NUMP1 : TP Unix - 2

Rémi Giraud
remi.giraud@enseirb-matmeca.fr

2026-2027

1 Rappels

1.1 L'interpréteur de commandes

Le shell est un interpréteur de commandes qui :
— initialise l'environnement, notamment en exécutant le script ∼/`.bashrc` (voir la seconde partie du TP)
— génère le prompt.
Quand une commande est validée, le shell :

1. effectue les substitutions de variables,
2. interprète les métacaractères,
3. gère les redirections et les pipes,
4. effectue les substitutions de commandes,
5. exécute la commande.

C'est le mécanisme d'évaluation du shell. **Ce n'est pas ce qui est saisi qui sera exécuté mais le résultat de l'évaluation de l'expression.**

1.2 Commandes

Une commande est constituée d'un nom suivi d'arguments.

```
$ commande [options] [arguments]
```

On différencie deux types de commandes :
— Externes au shell : ce sont des exécutables placés dans /bin, /sbin, /usr/bin, /user/sbin, etc. Elles sont exécutées dans un sous-processus. **Comme les exécutables peuvent se trouver dans différents répertoires, le shell doit savoir où les chercher. C'est la variable "PATH" qui définit la liste des répertoires à scruter et l'ordre dans lequel doit faire sa recherche.**
— Internes au shell : elles font partie intégrante de l'interpréteur de commandes, elles ne sont pas exécutées dans un processus fils.

`Remarque:` Toutes les commandes Unix distinguent les majuscules des minuscules. Par exemple "ls", "Ls", "lS" et "LS" sont quatre mots différents au niveau de l'interpréteur de commandes.

2 Surveillance et gestion des processus sous UNIX

La gestion des processus est une compétence essentielle sous UNIX. Cette section présente les commandes principales pour observer, filtrer et contrôler les processus via la ligne de commande.

2.1 `grep` et les pipes (`|`)

— **La commande** `grep` permet de rechercher un motif (texte, expression régulière) dans la sortie d'une commande ou dans un fichier.
— **Le pipe** (`|`) redirige la sortie standard d'une commande vers l'entrée d'une autre.
Exemples : `ls -l | grep .txt` Affiche uniquement les fichiers contenant .txt.

2.2 `ps` : visualiser les processus

`ps` affiche les processus en cours d'exécution.

Options principales :
— `ps` : affiche les processus du terminal courant.
— `ps -ef` : format complet (affiche UID, PID, PPID, etc.).
— `ps aux` : format BSD détaillé, souvent utilisé avec `grep`.
Exemple pratique : trouver le PID d'un processus `ps aux | grep "nginx"`
Permet d'identifier le PID (Process ID) associé au processus `nginx`.

2.3 `top` et `htop` : surveillance en temps réel

`top` affiche en temps réel les processus actifs, leur consommation CPU, mémoire, etc.
`htop` (version améliorée et interactive) permet :
— un tri dynamique par utilisation CPU/RAM,
— la recherche et le filtrage,
— la terminaison directe d'un processus (via F9).

2.4 `kill` et `pkill` : terminer des processus

`kill` envoie un signal à un processus via son PID : `kill [signal] <PID>`
`pkill` permet d'envoyer un signal à tous les processus correspondant à un nom : `pkill <nom>`

Signaux courants :
— `SIGTERM (15)` : arrêt gracieux (par défaut).
— `SIGKILL (9)` : arrêt immédiat (forcé).
— `SIGHUP (1)` : redémarrage ou relecture de configuration.

Exemples :

```
kill 2481
kill -9 2481
pkill firefox
```

Question 1 Identifier le PID du processus `firefox` et le terminer proprement.

2.5 Bonnes pratiques

— Toujours tenter d'arrêter un processus avec `SIGTERM` avant `SIGKILL`.
— Utiliser `pgrep <nom>` pour obtenir directement le PID sans `grep`.
— Vérifier l'arrêt d'un processus avec : `ps -p <PID>`
— Surveiller régulièrement les processus gourmands avec `top` ou `htop`.

3 Exécution d'une commande

3.1 Notion de variable

Sous votre shell, il est possible de définir des variables. Par exemple sous votre environnement est défini une variable `DISPLAY` contenant le nom de votre terminal.

Question 2 Exécutez les commandes `echo DISPLAY` et `echo $DISPLAY`. Qu'en déduisez-vous ? Comment accède-t-on au contenu d'une variable ?
Question 3 Affichez le contenu de la variable `FOO`.
Question 4 Exécutez la commande `FOO=DISPLAY`. Affichez le contenu de la variable `FOO`.
Question 5 Exécutez la commande `FOO=$DISPLAY`. Affichez le contenu de la variable `FOO`.

3.2 Recherche et lancement d'une commande

Question 6 Affichez le contenu de la variable `PATH`. Sauvegardez cette valeur dans une variable `PATHSVG` et vérifier la valeur sauvegardée. Mettre la chaîne vide dans `PATH` et essayez les commandes `cd`, `pwd`, `ls`. Que peut-on en conclure ? Comment lancer `ls` sans modifier `PATH` ?

Modifiez la variable `HOME` en lui affectant le chemin absolu vers l'un de vos répertoires (autre que votre répertoire personnel). Observez ce que produit la commande `cd` sans argument. **Restaurez les valeurs de `PATH` et de `HOME`.**

4 Exécution d'une commande

4.1 Valeur de retour d'une commande

Dans les TP précédents, les variables `PATH`, `DISPLAY` tout comme `FOO` ont en fait été définies dans votre environnement, et vous pouvez en modifier leur valeur. Mais il existe quelques variables particulières affectées par le shell lui même. L'une d'entre elles se nomme `?`. Celle-ci contient le code de retour du dernier programme exécuté.

Question 7 Testez sur les commandes suivantes :

```
$ [ 2 = 2 ] ; echo $?
$ [ 2 = 3 ] ; echo $?
$ test 2 = 2 ; echo $?
$ test 2 = 3 ; echo $?
```

4.2 Action conditionnelle

Question 8 Dans votre fenêtre terminal, exécutez la commande `ls -l && echo OK || echo KO`. Puis exécutez la commande `ls -z && echo OK || echo KO`. Quel est le rôle des opérateurs `&&` et `||` ? Examinez dans chacun des deux cas précédents, le contenu de la variable `?`. Expliquez son contenu.

Question 9 Réécrire les commandes de l'exercice précédent en utilisant la structure de contrôle `If` en une ligne de commande.

Syntaxe : `if` *command* `; then` *action1* `; else` *action2* `; fi`

5 Mécanismes de substitutions

!!!!Attention!!!!
Le caractère backquote ‘ (au clavier touches altGR +7) est différent du caractère quote ’ (au clavier touche 4)

Interprétez et commentez le résultat des commandes suivantes.

1. Substitution des accolades
   ```
   $ echo a{ab,c,edf}f
   $ mkdir ~/Textes/a{1,2,5}
   ```
2. Substitution du caractère tilde
   ```
   $ echo ~/bar
   $ echo ~test
   ```
3. Substitution de variables
   ```
   $ ab=totot
   $ echo $ab
   $ echo ${ab}ababab
   $ echo $abababab
   $ y=p ; ${y}wd
   ```
4. Substitution de commandes
   ```
   $ echo ‘ls -l‘
   $ echo $(ls ~/Textes)
   ```
5. Substitution arithmétique
   ```
   $␣echo␣p$((␣3␣+␣4␣))q
   ```
6. Substitutions des métacaractères
   ```
   $ cd
   $ echo */*
   $ echo ../?[r-u]*
   $ echo ../?[x-z]?
   $ echo ../?[^x-z]*?
   ```
7. Substitutions et exécution de commandes
   ```
   $ x=p; y=$(${x}wd); echo cwd=$y
   $ x=p; y=‘${x}wd‘; echo cwd=$y
   ```

5.1 Quotations

1. Déduire le rôle des quotes et doubles quotes
   ```
   $ x=p; y=’${x}wd’; echo $y
   $ echo "*/*" *
   $ echo "* $(pwd)"
   ```
2. Expliquer le comportement des commandes
   ```
   $ A=$(echo *)
   $ echo ’$A’
   $ echo "$A"
   $ echo $A
   $ $A
   $ "$A"
   ```

Question 10 Faites afficher les chaînes suivantes :

`***␣BONJOUR␣***`
`␣-␣␣␣Le␣caractere␣'*'`
`/Fichier␣d'entree\`
`*␣Mon␣Jan␣20␣17:59:48␣EET␣1992␣*` (la date doit être le résultat d'une exécution de `date`)
`$$␣Mon␣Jan␣20␣18:07:15␣EET␣1992␣$$` (idem)
`$6248$` (où le nombre entre `$` est le *pid* du shell)
`(␣/bin/cat␣/bin/tar␣)` c'est-à-dire la liste de fichiers dans le répertoire `/bin` dont le nom est à trois lettres et dont la deuxième lettre est une de lettres a–d (utiliser les métacaractères).

Question 11

Dans votre répertoire courant, créez à l'aide d'une seule commande les fichiers suivants : "fich1, fich2, fich11, fich12, fich1a, ficha1, fich33, .fich1, .fich2, toto, afich". Donnez les commandes permettant de lister les fichiers :

1. dont les noms commencent par fich,
2. dont les noms commencent par fich suivi d'un seul caractère,
3. dont les noms commencent par fich suivi d'un chiffre,
4. dont les noms commencent par .,
5. dont les noms ne commencent pas par f,
6. dont les noms contiennent fich,
7. dont les noms ne contiennent pas le chiffre 2 et la lettre a. (voir `grep`)

6 Les script shells et mécanismes de substitution

Un fichier script contient une suite de commandes shell. Pour exécuter le script, deux méthodes sont possibles. Soit vous tapez le nom du shell suivi du nom du script à interpréter :

```
$ bash mon-script
```

soit vous tapez directement le nom du script :

```
$ ./mon-script
```

Dans ce dernier cas, le fichier script doit posséder la permission en exécution et pour garantir son exécution par un shell de type `sh`, la première ligne de ce fichier doit indiquer le chemin absolu du shell qui sera utilisé. La syntaxe de cette ligne est la suivante :

```
#!/bin/sh
```

Si cette ligne est omise, c'est le shell de type `bash` qui est utilisé par défaut. Dans tous les cas, prenez la bonne habitude d'insérer systématiquement cette ligne particulière dans vos scripts shells.

Question 12 Créez un fichier exécutable `cmd.sh` contenant :

```
#!/bin/sh
ps -f
```

Dans votre fenêtre `terminal`, lancez la commande `cmd.sh`. Si tout va bien, vous devriez voir s'afficher la liste de vos processus et en particulier un processus (à peu de chose près) de la forme `/bin/sh cmd.sh`. Modifiez la ligne `#!/bin/sh` afin d'utiliser un autre shell que `sh` et réexaminez le résultat produit par l'exécution de la commande `cmd.sh`.

Question 13 Ecrire un petit script qui affiche "Bonjour *login*, il est 16 heures" (où login est votre nom de login, et 16 est l'heure actuelle. Bien évidemment, vous utiliserez la notion de variable).

6.1 Les variables prédéfinies

Le shell offre des variables prédéfinies facilitant la programmation sous le shell. Vous avez déjà pu observer le rôle des variables `?` ou `$`. Cherchez dans le manuel de `bash` le rôle des variables `#`, `*`, `@`, `0`, `1`, `2`, `3`, etc.

Question 14 Insérez dans un script shell, une commande permettant d'afficher le nom absolu du script lancé (chemin depuis la racine + nom du script). Pour en vérifier le bon fonctionnement, renommez votre fichier en `autre_nom.sh` et exécutez le.

Question 15 Insérez une commande affichant le nombre de paramètres du fichier de commandes de la manière suivante :

$$nomvar = \text{'}nb_param\text{'}$$

où *nomvar* est le nom de la variable contenant cette information et *nb_param* son contenu. Vérifiez que cela fonctionne en lançant le fichier de commandes avec plusieurs paramètres.

Question 16 De la même façon, insérez une commande affichant la liste des paramètres passés en argument.

6.2 Itérations

Question 17 Testez la boucle `for` au moyen des programmes suivants :

```
for N in un deux trois quatre
do
  echo "**$N**"
done
```

```
for N in $(ls)
do
  echo "Fichier -> $N"
done
```

Question 18 Ecrire un script listant chacun de ses paramètres en utilisant une boucle `while` avec la commande `shift` ou `for`. On appelera ce script `liste.sh`. Testez la avec la commande suivante `./list.sh "a b" 'c' d e`

6.3 Substitution de commandes et de variables

Question 19 Examinez les scripts qui suivent,

petit-script-4 :

```
#!/bin/sh
date
echo Il est $4\\
```

petit-script-5 :

```
#!/bin/sh
set ‘date‘
echo Il est $4\\
```

et exécutez les commandes :
`$ petit-script-4 donald riri fifi loulou picsou`
`$ petit-script-5 donald riri fifi loulou picsou`
afin d'interpréter le rôle de la commande : `set` ≺ *commande* ≻.

Question 20 Écrire un script qui affiche l'heure sous la forme : `Il est 17:45:26.`
Consultez le manuel de `bash` afin d'examiner le rôle de la variable IFS.

Question 21 En modifiant la variable `IFS`, écrire un script qui affiche : `Il est 17h 45mn.`

Question 22 Écrire un script `prepa` prenant en paramètre un nom de fichier : ≺*nom*≻ qui crée dans le répertoire courant un fichier exécutable ≺*nom*≻ dont la première ligne est `#!/bin/bash` suivi d'une ligne vide.

Question 23 Écrire, sans utiliser de structure de contrôle, les commandes suivantes :

1. `nf` : affiche le nombre de fichiers ne commençant pas par un point, du répertoire courant [1] ;
2. `ra` : affiche `oui` si le répertoire courant est le répertoire d'accueil et `non` sinon [2] ;
3. `prc` : affiche la profondeur du répertoire courant [3] ;

Question 24 Écrire le script `ifdef` tel que l'exécution de `ifdef` ≺ *nom* ≻`.h` place en tête de ≺ *nom* ≻`.h` les lignes

```
\#ifndef \meta{NOM}\_H\\
\#define \meta{NOM}\_H\\
```

et la ligne

```
\#endif /* \meta{NOM}\_H */
```

à la fin. On pourra utiliser la commande `tr(1)` pour les conversions de minuscules en majuscules.

7 BONUS : Enchaînement des substitutions

Question 25 Expliquez le comportement de chacune des lignes de commandes suivantes :

```
A='echo *'
$A
echo $A
echo '$A'
echo "$A"
```

1. Vous utiliserez les commandes `echo`, `wc` et `pwd`.
2. Vous utiliserez les commandes `test`, `pwd`, `echo` et les opérateurs `&&` et `||`.
3. Vous utiliserez les commandes `set`, `pwd`, `shift`, `echo` ainsi que la variable `IFS`.

De même avec cette fois ci le **remplacement de commandes** : `A=‘echo *‘`

Question 26 En observant le comportement des lignes de commandes suivantes, expliquez ce que fait `eval`.

```
CMD=echo
ARG='$$'
LIG='$CMD [$ARG]'
$LIG
eval $LIG
eval eval $LIG
eval eval eval $LIG
```

8 BONUS : Personnalisation de Bash

A chaque lancement d'un nouveau shell Bash, le script ∼`/.bashrc` est lancé. Celui-ci permet de configurer l'environnement Bash.

Lisez le fichier ∼`/.bashrc`.

Vous verrez qu'il exécute les scripts `.bash_alias` et `.bash_export`.

8.1 Alias

Il est possible de définir des **alias** sous Unix permettant de personnaliser vos appels de commandes les plus utilisées. Les commandes à utiliser ici sont `alias` et `unalias`. Lancer `man alias`. Attention, ces commandes ne sont valables seulement pour le shell courant. Si vous voulez que vos alias soient permanents, insérez les dans `.bashrc` ou `.bash_alias`.

Question 27 Affichez la liste des alias définis dans le shell courant.

Question 28 Définissez un alias dont le nom est `date` sur la commande `ls`.

Question 29 Détruisez l'alias précédent et rappelez la commande `date` afin d'en vérifier son bon fonctionnement.

Question 30 Ouvrez le fichier `.bash_alias` se trouvant dans votre répertoire d'accueil. Rajoutez les alias suivants à l'intérieur de ce fichier, à la suite des alias déjà existant :

```
alias rm='rm -i'
alias ll='ls -la'
```

Question 31 Ajoutez également un alias permettant d'avoir le man en anglais directement.

Pour que ces modifications soient prises en compte par bash dès maintenant, tapez la commande `source .bash_alias`.

8.2 Variables d'environnement

Les systèmes Unix utilisent des variables réservées pour la configuration de l'environnement.

Pour modifier une variable d'environnement, utilisez la syntaxe suivante : `export VAR=valeur`. La variable `PATH` donne la liste des répertoires où aller chercher les commandes. Vous pouvez voir son contenu avec la commande : `echo $PATH`

Si vous voulez, par exemple, que les fichiers qui sont dans le répertoire `bin` de votre répertoire d'accueil soient directement accessibles, ajoutez le chemin dans la variable PATH comme suit : `export PATH=$PATH:$HOME/bin`

Vous pouvez personnaliser l'affichage du prompt (en y indiquant le chemin du dossier courant par exemple) en éditant la valeur PS1 dans votre `.profile` ou `.bashrc` (ou `.bash_profile`). Un prompt secondaire Prompt String 2 (PS2) est utilisé lorsque la commande en cours a besoin de plusieurs lignes.

Question 32

- Exécutez `cat abc \`. A quoi sert le `\` après une commande ?
- Éditez ou ajoutez la ligne suivante dans votre `.bashrc`
  `PS2="Anything else to write?"`
- Exécutez `cat abc \`

9 BONUS : Environnement

Question 33 Observez et expliquez la suite de commandes :

```
echo $A
A=aaaaa
echo $A
bash
echo $A
exit
export A
bash
echo $A
A=bbbbb
echo $A
exit
echo $A
```

Question 34 Que fait le script suivant :

```
#!/bin/sh

echo " --> $$"
```

Testez ce script et comparez avec le résultat affiché en entrant directement la commande `echo $$` dans votre bash. Expliquez ce qui se produit à l'exécution. Essayez en préfixant la commande par `.` ou par `source`. Que peut-on en conclure ?

Question 35 Que fait le script suivant :

```
#!/bin/sh

if [ $# = 0 ]
then
    cd ..
```

```
else
    cd $1
fi
echo " --> `pwd`"
```

Testez cette commande et expliquez ce qui se produit à l'exécution. Essayez en préfixant la commande par `.` ou par `source`. Que peut-on en conclure ?

Question 36 Faites un script qui range les fichiers dans une arborescence du type année/mois/-jour à partir de la date de modification.