Pratique des machines, installation, utilisation

Utiliser le Shell bash : Chemins et redirections

Les types de fichiers

- `-` Les fichiers
- `d` Les répertoires
- `l` Les liens symboliques (raccourcis)

- `s` Communiquer par le réseau
  exemple, socket acpid : `/var/run/acpid.socket`
- `b` "block device"
  périphérique capable de stocker de la donnée
  exemple, la mémoire RAM : `/dev/ram*`
- `c` "character device"
  Périphérique incapable de stocker de la donnée
  exemple, la souris : `/dev/psaux`

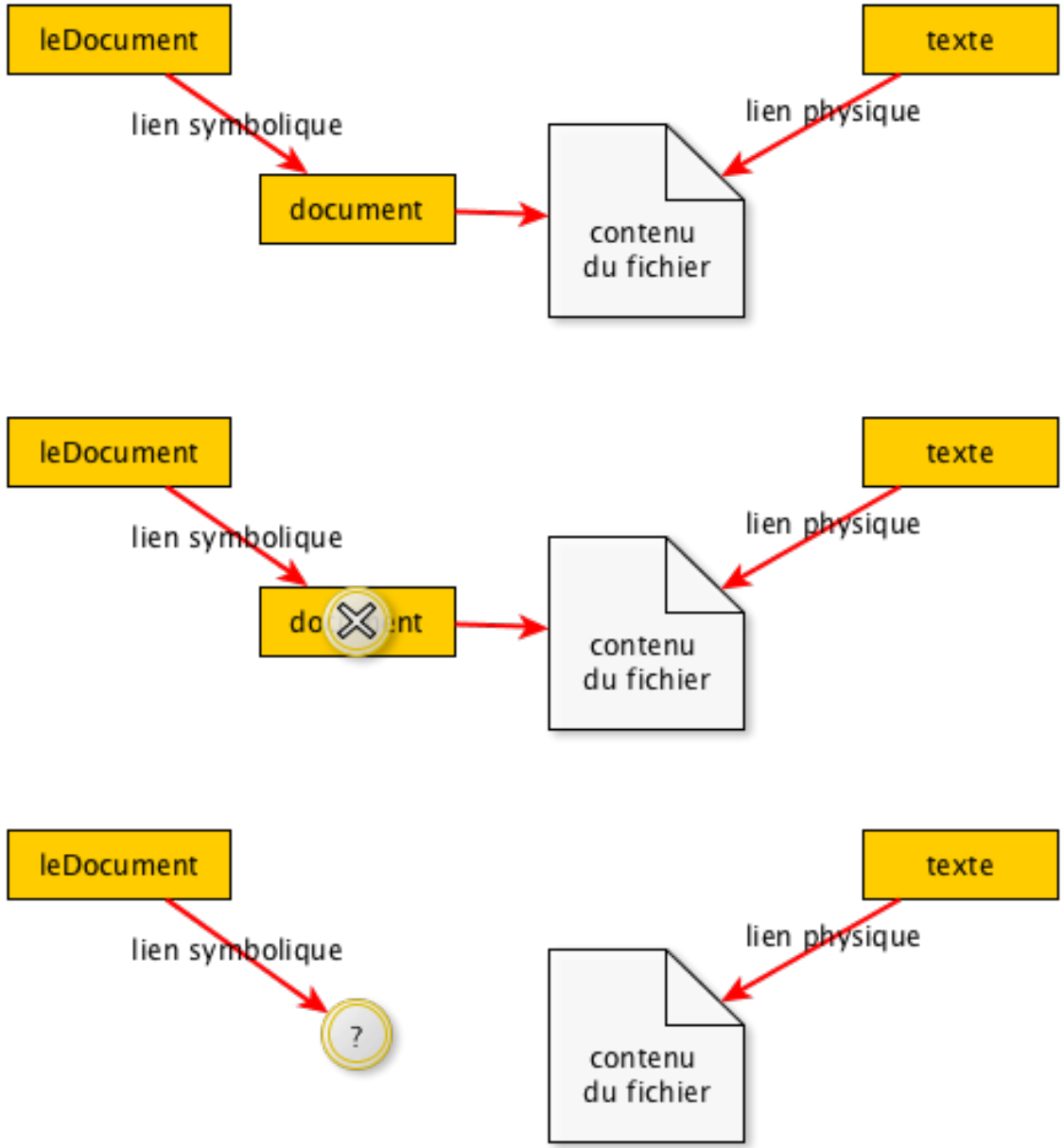
Les types de fichiers : exemples

```
$ ls -l
total 20
-rwxrwxr-- 1 moi moi  614 oct.  17 15:38 actualise.sh
lrwxrwxrwx 1 moi moi   74 oct.  17 16:35 l1_tp5 -> travail/cours/support/l1/tp5
-rwxrwxr-x 1 moi moi 1060 oct.  17 16:33 script.sh
drwxrwxr-x 4 moi moi 4096 sept.  4 18:55 travail

$ ls -l /var/run/acpid.socket
srw-rw-rw- 1 root root 0 oct.  13 12:40 /var/run/acpid.socket

$ ls -l /dev/ram*
brw-rw---- 1 root disk 1,  0 oct.  13 12:39 /dev/ram0
brw-rw---- 1 root disk 1,  1 oct.  13 12:39 /dev/ram1
brw-rw---- 1 root disk 1,  2 oct.  13 12:39 /dev/ram2
brw-rw---- 1 root disk 1,  3 oct.  13 12:39 /dev/ram3
(...)

$ ls -l /dev/psaux
crw------- 1 root root 10, 1 oct.  13 12:39 /dev/psaux
```

Liens symboliques, liens physiques

- Lien symbolique :
  - Un alias, un raccourcis vers le nom original du fichier.
  - Si le nom original est déplacé ou détruit, le lien ne pointe plus sur rien.
- Lien physique (vers un fichier, pas un répertoire) :
  - Un autre nom pour désigner la même chose.
  - Si le nom original est déplacé ou détruit, cet autre nom continue à designer l'objet.
  - On ne peut distinguer le nom original des autres liens physiques : ce sont tous des liens physiques vers un même fichier.
  - L'objet n'est plus accessible quand plus aucun nom ne le désigne.

Liens symboliques, liens physiques : exemple



Chemins relatifs et absolus

- **Un chemin absolu commence par /**
  Il désigne le chemin à suivre depuis la racine du disque jusqu'au fichier (ou répertoire)
- Un chemin relatif indique le chemin par rapport à la position courante

- chemin absolu à connaitre :
  `~` mon répertoire personnel (`/home/[pseudo]`)
- chemin relatif à connaitre :
  `.` répertoire courant
  `..` répertoire parent

Lancer une commande/un script perso

Créons un script perso, notre propre commande echo :

```
$ echo 'echo "$* une fois !"' > echo

$ pwd       # je demande où on est
/home/moi   # on est dans le répertoire /home/moi
$ ls        # je liste les fichiers
echo        # je vois ma commande perso
$ echo coucou
coucou    # commande echo du système, pas la mienne...
```

- Lancer 'echo' avec un chemin absolu :

Lancer une commande/un script perso

Créons un script perso, notre propre commande echo :

```
$ echo 'echo "$* une fois !"' > echo

$ pwd       # je demande où on est
/home/moi   # on est dans le répertoire /home/moi
$ ls        # je liste les fichiers
echo        # je vois ma commande perso
$ echo coucou
coucou    # commande echo du système, pas la mienne...
```

- Lancer 'echo' avec un chemin absolu :

```
$ /home/moi/echo coucou
```

Lancer une commande/un script perso

Créons un script perso, notre propre commande echo :

```
$ echo 'echo "$* une fois !"' > echo

$ pwd       # je demande où on est
/home/moi   # on est dans le répertoire /home/moi
$ ls        # je liste les fichiers
echo        # je vois ma commande perso
$ echo coucou
coucou   # commande echo du système, pas la mienne...
```

- Lancer 'echo' avec un chemin absolu :

```
$ /home/moi/echo coucou
```

- Lancer 'echo' avec un chemin relatif :

Lancer une commande/un script perso

Créons un script perso, notre propre commande echo :

```
$ echo 'echo "$* une fois !"' > echo

$ pwd       # je demande où on est
/home/moi   # on est dans le répertoire /home/moi
$ ls        # je liste les fichiers
echo        # je vois ma commande perso
$ echo coucou
coucou    # commande echo du système, pas la mienne...
```

- Lancer 'echo' avec un chemin absolu :

```
$ /home/moi/echo coucou
```

- Lancer 'echo' avec un chemin relatif :

```
$ ./echo coucou
```

Lancer une commande/un script : le $PATH

```
$ echo coucou  # la commande du système
coucou
$ ./echo coucou # ma commande perso
coucou une fois !
```

Lancer une commande/un script : le $PATH

```
$ echo coucou  # la commande du système
coucou
$ ./echo coucou # ma commande perso
coucou une fois !
```

- comment le Shell sait-il où est la commande du système ?

Lancer une commande/un script : le $PATH

```
$ echo coucou  # la commande du système
coucou
$ ./echo coucou # ma commande perso
coucou une fois !
```

- comment le Shell sait-il où est la commande du système ?
- Explication : Si je ne précise pas le chemin (relatif ou absolu) vers ma commande, le Shell utilise le `$PATH` pour trouver la commande

Lancer une commande/un script : le $PATH

```
$ echo coucou  # la commande du système
coucou
$ ./echo coucou # ma commande perso
coucou une fois !
```

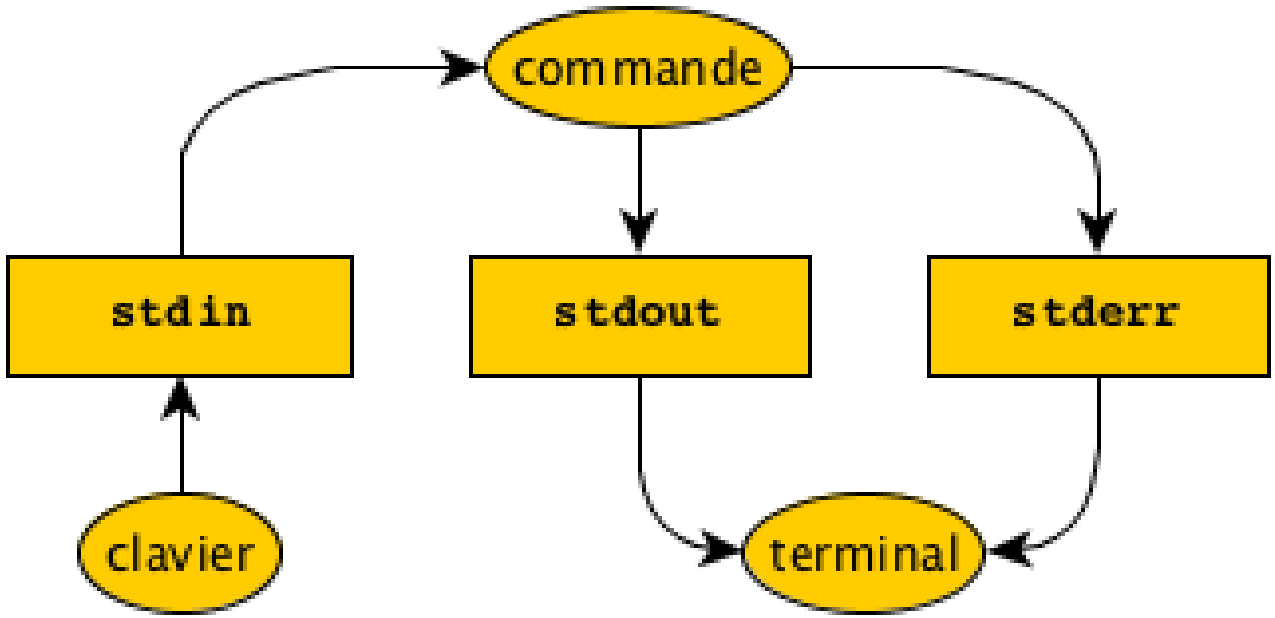
- comment le Shell sait-il où est la commande du système ?
- Explication : Si je ne précise pas le chemin (relatif ou absolu) vers ma commande, le Shell utilise le `$PATH` pour trouver la commande

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games:/usr/local/games
```

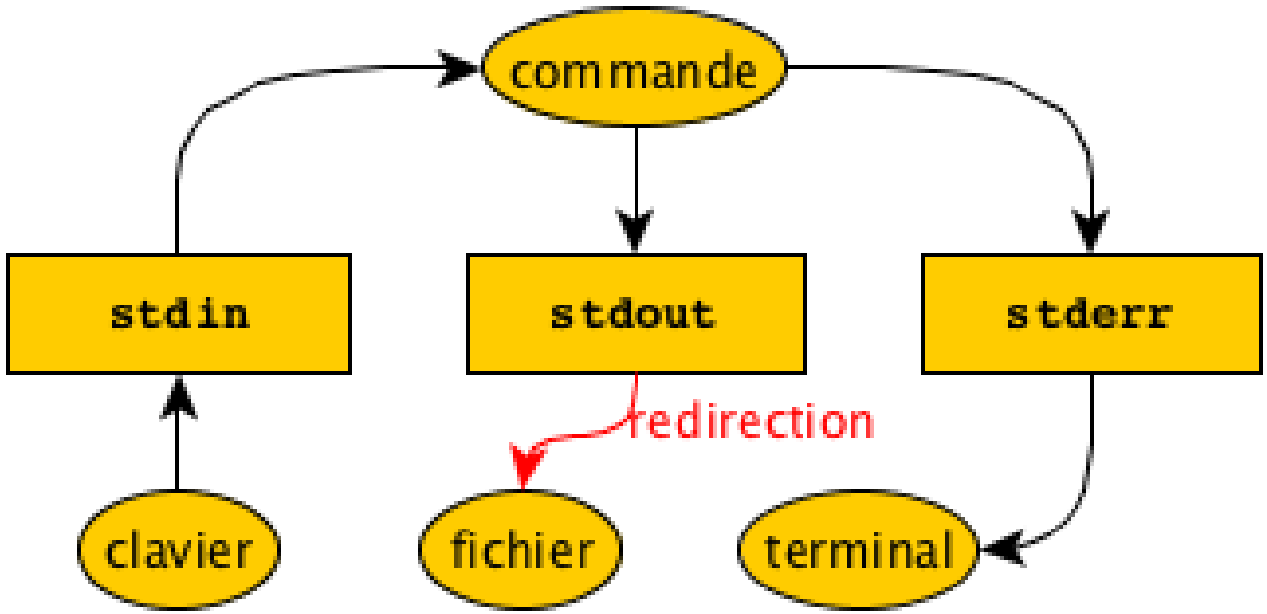
Les entrées / sorties

- **L'entrée standard** (`stdin`) : pour transmettre des données à un programme/commande/script
- **La sortie standard** (`stdout`) : le programme/script y place les informations qu'il nous donne en résultat.
- **La sortie d'erreur** (`stderr`) : le programme y place les messages d'erreur quand quelque chose se passe mal.

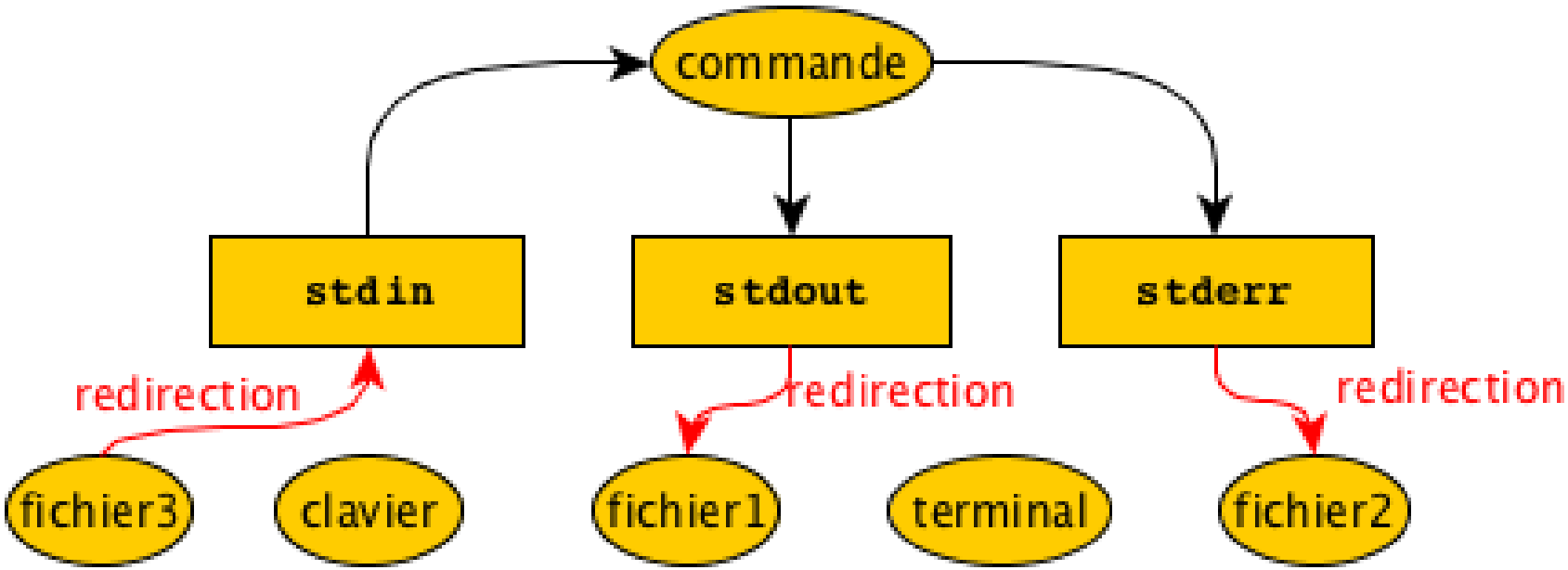
Les entrées / sorties : redirections



Les entrées / sorties : redirections



Les entrées / sorties : redirections



Les redirections 1/2

```
$ ls . toto > fichier
```

Rediriger la sortie dans "fichier", les erreurs sont affichées dans la console. Si fichier n'existe pas il est créé, si il existe son contenu est remplacé.

```
$ ls . toto 2> fichier2
```

Rediriger les erreurs dans "fichier2", la sortie est affichée dans la console. Si fichier n'existe pas il est créé, si il existe son contenu est remplacé.

```
$ ls . toto >> fichier2
```

Rediriger la sortie dans "fichier" sans écraser son contenu. Si fichier n'existe pas il est créé.

```
$ ls . toto 2>> fichier2
```

Rediriger les erreurs dans "fichier2" sans écraser son contenu. Si fichier n'existe pas il est créé.

Les redirections 2/2

```
$ ls . toto > fichier 2>&1
```

Rediriger la sortie dans "fichier", les erreurs sont envoyées au même endroit. Si fichier n'existe pas il est créé, si il existe son contenu est remplacé.

```
$ ls . toto >> fichier 2>&1
```

Idem, mais sans écraser le contenu de "fichier".

```
$ ls . toto 2> fichier 1>&2
```

Rediriger les erreurs dans "fichier", la sortie est envoyée au même endroit. Si fichier n'existe pas il est créé, si il existe son contenu est remplacé.

```
$ ls . toto 2>> fichier 1>&2
```

Idem, mais sans écraser le contenu de "fichier".

```
$ ls . toto &> fichier
```

Rediriger toute les sorties (standard, erreurs) dans "fichier".

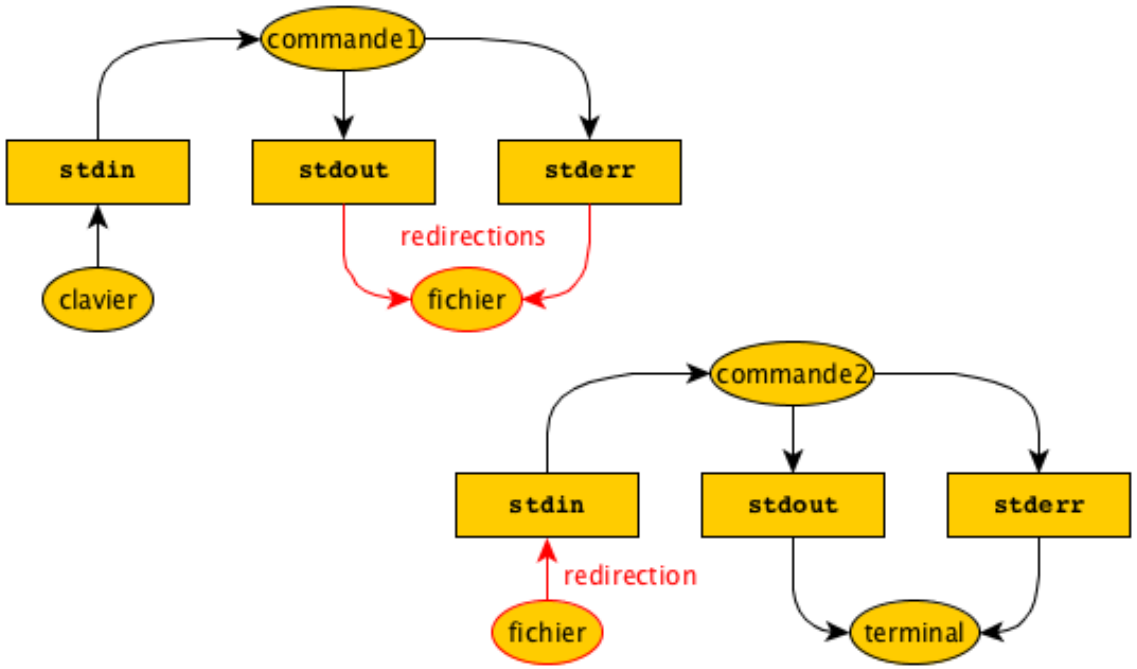
```
$ ls . toto &>> fichier
```

Idem, mais sans écraser le contenu de "fichier".

Les pipes 1/2

Utiliser un fichier pour conserver le résultat d'une commande et le transmettre à une autre commande : c'est pas pratique...

```
$ commande1 &> fichier
$ commande2 < fichier
```



Les pipes 2/2

Plus pratique, utiliser un pipe :

```
$ commande1 | commande2
```

Les sorties de "commande1" sont directement transmises à "commande2".

Exemple :
`wc` compte les lignes/mots/caractères (octets)

```
$ ls | wc
       17       38      448
```

Pratiquons !

- Copier l'archive contenant le TP dans votre répertoire personnel.

```
$ cp /opt/tp5.tar.gz ~/
```

- Décompresser le fichier tp5.tar.gz qui se trouve dans votre home.

```
$ tar xzvf tp5.tar.gz
```

- Rentrer dans le répertoire 'tp5' contenant le TP.

```
$ cd tp5
```

- Exécuter les commandes contenues dans 'start' dans l'environnement du shell (configuration pour le TP).

```
$ source start
```