
TD n° 3 - Shell

Les objectifs de ce TD sont :

- continuer la découverte des commandes;
- découvrir les variables du **shell**;
- étudier les possibilités de la recherche des mots dans les documents;
- étudier les méta-caractères des filtres;
- étudier l'avant-plan et l'arrière-plan.

Quelques commandes supplémentaires

Dans ce TD, nous utilisons les commandes que nous avons déjà exploré mais on enrichi l'ensemble de nos commandes par quelques nouvelles.

echo affiche le texte passé en argument à l'écran. Si ce texte contient une référence vers une variable d'environnement (`$nom_de_variable`) alors la valeur de la variable est affichée;

history permet de visualiser les dernières commandes exécutées. Le nombre de commandes conservées est dans la variable `HISTSIZE` (`echo $HISTSIZE` permet de connaître la valeur de cette variable);

alias permet de donner un nom à un ensemble de commandes. La syntaxe est `alias nom='commande'`.

tar permet d'archiver des fichiers et répertoires.

gzip permet de compresser des fichiers.

find est une commande très qui permet de parcourir l'arborescence du système à la recherche de fichiers présentant certaines caractéristiques.

```
find chemin -iname "motif" -exec commandes \;
```

cut Cette commande permet d'extraire certains champ d'une ligne.

Une forme qui nous intéresse :

```
cut -d [separateur] -f [champs]
```

Exemple :

```
cut -d @ -f 2,5 toto.txt
```

affiche les champs 2 et 5 de chaque ligne de toto.txt en utilisant @ comme séparateur

basename et dirname En shell, on manipule souvent des chemins et des noms de fichiers. Les commandes **basename** et **dirname** peuvent découper un chemin (l'argument) en deux partie : nom de fichier dans le répertoire et nom absolu du répertoire.

Exemple :

```
$ dirname /home/an1/brie/archi/toto.txt
/home/an1/brie/archi
$ basename /home/an1/brie/archi/toto.txt
toto.txt
```

N'oublions pas que pour sauvegarder la sortie d'une commande dans un fichier, on utilise la redirection
>

Exemple :

```
ls -l > liste.txt
```

sauvegarde les résultats de la commande `ls` dans le fichier `liste.txt`

Exercices

❶ Créez et testez les alias suivants (depuis votre répertoire personnel) :

- `h` pour `history`;
- `li` pour `ls -i`;
- `ll` pour `ls -l`;
- `la` pour `ls -a`.

Attention, dans la définition des alias, il faut "protéger" les caractères spéciaux et les séparateurs contre l'interprétation du shell.

❷ Le fichier caché `.bashrc` est un fichier qui est automatiquement exécuté à chaque démarrage du shell (et donc d'un terminal). Etudiez ce fichier.

❸ Ajoutez y les alias ci-dessus qui ne sont pas encore présents. Essayez les nouveaux alias. Attention, pour que les nouveaux alias soient connus, il faut redémarrer le terminal.

❹ Écrire une commande (ou un ensemble de commandes) qui affiche l'occupation disque du répertoire `/home`.

Filtrez et/ou affichez la ligne qui concerne `/home`.

❺ Créez un fichier `fruits` contenant exactement :

```
pomme:2:rouge:1,35:nzelande
cerise:6:rouge:3,25:france
pomme:3:jaune:2,25:bretagne
peche:4:jaune:1,70:espagne
peche:5:blanche:1,60:france
banane:9:jaune:1,15:guadeloupe
pomme:4:jaune:1,10:france
```

En utilisant la commande `cut`, affichez :

- a. uniquement les noms de fruits
- b. uniquement le nom et la couleur
- c. uniquement le nom, le prix et la provenance

❻

Utiliser la commande `head` pour n'afficher que les 3 premières lignes du fichier `fruits`, puis utiliser la commande `tail` pour n'afficher que les 2 dernières lignes. Utilisez ces deux commandes pour n'afficher que les lignes 3 et 4 du fichier des fruits.

❼ Toujours en utilisant la commande `cut`, donnez la liste des GIDs qui sont présents dans `/etc/passwd`.

❽ Analysez le type des fichiers présents dans `/etc` (Utilisez `ll` ou regardez la documentation de la commande `file`)

❾ Que fait la commande `find . -iname *.txt -exec wc -l {} \;`

❿ Après avoir créé un répertoire `Copie/` dans votre répertoire concernant le Système, testez le résultat de la commande suivante :

```
find -iname *.txt -exec echo cp {} Copie \;
```

Que se passera-t-il si vous supprimez `echo` de la commande ?

⓫ En utilisant `find`, cherchez dans `/etc` les fichiers dont le nom se termine par `.conf`. Pour chaque fichier trouvé, affichez son contenu (utilisez `exec`).

⓬ Faites une archive de votre répertoire personnel et compressez-là.

⓭ Décompresser et extrayez les fichiers de votre archive compressée.

⓮ Que font les commandes suivantes ?

```
mkdir redirection
echo "Bonjour"
echo "Bonjour"> redirection/bonjour.out
echo "Salut" > redirection/bonjour.out
echo "Bonjour" » redirection/bonjour.out
```

15 Dans un répertoire de tests, créez les fichiers suivants :

`prog.c prog.o projet.c projet.o projet.out presentation scene`

Donner les commandes utilisant `ls` les plus courtes possibles pour effectuer les opérations demandées ci-dessous.

- a. Afficher `prog.c` et `prog.o`
- b. Afficher `prog.c` et `projet.c`
- c. Afficher `projet.o` et `projet.out` (mais pas `projet.c` ni les autres)
- d. Afficher `projet.c`, `projet.o`, `projet.out` et `presentation`
- e. Afficher tous les fichiers sauf `presentation`
- f. Afficher `projet.c` et `projet.o` mais pas `projet.out`
- g. Afficher `presentation` et `scene` mais aucun autre
- h. Afficher `scene` seulement
- i. Afficher `projet.out` et `presentation`

Shell et variables

Pour créer une variable, on exécute une commande : `var=valeur`

Pour substituer/utiliser la valeur d'une variable : `$var`

Par exemple, pour afficher la valeur d'une variable : `echo $var`

Variables

16 Exécutez et expliquez les commandes suivantes :

```
message1=Bonjour
echo $message1
```

```
message2=message1
echo $message2
```

```
unset message2
echo $message2
```

```
message3=
echo $message3
```

```
message4 = Bonjour
echo $message4
```

```
bash
echo $message1
```

17 Portée d'une variable

- Créez une variable `varp=exemple`
- Affichez la valeur de `varp`
- Lancez un deuxième shell (en premier plan). Pour tester votre environnement, contrôlez la disponibilité de la variable `varp` dans le nouveau shell
- Quittez le deuxième shell (revenez au premier). Contrôlez la disponibilité de la variable `varp`

On va étudier l'export des variables un peu plus loin.

18 Modification d'une variable

- Créez maintenant la variable `varx` en lisant son contenu du clavier (commande `read`)
- Affichez la valeur de `varx`
- Modifiez la valeur de `varx` en dupliquant son contenu
- Affichez la nouvelle valeur de `varx`

Variables d'environnement

- 19 Affichez les variables d'environnement existantes avec la commande `printenv` (ou encore `set | more` si besoin). Analysez cette liste et essayez de deviner à quoi servent ces variables (suivez vos bonnes intuitions).
- 20 Affichez le contenu de la variable `PATH`. Vous pouvez y observer une liste. Quel est le séparateur des éléments ? Quel est le rôle de cette variable ?
- 21 Créez un répertoire `bin` dans votre répertoire personnel (`HOME`) et copiez-y quelques fichiers exécutables de `/bin` (par exemple : `echo`, `ls`, `pwd`,...) en changeant leur nom, par exemple : contre `monecho`, `monls`, etc. Vérifiez les droits avec `ls -l bin/*`
- 22 Essayez d'exécuter un des programmes déposés dans le répertoire `~/bin` en appelant directement le nom court du programme depuis votre répertoire personnel (par exemple `monpwd` si c'est son nouveau nom dans `~/bin`). Allez dans le répertoire `~/bin` et appelez la commande en tapant son nom (`monpwd`). Appelez maintenant la commande par `./monpwd`. Que peut-on voir et déduire ?
- 23 Ajoutez ce répertoire à la liste dans `PATH`. Essayez d'exécuter un des programmes déposés dans le répertoire `~/bin` en appelant directement le nom du programme depuis un autre répertoire.
- 24 Observez le contenu des variables `HOME`, `PATH`, `OLDPWD`, `PS1` et `PS2`. Quel est le rôle de ces variables ?
- 25 Changez les messages `PS1` et `PS2`.
- 26 Créez une variable d'environnement `PROMO` avec le contenu `' 1/2 '`. Vérifiez `PROMO` dans des shells fils.

Le shell est interactif

Ici, on vérifie les conséquences de cette interactivité.

- 27 Lancez un éditeur `gedit` en arrière plan (`gedit&`). Comment l'utiliser ? Comment le quitter ?
- 28 Lancez un nouveau shell en arrière plan (`bash&`). Comment l'utiliser ? Comment le quitter ?

Quelle est la différence entre les deux types de programme (éditeur avec interface graphique et shell) ?

Substitution des commandes et des variables

- 29 Dans cet exercice, faites attention à la différence entre les délimiteurs `' '` puis `` `` puis `" "`. Exécutez et expliquez ce que font les commandes suivantes :

```
echo 'ls -l'
```

```
echo `ls -l`
```

```
echo "je m'appelle $USER"
```

```
ls `ls`
```

Expliquez les effets des délimiteurs `' '` puis `` `` puis `" "`.

- 30 Positionnez-vous dans un répertoire quelconque dans le système. Affichez la première et la deuxième partie du nom absolu du répertoire où vous êtes.
- 31 On propose de faire une petite dialogue avec le shell. Le scénario envisagé est le suivant :

```
machine > Bonjour, qui es-tu ?
utilisateur > Mathilde
machine > Bonjour Mathilde
```

Proposez et essayez une séquence de commandes pour réaliser ce scénario. (Plus tard, on va voir, comment faire une nouvelle "commande", un script, pour appeler le dialogue.)

Rechercher des motifs dans un fichier, expressions régulières

Nous avons vu les méta-caractères du shell. Attention, ici on va étudier des expressions qui utilisent des méta-caractères différents mais **qui sont interprétés par certaines commandes**.

Une *expression régulière* est une suite de caractères appelée " motif " qui permet la recherche des chaînes de caractères correspondantes au motif à l'aide de commandes telles que `grep` par exemple.

La syntaxe simplifiée de `grep` est : `grep motif nom_fichier`.

De nombreux caractères spéciaux sont utilisés pour décrire les chaînes de caractères possible. Voici une liste des méta-caractères les plus importants :

- `.` : un caractère quelconque
- `x*` : un certain nombre de fois (éventuellement 0) le caractère `x`
- `x+` : au moins une fois le caractère `x` (équivalent à `xx*`)
- `x?` : 0 ou une fois le caractère `x`
- `[abc]` : un caractère parmi ceux indiqués. On peut utiliser des intervalles (par exemple `[a-zA-Z]` désigne une lettre majuscule ou minuscule)
- `[^abc]` : un caractère autre que ceux indiqués
- `x{n}` : exactement `n` copies du caractère `x`
- `x{n,m}` : entre `n` et `m` copies du caractère `x`
- `x{n, }` : au moins `n` copies du caractère `x`
- `^` : début de ligne
- `$` : fin de ligne
- `\<` : début d'un mot
- `\>` : fin d'un mot

Remarque : Pour désigner les caractères qui ont une signification spéciale (`$`, `*`, etc.) il faut les *échapper* en ajoutant `' \ '` devant (le caractère `' \ '` est donc également spécial et pour le désigner il faut écrire `" \ \"`).

Pour les questions suivantes, utilisez le fichier `TEXTE.txt`.

- 32 Trouvez les lignes contenant `bla`.
- 33 Trouvez les mots se terminant par `ques`.
- 34 Trouvez des lignes vides.
- 35 Trouvez les lignes commençant par une voyelle.
- 36 Trouvez les lignes commençant par une voyelle ou par un chiffre.
- 37 Trouvez les lignes commençant par une voyelle et se terminant par le caractère `?`
- 38 Trouvez les lignes ne commençant pas par une voyelle.

`grep` peut être utilisé dans des tubes.

- 39 Recherchez tous les fichiers du répertoire `/etc` qui appartient à `root` (leur ligne dans la liste étendue des fichiers contient la chaîne de caractères `" root "`).
- 40 Recherchez les fichiers du répertoire `/etc` qui ne peuvent pas être exécutés par les "autres" utilisateurs.
- 41 Recherchez les fichiers du répertoire `/etc` qui ne peuvent pas être exécutés par personnes.