INF203 - Flux d'entrée / sortie standard

Victor Lambret

2017

Les 3 flux

Entrée / sortie

Sous Unix chaque processus qui s'exécute est créé avec 3 flux :

- L'entrée standard
- La sortie standard
- La sortie d'erreur

Qu'est-ce qu'un flux ?

- Un flux est un descripteur de fichier ouvert par défaut à la création du processus.
- L'entrée standard est en lecture seule
- La sortie standard et la sortie d'erreur sont en écriture seule

Valeurs

Nom	I/O stream	file descriptor
entrée standard	stdin	0
sortie standard	stdout	1
sortie d'erreur	stderr	2

L'entrée standard

- Dans Unix c'est le fichier depuis lequel le processus lit ses données d'entrée.

Exemple de fonction C utilisant l'entrée standard :

```
int getchar(void);
```

Exemple d'instruction shell lisant l'entrée standard :

```
read variable
```

La sortie standard

Dans Unix la sortie standard est le fichier dans lequel le processus écrit ses résultats.
Exemple de fonction C utilisant la sortie standard :

```
printf("hacker vaillant rien d'impossible !\n");
fprintf(stdout, "hacker vaillant rien d'impossible !\n");
```

Exemple d'instruction shell utilisant la sortie standard :

```
echo "no place like $HOME"
```

La sortie d'erreur

A quoi sert cette seconde sortie ?

- Aux affichages d'erreurs pour l'utilisateur
- Aux affichages de debug
- Aux affichages de diagnostique

Cette seconde sortie laisse la possibilité de traiter autrement ces informations (c'est souvent utile)

Exemples de sorties d'erreurs

En C on peut préciser un flux pour certaines fonctions :

```
fprintf(stderr, "All your bases are belong to us");
```

En shell c'est le flux où s'affichent les erreurs :

```
user@machine:~$ ./main
bash: ./main: Permission non accordée
```

Comportement par défaut

Quand on lance une commande dans un interpréteur shell par défaut :

- L'entrée standard lit les données saisies depuis le clavier
- La sortie standard et la sortie d'erreur s'affichent dans le shell

Redirection de flux

Rediriger l'entrée standard en shell

Voici le programme `add.sh` qui additionne 2 entiers

```
#! /bin/bash

read NUMBER1
read NUMBER2
echo $(( NUMBER1 + NUMBER2 ))
```

Rediriger l'entrée standard

Par défaut avec ce programme l'utilisateur doit saisir les nombres à additionner au clavier

```
user@machine:~$ ./add.sh
3
4
7
```

Rediriger l'entrée standard

On peut créer un fichier `dataFile` qui contient les deux nombres :

```
3
4
```

Et utiliser l'opérateur < pour indiquer au shell d'utiliser le fichier comme entrée standard de la commande

```
user@machine:~$ ./add.sh < dataFile
5
```

Notes importantes

Bien que l'opérateur < soit spécifique au shell, la redirection est possible pour tout type d'exécutables : scripts shell, code compilé, etc.
On peut placer la redirection également en début de commande :

```
user@machine:~$ < dataFile ./add.sh
5
```

Rediriger la sortie standard

Pour rediriger la sortie standard il existe 2 possibilités

- On la redirige vers un nouveau fichier
- On souhaite ajouter la sortie du programme à un fichier existant.

Redirection simple

Pour cette redirection on utilise l'opérateur > comme ceci :

```
user@machine:~$ ls -l > contenuDisque.txt
```

- Si le fichier n'existe pas il est créé
- Si le fichier existe son contenu est écrasé

Redirection en mode ajout

Pour cette redirection on utilise l'opérateur >> comme ceci :

```
user@machine:~$ ls -l TP1 > ~/contenuDisque.txt
user@machine:~$ ls -l TP2 >> ~/contenuDisque.txt
user@machine:~$ ls -l TP3 >> ~/contenuDisque.txt
```

- Si le fichier n'existe pas il est créé
- Si le fichier existe la sortie du programme est ajoutée à la fin du fichier

Redirection de la sortie d'erreur

On utilise > et >> en les préfixant du numéro associé à la sortie d'erreur (2)

```
user@machine:~$ cp a 2> logError.txt
user@machine:~$ cp a 2>> logError.txt
```

Note : ça marche aussi pour la sortie standard, les deux commandes suivantes sont équivalentes :

```
user@machine:~$ ls -l > listeFichiers.txt
user@machine:~$ ls -l 1> listeFichiers.txt
```

/dev/null

Des fois on veut juste conserver la sortie standard en ignorant les erreurs. Sous Unix on peut le faire en redirigeant la sortie d'erreur vers un fichier spécial du système : `/dev/null`

```
user@machine:~$ cp a 2> /dev/null
```

`/dev/null` peut-être vu comme un trou noir du système : c'est un fichier dans lequel on peut écrire mais qui ne conserve pas son contenu

Redirection entre programmes

Le besoin

- Comme on l'a vu les opérateurs <, > et >> permettent d'effectuer des redirections avec des fichiers.
- Souvent on trouve très pratique d'enchaîner plusieurs programmes. Cela signifie rediriger la sortie d'un programme vers l'entrée d'un autre programme.
- En shell on effectue cette opérateur à l'aide de l'opérateur |

Note : l'opérateur | se prononce pipe (tube en anglais)

Exemple de besoin

On souhaite savoir combien de TP d'INF203 sont déjà passé. Pour cela on peut :

1. Lister le répertoire INF203
2. Dans ce résultat conserver uniquement les TPs
3. Dans ce résulat compter le nombre de lignes

Construisons la commande y répondant en utilisant des pipes

1ère étape

On liste le répertoire courant avec `ls`. La sortie comporte plusieurs lignes :

```
lambret@IBMdebian:~/test$ ls -1
memo_bash.pdf
td1
td2
td3
tp1
tp2
tp3
```

Seconde étape

On utilise sur cette sortie la commande `grep` afin de ne conserver que les lignes qui correspondent à un TP.
Note : l'option `-i` permet d'ignorer la casse

```
lambret@IBMdebian:~/test$ ls -1 | grep -i TP
tp1
tp2
tp3
```

Troisième étape

la commande `wc` compte des mots par défaut. En lui passant l'option `-l` elle compte des lignes à la place.

```
lambret@IBMdebian:~/test$ ls -1 | grep -i TP | wc -l
3
```

Qu'est-ce qu'un pipe ?

- Comme l'indique son nom anglais, un pipe est un tuyau.
- Ce qui est connecté à son entrée est redirigé vers sa sortie.
- L'entrée et la sortie standard sont des fichiers. Puisqu'on peut les manipuler avec un pipe cela signifie que l'entrée et la sortie du pipe sont aussi des fichiers.

Note importante

- Un concept d'Unix est : "tout est fichier"
- Un objet Unix est un fichier si on accède à la ressource via les fonctions {`open`, `read`, `write`, `close`}
- Il ne faut donc pas concevoir les fichiers Unix comme uniquement des fichiers stockés sur le disque.

Pour aller plus loin

Historique

- Les flux tels qu'on les utilise ici on été spécifiés dans les années 1970 dans la première version d'Unix.
- Les tubes ont été introduit plus tard dans la troisième version d'Unix (1973).
- On continue de les utiliser car ce sont des interfaces simples et efficaces.

En dehors d'Unix

- Les concepts présentés ici ne sont pas spécifiques au shell, ce sont des services fournis par le système d'exploitation.
- On retrouve les notions de flux standard en dehors d'Unix, par exemple dans le monde Windows.
- Sous Unix ce sont des interfaces historiquement privilégiées par les programmes car le shell fourni des outils puissants pour les manipuler
- Sous Windows, elles sont moins répandues car la philosophie n'est pas d'exposer à l'utilisateur la mécanique du système