Initiation à l'environnement Unix : TP n° 2

septembre 2024 — Pierre Rousselin

1 Éditer un fichier texte

Exercice 1 : Éditer le fichier de configuration du shell

Le fichier `.bashrc` contenu dans votre répertoire personnel est le fichier de configuration du shell `bash`. Chaque fois qu'il est lancé, `bash` commence par lire ce fichier et exécuter les commandes qu'il contient.

1. Pourquoi le fichier `.bashrc` n'apparaît-il pas lorsqu'on tape la commande `ls ~` ? Comment faire pour qu'il soit listé ?
2. Faire une copie, nommée `bashrc_copie`, du fichier `.bashrc` pour éviter les accidents.
3. Avec un éditeur de texte dans le terminal (`vim`, `emacs -nw` ou au pire `nano`), éditer `.bashrc` et inscrire à la fin du fichier la commande

   ```
   echo Salut génie !
   ```

   Ne pas oublier d'enregistrer les modifications puis quitter.
4. Quitter le terminal et en ouvrir un nouveau.
5. Bon, c'est amusant mais finalement ça n'aide pas beaucoup. Effacez (avec un éditeur de texte) la commande que vous avez entrée dans votre `.bashrc`.
6. Testez les commandes suivantes

   ```
   $ ls -larth --color=auto
   $ alias ll='ls -larth --color=auto'
   $ ll
   $ alias ri='rm -i'
   $ touch a_supprimer
   $ ri a_supprimer
   ```

   et dites à quoi servent les alias !
7. Ajoutez ces alias dans votre `bashrc`, fermez le terminal, ouvrez-en un nouveau et constatez que vous pouvez utiliser ces alias. Essayez de penser à d'autres alias qui pourraient vous faire gagner du temps.
8. Si vous êtes sûr de ne pas avoir fait de bêtise, vous pouvez supprimer `bashrc_copie`.

--- * ---

Exercice 2 : Les variables du shell

1. Entrer les commandes suivantes et commenter :

   ```
   $ nom_fich=hello.c
   $ echo $nom_fich
   $ echo ${nom_fich}
   $ touch $nom_fich
   $ echo $nom_fichc
   $ echo ${nom_fich}c
   $ rm ${nom_fich}
   ```
2. Que se passe-t-il lorsqu'on demande au shell de développer une variable qui n'est pas définie ?
3. Que se passe-t-il si l'on met un espace entre le nom de la variable et le signe = ? Et entre le signe = et la valeur qu'on lui affecte ? Essayer, par exemple avec les commandes

   ```
   $ a = truc
   $ b= truc
   ```
4. Exécuter le code suivant et commenter (les guillemets anglais `"` seront abordés plus tard).

```
$ sujet=Alice verbe=aime cod=piscine
$ phrase="$sujet $verbe la $cod."
$ echo $phrase
$ sujet=Bob verbe=mange cod=salade
$ echo $phrase
$ echo "$sujet $verbe la $cod."
```

--- * ---

Correction de l'exercice 2 :

1. L'affectation se fait au moyen du signe égal, le développement avec le symbole `$` qui précède le nom de la variable. On peut également utiliser la syntaxe `${...}` qui permet de concaténer le développement de la variable avec des caractères admis dans les noms de variable (ces caractères sont comme en C, les chiffres et lettres majuscules et minuscules et le souligné, avec la contrainte de ne pas commencer par un chiffre).
2. Le résultat du développement d'une variable non définie est une chaîne vide.
3. On aura une « commande » inconnue sur le premier mot ou le deuxième. Il ne doit pas y avoir d'espace d'un côté ou de l'autre du signe `=`.
4. Au moment de l'affectation de phrase, les variables `sujet`, `verbe` et `cod` sont développées, donc la variable `phrase` ne change pas de valeur, même si ces premières en changent.
   Autre remarque : on peut affecter plusieurs variables sur la même ligne (ce n'est pas forcément utile tout de suite mais ça le sera lorsqu'on voudra changer l'environnement d'une commande).

--- * ---

Exercice 3 : Variables du shell et bashrc

1. Créer dans votre répertoire personnel le répertoire `prog_rust/a_rendre/projet/version_beta`.
2. Dans le `.bashrc`, à la fin du fichier, écrire la commande permettant de mettre dans une variable (par exemple `rep_proj`) un chemin absolu vers ce répertoire.
3. Ouvrir un nouveau terminal et, en utilisant votre variable, se déplacer dans ce répertoire.
4. La variable `EDITOR`, lorsqu'elle est définie, est utilisée par divers programmes ou commandes pour ouvrir votre éditeur préféré. Dans le `.bashrc`, affecter à `EDITOR` le nom de votre éditeur favori.
5. Lancer un nouveau terminal ou bien sourcer (faire lire) le `.bashrc` avec la commande
   ```
   $ . ~/.bashrc
   ```
6. La commande `fc` pour *fix command*, c'est-à-dire réparer la commande, ouvre un éditeur de texte où la dernière commande est déjà entrée, afin de pouvoir la corriger. Lorsqu'on quitte l'éditeur, la commande est lancée par le shell. Essayer
   ```
   $ cal 13 2023
   $ fc
   ```
   et corriger la commande.

--- * ---

Exercice 4 : Premier script shell

Un script shell n'est rien d'autre qu'une succession de commandes qui va être interprétée par un shell.

1. Avec un éditeur de texte, écrire le script suivant que vous appellerez `bonjour`.
   ```
   #!/bin/sh
   # bonjour : donne des informations à l'utilisateur et lui dit bonjour.

   echo Bonjour, votre login est :
   logname
   echo Cette machine a pour nom :
   uname -n
   ```

La première ligne s'appelle un *shebang* et nous en reparlerons plus tard, contentez-vous pour l'instant de retenir que tous vos scripts shell doivent commencer par cette ligne.

2. Ensuite, on va rendre ce script exécutable, opération à ne faire qu'une seule fois, *ce n'est pas du tout une compilation* :

```
$ chmod u+x bonjour
```

On reparlera des permissions une autre fois, pour l'instant on se concentre sur les scripts.

3. Enfin exécution :

```
$ ./bonjour
```

Pourquoi on doit taper `./` ? Réponse aussi dans un prochain numéro.

4. Complétez ce script pour qu'il affiche également les informations suivantes :

a) Le nom de l'Unix utilisé (souvenez-vous du TP précédent).

b) La date et l'heure.

c) Le calendrier du mois en cours.

5. Noter et retenir les étapes nécessaires pour écrire et utiliser un script shell.

--- * ---

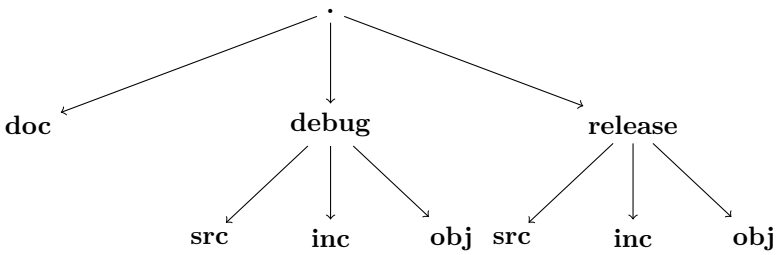
Correction de l'exercice 4 :

```
#!/bin/sh
# bonjour : donne des informations à l'utilisateur et lui dit bonjour.
echo Bonjour, votre login est :
logname
echo Cette machine a pour nom :
uname -n
echo Cet Unix a pour nom :
uname -s
echo Nous somme le :
date
echo Calendrier du mois :
cal
```

--- * ---

Exercice 5 : Création d'une arborescence

Même des scripts en apparence un peu idiots peuvent être très utiles. Par exemple, une association peut décider d'utiliser pour tous ses projets en C l'arborescence minimale suivante :



Écrire un script shell `arbo_projet` qui crée **dans le répertoire courant** cette arborescence, c'est-à-dire les répertoires **doc**, **debug** et **release** et leurs éventuels sous-répertoires. Vous pouvez vous contenter d'écrire dans le script les commandes que vous utiliseriez pour créer cette arborescence dans le terminal.

--- * ---

Correction de l'exercice 5 :

```
#!/bin/sh
# arbo_projet : crée une arborescence pour un projet en C

mkdir doc debug release
cd debug
mkdir src inc obj
cd ../release
mkdir src inc obj
```

--- * ---

Certains caractères ont une signification particulière pour le shell : on dit qu'ils sont « spéciaux ». À l'inverse, on dit d'un caractère qui n'a pas d'autre signification que lui-même, qu'il a son sens « littéral ». Nous listons ci-dessous les caractères spéciaux ; la plupart d'entre eux seront vus en détail plus tard dans le cours.

`; <newline> | &` Ils mettent fin à la commande qui les précède. On a utilisé **`<newline>`** pour représenter le caractère « nouvelle ligne » qu'on saisit avec la touche ↵. Le caractère spécial **`|`** est saisi avec la combinaison de touches Alt Gr+6, on l'appelle *pipe* (tube).

`< >` Les chevrons. Ils permettent les redirections.

`( )` Pour regrouper des commandes et les lancer dans un sous-shell.

`$` Pour le développement de variables, le développement arithmétique et la substitution de commande.

`` ` `` L'accent grave (en anglais, *backtick* ou *backquote*) pour la substitution de commandes (ancienne syntaxe). Il est saisi au clavier avec la combinaison de touches Alt Gr+7.

`<space> <tab>` L'espace et la tabulation délimitent les noms de commandes et arguments.

`\ ' "` La contre-oblique (aussi appelée *backslash*, *antislash*), l'apostrophe (en anglais *single quote*) et le guillemet anglais (en anglais *double quote*) qui permettent justement d'inhiber les caractères spéciaux, c'est-à-dire leur rendre leur sens littéral.

Enfin, les caractères suivants ont une signification particulière dans certains contextes et doivent donc parfois être inhibés :

`* ? [` Pour le développement de noms de chemins.

`#` Pour écrire des commentaires (sauf s'il est au milieu d'un mot).

`~` Pour le développement du tilde (répertoire personnel).

`=` Pour l'affectation de variables.

`%` Pour certains développements de variables.

Exercice 6 : L'inhibition des caractères spéciaux : la contre-oblique

1. Tester les commandes suivantes :

a) `echo a   b`
b) `echo a\ \ \ b`
c) `touch fichier\ vide`
d) `rm fichier vide`
e) `rm fichier\ vide`
f) `echo 3$canadiens`
g) `echo 3\$canadiens`
h) `echo ; echo *`
i) `echo \; echo \*`
j) `echo "salut"`

k) `echo \"salut\"`
l) `echo 'salut'`
m) `echo \'salut\'`
n)
```
echo Salut \
> toi
```
o) `echo \\`

2. Au vu des expériences précédentes (et d'autres à inventer si nécessaire), répondre aux questions suivantes :

a) Que fait le caractère `\` au caractère (autre que `<newline>`) qu'il précède ?
b) À quoi sert la chaîne `\<newline>` ?
c) Comment obtenir une contre-oblique `\` littérale ? Comment afficher `\\` à l'aide de la commande `echo` ?

--- * ---

Correction de l'exercice 6 :

1. a) Deux arguments : `a` et `b`.
b) Un seul argument car les 3 blancs ont été inhibés.
c) Un seul argument (blanc inhibé).
d) Deux arguments, la commande échoue.
e) Un seul argument.
f) La variable non définie `$canadiens` est développée en la chaîne vide.
g) Le caractère `$` est inhibé donc littéral.
h) Deux commandes, le `;` a son rôle spécial de terminateur de commande (on dit aussi *opérateur de contrôle*) et l'astérisque a son rôle spécial de caractère joker. On aura deux lignes affichées, la première est vide, la seconde contient la liste des fichiers du répertoire courant.
i) Une seule commande `echo` avec trois arguments : un `;`, le mot `echo` et un caractère `*`.
j) Les guillemets anglais sont spéciaux, ils inhibent certains caractères spéciaux dans `salut` (en l'occurence il n'y en a pas) puis sont enlevés par le shell.
k) Ils sont inhibés, donc traités comme n'importe quel caractère et apparaissent dans la chaîne affichée.
l) Idem avec l'apostrophe.
m) Idem avec l'apostrophe.
n) Un prompt secondaire apparaît, on doit appuyer sur entrée pour terminer la commande : on a en fait une commande sur deux lignes qui est un echo avec un seul argument.
o) La contre-oblique retrouve son sens littéral.

2. Le caractère `\` préserve le sens littéral du prochain caractère (sauf `<newline>`) puis est enlevé par le shell.

3. La séquence `\<newline>` permet d'écrire une commande sur plusieurs lignes. Ces deux caractères sont supprimés de la commande par le shell.

4. La séquence `\\` est interprétée comme une contre-oblique littérale. Pour afficher `\\` avec echo, on peut entrer la commande `echo \\\\`.

--- * ---

Exercice 7 : L'inhibition des caractères spéciaux : l'apostrophe

1. Tester les commandes suivantes :

a) `touch 'ceci est un horrible nom de fichier'` (puis voir le contenu du répertoire courant)
b) `rm -i ceci est un horrible nom de fichier`

c) `rm -i 'ceci est un horrible nom de fichier'`

d) `touch a b cd; echo le caractère * est-il spécial ? et ?`

e) `echo 'le caractère * est-il spécial ? et ?'`

f) Dans la commande suivante, le prompt secondaire > apparaît car la chaîne entre apostrophes n'est pas terminée.

```
$ echo 'en fait, même la fin de ligne
> est un caractère normal entre
> apostrophes'
```

2. Au vu des expériences précédentes (et d'autres à inventer si nécessaire), répondre aux questions suivantes :

a) Quels sont les caractères qui sont spéciaux entre apostrophes ?

b) Comment obtenir une apostrophe dans une chaîne entre apostrophes (question piège) ?

c) Comment, avec une combinaison de chaînes entre apostrophes et d'une inhibition par contre-oblique, obtenir avec `echo` l'affichage suivant ?

```
Un développement de variable (comme $var) peut s'inhiber ;
par exemple entre apostrophes.
```

--- * ---

Correction de l'exercice 7 : Correction globale : Dans une chaîne entre apostrophes, tous les caractères sont littéraux, y compris les espaces et les fins de ligne, sauf l'apostrophe elle-même qui met fin à l'inhibition. On ne peut donc pas mettre d'apostrophe à l'intérieur d'une chaîne entre apostrophes.

Si besoin, on peut toujours utiliser la séquence `'\''` pour mettre fin à la chaîne entre apostrophe, taper une apostrophe inhibée et recommencer une chaîne entre apostrophes, comme dans la dernière question :

```
$ echo 'Un développement de variable (comme $var) peut s'\''inhiber ;
> par exemple entre apostrophes.'
```

--- * ---

Exercice 8 : Cadeau

La vie est dure en ce moment, alors on vous fait un petit cadeau : la correction de l'examen d'Unix que vous aurez à la fin du semestre. Elle est dans une archive à l'url
`https://www.math.univ-paris13.fr/~rousselin/unix/cadeau.tgz`
et voici comment la télécharger et l'extraire en ligne de commande :

```
$ wget https://www.math.univ-paris13.fr/~rousselin/unix/cadeau.tgz
$ tar xvfz cadeau.tgz
```

L'archive ne sert plus à rien, on la supprime :

```
$ rm cadeau.tgz
```

1. Ah ah ! On vous a bien eu ! Maintenant vous avez plein de fichiers avec des noms horribles dans un nouveau répertoire `cadeau`. Combien y a-t-il de fichiers dans ce répertoire ? Et lesquels ont un caractère spécial dans leur nom ? Remarquez que GNU `ls` met le nom d'un fichier entre apostrophes s'il comporte un caractère spécial (avec un traitement spécial pour l'apostrophe et la fin de ligne). Exceptionnellement (si c'est possible) voir le contenu du répertoire cadeau dans un explorateur de fichiers graphique.

2. La commande suivante affiche les noms des fichiers du répertoire courant et numérote les lignes de l'affichage.

```
$ printf '%s\n' * | nl
```

Essayer cette commande (ce n'est pas grave de ne pas la comprendre maintenant) et dire s'il y a autant de lignes que de fichiers contenus dans le répertoire cadeau.

3. Supprimer tous les fichiers du répertoire un par un, à l'exception de `correction_exam2024.txt` en inhibant les caractères spéciaux. Attention aux fichiers `-i` et `--`, cette fois c'est la commande `rm` qui pose problème. La solution est dans le manuel (`man rm`).
4. Comment créer dans `cadeau` un sous-répertoire qui s'appelle ~ (le caractère tilde) ? Comment s'y déplacer ? Essayez et vérifiez avec `pwd` que vous êtes bien dans le bon répertoire.
5. *À ne faire que si vous êtes absolument sûr de vous !*
   Supprimer le sous-répertoire `~` de `cadeau` **avec la commande `rmdir`**.
6. Est-ce que vous pouvez créer un fichier qui s'appellerait littéralement `et/ou.txt` dans le répertoire cadeau ? À votre avis, pourquoi ?
7. À votre avis :
   a) Quels sont les caractères autorisés dans les noms de fichiers Unix ? Indices : penser au langage C et à la façon dont se terminent les chaînes de caractères.
   b) Qu'est-ce qu'un « bon » nom de fichier ? Quels caractères, bien qu'autorisés, devraient absolument être évités ?
8. Voir le contenu de `correction_exam2024.txt` avec une commande, puis supprimer le répertoire cadeau et tout ce qu'il contient.

--- * ---

Correction de l'exercice 8 :

1. Il y a 16 fichiers (si on ne compte pas `.` et `..` qui sont cachés de toute façon). Ils ont tous des noms plus ou moins abominables sauf `correction_examen2024.txt`. La palme revient sans doute à celui qui a des sauts de ligne et celui qui s'appelle `<space>`. Pro tip pour connaître le nombre de fichiers (on y reviendra plus tard) :
   ```
   $ set -- *; echo $#
   ```
2. Comme un des fichiers a un nom qui fait trois lignes, il y a deux lignes de plus que le nombre de fichiers.
3. Il faut, par exemple, utiliser l'option `--` de rm pour pouvoir supprimer `-i` et `--`. Ensuite on a le choix entre l'inhibition par contre-oblique et l'inhibition entre apostrophes, sauf pour
   — le fichier `'` (contre-oblique obligatoire)
   — le fichier dont le nom occupe 3 lignes (apostrophes obligatoires).
   ```
   $ rm -- -- -i ' ' \" \& \' \( \* \[ \\ \$SHELL ah\ ah '>lol' '<rofl' 'sur
   > plusieurs
   > lignes'
   ```
4. `mkdir '~'` et ensuite `cd '~'`. Sans inhibition, on arriverait dans son répertoire personnel.
5. `rm -r '~'` mais ça fait trop peur...
6. Impossible, si on pouvait créer ce genre de fichier, on ne pourrait pas faire la différence entre celui-ci et le fichier `ou.txt` du répertoire `et`. Donc le caractère `/` est interdit dans les noms de fichier.
7. Tous les caractères sont autorisés sauf `/` et le caractère nul (`'\0'` en C, le caractère de code ascii 0). Décision contestable ! On aurait au moins aimé qu'ils interdisent les fins de ligne, ça aurait réglé un nombre incalculable de problèmes. En contrepartie, on peut utiliser tous les alphabets du monde pour les noms de fichiers, surtout maintenant avec l'utf-8.
8. C'est vraiment bien d'éviter les caractères spéciaux dans les noms de fichier, au moins les fins de lignes ! C'est mieux (mais certainement pas aussi important) d'éviter les espaces, par exemple en les remplaçant par des blancs soulignés (`_`). Aussi c'est très embêtant s'ils commencent par un tiret. Mais dans les scripts sûrs, on doit prendre en compte tous les cas...

--- * ---