Commandes shell

Patrick MARCEL, Université d'Orléans

L2 Outils du développeur — S3

Commande simple

```
$ LANG=fi_FI.UTF-8 TZ=Asia/Tokyo date +%A
Perjantai
```

Une **commande simple** est une séquence :

- d'affectations de **variables d'environnement**,
- suivie d'une **commande à exécuter**,
- suivie d'**arguments**.

Commande à exécuter

La commande à exécuter est donnée par le **premier mot** qui ne soit pas une affectation (sans =).

Si la commande contient un / elle indique le **chemin** d'un fichier exécutable.

Sans / le shell essaie dans l'ordre :

- le nom d'une **fonction shell** ;
- une **commande interne** ;
- un **exécutable** dans un des répertoires du **PATH**.

Un **processus** est créé pour exécuter la commande.

Exécution en cascade

```
$ cat /etc/group | grep root | wc -l
      11
```

Une **cascade** est une suite de commandes séparées par des | (**pipe**).

La **sortie** de la première commande est connectée à l'**entrée** de la seconde et ainsi de suite.

Les commandes sont exécutées en **parallèle**, chacune dans son processus.

Descripteurs de fichiers

Pour interagir, le processus possède une liste de **descripteurs de fichiers** (*file descriptor*) qui pointent vers les **fichiers ouverts**.

Les premiers ont une signification particulière et contiennent une valeur par défaut :

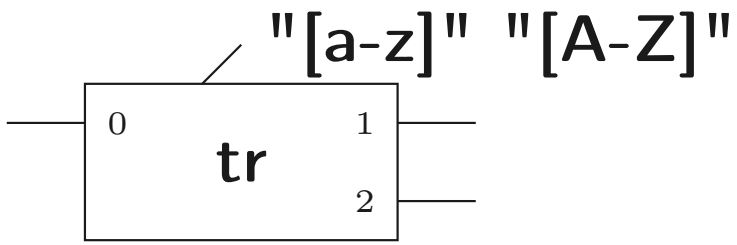
- **0** est l'**entrée standard** (**stdin**) : clavier
- **1** est la **sortie standard** (**stdout**) : écran
- **2** est la **sortie d'erreur** (**stderr**) : écran

Le modèle de la boîte

Pour représenter un processus et ses interactions, nous utiliserons un modèle symbolique de boîte qui met en valeurs :

- le **programme** ;
- ses **arguments** ;
- ses **entrées/sorties**.

Nous utiliserons la représentation ci-dessous :

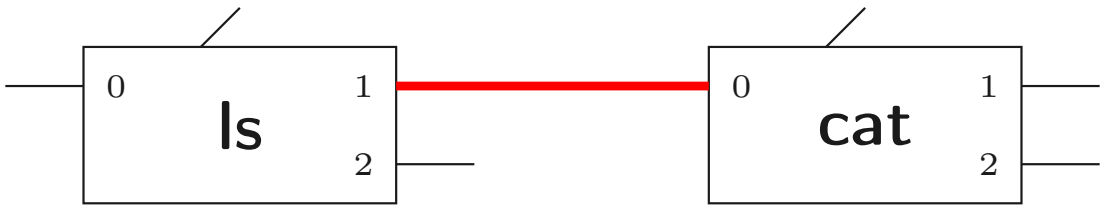


Tuyau (pipe)

On appelle **filtre** une commande qui transforme un contenu lu sur l'entrée standard en un résultat émis sur la sortie standard.

Le **pipe** du shell (tube en français) "branche" la sortie standard d'une commande sur l'entrée standard d'une autre.

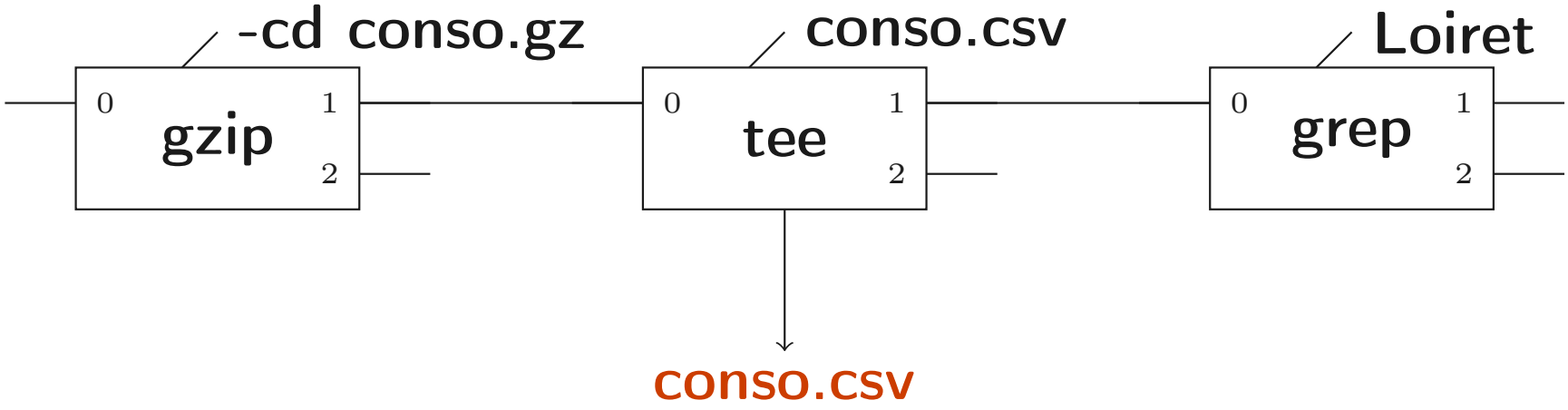
```
$ ls | cat
...
```



tee

La commande **tee *file*** recopie l'entrée standard sur la sortie et en fait une copie dans le fichier **file**.

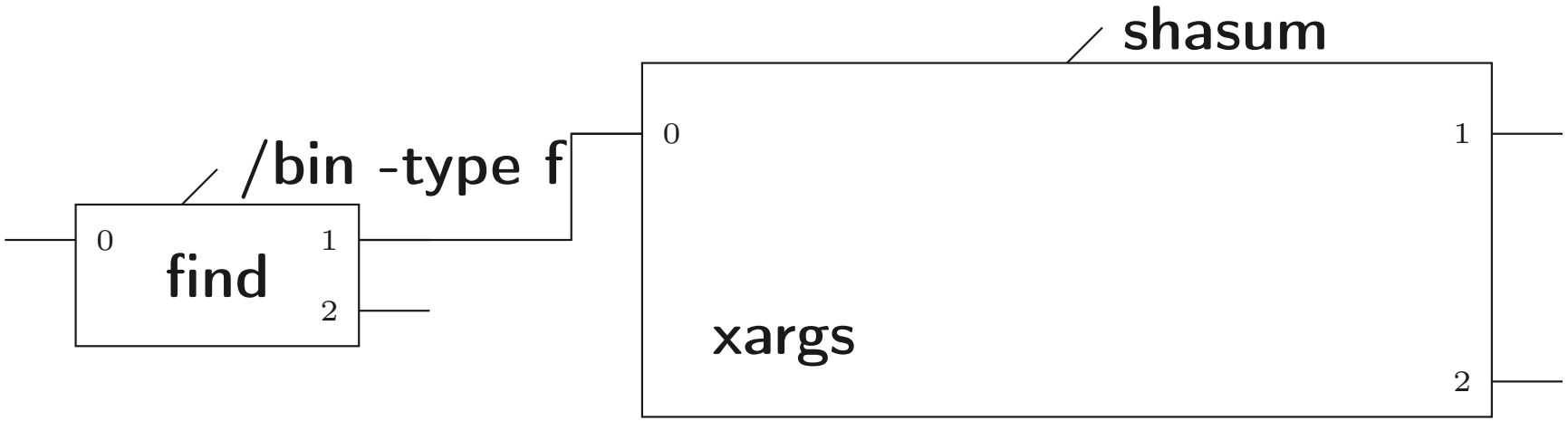
```
$ gzip -cd conso.gz |tee conso.csv |grep Loiret
...
2014-06-19T02:30:00+02:00;19/06/2014;02:30;41250;
```



xargs

La commande **xargs** transforme son entrée standard en arguments d'une commande.

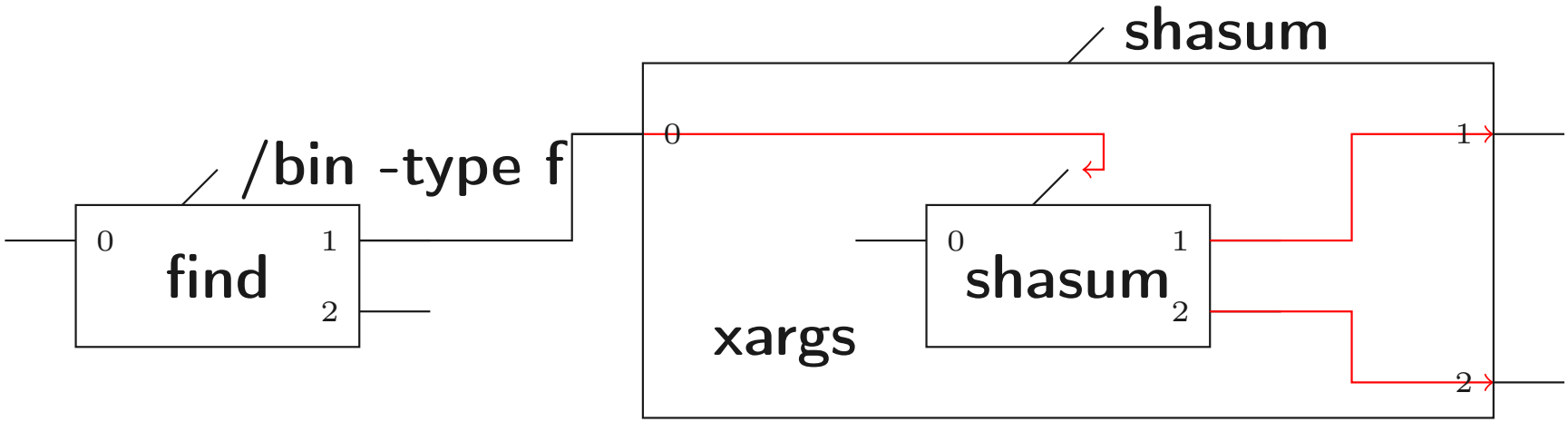
```
$ find /bin -type f | xargs shasum
0b486ff102197556b4e8c107d2aa5a4071363ef7 /bin/[
c429dd5f27c6b17061307a85f48ec492c865dc73 /bin/bash
0b73e354a968aa55c0c16890274396ce269dfd40 /bin/cat
d2974bf18bbe2a3e31ff8536cf5b442c51fdc062 /bin/chmod
...
```



xargs

La commande **xargs** transforme son entrée standard en arguments d'une commande.

```
$ find /bin -type f | xargs shasum
0b486ff102197556b4e8c107d2aa5a4071363ef7 /bin/[
c429dd5f27c6b17061307a85f48ec492c865dc73 /bin/bash
0b73e354a968aa55c0c16890274396ce269dfd40 /bin/cat
d2974bf18bbe2a3e31ff8536cf5b442c51fdc062 /bin/chmod
...
```



Liste de commandes

Une liste de commandes ou de cascades peut être exécutée grâce aux opérateurs `;`, `&`, `&&` ou `||`.

L'opérateur `;` désigne une exécution **séquentielle inconditionnelle**.

```
$ sleep 3; echo coucou
```

L'opérateur `&` désigne une exécution **inconditionnelle en arrière-plan** de la première commande.

```
$ sleep 3& echo coucou
```

Valeur de retour

Tout **processus** s'arrête avec une **valeur de retour**, un entier compris entre 0 et 255.

```
$ true; echo $?
0
$ false; echo $?
1
```

L'opérateur `&&` (resp. `||`) désigne une exécution **conditionnelle ET** (resp. **OU**) dépendant de la **valeur de retour** de la première commande.

```
$ ls backup || mkdir backup
```

Redirections

Il est courant de vouloir **rediriger** les sorties standard ou d'erreur dans des fichiers ou inversement de rediriger un fichier vers l'entrée standard. On parle alors de **redirection**.

Pour cela, il suffit de remplacer la valeur située à la position `0`, `1` ou `2` de la liste des descripteurs de fichiers par une nouvelle valeur.

Redirection de la sortie

```
$ echo 'Are you my mummy?' > question
```

La **sortie standard** est redirigée vers un fichier grâce à l'opérateur > suivi du nom de fichier à ouvrir en écriture.

Si le fichier n'existe pas il est créé. S'il existe son contenu est remplacé par la sortie de la commande.

L'opérateur peut être précédé d'un entier pour rediriger un autre descripteur.

L'opérateur >> permet d'**ajouter en fin de fichier**.

Redirection de l'entrée

```
$ tr a-z n-za-m < question
Aer lbh zl zhzzl?
```

L'**entrée standard** est redirigée vers un fichier grâce à l'opérateur `<` suivi du nom de fichier à ouvrir en lecture.

L'opérateur peut être précédé d'un entier pour rediriger un autre descripteur

Document en ligne

```
$ cat > greetings <<FIN
> Bonjour !
> Je m'appelle $USER.
> FIN
```

L'opérateur << suivi d'un **délimiteur** permet de rediriger sur l'entrée standard le texte jusqu'à une ligne contenant uniquement le délimiteur.

Plus de redirections

Il est également possible de dupliquer ou fermer les descripteurs de fichiers à l'aide des opérateurs suivants :

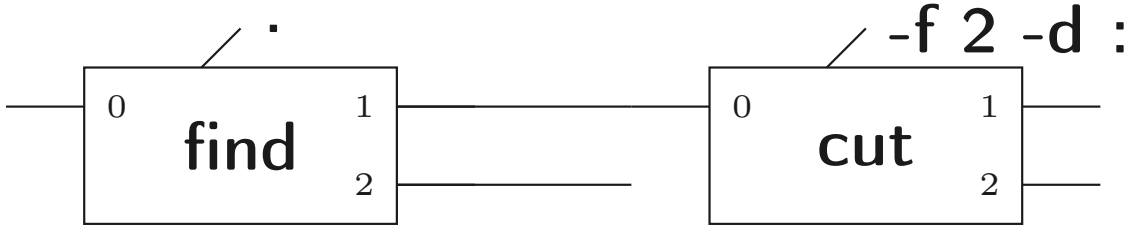
- *n1*`>&`*n2* copie le descripteur d'indice $n1$ vers celui d'indice $n2$. (si $n1$ n'est pas précisé, il s'agit de la sortie standard) ;
- *n1*`<&`*n2* similaire mais pour l'entrée ;
- `>&-` et `<&-` ferment respectivement la sortie standard, l'entrée standard.

Note : seul le premier est couramment utilisé.

Exemple

L'**ordre** des redirections est important.

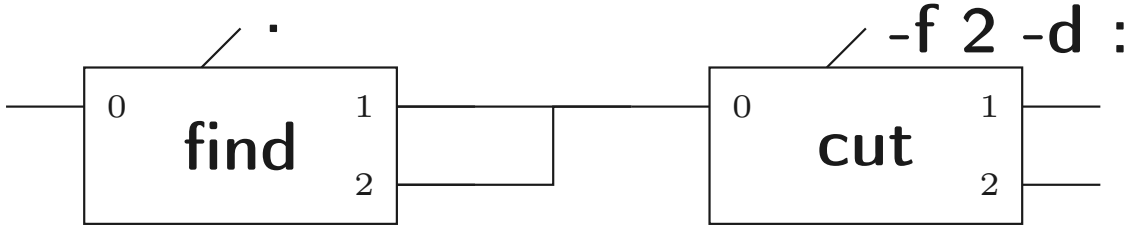
```
$ find . 2>&1 > /dev/null | cut -f 2 -d:
./.fseventsd
./.Spotlight-V100
./.Trashes
...
```



Exemple

L'**ordre** des redirections est important.

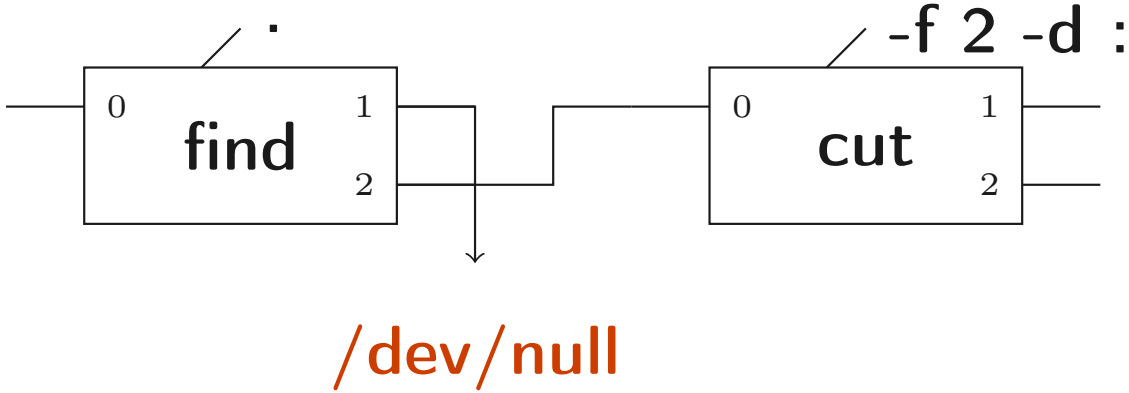
```
$ find . 2>&1 > /dev/null | cut -f 2 -d:
./.fseventsd
./.Spotlight-V100
./.Trashes
...
```



Exemple

L'**ordre** des redirections est important.

```
$ find . 2>&1 > /dev/null | cut -f 2 -d:
./.fseventsd
./.Spotlight-V100
./.Trashes
...
```



Ajouter des descripteurs de fichier

Seul 0, 1 et 2 sont prédéfinis...

On peut si besoin en ajouter :

```
$ echo “bla bla” > montfichier.txt
$ exec 3< monfichier.txt
$ cat <&3
bla bla
$ exec 3<&-
$ exec 3> monfichier.txt
$ echo hello >&3
$ exec 3>&-
$ cat monfichier.txt
hello
$
```

On peut aussi ouvrir en lecture/écriture (<>)

Commande composée

```
$ { head -3 lapin; echo poule;
tail -n +4 lapin ;} > lapin.new
```

Les accolades permettent de **grouper** des commandes.

```
$ (head -3 lapin; echo poule;
tail -n +4 lapin ;) > lapin.new
```

Les parenthèses exécutent les commandes dans un **sous-shell**.

```
$ (cd /; ls) ; ls
bin dev [...]
Documents Bureau [...]
$
```