LANGAGE DE SCRIPTS

Linux Operating System

HELHa
Haute École
Louvain en Hainaut

Plan

- **CONCEPTS DE BASE**
  - **Nomination des fichiers et métacaractères** (*, ? et [])
  - **Les commandes** (Syntaxe et manuel)
  - **Le traitement des entrées/sorties** (redirections, piping et enchaînement)

- **MANIPULATION DES FICHIERS**
  - **Manipulation des fichiers:** cat, touch, less, more, mv, cp, rm find, locate
  - **Manipulation des répertoires:** cd, pwd, mkdir, rmdir, ls, cp, mv
  - **Le traitement des fichiers:** grep, wc

- **AUTRES COMMANDES:**
  - cal, clear, cut, date, du, file, head, id, sed, sh, stty, su, tail, tee, tty, who, w, watch, whereis, which, free, vmstat
  - **Comparaison de quelques commandes: Dos - Linux**

Plan (suite)

- **LA PROGRAMMATION DU BASH**
  - **- La neutralisation des métacaractères**
  - **- Les commentaires**
  - **- La commande** echo
  - **- Les variables**
  - **- La substitution de commandes**
  - **- Les commandes** test, if, case et read
  - **- Les boucles** (for, while, do while, les tableaux)
  - **- Les commandes** expr, eval et :
  - **- Les alias**
  - **- Les extensions de la commande** cd
  - **- Les commandes** select, dirname , basename et seq
  - **- Les fonctions**

  - Travaux pratiques au laboratoire

Linux Operating System

CONCEPTS DE BASE

Concepts de base

- **NOMINATION DES FICHIERS** **Règles de nomination**

- 255 caractères au maximum
- les caractères utilisés par le shell sont à proscrire: ' " ` * ? > < ...
- . et .. sont interdits
- évitez les accentués et les combinaisons de touches
- souvenez-vous que LINUX respecte la casse
- éviter de donner à un fichier le nom d'une commande LINUX
- les fichiers dont le nom commence par un point seront cachés pour la plupart des commandes. Les commandes agiront en général sur tous les fichiers sauf sur les fichiers cachés à moins de le demander expressément par une option ou en mentionnant explicitement leur nom.

Exemples de noms de fichier valides:

README	*informix*
0	*INFORMIX*
important	*.cache*
fichier.old	*.profile*
fichier_new	*f1.old.2*

HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **NOMINATION DES FICHIERS: Repérage des fichiers**

L'identification d'un fichier est donnée par un *chemin d'accès*. Un chemin d'accès *(pathname)* est une suite de noms de répertoires séparés par des / et terminée par le nom de l'entité à identifier (fichier ou répertoire).

Chemin absolu:

Suite complète de noms de la racine du système *(root)* à l'entité.

ex. **/dir1/dir2/.../dirn/fichier** --> le premier / est la racine.

Chemin d'accès relatif:

Suite de noms relatifs à un répertoire quelconque du système ou au répertoire courant.

ex. **dir1/dir2/.../dirn/fichier** *ou* **./dir1/dir2/.../dirn/fichier**

Exemples:

/	représente le nom absolu de la racine du système;
/bin	répertoire directement sous root;
/home/jean/fich	le fichier *fich* appartient au répertoire *jean*, lui-même contenu dans le répertoire *home* du répertoire racine /.

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **NOMINATION DES FICHIERS: Symboles spéciaux**

De manière à faciliter la manipulation des noms dans le système et surtout les recherches, LINUX met à notre disposition un ensemble de symboles *(métacaractères)*:

***** Remplace un nombre indéfini (0 ou plus) de caractères.

? Remplace un seul caractère.

[abc] Limite le remplacement d'un caractère au choix entre abc.

[a-f] Limite le remplacement d'un caractère au choix entre abcdef.

Exemples:

*.BAK	Pour sélectionner tous les fichiers ayant l'extension .BAK.
chap?	Pour sélectionner tous les fichiers de 5 caractères commençant par chap.
[abc]*	Pour sélectionner tous les fichiers dont le nom commence par a,b ou c.
[!abc]*	Pour sélectionner tous les fichiers dont le premier caractère précisé dans la liste n'est pas a,b ou c.
[a-zA-Z]*	Pour sélectionner tous les fichiers dont le nom commence par une des lettres de l'alphabet (minuscule ou majuscule).

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **NOMINATION DES FICHIERS: Symboles spéciaux (Suite)**

/home/fred/??*	Pour sélectionner l'ensemble des noms des fichiers du répertoire /home/fred qui sont composés d'au moins 2 caractères.
/home/fred/*/core	Pour sélectionner tous les fichiers 'core' qui se trouvent dans tous les sous-répertoires du répertoire /home/fred.

Remarque:

	Microsoft	**Linux**
*	sélectionne les fichiers qui n'ont pas d'extension	sélectionne tous les fichiers
.	sélectionne tous les fichiers	sélectionne tous les fichiers dont le nom est composé d'un point au moins

Concepts de base

HELHa
Haute École
Louvain en Hainaut

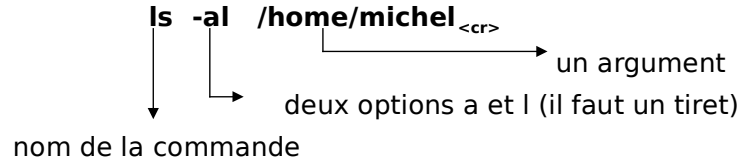
- **LES COMMANDES**

- Pour utiliser des tâches utilisateur ou système.
- Chaque commande est lue caractère par caractère par shell.
- Le shell analyse, décode et valide la commande introduite puis charge le programme correspondant à cette commande en Ram (si commande externe).
- Le processeur exécute ce programme.

SYNTAXE GENERALE

NOM_DE_COMMANDE [OPTIONS] [ARGUMENTS]

EXEMPLE



Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LES COMMANDES (Suite)**

REMARQUES

- Chaque commande est décrite dans un manuel (man pour les commandes externes et help pour les commandes internes).

- Une commande peut être décrite dans plusieurs manuels ...
ex. `# man crontab`

```
CRONTAB(1)                                                CRONTAB(1)

NAME
       crontab - maintain crontab files for individual users (V3)
...
SEE ALSO
           crontab(5), cron(8)


# man 5 crontab
CRONTAB(5)                                                CRONTAB(5)

NAME
       crontab - tables for driving cron
...
```

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LES COMMANDES (Suite)**

REMARQUES (Suite)

- Liste des manuels

Manuel 1:	Les commandes utilisateurs
Manuel 2:	Les appels systèmes
Manuel 3:	Les fonctions bibliothèques
Manuel 4:	Les fichiers périphériques
Manuel 5:	Les fichiers d'administration
Manuel 6:	Les jeux
Manuel 7:	Divers
Manuel 8:	Les commandes d'administration

- `man -k motclef` : affiche les commandes, brièvement définies, en rapport avec motclef (`ex. man –k compress`).
- Les shells (bash ,csh ...) gèrent l'historique des commandes (Flèche haut/bas) ainsi que la gestion de la frappe semi-automatique (Tab).
- `history`: liste des dernières commandes passées (`-c`: effacer l'historique).
- `whatis`: donne un descriptif succinct de la commande (`ex. whatis cp`).
- `makewhatis`: met à jour l'index du manuel.

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O** **Périphériques standards**

- Dispositifs de communication avec le système (écran, clavier, imprimante, ...).
- La plupart des commandes reçoivent leur entrée de ***l'entrée standard*** et envoient leur sortie vers la ***sortie standard*** :

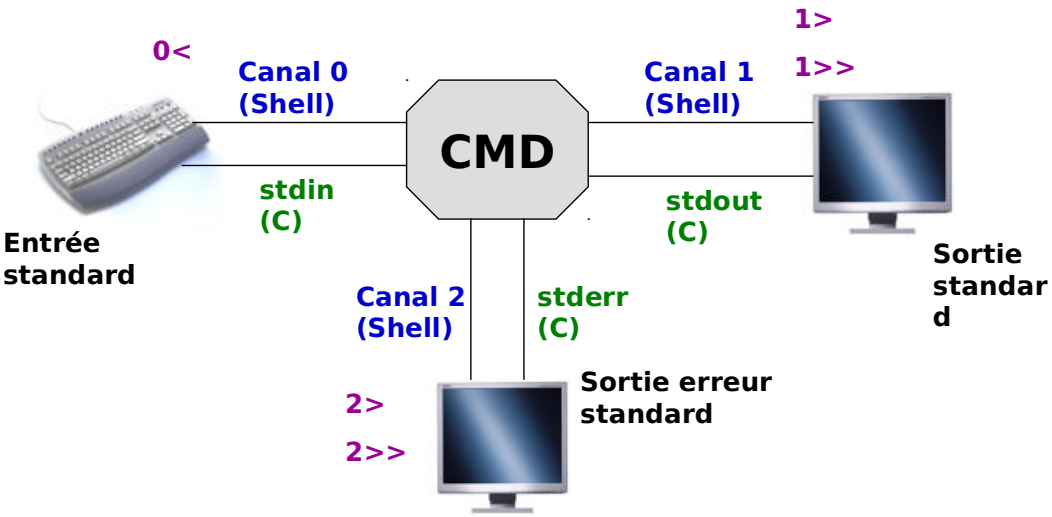
  . l'entrée standard correspond au clavier;
  . la sortie standard correspond à l'écran;
  . la sortie standard erreur correspond à l'écran.

- Linux associe un ***descripteur de fichier*** *(file descriptor)* à chaque entrée/sortie standard :

  . l'entrée standard (standard input) = 0 (canal 0)
  . la sortie standard (standard output) = 1 (canal 1)
  . la sortie standard erreur (standard error) = 2 (canal 2)

Concepts de base

- **LE TRAITEMENT DES I/O** **Synoptique**



Concepts de base

- **LE TRAITEMENT DES I/O** **En langage C**

  - En langage C,

```
printf("%d\n",nb);   est équivalent à  fprintf(stdout,"%d\n",nb);
scanf("%d\n",&nb);   est équivalent à  fscanf(stdin,"%d\n",&nb);
```

  - Par contre, pour envoyer un message via le flux stderr, on sera obligé d'écrire:

```
fprintf(stderr,"Mon message d'erreur\n");
```

Concepts de base

- **LE TRAITEMENT DES I/O** **Via un shell**

- Permet de remplacer les STDIN, STDOUT et STDERR par d'autres fichiers.
- Les informations peuvent être prises ou rangées dans un fichier ordinaire ou spécial associé à un périphérique.

< name *prendre l'entrée dans un fichier 'name'.*

<< mot *prendre l'entrée dans une séquence de lignes jusqu'à la ligne qui contient 'mot'.*

> name *mettre la sortie dans un fichier 'name'. Si 'name' existe déjà, il sera écrasé; sinon il est créé.*

>>name *ajouter la sortie à la fin du fichier 'name'. Si 'name' n'existe pas, il est créé.*

La syntaxe associée étant:

COMMANDE **<** **>** **>>** ***FICHIER***

HELHa
Haute École
Louvain en Hainaut

Concepts de base

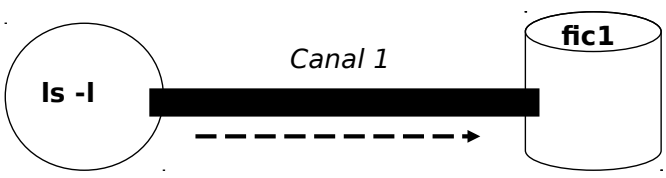
- **LE TRAITEMENT DES I/O** **Via un shell (suite)**

EXEMPLES

$ ls -l > fic1

La liste du contenu du répertoire est placée dans fic1.

Schéma de fonctionnement



Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

$ cat fic2 >> fic3

Le contenu de fic2 est ajouté à la fin de fic3.

Exercice: Faire le schéma de fonctionnement

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

$ date > /dev/tty2

La date est envoyée sur le terminal tty2

Exercice: Faire le schéma de fonctionnement

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

```
$  ls -l  toto 2> erreur
```

Le message d ’erreur éventuel engendré par la ‘ ls ’ est placé dans le fichier ‘ erreur ’.

Attention: pas d ’espace entre le descripteur et le symbole de redirection !

Exercice: Faire le schéma de fonctionnement

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

```
$  cat fich2  2>> temp  > inter
```

Si fich2 existe, son contenu est placé dans le fichier 'inter'. Dans le cas contraire, le message d'erreur est ajouté au contenu du fichier 'temp'.
*(**Remarque:** On aurait pu écrire cat fich2 2>> temp 1> inter).*

Exercice: Faire le schéma de fonctionnement

Concepts de base

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

$ cat > temp < fich

Le contenu du fichier 'fich' est placé dans le fichier 'temp'.
*(**Remarque:** On aurait pu écrire cat fich > temp).*

Exercice: Faire le schéma de fonctionnement

HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

$ grep '#!/bin/bash' /etc/init.d/* 1>> rech 2> /dev/null

Cette commande recherche la chaîne '#!/bin/bash' dans tous les fichiers du répertoire courant; le résultat de la recherche est redirigé dans le fichier 'rech'; les messages d'erreurs sont envoyés dans le fichier fantôme /dev/null, ce qui équivaut à les ignorer.

Exercice: Faire le schéma de fonctionnement

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

```
$ cat > fich1
```

Tout ce que l'utilisateur entrera au clavier, jusqu'au prochain <CTRL><D>, sera placé dans le fichier 'fich1'.

Exercice: Faire le schéma de fonctionnement

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

```
$ cat << ! >> fich2
```

Tout ce que l'utilisateur entrera au clavier, jusqu'au prochain !, sera ajouté au fichier 'fich2'.

! doit être en première position et seul sur sa ligne.

Exercice: Faire le schéma de fonctionnement

Concepts de base

- **LE TRAITEMENT DES I/O Via un shell (suite)**

EXEMPLES (Suite)

```
$ > /home/jean/essai
```

Crée un fichier '/home/jean/essai' vide (taille = 0). S'il existe déjà, il est vidé de tout son contenu.

Exercice: Faire le schéma de fonctionnement

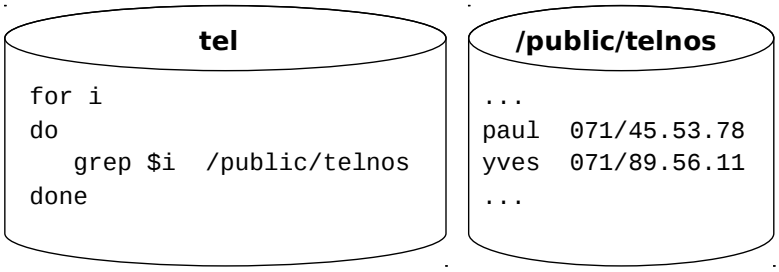
HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **LE TRAITEMENT DES I/O Via un shell (suite)**

REMARQUE: La double redirection en entrée (<<)

Soit la situation suivante:




```
$ ./tel paul
paul 071/45.53.78
```

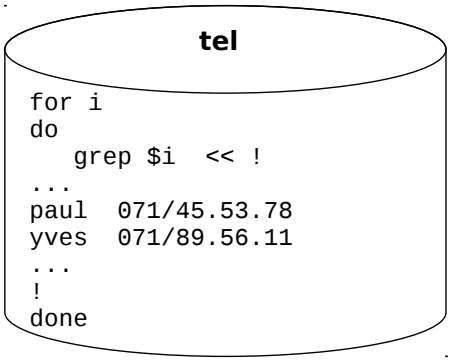
☹ Lecture des données dans un autre fichier => déplacement des têtes => lenteur

😄 Les données sont séparées du programme (ok pour mise à jour)

Concepts de base

- **LE TRAITEMENT DES I/O** **Via un shell (suite)**

REMARQUE: La double redirection en entrée (<<) (Suite)



$ ./tel paul

paul 071/45.53.78

😀 Lecture des données dans le programme même => moins de déplacement des têtes => plus rapide

☹ Les données ne sont pas séparées du programme (difficile à mettre à jour)

Conclusion: Cette solution est envisageable pour des données statiques.

HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **LE TRAITEMENT DES I/O** **Les canaux**

Soit la commande *cat /etc/group > fichier 2>&1*

> fichier redirige le canal de sortie vers le fichier *fichier.*
2>&1 réunit en un seul canal la sortie erreur et la sortie standard.

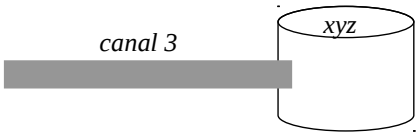
Exercice: Faire le schéma de fonctionnement

Concepts de base

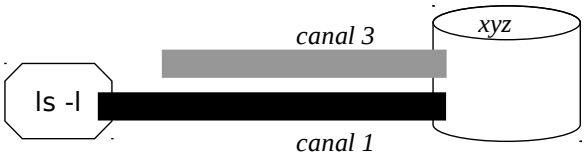
- **LE TRAITEMENT DES I/O** **Les canaux**

- Linux permet de gérer 7 autres canaux:

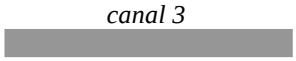
$ exec 3> xyz *Le fichier xyz est ouvert sans être directement fermé.*
Le canal 3 devient un canal de sortie, celui du fichier xyz



$ ls –l 1>&3 *Le fichier xyz reçoit le résulat du ls –l*



$ exec 3>&- *Fermeture du canal 3.*



HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **LE TRAITEMENT DES I/O** **Les canaux (suite)**

- Un nouveau canal peut aussi être ouvert en entrée:

$ exec 3< abc
$ wc < &3

} $ wc < abc

- Dans la commande

$ (ls –l ; ps –ef ; who) > Listing 2>&1 &

Toutes les commandes sont lancées en arrière plan (&).
Le résultat de toutes les commandes ainsi que les erreurs éventuelles sont redirigées vers ‘ Listing ’.

- Un canal ne peut pas être à la fois utilisé en entrée et en sortie.

$ exec 3>xyz 3<xyz ne marche pas !!

Concepts de base

- **LE TRAITEMENT DES I/O** **Les canaux (suite)**

- L'ordre d'apparition des redirections peut avoir de l'importance:

  $ ls –l 1> fich 2>&1

  Le résultat normal de la commande est redirigé vers fich tandis que les erreurs éventuelles sont redirigées vers le périphérique relié au canal 1 (également fich par conséquent)

  $ ls –l 2>&1 1> fich

  Le résultat normal est envoyé vers fich tandis que les erreurs éventuelles sont envoyées à l'écran.

  Exercice: Faire le schéma de fonctionnement de ces 2 commandes.

- Il est aussi possible d'ouvrir des canaux en mode ajout (append):

  $ exec 3>> abc *(ouverture en append)*
  $ echo bonjour >&3 *(ajout de bonjour au fichier abc)*

  ..... >>&3 *(génère un message d'erreur).*

HELHa
Haute École
Louvain en Hainaut

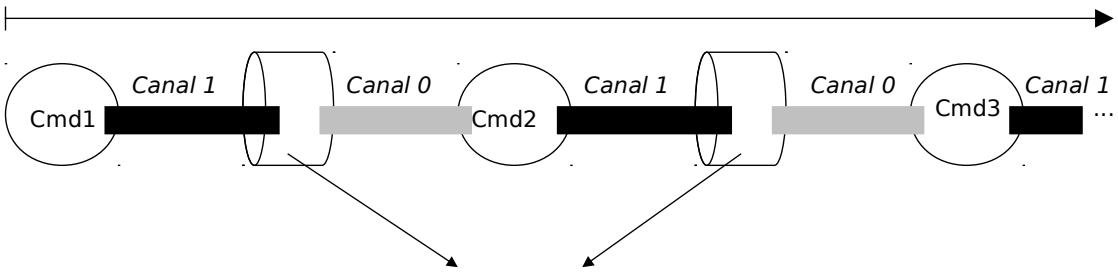
Concepts de base

- **LE TRAITEMENT DES I/O** **Les tubes (Pipes)**

- Mécanisme permettant l'enchaînement de plusieurs commandes.
- **Syntaxe d'utilisation:**

  Cmd1 | Cmd2 | Cmd3 …

- **Synoptique:**



Tubes créés par le shell en Ram et détruits automatiquement au fur et à mesure de l'exécution de la commande.

Ces tubes ne portent pas de nom. Ils sont dits tubes 'anonymes'.

Concepts de base

- **LE TRAITEMENT DES I/O** **Les tubes (Pipes) (Suite)**

- **Exemples:**

$ cat /etc/passwd | sort *Affiche le contenu trié en ordre croissant de '/etc/passwd'.*

$ who | wc -l *Compte le nombre d'utilisateurs connectés au système.*

- **Remarques:**

. Les tubes sont unidirectionnels. Pour synchroniser deux processus qui sont reliés par un tube, Linux utilise la technique des sémaphores ...

. La commande ***mkfifo*** permet de créer des tubes nommés.

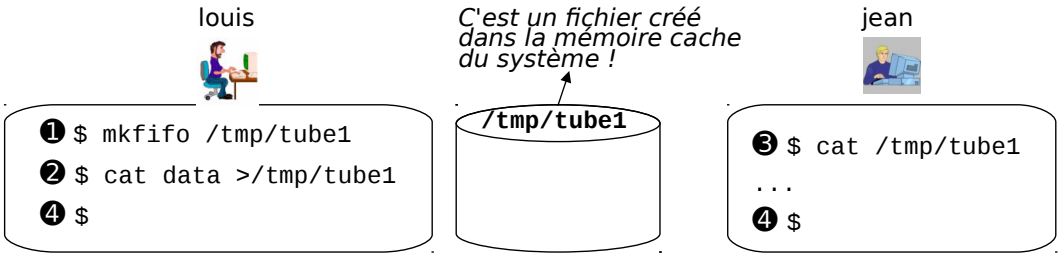
. Dans un tube quand on entre quelque chose, cela ressort.
Ceci est appelé une FIFO (*First-In-First-Out*, premier entré, premier sorti) car la première donnée entrée est aussi la première donnée sortie.

HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **LE TRAITEMENT DES I/O** **Les tubes (Pipes) (Suite)**

- **Illustration:** La commande *mkfifo*



❶ louis crée un tube (fichier de type p de taille nulle)

❷ Il tente de remplir le tube de données (la taille du tube est toujours nulle car ce n'est pas un fichier disque !)
Le processus ne rend pas la main: il attend que quelqu'un vienne lire dans le tube ('cat data' n'est pas encore exécuté !)

❸ 'cat data' de Louis est exécuté et Jean vide le tube en le lisant.

❹ jean et Louis recupèrent leur prompt.

Concepts de base

- **LE TRAITEMENT DES I/O** **Les tubes (Pipes) (Suite)**

- **Constatations:** La commande *mkfifo*

. La taille d'un tube est toujours nulle car il ne fait que lier deux processus entre eux. Il est en Ram et les données n'y font que passer.

. Dans tous les cas, puisque le tube a un nom, les deux processus n'ont pas à appartenir à la même ligne de commandes ou même à être lancés par le même utilisateur.

. Lors de l'envoi de données dans un tube nommé, il est conseillé de lancer le processus en arrière-plan pour récupérer le prompt.

. Un tube se détruit comme n'importe quel autre fichier.

. Les sockets sont similaires aux tubes mais elles ne fonctionnent qu'à travers un réseau (ceci est un autre sujet).

HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **LE TRAITEMENT DES I/O** **L'enchaînement des commandes**

- cmd1 ; cmd2 ; ... ; cmdN

Les commandes s'exécutent séquentiellement et indépendamment les unes des autres.

Exemple

$ date ; who am i ; echo fin des commandes

Affiche la date système, l'identification de l'utilisateur et la chaîne 'fin des commandes'.

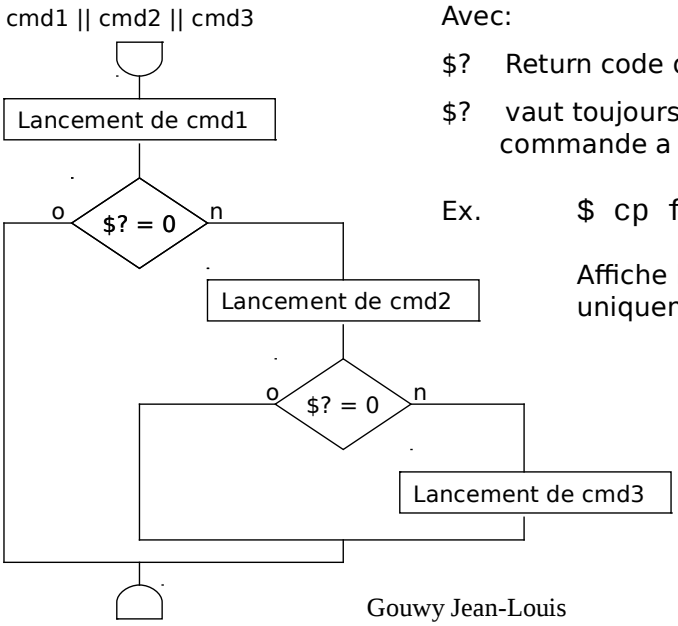
$ cp file1 file2; ls

Affiche les entrées du dossier courant et ce, quelle que soit l'issue de la copie.

Concepts de base

- **LE TRAITEMENT DES I/O L'enchaînement des commandes (Suite)**

- cmd1 || cmd2 || ... || cmdN



Avec:

$? Return code de la commande

$? vaut toujours 0 si la commande a atteint son but.

Ex. `$ cp file1 file2 || ls`

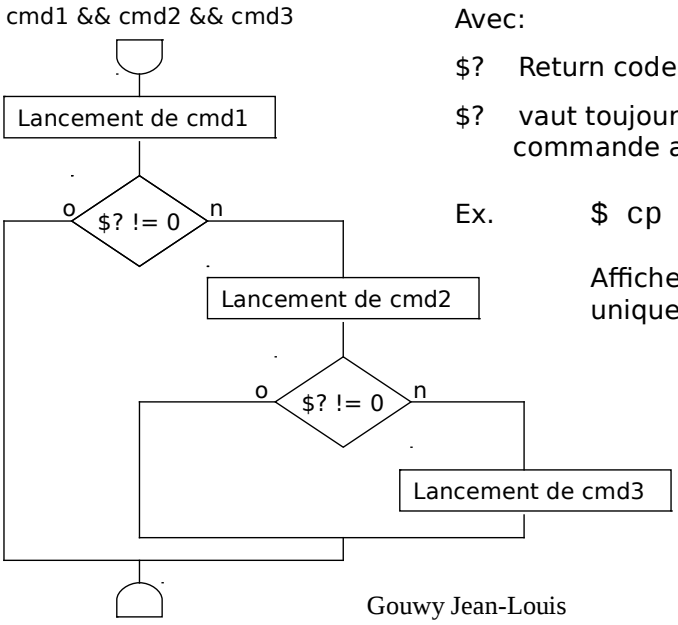
Affiche les entrées du dossier courant uniquement si la copie a échoué.

HELHa
Haute École
Louvain en Hainaut

Concepts de base

- **LE TRAITEMENT DES I/O L'enchaînement des commandes (Suite)**

- cmd1 && cmd2 && ... && cmdN



Avec:

$? Return code de la commande

$? vaut toujours 0 si la commande a atteint son but.

Ex. `$ cp file1 file2 && ls`

Affiche les entrées du dossier courant uniquement si la copie a réussi.

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O L'enchaînement des commandes (Suite)**

Remarque: Groupement de commandes **()**

$ (grep 'author' * ; grep 'auteur' *) | more

Toutes les lignes contenant les chaîne 'author' ou 'auteur' de tous les fichiers du dossier courant seront affichées page écran par page écran.

$ grep 'author' * ; grep 'auteur' * | more

Seules les lignes contenant les chaîne 'auteur' de tous les fichiers du dossier courant seront affichées page écran par page écran.
Les lignes contenant les chaîne 'author' seront, quant à elles, affichées à la volée.

Concepts de base

HELHa
Haute École
Louvain en Hainaut

- **LE TRAITEMENT DES I/O L'enchaînement des commandes (Suite)**

Expliquez les commandes suivantes:

$ grep "Tomates" Legume >/dev/null 2>&1 && echo "Tomates trouvées"

$ grep "Tomates" Legume >/dev/null 2>&1 || echo "Tomates pas trouvées"

$ ps –ax | grep cupsd >/dev/null 2>&1 && lpr Legume
Vous constaterez peut-être un petit problème de synchronisation. Comment le régler ?

$ rmdir mon_premier 2>/dev/null || echo "Suppression impossible"

$ (cd /tmp && rm -f poubelle) 2>/dev/null || echo "/tmp inaccessible ou poubelle absente"

Linux Operating System

MANIPULATION DES FICHIERS

HELHa
Haute École
Louvain en Hainaut

Manipulation des fichiers

- **CREATION DES FICHIERS**

  – Via des éditeurs tels que vi ou mcedit.
  – Via le mécanisme de redirection des entrées/sorties.

  $ cat > monfich

  Donne le contrôle au clavier pour entrer des informations dans le fichier "monfich". La fin de fichier est obtenue en frappant <***CTRL***><***D***>.

  $ > monfich

  Crée un fichier vide de nom "monfich".

  $ touch monfich

  Crée un fichier vide de nom "monfich" si ce fichier n'existe pas encore. S'il existe, la date et l'heure de la dernière modification est mise à jour dans son i-node.

Manipulation des fichiers

HELHa
Haute École
Louvain en Hainaut

- **VISUALISATION DU CONTENU DES FICHIERS**

  – Via des éditeurs ...
  – Via les commandes cat, less, more.

  $ cat monfich

  Affiche complètement le contenu du fichier "monfich".

  $ less monfich

  Affiche le contenu du fichier "monfich" page écran par page écran.

  $ more monfich

  Affiche le contenu du fichier "monfich" page écran par page écran.

Manipulation des fichiers

HELHa
Haute École
Louvain en Hainaut

- **VISUALISATION DU CONTENU DES FICHIERS**

  Exercice:

  Réalisez le schéma interne de fonctionnement de ces 3 commandes

  $ more monfich

  $ more < monfich

  $ cat monfich | more

Manipulation des fichiers

HELHa
Haute École
Louvain en Hainaut

- **CONCATENATION DE FICHIERS**

La commande cat *(concatenate)* cat [OPTION] [FILE]...

$ cat /etc/motd fich1

Affiche le contenu du fichier /etc/motd et du fichier fich1.

$ cat file1 file2 > file3

Le fichier file3 est créé par la concaténation de file1 et file2.

$ cat file1 >> file2

Ajoute le contenu de file1 à la fin de file2.

✓ Aucun fichier en entrée ne doit avoir le même nom que le fichier en sortie, à moins que ce ne soit un fichier spécial. Si la commande suivante est lancée, le système envoie un message d'erreur.

$ cat fich1 fich2 > fich1

Manipulation des fichiers

HELHa
Haute École
Louvain en Hainaut

- **DEPLACEMENT & CHANGEMENT DE NOM**

La commande mv *(move)* mv [OPTION]... SOURCE DEST (1)
mv [OPTION]... SOURCE... DIRECTORY (2)

(1) Permet de renommer le fichier source en dest.

$ mv file1 file2

Le fichier de nom file1 est renommé en file2.

$ mv *.obj /tmp

Tous les fichiers d'extension .obj sont transférés du répertoire de travail vers le répertoire /tmp.

Manipulation des fichiers

HELHa
Haute École
Louvain en Hainaut

- **COPIE DE FICHIERS**

La commande cp *(copy)*

cp [OPTION]... SOURCE DEST (1)
cp [OPTION]... SOURCE... DIRECTORY (2)

(1) Copie de source vers dest.

$ cp passwd passwd.old

Copie le fichier 'passwd' du répertoire courant dans un fichier 'passwd.old' de ce même répertoire.

$ cp *.c /home/joel

Copie tous les fichiers d'extension .c dans le répertoire /home/joel.

Manipulation des fichiers

HELHa
Haute École
Louvain en Hainaut

- **COPIE DE FICHIERS (Suite)**

La commande cp *(copy)* **(Suite)**

✓ Si le fichier spécifié existe déjà , il est écrasé sans laisser de message.

✓
```
$ echo $HOME
/home/gouwy
$ pwd
/home/gouwy/manip
$ cp /etc/passwd .
$ ls -l passwd
-rw-rw-r-- 1 gouwy prof 2279 Oct 06 13:45 passwd
```

On constate que le fichier /etc/passwd a bien été lu pour être recopier dans /home/gouwy/manip. Mais remarquons que l'utilisateur 'gouwy' devient propriétaire du fichier car il est à l'origine de la commande de copie.

```
$ cp passwd /etc
cp: Permission denied
```

Ici, l'utilisateur 'gouwy' n'a pas la permission d'écrire dans le répertoire /etc ... heureusement !

Manipulation des fichiers

HELHa
Haute École
Louvain en Hainaut

- **COPIE DE FICHIERS (Suite)**

La commande cp *(copy)* (Suite)

```
$ cp  file1.c  /tmp
$ ls -l /tmp
-rw-r--r--  1  gouwy  prof  .....  file1.c
```

Quand le propriétaire d'un fichier le copie vers un autre endroit, il en garde la propriété.
Si le fichier spécifié existe déjà , il est écrasé sans laisser de message.

Conclusion

C'est toujours celui qui est à l'origine de la copie qui s'approprie la propriété du fichier copié !

Manipulation des fichiers

- **SUPPRESSION DE FICHIERS**

La commande rm *(remove)*

rm [OPTION]... FILE...

La commande *rm* supprime les fichiers précisés.

$ rm fich1
Supprime le fichier 'fich1' après confirmation.

$ rm -f notexist
Un fichier « non supprimable » est ignoré. Aucun message d'erreur n'est affiché.
Aucune demande de confirmation quand on supprime un lot de fichiers.

$ rm -f /bin/login
Message d'erreur car un utilisateur normal n'a pas le droit d'écrire dans le répertoire bin.

$ rm -rf /home/joel
Suppression non-interactive de tous les fichiers du répertoire précisé, y compris les fichiers contenus dans les sous-répertoires.

Manipulation des fichiers

- **SUPPRESSION DE FICHIERS (Suite)**

La commande rm *(remove)* (Suite)

✓ Si un fichier a des liens (voir *ln*), seule l'entrée dans le répertoire est effacée.

✓ Pour pouvoir supprimer un fichier, l'utilisateur doit avoir la permission d'écriture dans le répertoire qui le contient.

✓ On peut supprimer un fichier ne nous appartenant pas et se trouvant dans notre espace de travail, si ce fichier a été copié chez nous.

Manipulation des fichiers

- **RECHERCHE DE FICHIERS**

La commande find

find [path...] [expression]

Recherche de manière récursive à partir de chemins "pathname...", tous les fichiers qui correspondent à une expression logique contenant les options décrites ci-dessous.

Quelques options

-name fichier	Vrai si le fichier est identique au nom de fichier en cours. Les métacaractères doivent être masqués avec \.
-type x	Vrai si le type du fichier est du type indiqué (b, c, d, p, l ou f)
-links n	Vrai si le fichier à n liens.
-user unom	Vrai si le fichier appartient à l'utilisateur unom.
-group gnom	Vrai si le fichier appartient au groupe gnom.
-size taille	Vrai si la taille du fichier correspond à celle indiquée.
-perm mask	Vrai si les permissions du fichier valent celles du masque.
-mtime n	Vrai si le fichier a été modifié il y a n jours.

Manipulation des fichiers

- **RECHERCHE DE FICHIERS (suite)**

La commande find (suite)

-exec cmd Vrai si la commande cmd exécutée retourne 0 comme code de sortie. La fin de cmd doit être ponctuée par un point-virgule masqué. Un argument {} est remplacé par le nom de chemin en cours.

-print Toujours vrai. Entraîne l'impression à l'écran du nom des fichiers correspondants (valeur par défaut).

(expression) Vrai si l'expression entre parenthèses est vraie (les parenthèses doivent être masquées).

Opérateurs

! Négation de l'option.

(AND) AND est implicite entre 2 options.

-o OR est spécifié par l'opérateur -o donné entre 2 options.

Manipulation des fichiers

- **RECHERCHE DE FICHIERS (suite)**

La commande find: Exemples

 # find / -name "fd*" -type b -print
Recherche et affiche, à partir du répertoire principal /, le nom complet de tous les fichiers spéciaux bloc commançant par fd.

 # find /usr/bin /bin -name fi -print
Recherche et affiche, à partir du répertoire /usr/bin puis de /bin, le nom complet de tous les fichiers s'appelant fi.

 # find . -name precis -links 2 -print
Recherche et affiche, à partir du répertoire courant, le nom complet de tous les fichiers s'appelant "precis" et ayant un nombre de links égal à 2.

 # find / -size +100k
Recherche et affiche, à partir du répertoire principal /, le nom complet de tous les fichiers dont la taille est supérieure à 100k (-print pas obligatoire).

Manipulation des fichiers

- **RECHERCHE DE FICHIERS (suite)**

La commande find: Exemples (suite)

 # find /home/gouwy \(-name essai1 -o -name "*cbl" \)
Recherche et affiche, à partir du répertoire /home/gouwy, le nom complet de tous les fichiers s'appelant "essai1" ou se terminant par cbl.

 # find / -name core -exec rm -f {} \;
Recherche à partir de / tous les fichiers de nom 'core' et les supprime.

 # find / -name README -exec ls -lh \;
Recherche, à partir de /, toutes les entrées s'appelant README et les affiche en format long et "humainement lisible".

Manipulation des fichiers

- **RECHERCHE DE FICHIERS (suite)**

Les commandes updatedb / locate

La commande *updatedb* met à jour la base de données (/var/lib/mlocate/mlocate.db) contenant les fichiers présents sur le système et elle permet via la commande *locate* de trouver un fichier dans l'arborescence.

😄 Le temps de recherche est très réduit par rapport à la commande find puisque cette recherche ne se fait que dans la base de données.

☹ La mise à jour de la base de données peut prendre beaucoup de temps.

man locate ...

Manipulation des répertoires

- **CHANGEMENT DE REPERTOIRE: cd**

cd [dir]

Permet de changer de répertoire.

$ cd /home/other/babar
Le répertoire courant devient /home/other/babar.

$ cd **..**/babar
Le répertoire courant devient le répertoire situé au même niveau que le répertoire "babar".

$ cd
Le répertoire courant devient celui de la "home directory".

$ cd **.**
On ne change pas de répertoire.

Manipulation des répertoires

- **NOM DU REPERTOIRE COURANT: pwd**

pwd

Permet de changer de répertoire.

$ cd **..**
$ pwd
/mnt/users/util

Ici le répertoire courant est /mnt/users/util.

Manipulation des répertoires

- **CREATION D'UN REPERTOIRE: mkdir**

mkdir [OPTION] DIRECTORY...

Permet de créer un répertoire.
Les entrées . et .. sont créées automatiquement.

```
$ mkdir bidon
$ ls -l
total 4
drwxr-xr-x    2 root     root         4096 Nov  2 17:52 bidon
```

Taille en Ko du répertoire.
Actuellement, le répertoire contenant l'entrée 'bidon' monopolise 4 Ko dans le file system. Pourquoi ?

```
$ mkdir –p perso/documents
```

Création du dossier 'perso' (s'il n'existe pas encore) et du dossier 'documents' dans celui-ci.

Manipulation des répertoires

- **SUPPRESSION D'UN REPERTOIRE: rmdir**

rmdir [OPTION]... DIRECTORY...

Tous les fichiers du répertoire doivent avoir été préalablement supprimés.
Il est impossible de détruire un répertoire dans lequel on se trouve, ou un répertoire en amont.

```
$ pwd
/usr/util1
$ rmdir dir1
```

Destruction du répertoire 'dir1' du répertoire /usr/util1.

✓ La commande ***rm -rf*** permet de supprimer tout un répertoire (y compris ses sous-répertoires) même s'il n'est pas vide.

A manipuler avec précaution.

Manipulation des répertoires

- **VISUALISATION D'UN CONTENU D'UN REPERTOIRE: ls**

ls [OPTION]... [FILE]...

- *ls* liste les répertoires et les fichiers précisés.
- Par défaut, la sortie est envoyée à l'écran par ordre alphabétique.
- Sans options, *ls* envoie le seul nom du fichier.
- Si aucun argument n'est précisé, le répertoire courant est pris par défaut.

Options:

-d	affiche les répertoires avec la même représentation que les fichiers, sans lister leur contenu.
-l	donne toutes les informations de service (format long).
-a	liste tous les fichiers y compris les fichiers cachés.
-R	liste récursive c'est-à-dire aussi celle des répertoires rencontrés.
-i	affiche le numéro d'i-node pour chaque fichier.
-h	humainement lisible (human readable).

Manipulation des répertoires

- **VISUALISATION D'UN CONTENU D'UN REPERTOIRE: ls (suite)**

Exemple:

Nbre de liens durs — *Nom du propriétaire* — *Nom du groupe d'appartenance*

```
$ ls -l
drwxr-x---  2  paul  ventes  4096  Apr  5  14:33   dir
-rw-r--r--  1  paul  produc  403   Apr  1  13:12   fich
```

Taille en bytes — *Nom du fichier* — *Date et heure de la dernière modification.*

Protection bits
r = Read w= Write x= eXecute
- = droit non accordé
dans l'ordre pour le propriétaire, groupe ou autres.

Le type de fichier
d = directory
- = fichier ordinaire
c = fichier spécial caractère
b = fichier spécial bloc
...

✓ Tous ces renseignements se trouvent dans l'i-node (excepté le nom du fichier !).

Manipulation des répertoires

- **COPIE DE REPERTOIRES: cp**

```
$ cp -R . /backup
```

Toute la structure hiérarchique du répertoire courant est recopiée sous le répertoire 'backup'.-
(R = récursif)

- **TRANSFERT/CHANGEMENT DE NOM D'UN REPERTOIRE: mv**

```
$ mv dir1 dir2
```

Le répertoire dir1 est renommé en dir2.

```
$ mv * /backup
```

Toute la structure hiérarchique du répertoire courant est transférée sous le répertoire 'backup'.

Traitement des fichiers

- **RECHERCHE D'UNE CHAINE DANS UN FICHIER**

La commande grep

grep [options] PATTERN [FILE...]

Filtre qui cherche dans les fichiers précisés les lignes correspondantes à un 'pattern' de recherche. Normalement, chaque ligne correspondant à l'expression donnée est affichée à l'écran.

Options:

-c seul le nombre de lignes correspondant à l'expression sont affichées.

-l le seul nom des fichiers contenant au moins une fois une ligne correspondant à l'expression est affiché.

-n donne le numéro de ligne dans un fichier.

-v toutes les lignes, sauf celles correspondant à l'expression, sont affichées.

-i ne respecte pas la casse.

-R recherche récursive à partir d'un répertoire donné.

Traitement des fichiers

- **RECHERCHE D'UNE CHAINE DANS UN FICHIER (Suite)**

La commande grep (Suite)

Exemples:

```
# grep 'shutdown' /var/log/messages
```

Affiche la ligne de /var/log/messages qui contient le littéral *shutdown*.

```
# grep 'shutdown' /var/log/*
```

Affiche les lignes des fichiers du dossier /var/log contenant la chaîne *shutdown* et affiche le nom des fichiers.

```
# grep –n 'shutdown' `ls /var/log/*`
```

Idem mais avec en plus le numéro des lignes contenant la chaîne *shutdown* .

```
# grep -l ‘shutdown' /var/log/*
```

Affiche le nom des fichiers de /var/log qui contiennent au moins une fois la chaîne *shutdown*.

Traitement des fichiers

- **RECHERCHE D'UNE CHAINE DANS UN FICHIER (Suite)**

La commande grep (Suite)

```
$ grep -Rni 'dhclient' /etc
```

Affiche les lignes et leur numéro relatif des fichiers de toute la structure hiérarchique du répertoire /etc contenant la chaîne *dhclient* et affiche le nom des fichiers. La recherche se fait sans respecter la casse.

Remarque:

Le code de sortie est:

0 si des lignes correspondantes sont trouvées;
1 si aucune ligne ne l'est;
2 pour les erreurs de syntaxe ou les fichiers inaccessibles.

Atelier:

Comment supprimer, en une seule ligne, tous les fichiers de la structure hiérarchique /tmp contenant la chaîne 'bidon' ?

Traitement des fichiers

- **COMPTAGE DANS UN FICHIER**

La commande wc *(word count)*

wc [OPTION]... [FILE]...

Compte les lignes, les mots et les caractères contenus dans les fichiers ou entrés à partir du clavier si le nom de fichier est omis.
Un mot est une chaîne de caractères délimitée par des espaces, une tabulation ou des retours chariot.

Options:

-l nombre de lignes;
-w nombre de mots;
-c nombre de caractères.

Par défaut, les 3 options sont données.

Traitement des fichiers

- **COMPTAGE DANS UN FICHIER**

La commande wc (suite)

Exemples:

$ wc men

Affiche le nombre de lignes, de mots et de caractères du fichier men. Le nom du fichier est aussi affiché.

$ who | wc -l

Affiche le nombre d'utilisateurs connectés au système.

$ wc -l < men

Affiche uniquement le nombre de lignes que contient le fichier men.

AUTRES COMMANDES

HELHa
Haute École
Louvain en Hainaut

Autres commandes

- **INTRODUCTION**

  - Ce chapitre décrit d'autres commandes Linux très intéressantes.
  - Elles sont classées par ordre alphabétique

Autres commandes

- **LA COMMANDE *cal*** *(calendrier)*

cal [-mjy] [month [year]]

Affichage d ’un calendrier

Exemples:

$ cal 5 2004
Affiche le calendrier du mois de Mai de l'année 2004.

$ cal 04
Affiche le calendrier de l'année 4 (et non de l'année 2004 !).

$ cal | lpr
Envoie le calendrier du mois courant dans le spool de l'imprimante.

Autres commandes

- **LA COMMANDE *clear*** *(nettoyer)*

clear

Efface l ’écran.

Exemple:

$ clear
Efface l ’écran de votre terminal (virtuel ou non).

Remarques: man tput

$ tput clear
Efface l ’écran de votre terminal (virtuel ou non).

$ bold=`tput smso` offbold=`tput rmso`
Configure la police d ’affichage en ‘ inverse gras ’

$ echo "$bold Votre nom: $offbold"
Affichage de ‘ Votre nom: ’ en inverse gras

Autres commandes

- **LA COMMANDE *cut*** *(couper)*

cut [OPTIONS]... [FILE]...

Filtre permettant de saisir un champ (ou plusieurs) de chaque ligne d ’un ou de plusieurs fichiers.

Exemples:

$ cut -c1 /etc/passwd

Affiche toute la première colonne du fichier /etc/passwd.

$ cut -d: -f6 /etc/passwd

Affiche chaque 6ème champ du fichier /etc/passwd (le séparateur de champs est le :).

$ cut -d: -f1,5 /etc/passwd

Affiche les champs 1 et 5 de chaque ligne du fichier /etc/passwd.

$ cut -c2-5 /etc/passwd

Affiche les caractères 2 à 5 de chaque ligne du fichier /etc/passwd.

Autres commandes

- **LA COMMANDE *cut* (Suite)**

$ cut -d: -f3- /etc/passwd

Affiche chaque ligne du fichier /etc/passwd à partir de son 3ème champ inclus.

Atelier:

Entrez la ligne permettant d'afficher le nom du groupe de connexion d'un l'utilisateur donné (ex. jean) ?

Autres commandes

- **LA COMMANDE *cut* (Suite)**

Remarques:

$ who am i

Zeus!jean pts/0 Nov 10 12:18

$ who am i | cut -f 1 -d" "
Zeus!jean

$ who am i | cut -f 2 -d" "
Ici, rien ne s'affiche => -d" " ne remplace pas une suite indéfinie d'espaces !!!

Remède: La commande *gawk*

$ who am i | gawk '{ printf("%s\n",$2); }'

pts/0

✓ La commande gawk est très puissante (man gawk) mais beaucoup plus gourmande en ressources que la commande cut.
Ne l'utiliser que si nécessaire !

Autres commandes

- **LA COMMANDE *date*** *(date)*

date [OPTION] [MMDDhhmm[[CC]YY][.ss]]

Affiche la date et l'heure courante ou change la date et l'heure du setup.

Exemples:

$ date
Wed Nov 10 13:43:18 CET 2004

date 0507134710.45

Mise à jour de la date et l'heure système:
Le 7 mai 2010 à 13h47'45"

Autres commandes

- **LA COMMANDE *du*** *(disk usage)*

du [OPTION]... [FILE]...

Affiche une estimation de la quantité d'espace utilisée par des fichiers dans un répertoire.

Exemples:

$ du -sh /etc (avec s: summarize => totaliser
9.9M /etc h: human readable)

$ du -sb /etc (avec b: réponse en bytes)
10321920 /etc

$ du -b /etc
Donne la taille de tous les fichiers de la structure /etc.

✓ Un fichier qui a plusieurs liens n'est compté qu'une seule fois.

Autres commandes

- **LA COMMANDE *file*** *(type de fichier)*

file file ...

Permet de déterminer le type du fichier passé en paramètre.

Exemples:

$ file /dev/tty1
/dev/tty1: character special (4/1)

$ file /etc/passwd
/etc/passwd: ASCII text

$ file /etc/init.d/gpm
/etc/init.d/gpm: Bourne-Again shell script text [1]

$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 ...

$ file fichvide
fichvide: empty

[1] Le script doit commencer par le commentaire: `#!/bin/bash`

Autres commandes

- **LA COMMANDE *head*** *(tête)*

head [OPTION]... [FILE]...

Ce filtre affiche les premières lignes de chacun des fichiers spécifiés.

Exemples:

$ head /etc/termcap
Affiche les 10 premières lignes du fichier /etc/termcap.

$ head -n 3 /etc/termcap
Affiche les 3 premières lignes du fichier /etc/termcap.

Autres commandes

- **LA COMMANDE *id*** *(identifier)*

id [OPTION]... [USERNAME]

Pour vérifier votre identification ou celle d'un utilisateur et voir le groupe(s) d'appartenance.

Exemples:

$ id
uid=550 (alice) gid=100(users) groups=100(users), 6(disk)

Groupe effectif

$ id pgouwyjl
uid=1307(pgouwyjl) gid=2168(profs) groups=2168(profs)

Autres commandes

- **LA COMMANDE *sed*** *(Stream Editor)* man sed ...

Commande très puissante permettant de filtrer et transformer un texte

Quelques exemples pratiques:

1.N'afficher que les lignes 10 à 12 d'un fichier

```
$ sed -n '10,12p' fichier
```

→ ***-n****: Ecrit seulement les lignes spécifiées (par l'option p) sur la sortie standard*

2. Afficher la 1ère ligne d'un fichier

```
$ sed -n '1p' fichier
```

3. Remplacer une chaîne par une autre dans un fichier

```
$ sed -e "s/chaine1/chaine2/g" fic > fic.tmp && mv -f fic.tmp fic
```

→ ***-e:*** *Permet de spécifier les commandes à appliquer sur le fichier.*
Ici, il s'agit d'une commande de substitution (s/) globale (/g)

ou

```
$ sed –i -e "s/chaine1/chaine2/g" fic
```

→ ***-i:*** *La commande s'exécute au sein même du fichier (le fichier est modifié)*

Autres commandes

- **LA COMMANDE *sed*** *(Stream Editor) (Suite)*

Quelques exemples pratiques: (Suite)

4. Afficher les lignes non commentées d'un fichier

```
$ cat fichier.conf | sed -e '/^#.*$/d'
```

→ *-e: Permet de spécifier les commandes à appliquer sur le fichier.*

Ici, il s'agit d'une commande de suppression (/.... /d) des lignes correspondant à une expression régulière

^	*la ligne doit commencer par*
#	
.	*suivi de n'importe quel caractère*
*	*répété 0, 1 ou n fois*
$	→ *fin de l'expression régulière*

Autres commandes

- **LA COMMANDE *sed*** *(Stream Editor) (Suite)*

Quelques exemples pratiques: (Suite)

5. Commenter une ligne contenant le mot httpd :

```
$ sed -i -e '/.*httpd.*/ s/^/#/g' fichier.txt
```

→ *Ici, le début (^) de chaque ligne contenant la chaîne httpd (/.*httpd.*/) est substitué (s/ ... /g) à un commentaire (#).*

6. Insérer une nouvelle ligne sous/après une autre

```
Le fichier contient:
  lol
  ola
```

```
$ sed -i '/lol/a \client' fichier
```

→ *on insère le mot 'client' après (/a) le mot 'lol' (rem: /i pour insérer avant).*

7. Suppression d'une ligne commençant par le mot 'football'

```
$ sed -i -e '/^football/d' fichier
```


Autres commandes

- **LA COMMANDE *sed*** *(Stream Editor) (Suite)*

Pour beaucoup plus d'informations

http://www.commentcamarche.net/faq/9536-sed-introduction-a-sed-part-i

http://www.shellunix.com/sed.html

http://fr.wikipedia.org/wiki/Stream_Editor

http://openclassrooms.com/courses/la-commande-sed

Autres commandes

- **LA COMMANDE *sh*** *(shell)*

sh [-x] [args]

Cette commande permet d'exécuter les commandes se trouvant dans un fichier précisé via un appel à shell secondaire (/bin/sh).

args	nom de fichier accompagné d'éventuels paramètres positionnels
-x	affiche une trace des commandes durant l'exécution

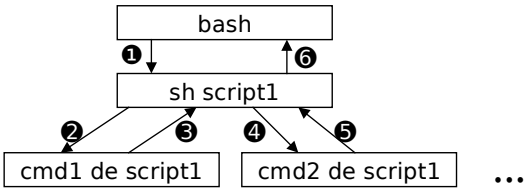
Exemples:

$ sh script1

Exécute script1 même s'il n'est pas dans le mode d'exécution *(x)*.

$ sh -x script1

Permet de tracer et d'exécuter les commandes de script1.



Autres commandes

- **LA COMMANDE *sh*** (suite)

Remarques:

- Souvent utilisée lors de la mise au point de scripts.
- N'importe quel shell Linux peut être appelé: sh - bash - csh - ash - bsh - tcsh ...

Autres commandes

- **LA COMMANDE *stty*** *(set tty)*

stty [-F device] [--file=device] [SETTING]...
stty [-F device] [--file=device] [-a|--all]

Permet de définir ou d'afficher les paramètres d'entrée/sortie d'un device.

Exemples:

$ stty → Résumé des caractéristiques du terminal courant.

```
speed 38400 baud; line = 0;
-brkint -imaxbel
```

$ stty -a → Toutes les caractéristiques du terminal courant.

```
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

Autres commandes

- **LA COMMANDE *stty*** (suite)

$ stty -F /dev/ttyS0 -a

Affichage des paramètres de transmission de la première interface série.

Signification de quelques caractéristiques:

speed 38400 baud	: vitesse de transmission en bauds.
intr = ^C	: touches d'interruption clavier d'un process lancé en avant plan
eof = ^D	: touches de fin de fichier clavier
susp = ^Z	: touches de mise en suspension d'un process lancé en avant plan.
echo	: saisie de touches avec echo
-echo	: saisie de touches sans echo

Modification d'une caractéristique:

$ stty -echo Quelle pourrait être une utilité pratique de cette commande?

Autres commandes

- **LA COMMANDE *su*** *(switch user)*

su [OPTION]... [-] [USER [ARG]...]

Permet de changer d'identité et ainsi d'exécuter un processus sous cette autre identité. Si cette commande est utilisée par root, aucun mot de passe ne sera demandé.

Exemples:

su louis
Prendre l'identité de louis.

su - louis
Prendre l'identité de louis en changeant l'environnement, c'est à dire en exécutant le fichier .bash_profile de louis.

su louis –c "cat /etc/shadow"
Exécute la commande cat avec l'identité de louis sans changer d'identité de départ.

☑ exit pour revenir à l'identité d'origine.

Autres commandes

- **LA COMMANDE *tail*** *(queue)*

tail [OPTION]... [FILE]...

Elle permet d'afficher les dernières lignes d'un fichier.

Exemples:

$ tail /etc/passwd
Affiche les 10 dernières lignes du fichier /etc/passwd.

$ tail -20 /etc/passwd
Affiche les 20 dernières lignes du fichier /etc/passwd.

$ tail --lines=+5 /etc/passwd
Affiche le fichier /etc/passwd à partir de la ligne 5.

Autres commandes

- **LA COMMANDE *tail*** (suite)

Remarque:

L'option -f demande à tail de ne pas s'arrêter lorsqu'elle a affiché les dernières lignes du fichier et de continuer à afficher la suite du fichier au fur et à mesure que celui-ci grossit jusqu'à l'interruption du processus par ^C.

```
$ tail  -f /var/log/messages
```

Permet de surveiller les connexions utilisateurs sur la machine.

```
$ tail  -f /var/log/secure
```

Permet de connaître les différents événements qui se produisent sur le système (impression, connexion à l'Internet, tâche de maintenance système...).

Autres commandes

- **LA COMMANDE *tee*** *(Duplication d'un flux de données)*

tee [OPTION]... [FILE]...

La commande tee permet de rediriger une commande tout en la transmettant à un tube.

Exemples:

```
$ cut  -d:  -f1,3  /etc/passwd | tee liste.txt | sort  -t:  +0  -1
```

Affichera à l'écran la liste des users suivit de leurs UID (*cut*), triée par ordre alphabétique (*sort*), tandis que la liste *non triée* sera redirigée vers le fichier liste.txt (*tee*). Cette redirection écrasera le contenu du fichier.
Avec l'option -*a* , la commande tee n' écraserait pas l'ancien fichier.

Autres commandes

- **LA COMMANDE *tee*** (suite)

```
$ sort ventes.dat | tee ventes.tri | mail -s "Données des ventes triées" sarah
```
Les données du fichier ventes.dat sont triées dans ventes.tri et envoyées en mail à sarah.

```
$ cat monps
screen=`tty`
ps  ax | tee $screen | wc -l

$ monps
.......
.......
38
```

Ce script permet d'afficher la liste et le nombre de processus en cours.

Autres commandes

- **ATELIER**

[gouwy] $ bash -i 2>&1 | tee /tmp/vue

[jean] $ tail -f /tmp/vue

[louis] $ tail -f /tmp/vue

A quoi pourrait servir la commande bash de 'gouwy' si 'jean' et 'louis' entrent par la suite la commande tail (comme indiquée ci-dessus) et contemplent leur terminal ? Expliquez.

Autres commandes

- **LA COMMANDE *tty*** *(teletypewriter)*

tty [OPTION]...

Affiche le nom du fichier connecté sur l'entrée standard (bien souvent le terminal de l'utilisateur).

L'option *-s* empêche l'affichage, autorisant à tester seulement le code de sortie.
Le code de sortie vaut 0 si le standard input est un terminal, 1 autrement.

Exemples:

```
$ tty
/dev/tty1

$ tty -s
$ echo $?
0
```

Autres commandes

- **LA COMMANDE *who*** *(qui est connecté)*

who [OPTION]... [FILE | ARG1 ARG2]

Affiche les utilisateurs actuellement connectés au système.

Exemples:

```
$ who
pgouwyjl   pts/0    Nov 12 14:53
root       pts/1    Nov 12 11:00
phainautp  pts/2    Nov 12 14:53

$ who -T

pgouwyjl   +   pts/0    Nov 12 14:53
root       +   pts/1    Nov 12 11:00
phainautp  +   pts/2    Nov 12 14:53
```

Les terminaux pts/0, pts/1, pts/2 sont accessibles en écriture (- dans le cas contraire).

Autres commandes

- **LA COMMANDE *who*** (suite)

```
$ who -q
pgouwyjl root phainautp
# users=3
```

Affiche la liste et le nombre de users connectés.

```
$ who -u
pgouwyjl  pts/0    Nov 12 14:53 00:06
root      pts/1    Nov 12 11:00    .
phainautp pts/2    Nov 12 14:53 00:06
```

Affiche les users connectés ainsi que le temps d'inactivité (en minutes).
Un point (.) signifie que l'utilisateur a utilisé le shell durant la dernière minute (autrement dit, on peut considérer qu'il est en train de travailler...).

```
$ who am i
Zeus!phainautp pts/2    Nov 12 14:53
```

Qui suis-je ?

Autres commandes

- **LA COMMANDE *who*** (suite)

Remarques:

- Le fichier /var/log/wtmp (binaire) contient l'historique des logins depuis son dernier nettoyage.

- $ who /var/log/wtmp

  Permet de le lire et de mettre en page sa visualisation.
  (La commande *last* s'en sert aussi pour mettre en page son affichage).

- Expérience: Vider ce fichier et réintroduire ces commandes.
  Que constatez-vous ?

Autres commandes

- **LA COMMANDE *w*** *(qui fait quoi?)*

w - [husfV] [user]

Affiche les utilisateurs actuellement connectés au système et ce qu'ils sont en train de faire.

Exemples:

```
$ w
 3:28pm  up 32 days,  3:30,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
pgouwyjl pts/0    213.213.198.142.  2:53pm  0.00s  0.06s  0.02s  w
root     pts/1    213.213.198.142. 11:00am  2:29   0.21s  0.02s  man w
```

Remarque:

`up 32 days, 3:30` signifie que le système n'a pas été redémarré depuis 32 jours 3 heures et 30 minutes (cette information peut être aussi obtenue par la commande ***uptime***).

Autres commandes

- **LA COMMANDE *watch*** *(répéter une commande)*

watch ... [-n <seconds>] ... <command>

Permet d'exécuter une commande toutes les x secondes.

Exemples:

[pgouwyjl] $ top

[phainaut] $ tail -f /var/log/messages

[root] # watch -n 2 'w'

```
Every 2s: w                                          Fri Nov 12 15:39:10 2004

  3:39pm  up 32 days,  3:41,  3 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
pgouwyjl pts/0    213.213.198.142.  2:53pm  1:43   0.29s  0.25s  top
root     pts/1    213.213.198.142. 11:00am 10.00s  0.25s  0.06s  watch -n 2 w
phainaut pts/2    213.213.198.142.  3:38pm 58.00s  0.05s  0.01s  tail -f /var/lo
```

Ici, la commande w (*qui fait quoi*) est exécutée toutes les 2 secondes sur pts/1.
Autrement dit, sur ce terminal, on voit "plus ou moins en temps réel" ce que font tous les utilisateurs du système...

Autres commandes

- **LA COMMANDE *whereis*** *(où parle-t-on de ?)*

whereis [-bmsu] ... filename ...

Localise le binaire, le source et les pages de manuel d'une commande.

Exemple:

```
# whereis cat
cat: /bin/cat    /usr/share/…/cat.1.gz
```

- **LA COMMANDE *which*** *(où est le binaire ?)*

which [options] [--] programname [...]

Affiche le chemin complet d'un binaire.

Exemples:

```
# which cat
 /bin/cat
```

Autres commandes

- **En vrac**

```
# cat /proc/cpuinfo          → informations techniques sur le processeur …

# cat /proc/meminfo
# free                       → informations techniques la Ram et son utilisation …
# vmstat
```

Comparaison de quelques commandes: Dos et Linux

	DOS	LINUX
Affichage du contenu d'un fichier	type	cat
Affichage du contenu d'un fichier page par page	more	more
Renommer un fichier	ren	mv
Déplacer un fichier	move	mv
Copier un fichier	copy	cp
Supprimer un fichier	del	rm
Recherche dans un fichier	find	grep
Changer de répertoire	cd	cd
Afficher le répertoire courant	cd	pwd
Créer un répertoire	mkdir	mkdir
Supprimer un répertoire	rmdir	rmdir
Lister les entrées d'un répertoire	dir	ls
Effacer l'écran	cls	clear
Manipulation de la date et de l'heure	time	date
Informations sur l'utilisation de la mémoire Ram	mem	free - vmstat
Aide sur une commande	/?	man

Linux Operating System

PROGRAMMATION DU BASH

HELHa
Haute École
Louvain en Hainaut

Neutralisation des métacaractères

Pour neutraliser des caractères ayant une signification particulière pour le shell: [ ] $ < > * ? | & ...

- **Le backslashe: **

  Neutralise le caractère suivant.

  ```
  $ echo Hello\>\>
  Hello>>
  $ echo Il fait beau aujourd\'hui
  Il fait beau aujourd 'hui
  ```

- **Le simple quote: '**

  Tous les caractères se trouvant entre ' sont neutralisés.

  ```
  $ echo 'Hello>>'
  Hello>>
  ```

Neutralisation des métacaractères (suite)

- **Le guillemet: "**

  Tous les caractères se trouvant entre " sont neutralisés sauf: $ ` \

  ```
  $ echo $?
  0
  $ echo '$?'
  $?
  $ echo "$?"
  0
  ```

Les commentaires dans un script

Les commentaires sont en ligne et commencent par #.

```
$ cat ex1
#!/bin/bash
# Exercice 1: Recherche d'une chaîne
# Auteur:      Gouwy JL
```

La commande echo

- Affiche le contenu d'une variable, d'une constante numérique ou alpha numérique.

```
$ echo $var          $ echo BONJOUR      $ echo 747
4                    BONJOUR             747
```

- Elle reconnaît certains caractères spéciaux:

\n	Saut de ligne.	\c	Annulation du saut de ligne.
\b	Retour arrière.	\0n	Code octal d'un caractère.
\t	Tabulation.		

```
$ echo -e "Votre nom: \c "      ou      echo -n "Votre nom: "
Votre nom: $

$ echo -e "Le système annonce\nune erreur grave\007"
Le système annonce
une erreur grave (+bip sonore)
```

La commande echo (suite)

- La gestion des couleurs:

Pour chaque code d'affichage, vous devez taper:

echo **-e "\033**[**x**m**...**"

Séquence d'escape

Message

N° de couleur

```
$ echo -e "\033[31mBonjour Monsieur\033[0m"
Bonjour Monsieur
$
```

Rouge

Couleur par défaut

Quelques couleurs:

Noir	0;30	Rouge	0;31	Bleu	0;34	Bleu clair	1;34	Jaune	1;33
Vert	0;32	Vert clair	1;32	Brun	0;33	Pourpre	0;35		
Cyan	0;36	Gris clair	0;37	Gris foncé	1;30				

(le 0 est optionnel mais pas le 1)

Les variables

- **Les variables utilisateurs**
  - Celles définies par l 'utilisateur.
  - Leur nom peut être composé de minuscules, chiffres et de caractères de soulignement.
  - Ne peuvent pas commencer par un chiffre.
  - La longueur du nom n 'a pas de limite théorique.

```
$ a=tomates        -> Affectation (la variable ne commence pas par $)
$ echo $a          -> Affichage (la variable est préfixée par $ car elle
tomates                est lue).
$ b=               -> b devient une variable définie mais vide.
$ b="ce sont des $a vertes"
$ echo $b
ce sont des tomates vertes
$ unset a          -> Suppression de la variable a.
$ readonly var     -> Protection en écriture et en suppression.
```

Les variables (suite)

- **Les variables utilisateurs (suite)**
  - $ **a=Listing** -> *Affectation (la variable ne commence pas par $)*
    $ **cp $a1 $a2** -> *Tentative de copie du fichier dont le nom est dans a1 vers un autre fichier dont le nom se trouve dans a2.*
    $ **cp ${a}1 ${a}2**
    -> *Tentative de copie d'un fichier s'appelant Listing1 vers un autre fichier Listing2.*
  - **echo ${#var}** -> *Affichage de la longueur de la variable var.*

- **Les variables systèmes**

  - Celles définies par le système Linux.

  - Une modification de l'une d'entre elles peut avoir des interférences sur toute une palette de programmes utilitaires ou d'applications qui y font appel. (ex. HOME, PATH, PS1, PS2, PWD, PPID, RANDOM, SECONDS ...)

Les variables (suite)

- **Les variables spéciales**

  - Ne sont accessibles qu'en lecture !

  - **$?** Valeur de retour de la dernière commande synchrone exécutée.

    $$ PID du shell actif.

    $! PID du dernier processus en arrière plan.

    $# Nombre de paramètres (arguments) passé au script.
    Mis à jour par la commande shift.

    $* Chaîne constituée de tous les paramètres d'appel au script (nom du script exclu). Elle est mise à jour par la commande shift.

```
$ cat Legumes
Cornichon
Chou_de_Bruxelles
Tomate
```

Les variables (suite)

- **Les variables spéciales (suite)**

```
$ grep Poire Legumes
$ echo $?
1

$ grep Tomate Legumes > /dev/null
$ echo $?
0

$ echo $$
732
$ ls -lR / > /tmp/liste.$$ 2> /dev/null &
$ kill -9 $!

$ imprime test1 test2 test3
Lors du <Enter>  $0=imprime  $1=test1  $2=test2  $3=test3
                 $#=3  $*=test1 test2 test3
```

Les variables (suite)

- **Les paramètres positionnels**
  - De $0 à $9 maximum.
  - $0 contient le nom du script et $1 et suivant les paramètres d'appel.
  - La commande *set* permet de redéfinir ces parmètres.
  - La commande *shift* permet de décaler d'une position vers la gauche tous les paramètres d'appel ($0 n'est pas écrasé !).
  - shift n permet un décalage de n positions.

```
$ cat script
while [ $# -ne 0 ]  # Tant qu'on n'a pas exploité tous les param.
do
   echo $1          # Affichage du paramètre courant.
   shift            # Décalage vers la gauche.
done

$ script p1 p2 p3
p1
p2
p3
```

Les variables (suite)

- **Les paramètres positionnels (suite)**

...

set Tomates Cornichons Betteraves

... *-> Les paramètres positionnels sont redéfinis par ces valeurs.*
+ MAJ de $# et $.*

...

set a* *-> Pour prendre comme paramètres tous les noms de fichiers commençant par la lettre 'a'.*

Substitution de commandes

Commande placée entre ` et remplacée ensuite par ses sorties à travers le canal de sortie.

- $ **pwd**
  /home/gouwy
  $ **je_suis_la=`pwd`** *ou* $ **je_suis_la=$(pwd)**
  $ **echo $je_suis_la**
  /home/gouwy

- $ **tout=`ls -lR / 2> /dev/null`**
  $ **echo $tout**
  L'ensemble de la sortie de la commande ls -lR est placée dans la variable tout.

- $ **utilisateurs=`who | cut -d ' ' -f1 | sort`**
  La variable 'utilisateurs' contiendra le nom de tous les utilisateurs connectés au système.

La commande test

Elle renvoie une valeur de retour 0 si la condition testée est vraie et différente de 0 si la condition testée est fausse. C'est le contraire du C.

- Tests sur les fichiers

```
$ test -e file          S'agit-il d'une entrée dans le répertoire ?
$ echo $?
0

$ test -f /etc/passwd   S'agit-il d'un fichier ordinaire ?
$ echo $?
0

$ test -c /dev/tty0     S'agit-il d'un fichier spécial caractère ?
$ echo $?
0

$ test -b /dev/fd0      S'agit-il d'un fichier spécial bloc ?
$ echo $?
0
```

La commande test (suite)

- Tests sur les fichiers (suite)

```
$ z=fich
$ test -s $z        S'agit-il d'un fichier d'au moins un octet ?
$ echo $?
0

$ test -p fpipe     S'agit-il d'un fichier pipe ?

$ test -r fich      L'utilisateur à l'origine de la commande a-t-il
                    le droit de lecture sur fich ?
```

(**-x** = droit d'exécution **-w** = droit d'écriture)

```
$ test -r xyz && echo "Le fichier xyz est lisible"
Le fichier xyz est lisible

$ a=fich
$ test -s "$a" || echo "Le fichier $a est vide"
$
```

La commande test (suite)

- Tests sur les chaînes

$ **test -z "$a"** *La variable a est-elle vide ?*

$ **test -n "$a"** *La variable a est-elle garnie ?*

$ **test "$a" = Isat** *La variable a contient-elle la chaîne Isat ?*

$ **test "$a" != Isat** *La variable a ne contient-elle pas la chaîne Isat ?*

$ **test "$a" != "$b"** *Le contenu caractère des 2 variables est-il différent ?*

$ **test "$a" \< "$b"** *Comparaison selon l'ordre Ascii du contenu des 2 variables (autre opérateur: >).*

- Tests numériques

$ **Longueur=5**

$ **test "$Longueur" -lt 10 && echo "La longueur est ok"**

La longueur est ok

Opérateurs de comparaison numérique

-eq	Egal	**-lt**	Inférieur à	**-le**	Inférieur ou égal
-ne	Différent	**-gt**	Supérieur à	**-ge**	Supérieur ou égal

La commande test (suite)

- Combinaison de tests

Types d'opérateurs logiques

! Négation -a ET logique -o OU logique

$ **test ! -s "$a"** *Le fichier dont le nom se trouve dans a est-il vide ?*

$ **test -d "$fichier" -a -x "$fichier"**

Vrai s'il s'agit d'un répertoire et si ce répertoire dispose des autorisations d'accès en exécution.

$ **test -c "$1" -o -b "$1" && echo "Fichier device"**

Affichage du message s'il s'agit d'un fichier de type caractère ou de type bloc

$ **test \(-d "$a" -a -x "$a" \) -o \(-f "$a" -a -r "$a" \)**

Vrai s'il s'agit d'un répertoire muni de l'autorisation d'accès en exécution ou s'il s'agit d'un fichier ordinaire muni de l'autorisation d'accès en lecture.

La commande test (suite)

- Forme abrégée de la commande test

Le mot test est simplement remplacé par [ ]

Cette forme abrégée est implémentée dans le shell

=> pas de processus enfant

$ **test -d /etc**	ou	$ **[-d /etc]**
$ **test "$a" != Isat**	ou	$ **["$a" != Isat]**
$ **test "$Ln" -lt 10**	ou	$ **["$Ln" -lt 10]**

La commande if

Elle permet de vérifier une condition et de prendre des actions suivant que la condition est vérifiée ou non.

- Première forme

```
if condition
then                    -> C'est l'alternative à une branche.
    Commande(s)
fi
```

- Deuxième forme

```
if condition
then
    Commande(s)         -> C'est l'alternative à deux branches.
else
    Commande(s)
fi
```

La commande if (suite)

- ```
  if grep  "arthur" /etc/passwd > /dev/null 2>&1
  then
          echo "arthur est un utilisateur du système"
  fi
  ```

- ```
  if grep   "$1" /etc/passwd > /dev/null 2>&1
  then
          echo "$1 est un utilisateur du système"
  else
          echo "$1 n'est pas un utilisateur du système"
  fi
  ```

- ```
  if rm -f  "$a"  2> /dev/null
  then
          echo "Le fichier $a est supprimé"
  else
          echo "Impossible de supprimer le fichier $a"
  fi
  ```

La commande if (suite)

- Les if imbriqués

```
if cond1
  then  trait1
  else   if cond2
         then trait2
         else  if cond3
               then trait3
               fi
          fi
fi
```

ou

```
if cond1
  then  trait1
  elif   cond2
         then trait2
         elif   cond3
                then trait3
fi
```

☑ La forme thif n'existe pas !


```

La commande case

- Il s'agit d'une commande permettant un branchement multiple suivant la valeur d'une variable ou d ’une substitution de commande.
- Elle permet l'emploi de métacaractères (* ? [...] [! ...] |) pour redéfinir le critère.

```
case valeur in
    critère1)  commande(s);;
    critère2)  commande(s);;
    ...
esac
```

La commande case (suite)

```
case  $#  in
    0)      echo "$0: Erreur: Arguments absents" >&2
            exit 1;;
    1)      ...
            ... ;;
    2)      ... ;;
    *)      echo "$0: Erreur: Trop d'arguments" >&2
            exit 2;;
esac
```

```
case  $LOGNAME  in
    root)     PS1="#  ";;
    hugo|victor)       PS1="Hello, ici $LOGNAME  $  ";;
    [A-Z]*)            PS1="Hello $ ";;
    *)                 PS1="$ ";;
esac
export PS1
readonly PS1
```

La commande case (suite)

```
case  $reponse  in
      [yYoO]*)      echo "Ok! ce sera fait";;
      [nN]*)        echo "tant pis";;
      *)            echo "ERREUR: Saisie invalide" >&2;;
esac
```

La commande read

- Elle lit une ligne sur le canal 0.

Le premier mot est assigné à la première variable.
Le deuxième mot est assigné à la seconde variable.
Les mots excédant le nombre requis sont assignés à la dernière variable.

```
$ cat script
echo -e "Entrez votre nom et votre prénom : \c"
read  nom1  nom2
echo  "Votre nom est $nom1"
echo "Votre prénom est $nom2"
```

```
$ read a  b  c
Rien ne sert de courir
$ echo $a
Rien
$ echo $b
ne
$ echo $c
sert de courir
$
```

Les boucles

- **La boucle for**

  - Elle permet d'exécuter plusieurs fois les mêmes commandes.

```
for  variable  [ in  liste ]
do
        Commande(s)
done
```

$ **cat monmail**

```
# Envoi d'un courrier électronique à 3 utilisateurs
for  name  in  hugo  victor  jules
do
        mail  $name  <<  Fin
        Rendez-vous au restaurant ce soir à 17h00
Fin
done
```

$ **cat editautoc**

```
# Edition de tous les fichiers sources C du répertoire courant
for  fichier  in  *.c
do
    mcedit $fichier
done
```

Les boucles (suite)

- **La boucle for (suite)**

$ **cat snifintru**

```
# Recherche pour tous les users connectés de leur UID
for  name  in  `who | cut -d" " -f1`
do
   noutil=`grep "$name" /etc/passwd 2> /dev/null | cut -d: -f3`
   if  [ -z "$noutil" ]
   then
        echo "Oops !" >&2
   fi
   echo "$name: $noutil"
done
```

$ **cat editandsave**

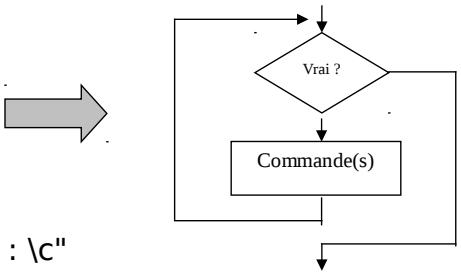
```
# Edition et recopie de tous les fichiers passés en arguments
for  fichier
do
    mcedit ${fichier}
    cp ${fichier} $HOME/sauvegarde
done
```

Les boucles (suite)

- **La boucle while**

  Elle permet d'exécuter la même séquence de commandes tant qu'une condition spécifiée est vraie.

```
while  condition
do
     commande(s)
done
```




```
echo  -e "Nom de fichier : \c"
read  fichier
while  [  -z  "$fichier"  ]
do
        echo "ERREUR: Pas de saisie"
        echo  -e "Nom de fichier: \c"
        read fichier
done
```

Les boucles (suite)

- **La boucle while (suite)**

```
$ cat suppressauto
while true                  #  Boucle sans fin
do
     echo -e "Nom de fichier : \c"
     read nom
     if [ -z "$nom" ]       # Si Return, on sort du script ...
     then
          exit 0
     fi
     echo "Supprimer fichier  \"$nom\"  (o/n) ?"
     read reponse
     if [ "$reponse" = o ]
     then
          rm -f $nom
     fi
done
```

Les boucles (suite)

- **Lecture d'un fichier texte ligne par ligne**

```
$ cat lecture.sh
fich="/etc/passwd"
cat $fich | while read ligne
            do
                  echo $ligne
                  sleep 1
            done
```

ou

```
fich="/etc/passwd"
while read ligne
do
         echo $ligne
         sleep 1
done < $fich
```

Les boucles (suite)

- **Les tableaux**

```
$ cat maniptab
i=0                          #  Garnissage
while [ $i –le 9 ]
do
   tab[$i]=$(($i + 100))
   i=$(($i + 1))
done

i=0                          #  Lecture
while [ $i –le 9 ]
do
   echo ${tab[$i]}
   i=$(($i + 1))
done

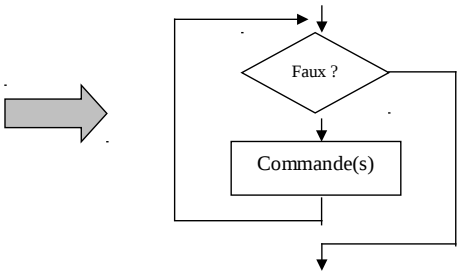
echo ${tab[*]}   # Ensemble des éléments du vecteur
echo ${#tab[5]}  # Longueur d'un élément
echo ${#tab[*]}  # Nombre d'éléments dans le tableau
```

Les boucles (suite)

- **La boucle until**

  Elle permet d'exécuter la même séquence de commandes tant qu'une condition spécifiée est fausse.

```
until  condition
do
     commande(s)
done
```




```
until test -f fich     # On sort de la boucle dès que fich existe.
do
    sleep 300          # On attend 5 minutes.
done
```

Les boucles (suite)

- **La boucle until (suite)**

```
$ cat suppressauto
until false                 #  Boucle sans fin
do
     echo -e "Nom de fichier : \c"
     read nom
     if [ -z "$nom" ]       # Si Return, on sort du script ...
     then
          exit 0
     fi
     echo "Supprimer fichier  \"$nom\"  (o/n) ?"
     read reponse
     if [ "$reponse" = o ]
     then
          rm -f $nom
     fi
done
```

La commande expr

- Elle réalise des calculs et des manipulations de nombres et retourne le résultat par le canal de sortie standard.

- Opérateurs notamment disponibles: + - * %

- $ **expr 4 + 3** ou **echo $((4+3))**
  7

- $ **a=`expr 6 \* 7`**
  $ **echo $a**
  42

- $ **cat boucle**

```
a=10
while [ $a -ge 1 ]
do
   echo $a
   a=`expr $a - 1`
done
```

La commande eval

- Une ligne de saisie du shell n'est traitée qu'une seule et unique fois. Mais si une variable contient le nom d'une autre variable, un unique niveau d'exploitation est insuffisant.
- La commande *eval* permet une exploitation multiple de la ligne de commandes.

- $ **a=Tomates**
  $ **b=a**
  $ **echo $b**
  a
  $ **eval echo \$$b**
  Tomates

- $ **cat struct**

```
for  n  in prenom  nom  adresse
do
        echo -e "$n: \c" # Lecture des valeurs de ces 3 variables
        read  $n         # à partir de ces mêmes variables.
done
echo  "Les valeurs suivantes ont été lues : "
for  n  in  prenom  nom  adresse
do
        eval  echo $n: \$$n
done
```

La commande :

- Elle retourne toujours la valeur 0 et elle est interne contrairement à la commande *true*.

```
while :
do
   ...
done
```

Les alias

- C'est une abréviation représentant une commande ou un groupe de mots dans une ligne de saisie.
- Ils seront souvent définis dans le fichier /etc/profile et/ou dans le fichier $HOME/.bash_profile.

```
$ alias  md=mkdir
$ alias  ls='ls --color'
$ alias  sshzeus='ssh 212.68.208.163'

$ alias              # Liste des alias prédéfinis.

$ unalias  md        # Suppression de l'alias md.
```

Les extensions de la commande cd

- La variable OLDPWD contient le nom du répertoire dans lequel on se trouvait avant le dernier appel de cd.

⊙ $ **cd -** On passe dans le répertoire dont le nom est contenu dans OLDPWD.

⊙
$ **pwd**
/etc
$ **cd ~/prog** ~ équivaut au $HOME.
$ **pwd**
/home/gouwy/prog
$ **cd ~victor** ~ suivi d'un nom d'utilisateur
$ **pwd** => connexion directe à sa home directory.
/home/victor

La commande select (suite)

- Pour créer des menus simples.

select Variable in Listedevaleurs
do
Sériedecommandes
done

⊙ $ **cat menu1**

```
PS3="Numéro:  "
select name in victor hugo erwin quitter
do
  if  [ -n "$name" ]
    then
         if [ $name = quitter ]
         then
              break       # On sort du select
         fi
         echo  "C'est $name qui a été choisi"
  fi
done
```

La commande select (suite)

```
$ sh menu1
1)victor
2)hugo
3)erwin
4)quitter
Numéro:
```

- La liste des valeurs est affichée par stderr. Devant chaque mot est placé un numéro. Puis le contenu de la variable PS3 est affiché, dans l'attente d'une saisie. Si l'expression select reçoit un chiffre correspondant à un des numéros de la liste, la variable prend la valeur du mot concerné et la boucle est parcourue.

- Lorsque la saisie est vide, la liste des mots est recréée. En cas de saisie erronée, la variable est affectée d'un texte vide et la suite des commandes est exécutée.

- Si l'expression select n'a pas de liste de valeurs alors elle sera traitée avec les paramètres de position passés lors de l'appel du script.

En vrac

```
$ basename  /tmp/seb/liste
liste

$ dirname  /tmp/seb/liste
/tmp/seb
```

`$ seq 3`	`$ seq 12 14`	`$ seq -w 10`
1	12	01
2	13	02
3	14	03 ...

```
for i in `seq -w 10`
do
        adduser sisat2ig${i} ...
done
```

Les fonctions

- Les scripts sont exécutés en tant que processus enfant => aucune répercussion des modifications des variables de l'enfant dans le shell parent.

- Les scripts sont recherchés à l'aide de la variable *PATH.*

- Les fonctions du shell seront exécutées dans le shell courant et seront enregistrées comme des variables et ne seront donc pas recherchées à l'aide de la variable PATH.

- Les fonctions apparaissent lors de l'utilisation de la commande 'set'.

- Lors de l'exécution d'une commande, le shell vérifie d'abord s'il s'agit d'un alias , puis d'une fonction , puis d'une commande interne du shell avant de tenter de charger et d'exécuter une commande externe.

Les fonctions (suite)

- $ **monll()** Définition en mode direct de la fonction monll.

```
>{<cr>
>   /bin/ls -l $* | more<cr>
>}<cr>
$
```

- $ **declare -f** Liste des fonctions déclarées.

```
monll()
{
   /bin/ls -l  $* | more
}
$
```

- $ **declare -fx monll** Exportation de la fonction monll qui devient
  $ **declare -f** donc reconnue dans les sous-shells.

```
 monll()
{
   /bin/ls -l  $* | more
}
declare –fx monll
$
```

Les fonctions (suite)

- Comment récupérer ses fonctions dans son shell de connexion ?

Les enregistrer dans le .bash_profile et/ou à partir de ce shell, exécuter un fichier de fonctions à l'aide de la commande.

```
$ cat sh.funcs
monll()
{
  /bin/ls -l $* | more
}
monrm()
{
 for i
 do
   echo -e "Supprimer $i (o/n) ? \c"
   read reponse
   case ${reponse} in
      [oOyY]*) /bin/rm -f $i;;
   esac
 done
}
```

```
$ . sh.funcs
$ monrm essai
Supprimer essai (o/n) ?
```

Les fonctions (suite)

- Modularisation de scripts à l'aide de fonctions:

```
$ cat test.sh
#!/bin/bash
test_fact()
{
 if [ $1 -lt 0 ]; then
    return 1
 else
    return 0
 fi
}

fact()
{
 n=$1
 res=1
 while [ $n -gt 1 ]; do
    res=`expr ${res} \* ${n}`
    n=`expr ${n} - 1`
 done
 echo ${res}
 return 0
}

echo -n "Une valeur: "
read val
if test_fact ${val}; then
  echo "Factoriellle de ${val} = `fact ${val}`"
else
  echo "Valeur incorrecte"
fi
```

```
$ sh test.sh
Une valeur: 6
Factorielle de 6 = 720
$ sh test.sh
Une valeur: -2
Valeur incorrecte
$
```