Mémo Unix

1 Les commandes

La commande `a2ps`

```
bash $ a2ps -Pimprimante tp1.adb
```

Imprime le fichier `tp1.adb` sur l'imprimante `imprimante`. Ce fichier texte est converti en postscript avant d'être envoyé à l'imprimante.

La commande `bg`

Voir section 6.

La commande `cat`

```
bash $ cat toto
```

Affiche le contenu du fichier `toto` à l'écran.

La commande `cd`

```
bash $ cd ..
```

Effectue un déplacement dans le répertoire père.

```
bash $ cd /
```

Effectue un déplacement dans le répertoire racine.

```
bash $ cd ../rep/rep1
```

Effectue un déplacement dans le répertoire `../rep/rep1` (chemin relatif).

```
bash $ cd /home/albert/rep/rep1
```

Effectue un déplacement dans le répertoire `/home/albert/rep/rep1` (chemin absolu).

La commande `chmod`

```
bash $ chmod g+w toto
```

Ajoute le droit d'écriture au groupe du fichier `toto`.

```
bash $ chmod u-r toto
```

Retire le droit de lecture au propriétaire du fichier `toto`.

```
bash $ chmod o-x toto
```

Retire le droit d'exécution aux "autres" utilisateurs (ni le propriétaire, ni les membres de groupe) du fichier `toto`.

```
bash $ chmod 700 toto
```

Positionne les droits `rwx------` sur le fichier `toto`.

```
bash $ chmod g+w rep
```

Ajoute le droit d'écrire dans le répertoire `rep` au membres du groupe de ce répertoire (ce qui signifie le droit de créer et effacer des fichiers).

```
bash $ chmod u-r rep
```

Retire le droit de lecture au propriétaire du répertoire `rep` (ce qui signifie interdiction de "voir" ce qu'il contient)

```
bash $ chmod o-x rep
```

Retire le droit d'exécution aux "autres" utilisateurs (ni le propriétaire, ni les membres de groupe) du répertoire `rep` (ce qui signifie interdiction de le traverser).

```
bash $ chmod -R g+w rep
```

Ajoute le droit d'écriture au groupe de façon récursive

La commande `clear`

```
bash $ clear
```

Efface l'écran du terminal.

La commande `cp`

```
bash $ cp toto titi
```

Effectue la copie du fichier `toto` dans le répertoire courant ; la copie s'appelle `titi`.

```
bash $ cp toto rep
```

Effectue la copie du fichier `toto` dans le répertoire `rep`.

```
bash $ cp -R  rep1 rep
```

Si `rep` existe déjà : copie `rep1` et son contenu dans `rep` ; un répertoire `rep1` est donc créé dans `rep`.

Si `rep` n'existe pas : copie `rep1` et son contenu dans le répertoire courant ; la copie s'appelle `rep`

La commande `date`

```
bash $ date
Fri Feb  4 09:01:45 MET 2005
```

Affiche la date et l'heure du jour. Exemple d'exécution fourni.

La commande `diff`

```
bash $ diff toto titi
```

Indique si les fichiers `toto` et `titi` sont identiques et affiche toutes les lignes où les fichiers sont différents.

La commande `du`

```
bash $ du -sk rep
```

Indique la taille en koctets du répertoire `rep` (pas de détails).

```
bash $ du -k rep
```

Indique la taille en koctets du répertoire `rep` (des détails sont affichés).

La commande `echo`

```
bash $ echo "oulala"
```

Affiche à l'écran la chaîne de caractères `oulala`.

La commande `exit`

```
bash $ exit
```

Termine le shell dans lequel cette commande est tapée.

La commande `export`

Voir section 4.

La commande `file`

```
bash $ file toto
```

Indique le type du fichier (texte, image, binaire, ...).

La commande `find`

```
bash $ find rep -name toto
```

Cherche récursivement dans le répertoire `rep` un fichier dont le nom est `toto` (ne s'arrête **PAS** à la première occurrence).

```
bash $ find rep -name "*toto*"
```

Cherche récursivement dans le répertoire `rep` un fichier dont le nom contient la chaînes de caractères "`toto`" (ne s'arrête **PAS** à la première occurrence).

```
bash $ find rep -name toto -size +0
```

Cherche récursivement dans le répertoire `rep` un fichier dont le nom est `toto` et dont la taille est supérieure à 0 (ne s'arrête **PAS** à la première occurrence).

La commande `fg`

Voir section 6.

La commande `grep`

```
bash $ grep chaine toto
```

Cherche toutes les occurrences de la chaîne de caractères `chaine` dans le fichier `toto` et affiche toutes les lignes où cette chaîne est trouvée.

La commande `gzip/gunzip`

```
bash $ gzip fichier
```

Compresse le fichier `fichier`, ce qui créé un fichier `fichier.gz`.

```
bash $ gunzip fichier.gz
```

Décompresse le fichier `fichier.gz` qui a été compressé avec `gzip`, ce qui créé un fichier `fichier`.

La commande `head`

```
bash $ head -5 toto
```

Affiche les 5 premières lignes du fichier `toto`

La commande `id`

```
bash $ id
uid=100(raymond) gid=100(users) groups=100(users)
```

Affiche l'`uid` (*user identificator*) et le `gid` (*group identificator*) de l'utilisateur. Exemple d'exécution fourni.

La commande `jobs`

Voir la section 6.

La commande `kill`

Voir la section 6.

La commande `ls`

```
bash $ ls rep
```

Affiche le contenu du répertoire `rep`.

```
bash $ ls -l rep
```

Affiche de façon détaillée le contenu du répertoire `rep`.

```
bash $ ls -a rep
```

Affiche le contenu du répertoire `rep` ainsi que les fichiers cachés de ce répertoire.

```
bash $ ls -R rep
```

Affiche récursivement le contenu du répertoire `rep`.

La commande `man`

```
bash $ man ls
```

Affiche la documentation page par page de la commande `ls`.

La commande `mkdir`

```
bash $ mkdir rep
```

Crée le répertoirte `rep`.

La commande `more`

```
bash $ more toto
```

Affiche page par page le contenu du fichier `toto`.

La commande `mv`

```
bash $ mv toto titi
```

Change le nom du fichier `toto` qui devient `titi`.

```
bash $ mv toto rep
```

Déplace le fichier `toto` dans le répertoire `rep`.

```
bash $ mv toto rep/titi
```

Déplace le fichier `toto` dans le répertoire `rep` et le renomme `titi`.

```
bash $ mv rep1 rep2
```

Si `rep2` n'existe pas : renomme le répertoire `rep1` qui devient `rep2`.
Si `rep2` existe : déplace le répertoire `rep1` dans le répertoire `rep2`.

La commande `printenv`

Voir la section 4.

La commande `ps`

Voir la section 6.

La commande `pwd`

```
bash $ pwd
/home/raymond
```

Affiche le répertoire courant. Exemple d'exécution fourni.

La commande `rm`

```
bash $ rm toto
```

Efface le fichier `toto`.

```
bash $ rm -i toto
```

Efface le fichier `toto` et demande confirmation.

```
bash $ rm -r rep
```

Efface récursivement le répertoire `rep` **et tout ce qu'il contient**.

La commande `rmdir`

```
bash $ rmdir rep
```

Efface le répertoire `rep` **seulement s'il est vide**.

La commande `sed`

```
bash $ sed '2,4!d' fichier
```

Affiche à l'écran les lignes 2 à 4 du fichier `fichier`

```
bash $ sed 's/aaa/bb/g' fichier
```

Affiche à l'écran le contenu du fichier `fichier` en remplaçant toutes les occurences de 'aaa' par 'bb'.

La commande `set`

Voir la section 4.

La commande `tail`

```
bash $ tail -5 toto
```

Affiche les 5 dernières lignes du fichier `toto`.

La commande `tar`

```
bash $ tar cvf toto.tar rep
```

Créé une archive `toto.tar` du répertoire `rep`.

```
bash $ tar tvf toto.tar
```

Affiche le contenu de l'archive `toto.tar`.

```
bash $ tar xvf toto.tar
```

Extrait dans le répertoire courant le contenu de l'archive `toto.tar`.

La commande `touch`

```
bash $ touch fichier
```

Si `fichier` n'existe pas, crée `fichier`.

Si `fichier` existe, positionne l'heure de dernière modification de `fichier` à l'heure courante.

La commande `tr`

```
bash $ tr a b < fichier
```

Remplace toutes les occurrences de la lettre `a` par le lettre `b` dans le fichier `fichier` et affiche le résultat à l'écran.

La commande `wc`

```
bash $ wc -l toto
```

Compte le nombre de lignes du fichier `toto`.

```
bash $ wc -w toto
```

Compte le nombre de mots du fichier `toto`.

```
bash $ wc -c toto
```

Compte le nombre de caractères du fichier `toto`.

La commande `which`

```
bash $ which cat
```

Donne le chemin absolu de la commande externe `cat` (affiche `"shell built-in command"` si la commande donnée en paramètre à `which` est interne).

La commande `who`

```
bash $ who
raymond    pts/4        Feb  3 10:05    (machine1)
albert     pts/9        Feb  3 15:12    (machine2)
georges    pts/6        Feb  3 11:18    (machine3)
```

Affiche les utilisateurs connectés sur la machine et depuis quelle machine ils sont connectés (dans l'exemple d'exécution fourni, `raymond` est connecté sur la machine locale et il s'est connecté par le réseau depuis la machine `machine1`, etc).

2 Les commandes réseaux

```
bash $ ssh machine1 -l raymond
```

Effectue une connexion interactive sécurisée sur la machine `machine1`. La connexion se fait avec le login `raymond`. Si l'authentification réussit, un shell est ouvert sur `machine1`.

```
bash $ sftp raymond@machine1
```

Effectue une connexion à la machine `machine1` pour transférer des fichiers. La connexion se fait avec le login `raymond`. Si la connexion réussit, on pourra transférer des fichiers depuis ou vers la machine `machine1` grâce aux commandes `put` et `get`. La commande `quit` permet de quitter.

3 Les métacaractères de substitution

```
bash $ ls *.txt
```

Affiche tous les noms de fichiers dont l'extension est `txt`.

```
bash $ ls a??.t*
```

Affiche tous les noms de fichiers dont le nom est composé de la lettre `a` suivi de deux lettre quelconques suivies du caractère `.`, du caractère `t` et de n'importe quoi ensuite.

```
bash $ ls [abc]*.ads
```

Affiche tous les noms de fichiers dont le nom commence par la lettre `a` ou `b` ou `c` suivi de n'importe quoi suivi du caractère `.` et de la chaîne `ads`.

4 Les variables

```
bash $ echo $var
```

Affiche la valeur de la variable `var`.

```
bash $ var=toto  (Bourne Shell)
tcsh $ set var=toto  (CShell)
```

Initialise la variable locale `var` avec la valeur `toto`.

```
bash $ export VAR=toto  (Bourne shell)
tcsh $ setenv VAR toto  (CShell)
```

Initialise la variable d'environnement `var` avec la valeur `toto`.

```
tcsh $ @ var=12
```

Initialise la variable numérique `var` avec la valeur `12` (CShell seulement).

```
tcsh $ set tab = (lundi mardi mercredi jeudi vendredi samedi dimanche)
tcsh $ echo $tab[1]
```

Initialise le tableaux de 7 chaînes de caractères `tab` et affiche le premier élément de ce tableau (Cshell seulement).

```
bash $ set
```

En CShell, affiche les variables locales du shell courant.
En Bourne Shell, affiche toutes les variables.

```
bash $ printenv
```

Affiche les variables d'environnement du Shell.

```
bash $ echo $?
```

Affiche le code de retour de la dernière commande tapée. `0` signifie qu'elle s'est bien passée, un entier différent de `0` signifie qu'elle a échoué.

```
echo $1
```

Dans un script Shell, cette ligne affiche la premier paramètre du script. `$2` représente le second, etc.

```
for i in  $*
```

Dans un script Shell, cette ligne effectue une boucle sur tous les paramètres du script. `i` prend comme valeurs successives le premier paramètre, le second, etc.

5 Les redirections et les pipes

5.1 La redirection de sortie standard

Principe

```
bash $ cmd > fichier
```

Redirige la sortie de la commande `cmd` vers le fichier `fichier`.

```
bash $ cmd >> fichier
```

Concatène la sortie de la commande `cmd` au fichier `fichier`.

Exemples

```
bash $ echo salut > toto
```

Affiche la chaîne de caractères `salut` dans le fichier `toto` et écrase le contenu de `toto`.

```
bash $ cat toto >> titi
```

Concatène le contenu du fichier `toto` au fichier `titi`.

5.1.1 La redirection de l'entrée standard

Principe

```
bash $ cmd < fichier
```

La commande `cmd` lit ses entrées dans le fichier `fichier`. Ne fonctionne que si `cmd` lit en principe ses entrées sur l'entrée standard.

Exemples

```
bash $ read s < f  (bourne Shell)
```

La variable `s` va prendre pour valeur la chaîne de caractères trouvée dans le fichier `f`.

5.1.2 La redirection de l'erreur en bourne Shell

Principe

```
bash $ cmd 2> /tmp/erreur
```

Exécute la commande `cmd` et redirige les messages d'erreurs de cette commande dans le fichier `/tmp/erreur`.

```
bash $ cmd > /tmp/sortie 2> /tmp/erreur
```

Exécute la commande `cmd` et redirige les messages de sortie de cette commande dans le fichier `/tmp/sortie` et les messages d'erreurs de cette commande dans le fichier `/tmp/erreur`.

```
bash $ cmd > /tmp/sortie_erreur 2>&1
```

Exécute la commande `cmd` et redirige les sorties et les erreurs de cette commande dans le fichier `/tmp/sortie_erreur`.

Exemples

```
bash $ find / -name toto > sortie 2> erreur
```

Cherche récursivement dans la répertoire `rep` les fichiers dont le nom est `toto`, écrit la liste de ces fichiers (le chemin absolu) dans le fichier `sortie` et écrit les messages d'erreurs dans le fichier `erreur`.

```
bash $ find / -name toto > sortie_erreur 2>&1
```

Cherche récursivement dans la répertoire `rep` les fichiers dont le nom est `toto`, écrit la liste de ces fichiers (le chemin absolu) dans le fichier `sortie_erreur` ainsi que les messages d'erreurs.

5.1.3 La redirection de l'erreur en CShell

Principe

```
tcsh $ cmd >& /tmp/sortie_erreur
```

Exécute la commande `cmd` et redirige les sorties **et** les erreurs de cette commande dans le fichier `/tmp/sortie_erreur`.

Exemples

```
tcsh $ find / -name toto >& sortie_erreur
```

Redirige les sorties (les chemins trouvés) **et** les erreurs (messages tels que `"permission denied"`) de la commande `find` dans le fichier `sortie_erreur`.

5.1.4 Les pipes

Principe

```
bash $ cmd1 | cmd2
```

La sortie de la commande `cmd1` est redirigée vers l'entrée de la commande `cmd2`. Affiche le contenu détaillé d'un répertoire page par page.

Exemples

```
bash $ ls -l | more
```

Affiche le contenu détaillé d'un répertoire page par page.

```
tcsh $ ls -l  | grep toto
```

6 Les processus

Modes d'exécution

```
bash $ cmd
```

Exécution en mode foreground de la commande `cmd`. Le chemin de la commande doit être dans le `PATH`.

```
bash $ ./cmd
```

Exécution en mode foreground de la commande `cmd` qui se trouve dans le répertoire courant mais le `PATH` ne contient pas le `"."`.

```
bash $ cmd &
```

Exécution en mode background de la commande `cmd`. On recupère la main dans le terminal.

Commandes

```
bash $ ps
```

Affiche la liste des processus du terminal.

```
bash $ ps auwx
```

Affiche de façon détaillée tous les procesus de la machine.

```
bash $ ps -u toto
```

Affiche tous les procesus de l'utilisateur `toto`.

```
bash $ CTRL c
```

Raccourci clavier (les touches `CONTROL` et `c` en même temps) pour envoyer le signal `SIGINT` au processus s'éxecutant en foreground dans le terminal. En général, ce signal provoque l'arrêt du processus.

```
bash $ CTRL z
```

Raccourci clavier (les touches `CONTROL` et `z` en même temps) pour envoyer le signal `SIGTSTP` au processus s'éxecutant en foreground dans le terminal. CE SIGNAL NE PROVOQUE PAS LA MORT DU PROCESSUS !! IL EST SIMPLEMENT SUSPENDU !.

```
bash $ jobs
```

Donne la liste des processus lancés en mode background dans le terminal.

```
bash $ bg %1
```

Passe en mode background le processus suspendu (voir `CTRL z`) d'identificateur **local** `1` (identificateur local valable uniquement dans le terminal de lancement).

```
bash $ fg %1
```

Passe du mode background au mode foreground le processus d'identificateur **local** `1` (identificateur local valable uniquement dans le terminal de lancement).

```
bash $ kill 518
```

Envoie le signal `SIGTERM` (-15) au processus 518.

```
bash $ kill -9 654
```

Envoie le signal `SIGKILL` (-9) au processus 654.

```
bash $ kill %1
```

Envoie le signal `SIGTERM` (-15) au processus d'identificateur **local** `1` (identificateur valable uniquement dans le terminal de lancement).

```
bash $ killall find
```

Envoie le signal `SIGTERM` (-15) à tous les processus dont la commande correspondante contient `find`.