Initiation à l'environnement Unix
Bilan et perspectives

Pierre Rousselin

Université Paris 13
L1 informatique
décembre 2021

Introduction

Première partie : être autonome sur sa machine

Deuxième partie : les fichiers

Troisième partie : les processus

Récapitulatif sur le shell

Regrets et perspectives

Dernier cours magistral

Qu'est-ce que je sais faire après ce module que je ne savais pas faire avant ?

Dernier cours magistral

Qu'est-ce que je sais faire après ce module que je ne savais pas faire avant ?

Oublier définitivement la « magie des 4 croix » (culte du cargo).

Introduction

Première partie : être autonome sur sa machine

Deuxième partie : les fichiers

Troisième partie : les processus

Récapitulatif sur le shell

Regrets et perspectives

Utiliser le shell

- (noyau de) système d'exploitation : couche logiciel entre le matériel (mémoire, processeur, périphériques) et les autres programmes
- le shell : interface textuelle entre l'utilisateur et le noyau du système
- nom de commandes et arguments, séparés par des blancs
- caractères spéciaux du shell, parfois besoin de les inhiber
- cuisine du shell : le shell transforme la ligne de commande avant qu'elle soit lancée

Arborescence des fichiers

- Répertoire : fichiers qui ne font que « contenir » (en fait nommer) d'autres fichiers (normaux au répertoire).
- Tout fichier est contenu dans un répertoire.
- Sauf le répertoire racine, noté `/`, de l'arborescence.
- Chemin absolu : suite de noms de fichier, dont le premier est le répertoire racine `/`, suivi de noms de répertoire séparés par le caractère `/`, avec éventuellement un nom de fichier non répertoire à la fin. Exemples : `/usr/include/stdio.h` et `/etc/config` (ou `/etc/config/`).

Arborescence des fichiers

- Répertoire : fichiers qui ne font que « contenir » (en fait nommer) d'autres fichiers (normaux au répertoire).
- Tout fichier est contenu dans un répertoire.
- Sauf le répertoire racine, noté `/`, de l'arborescence.
- Chemin absolu : suite de noms de fichier, dont le premier est le répertoire racine `/`, suivi de noms de répertoire séparés par le caractère `/`, avec éventuellement un nom de fichier non répertoire à la fin. Exemples : `/usr/include/stdio.h` et `/etc/config` (ou `/etc/config/`).
- Chemin relatif (au répertoire courant) : ne commence pas par le répertoire racine `/` : le chemin absolu correspondant s'obtient en concaténant le chemin absolu du répertoire courant à ce chemin relatif. Exemples : si le répertoire courant est `/home/alice`
  - `unix/tp2/tp2.pdf` → `/home/alice/unix/tp2/tp2.pdf`
  - `../bob/.bashrc` → `/home/alice/../bob/.bashrc` → `/home/bob/.bashrc`
  - `./a.out` → `/home/alice/./a.out` → `/home/alice/a.out`

Répertoire personnel

- répertoire personnel à un utilisateur donné
- en général, il y a tous les droits et les autres utilisateurs assez peu
- répertoire courant initial lorsqu'il lance un shell
- en shell : `~` (développement du tilde) ou `$HOME` (développement de variable) sont développés en le chemin absolu du répertoire personnel. Si celui-ci est `/home/alice` :
  - `~/unix/tp2` → `/home/alice/unix/tp2`
  - `$HOME/.bashrc` → `/home/alice/.bashrc`

Arborescence : commandes usuelles

- `pwd`
- `cd`
- `ls` (options à connaître `-l -d -a -i -h -t -r`).
- `mkdir` (option `-p` à connaître) et `rmdir`
- `cp` (options à connaître `-R` et `-i`), deux syntaxes différentes à connaître
  - `cp src dest` où `dest` n'est pas un répertoire existant
  - `cp src... rep` où `rep` est un répertoire existant
- `mv` (option à connaître `-i`), deux syntaxes comme `cp`
- `rm` (options à connaître : `-r -f -i`)
- `touch` permet de créer un fichier normal vide

Type de commande et aide

- Types de commande : *alias*, *primitive* (commande interne du shell), *commande externe* (et *fonction shell* dont on n'a pas parlé)
- `type` : connaître le type d'une commande
- `man` : afficher une page de manuel pour une commande externe
- `help` (dans `bash`) : afficher de l'aide pour une primitive

Être bien chez soi

- Fichier de configuration `.basrhc` (caché car le nom commence par un point) lu à chaque lancement de `bash`
- Possible d'ajouter ses propres `alias` ou définitions de variables shell pour toujours en profiter
- Recherche des commandes externe par le shell : dans les répertoires qui apparaissent dans la variable d'environnement `PATH`.
- Possible de créer un répertoire (par exemple `~/bin/`) pour y mettre ses propres programmes ou scripts (ou ceux des autres) puis l'ajouter au `PATH` en écrivant dans `.bashrc` :

```
PATH="$HOME/bin:$PATH"
```

Super-utilisateur

- Pour des changements plus radicaux (installation de programme et bibliothèques pour *tous* les utilisateurs, mise à jour du système, ajout ou suppression d'utilisateurs...) on doit être super-utilisateur (utilisateur `root`)
- Pour exécuter une commande en tant que `root`, on la fait précéder de `sudo` (bien sûr si on a le droit).
- `root` peut toujours tout faire (grandes responsabilités, besoin d'une grande éthique personnelle)

Introduction

Première partie : être autonome sur sa machine

Deuxième partie : les fichiers

Troisième partie : les processus

Récapitulatif sur le shell

Regrets et perspectives

Fichier

- tout ce dans quoi on *lit* et/ou *écrit* des données est un fichier, même les périphériques ;
- un fichier ne contient *rien d'autre* que la suite d'octets qui a été écrite dedans ;
- *l'inode* d'un fichier, géré par le système, contient les attributs du fichier :
  - numéro d'inode
  - utilisateur propriétaire
  - groupe propriétaire
  - périphérique (où trouver le contenu ?)
  - dates (accès, modification, changement)
  - permissions (`rwx`) pour `ugo`
  - taille
  - nombre de liens
- nom de fichier : dans le ou les répertoires qui le contiennent : commande `ln` pour créer un lien nom $\leftrightarrow$ inode dans un répertoire (et en fait `rm` supprime un tel lien).

Manipuler l'inode

La commande non standard `stat` permet de lire l'inode.
Voir aussi `ls -li` (ou `ls -lid` pour voir les informations sur un répertoire).
Possible de manipulier l'inode seulement pour le propriétaire du fichier (et évidemment `root`) :

- `chmod` : changer les permissions
- `touch` : changer les dates
- `chown` (seul root peut le faire) : changer le propriétaire
- `chgrp` : changer le groupe propriétaire (seulement avec un autre de ses groupes ou si on est `root`)

Permissions

- Fichiers normaux : `r` pour l'ouvrir en lecture, `w` pour l'ouvrir en écriture, `x` pour (demander au noyau de) l'exécuter, c'est-à-dire faire du fichier le programme (ou script) d'un processus
- Répertoire : `r` pour lire les noms des fichiers qu'il contient, `w` pour modifier ces noms (`mv`), ou en ajouter ou supprimer (`rm`) et enfin `x` pour permettre l'association nom $\leftrightarrow$ inode (pour, par exemple, l'ouvrir, ou chercher des informations contenues dans l'inode). Donc presque rien n'est possible sans `x`.

Fichiers texte et fichiers « binaires »

- Fichier texte : lisible par un humain dans un certain encodage de caractères (ASCII, UTF-8).
  Exemples : scripts, fichiers source `.c`, ...
  - ASCII et dérivés obsolètes (Latin-1, ...) : 1 caractère prend 1 octet (8 bits)
  - UTF-8 : 1 caractère prend entre 1 octet (pour les caractères représentables en ASCII et seulement ceux-là) et 6.

  Attention aux fins de ligne (CRLF, c'est-à-dire `\r\n` sous Windows, seulement `\n` sur Unix).
- Fichier binaire : lisible par un autre programme, pas par un humain.
  Exemples : programmes compilés, fichiers `pdf`, `docx`, ...
- La commande `file` tente (souvent avec succès) de « deviner » le « type » (notion volontairement floue sous Unix) du fichier.

Introduction

Première partie : être autonome sur sa machine

Deuxième partie : les fichiers

Troisième partie : les processus

Récapitulatif sur le shell

Regrets et perspectives

Processus

- « Instance d'un programme en cours d'exécution »
- Plusieurs processus (typiquement quelques centaines) vivants « en même temps »
- L'ordonnanceur du système organise le partage du processeur entre tous les processus (tourniquet).

Processus : arborescence

- Tout processus naît d'un parent, sauf le processus numéro 1 (`init` ou `systemd` ou `launchd` ...).
- Le parent *fork* pour créer un enfant, cet enfant exécutera ensuite un programme.
- Un processus a un numéro qui l'identifie : le `PID` (en shell, variable `$$`). Le numéro de son parent est le `PPID` (*parent PID*), dans `bash` variable `$PPID`.
- Pour chaque commande externe, le shell *fork*.

Attributs des processus

- utilisateur propriétaire, groupe propriétaire ;
- `PID` et `PPID` ;
- date de départ et temps d'utilisation processeur ;
- terminal de contrôle ;
- nom de commande, en shell `$0` ;
- arguments, en shell `$1`, `$2`, ... ;
- environnement ;
- descripteurs de fichier pour les fichiers ouverts ;
- pointeur d'instruction (où en sommes-nous dans le programme ?) ;
- état (`R` pour *running*, `S` pour *sleeping*, `T` pour *inTerrupted*, ...) ;
- ...

Commande `ps` et (non standard) `pstree`, `top`, `htop`, ...

Mort d'un processus

- Lorsqu'un processus se termine (meurt), il transmet un petit entier à son parent : son statut de sortie (*exit status*).
- 0 pour succès ou vrai
- > 0 pour échec ou faux
- en shell, statut de la dernière commande : `$?`

Introduction

Première partie : être autonome sur sa machine

Deuxième partie : les fichiers

Troisième partie : les processus

Récapitulatif sur le shell

Regrets et perspectives

Cuisine du shell

- Développement du tilde (`~`)
- Développement de variable `$var` ou `${var}`, développement arithmétique `$((calcul))` et substitution de commande `$(commande)`
- Séparation des champs (*field splitting*) pour les développements de variable, arithmétique et substitution de commande qui ne sont pas entre `""`, selon les caractères contenus dans la variable `IFS`
- Développement de noms de chemins `* ? [...]`
- Suppression des caractères inhibiteurs (*quote removal*) : `''`, `""` et `\`.

Paramètres spéciaux

- `$0` : nom de commande
- `$1`, `$2`, `$3`, ... : arguments de la ligne de commande (ou paramètres positionnels)
- `$#` : nombre d'arguments
- `"$@"` : liste de tous les arguments
- `set -- arg1 arg2 ...` : redéfinir les paramètre positionnels
- `shift [n]` : décaler les paramètres positionnels
- `$$` : PID
- `$?` : statut de sortie de la dernière commande

Construction case

```
case chaine_entre_case_et_in in
motif11 | motif12 | ...)
    commandes1
    ;;
motif21 | motif22 | ...)
    commandes2
    ;;
...
motifn1 | motifn2 | ...)
    commandesn
    [;;]
esac
```

Construction `for`

```
for var in mot1 mot2 ... motn; do
    commande1
    commande2
    ...
done
```

Sans `in` : itérer sur les arguments (comme `in "$@"`).

Construction `if`

```
if commande_if; then
    commandes_bloc_if
elif commande_elif1; then
    commandes_bloc_elif1
elif commande_elif2; then
    commandes_bloc_elif2
else
    commandes_bloc_else
fi
```

- Les *statuts de sortie* de `commande_if`, `commande_elif1`, ... (0 pour « vrai », > 0 pour « faux ») déterminent le bloc qui sera exécuté (s'il y en a un).
- Les parties `elif` et `else` sont facultatives.
- Voir aussi commande `test` ou `[ ... ]`.

Construction `while`

```
while commande; do
    cmds_bloc_while
done
```

- Comme pour `if`, c'est le statut de sortie de `commande` (0 pour « vrai ») qui détermine si on va exécuter le bloc `cmds_bloc_while`.
- Remarque utile : `! cmd` pour exécuter `cmd` en changeant succès en échec et réciproquement. Opérateurs logique `&&` pour « et » et `||` pour « ou ».

Redirections et tubes

- `>fichier` ou `1>fichier` : rediriger la sortie standard de la commande vers `fichier` (créé s'il n'existe pas, vidé s'il existe) ;
- `>>fichier` ou `1>>fichier` : rediriger la sortie standard de la commande vers `fichier` (créé s'il n'existe pas, on écrit à la suite s'il existe) ;
- pour l'erreur standard : `2>fichier` ou `2>>fichier` ;
- entrée standard : `<fichier` ;
- tube pour envoyer la sortie standard de `cmd1` dans l'entrée standard de `cmd2` : `cmd1 | cmd2` ;
- nouvelle façon de penser : fichier comme flux et commande comme filtre (ou fonction qui transforme le flux).

Introduction

Première partie : être autonome sur sa machine

Deuxième partie : les fichiers

Troisième partie : les processus

Récapitulatif sur le shell

Regrets et perspectives

Commandes non abordées :

- `find` : pour chercher des fichiers dans une sous-arborescence et exécuter des commandes dessus, surpuissant (et dangereux) ;
- `tar` : archiver, compresser et désarchiver, décompresser ;
- expressions rationnelles (ou régulières) et plus de choses sur `grep`, `sed`, `awk` ;
- `make` : gérer les dépendances entre fichiers pour la compilation (voir second semestre).
- `git` : gestionnaire de versions et travail collaboratif.

Concepts non abordés :

- Signaux (commande `kill`) entre processus.
- Interception des signaux (commande `trap`).
- Systèmes de fichiers (*filesystems*), points de montage, *block devices* : organisation concrète des données des fichiers sur les périphériques.

Parties du langage *shell* non abordées :

- Le *job control* : lancer des commandes en arrière-plan. Terminateur de commandes `&`, commandes `fg`, `bg` et `jobs`, signal `SIGTSTP` (Ctrl-Z).
- La duplication de descripteurs de fichiers, par exemple `2>&1`.
- Les fonctions shell.
- Les développements de variable avancés :

```
${var%suffixe_a_enlever}
${var%%suffixe_a_enlever}
${var#prefixe_a_enlever}
${var##prefixe_a_enlever}
${var-valeur_par_defaut}
${var=valeur_par_defaut}
...
```

- `eval` (demander au shell de faire 2 fois sa cuisine) et `exec` (ouvrir des fichiers ou exécuter un autre programme dans le même processus).