Aide-mémoire — Commandes et scripts Bash

Avant-propos. Toutes options ne sont pas indiquées. Pour plus de détails, veuillez vous référencer aux pages de manuel (cf la commande man). Une commande *interne* est une commande fournie par le shell bash, il faut alors regarder la page de manuel du shell.
Dans la suite, <fich> est le chemin (absolu ou relatif) d'un fichier, <rep> est le chemin (absolu ou relatif) d'un répertoire.

1 Commandes de gestion des fichiers et répertoires

- ls : **Lis**te le contenu d'un répertoire. Si aucun argument n'est donné, c'est le contenu du répertoire courant qui est affiché. Sinon, c'est le contenu des répertoires indiqués en paramètres qui est listé.
  Options :
  · -a : liste également les fichiers et répertoires cachés (*i.e.,* dont le nom commence par un point).
  · -l : liste en plus les attibuts des fichiers.
  · -h : avec -l donne les tailles des fichiers sous forme plus lisible.

- cd (**C**hange **D**irectory) : **C**hange le **d**ossier/répertoire courant. Commande interne.
  Exemples :
  · cd <rep> : se déplace dans le répertoire indiqué en paramètre.
  · cd : se déplace dans le répertoire personnel (/)
  · cd .. : remonte dans le répertoire supérieur/parent.
  · cd - : se déplace dans le dernier répertoire visité.

- mkdir <rep1> <rep2> ... (**Ma**k**e** **Dir**ectory) : crée les répertoires indiqués en paramètre (au moins un). Les répertoires pères de <rep1>, <rep2> ...doivent déjà exister.
  Option :
  · -p : si les répertoires pères n'existent pas, ils sont également créés.

- rmdir <rep1> <rep2> ... (**Re**mo**v**e **Dir**ectory) : Supprime les répertoires indiqués (au moins 1). Les répertoires doivent être vides.

- rm <fich1> <fich2> ... (**Rem**ove) : supprime les fichiers passés en paramètres. **ATTENTION** : aucun moyen de les récupérer ensuite. **Options** :
  · -i : demande confirmation avant chaque effacement.
  · -f : ne demande jamais de confirmation
  · -r : effacement récursif. **Ex :** rm -r <rep1> <rep2> ... permet d'effacer les répertoires indiqués et tout ce qu'ils contiennent.

- cp (**Cop**y) : **Cop**ie de fichiers et de répertoires.
  Utilisation :
  · cp <fich1> <fich2> : crée un nouveau fichier de chemin <fich2> et copie dedans le contenu de <fich1>. Si <fich2> existait déjà, il est écrasé.
  · cp <fich1> <fich2> ...<rep> : copie dans le répertoire <rep> les fichiers indiqués en paramètres. <rep> doit déjà exister.
  · cp -r <rep1> <rep2> : si <rep2> existe, alors copie récursivement dedans le répertoire <rep1> et tout son contenu. Sinon, crée le répertoire <rep2> et copie dedans tout le contenu de <rep1>.

- mv (**Mov**e) : Déplacer/renommer des fichiers et des répertoires.
  Utilisation :
  · mv <fich1> <fich2> : déplace le fichier <fich1> pour que son chemin devienne <fich2>.
  · mv <fich1_ou_rep1> <fich2_ou_rep2> ...<rep> : déplace, dans le répertoire <rep>, les fichiers ou répertoires indiqués en paramètre. <rep> doit exister.
  · mv <rep1> <rep2> : si le répertoire <rep2> existe, alors déplace <rep1> dedans. Sinon, déplace le répertoire <rep1> pour que son chemin devienne <rep2>.

- `ln <fich1> <fich2>` (**Lin**k) : Crée un **lien** physique du fichier `<fich1>` **vers** `<fich2>`.
  Option :
  · `-s` : crée un lien symbolique au lieu d'un lien physique. On peut créer un lien symbolique d'un répertoire vers un autre.

- `touch <fich1> <fich2> ...` : Crée des fichiers vides. Si les fichiers existent déjà, alors leur date de dernière modification est mise à la date courante.

- `tar`, `zip`, `unzip` : Créer une archive ou extraire des fichiers d'une archive (voir les pages du `man`).

- `gzip <fich>`, `gunzip <fich>` : Compresser ou décompresser un fichier.

- `diff <fich1> <fich2>` : affiche les différences de lignes entre les argements. Fonctionne également pour les répertoires.

2 Commandes sur les fichiers

Pour toutes les commandes suivantes :
↪ **si aucun chemin de fichier n'est donné en paramètre, la commande lit son entrée standard (`stdin`) ;**
↪ **le résultat de la commande est afficher sur sa sortie standard (`stdout`).**

- `cat <fich1> <fich2> ...` (**Cat**enate) : Affiche le contenu du (ou des) fichiers les uns à la suite des autres.

- `wc <fich>` (**W**ord **C**ount) : Compte le nombre de lignes, mots et caractères d'un texte.
  Options :
  · `-l` (**L**ine) : le nombre de lignes.
  · `-c` (**C**haracter) : le nombre de caractères.
  · `-w` (**W**ord) : le nombre de mots.

- `head -n <nb> <fich>` : Extraire les `<nb>` premières lignes.

- `tail -n <nb> <fich>` : Extraire les `<nb>` dernières lignes. Si `<nb>` est de la forme $+n$, alors extraire à partir de la n-ième ligne.

- `grep <motif> <fich>` (**G**lobal **R**egular **E**xpression **P**rint) : Afficher les lignes contenant le `<motif>`.
  Options :
  · `-c` : afficher le nombre de lignes contenant le `<motif>`.
  · `-n` : afficher en plus le numéro de la ligne.
  · `-v` : afficher les lignes qui ne contiennent pas le `<motif>`.

- `cut <colonnes> <fich>` : Extrait certaines parties dans chaque ligne.
  Options :
  · `-c` : indique la ou les positions des parties à extraire.
  · `-f` : indique un numéro de champ.
  · `-d` : indique un caractère délimiteur de champ.
  Exemples :
  · `cut -c5-15,33-37-` : Extraire dans chaque ligne les caractères 5 à 15, le caractères 33 et les caractères de 37 jusqu'à la fin de la ligne.
  · `cut -d'',''  -f3-5` : Extraire les champs 3 à 5 de chaque ligne en utilisant le caractère “,” comme délimiteur de champ.

- `tr <liste1> <liste2>` (**Tr**ansform), où `<liste1>` et `<liste2>` sont des listes de caractères : Remplace les caractères de `<liste1>` par le caractère à la même position dans `<liste2>`. Cette commande lit sur l'entrée standard `stdin` et envoie le résultat sur la sortie standard `stdout`.
  Options :
  · `tr -d <liste>` : Supprime de `stdin` tous les caractères de `<liste>`.
  · `tr -s <liste>` : Supprime de `stdout` toutes les répétitions des caractères de `<liste>`.

  N.B. : Les listes de caractères peuvent se définir en les écrivant entre guillemets ou en utilisant des listes prédéfinies (voir la page de `man`).

- `sort <fich>` : Trie les lignes par ordre alphabétique croissant.
  Options :
  · `-r` : tri décroissant.
  · `-n` : suppose que les lignes commencent par un nombre, trie en utilisant la valeur de ce nombre.

  `uniq <fich>` : Supprime les multiples occurrences consécutives d'une même ligne, pour n'en garder qu'une seule
  Options :
  · `-c` : affiche en plus le nombre d'occurence de chaque ligne.
  · `-u` : affiche seulement les lignes n'apparaissant qu'une seule fois.
  · `-d` : Affiche seulement les lignes répétées.

- `more` : Affiche le contenu d'un fichier page par page. La touche espace [ ] permet de passer à la page suivante La touche [q] permet de quitter.

- `less` : Affiche le contenu d'un fichier page par page. Les touches [↑] et [↓] permettent de se déplacer dans le texte . La touche [/] permet de rentrer au clavier une chaîne à rechercher dans le texte et les touches [n] et [N] permettent de se déplacer sur les différentes occurences de la chaîne. La touche [q] permet de quitter.

- `which <fich>` : Localise la commande `fich`.

- `file <fich>` : Donne le type du fichier.

- `strings <fich>` : Affiche les chaînes de caractères affichables contenues dans `<fich>` (principalement utilisé pour récupérer les chaînes cpntenues dans les fichiers non-ASCII).

- `nm <fich>` (**N**a**m**es) : Affiche la table des symboles de `<fich>`(si c'est un fichier objet ou exécutable).

- `od <fich>` (**O**ctal **D**ump) : Affiche le contenu du fichier en octal (par défaut) ou avec d'autres codages (hexadécimal, ASCII,...). Utile pour visualiser des fichiers binaires.

3 Aide

- `man <section> <commande>` : affiche la page du **man**uel de la commande `<commande>`. Le paramètre `<section>` est facultatif et permet de spécifier la section du manuel où recherche la commande.

- `apropos <mot>` : recherche une page de manuel contenant le modt dans sa description résumée.

- `info` : présente les pages d'infos qui sont en général plus détaillées et plus lisibles que les pages du manuel. Ces pages sont structurées en arbre. Le simple est de les visualiser dans `emacs` : dans `emacs` taper [Ctrl]-[h] [i] . La plupart des commande présentées dans ce document sont accessible dans la section *CoreUtils*.

4 Gestion des permissions

- chmod <mode> <fich1_ou_rep1> <fich2_ou_rep2> ... (**Ch**ange **mod**e) : Modifie les permissions d'accès de chacun des fichiers et répertoires indiqués, en suivant l'indication donne par <mode>. Ce <mode> est :
  · soit un nombre octal de 3 chiffres représentant les nouvelles permissions. Le tableau ci-dessous indique la correspondance des chiffres avec les droits.

Table 1 – Correspondances de représentation des droits

Droit	Valeur alphanumérique	Valeur octale
aucun droit	—	0
exécution seulement	–x	1
écriture seulement	-w-	2
écriture et exécution	-wx	3
lecture seulement	r–	4
lecture et exécution	r-x	5
lecture et écriture	rw-	6
tous les droits (lecture, écriture et exécution)	rwx	7

  · soit une représentation symbolique du changement à effectuer, de la forme CSP où :
    - C est une suite de lettres indiquant à quelle(s) catégorie(s) d'utilisateurs s'applique les modifications de droits. Les choix possibles sont :
      · 'u' *(user)* pour le propriétaire ;
      · 'g' *(group)* pour le groupe d'utilisateurs ;
      · 'o' *(other)* pour les autres utilisateurs ;
      · 'a' *(all)* pour tous les utilisateurs.
    - S peut prendre comme valeur :
      · '+' pour ajouter des droits ;
      · '-' pour enlever des droits.
    - P est une suite de lettres indiquant quels sont les droits modifiés :
      · 'r' *(read)* pour la lecture ;
      · 'w' *(write)* pour l'écriture ;
      · 'x' *(execute)* pour l'execution.

  Exemples :
  · chmod 744 toto (mode numérique)
  · chmod ug+rw titi (mode symbolique) : il faut rajouter des droits (+) en lecture et écriture (rw) au propriétaire et au groupe (ug).

5 Opérations sur les chemins

Les opérations suivantes sont principalement utiles dans les scripts.

- basename <chemin> : affiche le nom seul du fichier indiqué par <chemin>.
  Exemple : basename /truc/top/hop.txt affiche hop.txt.

- dirname <chemin> : affiche le nom des répertoires.
  Exemple : dirname /truc/top/hop.txt affiche /truc/top.

- realink -f <chemin> : affiche le chemin absolu correspondant au chemin <chemin>.

6 Commandes sur les processus

- `ps` (**P**rocessus **S**tatus) : affiche des informations sur les processus en cours d'exécution.
  Exemples :
  · `ps x` : tous les processus de l'utilisateur ;
  · `ps ax` : tous les processus de tous les utilisateur.

- `pstree` : affiche l'arbre des processus.

- `top` : affiche, en vue temps réel, les processus actuellement dans ls système, avec des informations sur l'utilisateur de la mémoire, du processeur, etc. Cette vue est actualisée périodiquement.
  · [h] : affiche l'aide de `top`.
  · [s] : modifie la période de rafraichissment (par défaut : 3 s).
  · [u] : affiche seulement les processus d'un utilisateur particulier.
  · [k] : envoie un signal à un processus (comme la commande `kill`).
  · [r] : change le nice d'un processus.

- `kill <PID>` : tue le processus de PID indiqué (pour utiliser le PID d'un processus, utiliser `top` ou `ps`).
  Options :
  · `-s <signal>` : envoie le signal `<signal>` au processus au lieu de le tuer ;
  · `-l` : affiche la liste des signaux disponibles.

- `killall <prog>` : tue tous les processus de nom "prog".
  Option :
  · `-s <signal>` : envoie le signal `<signal>` aux processus au lieu de les tuer.

- `nice +<valeur> <commande>` : lance la commande `<commande>` avec un niveau de nice égal à `<valeur>`.
  Exemple : `nice +15 emacs`

7 Autres commandes

- `time <commande>` : exécute la `<commande>` et affiche le temps utilisée par celle-ci.

- `date` : affiche la date et l'heure.

- `bc` (**B**asic **C**alculator) : calculatrice.

- `find <rep> <expression>` : recherche les fichiers ou les repertoires satisfaisant `<expression>` dans l'arborescence de racine `<rep>`.
  Options (voir la page de manuel pour d'autres options) :
  · `-name` : recherche par nom de fichier ;
  · `-type` : recherche par type de fichier ;
  · `-size` : recherche par taille de fichier.
  Exemples :
  · `find toto -name "hop"` : recherche les fichiers ou répertoires de nom `hop` dans l'arborescence de racine `toto` ;
  · `find . -name "*.txt"` : recherche les fichiers ou répertoires dont le nom se fini par `.txt` dans l'arborescence du répertoire courant ;
  · `find . -type f` : recherche les fichiers dans la'rborescence de racine le répertoire courant.
  On peut évidemment combiner les critères.

- `du` (**D**isk **U**sage) : affiche la taille (en ko) de tous les répertoires et sous-répertoires du répertoire courant. On peut l'utiliser dans un tube avec `sort` pour tier les résultats : `du | sort -n`.

8 Commandes internes

Les commandes internes sont fournies par le shell bash lui-même. Si vous voulez plus d'informations sur ces commandes il faut donc ilire la page de `man` de bash. Certaines de ces commandes se sont réellement utiles que dans un script.

- `echo <chaine>` : affiche la `<chaine>` sur la sortie standart suivi d'un retour à la ligne.
  Option :
  · `-n` : pas de retour à la ligne.

- `read <variable>` : **voir section 9.1.** Permet de mettre ce qui est lu sur l'entrée standard (via `echo`) dans la variable `<variable>`.

- `shift <n>` : décale les paramètres de `<n>` positions vers la gauche (`<n>` est facultatif, il vaut 1 par défaut).
  Exemple : Soit `$1="un"`, `$2="six"`, `$3="toto"` et `$4="hop"`. Après la commande `shift`, on a `$1="6"`, `$2="toto"`, `$3="hop"` et `$4` est vide.
- `exit <n>` : termine un script avec comme code de retour `<n>`.
- `time <commande>` : exécute la `<commande>` et affiche le temps utilisée par celle-ci.

9 Scripts bash

Un script `bash` commence toujours par la ligne :

```
#!/usr/bin/env bash
```

Les scripts peuvent comporter des commentaires qui sont introduits par le caratére `#`.

9.1 Variables

Les variables (ou les variables d'environement) du shell sont de type *chaine de caractères* ou *entier*. On peut leur affecter :

- Des valeurs constantes :
  ```
  a="bonjour"
  b=34
  ```
- Le contenu d'une autre variable ou variable d'environnement :
  ```
  aa="$a $USER, ca va bien ?"
  ```
- Le résultat d'un calcul avec `$(( ))` (attention, le shell ne gére QUE les entiers) :
  ```
  c=$((324 * 432 - $b ))
  ```
- le résultat d'une commande avec `$( )` :
  ```
  bb=$(ls *.txt)
  bb=$(ls *.txt | wc -l)
  ```
  Par exemple on peut récupérer le contenu d'un fichier comme suit :
  ```
  bb=$(cat nom_fichier)
  ```
- ce qui est lu sur l'entrée standard en utilisant `read` :
  ```
  echo "age ?"; read n; echo "vous avez $n ans."
  ```
  (met ce qui est lu sur l'entrée standard dans la variable `n`)

ATTENTION : Il ne faut pas mettre d'espaces avant et après le signe =.

9.1.1 Variables spéciales (dans un script)

- `$0` : nom de la commande
- `$1`, `$2`, `$3`,... : paramètres
- `"$@"` ou `$*` : liste de tous les paramètres à partir de `$1`
- `$#` : nombre de paramètres (sans compter `$0`)

9.1.2 Guillemets

- Le guillemet simple ' dénote une chaîne dans laquelle **aucune** substitution de variables n'est faite.
- Le guillemet double " dénote une chaîne dans laquelle **toutes** les substitutions de variables sont faites.

Exemple :

```
morvant:~/Documents> ch1="Bonjour $USER"
morvant:~/Documents> ch2='BONJOUR $USER'
morvant:~/Documents> echo $ch1
Bonjour morvant
morvant:~/Documents> echo $ch2
Bonjour $USER
```

9.1.3 Portée des variables

Une variable est locale à son interpréteur. Pour la rendre visible aux processus fils de l'interpréteur qui l'a définie, il faut l'**exporter** comme suit.

```
  export ma_variable
```

9.2 Conditionnelle

La syntaxe des conditionnelles est :

```
if <commande>
then
    <liste de commandes1>
else
    <liste de commandes2>  # facultatif
fi;
```

ATTENTION : Si on met sur la même ligne les mots-clés `if`, `then` et/ou `fi` il faut mettre un point-virgule ; séparateur.

```
if <commande> ; then <liste de commandes1> ;else <liste de commandes2>; fi
```

Si <commande> renvoie un code de retour égal à 0 alors <liste de commande1> est exécutée sinon <liste de commande2> (s'il y a un `else`). <commande> peut être un test (*c.f.* section 9.5).

Exemple :

```
if grep -q voiture fich; then echo "c'est vrai"; else echo "c'est faux"; fi
```

9.3 Boucle `while`

La syntaxe des boucles *tant que* est :

```
while <commande>
do
  <liste de commandes>
done
```

ATTENTION : Si on met sur la même ligne les mots-clés `while`, `do` et/ou `done` il faut mettre un point-virgule ; séparateur.

```
while <commande> ; do <liste de commandes>; done
```

Tant que la valeur de retour de la commande <commande> vaut 0, <liste de commandes> est exécutée. <commande> peut être un test (*c.f.* section 9.5).

Exemple :

```
a=0;
while (( a < 10 )); do a=$(( a + 1 )); echo $a; done;
```

9.4 Boucle for

La syntaxe des boucles *pour* est :

```
for <nom> in <liste>
do
  <liste de commandes>
done
```

ATTENTION : Si on met sur la même ligne les mots-clés for, do et/ou done il faut mettre un point-virgule ; séparateur.

```
for <nom> in <liste>; do <liste de commandes> ; done
```

<nom> est un nom de variable et <liste> est une liste de mots. La <liste de commandes> est exécutée une fois pour chaque mot de la <liste> en affectant à chaque fois ce mot à la variable <nom>.

Exemple :

```
for aa in toto titi tata hop 32; do
  echo "bonjour $aa";
done;
```

9.5 Tests dans les conditionnelles ou les boucles

Certaines commandes spéciales du shell permettent de faire des tests.

9.5.1 Tests arithmétiques (( ))

Les tests arithmétiques doivent être écrits entre des paires de doubles parenthèses ((<test>)) (voir le man de bash section : CALCUL ARITHMÉTIQUE).
Les tests possibles sont : == , != , <, >, <=, >=.
Les tests peuvent être combinés avec &&, ! (négation) et || et utiliser des parenthèses.

ATTENTION : espaces obligatoires après ((, avant)) et entre les opérateurs.

Exemples :

```
(( 20 > 100 ))
(( $b > 3 ))
(( (3*$b == 57 || $a > 3) && ! ($a > 23) ))
```

9.5.2 Tests sur des chaînes de caractères et des fichiers [[ ]]

Les tests sur les chaînes de caractères et les fichiers doivent être écrits entre des paires de doubles crochets [[<test>]] (voir le man de bash section : EXPRESSIONS CONDITIONNELLES).
Les tests possibles sont :

- [[-e <nom>]] vrai si <nom> existe (fichier, répertoire ou lien)
- [[-f <nom_fich>]] vrai si <nom_fich> existe et est un fichier ou un lien sur un fichier
- [[-d <nom_rep>]] vrai si <nom_rep> existe et est un répertoire ou un lien sur un répertoire
- [[-h <nom_lien>]] vrai si le lien symbolique de nom <nom_lien> existe
- [[-r <nom>]] vrai si nom existe et est accessible en lecture
- [[-z $c]] vrai si la variable c est vide
- [[-n $c]] vrai si la variable c est non vide
- [[$c == "toto"]] test d'égalité
- [[$a > "toto"]] vrai si $a est après toto dans l'ordre alphabétique.
  En particulier [[20 > 100]] est vrai !

Les tests peuvent être combinés avec &&, ||, ! et utiliser les parenthèses.

ATTENTION : espaces obligatoires après [[, avant]] et entre les opérateurs.

9.6 Fonctions

Il est tout à fait possible de définir des fonctions en bash. Une fois définie, elle s'utilise comme une commande normale. La syntaxe est :

```
function <nom fonction> () {
<liste de commandes>
}
```

La liste d'arguments est toujours vide !

L'utilisation de `return <n>` à l'intérieur d'une fonction permet d'en sortir. L'entier `<n>` est la valeur de retour de la fonction.

Une fonction peut prendre des paramètres en les référençant dans le corps de la fonction de la même façon que dans un script, c'est-à-dire avec les notations `$1`, `$2`, `$3`, ...

Exemple :

```
function affiche(){
   echo $1
}

affiche "Bonjour"
```

N.B. : Ce mémo est inspiré de celui proposé par B. Jeudy et P. Ezequel.