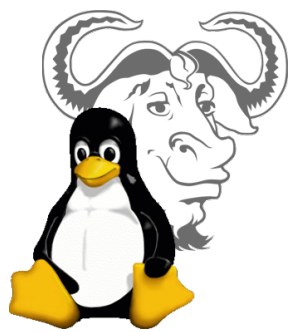
Administration d'un système GNU / Linux

03 — Entrées, sorties, et *pipes*

Anthony Labarre

上海师范大学

23 octobre 2024



Plan d'aujourd'hui

1. Entrées et sorties
2. Les filtres
3. Les *pipes*

Entrées et sorties

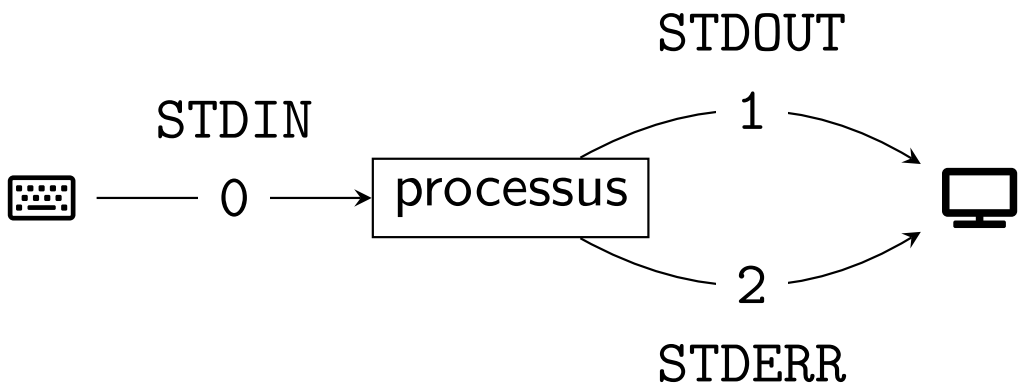
Processus

- Rappel: un **processus** est une instance d'un programme en cours d'exécution;
- En général, ils reçoivent et produisent des données;
- On va voir comment faire communiquer les processus;
- Cela revient à manipuler leurs **entrées** et **sorties**;

Entrées et sorties

- Un processus manipule deux types de flux:
  1. des **entrées**: les données qu'il reçoit;
  2. des **sorties**: les données qu'il produit;
- Nous allons voir aujourd'hui comment exploiter ces entrées et sorties;

Entrées et sorties standard



- En plus des entrées et sorties d'un processus, il existe trois types d'entrées / sorties "généraux", qui sont numérotés:
  0. l'**entrée standard** (ou `STDIN`) est le clavier (`/dev/stdin`);
  1. la **sortie standard** (ou `STDOUT`) est l'écran (`/dev/stdout`);
  2. l'**erreur standard** (ou `STDERR`) est aussi l'écran (`/dev/stderr`);

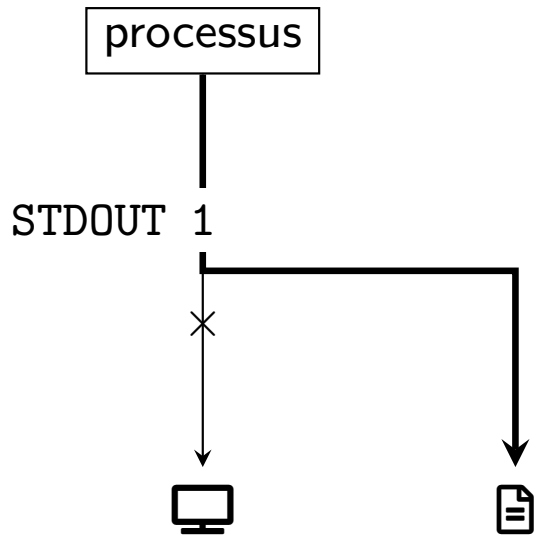
Redirection des sorties: >

- La plupart des commandes affichent leur résultat sur la sortie standard;
- Comment faire pour enregistrer ce résultat dans un fichier?
- Il suffit de **rediriger** sa sortie avec l'opérateur >;

Exemple

```
$ ls -l 1> contenu_répertoire.txt
```

- `processus 1> sortie` redirige ce que `processus` devrait écrire sur `STDOUT` vers le fichier `sortie`;
- Si `sortie` existe, son contenu est remplacé; sinon, `sortie` est créé par la commande;



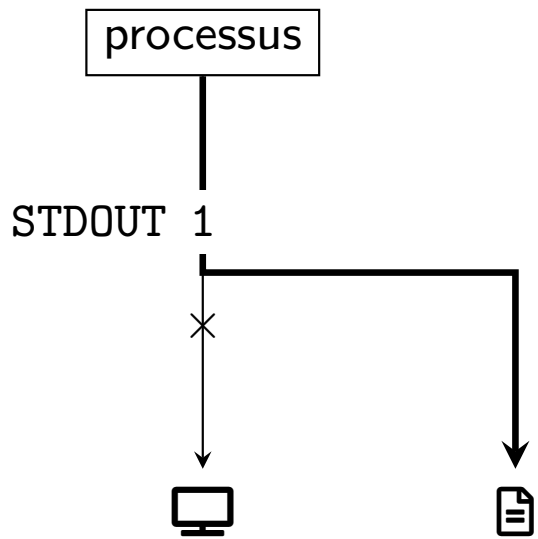
Redirection des sorties: >>

- Comment faire si l'on veut **ajouter** un résultat à une sortie?
- On utilise >> au lieu de > pour la redirection;

Exemple

```
$ ls -l 1> contenu_répertoire.txt
$ ls -l .. 1>> contenu_répertoire.txt
```

- La sortie de la seconde commande est rajoutée à la fin de `contenu_répertoire.txt`;



Redirection de l'erreur standard

On peut rediriger la sortie standard et l'erreur standard vers **deux fichiers distincts**: il suffit de préciser le numéro de la sortie à rediriger.

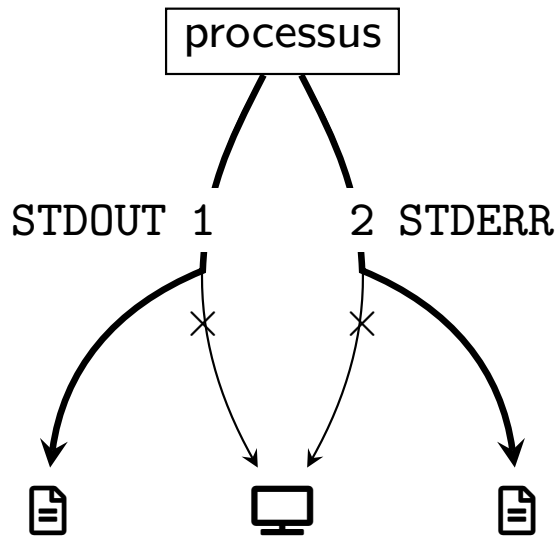
Exemple

(du = **d**isk **u**sage: affiche l'espace consommé par un répertoire.)

```
$ du -sh /
# erreurs
# espace utilisé sur le disque
```

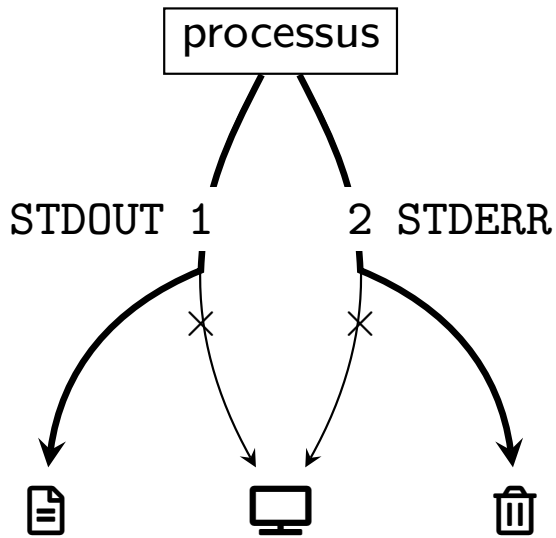
```
$ du -sh / 1> consommation.txt
# erreurs
# espace -> consommation.txt
# espace utilisé sur le disque
```

```
$ du -sh / 1> consommation.txt 2> log.txt
# aucun message
# espace -> consommation.txt
# erreurs -> log.txt
```



/dev/null: la poubelle

- Et si on ne veut ni voir les erreurs, ni les placer dans un fichier?
- On peut rediriger l'erreur standard vers `/dev/null`.



Exemple

```
$ du -sh / 1> consommation.txt 2> /dev/null
# espace dans consommation.txt;
# erreurs à la poubelle
```

Le `1` n'est pas nécessaire, car c'est une valeur par défaut: on peut écrire `processus > sortie` au lieu de `processus 1> sortie`.

Redirection des entrées: <

- Si l'on veut utiliser une autre entrée que le clavier, on utilise l'opérateur <;
- On est obligé de le faire avec les programmes qui ne travaillent qu'avec l'entrée standard;

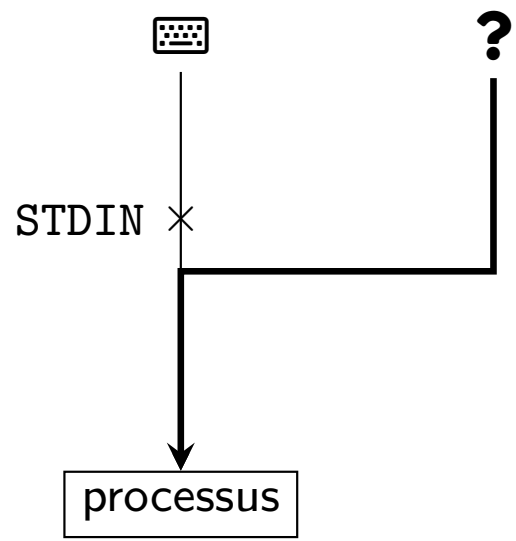
Exemple

Pour installer un nouveau système identique à l'ancien, on peut enregistrer les noms des paquets installés:

```
$ dpkg --get-selections > paquets
```

et puis les réinstaller sur le nouveau système:

```
$ sudo dpkg --set-selections < paquets
$ sudo apt-get dselect-upgrade
```

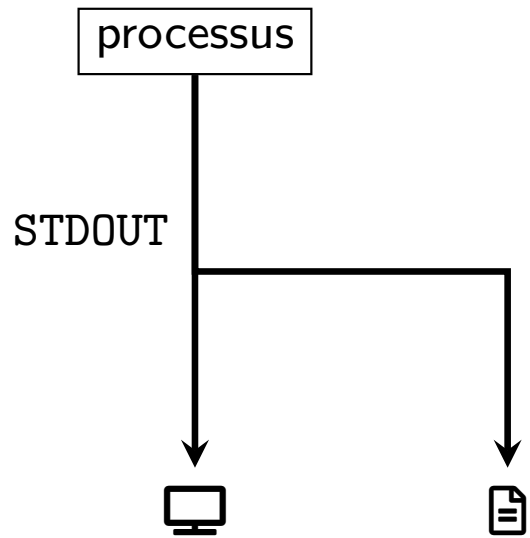


tee

- Quand on redirige la sortie standard, on ne voit plus ce que produit le processus;
- Le programme `tee` permet de régler ce problème: on voit la sortie, et on l'enregistre en même temps dans un fichier;

Exemple

```
$ ls -l | tee sortie.txt
# équivalent de >
$ ls -l | tee -a sortie.txt
# équivalent de >>
```



- Le symbole | correspond à un *pipe* (= tuyau), qu'on va maintenant expliquer;

Les filtres

Les filtres

- Avant de parler des *pipes*, examinons quelques **filtres** qui leur seront utiles;
- Ce sont des programmes parfois très basiques modifiant les données avant de les communiquer à un autre processus;
- Nous allons voir ensemble quelques-uns des filtres les plus fréquents;

Quelques filtres fréquents

- `cut` sélectionne des colonnes dans un texte
- `grep` cherche des mots ou des expressions dans un texte
- `head` affiche le début d'un fichier
- `tail` affiche la fin d'un fichier
- `nl` numérote les lignes d'un fichier
- `shuf` mélange aléatoirement les lignes d'un fichier
- `sort` trie les lignes d'un fichier
- `uniq` élimine les doublons consécutifs
- `wc` compte les caractères, mots ou lignes d'un fichier

cut

cut sélectionne des colonnes dans un texte.

Exemple

Les noms d'utilisateurs forment la première colonne du fichier /etc/passwd:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...
```

On les récupère avec:

```
$ cut -f1 -d: /etc/passwd
```

- -f: le(s) numéro(s) des champs à récupérer (**f**ields)
- -d: le **d**élimiteur de colonnes;

grep

grep cherche des mots ou des expressions dans un texte.

Exemple

Le fichier /etc/group contient tous les groupes du système et leurs membres:

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:logcheck
...
```

On peut trouver tous les groupes contenant anthony avec:

```
$ grep anthony /etc/group
cdrom:x:24:anthony
floppy:x:25:anthony
audio:x:29:pulse,anthony
...
```

head et tail

`head` affiche le début d'un fichier.
`tail` affiche la fin d'un fichier.

Exemple

```
$ head /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:logcheck
...
```

Exemple

```
$ tail /etc/group
...
gnome-initial-setup:x:997:
rdma:x:133:
vboxusers:x:134:
sddm:x:135:
i2c:x:136:
```

Par défaut, `head` et `tail` affichent 10 lignes, mais on peut changer le nombre avec l'option `-n`.

nl

`nl` numérote les lignes d'un fichier.

Exemple

```
$ nl /etc/group
    1  root:x:0:
    2  daemon:x:1:
    3  bin:x:2:
    4  sys:x:3:
    5  adm:x:4:logcheck
...
```

shuf

`shuf` mélange aléatoirement les lignes d'un fichier.
On peut aussi l'utiliser sur des arguments avec l'option `-e`.

Exemple

```
$ shuf -e 1 2 3 4 5
4
1
2
5
3
```

≠ uniq

`uniq` élimine les doublons consécutifs.

Exemple

```
$ cat texte.txt
Bonjour
Bonjour
Ça va?
Ça va?
Ça va?
Bonjour
$ uniq texte.txt
Bonjour
Ça va?
Bonjour
```

`wc`

`wc` compte les caractères, mots ou lignes d'un fichier.

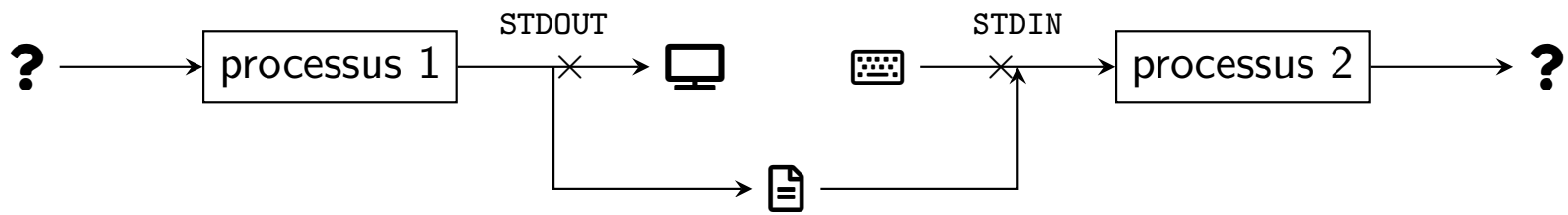
Exemple

```
$ cat texte.txt
Bonjour
Bonjour
Ça va?
Ça va?
Ça va?
Bonjour
$ wc texte.txt
  6  9 48 texte.txt
# |  |  |
# |  |  +- taille du fichier en octets
# |  +---- nombre de mots
# +------- nombre de lignes
```

Les *pipes*

Les *pipes*: motivations

- Les **pipes** sont un mécanisme permettant d'utiliser la sortie d'un processus comme entrée d'un autre processus;
- On peut y arriver avec des redirections:



- . . . mais cela nous oblige à créer un fichier intermédiaire, avec plusieurs inconvénients:
  - taille: le fichier peut être gros, et ne sert qu'une fois;
  - lenteur: on doit d'abord tout écrire dans le fichier, et ensuite le lire en entier;

Les *pipes*: utilisation

- Au lieu de cela, on peut utiliser un *pipe* pour que la sortie d'un processus devienne l'entrée du processus suivant:

- On obtient ce comportement avec la syntaxe `processus_1 | processus_2`

Exemple

Extrayons des champs de `/etc/passwd` et trions-les:

```
$ cut -f1 -d: /etc/passwd | sort    # usernames
$ cut -f3 -d: /etc/passwd | sort -h # UIDs
```

Contraintes des pipes

Exemple

Extrayons des champs de `/etc/passwd` et trions-les:

```
$ cut -f1 -d: /etc/passwd | sort    # usernames
$ cut -f3 -d: /etc/passwd | sort -h # UIDs
```

Remarquons que le paramètre de `sort` a disparu!

- dans son utilisation normale, on écrit `sort fichier`;
- avec les pipes, `fichier` est remplacé par la sortie du processus précédent;
- ici, l'entrée de `sort` est donc le résultat de `cut`;

Contraintes des pipes

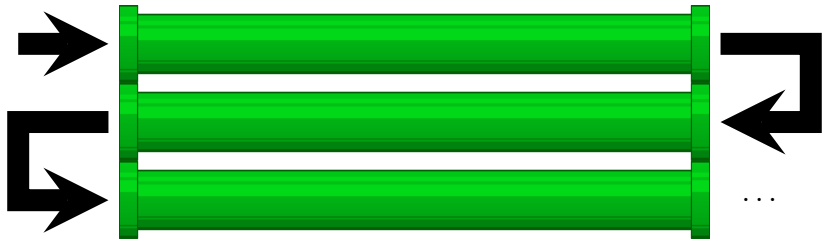
- La seule contrainte des *pipes* est le fonctionnement des programmes que l'on combine;
- En effet, quand on écrit `processus_1 | processus_2`:
  1. on redirige `STDOUT` vers l'entrée du *pipe* pour `processus_1`;
  2. on remplace `STDIN` par la sortie du *pipe* pour `processus_2`;
- Si `processus_1` n'écrit pas sur `STDOUT` (ou si `processus_2` ne lit pas sur `STDIN`), il y aura des redirections à faire;

Avantages des pipes

Si vous n'avez pas besoin des données intermédiaires, utilisez les *pipes*!

- ✓ pas de données grosses et inutiles à stocker;
- ✓ plus rapide: on peut traiter chaque ligne directement au lieu d'attendre le résultat complet;

Pipes multiples



On peut combiner autant de *pipes* qu'on veut! Dans ce cas, la sortie du processus avant le *pipe* `i` devient l'entrée du processus suivant ce même *pipe*;

Exemple

La commande suivante permet d'obtenir la liste triée des noms des processus qui travaillent sur un fichier ouvert:

```
$ lsof | cut -f1 -d' ' | sort | uniq
```

Celle-ci montre les applications utilisant le réseau:

```
$ lsof -P -i -n | cut -f 1 -d " "| uniq | tail -n +2
```

source: https://www.commandlinefu.com/commands/view/3543/show-apps-that-use-internet-connection-at-the-moment.