Document Numérique – 1

Michel Beigbeder

19 septembre 2017

Introduction

- Documents - traitement
- Fichiers - Programme
  - Flots - Processus
- Format de fichier

Plan

- Modèle de processus (entrées-sorties)
- Quelques commandes
- Script shell (*sh*)
- Expressions rationnelles

Un modèle de processus Unix

- **stdin** – entrée standard

  les données traitées, à moins que des noms de fichiers soient donnés en arguments.
- **stdout** – sortie standard

  le résultat du traitement.
- **stderr** – sortie d'erreur standard

  Les messages d'erreurs destinés à l'utilisateur, visibles même si la sortie standard a été redirigée.

- **argc, argv** – les arguments

  typiquement des options et éventuellement les noms des fichiers à traiter.

  nombre limité (de l'ordre du millier)
- **ENV** – les variables d'environnement

  dont certaines peuvent modifier le comportement.

- **DIAGNOSTICS** – le code de retour du processus

  tel que retourné par `exit()` ou `return` dans la fonction `main()` ;

  utilisé par `if`, `while`, `until`, `&&`, `||`.

Les redirections

- **stdout** vers un fichier : `> filename`
- **stdout** vers **stdin** : `cmd1 | cmd2`
  ou les sorties standards de plusieurs commandes vers **stdin** :
  `{ cmd1a ; cmd1b ; } | cmd2`
- un fichier vers **stdin** : `< filename`
  ou plusieurs fichiers vers **stdin** : `cat file1 file2 | cmd2`
- depuis le script vers **stdin** : `<< marker`
  jusqu'à la ligne qui contient *marker* et uniquement *marker*. Souvent on choisit le marqueur *EOF*.
  Si le marqueur n'est pas banalisé, les variables sont remplacées par leur valeur.
- **stdout** vers des arguments : `` `cmd` `` ou `xargs`

Désignation des fichiers

- absolue : le cheminon commence par un caractère /
- relative : le cheminon ne commence pas par un caractère /

Exemples (1/2)

- **Énumérer des fichiers**
  - avec le shell : `*`, `.*`, `*.c`, `[a-z]*.c`
  - avec la commande `ls`
  - avec la commande `find`
- **Numéroter les lignes :** `cat`, `nl`
- **Extraire des lignes**
  - le début : `head`
  - la fin : `tail`
  - avec leur numéro : `sed`

    C'est un usage dégradé de `sed` qui est un ≪ éditeur de flot ≫ (*stream editor*)
  - selon un patron : `grep`
- **Trier des lignes :** `sort`

  Attention, l'ordre des caractères et donc le tri dépendent de variables d'environnement
- **Intersection et différence entre ensembles :** `comm`

  un élément par ligne, les fichiers qui représentent les ensembles doivent être triés

Exemples (2/2)

- **Remplacement de caractères :** `tr`, `expand` pour développer les tabulations
- **Traitement de chaînes de caractères** : `basename` et `dirname` pour les cheminons, `expr`, ou `sed` pour ce qui est plus compliqué, et. . . assemblage de variables
- **Formater une donnée :** `printf`, `echo`
- **Extraire des colonnes :** `cut`
- **Regrouper des colonnes :** `paste`

Les scripts

Des commandes (internes ou externes), des structures de contrôle, des données dans un fichier. . . de la programmation dans un langage, interprété.

- le fichier doit commencer par une ligne qui indique au système d'exploitation l'interprète à lancer : `#! /bin/sh`

  L'usage veut que le nom du fichier reflète la nature de ce qu'il contient et se terminera par `.sh`
  Le fichier est rendu exécutable avec la commande `chmod` :

  `chmod +x` *`script`*`.sh`
- **Commentaires** : introduits par le caractère #
- **Séparateurs de commandes** : les caractères `;` et `NL`
- **Caractères de banalisation** : les caractères \, ’ et "
- **Variables** : de type chaînes de caractères

  Calculs par la commande `expr` : `n=‘expr $n + 1‘`. Tous les espaces sont requis.

Les structures

- Les groupements de commandes : `{ cmd ; ...cmd ; }`

  Il faut un espace après l'accolade ouvrante !

  Ces groupements peuvent être nommés : notion de fonction, différente de la notion de sous-shells. `functionname () { cmd ; ...cmd ; }`
- Les sous-shells `(cmd ; ...cmd)`

  (parfois implémentés par de nouveaux processus)
- Structure conditionnelle :

  `if cmd ; then cmd ; else cmd ; fi`

  Toutes les commandes retournent un code (cf. `exit`). La valeur 0 indique la valeur booléenne VRAI. Toute autre valeur indique FAUX. La commande `:` retourne toujours 0. La commande `[` (aussi appelée `test`) permet les tests au sens habituel ((sur des fichiers) `-e` `-r` `-d` ... (numériques) `-eq` `-gt` ... (alphanumériques) `=` `>` ... (logiques) `-a` ...
- Enchaînements conditionnels : `cmd && cmd` `cmd || cmd`
- Énumération : `for var [in liste] ; do cmd ; done`

  énumère par défaut les arguments (`"$@"`)
- Boucle : `while cmd ; do cmd ; done`

Les arguments

- typiquement séparés sur la ligne de commande par un caractère d'espacement (SP *space*, HT *Horizontal Tab*, ou NL *New Line*)

  ensemble de séparateurs modifiable grâce à la variable IFS
- $# – pseudo-variable donnant le nombre d'arguments

  (≪ équivalente ≫ à *argc* en C)
- $0 – nom du programme exécuté
- $1...$9 – les neuf premiers arguments
- **shift** – renommage, ou décalage, ou suppression
- $* – les valeurs des arguments
- ¨$*¨ – la contaténation des valeurs des arguments
- ¨$@¨ – le tableau des arguments

Expressions rationnelles (ou : expressions régulières)

Notion commune à plein d'outils :

- **more, less** – *pagers*
- **ed, vi, vim** – *text editors*
- **sed** – *stream editor*
- **awk** – *pattern-directed scanning and processing language*
- **perl** – *Practical Extraction and Report Language*
- **grep, egrep, fgrep** – *print lines matching a pattern*
- **csplit** – *split files based on context*

Fonctions de la bibliothèque standard de C

- **regex(3)** – *regular-expression library*

Une expression rationnelle est une chaîne de caractères appelée « motif » ou « patron » (« *pattern* » en anglais) décrivant un ensemble de chaînes de caractères dans le but d'en trouver les instances dans un bloc de texte. (Inspiré de la page de `https://fr.wikipedia.org/wiki/Expression_rationnelle`)

Les mécanismes de base pour former de telles expressions [les motifs] sont basés sur des caractères spéciaux de substitution, de groupement et de quantification.
Quantificateurs :

- `?` groupe qui existe zéro ou une fois
- `*` groupe qui existe zéro, une ou plusieurs fois
- `+` groupe qui existe une ou plusieurs fois

Autres opérateurs :

- `[ ]` définition d'ensembles de caractères
- `|` alternative
- `^` prédicat « début de ligne »
- `$` prédicat « fin de ligne »

Documentation

À compléter par la lecture de pages de manuel sur votre machine

- `man ls`
- `man find`
- `man man`
- ...

et par un document rédigé, par exemple :

- *Programmation shell sous Unix/Linux - sh, ksh, bash (exercices pratiques et corrigés)* de Christine DEFFAIX RÉMY ;
- *Shell : Programmation sous Unix/Linux*, Éditions ENI ;
- *Scripts shell, linux et unix* de Christophe Blaess ;
- *Le shell* document rédigé par des tuteurs de l'ENS ;
  `http://www.tuteurs.ens.fr/unix/shell/index.html`
- *Introduction aux scripts shell* de Mathieu Nebra ;
  `https://openclassrooms.com/courses/reprenez-le-controle-a-l-aide-de-linux/introduction-aux-scripts-shell`
- *Introduction à la programmation en Bash* de Eric Sanchis ;
  `ftp://ftp-developpez.com/eric-sanchis/IntroProgBash.pdf`