Systèmes d'exploitation de type Unix

École des Mines de Nancy
Séminaire Informatique et Internet

Gaëtan Bisson
gaetan.bisson@loria.fr

Présentation

Un *système d'exploitation* est une collection de programmes fournissant une interface qui permet aux utilisateurs de tirer parti des ressources d'une machine ; aujourd'hui, le plus connu est doute Microsoft Windows (apparu en 1985) mais il en existe une multitude d'autres, certains bien antérieurs.

Nous parlerons ici des systèmes *de type Unix*, [1] comprenant notamment BSD, Solaris et Linux. Ce dernier connaît récemment un relatif succès : ses fonctionnalités, sa stabilité et sa flexibilité en font un système de choix aussi bien pour des ordinateurs de bureau que des applications embarquées (e.g. téléphones portables).

Les *distributions* «léchées» (comme Ubuntu, distribution d'un système Linux) sont entièrement configurables par le biais d'interfaces graphiques intuitives — penser à Mozilla Firefox. Nous supposerons le lecteur familier avec ce type d'environnement (comportant souris, boutons, fenêtres, bureaux virtuels, etc.) et mettrons l'accent sur ce qui est spécifique à Unix :

1. ses principes d'organisation ;
2. l'interface centrale qu'est le shell.

Essayer Linux. La distribution Ubuntu est probablement la plus facile à prendre en main. [2]

1 Principes d'organisation

En marge, nous noterons les noms des programmes standards permettant de manipuler les notions décrites ; le lecteur pourra plus tard s'y référer via la section 3.

1.1 Système de fichiers

Toute donnée est contenue dans un fichier. Ces fichiers forment les feuilles d'une arborescence de répertoires, dont un emplacement est noté par la succession des répertoires le contenant, séparés par le caractère «`/`». On écrira donc

`pwd` `ls` `cd` `cp` `mkdir` `rm`

le répertoire `/nombres/premiers/`
contient le répertoire `/nombres/premiers/impairs/`
ainsi que le fichier `/nombres/premiers/deux`

1. C'est-à-dire qui ressemblent au système Unix original, développé dans les années 1970.
2. Au sens où ce cours n'est en rien un prérequis à son utilisation.

et l'on notera la racine «/».

Chaque élément de l'arbre possède des attributs, aussi appelés *métadonnées*, par exemple les dates de dernier accès et de dernière modification, ou encore un *bit* indiquant si le fichier est exécutable ou non.

`stat`

Cette structure d'arbre est élégante et, pour en décupler l'impact, un des principes d'Unix est de tout y regrouper. Ainsi, on retrouvera l'arborescence de son disque dur en `/` et celle de sa clef USB en `/media/usb/`,[3] mais on pourra aussi lire des informations sur la batterie dans `/proc/acpi/battery/BAT0/state` voire même récupérer le flux vidéo de la webcam par `/dev/video0`. Ce dernier fichier, qui ne contient pas de données «figées» mais sert d'interface avec un périphérique, est qualifié de *spécial*.

`mount`

Tout cela est organisé sémantiquement, les principales branches étant :

`/`	la racine, contient quasi-exclusivement ce qui suit ;
`/bin/`	contient les programmes *essentiels* (en nombre réduit) ;
`/dev/`	où se trouvent les fichiers spéciaux ;
`/etc/`	où sont placés les configurations des différents programmes ;
`/home/`	renferme les répertoires des utilisateurs ;
`/proc/`	renferme des fichiers *virtuels* qui documentent le système ;
`/usr/`	similaire à `/` mais en moins restrictif ;
`/usr/bin/`	contient la majorité des programmes.

1.2 Multi-utilisateur

Dès l'origine, Unix a été conçu pour être un système *multi-utilisateur*, c'est-à-dire que plusieurs personnes peuvent utiliser simultanément, en ayant des fichiers et applications séparées. Au niveau du système de fichiers, cela prend la forme de métadonnées appelées *droits d'accès* qui spécifient quels utilisateurs sont autorisés à accéder aux fichiers et répertoires.

`chown`
`chmod`

Naturellement, on ne peut concéder à tous le droit de modifier `/bin/` ou encore `/dev/` (qui contient une interface vers le contenu entier de la mémoire vive), mais l'on souhaiterait tout de même que quelqu'un puisse ne serait-ce que mettre à jour les programmes qui s'y trouvent. Il existe donc un utilisateur tout-puissant, nommé `root`, qui est l'*administrateur système* de la machine ; l'utilisateur lambda n'aura quant à lui permission de ne modifier que ce qui se trouve dans son répertoire, `/home/lambda/`.

`su`

Les utilisateurs sont authentifiés par leurs noms et mots de passe lorsqu'ils se *loguent*, que ce soit au clavier de la machine ou à distance par un réseau — penser au Minitel. La liste des mots de passe est stockée chiffrée dans le fichier `/etc/shadow` (lisible par `root` seul) et à chaque tentative de logue le système y compare simplement le chiffré du mot de passe tapé.

`login`

1.3 Processus

Au cœur du système se trouve le *noyau*, un programme qui coordonne tout ; nous le distinguerons des autres programmes lancés en appelant ces derniers *processus*. Les tâches incombant au noyau sont de :

- distribuer équitablement les ressources du système (processeur, bande passante, etc.) entre les différents processus ;

`top`

- offrir un certain nombre d'interfaces permettant à ces derniers d'interagir avec les périphériques de la machine.

`modprobe`
`lsmod rmmod`

3. On peut décider soi-même de l'emplacement, appelé *point de montage*.

De façon pragmatique, un programme n'est qu'un fichier contenant une suite d'instructions destinées au noyau et au processeur ; lorsqu'un programme est lancé, son contenu est d'abord copié en mémoire puis exécuté de là — on peut alors modifier (notamment, mettre à jour) le fichier sous-jacent sans perturber le processus ; en outre, un même programme lancé par plusieurs utilisateurs donnera lieu à plusieurs processus distincts coexistant en mémoire (chacun n'ayant accès qu'à sa propre zone mémoire).

Chaque processus est identifié par un entier unique appelé *PID*. Le premier processus, nommé `init`, à partir duquel sont (directement ou indirectement) lancés tous les autres, a pour PID un ; on le qualifie de *démon* car il est amené à s'exécuter en arrière-plan jusqu'à l'arrêt complet du système, ce qui le distingue des processus lancés plus ponctuellement, comme par exemple OpenOffice. Ce type de processus peut, entre autres, être contrôlé par le biais de *signaux* ; il en existe notamment pour mettre en pause, reprendre l'exécution et arrêter totalement un processus.

`ps`

`kill`

La majorité des programmes est paramétrable. Ces paramètres sont souvent optionnels lorsque les valeurs par défaut conviennent aux cas d'utilisation typiques ; ils sont habituellement spécifiés :

- dans un fichier de `/etc/` (commun à tous) ;
- dans un fichier de `/home/utilisateur/` (propre à l'utilisateur) ;
- par des arguments.

Ces derniers sont directement transmis au processus lors de son lancement ; il doit par contre lui-même aller lire les éventuels fichiers de configuration.

Les processus interagissent avec (c'est-à-dire, lisent et écrivent) des fichiers par des *descripteurs de fichiers*. Trois sont réservés pour permettre une interactivité minimale du processus avec son environnement, et plus précisément son processus *père* (celui qui l'a lancé) :

`lsof`

0. `stdin`, l'entrée standard ;
1. `stdout`, la sortie standard ;
2. `stderr`, l'erreur standard.

Quand il se termine, chaque processus renvoie à son père un entier appelé *code de retour*, par convention nul s'il s'est exécuté normalement et non nul lorsqu'une erreur est survenue, la valeur pouvant indiquer la nature de l'erreur.

2 Interagir via le shell

L'interface traditionnelle avec un système de type Unix est le shell ; c'est aussi la plus complète. Elle est intimement liée au fonctionnement interne du système. Un *shell* interprète des *lignes de commandes*, ce qui a pour effet de lancer (ou contrôler) des processus ; l'interaction avec l'utilisateur (réception et affichage des données) est quant à elle gérée par un autre programme, appelé *terminal*.

Nous sommes en mode graphique ; lançons un terminal et plaçons-y un shell par la commande `xterm -e bash`.[4] Le terminal est vide à l'exception d'un début de ligne contenant de succinctes informations suivies du caractère «`$`» puis d'un curseur ; c'est à nous de jouer.

2.1 Premiers pas

Lorsque la ligne de commande ne comporte qu'un nom de programme, éventuellement suivi d'arguments, le shell se contente de lancer le processus correspondant, redirigeant "le clavier" vers son entrée standard et sa sortie standard vers "l'écran" ; essayons par exemple :

4. Comme souvent dans le monde *libre*, il existe une grande diversité de terminaux et de shells.

```
$ date
Wed Sep 16  8:29:59 CEST 2009
$ date -u
Wed Sep 16 10:30:00 UTC 2009
```

Remarquer qu'on a omis de spécifier l'emplacement du programme à lancer : le shell cherche alors un fichier exécutable à ce nom parmi la liste de répertoires que contient la variable `PATH` (ci-dessus, il aura finalement lancé `/bin/date`), que l'on peut par ailleurs afficher en donnant sa valeur `$PATH` comme argument au programme `echo` (qui se contente d'afficher ses arguments), comme il suit :

```
$ echo $PATH
/bin:/usr/bin
```

L'interactivité potentielle du terminal sera certainement mieux illustrée par l'emploi d'un programme primitif de gestion de boîte aux lettres électronique, où nous indiquons en rouge les caractères tapés par l'utilisateur :

```
$ mail
"/var/spool/mail/claude": 2 messages 2 new
>N  1 pierre@habert.org     Sat Feb 19 11:06   6/157   "All in?"
 N  2 constance@djvu.org    Sat Feb 19 12:09   7/149   "Jamais !"
& 1
Message 1:
From pierre@habert.org  Sat Feb 19 11:06:13 1558
To: claude@garamond.org, constance@djvu.org
Subject: All in?

Quid d'une soirée poker chez moi ce soir ?

& q
Saved 1 message in mbox
Held 1 message in /var/spool/mail/claude
$ mail -s "Poker ce soir" pierre@habert.org
Ça roule !
P.S. Il semblerait que David ait vendu la mèche à Constance.
^DEOT
$
```

Le sigle «`^D`» dénote classiquement l'appui simultané des touches `Ctrl` et `D` ; ce caractère spécial n'est en réalité pas affiché à l'écran. Il signifie la fin du *flux* «entrée standard», c'est-à-dire ici qu'on a fini de taper notre message ; la seconde instance du programme `mail` se termine alors et le shell, reprenant la main, imprime une nouvelle invite de commande.

Lancez vous-même les programmes `cal`, `df`, `xeyes` [5] et `eject` ; on pourra obtenir des informations sur chacun en donnant leur nom comme argument au programme `man` (qui se quitte avec la touche `Q`). Apprenez aussi à vous déplacer dans l'arborescence des répertoires en utilisant les commandes mentionnées en 1.1.

Le programme `cat` imprime sur sa sortie standard le contenu des fichiers dont les noms lui sont fournis en argument ; la commande `cat /etc/group` affichera donc la liste des groupes d'utilisateurs ainsi que leurs membres. Souvent, il est bien plus commode d'utiliser un *éditeur*

5. Pour envoyer le signal d'interruption à un processus tournant dans un terminal, on y tapera «`^C`».

de texte offrant des fonctionnalités de recherche et d'édition, comme par exemple `nano` ; vous n'avez naturellement pas la permission de modifier les fichiers situés dans `/etc/`, mais vous pourrez créer un fichier dans votre propre répertoire (noté `~/`), en exécutant la commande `nano ~/fichier_exemple`.

2.2 Combiner des commandes

On peut, en une ligne de commande, lancer séquentiellement plusieurs programmes, en intercalant le séparateur « ; ». Un autre séparateur est «`&`», qui a pour effet de lancer en arrière-plan le programme le précédant (voir la commande `jobs`).

D'autres séparateurs font davantage, à commencer par le *pipe* « | » qui redirige la sortie standard du programme précédant vers l'entrée standard du programme suivant ; ses applications son innombrables ! Listons, par exemple, les groupes dont on fait partie :

```
cat /etc/group | grep $USER
```

ou bien les dix derniers utilisateurs logués :

```
last | head
```

Jouons encore un fichier sur un ordinateur distant :

```
cat music/save_the_king.mp3 | ssh computer.in.uk mplayer -
```

Citons enfin les deux autres séparateurs que sont «`&&`» et « | | » ; ils lancent le programme suivant respectivement lorsque le précédent a réussi (code de retour nul) ou échoué. Cela joue le rôle de *branchement conditionnel*, par exemple pour rechercher des ordinateurs à l'heure japonaise :

```
for machine in garamond.org djvu.org www.loria.fr; do
  ssh $machine 'date | grep -q JST' && echo $machine
done
```

3 Trouver de l'aide

La première source d'information est *le manuel*, accessible par le programme `man`, qui documente tous les programmes standards d'Unix ; son format est spartiate, aussi faut-il généralement savoir ce qu'on y recherche. Une référence non moins sèche mais plus introductive est le vieil ouvrage [1] ; plus concis, [2] plaira aux adeptes du français. Les plus curieux pourront enfin consulter [3].

Lorsqu'une erreur survient, notamment quand la machine ne comprend pas votre requête, ***lisez le message d'erreur*** ; généralement, c'est ensuite une bonne idée d'interroger Google. Dans tous les cas, pendant le TP, n'hésitez pas à questionner l'enseignant.

Références

[1] Brian W. Kernighan et Rob Pike. *The Unix Programming Environment*.

[2] Tuteurs informatique de l'ENS. *Documentation sur Unix*.

[3] Jacques Beigbeder. *Programmation Système Unix*.

A Exercices

Familiarisation

Prenez quelques minutes pour vous approprier l'environnement graphique : commencez par vous loguer puis vous déloguer, lancez ensuite quelques programmes tels Mozilla Firefox, explorez le système de fichier via Nautilus, etc.

A.1 Prise en main

Fichiers. Créez un fichier `toto` et un répertoire `bla` ; copiez `toto` dans `bla`. Allez en `bla` et déplacez-y le fichier `toto` original. Supprimez enfin le répertoire `bla`.

Métadonnées. Quand le fichier `/etc/profile` a-t-il été modifié pour la dernière fois ? Pouvez-vous changer cette date (cf. `touch`) ? Créez un fichier que vous seul pourrez lire ; d'autres pourraient-ils tout de même y écrire ?

Détente. Lancez des programmes sur la machine du voisin.

A.2 Découverte

Recherche. Les fichiers situés dans `/usr/share/dict/` listent des mots courants ; via le programme `grep`, trouver ceux formés uniquement de voyelles.

Mystère. Que fait la commande suivante ?

```
md5sum * | sort | uniq -d -w 32 | cut -c 35- | xargs rm
```

Sécurité. Dans le fichier `/etc/shadow`, les mots de passe sont chiffrés comme le fait la commande `openssl passwd` ; sachant qu'il s'agit bêtement d'un nom commun, retrouver celui de l'utilisateur ayant pour entrée :

```
gerald:$1$rcWVxfgW$m5TaUuu6oBiQTpAG681OE1:0:0:99999:7:::
```

A.3 Perfectionnement

Automatisation. Faites usage des programmes `wget` et `mogrify` afin de retourner toutes les images de la page Web `http://education.gouv.fr/`.

Remplacement. Apprenez à utiliser l'outil `sed` afin de mettre joliment en forme, par exemple au format HTML, le texte d'un vieil ouvrage tel :

```
http://www.gutenberg.org/files/13951/13951-8.txt
```

Scripts shell. Un script est un fichier exécutable dont la première ligne est `#!/bin/sh` ; les lignes de commandes qu'il contient sont alors lancées par le programme `sh`, qui présente ses arguments par la variable `@`. Écrire un script qui détermine dans quelle ville d'un pays donné la température est la plus élevée aujourd'hui.