LING 408/508: Programming for Linguists

Lecture 8
September 23rd
Adminstrivia

• Homework 4 graded
Today's Topics

• Homework 4 review
• Ungraded Exercise review

• There's SO much more...
  – but this is probably the 2nd last lecture about the bash shell
Homework 4 Review

• Exercise 1:
  – Write a bash shell script that simple accepts command line arguments and prints out the number of arguments, and each argument numbered on a separate line

• Examples:
  – `bash args.sh a b c`
    Args: 3
    #1: a
    #2: b
    #3: c
Homework 4 Review

Many different ways to write the solution...

```bash
#!/bin/bash
args=0
echo "Args: $#"
while [ $# -ne 0 ]
do
  ((args=args+1))
echo "#$args: $1"
  shift
done
exit 0
```

```bash
#!/bin/bash
echo "Args: $#"
c=1
for i in $@
do
echo "#$c: $1"
  ((c=$c+1))
  shift
done
exit 0
```
Homework 4 Review

Exercise 2

• Consider:
  – \( x=$(echo "scale=4;4*a(1)" \| bc -l) \)
  – echo \$x
  – 3.1412 \( (\text{computes } \pi \text{ to } 4 \text{ decimal places}) \)

• Explain why
  – \( x=$(echo "scale=4;4*a(1)" \| bc -l)) \) is wrong

\( $(\ldots \text{shell command substitution } \ldots ) \)
also backquote: `\ldots \text{command} \ldots`
Homework 4 Review

Exercise 3: Remove File/Directory

- `rm` removes a file (*dangerous!*)
- `rm -d` removes a directory (*empty*)
- `rm -r` recursive remove (*extreme danger!*)
- `rm -rf` forced recursive remove (!!!)

• Examples:
  - `touch file.txt`  
  - `rm file.txt` (you have default write permission)
  - `touch protected.txt`
  - `chmod u-w protected.txt` (*u = user, -w = remove write permission*)
  - `rm protected.txt`
  - `override r--r--r-- sandiway/staff for protected.txt?`
  - `rm -f protected.txt` (*no interaction: forced removal*)
  - `rm -i file.txt` (ask it to ask you for confirmation)
  - `remove file.txt?`
best used in interactive shell

• can put alias shortcut in terminal startup `~/.bash_profile` (`OSX`) or `~/.bashrc`

• alias `rm="rm -i"` not recursively expanded (considered dangerous: why?)

• alias

• unalias `rm`

• Aliases don't work in shell scripts:

```bash
#!/bin/bash
if [ $# -ne 1 ]; then
    echo "usage: filename"
    exit 1
fi
touch $1
rm $1
```

```bash
define a function in `~/.bash_profile` (absolute path: otherwise recursive)
rm () {
    /bin/rm -i "$@
}
```

At least two reasons:  
1. another computer  
2. shell scripts

• `rm -i` won't be called!
Other commands with -i

- i (interactive confirm option) before overwriting a file
  - mv  -i  rename file
  - cp  -i  copy file
awk

• Example:
  – Top 30 surnames and percentages in the Canary Islands according to wikipedia
  – [Link](https://en.wikipedia.org/wiki/List_of_the_most_common_surnames_in_Europe)
  – Filename: surnames.txt (3 fields: rank, name, and percentage of population)

```
1  González  4.79
2  Rodríguez  4.64
3  Hernández  4.01
4   Pérez  3.35
5   García  3.25
6   Martín  2.21
7  Santana  2.18
8    Díaz  1.86
9   Suárez  1.38
10  Sánchez 1.29
```

Run the following awk code to figure out what the code does:
1. ```awk '{ print $2; }' surnames.txt```
2. ```awk '{ if ($3>=1) {print $2;}}' surnames.txt```
Homework 4 Review

Exercise 4

- Write awk code to:
  1. print a table of and calculate the total percentage of population for the top 10, 20 and 30 surnames
  2. read and print out the table with table headings aligned with the field values

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Approximate percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>González</td>
<td>4.79</td>
</tr>
<tr>
<td>2</td>
<td>Rodríguez</td>
<td>4.64</td>
</tr>
<tr>
<td>3</td>
<td>Hernández</td>
<td>4.01</td>
</tr>
<tr>
<td>4-5</td>
<td>Pérez, García</td>
<td>3.35, 3.25</td>
</tr>
</tbody>
</table>
Homework 4 Review

• Exercise 4: *Let's develop the answer step-by-step*
  
  - `surnames.txt`: $1 \quad $2 \quad $3
  
  - `rank name percentage`

  - `awk '{ sum+=$3 } END {print sum}' surnames.txt`
  
  - 42.42

  - `awk '{ if ($1 <= 10) sum+=$3 } END {print sum}' surnames.txt`
  
  - 28.96

  - ...

  - `awk 'BEGIN {format=%-5s %-10s %s\n}; printf format, "Rank", "Name", "%"} { if ($1 <= 10) {sum+=$3; printf format, $1, $2, $3} } END {print "Total:", sum, "%"}' surnames.txt`
Homework 4 Review

• Throwaway programming:
  – awk 'BEGIN {format="%-5s %-10s %s\n"; printf format, "Rank", "Name", "%"};
    { if ($1 <= 10) {sum+=$3; printf format, $1, $2, $3} } END
    {print "Total:\", sum, "\%"};' surnames.txt

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>González</td>
<td>4.79</td>
</tr>
<tr>
<td>2</td>
<td>Rodríguez</td>
<td>4.64</td>
</tr>
<tr>
<td>3</td>
<td>Hernández</td>
<td>4.01</td>
</tr>
<tr>
<td>4</td>
<td>Pérez</td>
<td>3.35</td>
</tr>
<tr>
<td>5</td>
<td>García</td>
<td>3.25</td>
</tr>
<tr>
<td>6</td>
<td>Martín</td>
<td>2.21</td>
</tr>
<tr>
<td>7</td>
<td>Santana</td>
<td>2.18</td>
</tr>
<tr>
<td>8</td>
<td>Díaz</td>
<td>1.86</td>
</tr>
<tr>
<td>9</td>
<td>Suárez</td>
<td>1.38</td>
</tr>
<tr>
<td>10</td>
<td>Sánchez</td>
<td>1.29</td>
</tr>
<tr>
<td></td>
<td>Total:</td>
<td>28.96 %</td>
</tr>
</tbody>
</table>
awk and UTF-8

• By default, awk's formatted printing (printf) counts bytes not # of characters.
  – (This can be considered to be a BUG!)
  – with UTF-8 character encoding (up to 4 bytes used per character) this can create incorrect formatting →

• Example:
  – awk '{ printf "|%-13s|" $2 }' surnames.txt

• Sometimes using gawk (GNU awk) does the right thing
  – sudo apt-get install gawk
awk and UTF-8

• set the language environment
  – export LANG=en_US.UTF-8

• Re-run command:
  – awk '{ printf "|%-13s|" $2 }' surnames.txt
awk and UTF-8

• May also need:
awk and UTF-8

• To make default on login:

```bash
sandiway@sandiway-VirtualBox:~$ more /etc/default/locale
LANG=en_US.UTF-8
LC_ALL=en_US.UTF-8
sandiway@sandiway-VirtualBox:~$ sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
sandiway@sandiway-VirtualBox:~$ more /etc/default/locale
LANG=en_US.UTF-8
LC_ALL=en_US.UTF-8
```
Ungraded Exercise Review

• Changing the line spacing of a text file
  1. Write a script that reads each line of a file, then writes the line back out, but with an extra blank line following. This has the effect of *double-spacing* the file.

  **Note:** include if possible a check to see whether the script gets the necessary command-line argument (a filename), and whether the specified file exists.

(Exercise adapted from http://tldp.org/LDP/abs/html/writingscripts.html)
Ungraded Exercise Review

• *double-spacing* the file:

```bash
#!/bin/bash
read ln
while [[ -n "$ln" ]]; do
  echo $ln
  echo
done
```

If this option is given, backslash does not act as an escape character.

```
sBS2893:ling508-15 sandiway$ more test.txt
this is line one.
this is line two.
this is line three.
this is the last line.
sBS2893:ling508-15 sandiway$ bash doublespace.sh < test.txt
this is line one.
this is line two.
this is line three.
this is the last line.
```

`read -r`
If this option is given, backslash does not act as an escape character.
Ungraded Exercise Review

• **double-spacing** the file:

```bash
#!/bin/bash
if [[ -r $1 ]]; then
    while read -r ln; do
        echo $ln
        echo done < "$1"
    done
else
    echo "Can't read $1"
    exit 1
fi
```

whitespace trim problem workaround:

```bash
while IFS=''; read -r ln; do
```
Ungraded Exercise Review

• Changing the line spacing of a text file

2. Next, write a script to echo all lines of a file except for blank lines.

```bash
#!/bin/bash
if [[ -r $1 ]]; then
    while IFS=''; read -r ln; do
        if [[ -n $ln ]]; then
            echo $ln
        fi
    done < "$1"
else
    echo "Can't read $1"
exit 1
fi
```
Useful tidbits

• Pre-programmed interaction:
  – (here document: inline file)

```
rm () {
  /bin/rm -i "@$"
}
export -f rm

#!/bin/bash
if [ $# -ne 1 ]; then
  echo "usage: filename"
  exit 1
fi
touch $1
rm $1
```

```
#!/bin/bash
bash confirm.sh $1 <<EOF
y
EOF
```

```
#!/bin/bash
bash confirm.sh $1 <<<y
```
Interaction using expect

- **expect** is a program that executes preprogrammed interaction using commands including:
  - `expect string` *look for string in input*
  - `send string` *respond with string (/r for return)*
  - `spawn string` *execute a program under expect*
  - `interact` *gives control to user*

  *output will appear on terminal*

```bash
#!/usr/bin/expect -f
expect "hello"
send "world"
```
Interaction using expect

• Knock knock joke expect script:
  1. `#!/usr/bin/expect -f`
  2. `expect "knock knock\n"
  3. `send "who's there?\n"
  4. `expect -re "(.*)\n"
  5. `send "$expect_out(1,string) who?\n"
  6. `expect "\n"
  7. `send "Very funny\n"

Kangaroo knock knock joke...
Interaction using expect

• Let's interact with the BMI homework solution ...