Commandes principales LINUX
Approfondissement (2)

Objectifs du TP : utiliser un ordinateur sous *Linux*, sans interface graphique, uniquement en ligne de commande (suite). Donc :

- Gérer les processus, notamment leur redirection d'entrée et de sortie, et leur «chaînage»,
- Connaître les commandes *shell* de type "filtre" (*tac*, *head*, *tail*, *sort*, *grep*, *tr*, *cut*,...).

1 Gestion des processus

1.1. Rappel de la notion de processus (process)

Un processus est un programme en cours d'exécution. Il est toujours lancé par un utilisateur (personne physique ou entité "système").

Un processus est identifié par un numéro unique (*Process Id* ou ***PID***). Les *PID* sont attribués par le système. Le premier processus lancé après le démarrage du système *Linux* est le programme *init*, lancé par *root*, qui porte le *PID* 1.

Tous les processus sont lancés à partir d'un processus parent. Par exemple, les commandes ou programmes que vous lancez dans un terminal sont des processus fils d'un processus *bash*, lui-même fils de ce terminal. Le *PPID* (***P**arent **P**rocess **Id***) d'un processus est le *PID* de son processus parent.

1.2. Rappel des commandes de base

1.2.1 lister les processus

La commande ***ps*** (*process status*) : liste les processus en cours d'exécution. Il existe beaucoup d'options, nous allons voir les plus courantes.

– "*ps -ef*" affiche tous les processus de la machine (e=*every*) avec beaucoup d'informations (f=*full*). C'est la commande dont il faut se rappeler.

```
$ ps -ef
```

– "*ps -u isen*" affiche tous les processus lancés par l'utilisateur *isen*.

```
$ ps -u isen
```

La signification des différentes colonnes (format *full*) est la suivante :

- **UID** : nom de l'utilisateur qui a lancé le processus
- **PID** : numéro du processus
- **PPID** : numéro du processus parent
- **C** : facteur de priorité (plus la valeur est grande, plus le processus est prioritaire)
- **STIME** : heure de lancement du processus
- **TTY** : nom du terminal
- **TIME** : durée de traitement du processus
- **CMD** : nom du processus.

- la commande *top* : affiche en temps réel la liste des processus, triés par occupation décroissante de la CPU, mis à jour en permanence (toutes les 3 secondes par défaut). Seuls les 15 à 20 processus les plus actifs sont affichés (d'où le nom de *top*) :

```
$ top
```

Tapez "q" pour sortir de *top*.

- la commande *pstree* : affiche les processus en cours d'exécution sous la forme d'un arbre, ce qui montre leurs liens de filiation (sous Linux, un processus est toujours lancé par un processus parent, le processus *init, de* PID = 1, étant à la racine de l'arbre) :

```
$ pstree
```

1.2.2 arrêter un processus

Un processus qui a été lancé en premier plan depuis un terminal peut être arrêté ("tué") facilement en tapant **<*Ctrl-C*>** dans le terminal.

C'est plus difficile pour un processus s'exécutant en arrière-plan : il faut lui envoyer un signal d'arrêt, à l'aide de la commande *kill*. Il faut pour cela connaître son PID (à l'aide de la commande *ps*).

1.3. Les entrées/sorties d'un processus et les redirections

Chaque processus a par défaut 3 entrées/sorties : *stdin*, *stdout* et *stderr* correspondant à l'entrée standard (N° logique 0), la sortie standard (N° logique 1) et l'erreur standard (N° logique 2).

Il est possible de rediriger chacune de ces entrées/sorties vers des **fichiers** :

- la redirection d'entrée (à partir d'un fichier) se fait avec le symbole **<**
- la redirection de sortie (vers un fichier) avec les symboles **>** (création/eff.) et **>>** (ajout)
- la redirection d'erreur (vers un fichier) avec les symboles **2>** (création/eff.) et **2>>** (ajout)

Attention ! avec **>** et **2>**, le fichier est créé, mais s'il existe déjà il est effacé.

Le tableau ci-dessous vous donne quelques exemples de redirection d'entrée/sortie :

`$ ls > fic`	envoi du résultat de la commande *ls* dans le fichier *fic*
`$ diff -u fic1 fic2 > fdif`	envoi du résultat de la commande *diff -u fic1 fic2* (cf. TP précédent) dans le fichier *fdif*
`$ date >> fic`	ajout du résultat de la commande *date* dans le fichier *fic*
`$ cp *.* /tmp 2>/dev/null`	envoi des erreurs retournées par *cp* dans le fichier *null* (poubelle)
`$ cp *.* /tmp 2>> err.log`	ajout des erreurs retournées par *cp* dans le fichier *err.log* (archivage d'erreur)
`$ patch < fdif`	(suite de la commande *diff -u fic1 fic2* plus haut dans le tableau). La commande *patch* modifie le fichier *fic1* pour prendre en compte les modifications stockées dans *fdif*.

Il est possible de cumuler plusieurs redirections. Ex :

```
$ ls -l /* > resultat.txt 2> erreur.txt
```

1.4. Les tubes (pipe)

Les **tubes** (en anglais "*pipes*", littéralement *tuyaux*) constituent un mécanisme de communication entre processus extrêmement utilisé sous *Unix*/*Linux*.

Un tube, symbolisé par une barre verticale (caractère **|**), permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre. Il s'agit donc d'une **chaîne** de redirection entre processus qui ne passe pas par un fichier, mais par une zone mémoire du système.

Exemples :

- Connaître le nombre de fichiers du répertoire */usr/bin* :

```
$ ls /usr/bin | wc -l
```

- Afficher tous les processus page par page :

```
$ ps -ef | more
```

- Afficher tous les processus, les trier (cf §2.4) par nom de *user* et les afficher page par page :

```
$ ps -ef | sort | more
```

Exercices :

- Donnez une commande (une seule) pour afficher le nombre total de processus qui sont en cours d'exécution sur votre machine. Vérifiez que le nombre trouvé est bien le nombre exact, sinon modifiez un peu la commande.
- Même chose pour compter les processus que vous (utilisateur *isen*) avez lancé.
- Exécutez la commande suivante et expliquez ce qu'elle fait :

  `cp -v xx247 * /tmp 2> yxy`

  Essayez de deviner, sans regarder dedans, ce que contient le fichier `yxy`. Vérifiez si vos suppositions sont bonnes. N'oubliez pas de supprimer les éventuels fichiers mis dans */tmp*.

2 Les commandes *shell* de type "filtre"

- Un filtre est une commande qui **lit des données** sur l'entrée standard (clavier), **effectue des traitements** sur les lignes reçues et **écrit le résultat** sur la sortie standard (écran). Bien sûr les entrées/sorties peuvent être redirigées et/ou enchainées avec des "|". La syntaxe générale d'un filtre est la suivante :

  `filtre [options] [paramètres] [fic_source]`

  Remarque : dans la ligne ci-dessus les crochets (**[** et **]**) indiquent une partie facultative.

  NB : si le filtre veut lire les données dans un fichier d'entrée plutôt qu'au clavier (cas très fréquent) la redirection d'entrée(<) n'est pas obligatoire car il suffit dans ce cas de donner un nom de fichier à la fin de la commande, c'est-à-dire que les deux commandes suivantes sont équivalentes :

  `filtre paramètre < fic_source`

  `filtre paramètre fic_source`

- *Remarque* : dans les TP précédents nous avons déjà rencontré des filtres : *cat*, *more*, *less*, *wc*. Nous n'allons donc pas les redécrire ici.

2.1. La commande tac :

C'est l'équivalent de *cat* mais à l'envers, c'est-à-dire que le texte va apparaître en commençant par la dernière ligne. C'est pratique (chaîné avec *more*) pour visualiser des gros fichiers dont vous vous intéressez surtout à la partie finale. Ex :

- Lire le fichier */etc/passwd* page par page en partant de la fin :

```
$ tac /etc/passwd | more
```

2.2. *La commande head :*

Elle permet de lire le début (par défaut les 10 premières lignes) d'un fichier. La principale option (`-n` suivi du nombre de lignes souhaité) permet d'afficher un nombre de lignes différent de 10. Il existe une forme simplifiée où on omet le `n`. Ex :

- Lire les 10 premières lignes du fichier */etc/passwd* :

```
$ head /etc/passwd
```

- Lire les 5 premières lignes du fichier */etc/passwd* :

```
$ head -n 5 /var/log/messages
$ head -5 /var/log/messages
```

2.3. *La commande tail :*

Elle permet de lire la fin (par défaut les 10 dernières lignes) d'un fichier. Les principales options sont :

- `-n` suivi de N permet d'afficher les N dernières lignes. Il existe une forme simplifiée où on omet le `n`. Par ex. `-20` est équivalent à `-n 20`.
- `-f` laisse tourner *tail* en affichant en temps réel les dernières lignes du fichier (même pendant que le fichier grossit),. Taper <*Control C*> pour arrêter.

Ex :

- Lire les 10 dernières lignes du fichier */etc/passwd* :

```
$ tail /etc/passwd
```

- Lire les 5 dernières lignes du fichier */etc/passwd* :

```
$ tail -n 5 /etc/passwd
$ tail -5 /etc/passwd
```

- Lire en temps réel les dernières lignes du fichier */var/log/secure* :

```
$ tail -f /etc/passwd
```

2.4. *La commande sort :*

sort ("tri" en anglais) permet de trier les lignes d'un fichier selon l'ordre lexicographique (=ordre du dictionnaire). Par défaut, la clé de tri est la ligne entière (c'est-à-dire que tous les caractères de la ligne sont comparés). La forme générale est :

`sort [options générales] [-k n1[,n2]] [fic_source]`

Remarque : dans la ligne ci-dessus les crochets (**[** et **]**) indiquent une partie facultative.

Il faut bien noter que le fichier source lui-même n'est pas trié : comme pour toutes les commandes de type filtre, le résultat est affiché sur la sortie standard. Pour obtenir un fichier trié en sortie, il faut rediriger la sortie vers un fichier, mais **attention!** le fichier de sortie ne doit pas être le même que le fichier source. Si on veut trier le fichier source "sur lui-même" c'est possible en rajoutant l'option `-o nom_fichier`. On a donc deux possibilités :

`sort fic_source > fic_dest` (`fic_source` et `fic_dest` doivent être différents)

`sort -o fic_dest fic_source` (`fic_source` et `fic_dest` peuvent être les mêmes)

Il existe beaucoup d'options générales, dont voici les plus courantes :

`-n` (numérique) : tri numérique (ex : 2 < 11) sinon tri ASCII (2>11). Cette option peut être appliquée à un champ particulier (cf. option `-k` ci-dessous),

`-t c` : utilise le caractère `c` comme **séparateur de champs** dans la ligne (**par défaut, l'espace**),

`-f` : insensible à la casse (ne pas différencier majuscule et minuscule),
`-u` (unique) : élimine les doublons (lignes identiques).

L'option **-k** (*key* = clé de tri) est importante car elle permet de spécifier le(les) champ(s) de tri. -k est suivi d'une ou deux valeurs numériques avec la signification suivante :

- `n,n` : tri selon le n^{ème} champ,
- `n` : tri selon la fin de la ligne à partir du n^{ème} champ,
- `n.m` : tri selon la fin de la ligne à partir m^{ème} caractère du n^{ème} champ,
- `n1,n2` : tri selon les champs à partir du champ `n1` et jusqu'au champ `n2`.

Exemples :

- trie le fichier */etc/passwd* selon le 6ème champ (home directory), avec sortie écran :

```
$ sort -t : -k6,6 /etc/passwd
```

- trie le fichier */etc/password* selon l'UID (3ème champ), avec sortie dans un fichier :

```
$ sort -n -t : -k3,3 /etc/passwd > passwd_trie.txt
```

- trie la liste des fichiers du répertoire */etc* (résultat de la commande *ls -l /etc*) selon la taille des fichiers puis, à égalité de taille (1ère clé de tri), selon leur nom (2ème clé de tri) :

```
$ ls -l /etc | sort -k5,5n -k9
```

2.5. *La commande grep :*

Le nom *grep* est une abréviation de "***g**lobal **r**egular **e**xpression **p**rint*", c'est-à-dire "rechercher globalement l'expression régulière et imprimer". Cette commande permet de rechercher des chaînes de caractères dans un texte. Elle est extrêmement utilisée. En sortie sont affichées uniquement les lignes qui contiennent les chaînes recherchées.

La chaîne recherchée par *grep* est un "motif d'expression régulière". Dans sa forme la plus simple, le motif est la chaîne de caractères précise à rechercher, mais les expressions régulières (*regex*) permettent la définition de motifs très complexes (par exemple, si on cherche une ligne qui commence par b dont le 5ème caractère est un x ou un y et est répété 3 fois, le motif est : `'^b...[xy]\{3\}'`). L'étude des expressions régulières est au-delà de la portée de ce TP, d'autant plus qu'il existe plusieurs normes distinctes (POSIX BRE, POSIX ERE, PCRE,...). Un **encadré page suivante** vous donne cependant quelques bases simples. Par défaut *grep* utilise les expressions POSIX BRE, mais ceci peut être changé par les options.

La forme générale de la commande *grep* est :

`grep [options] motif_regex [fics_source]`

Remarque : dans la ligne ci-dessus les crochets (**[** et **]**) indiquent une partie facultative.

Il existe beaucoup d'options, dont voici les plus courantes :

- `-i` : insensible à la casse (ne pas différencier majuscule et minuscule),
- `-v` (*invert*) : recherche inverse : sélectionne les lignes où le motif n'apparaît pas,
- `-o` : n'affiche que le motif trouvé et non pas toute la ligne,
- `-E` : utilise les expressions POSIX ERE (ou commande *egrep* équivalente à *grep -E*),
- `-P` : utilise les expressions PCRE (expérimental selon *man grep*).

Il est recommandé de mettre systématiquement le motif entre *quotes* (' ou "), pour que les méta-caractères (comme *, ? par ex.) ne soient pas interprétés par le *shell* ("*globbing*",cf. TP1).

Exemples :

- recherche les lignes contenant le mot *isen* dans le fichier */etc/group* :

```
$ grep isen /etc/group
```

- recherche les lignes débutant par le mot *isen* dans le fichier */etc/group* :

```
$ grep '^isen' /etc/group
```

- lister les sous-répertoires (pas les fichiers) du répertoire */etc* :

```
$ ls -l /etc | grep '^d'
```

- recherche des URL dans un fichier *html* lu sur le *Web* (commande *wget -qO-*) :

```
$ wget -qO- "http://www.google.fr" | grep -o 'href="[^"]\+"'
```

Les **expressions régulières** ou rationnelles (*regular expressions*) :

Une expression régulière est un motif ou modèle (*pattern*) qui décrit un ensemble de chaînes de caractères. Les expressions régulières sont construites de façon analogue aux expressions arithmétiques, en utilisant divers opérateurs pour combiner des expressions plus petites. Nous allons présenter ici les symboles utilisés pour les opérateurs les plus importants.

Il existe deux grandes familles pour la symbolisation de ces opérateurs : les symboles POSIX RE (normalisés) et PCRE (*Perl Compatible Regular Expressions*). La famille POSIX est elle-même séparées en deux : BRE (*Basic Regular Expressions*) et ERE (*Extended Regular Expressions*) . Les symboles de base sont cependant communs aux différentes familles : les méta-caractères `.`, `*`, `^`, `$`, `[`, `]`, `{`, `}`, `(`, `)`, `|`, `?` et `+` ont toujours la même signification. La différence majeure entre BRE et ERE est le `\` qui doit être mis en BRE devant les caractères `{`, `}`, `(`, `)`, `|`, `?` et `+` pour qu'ils soient considérés comme des méta-caractères. La famille PCRE est considérablement enrichie par rapport à POSIX.

Les symboles présentés ci-dessous sont les symboles de la norme POSIX BRE utilisés par la version GNU de *grep*.

Signification de quelques symboles de base :

- `.` (le point) : un caractère quelconque
- `[abc]` : un caractère parmi `a`, `b` ou `c` (ceux spécifiés entre crochets)
- `[^abc]` : un caractère qui n'est pas parmi `a`, `b` ou `c` (aucun de ceux spécifiés entre crochets)
- `[a-z]` : un caractère entre `a` et `z` (minuscules)
- `[a-zA-Z]` : un caractère entre `a` et `z` (minuscules ou majuscules)

Signification des symboles de répétition : on répète une expression en plaçant immédiatement après un des opérateurs suivants :

- `*` : se répète un nombre quelconque de fois (0 compris)
- `\+` : se répète au moins une fois
- `\?` : se répète au plus une fois
- `\{n\}` : se répète exactement `n` fois
- `\{n,m\}` : se répète au moins `n` fois et au plus `m` fois

Signification des symboles de position (ou ancres) :

- `^` : début de ligne
- `$` : fin de ligne
- `\<` : début de mot
- `\>` : fin de mot

Signification des symboles de groupement :

- `exp1exp2` : correspond à `exp1` puis `exp2` (concaténation)
- `exp1\|exp2` : correspond à `exp1` ou `exp2` (alternative)
- `\(exp\)` : groupe le contenu de l'expression `exp` et la mémorise
- `\n` : correspond à la n^{ème} expression mémorisée par des parenthèses (référence arrière).

2.6. *La commande tr :*

tr (***tr**anslate character*) permet de convertir ou d'éliminer des caractères dans un texte. **Attention!** Contrairement à la règle générale pour les filtres, la commande *tr* ne peut lire que l'entrée standard : pour travailler sur un fichier, il faut donc une redirection d'entrée.

Remarque importante : *tr* n'utilise pas les expressions régulières, mis à part quelques raccourcis, dont le "`-`" pour spécifier un intervalle, par exemple `'a-z'` pour les minuscules.

La forme générale est :

`tr [options] ensemble1 [ensemble2]`

`ensemble1` et `ensemble2` sont des chaînes de caractères. Habituellement :

– soit `ensemble1` et `ensemble2` sont de même longueur et chaque caractère de `ensemble1` est converti dans le caractère de même rang dans `ensemble2`, Ex : si `ensemble1` vaut `'abc'` et `ensemble2` vaut `'def'`, tous les `a` du texte seront convertis en `d`, les `b` en `e` et les `c` en `f`.

– soit `ensemble2` ne contient qu'un caractère et chaque caractère d'`ensemble1` est converti dans le caractère d'`ensemble2`, Ex : si `ensemble1` vaut `'abc'` et `ensemble2` vaut `'d'`, tous les `a` les `b` et les `c` du texte seront convertis en `d`.

Il existe quelques options, dont les plus utiles sont :

`-d` (*delete*) : détruit dans le texte les caractères listés dans `ensemble1`,

`-s` (*squeeze repeats*) : détruit dans le texte les occurrences multiples des caractères listés dans `ensemble1` (n'en garde qu'un).

Exemples :

- met tous les mots du fichier */etc/passwd* en majuscules, avec sortie écran :


```
$ tr 'a-z' 'A-Z' < /etc/passwd
```

- efface tous les retours à la ligne du fichier */etc/passwd*, avec sortie écran :

```
$ tr -d '\n' < /etc/passwd
```

- remplace tous les `':'` par une tabulation dans le fichier */etc/passwd* , avec sortie écran

```
$ tr ':' '\t' < /etc/passwd
```

- transforme le résultat de la commande `ps -ef` en remplace toutes les séries de `' '` par un seul `' '`

```
$ ps -ef | tr -s ' '
```

2.7. *La commande cut :*

cut ("coupe" en anglais) permet d'extraire certaines parties de chaque ligne d'un texte. Cette extraction peut typiquement être faite :

– par octets (*bytes*) avec l'option `-b`,

– par caractères avec l'option `-c`,

– par champs (*fields*) avec l'option `-f`. Les champs sont séparés par un délimiteur précisé par l'option `-d` (**tabulation par défaut**).

Dans chaque cas, un ou plusieurs intervalles doivent être fournis, spécifiés selon une des façons suivantes, où N et M sont les positions des octets, caractères ou champs, comptés à partir de 1 :

– N : à la position N,

- N-M : entre les positions N et M,
- N- : de la position N à la fin de la ligne,
- -N : du début de la ligne à la position N.

S'il y a plusieurs intervalles, ils doivent être séparés par des virgules.

La forme générale est :

`cut [options] intervalle1[,intervalle2] [fics_source]`

Exemples :

- affiche tous les *users* (1er champ) lus dans le fichier */etc/passwd* :

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

- affiche tous les *users* (1er champ) avec leur *uid* (3ème champ) lus dans le fichier */etc/passwd* :

```
$ cut -d ':' -f 1,3 /etc/passwd
```