       fic
```

- Commande «egrep»

```
$ egrep    'var|user'   /etc/passwd
```

```
$ grep     'var'    /etc/passwd   |   grep   'user'
```

Les commandes «fgrep» et «egrep»

La commande «fgrep»

Cette commande n'interprète pas les caractères spéciaux.

Exemple :

```
$ grep      '.gif'      fic
```

Ci-dessus avec grep, le «**.**» représente le caractère spécial définissant un caractère quelconque.

```
$ fgrep     '.gif'      fic
```

Ci-dessus avec fgrep, le «**.**» est inhibé et représente donc le simple caractère «**.**». La commande affichera toutes les lignes du fichier «**fic**» qui contiennent la chaîne de caractères «**.gif**».

La commande «egrep»

Cette commande est plus puissante que la commande «**grep**» mais elle est moins utilisée. La commande «**egrep**» supporte des fonctionnalités et caractères spéciaux supplémentaires.

Par exemple, le caractère «**|**» représente un «**OU logique**» pour deux chaînes de caractères.

Exemple :

```
$ egrep      'var|user'      /etc/passwd
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

Dans l'exemple ci-dessus, la caractère «**|**» (pipe) signifie un «**OU logique**». Cette commande affichera toute les lignes qui contiennent «**var**» OU «**user**».

Remarque complémentaire

Pour réaliser un «**ET logique**», il faut simplement utiliser le pipe.

Exemple :

```
$ grep    'var'    /etc/passwd    |    grep    'user'
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

Notes

Les commandes sed et awk

Dans ce chapitre nous allons nous traiter les syntaxes des commandes sed et awk.

Les commandes «sed» et «awk»

- La commande «sed» - Les bases
- La commande «sed» - Les options d, p et w
- La commande «sed» - L'insertion
- La commande «sed» - Les compléments
- La commande «sed» - Quelques cas
- La commande «awk» - Les bases
- La commande «awk» - Les filtres
- La commande «awk» - Les compléments

Les commandes «sed» et «awk»

La commande «sed» - Les bases

```
sed   'action'   fichier
sed   's/ancien/nouveau/g'   fichier

sed   'zone_de_traitement   action'   fichier
      '5                    action'
      '5,10                 action'          1,$
      '/filtre/             action'
      '/debut/,/fin/        action'
```

La commande «sed» - Les bases

Présentation

La commande «**sed**» réalise les opérations d'édition de texte de manière non interactive :
Substitution de chaînes de caractères, suppression de lignes, insertion de texte…

Par défaut, «**sed**» n'est pas destructrice du fichier d'origine car le résultat de cette commande est uniquement envoyé vers la sortie standard.

Syntaxe : $ sed 'action' fichier

La substitution

action = s/ancienne_chaîne/nouvelle_chaîne/

Substitue l'ancienne chaîne de caractères par la nouvelle chaîne de caractères. Cette modification est réalisée uniquement pour la première occurrence sur la ligne.

action = s/ancienne_chaîne/nouvelle_chaîne/g

Comme précédemment mais pour toutes les occurrences sur la ligne.

Exemple :

```
$ cat       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
Oranges 40 dollars Mures 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande


$ sed       's/dollar/euro/'       fichier
Votre premiere commande :
Pommes 10 euros Cerises 20 dollars Fraises 30 dollars
Oranges 40 euros Mures 50 dollars Framboises 60 dollars
Mangues 40 euros Ananas 50 dollars Bananes 60 dollars
Prunes 40 euros Oranges 50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 euros Mirabelles 50 dollars Peches 60 dollars
Mures 40 euros Fraises 50 dollars Oranges 60 dollars
Groseilles 40 euros Pommes 50 dollars Raisins 60 dollars
Oranges 40 euros Mandarines 50 dollars Clementines 60 dollars
Framboises 40 euros Raisins 50 dollars Mures 60 dollars
Fin de la commande


$ sed       's/dollars/euros/g'       fichier
Votre premiere commande :
Pommes 10 euros Cerises 20 euros Fraises 30 euros
Oranges 40 euros Mures 50 euros Framboises 60 euros
Mangues 40 euros Ananas 50 euros Bananes 60 euros
Prunes 40 euros Oranges 50 euros Poires 60 euros
Votre deuxième commande :
Fraises 40 euros Mirabelles 50 euros Peches 60 euros
Mures 40 euros Fraises 50 euros Oranges 60 euros
Groseilles 40 euros Pommes 50 euros Raisins 60 euros
Oranges 40 euros Mandarines 50 euros Clementines 60 euros
Framboises 40 euros Raisins 50 euros Mures 60 euros
Fin de la commande
```

Zone de traitement

Il est possible d'insérer une zone de traitement avant d'effectuer l'action. Cela permet de limiter les lignes du fichier qui subiront l'action.

Syntaxe : $ sed 'zone_de_traitement action' fichier

numéro_de_ligne

L'action est appliquée que sur une seule ligne du fichier.

Exemple :

```
$ sed       '3s/Oranges/KIWIS/'       fichier
1     Votre premiere commande :
2     Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
3     KIWIS 40 dollars Mures 50 dollars Framboises 60 dollars
4     Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
5     Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
6     Votre deuxième commande :
7     Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
8     Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
9     Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
10    Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
11    Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
12    Fin de la commande
```

Dans l'exemple ci-dessus, nous avons attribué le numéro «**3**» à la zone de traitement puis une action de substitution.
Ce qui entraîne la substitution du terme «**Oranges**» par «**KIWIS**» uniquement sur la 3ème ligne.

numéro_de_ligne_de_début,numéro_de_ligne_de_fin

L'action est appliquée sur un ensemble de ligne, depuis le numéro de ligne de début jusqu'au numéro de ligne de fin.

Exemple :

```
$ sed       '4,8 s/Oranges/KIWIS/'       fichier
1     Votre premiere commande :
2     Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
3     Oranges 40 dollars Mures 50 dollars Framboises 60 dollars
4     Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
5     Prunes 40 dollars KIWIS 50 dollars Poires 60 dollars
6     Votre deuxième commande :
7     Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
8     Mures 40 dollars Fraises 50 dollars KIWIS 60 dollars
9     Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
10    Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
11    Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
12    Fin de la commande
```

Dans l'exemple ci-dessus, nous avons attribué les lignes de «**4**» à «**8**» à la zone de traitement puis une action de substitution.
Ce qui entraîne la substitution du terme «**Oranges**» par «**KIWIS**» à partir de la ligne «**4**» jusqu'à la ligne «**8**».

/chaîne_de_caractères/

L'action est appliquée sur la ligne contenant la chaîne de caractères précisée (un filtre).

Exemple :

```
$ sed       '/Oranges/s/Mures/KIWIS/'       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
Oranges 40 dollars KIWIS 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
KIWIS 40 dollars Fraises 50 dollars Oranges 60 dollars
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

Dans l'exemple ci-dessus, nous avons attribué le filtre «**Oranges**» à la zone de traitement puis une action de substitution du terme «**Mures**» par «**KIWIS**».
Ce qui entraîne la substitution du terme «**Mures**» par «**KIWIS**» uniquement sur les lignes qui contiennent le mot «**Oranges**».

/chaîne_de_début/,/chaîne_de_fin/

L'action est appliquée sur une zone délimitée par deux chaînes de caractères, de la ligne contenant la chaîne de début jusqu'à la ligne contenant la chaîne de fin.

Exemple :

```
$ sed        '/Pommes/,/Poires/s/0/5/g'       fichier
Votre premiere commande :
Pommes 15 dollars Cerises 25 dollars Fraises 35 dollars
Oranges 45 dollars Mures 55 dollars Framboises 65 dollars
Mangues 45 dollars Ananas 55 dollars Bananes 65 dollars
Prunes 45 dollars Oranges 55 dollars Poires 65 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
Groseilles 45 dollars Pommes 55 dollars Raisins 65 dollars
Oranges 45 dollars Mandarines 55 dollars Clementines 65 dollars
Framboises 45 dollars Raisins 55 dollars Mures 65 dollars
Fin de la commande
```

Dans l'exemple ci-dessus, nous avons attribué un filtre de début «**Pommes**» et un filtre de fin «**Poires**» à la zone de traitement, puis une action de substitution du caractère «**0**» par «**5**».
Ce qui entraîne la substitution des «**0**» par des «**5**» dans l'intervalle des lignes qui contiennent le terme «**Pommes**» à «**Poires**».

Les commandes «sed» et «awk»

La commande «sed» - Les options d, p et w

d suppression de lignes

'5d'

'5,10d'

'/filtre/d'

'/début/,/fin/d'

sed -n '/Pommes/p' fichier

sed -n '/Pommes/w newfic' fichier

La commande «sed» - Les options d, p et w

La suppression «d»

Le caractère «**d**» (delete) permet de supprimer une ligne.

Syntaxe : $ sed 'filtre d' fichier

Exemples :

```
$ sed '5d' fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
Oranges 40 dollars Mures 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Votre deuxième commande :
Etc…
```

Entraîne la suppression de la ligne «**5**» du fichier.

```
$ sed '2,11d' fichier
Votre premiere commande :
Fin de la commande
```

Entraîne la suppression des lignes «**2**» à «**11**» du fichier.

```
$ sed       '/Pommes/d'       fichier
Votre premiere commande :
Oranges 40 dollars Mures 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

Entraîne la suppression des lignes contenant le terme «**Pommes**» du fichier.

```
$ sed       '/Pommes/,/Poires/d'       fichier
Votre premiere commande :
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
```

Entraîne la suppression des lignes contenant le terme «**Pommes**» jusqu'à la ligne contenant le terme «**Poires**».

Le print «p»

Le caractère «**p**» permet d'afficher les lignes qui contiennent le terme précisé dans le filtre.
L'option «**-n**» permet de ne pas afficher le reste du fichier.

Syntaxe : $ sed -n '/filtre/p' fichier

Exemple :

```
$ sed       -n       '/Pommes/p'       fichier
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
```

Affiche uniquement les deux lignes du fichier contenant le terme «**Pommes**».

Le write «w»

Le caractère «**w**» permet de créer un nouveau fichier contenant le résultat de la commande.

Syntaxe : $ sed -n '/filtre/w nouveau_fichier' fichier

Exemple :

```
$ sed       -n       '/Pommes/w    newfic'       fichier
```

Un fichier nommé «**newfic**» à été créé avec toutes les lignes contenant «**Pommes**».

Les commandes «sed» et «awk»

La commande «sed» - L'insertion

```
sed          '/filtre/i\
> nouveau_texte'        fichier

sed          '/filtre/a\
> nouveau_texte'        fichier

sed          '/filtre/c\
> nouveau_texte'        fichier

sed          '/filtre/r  autre_fichier'        fichier
```

La commande «sed» - L'insertion

Cette commande nous permet d'insérer du texte avant ou après une ligne, ou insérer le contenu d'un autre fichier.
Il peut aussi remplacer le texte initial par un nouveau.

Utilisation du code «i»

La caractère d'insertion «**i**» (insert) permet d'insérer du texte avant la ligne définie par le filtre.
Dans l'exemple ci-dessous, nous avons inséré du texte avant chaque ligne contenant «**Oranges**».

Exemples :

```
$ sed       '/Oranges/i\
> un nouveau texte'       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
un nouveau texte
Oranges 40 dollars Mures 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
un nouveau texte
Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
un nouveau texte
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
un nouveau texte
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

Utilisation du code «a»

La caractère d'insertion «**a**» (append) permet d'ajouter du texte après la ligne définie par le filtre.
Dans l'exemple ci-dessous, nous avons inséré du texte après chaque ligne contenant «**Oranges**».

Exemples :

```
$ sed       '/Oranges/a\
un nouveau texte'       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
Oranges 40 dollars Mures 50 dollars Framboises 60 dollars
un nouveau texte
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
un nouveau texte
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
un nouveau texte
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
un nouveau texte
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

Utilisation du code «c»

La caractère d'insertion «**c**» (change) permet de remplacer la ligne définie par le filtre.
Dans l'exemple ci-dessous, les lignes contenant le terme «**Oranges**» ont été remplacé par «**un nouveau texte**».

Exemples :

```
$ sed       '/Oranges/c\
un nouveau texte'       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
un nouveau texte
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
un nouveau texte
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
un nouveau texte
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
un nouveau texte
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

Utilisation du code «r»

La caractère d'insertion «**r**» permet de rajouter le contenu d'un fichier après la ligne définie par le filtre.
Dans l'exemple ci-dessous, nous avons rajouté le contenu du fichier «**autrefichier**» à la suite des lignes contenant le terme «**Pommes**».

Exemples :

```
$ cat       autrefichier
et voici un nouveau fichier
un texte quelconque
sur 3 lignes

$ sed       '/Pommes/r      autrefichier'       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
et voici un nouveau fichier
un texte quelconque
sur 3 lignes
Oranges 40 dollars Mures 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
et voici un nouveau fichier
un texte quelconque
sur 3 lignes
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

Les commandes «sed» et «awk»

La commande «sed» - Les compléments

-e

```
sed  –e 's/Pommes/Bananes/g' –e 's/Poires/Bananes/' –e '/Mures/d'  fic
```

-f

```
cat       fic_actions
s/Pommes/Bananes/g
s/Poires/Bananes/
/Mures/d

sed       –f  fic_actions       fichier
```

Remplacement du caractère spécial /

La commande «sed» - Les compléments

Option «-e»

L'option «**-e**» permet de cumuler plusieurs actions sur une même ligne de commande.

Exemple :

```
$ sed –e 's/Pommes/Bananes/g' –e 's/Poires/Bananes/' –e '/Mures/d'
fichier
Votre premiere commande :
Bananes 10 dollars Cerises 20 dollars Fraises 30 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Bananes 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Groseilles 40 dollars Bananes 50 dollars Raisins 60 dollars
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
Fin de la commande
```

Dans l'exemple ci-dessus, nous avons remplacé les «**Pommes**» et les «**Poires**» par des «**Bananes**», et nous avons supprimé les lignes contenant «**Mures**» du fichier.

Option «-f»

L'argument suivant l'option «**-f**» indique un fichier où se trouvent les actions qui doivent être appliquées à la commande sed.

Syntaxe : $ sed -f fichier_des_actions fichier

Exemple :

```
$ cat       fic_actions
s/Pommes/Bananes/g
s/Poires/Bananes/
/Mures/d

$ sed       -f  fic_actions       fichier
Votre premiere commande :
Bananes 10 dollars Cerises 20 dollars Fraises 30 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Bananes 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Groseilles 40 dollars Bananes 50 dollars Raisins 60 dollars
Oranges 40 dollars Mandarines 50 dollars Clementines 60 dollars
Fin de la commande
```

Remplacement du caractère spécial «/»

Pour remplacer le caractère spécial «**/**» que nous avons communément l'habitude d'utiliser, nous pouvons en réalité utiliser n'importe qu'elle autre caractère. Ceci est tout particulièrement utile pour éviter un conflit dans la ligne de commande.

Exemple :

```
$ cat      fichier2
/home/user1/rep1
/home/user1/repertoire
/home/user222/fic
/home/user1/fichier

$ sed       's?/?|?g'       fichier2
|home|user1|rep1
|home|user1|repertoire
|home|user222|fic
|home|user1|fichier
```

Dans l'exemple ci-dessus, nous avons remplacer le caractère spécial «**/**» par le «**?**», ce qui nous a permis d'obtenir une syntaxe simple pour la substitution de «**/**» par «**|**».

Les commandes «sed» et «awk»

La commande «sed» - Quelques cas

```
sed    's/Oranges//'    fichier

sed    's/^Oranges/Mures/'    fichier

sed    's/\//|/g'    fichier

find   /   -name vi   2>   /dev/null    |    sed    's/\//|/g'

sed    '/^[   \t]*$/d'    fichier
```

La commande «sed» - Quelques cas

sed 's/Oranges//' fichier

```
$ sed       's/Oranges//'       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
 40 dollars Mures 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars  50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars  60 dollars
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
 40 dollars Mandarines 50 dollars Clementines 60 dollars
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

On constate que la section de «**la nouvelle chaîne**» est vide ou tout simplement inexistante.
L'instruction «**sed**» permet dans cet exemple de supprimer la chaîne de caractères «**Oranges**».

sed 's/^Oranges/Mures/' fichier

```
$ sed       's/^Oranges/Mures/'       fichier
Votre premiere commande :
Pommes 10 dollars Cerises 20 dollars Fraises 30 dollars
Mures 40 dollars Mures 50 dollars Framboises 60 dollars
Mangues 40 dollars Ananas 50 dollars Bananes 60 dollars
Prunes 40 dollars Oranges 50 dollars Poires 60 dollars
Votre deuxième commande :
Fraises 40 dollars Mirabelles 50 dollars Peches 60 dollars
Mures 40 dollars Fraises 50 dollars Oranges 60 dollars
Groseilles 40 dollars Pommes 50 dollars Raisins 60 dollars
Mures 40 dollars Mandarines 50 dollars Clementines 60 dollars
Framboises 40 dollars Raisins 50 dollars Mures 60 dollars
Fin de la commande
```

Nous pouvons utiliser les expressions régulières. L'exemple ci-dessus substitue le mot «**Oranges**» en début de ligne par «**Mures**».

sed 's/\//|/g' fichier

```
$ cat       fichier2
/home/user1/rep1
/home/user1/repertoire
/home/usre222/fic
/home/user1/fichier

$ sed       's/\//|/g'       fichier2
|home|user1|rep1
|home|user1|repertoire
|home|usre222|fic
|home|user1|fichier
```

comme précédemment, le «****» est le caractère spécial inhibant le caractère suivant. Ainsi, les caractères «**/**» du fichier sont remplacé par «**|**».

commande | sed 'action'

```
$ find / -name vi 2> /dev/null | sed 's/\//|/g'
|usr|bin|vi
|usr|share|locale|vi
|usr|share|help|vi
|usr|share|vim|vim74|lang|vi
|usr|share|espeak-data|voices|asia|vi
```

L'utilisation de la commande «**sed**» est pratique avec le pipe pour modifier l'affichage du résultat.

sed '/^[\t]*$/d' fichier

```
$ cat       fichier3
Bonjour,

ceci est nouveau

puis

suite

fin

$ cat       -evt       fichier3
Bonjour,$
$
ceci est nouveau$
  $
puis$
  ^I^I^I^I$
suite$
^I^I$
fin$

$ sed       '/^[ \t]*$/d'       fichier3
Bonjour,
ceci est nouveau
puis
suite
fin
```

Cette commande supprime les lignes blanches. C'est à dire les lignes vides ou contenant une combinaison du caractères espaces et/ou de tabulations.

[\t] : liste de caractères, ici le caractère espace ou le caractère de tabulation (\t).

[\t]* : * est la «**closure**» pour définir 0 à n fois le caractère précédent. Ici l'espace ou la tabulation. 0 fois représente une ligne vide, n fois représente 0 à n espaces éventuellement combinés avec 0 à n tabulations.

^xxx$ sont respectivement les caractères spéciaux de marqueurs de début de ligne et de fin de ligne. Ainsi la ligne est exclusivement constitué de descriptif 'xxx'.
Dans notre exemple cela correspond aux lignes dites blanches.

Les commandes «sed» et «awk»

La commande «awk» - Les bases

awk '{print $1}' /etc/hosts

awk -F: '{print $3, $1}' /etc/passwd

awk -F: '{print "Uid = "$3" Login = " $1}' /etc/passwd

La commande «awk» - Les bases

La commande awk (nawk, gawk) permet de manipuler le contenu d'un fichier en exploitant les champs. Cette instruction peut avoir une syntaxe très complexe car elle intègre toutes les fonctionnalités des scripts.

Syntaxe : awk [-options] 'actions' fichier

Les bases

```
$ awk '{print $1}' /etc/hosts
127.0.0.1
::1
```

La section entre **{ }** définit la liste des actions à réaliser sur chaque ligne du fichier.
La référence à un champ est indiquée par le caractère **$** suivi du numéro de champ, les caractères 'espace' et 'tabulation' sont les caractères séparateurs de champs par défaut.
Ainsi l'exemple ci-dessus affiche le premier champ du fichier /etc/hosts.

```
$ awk       -F:       '{print $3, $1}'       /etc/passwd
0 root
1 bin
2 daemon
3 adm
4 lp
5 sync
Etc..
```

L'option **-F** redéfinie le caractère séparateur de champs (ici le «**:**»).
L'exemple affiche le champ 3 puis le champ 1 séparés par un caractère «**espace**». Ce dernier est présent à cause du caractère spécial «**,**».

```
$ awk    -F:     '{print "Uid = "$3" Login = " $1}'     /etc/passwd
Uid = 0 Login = root
Uid = 1 Login = bin
Uid = 2 Login = daemon
Uid = 3 Login = adm
Uid = 4 Login = lp
Uid = 5 Login = sync
Etc..
```

Il est possible d'agrémenter l'affichage avec du texte, il suffit de l'écrire entre guillemets.

Les commandes «sed» et «awk»

La commande «awk» - Les filtres

awk -F: '/root/{print "Login "$1" Ligne : "$0}' /etc/passwd

awk -F: '/^root/{print "Login "$1" Ligne : "$0}' /etc/passwd

awk -F: '($3 > 99){print "Utilisateur Ligne : "$0}' /etc/passwd

ou

awk -F: '{if ($3 > 99) print "Utilisateur Ligne : "$0}' /etc/passwd

La commande «awk» - Les filtres

Filtres

Syntaxe : awk '**/filtre/**{action}' fichier

```
$ awk -F: '/root/{print "Login "$1" Ligne : "$0}' /etc/passwd
Login root Ligne : root:x:0:0:root:/root:/bin/bash
Login operator Ligne : operator:x:11:0:operator:/root:/sbin/nologin
```

Le filtre permet de sélectionner les lignes qui vont subir l'action. Il est utilisé le «**/**» pour définir une chaîne de caractères.
L'exemple ci-dessus applique l'action «**print**» pour toutes les lignes qui contiennent la chaîne de caractères «**root**».
L'expression «**$0**» représente la ligne complète.

```
$ awk -F: '/^root/{print "Login "$1" Ligne : "$0}' /etc/passwd
Login root Ligne : root:x:0:0:root:/root:/bin/bash
```

L'utilisation des expressions régulières est autorisée.

Syntaxe : awk '**(test)** {action}' fichier

ou

awk '{ **if(test) action**}' fichier

```
$ awk  -F: '($3 > 99){print "Utilisateur Ligne : "$0}'     /etc/passwd

ou

$ awk  -F: '{if ($3 > 99) print "Utilisateur Ligne : "$0}' /etc/passwd

      Utilisateur Ligne : polkitd:x:999:998:User for polkitd:/:/sbin/nologin
      Utilisateur Ligne : abrt:x:173:173::/etc/abrt:/sbin/nologin
      Utilisateur Ligne : colord:x:997:995:User for colord:/var/lib/colord:/sbin/nologin
      Utilisateur Ligne : usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
      Utilisateur Ligne : qemu:x:107:107:qemu user:/:/sbin/nologin
      Utilisateur Ligne : rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
      Utilisateur Ligne : chrony:x:994:993::/var/lib/chrony:/sbin/nologin
      Utilisateur Ligne : user1:x:1000:1003:user1:/home/user1:/bin/bash
      Utilisateur Ligne : user2:x:1001:1004::/home/user2:/bin/bash
      Utilisateur Ligne : user3:x:1002:1005::/home/user3:/bin/bash
```

Les deux syntaxes sont équivalentes, elles permettent de sélectionner les lignes sur lesquelles vont s'appliquer l'action en fonction d'un test.
L'exemple ci-dessus sélectionne les lignes dont le contenu du champ 3 est strictement supérieur à 99.

Les opérateurs possibles sont :

<, <=, >, >=,
== (équivalent à), != (différent), ~ (qui contient), !~ (qui ne contient pas)

Les commandes «sed» et «awk»

La commande «awk» - Les compléments

L'option -f fichier_des_actions

Les blocs BEGIN { … } et END { … }

Variables internes

Opérations arithmétiques

Fonctions internes

La commande «awk» - Les compléments

L'option -f

Cette option permet de simplifier la syntaxe de la commande awk, ainsi que de mémoriser au sein d'un fichier une liste d'actions. C'est tout particulièrement utile pour une utilisation répétée, ou lorsqu'il y a de nombreuses actions à réaliser.

```
$ cat    actions_awk
/root/{print "Login "$3" Ligne : "$0}

/^root/{print "La ligne commence par root : "$0}

($3 > 99){print "Utilisateur Uid="$3" et Login="$1}

{print "Uid="$3" et Login="$1}

$ awk    -F:   -f   actions_awk   /etc/passwd
Login 0 Ligne : root:x:0:0:root:/root:/bin/bash
La ligne commence par root : root:x:0:0:root:/root:/bin/bash
Uid=0 et Login=root
Uid=1 et Login=bin
Uid=2 et Login=daemon
Login 11 Ligne : operator:x:11:0:operator:/root:/sbin/nologin
Uid=11 et Login=operator
Uid=12 et Login=games
Utilisateur Uid=999 et Login=polkitd
Uid=999 et Login=polkitd
Utilisateur Uid=173 et Login=abrt
Uid=173 et Login=abrt
Utilisateur Uid=998 et Login=unbound
Uid=998 et Login=unbound
Etc...
```

Les blocs BEGIN et END

Syntaxe :

```
BEGIN {
   action-1
   action-2
   …
}

END {
   action-1
   …
}
```

Les actions définies au sein du bloc BEGIN sont exécutées avant le traitement des lignes du fichier.

Les actions définies au sein du bloc END sont exécutées après le traitement des lignes du fichier.

```
$ cat    actions_awk_2
BEGIN {
  FS=":"
  print "Debut du traitement"
}

{print "Uid="$3" et Login="$1" et la ligne : "$0}

END {
  print "Fin du traitement"
}

$ awk    -f    actions_awk_2    /etc/passwd
Debut du traitement
Uid=0 et Login=root et la ligne : root:x:0:0:root:/root:/bin/bash
Uid=1 et Login=bin et la ligne : bin:x:1:1:bin:/bin:/sbin/nologin
Uid=2 et Login=daemon et la ligne : daemon:x:2:2:daemon:/sbin:/sbin/nologin
Uid=3 et Login=adm et la ligne : adm:x:3:4:adm:/var/adm:/sbin/nologin
Uid=4 et Login=lp et la ligne : lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
Uid=5 et Login=sync et la ligne : sync:x:5:0:sync:/sbin:/bin/sync
Uid=7 et Login=halt et la ligne : halt:x:7:0:halt:/sbin:/sbin/halt
Uid=8 et Login=mail et la ligne : mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
Uid=12 et Login=games et la ligne : games:x:12:100:games:/usr/games:/sbin/nologin
Uid=14 et Login=ftp et la ligne : ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
Uid=99 et Login=nobody et la ligne : nobody:x:99:99:Nobody:/:/sbin/nologin
… etc …
Uid=72 et Login=tcpdump et la ligne : tcpdump:x:72:72::/:/sbin/nologin
Uid=1001 et Login=user2 et la ligne : user2:x:1001:1004::/home/user2:/bin/bash
Uid=1002 et Login=user3 et la ligne : user3:x:1002:1005::/home/user3:/bin/bash
Fin du traitement
```

Les variables internes

Il existe des variables spécifiques à la commande awk, entre autres :

FS	:	caractère séparateur de champs en entrée.
OFS	:	caractère séparateur de champs en sortie.
RS	:	caractère séparateur d'enregistrement en entrée.
ORS	:	caractère séparateur d'enregistrement en sortie.
NF	:	nombre de champs sur l'enregistrement courant.
NR	:	nombre d'enregistrement lus.

```
$ cat    actions_awk_3
BEGIN {
  FS=":"
  OFS="?"
  ORS="|"
}
{print $3,$1}

$ awk    -f    actions_awk_3    /etc/passwd
0?root|1?bin|2?daemon|3?adm|4?lp|5?sync|6?shutdown|7?halt|8?mail|11?operator|12?
games|14?ftp|99?nobody|81?dbus|999?polkitd|173?abrt|998?unbound|997?colord|113?
usbmuxd|38?ntp|70?avahi|170?avahi-autoipd|996?saslauth|107?qemu|995?
libstoragemgmt|32?rpc|29?rpcuser|65534?nfsnobody|172?rtkit|75?radvd|994?chrony|
171?pulse|42?gdm|993?gnome-initial-setup|89?postfix|74?sshd|72?tcpdump|1000?user1|
1001?user2|1002?user3|$
```

```
$ cat    actions_awk_3_bis
BEGIN {
  FS=":"
}
{print "La ligne " NR " login="$1" avec " NF " champs"}

$ awk    -f    actions_awk_3_bis    /etc/passwd
La ligne 1 login=root avec 7 champs
La ligne 2 login=bin avec 7 champs
La ligne 3 login=daemon avec 7 champs
La ligne 4 login=adm avec 7 champs
La ligne 5 login=lp avec 7 champs
La ligne 6 login=sync avec 7 champs
La ligne 7 login=shutdown avec 7 champs
La ligne 8 login=halt avec 7 champs
La ligne 9 login=mail avec 7 champs
```

Les opérations arithmétiques

```
$ cat     actions_awk_4
BEGIN {
  FS=":"
  num=0
}
{ num = num + $3}
END {
  print "Somme = "num
}

$ awk     -f     actions_awk_4     /etc/passwd
Somme = 77189
```

Les fonctions internes

Il existe des fonctions spécifiques à la commande awk, telles que :

tolower(chaîne)	:	converti en minuscule.
toupper(chaîne)	:	converti en majuscule.
length(chaîne)	:	nombre de caractères de la chaîne.
index(chaîne,caractère):		indique la position du caractère au sein de la chaîne.
substr(chaîne,position):		récupère une sous-chaîne de la position à la fin de la chaîne.
sqrt(variable)	:	récupère la racine carré de la variable.
int(variable)	:	récupère la partie entière de la variable.
rand()	:	fourni une valeur aléatoire comprise entre 0 et 1.

```
$ cat     actions_awk_5
BEGIN {
  FS=":"
  num=0
}
{ num = num + $3
  if( toupper($1) ~ "USER") print "Utilisateur : "$1,toupper($1)}
END {
  print "Resultat sqrt=" sqrt(num) " partie entière du résultat=" int(sqrt(num))
}

$ awk     -f     actions_awk_5     /etc/passwd
Utilisateur : rpcuser RPCUSER
Utilisateur : user1 USER1
Utilisateur : user2 USER2
Utilisateur : user3 USER3
Resultat sqrt=277.829 partie entière du résultat=277
```

Notes

Le réseau

Dans ce chapitre nous allons nous apprendre l'utilisation des commandes réseaux, découvrir des fichiers de configurations et des mécanismes réseaux.

Le réseau

- Les commandes
- Les fichiers de configurations
- Les commandes SSH
- L'utilisation des clefs SSH
- Le mail
- Les commandes mesg, write et wall
- Les serveurs DNS, DHCP, NFS et LDAP

Le réseau

Les commandes

ifconfig	-a			
ping	-c			
netstat	-a	-an	-r	-nr
arp	-n			
hostname	domainname			
nslookup				
route	traceroute			

Les commandes

La commande «ifconfig»

La commande «**ifconfig**» affiche la configuration des interfaces actives de votre système.

```
$ ifconfig
eth0   Link encap:Ethernet  HWaddr 08:00:27:A2:D4:ED
       inet adr:192.168.1.6  Bcast:192.168.1.255  Masque:255.255.255.0
       adr inet6: fe80::a00:27ff:fea2:d4ed/64 Scope:Lien
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:1775 errors:0 dropped:0 overruns:0 frame:0
       TX packets:621 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 lg file transmission:1000
       RX bytes:191549 (187.0 KiB)  TX bytes:105329 (102.8 KiB)

lo     Link encap:Boucle locale
       inet adr:127.0.0.1  Masque:255.0.0.0
       adr inet6: ::1/128 Scope:Hôte
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
       RX packets:134 errors:0 dropped:0 overruns:0 frame:0
       TX packets:134 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 lg file transmission:0
       RX bytes:6740 (6.5 KiB)  TX bytes:6740 (6.5 KiB)
```

Le nommage des périphériques dépend du type de cartes réseaux physiques installées sur la machine. La 1ère carte réseau porte le nom du pilote (eth, qfe, hme, ..) suivi du chiffre 0 (eth0 par exemple). Si plusieurs interfaces identiques sont présentes, elles sont incrémentées (eth1, eth2,...). L'interface lo représente l'adresse de bouclage (loopback).

L'option «-a»

L'option «**-a**» liste toutes les interfaces actives de votre système, ainsi que celles qui sont inactives.

```
$ ifconfig    -a
eth0   Link encap:Ethernet  HWaddr 08:00:27:A2:D4:ED
       inet adr:192.168.1.6  Bcast:192.168.1.255  Masque:255.255.255.0
       adr inet6: fe80::a00:27ff:fea2:d4ed/64 Scope:Lien
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:1798 errors:0 dropped:0 overruns:0 frame:0
       TX packets:628 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 lg file transmission:1000
       RX bytes:193938 (189.3 KiB)  TX bytes:107099 (104.5 KiB)

eth1   Link encap:Ethernet  HWaddr 08:00:27:1E:4E:D5
       BROADCAST MULTICAST  MTU:1500  Metric:1
       RX packets:1013 errors:0 dropped:0 overruns:0 frame:0
       TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 lg file transmission:1000
       RX bytes:114839 (112.1 KiB)  TX bytes:1152 (1.1 KiB)

eth2   Link encap:Ethernet  HWaddr 08:00:27:CA:FC:FE
       BROADCAST MULTICAST  MTU:1500  Metric:1
       RX packets:1583 errors:0 dropped:0 overruns:0 frame:0
       TX packets:469 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 lg file transmission:1000
       RX bytes:172425 (168.3 KiB)  TX bytes:83886 (81.9 KiB)

lo     Link encap:Boucle locale
       inet adr:127.0.0.1  Masque:255.0.0.0
       adr inet6: ::1/128 Scope:Hôte
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
       RX packets:134 errors:0 dropped:0 overruns:0 frame:0
       TX packets:134 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 lg file transmission:0
       RX bytes:6740 (6.5 KiB)  TX bytes:6740 (6.5 KiB)
```

La commande «ping»

Cette commande test si une machine est joignable sur le réseau. Son principe de fonctionnement est d'envoyer des paquets à une adresse IP et d'avoir des informations en retour.

```
$ ping  8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=13.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=32.7 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=12.2 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 12.221/18.153/32.769/7.459 ms
```

Nous avons utilisé la commande «**Ctrl + C**» (**^C**) pour stopper l'envoi de paquets.

L'option «-c»

Cette option nous permet de donner une valeur au nombres de paquets envoyés.

```
$ ping     -c 5       8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=15.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=30.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=14.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=53 time=15.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=53 time=14.2 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 14.293/17.887/30.395/6.269 ms
```

Dans l'exemple ci-dessus, la commande «**ping**» envoie 5 paquets puis s'arrête.

La commande «netstat»

Cette commande affiche des statistiques réseaux.

-a : liste tous les ports (TCP et UDP).

```
$ netstat     -a     |     more
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 *:sunrpc                    *:*                         LISTEN
tcp        0      0 *:ftp                       *:*                         LISTEN
tcp        0      0 *:ssh                       *:*                         LISTEN
tcp        0      0 localhost:ipp               *:*                         LISTEN
... sortie tronquée
```

-n : affichage numérique, sans la résolution de noms.

```
$ netstat     -an     |     more
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 0.0.0.0:111                 0.0.0.0:*                   LISTEN
tcp        0      0 0.0.0.0:21                  0.0.0.0:*                   LISTEN
tcp        0      0 0.0.0.0:22                  0.0.0.0:*                   LISTEN
tcp        0      0 127.0.0.1:631               0.0.0.0:*
```

-r : affiche la table de routage du système.

```
$ netstat     -r     |     more
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic   MSS Fenêtre irtt Iface
192.168.1.0     *               255.255.255.0   U         0 0          0 eth0
link-local      *               255.255.0.0     U         0 0          0 eth0
default         gestionbbox.lan 0.0.0.0         UG        0 0          0 eth0
```

-nr : affiche la table de routage du système sans la résolution de noms.

```
$ netstat     -nr     |     more
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic   MSS Fenêtre irtt Iface
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
192.168.1.254   0.0.0.0         0.0.0.0         UG        0 0          0 et
```

La commande «arp»

Affiche le cache des translations des adresses IP et des adresses ethernet.

```
$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
spheriusform-PC.home     ether   a0:63:91:89:52:94   C                     enp0s3
gestionbbox.lan.home     ether   90:01:3b:cf:e8:59   C                     enp0s3
$
$ arp  -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.6              ether   a0:63:91:89:52:94   C                     enp0s3
192.168.1.254            ether   90:01:3b:cf:e8:59   C                     enp0s3
```

Les commandes «hostname» et «domainname»

Ces commandes affiche le nom de votre machine et le nom du domaine de la machine.

```
$ hostname
poste1

$ domainname
spherius.fr
```

La commande «nslookup»

Cette commande identifie, entre autres, le serveur de service de noms que le système sollicite.

```
$ nslookup  spherius.fr
Server:         89.2.0.1
Address:        89.2.0.1#53

Non-authoritative answer:
Name:   spherius.fr
Address: 213.186.33.18
```

La commande «route»

Cette commande affiche la table de routage.

```
$ route
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
default         gestionbbox.lan 0.0.0.0         UG    1024   0        0 enp0s3
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 enp0s3
$
$ route  -n
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
0.0.0.0         192.168.1.254   0.0.0.0         UG    1024   0        0 enp0s3
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 enp0s3
```

La commande «traceroute»

Cette commande permet, en autres, de visualiser les machines intermédiaires sollicitées.

```
$ traceroute  linuxfoundation.org
traceroute to linuxfoundation.org (140.211.169.4), 30 hops max, 60 byte packets
 1  gestionbbox.lan.home (192.168.1.254)  5.868 ms  9.570 ms  9.313 ms
 2  21.16.2.1 (21.16.2.1)  14.108 ms  17.986 ms  17.471 ms
 3  man1rj-ge-1-1-5.200.numericable.net (195.132.11.241)  29.106 ms  30.740 ms  31.062 ms
 4  ip-49.net-80-236-1.static.numericable.fr (80.23.1.4)  16.816 ms  16.567 ms  16.598 ms
 5  lag101.350.ncc-cbv.net.bbox.fr (212.194.172.189)  20.227 ms  20.287 ms  20.579   ms
 6  * * *
 7  be16.cbr01-ntr.net.bbox.fr (212.194.171.82)  24.133 ms  22.354 ms  23.094 ms
 8  la12.rpt01-ix2.net.bbox.fr (212.194.171.86)  18.265 ms  18.407 ms  20.332 ms
 9  lag-105.ear1.Paris1.Level3.net (212.73.206.181)  20.069 ms  19.639 ms  20.788    ms
10  ae-2-52.edge2.Seattle3.Level3.net (4.69.147.171)  165.847 ms  163.339 ms  170.363 ms
11  ae-2-52.edge2.Seattle3.Level3.net (4.69.147.171)  172.806 ms  172.479 ms  168.751 ms
12  UNIVERSITY.edge2.Seattle3.Level3.net (4.59.23.70)  174.308 ms  173.362 ms  181.871 ms
13  corv-car1-gw.nero.net (207.98.64.39)  181.880 ms  181.799 ms  181.755 ms
14  * * *
15  * * *
16  * * *
17  * * *
18  * corv-car1-gw.nero.net (207.98.64.39)  162.685 ms !X *
```

Le réseau

Les fichiers de configurations

- /etc/sysconfig/network
- /etc/hosts
- /etc/resolv.conf
- /etc/nsswitch.conf
- /etc/services

Les fichiers de configurations

Le fichier /etc/sysconfig/network

Le fichier **/etc/sysconfig/network** contient le nom de la machine (la variable HOSTNAME=<nom_machine>) pour les versions antérieures à CentOS 7.
Depuis CentOS 7, le nom de la machine est stocké dans le fichier **/etc/hostname** (/etc/HOSTNAME pour une distribution Suse).

Versions antérieures à CentOS 7 :

```
# more /etc/sysconfig/network
NETWORKING=yes
HOSTNAME=formateur
```

Depuis CentOS 7 :

```
$ more /etc/hostname
formateur
```

Chaque carte réseau possède un fichier de configuration dans le répertoire **/etc/sysconfig/network-scripts** qui porte le nom ifcfg-<nom_de_la_carte>.

```
$ ls    /etc/sysconfig/network-scripts
ifcfg-enp0s3  ifdown-isdn      ifdown-tunnel  ifup-isdn    ifup-Team
ifcfg-lo      ifdown-post      ifup           ifup-plip    ifup-TeamPort
ifdown        ifdown-ppp       ifup-aliases   ifup-plusb   ifup-tunnel
ifdown-bnep   ifdown-routes    ifup-bnep      ifup-post    ifup-wireless
ifdown-eth    ifdown-sit       ifup-eth       ifup-ppp     init.ipv6-global
ifdown-ippp   ifdown-Team      ifup-ippp      ifup-routes  network-functions
ifdown-ipv6   ifdown-TeamPort  ifup-ipv6      ifup-sit     network-functions-ipv6
```

Le fichier de configuration d'une carte réseau :

```
$ more  /etc/sysconfig/network-scripts/ifcfg-enp0s3
HWADDR="08:00:27:D9:B4:4B"
TYPE="Ethernet"
BOOTPROTO="dhcp"
DEFROUTE="yes"
PEERDNS="yes"
PEERROUTES="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_PEERDNS="yes"
IPV6_PEERROUTES="yes"
IPV6_FAILURE_FATAL="no"
NAME="enp0s3"
UUID="3687b4c0-9a89-48c5-903b-3dad26f2062e"
ONBOOT="yes"
```

Le fichier /etc/hosts

Ce fichier permet d'effectuer la résolution de nom en local. Il est constitué d'au moins 2 champs. Le 1er champ est l'adresse IP de la machine, le second champ est le nom de la machine, les champs suivant sont des noms d'alias pour la machine.

```
$ cat    /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.10.25       mars
192.168.10.30       lune
192.168.10.35       soleil
```

Le fichier /etc/resolv.conf

Ce fichier contient les adresses des serveurs DNS. Chaque serveur DNS est identifié avec l'entrée '**nameserver**'. Ils sont interrogés dans l'ordre chronologique d'apparition dans le fichier. Les mots clefs '**domain**' ou '**search**' permettent de spécifier un suffixe DNS.

```
$ more     /etc/resolv.conf
# Generated by NetworkManager
nameserver 89.2.0.1
nameserver 89.2.0.2
domain spherius.fr
```

Le fichier /etc/nsswitch.conf

Ce fichier définit l'ordre dans lequel les services de noms seront scrutés. Donc, toute la politique de résolution de noms dépend de ce fichier.

```
$ more     /etc/nsswitch.conf
#hosts:     db files nisplus nis dns
hosts:      files dns
passwd:     files
ethers:     files
... etc
```

La ligne 'hosts' indique que pour la résolution de noms des machines, le système vérifie d'abord le fichier **local** (**/etc/hosts**). Si le système ne trouve pas la réponse, il interroge le serveur **DNS**.

Le fichier /etc/services

Ce fichier permet de lister des services internet.

```
$ more     /etc/services
# /etc/services:
# $Id: services,v 1.55 2013/04/14 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2013-04-10
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, ``Assigned Numbers'' (October 1994).  Not all ports
# are included, only the more common ones.
#
# The latest IANA port assignments can be gotten from
#       http://www.iana.org/assignments/port-numbers
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 through 65535
#
# Each line describes one service, and is of the form:
#
# service-name  port/protocol  [aliases ...]   [# comment]

ftp-data        20/tcp
ftp-data        20/udp
# 21 is registered to ftp, but also used by fsp
ftp             21/tcp
ftp             21/udp          fsp fspd
ssh             22/tcp                          # The Secure Shell (SSH) Protocol
ssh             22/udp                          # The Secure Shell (SSH) Protocol
telnet          23/tcp
telnet          23/udp
# 24 - private mail system
lmtp            24/tcp                          # LMTP Mail Delivery
lmtp            24/udp                          # LMTP Mail Delivery
smtp            25/tcp          mail
smtp            25/udp          mail
http            80/tcp          www www-http    # WorldWideWeb HTTP
http            80/udp          www www-http    # HyperText Transfer Protocol
http            80/sctp                         # HyperText Transfer Protocol
```

Le réseau

Les commandes SSH

- ssh

```
$ ssh  -l theo   mars
$ ssh  theo@mars

$ ssh  root@mars  cat /etc/passwd
```

- scp

```
$ scp  [options_ssh]  user@machine:/fichier_source  /fichier
$ scp  [options_ssh]  /fichier  user@machine:/fichier_destination
```

- sftp

```
$ sftp  [options_ssh]  machine
cd chemin       lcd chemin          exit ou quit ou bye
                get fic         mget fic*
                put fic         mput fic*
```

Les commandes SSH

SSH est un mécanisme qui permet une communication entre machines de façon sécurisé, toute la communication étant cryptée.

Le mécanisme SSH repose sur l'existence d'une pair de clefs : la clef publique et la clef privée.
La clef publique est envoyée sur les serveurs auxquels nous voulons nous connecter, la clef privée étant conservée bien précieusement sur la machine sur laquelle nous nous connectons.

Les commandes clientes SSH (Secure Shell) sont des commandes de communications sécurisées, utilisant des clés d'authentification RSA ou DSA : ssh, scp, sftp.

La commande ssh

La commande ssh sert à ce connecter à une machine distante ou à exécuter une séquence de commande sur une machine distante.

$ ssh -l nom_utilisateur machine_distante [séquence_de_commandes]

$ ssh nom_utilisateur@machine_distante [séquence_de_commandes]

```
$ ssh  -l theo   mars
$ ssh  theo@mars

$ ssh  root@mars  cat /etc/passwd
```

Lors d'une première connexion sur un serveur avec SSH, le système demande si on veut ajouter le serveur à la liste des hôtes connus. En répondant « oui » à cette question, nous sauvegardons la clef publique du serveur dans le fichier $HOME/.ssh/known_hosts. Pour se connecter vous devez fournir le mot de passe de l'utilisateur avec lequel vous essayer de vous connecter sur le serveur.

La commande scp

La commande scp sert à copier des fichiers entre deux machines.

Pour récupérer des fichiers d'une machine distante :

```
$ scp [options_ssh] utilisateur@machine:/fichier_source /fichier_destination
```

Pour recopier des fichiers sur une machine distante :

```
$ scp [options_ssh] /fichier_source utilisateur@machine:/fichier_destination
```

La commande sftp

La commande sftp sert à transférer des fichiers entre deux machines.

```
$ sftp [ options_ssh ] machine
```

Cette commande a les sous commandes équivalentes à la commande 'ftp'.

Quelques sous commandes :

cd chemin	:	pour se déplacer sur l'arborescence de la machine distante.
lcd chemin	:	pour se déplacer sur l'arborescence de la machine locale.
get fichier	:	pour récupérer un fichier.
mget fic*	:	pour récupérer plusieurs fichiers.
put fichier	:	pour déposer un fichier.
mput fic*	:	pour déposer plusieurs fichiers.
exit ou quit ou bye	:	pour quitter ftp.

Les fichiers de configurations

Le fichier de configuration du serveur SSH : /etc/ssh/sshd_config

Le fichier de configuration des commandes clientes SSH : /etc/ssh/ssh_config

Le fichier $HOME/.ssh/authorized_keys : il est présent sur le poste serveur SSH. Il contient la liste des clés autorisées pour l'authentification utilisateur.

Le fichier $HOME/.ssh/known_hosts : il est présent sur le poste client SSH. Il contient la liste des clés autorisées pour l'authentification machine.

Le réseau

L'utilisation des clefs SSH

- Création de la clef sur le serveur maître

```
Serveur$  cd   $HOME/.ssh
Serveur$  ssh-keygen  -t  rsa  -f  ma_clef
Serveur$  ls  -l
    ma_clef     ma_clef.pub
```

- Mise à jour des serveurs clients

```
Serveur$ cd  .ssh
Serveur$ ssh-copy-id  -i ma_clef.pub  user1@Client

user1:Client$ cat  $HOME/.ssh/authorized_keys
```

- Vérification

```
user1:Client$ cat  $HOME/.ssh/authorized_keys
```

L'utilisation des clefs SSH

Les clefs doivent être créées sur le poste qui exécute la commande ssh, en l'occurrence sur le serveur maître. La clef publique sera localisée au sein du fichier authorized_keys des serveurs clients.

La clef ne sera pas nommée avec le nom par défaut (id_rsa ou id_dsa) mais avec un nom particulier (comm_serveur_key). L'avantage est de disposer d'une clef spécifique utilisée pour un usage bien particulier dans un contexte donné. Chaque application réseau disposera de sa propre clef de sécurité qui pourra être gérée de manière complètement autonome.

Création de la clef sur le serveur maître :

```
Serveur$  cd   $HOME/.ssh
Serveur$  ssh-keygen  -t  rsa  -f  ma_clef
Serveur$  ls  -l
    ma_clef     ma_clef.pub
```

Mise à jour des serveurs clients :

```
Serveur$ cd  .ssh
Serveur$ ssh-copy-id  -i ma_clef.pub  user1@Client

user1:Client$ cat  $HOME/.ssh/authorized_keys
```

Vérification :

```
Serveur$ ssh    -i $HOME/.ssh/ma_clef    user1@Client
```

Le réseau

Le mail

- Envoyer un email

```
$ mailx -s «sujet de message» destinataire@machine

$ echo «corps du message» | mailx user@machine
$ ps -ef | mailx user@machine
$ mailx user@machine < /etc/passwd
```

- Consulter ses emails

```
$ mailx
Heirloom Mail version 12.4 7/29/08.  Type ? for help.
"/var/spool/mail/jmb": 2 messages 2 new
>N  1 jmb@jeanmarc.spheriu  Fri Apr 29 23:01  19/573   "info serveur1"
 N  2 jmb@jeanmarc.spheriu  Fri Apr 29 23:03  19/612   "contenu du sujet"
&
```

+ - 2 h q R w fichier s fichier

Le mail

Un utilisateur peut utiliser les commandes «mail» ou «mailx» pour envoyer un email.
La commande «mailx» est plus récente que «mail», elle possède également plus d'options.

Pour envoyer un email

Email envoyé à l'utilisateur 'destinataire' sur le poste local :

```
$ mailx destinataire
Subject : contenu du subject
texte du message
du mail
^D
$
```

Terminer le message par « Ctrl D » (^D).

Email envoyé à l'utilisateur 'destinataire' d'un poste distant 'machine' (ou l'adresse IP) :

```
$ mailx destinataire@machine
```

L'option '-s subject' pour le sujet du mail :

```
$ mailx -s «sujet de message» destinataire@machine
```

L'option '-b cc' fournie une liste de destinataires en copies, et l'option '-c bcc' les destinataires en copies cachées. S'il y a plusieurs destinataires, utiliser le caractère ','.

Autre syntaxe pour envoyer un email

```
$ echo «corps du message» | mailx user@machine

$ ps -ef | mailx user@machine

$ mailx user@machine < /etc/passwd
```

Pour consulter ses emails

Pour consulter ses emails, il suffit de saisir la commande « mailx » (ou « mail ») sans argument. L'affichage liste les messages, un 'N' pour New s'il n'a pas encore été consulté, puis le numéro d'identification, l'expéditeur, la date et le sujet de l'email.
Le '& ' est le prompt de l'utilitaire mail.

```
$ mailx
Heirloom Mail version 12.4 7/29/08.  Type ? for help.
"/var/spool/mail/jmb": 2 messages 2 new
>N  1 jmb@jeanmarc.spheriu  Fri Apr 29 23:01  19/573   "info serveur1"
 N  2 jmb@jeanmarc.spheriu  Fri Apr 29 23:03  19/612   "contenu du sujet"
&
```

Pour consulter un email :

- `+` pour le message suivant.
- `-` pour le message suivant.
- `n` pour consulter le message ayant le numéro d'identification 'n'.

```
& 2
Message  2:
From jmb@jeanmarc.spherius  Fri Apr 29 23:03:52 2016
Return-Path: <jmb@jeanmarc.spherius>
X-Original-To: jmb
Delivered-To: jmb@jeanmarc.spherius
Date: Fri, 29 Apr 2016 23:03:51 +0200
To: jmb@jeanmarc.spherius
Subject: contenu du sujet
User-Agent: Heirloom mailx 12.4 7/29/08
Content-Type: text/plain; charset=iso-8859-1
From: jmb@jeanmarc.spherius
Status: R

et voici un autre email
envoyé par mailx

&
```

La sous commande 'h' pour afficher la liste des emails.

```
& h
    1 jmb@jeanmarc.spheriu  Fri Apr 29 23:01  19/573   "info serveur1"
>   2 jmb@jeanmarc.spheriu  Fri Apr 29 23:03  19/612   "contenu du sujet"
&
```

La sous commande 'R' pour répondre qu'à l'expéditeur, et 'r' à tous.

```
& R
To: jmb@jeanmarc.spherius
Subject: Re: contenu du sujet

jmb@jeanmarc.spherius wrote:

> et voici un autre email
> envoyÃ© par mailx
voici ma reponse
a ton email que je viens
de consulter
EOT
&
```

La sous commande 'w nom_fichier' pour sauvegarder le corps de l'email, sans l'entête, au sein du fichier 'nom_fichier'.

```
& w  fichier_write
"fichier_write" [New file] 2/41
```

La sous commande 's nom_fichier' pour sauvegarder l'email, le corps et l'entête, au sein du fichier 'nom_fichier'.

```
& s  fichier_save
"fichier_save" [New file] 29/861
```

La sous commande 'q' pour quitter l'utilitaire mailx.

```
& q
$
```

Le réseau

Les commandes mesg, write et wall

- mesg

```
$ mesg

$ mesg  n            pour interdire
$ mesg  y            pour autoriser
```

- write

```
$ write  jmb

$ write  jmb  pts/0
```

- wall

```
$ wall  "veuillez vous déconnecter, opération de maintenance en cours"
```

Les commandes mesg, write et wall

Il est possible d'envoyer un message directement sur un pseudo-terminal d'un utilisateur sur la même machine sur laquelle on est connecté.

Évidemment, nous pouvons également être le destinataire de tels messages.

La commande «mesg»

La commande «mesg» permet d'interdire ou d'autoriser la réception de ces messages. Par défaut, c'est autorisé.

```
$ mesg  n                  pour interdire
$ mesg
is n

$ mesg  y                  pour autoriser
$ mesg
is y
```

La commande «write»

La commande «write» envoie un message directement sur un pseudo-terminal d'un utilisateur.

```
$ write  jmb
write: jmb est loggé à plus d'un endroit; on écrit à pts/3
bonjour
et à bientôt
$ write  jmb  pts/0
hello
bonne journée à toi.
$
```

La commande «wall»

La commande «wall» envoie un message sur tous les pseudo-terminaux de tous les utilisateurs qui sont connectés sur notre machine.

```
$ wall  "veuillez vous déconnecter, opération de maintenance en cours"
$
```

Le réseau

Les serveurs DNS, DHCP, NFS et LDAP

- DNS Service de noms
- DHCP Service d'adressage réseau
- NFS Serveur de fichiers
- LDAP Service d'annuaire

Les serveurs DNS, DHCP, NFS et LDAP

Serveur DNS **Domain Name System**

Un serveur DNS est un serveur de noms de domaines.

Une machine a besoin de l'adresse IP d'une machine distante pour communiquer avec elle. Lorsqu'une commande utilise un nom de machine, il est donc nécessaire de récupérer l'adresse IP correspondante. Si cette résolution d'IP n'est pas faite un locale, un serveur DNS peut le faire.

Un serveur DNS centralise la correspondance entre des noms de machines et des adresses IP. Un serveur DNS se charge d'un domaine ou nom de domaine. Plusieurs serveurs DNS peuvent communiquer entre eux pour la résolution entre différents domaines.

L'infrastructure du web fonctionne avec ce type de serveurs.

Serveur DHCP **Dynamic Host Configuration Protocol**

Un serveur DCHP délivre des adresses IP aux machines clientes du réseau. C'est donc un fournisseur d'adressage réseau dynamique.

Ainsi, à chaque démarrage d'une machine cliente DHCP, le serveur lui fournira son adresse IP et sa configuration réseau.

L'intérêt est que la configuration réseau d'une machine n'est pas définie en locale, mais centralisée sur un serveur. Cela simplifie la gestion, l'administration et la maintenance des configurations réseaux des postes clients.

Serveur NFS **Network File Server**

Un serveur NFS est un serveur de fichiers. Il centralise des données.

Les machines clientes peuvent accéder à des fichiers localisés sur ce serveur. L'accès à ces données est transparent pour les utilisateurs d'un poste client.

Serveur LDAP **Lightweight Dirctory Access Protocol**

Un serveur LDAP est un annuaire qui fournit des informations à la demande des clients LDAP.
Ces serveurs sont optimisés pour les opérations de lectures, donc pour répondre rapidement aux sollicitations des clients.

Ce type d'annuaire peut être configuré pour contenir un grand nombre d'informations de différents types. Il peut centraliser beaucoup de données de configuration indispensables à des machines clientes, telles que :

- la résolution de noms de machines en adresse IP,
- la définition des comptes utilisateurs,
- la résolution pour les numéros de réseaux,
- la correspondance entre des protocoles et des ports réseaux,
- etc...

Cette centralisation d'informations en simplifie la gestion, l'administration et la maintenance pour l'équipe d'administration de l'infrastructure informatique et réseau de l'entreprise.

Notes

Fin de session de Formation

Je vous recommande de relire ce support de cours d'ici les deux semaines à venir, et de refaire des exercices.

Il ne vous reste plus qu'à mettre en œuvre ces nouvelles connaissances au sein de votre entreprise.

Merci, et à bientôt.

Jean-Marc Baranger

Theo Schomaker