

# Cours des Systèmes d'Exploitation LINUX

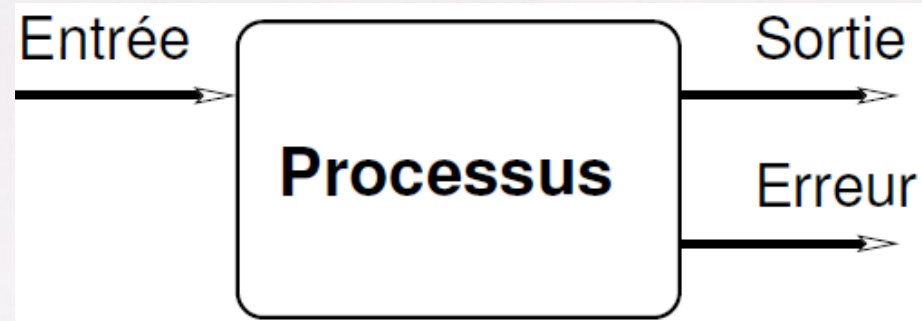
ENSAM – Casablanca  
2020-2021

# Chapitre 5

## Boite à outils de commandes

# Redirections

Par défaut, tout processus a accès à trois canaux de communications :



1. le flux de sortie (**Eng** - *standard output*) par défaut dirigé vers le terminal courant.
2. le flux d'erreur (**Eng** - *standard error*) également dirigé par défaut vers le terminal courant.
3. le flux d'entrée (**Eng** - *standard input*) sur lequel transitent par défaut les données provenant du clavier.



# Redirections

Il est possible de rediriger les trois flux présentés grâce à une syntaxe particulière qui varie très légèrement selon le shell utilisé.

## Redirection du flux de sortie

On peut rediriger le flux de sortie d'une vers un fichier .

Le caractère > permet la redirection dans le fichier test.txt (si le fichier existe déjà il sera vidé de son contenu).

Si on désire ajouter au fichier un autre résultat de commande on utilise >> :

```
smi@ubuntu:~$ echo Bonjour > test.txt
smi@ubuntu:~$ cat test.txt
Bonjour
smi@ubuntu:~$ echo SMI >> test.txt
smi@ubuntu:~$ cat test.txt
Bonjour
SMI
```

# Redirections

## Redirection du flux d'erreur

On peut dans certaines situations vouloir récupérer les informations envoyées par un programme sur le flux d'erreur. Le principe est le même que pour le flux de sortie, seul l'opérateur est différent (le flux d'erreur est symbolisé par 2) :

```
smi@ubuntu:~$ qlqchose
qlqchose: command not found
smi@ubuntu:~$ qlqchose 2>test.txt
smi@ubuntu:~$ cat test.txt
qlqchose: command not found
```

On peut également coupler les deux redirections de la manière suivante :

**unprogramme > sortie.txt 2> erreur.txt**

Pour rediriger les deux flux (sortie et erreur) dans un même, on pourra utiliser :

**unprogramme > sortie-erreur.txt 2>&1**

Ou plus simple :

**unprogramme >& sortie-erreur.txt**

# Redirections

## Redirection du flux d'entrée

Lorsqu'un programme attend des données depuis le clavier, on peut lui fournir directement ces données en utilisant le contenu d'un fichier. On dit alors qu'on redirige le **flux d'entrée** du programme en question. Le premier exemple que nous proposons utilise la commande **bc** qui est d'après la page de manuel, *an arbitrary precision calculator language*. Une session avec **bc** se présente comme suit :

```
smi@ubuntu:~$ bc -q
30*78+3
2343
quit
smi@ubuntu:~$ gedit calcul.txt
smi@ubuntu:~$ cat calcul.txt
30*78+3
smi@ubuntu:~$ bc < calcul.txt
2343
```

On dit qu'on a redirigé le flux d'entrée de **bc** depuis le fichier calcul.txt



# Redirections

## Le trou noir

On trouve généralement sur les systèmes Unix un fichier particulier dont le nom est **/dev/null** : c'est un « trou noir » qui absorbe toute redirection sans broncher. Ce fichier est un fichier spécial, il ne grossit pas lorsqu'on y redirige des données. Il peut être utile pour ne pas être dérangé par les messages d'erreur par exemple :

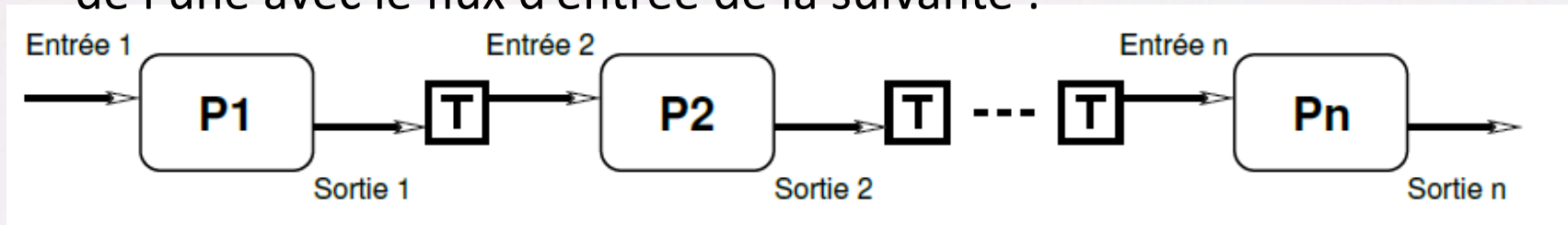
```
id user > test 2> /dev/null
```

cette commande écrit dans le fichier test les numéros identifiant l'utilisateur user.

Si l'utilisateur n'existe pas le message d'erreur n'est pas affiché puisqu'envoyé dans **/dev/null**.

# Les tubes (pipes)

Les tubes ou pipes offrent un mécanisme permettant de composer plusieurs commandes en connectant le flux de sortie de l'une avec le flux d'entrée de la suivante :



Par exemple, la commande `whoami` fournit l'utilisateur courant comme **sortie**. La commande `id` fournit des informations sur un utilisateur en **entrée**.

On utilise la sortie de la première commande comme entrée de la seconde à travers le symbole `|`

```
smi@ubuntu: ~  
smi@ubuntu:~$ whoami  
smi  
smi@ubuntu:~$ id smi  
uid=1000(smi) gid=1000(smi) groups=1000(smi),4(adm)  
6(plugdev),107(lpadmin),124(sambashare),2047(TP)  
smi@ubuntu:~$ whoami | id  
uid=1000(smi) gid=1000(smi) groups=1000(smi),4(adm)  
6(plugdev),107(lpadmin),124(sambashare),2047(TP)  
smi@ubuntu:~$
```



# Les filtres

Toute utilisation « sérieuse » du shell d'Unix passe par une utilisation fréquente des tubes et redirections. Les utilitaires présentés dans la suite de ce chapitre peuvent se voir comme des filtres, c'est-à-dire que l'on peut considérer qu'ils filtrent le flux de sortie des programmes auxquels ils s'appliquent. En d'autres termes, un filtre peut être utilisé avec un tube puisqu'il attend systématiquement des données sur son flux d'entrée et envoie le résultat du « filtrage » sur son flux de sortie.

# Le tri - sort

La commande **sort** permet de trier les lignes d'un ensemble de données en entrée (par défaut par rapport à la première colonne):

```
smi@ubuntu: ~  
smi@ubuntu:~$ cat etudiants.txt  
Ahmed jbali 1995  
Zineb filali 1994  
Samir talssi 1997  
Fahd arrabi 1996  
Mohammed charfi 1995  
smi@ubuntu:~$ sort etudiants.txt  
Ahmed jbali 1995  
Fahd arrabi 1996  
Mohammed charfi 1995  
Samir talssi 1997  
Zineb filali 1994  
smi@ubuntu:~$ █
```

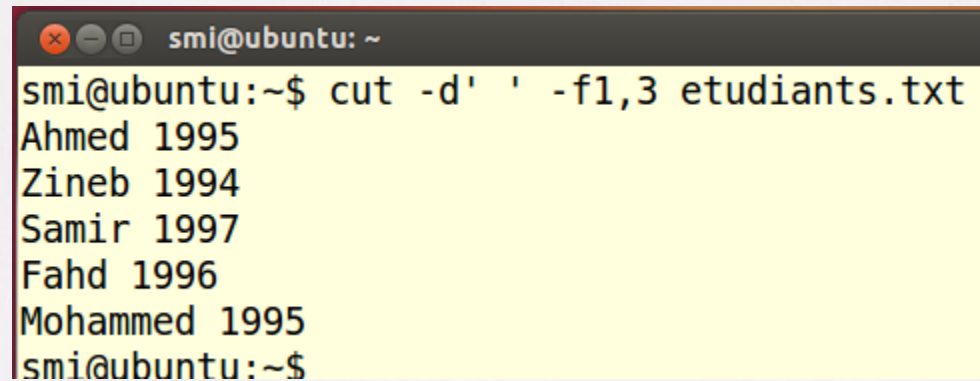
```
smi@ubuntu: ~  
smi@ubuntu:~$ sort -k3 etudiants.txt  
Zineb filali 1994  
Ahmed jbali 1995  
Mohammed charfi 1995  
Fahd arrabi 1996  
Samir talssi 1997  
smi@ubuntu:~$
```

L'option **-k** pour *key* en anglais spécifie la clef du tri. Pour trier selon les dates l'option **-k3** est utilisée.

Notez l'utilisation de l'option **-n** qui permet d'utiliser l'ordre numérique (par défaut c'est l'ordre lexicographique). L'option **-u** peut quant à elle être utilisée pour supprimer les doublons

# Découper en colonnes - cut

La commande **cut** permet d'afficher une ou plusieurs colonnes d'un flux de données :

A terminal window titled 'smi@ubuntu: ~' showing the execution of the 'cut' command. The command 'cut -d' ' -f1,3 etudiants.txt' is entered, and the output displays the first and third fields of the 'etudiants.txt' file, separated by spaces. The output lines are: 'Ahmed 1995', 'Zineb 1994', 'Samir 1997', 'Fahd 1996', and 'Mohammed 1995'. The prompt 'smi@ubuntu:~\$' is shown at the bottom.

```
smi@ubuntu:~$ cut -d' ' -f1,3 etudiants.txt
Ahmed 1995
Zineb 1994
Samir 1997
Fahd 1996
Mohammed 1995
smi@ubuntu:~$
```

Le séparateur de colonnes est par défaut la tabulation. On peut en spécifier un autre avec l'option **-d** (pour délimiteur). Ici on a utilisé le caractère espace comme délimiteur.

On a ensuite spécifié les champs (option **-f** pour field) numéro 1 et numéro 3 (respectivement prénom et date).

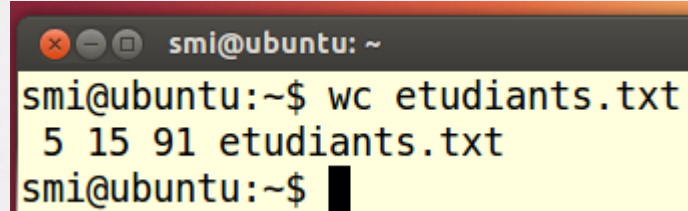
Voici pour s'amuser un exemple d'utilisation des tubes :

```
cut -d' ' -f1,3 etudiants.txt | sort -k2 -n
```



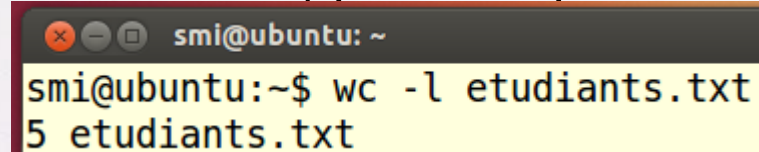
# Compter - wc

La commande `wc` (pour word count) permet de compter les caractères, les mots et les lignes d'un flux de données :



```
smi@ubuntu: ~  
smi@ubuntu:~$ wc etudiants.txt  
5 15 91 etudiants.txt  
smi@ubuntu:~$
```

Ce qui signifie que le fichier en question possède 5 lignes, 15 mots et 91 octets (les espaces et sauts de ligne sont comptabilisés). On peut également afficher uniquement l'un de ces nombres, par exemple le nombre de lignes :



```
smi@ubuntu: ~  
smi@ubuntu:~$ wc -l etudiants.txt  
5 etudiants.txt
```

**wc -l** <nom\_du\_fichier> # affiche le nombre de lignes

**wc -c** <nom\_du\_fichier> # affiche le nombre de bytes

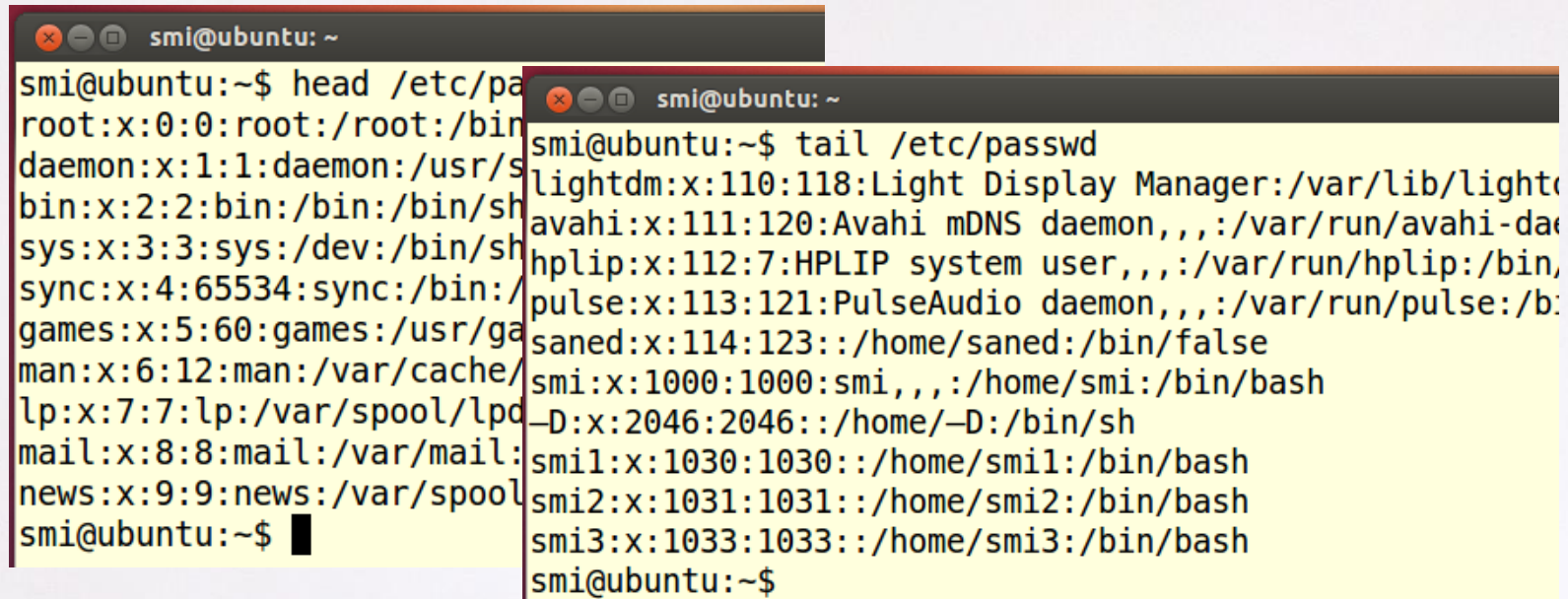
**wc -m** <nom\_du\_fichier> # affiche le nombre de caractères

**wc -L** <nom\_du\_fichier> # indique la longueur de la plus longue ligne

**wc -w** <nom\_du\_fichier> # affiche le nombre de mots

# Tête et queue – head –tail

Les commandes **head** et **tail** comme leur nom l'indique, permettent d'afficher la tête ou la queue d'un fichier ou du flux d'entrée. Par exemple :



```
smi@ubuntu:~$ head /etc/passwd
root:x:0:0:root:/root:/bin/
daemon:x:1:1:daemon:/usr/s
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/
games:x:5:60:games:/usr/ga
man:x:6:12:man:/var/cache/
lp:x:7:7:lp:/var/spool/lpd
mail:x:8:8:mail:/var/mail:
news:x:9:9:news:/var/spool
smi@ubuntu:~$

smi@ubuntu:~$ tail /etc/passwd
lightdm:x:110:118:Light Display Manager:/var/lib/lightdm:
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:
hplip:x:112:7:HPLIP system user,,,:/var/run/hplip:/bin/
pulse:x:113:121:PulseAudio daemon,,,:/var/run/pulse:/bin/
saned:x:114:123:./home/saned:/bin/false
smi:x:1000:1000:smi,,,:/home/smi:/bin/bash
-D:x:2046:2046:./home/-D:/bin/sh
smi1:x:1030:1030:./home/smi1:/bin/bash
smi2:x:1031:1031:./home/smi2:/bin/bash
smi3:x:1033:1033:./home/smi3:/bin/bash
smi@ubuntu:~$
```

Si on veut afficher un certain nombre d'octets :

`head -c 7 /etc/passwd`

Un certain nombre de lignes :

`head -n 2 /etc/passwd`

Options similaire pour tail



# Recherche de fichier - find

Pour chercher un fichier ou un répertoire on peut utiliser la commande **find**. Cette commande est très puissante et permet de chercher des fichiers avec de très nombreux critères (type de fichier, nom, taille, permission, propriétaires, date de modifications, etc.). Chacun de ces critères peut être combiné par des « ou » ou des « et ». Nous proposerons ici juste quelques exemples :

```
find ~ -name core
```

Cette commande cherche les fichiers dont le nom est core à partir du répertoire personnel ~.

```
find . -name "*.tex"
```

Celle-ci trouve les fichiers dont l'extension est .tex à partir du répertoire courant «. ».

```
find ~/cours -type d -a -name "po*"
```

trouve tous les répertoires dont le nom commence par po. L'option -a couple les critères par un « et ». Cette option est implicite et peut donc être omise.



## Recherche de fichier - locate

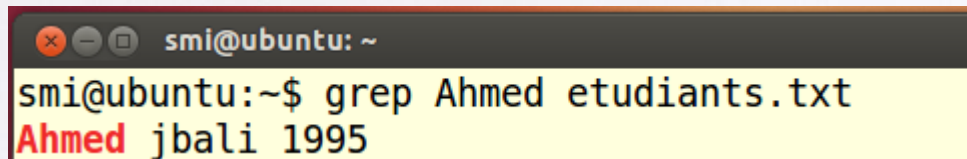
La commande **find** peut-être particulièrement coûteuse lorsqu'on doit scanner toute l'arborescence. C'est pour cette raison que certains Unix proposent la commande **locate** qui utilise un fichier de données contenant l'emplacement de tous les fichiers sur le système. Ce fichier de données doit être mis à jour régulièrement (généralement une ou deux fois par jour) par **cron**. L'avantage de cette méthode est bien entendu la rapidité de la recherche, l'inconvénient réside dans le fait que le fichier de données n'est pas à jour en temps réel et ne contient pas d'autres informations que les noms des fichiers (et pas leur contenu).

# grep et la notion d'expressions régulières

La commande **grep** est une commande « célèbre » d'unix et permet d'afficher les lignes d'un flux correspondant à un motif donné.

Le comportement habituel de **grep** est de recevoir une expression rationnelle en argument, de lire les données sur l'entrée standard ou dans une liste de fichiers, et d'écrire les lignes qui contiennent des correspondances avec l'expression rationnelle sur la sortie standard.

Par exemple, **grep** permet de chercher les lignes contenant le motif dans le fichier `etudiants.txt` :

A terminal window with a dark title bar showing 'smi@ubuntu: ~'. The command 'smi@ubuntu:~\$ grep Ahmed etudiants.txt' is entered. The output 'Ahmed jballi 1995' is displayed, with 'Ahmed' highlighted in red and 'jballi' underlined.

```
smi@ubuntu: ~  
smi@ubuntu:~$ grep Ahmed etudiants.txt  
Ahmed jballi 1995
```

# grep et la notion d'expressions régulières

Les motifs en question (Ahmed et 199) dans les deux exemples précédents se nomment en jargon Unix une expression régulière (ou une expression rationnelle ) (regular expression parfois abrégé en regexp). L'étude de la théorie des expressions régulières pourrait faire l'objet d'un document de la taille de ce guide de survie. Voici cependant sous forme d'un tableau à quoi correspondent quelques-uns des motifs d'une expression régulière.

.	désigne n'importe quel caractère	B
*	zéro ou plusieurs fois l'élément précédent	B
?	zéro ou une fois l'élément précédent	E
+	une ou plusieurs fois l'élément précédent	E
^	correspond au début de la ligne	B
\$	correspond à la fin de la ligne	B
[abc]	un caractère parmi abc	B
[^abc]	tout caractère sauf a, b ou c	B
{n}	exactement n fois l'élément précédent	E
{n,m}	au moins n fois et au plus m fois l'élément précédent <sup>5</sup> .	E



# grep et la notion d'expressions régulières

La version de grep utilisée pour l'exemple est celle de chez gnu. Cette commande comprend trois ensembles d'expressions régulières (basique, étendue et Perl). Dans le tableau précédent les deux premiers ensembles sont indiqués respectivement par B et E dans la colonne de droite.

.	désigne n'importe quel caractère	B
*	zéro ou plusieurs fois l'élément précédent	B
?	zéro ou une fois l'élément précédent	E
+	une ou plusieurs fois l'élément précédent	E
^	correspond au début de la ligne	B
\$	correspond à la fin de la ligne	B
[abc]	un caractère parmi abc	B
[^abc]	tout caractère sauf a, b ou c	B
{n}	exactement n fois l'élément précédent	E
{n,m}	au moins n fois et au plus m fois l'élément précédent <sup>5</sup> .	E