

IF 110 – Les Tubes



Joachim Bruneau-Queyreix

ENSEIRB-MATMECA

Bordeaux-INP

jbruneauqueyreix@enseirb-matmeca.fr

D'après le cours d'introduction aux systèmes d'exploitation de Télécom SudParis



Présentation du cours

- ❑ Contexte : comprendre comment les processus interagissent

- ❑ Objectifs :
 - Savoir utiliser les tubes

- ❑ Notions clés :
 - Les tubes (`mkfifo`, redirections avancées, `|`)

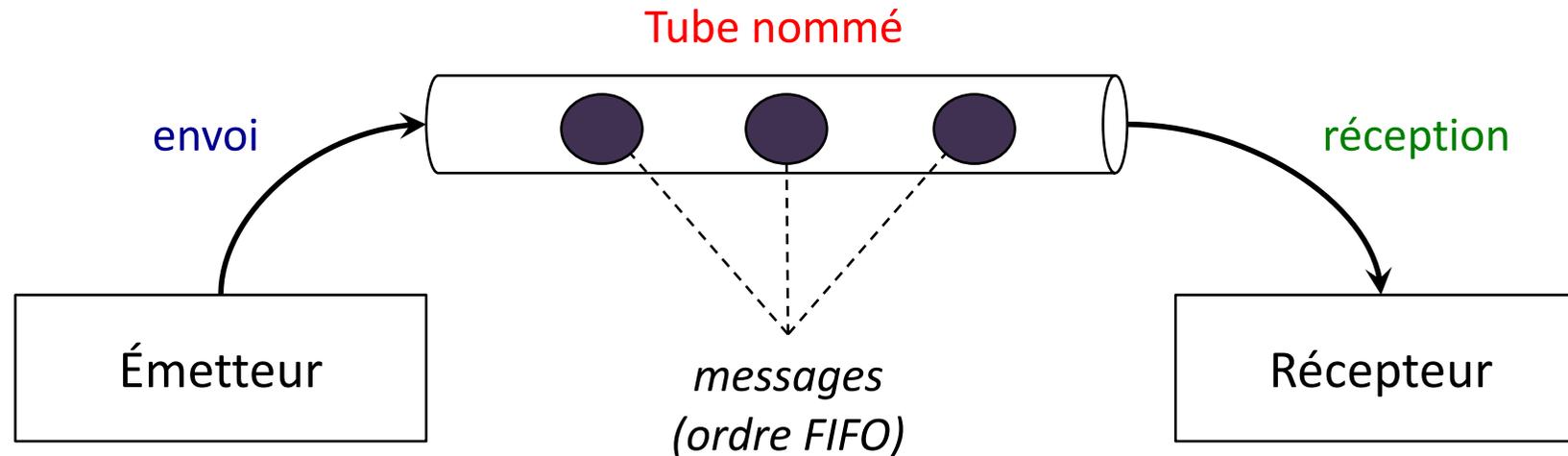


Les tubes

- ❑ Tube = mécanisme de communication par envoi de messages
 - Via un canal de communication
 - Message = données quelconques
 - Pas de perte de message,
 - Réception dans l'ordre d'émission (réception dite FIFO)

- ❑ Souvent utilisé pour rendre de nouveaux services
 - Une base de données + un serveur Web
 - Un processus d'affichage de notifications pour les autres processus
 - De façon générale, pour mettre en place une architecture de type client/serveur

Tubes nommés

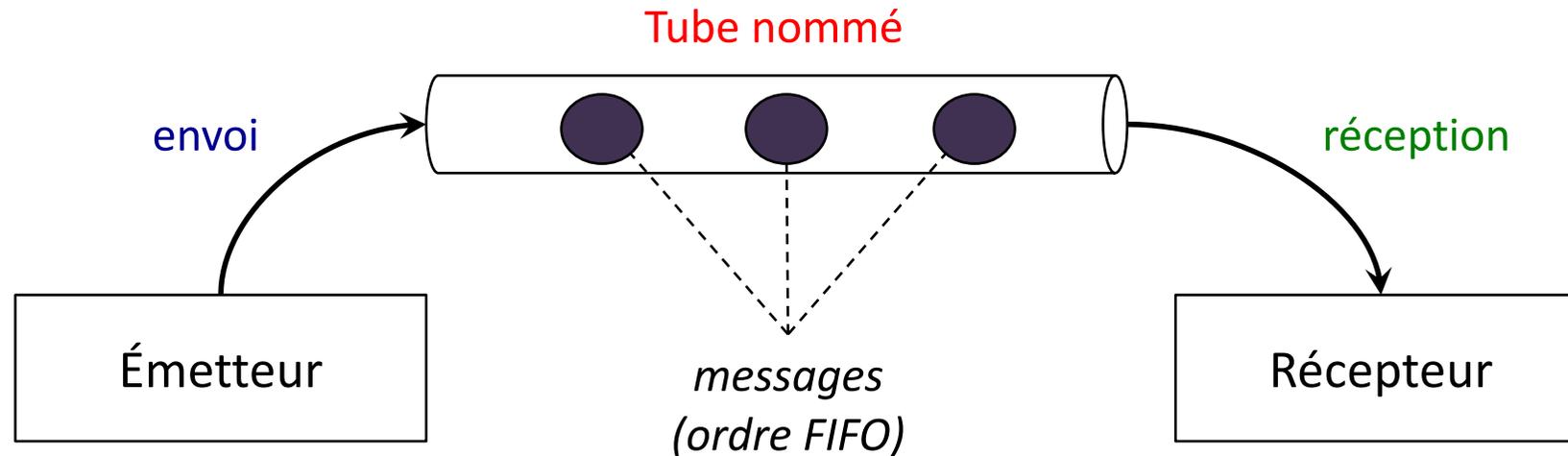


□ Principe :

- Un tube est un fichier spécial dans le système de fichiers
- L'émetteur écrit dans le tube
- Le récepteur lit dans le tube



Tubes nommés

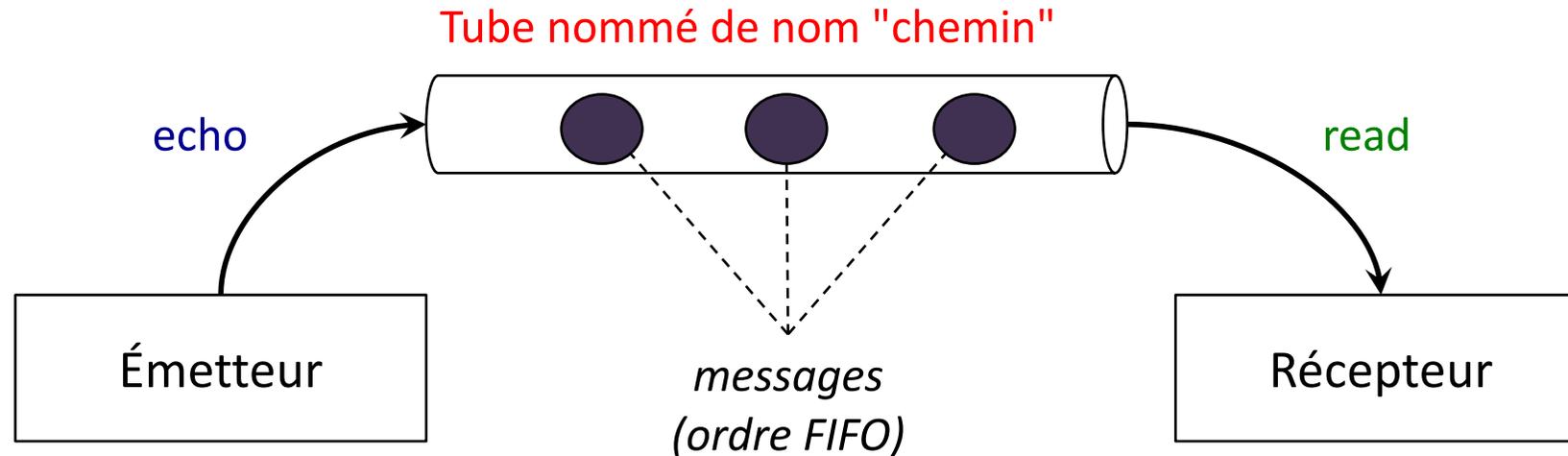


❑ Nota bene :

- L'émetteur bloque en attendant un message
- Ouverture bloque si interlocuteur n'existe pas
- Erreur si émission alors qu'il n'existe plus de récepteur



Principe de mise en œuvre



❑ Principe de mise en œuvre :

- `mkfifo chemin` : crée un tube nommé (visible avec `ls`)
- `echo message >chemin` : écrit dans le tube nommé
- `read message <chemin` : lit à partir du tube nommé
- `rm chemin` : détruit le tube nommé



Principe de mise en œuvre

```
#!/bin/bash
```

```
mkfifo /tmp/canal
```

```
echo "Bonjour" >/tmp/canal
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

Création du canal

bonjour.sh



/tmp/canal

```
#!/bin/bash
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

```
echo "Au revoir" >/tmp/canal
```

au-revoir.sh



Principe de mise en œuvre

```
#!/bin/bash
```

```
mkfifo /tmp/canal
```

```
echo "Bonjour" >/tmp/canal
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

"Bonjour"

bonjour.sh



/tmp/canal

```
#!/bin/bash
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

```
echo "Au revoir" >/tmp/canal
```

au-revoir.sh



Principe de mise en œuvre

```
#!/bin/bash
```

```
mkfifo /tmp/canal
```

```
echo "Bonjour" >/tmp/canal
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

bonjour.sh



/tmp/canal

```
#!/bin/bash
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

```
echo "Au revoir" >/tmp/canal
```

au-revoir.sh

"Bonjour"



Principe de mise en œuvre

```
#!/bin/bash
```

```
mkfifo /tmp/canal  
echo "Bonjour" >/tmp/canal  
read a </tmp/canal  
echo "Reçoit $a"
```

bonjour.sh

```
#!/bin/bash
```

```
read a </tmp/canal  
echo "Reçoit $a"  
echo "Au revoir" >/tmp/canal
```

au-revoir.sh



/tmp/canal

"Au revoir"



Principe de mise en œuvre

```
#!/bin/bash
```

```
mkfifo /tmp/canal
```

```
echo "Bonjour" >/tmp/canal
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

bonjour.sh

"Au revoir"



/tmp/canal

```
#!/bin/bash
```

```
read a </tmp/canal
```

```
echo "Reçoit $a"
```

```
echo "Au revoir" >/tmp/canal
```

au-revoir.sh



La vie est malheureusement complexe !

□ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube



La vie est malheureusement complexe !

□ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube

2. Ouvre tube



La vie est malheureusement complexe !

□ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube
3. Écrit "yes"

2. Ouvre tube



La vie est malheureusement complexe !

□ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube
3. Écrit "yes"
4. Ferme tube

2. Ouvre tube



La vie est malheureusement complexe !

□ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube
2. Ouvre tube
3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube (ok car \exists récepteur)



La vie est malheureusement complexe !

❑ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube
2. Ouvre tube
3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube (ok car \exists récepteur)
6. Lit le premier yes



La vie est malheureusement complexe !

□ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube
2. Ouvre tube
3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube (ok car \exists récepteur)
6. Lit le premier yes
7. Ferme tube



La vie est malheureusement complexe !

□ On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
while true; do
  echo "yes" >tube
done
```

emetteur.sh

```
#!/bin/bash
while true; do
  read line <tube
done
```

recepteur.sh

1. Ouvre tube
2. Ouvre tube
3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube (ok car \exists récepteur)
6. Lit le premier yes
7. Ferme tube
- 8. Second envoi plante car plus de récepteur**



Tube et redirection avancée

- ❑ Pour éviter les fermetures intempestives de tubes

Toujours ouvrir un tube avec une redirection avancée en lecture/écriture

- Si le récepteur ferme le tube, l'émetteur agissant comme récepteur, plus de plantage
- Effet connexe : ni l'émetteur ni le récepteur ne bloquent pendant l'ouverture si il n'existe pas encore d'interlocuteur



Tube et redirection avancée

- ❑ **Toujours ouvrir un tube avec une redirection avancée en lecture/écriture**

```
#!/bin/bash
```

```
exec 3<>tube
```

```
while true; do
```

```
    echo "yes" >&3
```

```
done
```

emetteur.sh

```
#!/bin/bash
```

```
exec 3<>tube
```

```
while true; do
```

```
    read line <&3
```

```
done
```

recepteur.sh

(redirection avancée pas nécessaire chez le récepteur dans ce cas, mais bonne habitude à prendre car souvent, un récepteur est aussi un émetteur)



Retour sur les tubes anonymes

❑ Tube anonyme : comme tube nommé, mais sans nom

❑ Le « | » entre deux processus shell crée un tube anonyme

```
cmd_gauche | cmd_droite
```

➤ Sortie standard de `cmd_gauche` connectée au tube

➤ Entrée standard de `cmd_droite` connectée au tube

❑ A haut niveau, équivalent à

```
mkfifo tube-avec-nom
```

```
exec 3<>tube-avec-nom
```

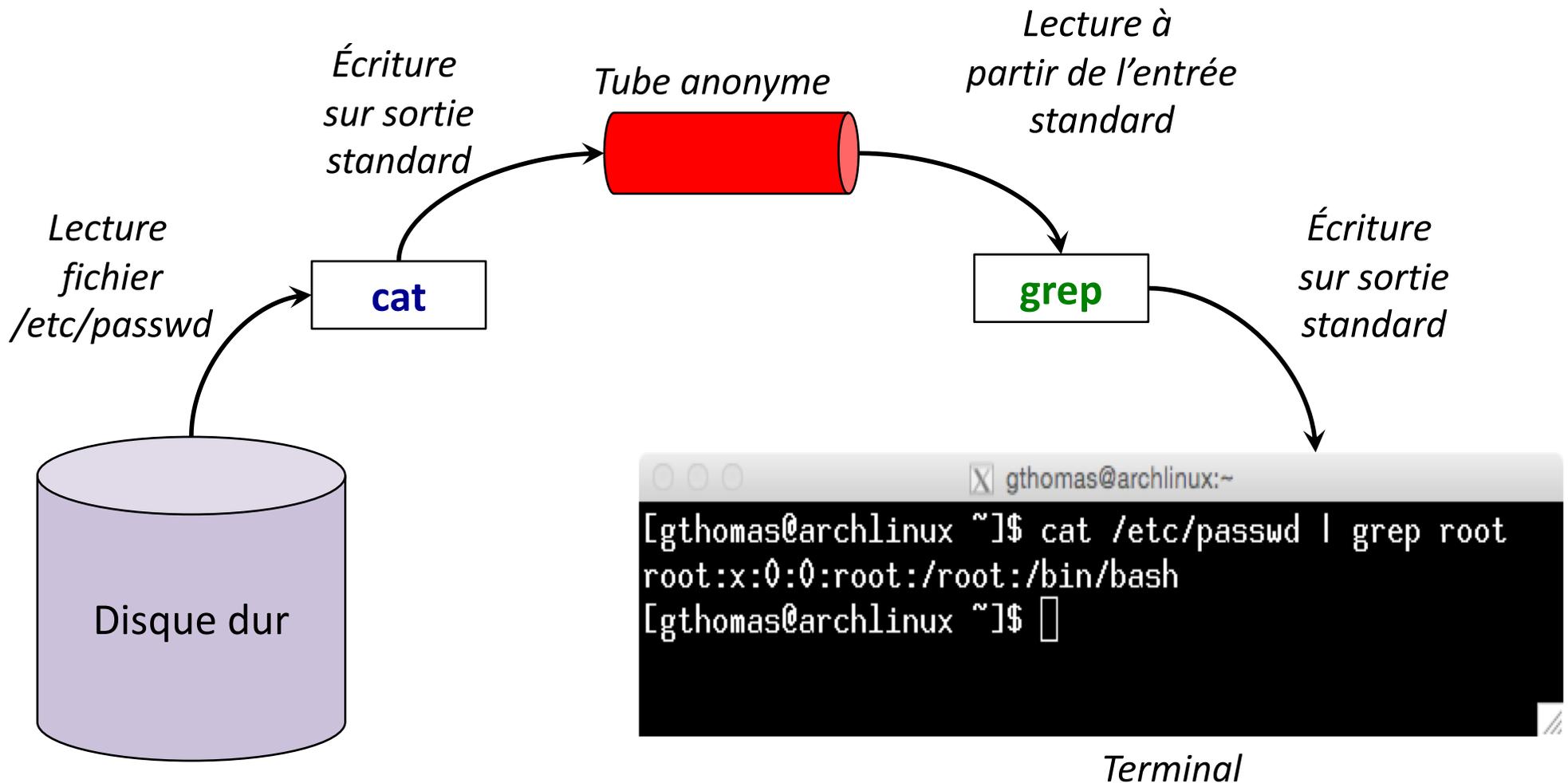
```
cmd_gauche >&3 &
```

```
cmd_droite <&3
```



Retour sur les tubes anonymes

```
cat /etc/passwd | grep root
```

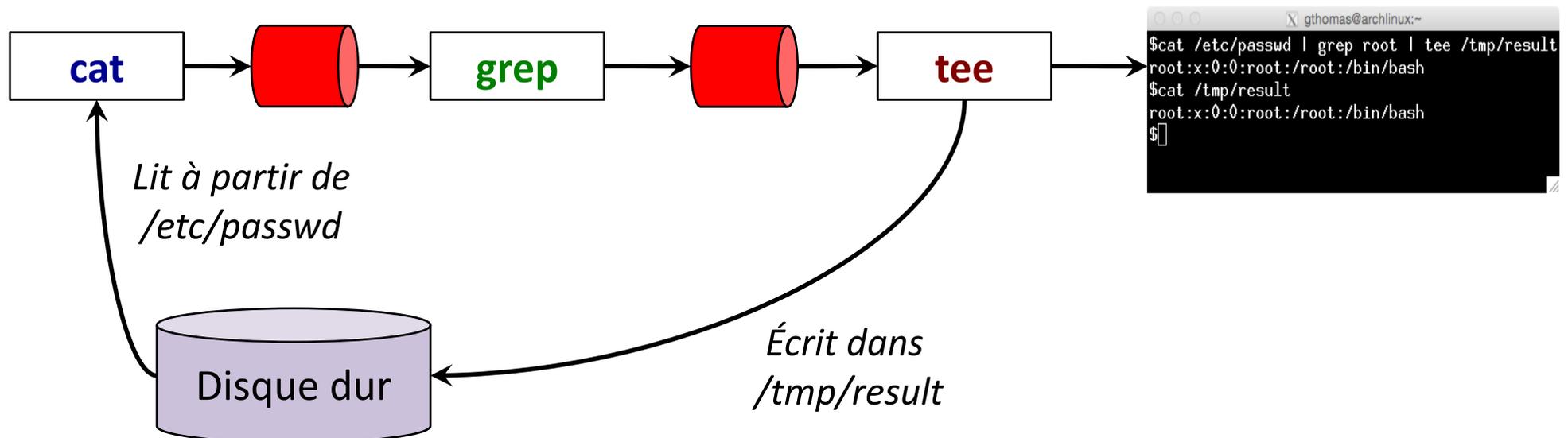




Commande utile : tee

- ❑ Écrit les données lues à partir de l'entrée standard
 - Sur la sortie standard
 - Et dans un fichier passé en argument

❑ Exemple : `cat /etc/passwd | grep root | tee /tmp/result`





Notions clés

❑ Tube nommé

- Fichier spécial dans le système de fichier
- Envoi de messages de taille quelconque, ordre de réception = ordre d'émission, pas de perte de message
- `mkfifo nom-tube` : crée un tube nommé mon-tube
- Toujours utiliser des redirections avancées

❑ Tubes anonyme (|)

- Comme un tube nommé, mais sans nom dans le système de fichier