Bases de l'Architecture et des Systèmes (BAS)
Partie Systèmes
Eléments de shell de Bourne

Équipe pédagogique Système, Univ. Toulouse

Références :

- Cours de Avi Silberschatz, Peter Baer Galvin et Greg Gagne, site `https://codex.cs.yale.edu/avi/os-book/OS10/slide-dir/index.html`
- Livre : « Operating System Concepts Tenth Edition » Avi Silberschatz, Peter Baer Galvin and Greg Gagne, John Wiley & Sons, Inc. ISBN 978-1-118-06333-0

Sommaire

Métacaractères du shell

- Caractères génériques (ou jokers)
- Permettant de
  - Construire des listes de noms de fichiers
  - Modifier l'interprétation des commandes
- Exemple de métacaractères
  - *
  - ?
  - \

Métacaractères du shell I

Construire des listes de noms de fichiers

- Exemples de problèmes
  - Dans un répertoire, on souhaite effacer les fichiers ayant une extension « jpg » mais pas les autres
  - Dans un répertoire, on souhaite effacer les fichiers image001.jpg . . .image999.jpg
- Solution
  - Construire la liste des fichiers d'extension « jpg »
  - Construire la liste des fichiers de nom imageXXX.jpg
- Rôle des métacaractères
  - Construire des listes de listes de noms de fichiers, séparés par des espaces, correspondant à un certain modèle

Métacaractères du shell II

Construire des listes de noms de fichiers

- Si aucun nom de fichier ne correspond au modèle, le modèle n'est pas remplacé
- `*` désigne une chaîne de caractères quelconque (éventuellement vide) ne contenant pas de caractère `/`, et ne commençant pas par un caractère `.` si `*` est placé en début de modèle
- `?` désigne un caractère quelconque, à l'exception d'un caractère `/`, ou d'un caractère `.` si `?` est situé en début de modèle.
- `[liste_caractères]` désigne un caractère dans la liste placée entre crochets
  - définie par énumération, par intervalle (avec le caractère `-`)
  - par négation (avec le caractère `!` placé juste après le crochet ouvrant)
  - `[Aa]` désigne le caractère `A` ou le caractère `a`
  - `[0-9a-zA-Z]` désigne un caractère alphanumérique quelconque
  - `[!0-9]` désigne n'importe quel caractère sauf un chiffre

Métacaractères du shell III

Construire des listes de noms de fichiers

- Le shell commence par interpréter ces métacaractères
  - il forme les listes des noms de fichiers qui correspondent au modèle
  - il transmet ces noms de fichiers aux commandes qui sont alors exécutées
- Les commandes reçoivent en paramètre le résultat de cette interprétation des métacaractères
- Exemple
  - `rm *.jpg`
  - `rm image???.jpg`

Métacaractères du shell I

Modifier l'interprétation des commandes

- ; sépare plusieurs commandes situées sur une même ligne
- ' délimite une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification)
- " délimite une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification, à l'exception des métacaractères `, $ et \)
- ` « capture » la sortie standard pour former un nouveau paramètre ou une nouvelle commande
- < permet de réaliser des redirections de l'entrée standard des commandes
- > permet de réaliser des redirections des sorties standard des commandes

Métacaractères du shell II

Modifier l'interprétation des commandes

- & suivi d'un chiffre permet de désigner les unités standard lors des redirections. & placé après une commande permet de lancer la commande en arrière plan
- | permet de réaliser un branchement de commandes
- $ retourne la valeur de la variable qui suit (il ne doit pas y avoir d'espace entre le métacaractère $ et le nom de la variable) et joue également un rôle très particulier vis-à-vis des « paramètres d'un shell »
- \ protège le caractère qui suit
  - que ce caractère soit normal ou un métacaractère du shell
  - Sauf à l'intérieur d'une chaîne délimitée par des '
- A l'intérieur d'une chaîne de caractères délimitée par des "
  - le caractère \ n'est interprété comme un métacaractère que s'il est suivi de l'un des quatre métacaractères ", `, $ ou \

Métacaractères du shell III

Modifier l'interprétation des commandes

- Dans tous les autres cas, il n'est pas interprété comme un métacaractère
- Exemple : la chaîne "*" est interprétée de la même façon que la chaîne '*'

Redirections

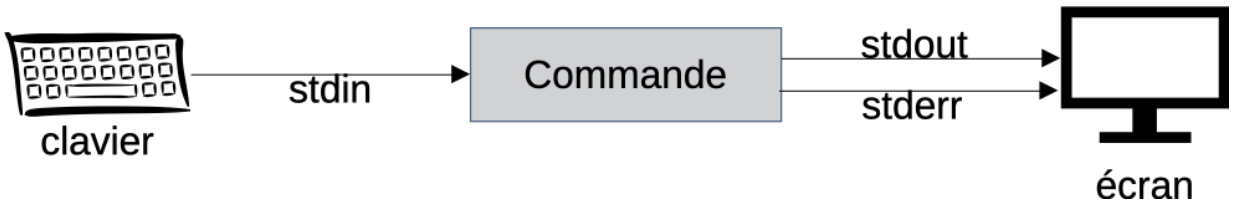
Unités standard

- Chaque processus a accès à trois unités logiques
  - l'entrée standard (`stdin`) : par défaut, le clavier
  - la sortie standard (`stdout`) : par défaut, l'écran
  - la sortie standard des erreurs (`stderr`) : par défaut, l'écran
- Les redirections permettent de changer les « connexions » par défaut entre les unités logiques et les unités physiques
  - `stdin` $\leftarrow$ clavier
  - `stdout` $\rightarrow$ écran
  - `stderr` $\rightarrow$ écran

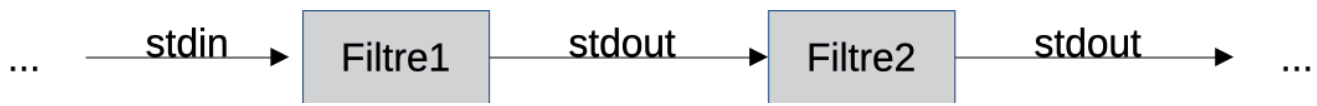
Redirections I

Filtres

- Certaines commandes peuvent
  - Recevoir des données par l'intermédiaire de l'entrée standard (`stdin`)
  - Fournir des informations
    - Sur la sortie standard (`stdout`) pour afficher leurs résultats
    - Sur la sortie standard des erreurs (`stderr`) pour afficher des messages d'erreur
- Ces commandes sont des « filtres »



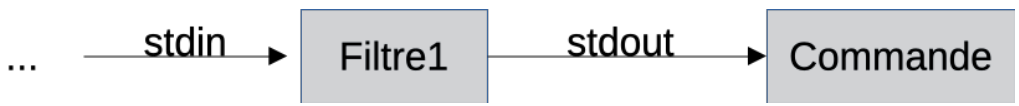
- Les filtres peuvent être « branchés » les uns à la suite des autres



Redirections II

Filtres

- Une commande ne fournissant pas de résultat sur `stdout` mais recevant des données sur `stdin` peut se trouver en fin de « branchement » (sortie)



- Une commande ne recevant pas de données de `stdin` mais fournissant un résultat sur `stdout` peut se trouver en début de « branchement » (entrée)

Redirections I

Quelques commandes

- `echo [chaîne...]` → affiche (sur `stdout`) les chaînes de caractères paramètres
- `cat [désignation_fichier...]` → affiche le contenu des fichiers dont les désignations sont en paramètres. Sans paramètre, cat affiche le contenu de `stdin`
- `grep chaîne [désignation_fichier...]` → affiche les lignes des fichiers dont les désignations sont en paramètres (ou de `stdin` si aucun paramètre) qui contiennent au moins une occurrence de chaîne
- `wc -l [désignation_fichier...]` → affiche le nombre de lignes suivi des désignations des fichiers passées en paramètre (ou le nombre de lignes de `stdin` si aucun paramètre)

Redirection de l'entrée standard d'une commande I

Entrée par défaut

- Quand une commande attend des lignes sur `stdin`, on peut les taper au clavier en utilisant l'une des deux syntaxes suivantes

```
Commande
Ligne1
Ligne2
. . .
<Ctrl>D
```

```
Commande << séparateur
Ligne1
Ligne2
. . .
séparateur
```

Redirection de l'entrée standard d'une commande II

Entrée par défaut

- Exemples
  - Ce qui est noté en bleu est tapé par l'utilisateur

```
$ cat
bonjour
bonjour
monsieur
monsieur
<Ctrl>D
$
```

```
$ cat << FIN
>bonjour
>monsieur
FIN
bonjour
monsieur
$
```

Redirection de l'entrée standard d'une commande III

Entrée par défaut

- Remarques sur la première syntaxe
  - <Ctrl>D (caractère non éditable de code ASCII égal à 4) simule la fin d'un fichier en mode interactif
  - Le texte tapé au clavier est envoyé à la commande ligne par ligne $\longrightarrow$ syntaxe inadaptée pour certaines commandes (par exemple, pour une commande dont le rôle serait de retourner les deux dernières lignes provenant de l'entrée standard)
- Remarques sur la seconde syntaxe
  - Les deux caractères << doivent être collés
  - Cette syntaxe peut être utilisée dans les scripts

Redirection de l'entrée standard d'une commande

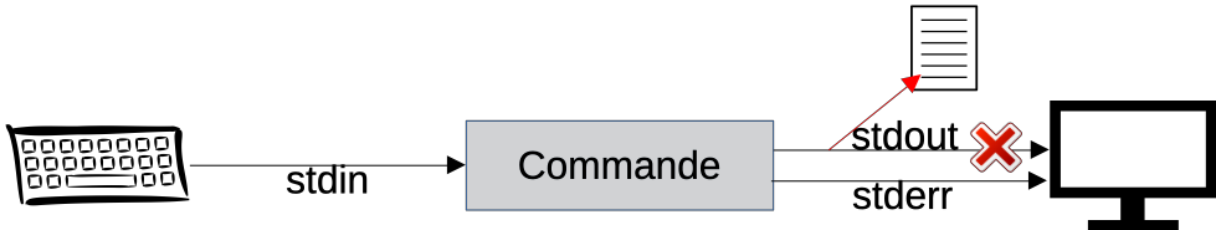
- `commande < désignation_fichier`
  - L'entrée standard de commande est connectée au fichier `désignation_fichier`
  - Ce fichier est ouvert en mode « lecture »

- Exemple : `grep 'e' < fich_entree.txt`
  - affiche les lignes du fichier `fich_entree.txt` qui contiennent au moins un `e`

Redirection de la sortie standard d'une commande

Redirection simple

- `commande > désignation_fichier`
  - La sortie standard de commande est connectée au fichier `désignation_fichier`
  - Si ce fichier n'existe pas, il est créé
  - Sinon, son contenu est « écrasé »
  - Ce fichier est ouvert en mode « écriture »



- Exemple : `grep 'e' > fich_sortie.txt`
  - écrit, dans le fichier `fich_sortie.txt`, les lignes tapées au clavier qui contiennent au moins un caractère `e`

Redirection de la sortie standard d'une commande

Redirection simple avec concaténation

- `commande >> désignation_fichier`
  - Si le fichier `désignation_fichier` n'existe pas, il est créé
  - Sinon, la sortie est ajoutée en fin du fichier `désignation_fichier`
  - Ce fichier est ouvert en mode « écriture en fin de fichier »
- Exemple : `grep 'e' >> fich_sortie.txt`
  - ajoute, à la fin du fichier `fich_sortie.txt`, les lignes tapées au clavier qui contiennent au moins un caractère `e`

Redirection de la sortie standard d'une commande

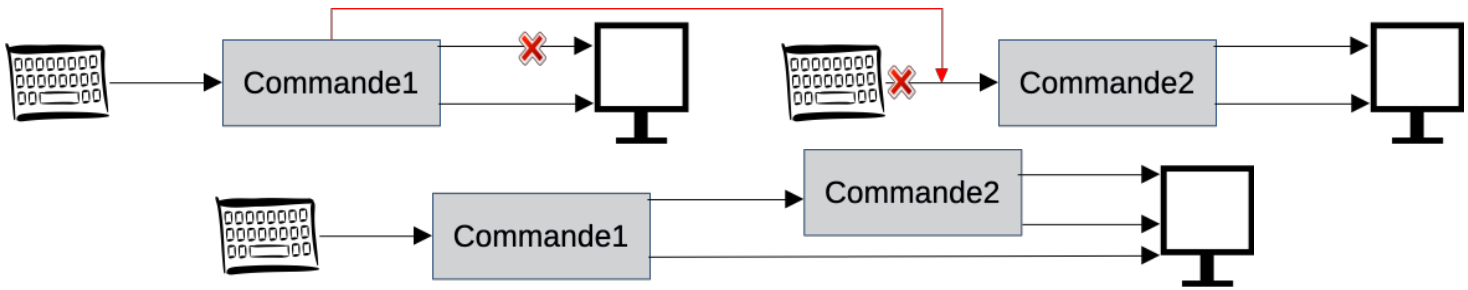
Redirection vers la sortie standard des erreurs

- `commande >&2`
  - La sortie standard de commande est connectée à sa sortie standard des erreurs
  - Ceci est très utile pour afficher des messages d'erreurs dans les scripts
  - Les caractères `>&2` doivent être « collés »

- Exemple : `echo "Fichier introuvable !" >&2`
  - affiche la chaîne de caractères `Fichier introuvable !` sur `stderr`

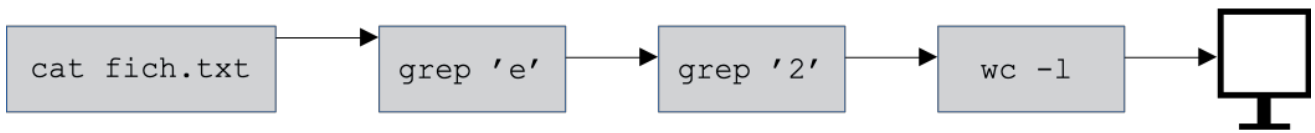
Branchement de commandes

- `commande1 | commande2`
  - Le métacaractère | est appelé « pipe » (ou « tube »)
  - La sortie standard de `commande1` est connectée à l'entrée standard de `commande2`
  - Ce branchement permet de réaliser des enchaînements de commandes



Branchement de commandes

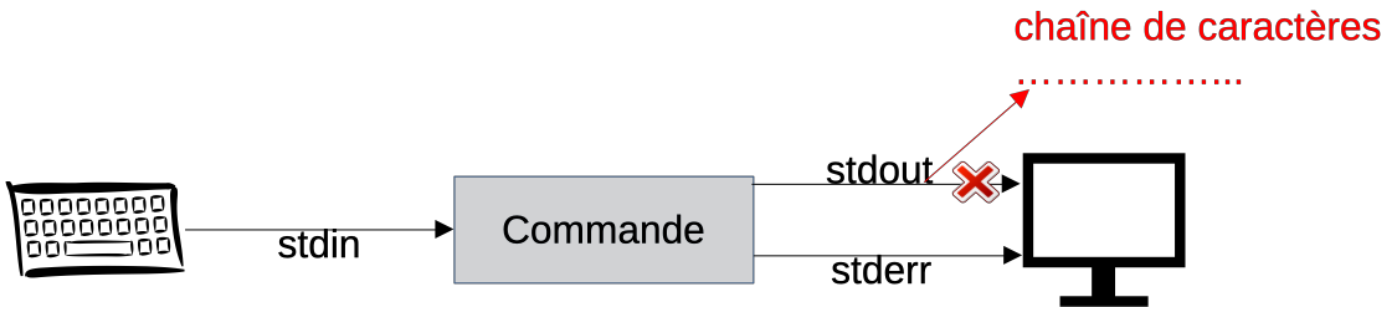
- `cat fich.txt | grep 'e' | grep '2' | wc -l`



- Affiche à l'écran le nombre de lignes du fichier `fich.txt` qui contiennent au moins un caractère `e` et au moins un caractère `2`

Capture de la sortie standard d'une commande

- `` `commande` `` (ou `$(commande)`)
  - La sortie standard de commande est transformée en une chaîne de caractères dans laquelle les éventuels sauts de lignes sont transformés en espaces
  - Cette chaîne de caractères peut constituer une autre commande
  - Qui peut être utilisée pour former les paramètres d'une autre commande



Capture de la sortie standard d'une commande

Exemple

- Si la commande `ls` envoie les deux lignes suivantes sur sa sortie standard :
  `fich1.txt`
  `fich2.txt`
  alors la commande `grep 'e' \`ls\``
- Équivaudra à `grep 'e' fich1.txt fich2.txt` et affichera les lignes du fichier `fich1.txt` contenant au moins un caractère `e`, suivies des lignes du fichier `fich2.txt` contenant au moins un caractère `e`

Capture de la sortie standard d'une commande

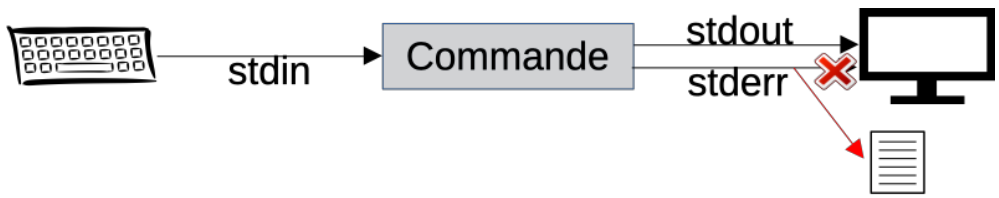
Remarques

- Différence entre `grep 'e' \`ls\`` et `ls | grep 'e'`
  - La 2^{e} affiche les **noms** de fichiers (parmi ceux retournés par `ls`) qui contiennent au moins un caractère `e`
- Les caractères `<`, `>` ou `>>` peuvent être suivis d'une capture (exemple : `grep 'e' < \`ls\``) mais de telles écritures seront incorrectes si la commande capturée (ici, `ls`) retourne autre chose qu'un nom unique de fichier. Dans la pratique, on évitera ce genre d'écriture
- Différence entre `ls > fich.txt` et `echo \`ls\` > fich.txt`
  - Dans les deux cas, le résultat de la commande `ls` est écrit dans le fichier `fich.txt`
  - 1re commande : un seul nom de fichier ou de sous-répertoire est écrit par ligne
  - 2^{e} commande : tous les noms de fichiers et de sous-répertoires sont écrits sur une même ligne, séparés par des espaces

Redirection de la sortie standard des erreurs d'une commande

Redirection de la sortie standard des erreurs d'une commande Redirection simple

- `commande` 2> `désignation_fichier`
  - La sortie standard des erreurs de `commande` est connectée au fichier `designation_fichier`



- Exemple : `ls fichier.txt 2> /dev/null`
  - écrit, dans le fichier « poubelle », les éventuels messages d'erreur de `ls`
  - Ce fichier poubelle est un trou noir : ce qui y est écrit est définitivement perdu
  - Utile pour éviter que certains messages d'erreurs ne s'affichent à l'écran

Redirection de la sortie standard des erreurs d'une commande

Redirection avec concaténation

- `commande 2>> désignation_fichier`
  - Si le fichier `désignation_fichier` n'existe pas, il est créé
  - Sinon, la sortie des erreurs de `commande` est ajoutée à la fin de `désignation_fichier`

Redirection de la sortie standard des erreurs d'une commande

Redirection vers sa sortie standard

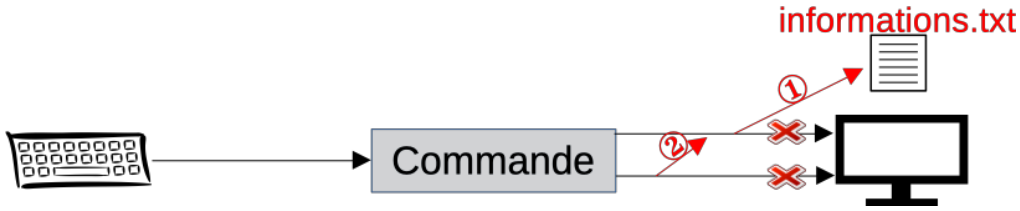
- `commande 2>&1`
  - La sortie standard des erreurs de `commande` est connectée à sa sortie standard

Redirection de la sortie standard des erreurs d'une commande

Redirection vers sa sortie standard

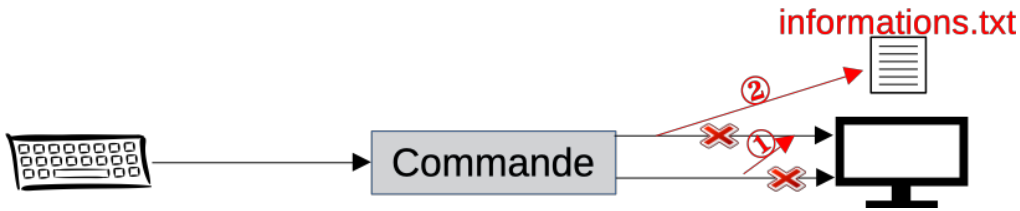
Exemples

- `ls -l fichier.txt > informations.txt 2>&1`



  - redirige à la fois la sortie standard et la sortie standard des erreurs vers le fichier `informations.txt`

- `ls -l fichier.txt 2>&1 > informations.txt`



  - ne redirige que la sortie standard vers le fichier `informations.txt`, car la sortie standard des erreurs a été connectée à la sortie standard avant que celle-ci ne soit redirigée vers `informations.txt`

Redirection de la sortie standard des erreurs d'une commande

Redirection vers sa sortie standard

Remarques

- Les caractères des séquences `2>`, `2>>` ou `2>&1` doivent être collés. Ces syntaxes varient d'un shell à l'autre
- `echo "Problème" 2> erreurs.txt >&2` équivaut à `echo "Problème" > erreurs.txt`
  - Elle provoquent l'écriture de la chaîne `Problème` dans le fichier `erreurs.txt`
- `echo "Problème" >&2 2> erreurs.txt` provoque l'affichage de la chaîne `Problème` à l'écran
  - Car la redirection de la sortie standard des erreurs vers le fichier `erreurs.txt` a été faite après la redirection de la sortie

Redirections : Attention !

- Il est déconseillé de rediriger l'entrée standard ou la sortie standard d'un branchement, pour ne pas écrire des lignes incompréhensibles et pas toujours interprétables par le shell
- Par exemple, au lieu de rediriger l'entrée standard, dans la ligne de commande suivante :
  `grep 'e' | grep '2' < fich_entree.txt`
  pour laquelle le shell indique qu'il y a une ambiguïté sur la redirection en entrée, on conseille d'utiliser la commande cat de la manière suivante :
  `cat fich_entree.txt | grep 'e' | grep '2'` ou
  `grep 'e' fich_entree.txt | grep '2'`
- Remarque : bien que syntaxiquement correcte, la séquence `...|cat|...` est toujours inutile