Université de Genève

-

Sciences Informatiques

Système d'Exploitation - TP 01

Noah Munz (19-815-489)

Septembre 2022

Contents

TP 01

2 Exercices

2.2 Commande echo

Echo affiche une ligne de texte sur la sortie standard du shell.

2.3 Create a file

Avec la fonction `touch`, e.g. `touch <filename>` pour créer le fichier `<filename>` (vide) dans le current directory.

2.4 Lancement de Programme

On voit que nautilus est un explorateur de fichier et que l'argument qu'on lui donne sera juste le starting directory de l'explorateur.

2.5 Fichier

(a) D'après le manuel, la commande `wc` réalise la chose suivante: "wc - print newline, word, and byte counts for each file".
I.e. `wc` donne des informations sur certaines caractéristiques d'un ou plusieurs fichiers. En rajoutant `-w` la commande retourne le nombre de mots du fichier en argument.

- `wc -w < foo.txt` retourne le nombre de mots du fichier foo.txt qu'on a redirigé sur l'entrée standard. En appelant wc sans arguments (directs), la commande va attendre/demander qu'on lui donne des mots en entrée et en utilisant `< foo.txt` on lui a justement redirigé le contenu du fichier foo.txt sur cette entrée.

- `cat foo.txt | wc w`. Avec `cat foo.txt` on obtient le contenu de foo.txt qu'on a "pipe" vers `wc -w`. On a redirigé l'output de `cat foo.txt` vers l'input de `wc`.

- `wc -w foo.txt` retourne le nombre de mots dans foo.txt suivi du nom du fichier qu'on lui a passé (e.g. `13 foo.txt`). Ici on lui a passé le fichier au lieu de directement son contenu comme au point (a) et (b).

(b) `head` retourne les n premières lignes d'un fichier (par défaut $n = 5$). `tail` fait la même chose mais commence par la fin, i.e. n dernières lignes

- `head foo.txt -n 6` Retourne les 6 premières lignes de foo.txt
- `tail foo.txt -n 6` Retourne les 6 dernières lignes de foo.txt

(c) `sort` trie les lignes d'un fichiers (en les concaténant s'il y en a plusieurs) puis écrit le tout sur la sortie standard. Quant à lui, `cmd args 2> file` redirige le flux d'erreur d'erreur de l'appel à la commande `cmd` (avec les arguments `args`) dans le fichier `file`.
Donc `sort foo.txt >out1.txt 2>out2.txt` va trier le contenu de foo.txt puis écrire le tout au début de out1.txt (en écrasant tout ce qui s'y trouvait potentiellement avant), puis redirige le flux d'erreur vers out2.txt.

Comme dit avant, out1 va contenir le résultat de sort et out2, les erreurs potentielles qui se sont produite.
Si foo.txt n'existe pas le contenu de `out2.txt` sera l'erreur indiquant que le fichier n'existe pas i.e. `sort: cannot read: foo.txt: No such file or directory` et `out1.txt` sera vide.
En revanche si foo.txt existe, `out1.txt` sera le résultat de `sort` et `out2.txt` sera vide.

2.6 Écrivez un script

Dans cet exercice, un script qui convertit en PNG toutes les images d'un dossier donnée vers un autre dossier, tout en prenant garde à "sanitize" les noms des copies converties dans le nouveau dossier. (i.e. enlever espaces...) a été réalisé. Il prend 3 arguments, le dossier de départ, le dossier d'arrivé et la résolution de l'image. (le dernier est optionnel)

Le script peut être trouvé ci-dessous (p. 3) et dans le fichier `script-Exo2.6` sur GitHub à l'adresse: 12X009-OS-TPs/TP01/script-Exo2.6 (où vous aurez un bien meilleur rendu.)

Que vous pouvez télécharger directement à partir de:
https://raw.githubusercontent.com/David-Kyrat/12X009-OS-TPs/master/TP01/script-Exo-2.6

```
#!/bin/bash
oldPath=$(pwd)
# Parse args
if [ "$#" -lt 2 ]; then
    >&2 echo "Not enough arguments, expecting at least 2"
    #exit 2
fi
res=""
if [ "$#" -ge 3 ]; then
    if [ "$#" != 3 ]; then echo "Too many arguments, only taking the first 3 in consideration";
        fi
    res="$3";
fi
beg="$1"; dest="$2"

echo $'\n Starting (png) conversion of images from'" \"$1\" to \"$2\" ..."
if [ "$3" ]; then echo $'\t (specified res: '"\"$3\")"; fi
echo ""


# Create output dir if does not exists
mkdir -p -v "$dest"

function is { [[ $1 ]] && echo "true" || echo "false" }

function isImg {
    crtFile="$1"
    is '$(file "$crtFile" -i) =~ image'
}

function renameBeforeExecution {
    mkdir tmp; cd tmp
    cp ../* . -f 2> err
    for f in * ; do mv "$f" $(tr -d "[' ''\"\"]" <<< "$f") -f 2> err;
    done
}


function copyAndConvertAll {
    renameBeforeExecution
    for f in * ; do
        if [[ $(isImg "$f") ]]; then
            mv "$f" "$oldPath/$dest"
        fi
    done
    cd ..
    rm tmp -r
    cd "$oldPath/$dest";

    if [ "$res" ]; then mogrify -format png -resize "$res" * 2> err
    else mogrify -format png * 2> err; fi

    echo $'\nConverted Files:\n\t'; ls *.png -l ;
    echo $'_____________________________________________\n'
}

cd "$beg"

copyAndConvertAll

cd $oldPath #go back to path where script was called
```