DV1457 - DV1578 Programming in UNIX Environments

Shell Programming Lab

Code Reference

September 9th, 2024.

Basic navigation via terminal.

```
ls [-l | -a] #list contents of the current directory.
cd <directory name> #change directories.
cat <file name> #prints the content of a file in the terminal.
mkdir <directory name> #creates a new directory.
touch <file name> #creates a new file.
cp <source file name> <copied file name> #creates a copy of a file.
rm <file name> #deletes a file.
```

Bash programming tools example.

```
#!/bin/bash

echo "Hello world from Kate and KDE"

read -p "Enter the value for the first variable: " VAR1
read -p "Enter the value for the second variable: " VAR2

if [ $VAR1 -gt $VAR2 ]; then
  echo "$VAR1 is greater than $VAR2"
else
  echo "$VAR2 is greater than $VAR1"
fi
```

Reinforcement exercises

Example 1: Line Generator.

Create a bash script that writes, in a separate file, 50 lines. Each line is composed by 50 randomly selected uppercase letters from a given string (i.e. vowels and consonants from the English alphabet). In between each letter, there should also be a withe space.

```
#!\bin\bash

random_letter() {
  chars="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  new_string=""
  separator=" "
  for i in {1..50}; do
    for j in $(seq 1 50); do
      if [ $j -eq 50 ]; then
        new_string=${new_string}${chars:$(( RANDOM % ${#chars} ))
    :1}
      else
        new_string=${new_string}${chars:$(( RANDOM % ${#chars} ))
    :1}
        new_string=${new_string}${separator}
      fi
    done
    if [ -e letters.txt ]; then
      echo -e "$new_string" >> letters.txt
    else
      echo -e "$new_string" > letters.txt
    fi
    new_string=""
  done
}

random_letter
```

Example 2: Export System Information.

Create a bash script that retrieves specific system information, according to the following arguments:

- -o: Operative system type, distribution, and architecture.
- -k: Kernel name and release version.
- -c: Complete CPU Information (per core).

The information retrieved by every argument must be exported into a separate external file. Also, the script can receive any number of these arguments (i.e. from 1 to 3).

```
#!\bin\bash

while getopts "okc" option; do
  case $option in
    o) echo "Exporting OS information.";
        echo "Operative system: $(uname -o)" > OS_info.txt;
        echo "Distribution: $(uname -n)" >> OS_info.txt;
        echo "Architecture: $(uname -m)" >> OS_info.txt;;
    k) echo "Exporting Kenel information.";
        echo "Kernel: $(uname -s)" > Kernel_info.txt;
        echo "Release Version: $(uname -r)" >> Kernel_info.txt;;
    c) echo "Exporting CPU information.";
        cat /proc/cpuinfo > CPU_info.txt;;
    *) echo "Error: An invalid argument has been given. Only use -o
     -k or -c as arguments.";;
  esac
done
```

Example 3: Words, Vowels, and Consonants.

Given the file sentences.txt, implement a bash script that:

1. Counts the total amount of words in a single sentence, and in all sentences in the file.
2. Counts the total amount of vowels in a single sentence, and in all sentences in the file.
3. Counts the total amount of consonants in a single sentence, and in all sentences in the file.

You can assume that only one sentence is written per line in sentences.txt.

```
#!/bin/bash

count_words=0
total_vowels=0
total_consonants=0

while read -r line; do
  words=$(echo "$line" | tr ' ' '\n' | wc -l)
  count_words=$((count_words + words))

  vowels=$(echo "$line" | grep -Eo '[aeiouAEIOU]' | wc -l)
  total_vowels=$((total_vowels + vowels))

  consonants=$(echo "$line" | grep -Eo '[
    bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]' | wc -l)
  total_consonants=$((total_consonants + consonants))

  echo "Sentence: $line"
  echo "Vowel count: $vowels"
  echo "Consonant count: $consonants"
  echo "Word count: $words"

done < sentences.txt

echo "Total words: $count_words"
echo "Total vowels: $total_vowels"
echo "Total consonants: $total_consonants"
```

Example 4

Implement a bash script that reads the contents of a string and slices a section of it. The sliced section starts from the first uppercase letter it finds, and finishes at the second upper letter.

```
#!/bin/bash

string="For infrastructure technology, C will be hard to displace. Dennis Ritchie."

uppercase_found=0
start=-1
end=-1

for (( i=0; i<${#string}; i++ )); do
  if [[ "${string:$i:1}" =~ ^[A-Z]$ ]]; then
    if [[ $uppercase_found -eq 0 ]]; then
      start=$i
      uppercase_found=1
    elif [[ $uppercase_found -eq 1 ]]; then
      end=$i
      break
    fi
  fi
done

if [[ $start -gt -1 && $end -gt -1 ]]; then
  echo "${string:$start:$(($end - $start + 1))}"
  #echo "plop"
else
  echo "Error: Could not find two uppercase letters."
fi
```

Other useful tools

cut

```
# Cut slices strings in terms of specific parameters.
sentence="For infrastructure technology, C will be hard to displace. Dennis Ritchie."

# -c (characters) cuts the string in terms of its characters (in the order they are written).
echo "$sentence" | cut -c 32

# It cuts multiple letters.
echo "$sentence" | cut -c 1,32

# Or segments of letters determined by their start and end positions.
echo "$sentence" | cut -c 1-32
```

awk

```
# awk understand the strings in terms of fields.
# Field $0 is the whole line, $1 is the forst field, $2 the second, etc.
cat w40k.txt | awk '{print $0}'

cat w40k.txt | awk '{print $1}'

cat w40k.txt | awk '{print $1,$2}'

# $NF denotes the last field.
cat w40k.txt | awk '{print $1,$NF}'

# OFS allows us to declare custom separators.
cat w40k.txt | awk 'OFS="-" {print$1,$2,$3}'

# Separators can be used together.
cat w40k.txt | awk 'OFS="\t-\t" {print$1,$2,$3}'

# We can Declare BEGIN and END rules.
# They are executed before and after the text processing.
awk 'BEGIN {print "* DISCOVER THE WARHAMMER FACTIONS *"} OFS="-" {print $1,$2}' w40k.txt
```

Developed by Diego Navarro.