Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2019 – 2020
DEV1 – ENVL

DEV1 – ENVL – Laboratoire d'environnement système

TD 5 – GNU/Linux (partie III)

Table des matières

Dans votre répertoire `~/dev1`, créez un répertoire `td5`. Il contiendra tous les fichiers que vous allez créer aujourd'hui.

Version du document
21 septembre 2019

1 Jokers

Le dossier `/eCours/dev1/envl` contient plusieurs programmes Java. Imaginons qu'on veuille les copier tous chez nous. Comment faire ?

1. On peut entrer une commande `cp` par fichier à copier. C'est pénible.
2. La commande `cp` accepte plusieurs noms de fichiers à copier pour autant que toutes les copies se fassent au même endroit. Ainsi on peut écrire :
   `cp fichier1 fichier2 fichier3 dossierOùLesMettre`.
   C'est mieux mais encore pénible.
3. S'il y a une certaine logique dans les fichiers à copier, on peut utiliser les *jokers*.

Les jokers

Dans un nom de fichier/dossier

- ▷ « `?` » remplace *un et un seul* caractère quelconque.
- ▷ « `*` » remplace *0, 1 ou plusieurs* caractères quelconques.

Expérience 1 **Copie de plusieurs fichiers**

Utilisons les jokers pour facilement tout copier d'un coup.

- ✎ Si ce n'est pas encore fait, créez un dossier `td5` et placez-vous dedans.
- ✎ Entrez `cp /eCours/dev1/envl/*.java .`
  Que signifie le « `.` » (point) dans la commande ?
- ✎ Affichez le contenu de `td5`. Vous constatez que plusieurs fichiers y ont été copiés.

Concrètement, lorsqu'il voit un nom avec un joker, le shell le remplace par tous les fichiers/dossiers qui correspondent au motif. Ensuite il appelle la commande comme si tous les noms de fichiers/dossiers avaient été entrés explicitement.

Exercice 1 **Comprendre le `*`**

Quels sont les noms qui correspondent au motif `*1*.java` ?

- ☐ `Ex10.java`
- ☐ `Ex1.java`
- ☐ `Ex10.class`
- ☐ `1.java`

Exercice 2 **Comprendre le `?`**

Quels sont les noms qui correspondent au motif `Ex?.java` ?

- ☐ `Ex10.java`
- ☐ `Ex1.java`
- ☐ `10.java`
- ☐ `1.java`

Exercice 3 **Compiler en bloc**

Sachant que la commande `javac` accepte plusieurs noms de fichiers à compiler, comment compiler facilement tous les fichiers Java du dossier `td5`.

2 Recherche de fichiers

Vous commencez à avoir beaucoup de fichiers. Dans pareil cas, il arrive souvent qu'on ne sache plus exactement où se trouve un fichier. Est-ce que le système peut nous aider à le trouver ?

Les jokers peuvent parfois nous aider. Par exemple, si on sait que le fichier s'appelle `Hello.java` et qu'il se trouve directement dans un des dossiers `td`*i*, on peut le trouver avec la commande `ls ~/dev1/td*/Hello.java`.

Dans des cas plus complexes, la commande `find` vient à notre secours.

`find`

`find <oùChercher> <quoiChercher>`

Permet de chercher des fichiers sur le disque dur en fonction de critères : le nom, la date de création, le propriétaire, la taille, les permissions… mais pas le contenu ! Elle affiche la liste des fichiers trouvés.

- ▷ « oùChercher » : le dossier dans lequel chercher ;
- ▷ « quoiChercher » : le ou les critère(s) de recherche.

Exemple : `find ~ -name "*.java"`
permet de trouver tous les fichiers Java [1] *quelque part* dans son répertoire personnel.

On vous laisse consulter la page de manuel pour trouver comment exprimer les autres critères.

Exercice 4 Recherche 1

Cherchez dans votre répertoire personnel, tous les fichiers ou dossiers qui sont vides (il n'y en a peut-être pas !).

Exercice 5 Recherche 2

Cherchez dans votre répertoire personnel, tous les fichiers qui ont une taille de plus de 512 octets.

Exercice 6 Recherche 3

Listez tous les répertoires personnels qui sont composés d'exactement 3 caractères.

Exercice 7 Recherche 4

Listez tous vos fichiers qui ont été modifiés depuis moins de 24h.

1. Les guillemets demande au shell de ne *pas* traiter le joker. Le motif est passé tel quel à la commande et c'est le `find` qui va en tenir compte.

3 Redirection

Imaginons qu'un programme Java produise un gros output. Est-il possible de le sauver dans un fichier pour le consulter tranquillement ? Oui ! Ceci, et bien d'autres choses encore, est possible grâce aux redirections.

3.1 Redirection de la sortie

Expérience 2

Redirection de la sortie d'un programme Java

Si ce n'est pas encore fait, créez un dossier `td5` et placez-vous dedans.

- `cp /eCours/dev1/envl/PasCroissant.java .` pour copier un programme Java qui produit un output très long.
- Examinez le code et essayez de comprendre ce qu'il fait.
- `javac PasCroissant.java` pour le compiler.
- `java PasCroissant` pour l'exécuter. Vous constatez que le résultat est très long.
- `java PasCroissant > res` Plus rien n'est affiché !
- `ls` pour constater qu'un fichier `res` est apparu.
- `cat res` pour constater qu'il contient bien tout l'output du programme.

Ce qu'on vient de voir avec un programme Java est en fait vrai pour n'importe quel exécutable (les commandes aussi donc).

Redirection de la sortie

La *sortie standard* d'un exécutable est par défaut l'écran. Ça peut être changé.

- ▷ `> nomFichier` : à la suite d'une commande indique que les affichages se feront *dans le fichier* indiqué (qui est créé ou vidé avant)
- ▷ `>> nomFichier` : pour *ajouter* les affichages à la *fin* du fichier.

Transparent pour les exécutables

Comprenez-bien que ceci est *transparent* pour les commandes. Le programme Java n'a pas été modifié ! Pour lui, `System.out` représente la *sortie standard*. Par défaut c'est l'écran mais la redirection permet de rediriger vers un fichier.

Exercice 8

Rediriger la sortie d'une commande

Sauvez dans un fichier toute l'arborescence (commande `tree`) de votre répertoire personnel.

3.2 Redirection de l'erreur

En Java, le code `System.out.println()` est utilisé pour afficher quelque chose à l'écran. Il existe également le code `System.err.println()` pour afficher un message d'erreur. Pourquoi fournir 2 techniques différentes si, au final, le résultat apparait au même endroit ?

Redirection de la sortie d'erreur

L'*erreur standard* d'un exécutable est par défaut l'écran. Ça peut être changé.

▷ `2> nomFichier` : pour envoyer les *messages d'erreur* dans le fichier.

Lorsqu'un programme affiche des messages d'erreurs, ils sont mélangés aux affichages normaux. Une redirection permet de les séparer pour mieux les voir.

Ceci explique pourquoi c'est une bonne pratique en JAVA d'utiliser `System.err` lorsque vous affichez un message d'erreur.

Exercice 9

Recherche

Si on utilise la commande `find` pour chercher un fichier, il est possible que la commande tente d'accéder à des sous-dossiers pour lesquels vous n'avez pas les droits [2]. Dans ce cas, elle affiche un message d'erreur.

Affichez à l'écran la liste des fichiers `Hello.java` qui existent quelque part dans le système de fichier en enlevant les messages d'erreurs.

3.3 Redirection de l'entrée

Tout comme on peut rediriger la sortie, on peut rediriger l'entrée.

Redirection de l'entrée

L'*entrée standard* d'un exécutable est par défaut le clavier. Ça peut être changé.

▷ `< nomFichier` : pour indiquer que les lectures ne se font pas au clavier mais *depuis le fichier* (qui doit exister).

Expérience 3

Redirection de l'entrée d'un programme Java

- `cp /eCours/dev1/envl/Multiple4.java .` pour copier un programme JAVA qui lit des nombres au clavier (plus précisément, sur l'entrée standard).
- Examiner le code et essayez de comprendre ce qu'il fait.
- `javac Multiple4.java` pour le compiler.
- `java Multiple4` pour l'exécuter. Il vous demande d'entrer des entiers et affiche uniquement ceux qui sont multiples de 4.
- Pour indiquer qu'il n'y a plus de données, il faut appuyez sur `CTRL-D` (et pas `CTRL-C` qui tue brutalement le processus).

Si on veut exécuter à nouveau le programme pour les mêmes valeurs, il faudra retaper les nombres ? Non !

- Créez un fichier `data` avec quelques nombres.
- `java Multiple4 <data` pour exécuter le programme en demandant que les lectures se fassent dans le fichier indiqué et pas au clavier.

2. Un prochain TD sera consacré aux permissions sous GNU/LINUX.

Transparent pour les exécutables
À nouveau, ceci est *transparent* pour les commandes. C'est toujours `System.in` (l'entrée standard) qui est donné à `Scanner`.

Exercice 10

Pas croissant multiple de 4

Produisez un fichier `croissant4` qui contient parmi les 1000 premiers nombres de la suite des pas croissants (produite par le programme `PasCroissant`) uniquement ceux qui sont des multiples de 4.
Aide : la sortie de l'un devient l'entrée de l'autre.

Exercice 11

cat et ses variantes

La commande `cat` est plus riche que ce que vous avez vu.

1. Faites l'expérience de taper juste `cat`. Entrez quelques phrases puis appuyez sur `CTRL-D`. En vous aidant du manuel, comprenez ce qui se passe.
2. Entrez maintenant `cat <fichier` où `fichier` est le nom d'un fichier qui existe. Que se passe-t'il et pourquoi ?
3. Comment pouvez-vous utiliser la commande `cat` pour créer un fichier contentant juste le mot `Bonjour` (sans utiliser un éditeur donc) ?
4. Comment pouvez-vous utiliser la commande `cat` pour copier un fichier ?
5. Comment pouvez-vous utiliser la commande `cat` pour fusionner le contenu de deux fichiers ?

3.4 Tubes (pipes en anglais)

Pour résoudre l'exercice qui produit les pas croissants multiples de 4 de la section précédente, vous avez dû créer un fichier temporaire qui n'a servi qu'à ça. Vous avez redirigé la sortie de la première commande dans un fichier et redirigé ce fichier comme entrée de la seconde commande. Les tubes permettent de simplifier le processus.

Tubes

```
commande1 | commande2
```

Un *tube* (ou *pipe*) lie 2 commandes : la sortie de la première devient l'entrée de la seconde.

Exemple 1

Enchaîner les commandes

La commande `less` permet d'afficher un message page par page. Si on ne lui donne pas de nom de fichier, il pagine les données reçues sur l'entrée standard. On peut donc remplacer

```
ls /home >temp
less temp
```

par

```
ls /home | less
```

Exercice 12

Utilisation des tubes

Utilisez un tube pour refaire l'exercice précédent (afficher parmi les 1000 premiers nombres de la suite des pas croissants, tous ceux qui sont des multiples de 4).

4 Conclusion

Notions importantes de ce TD

Voici les notions importantes que vous devez avoir assimilées à la fin de ce TD.

- ▷ Comprendre et savoir utiliser les jokers (`*` et `?`).
- ▷ Savoir utiliser la commande `find` pour rechercher des fichiers. Pouvoir trouver dans la page de manuel la syntaxe pour un critère de recherche qui n'a pas été testé au laboratoire.
- ▷ Comprendre et savoir utiliser à bon escient les redirections et les tubes.

Pour aller plus loin...

Ceux qui ont du temps peuvent aborder les exercices suivants.

Exercice 13 **Traiter les fichiers trouvés**

Comment afficher le *contenu* de tous vos fichiers Java ? Aide : consultez la section `ACTIONS` de la page de manuel de `find`.