Bash Scripting Cont’d

# Exercise

Rewrite our ls\_timer4.sh script (the one where we specify options for our parameter, -l -e etc) to utilize a case statement. Also, add in usage information for the script in the event that the user does not provide correct parameters

For example, running the command ls\_timer4.sh with incorrect parameters would output the following:

USAGE: ./ls\_timer4.sh [-l|-e|-?] <directory>\b\t\-l – Specify file to save listing to\n\t-e – Specify file to save error log to\n\t-? – Display these usage notes”

|  |
| --- |
| #!/bin/bash  # Does a recusive directory listing and times it  USAGE="USAGE: ./ls\_exercise.sh [-l|-e|-?|-h] <directory>\n\t-l - Specify file to save listing to\n\t-e - Specify file to save error log to\n\t-? - Display these usage notes"  directory\_listing=my\_dirs  error\_log=my\_errs  while [[ $# > 1 ]]  do  key=$1  shift  case $key in  -l)  #sets the location of the directory listing  echo "Directory listing will be in $1"  directory\_listing=$1  shift  ;;  -e)  #set the location of the error log  echo "Error log will be in $1"  error\_log=$1  shift  ;;  -\?|-h)  echo -e $USAGE  exit 1  ;;  \*)  #if none of our arguments match, we've got something invalid  echo "Invalid arguments. Specify the -h option for help."  echo -e $USAGE  ;;  esac  done  target\_directory=$1 # since we shifted a couple of times above, this will be <mytarget>  echo $target\_directory  if [ ! -d $target\_directory ] # tests whether our directory is in fact a directory, ! negates  then  # not a directory  echo "$target\_directory is not a valid directory. Need to specify a valid target directory."  exit 1  fi  echo "Doing a recursive listing of $target\_directory"  start\_time=$(date +%s%N)  if ls -R $target\_directory > ${directory\_listing} 2> ${error\_log}  then  echo "Done. Took $((($(date +%s%N) - start\_time)/1000000)) ms."  else  echo "Could not list all directories."  fi |

Comments:

* USAGE variable will be printed out on the event that the actual usage of the script does not conform to the intended design
* **while [[ $# > 1 ]]**
  + This loops through the arguments provided to the script at runtime
  + $# is a placeholder for “number of arguments”
* -\?|-h)
  + Specifies an or clause that checks for -? or -h

|  |
| --- |
| !/bin/bash  # Does a recusive directory listing and times it  USAGE="USAGE: ./ls\_ex2.sh [-l|-e|-?|-h] <directory>\n\t-l - Specify file to save listing to\n\t-e - Specify file to save error log to\n\t-? - Display these usage notes"  directory\_listing=my\_dirs  error\_log=my\_errs  while [[ $# > 1 ]]  do  key=$1  shift  case $key in  -l) #sets the location of the directory listing  echo "Directory listing will be in $1"  directory\_listing=$1  shift  ;;  -e) #set the location of the error log  echo "Error log will be in $1"  error\_log=$1  shift  ;;  -\?|-h) echo -e $USAGE  ;;  -i) #interactive mode - take our directyory from the user via prompt  interactive\_mode=1  ;;  \*) #if none of our arguments match, we've got something invalid  echo "Invalid arguments. Specify the -h option for help."  echo -e $USAGE  exit 1  ;;  esac  done  target\_directory=$1 # since we shifted a couple of times above, this will be <mytarget>  if [ ! -d $target\_directory ] # tests whether our directory is in fact a directory, ! negates  then  # not a directory  echo "$target\_directory is not a valid directory. Need to specify a valid target directory."  echo -e $USAGE  exit 1  fi  echo "interactive mode: $interactive\_mode"  if [[ interactive\_mode -eq 1 ]]  then  echo "In if statement"  read -p "About to do a recursive listing of $target\_directory. Continue? [Y/n]" cprompt  cprompt=$(echo "$cprompt" | awk '{print tolower($0)}')  if [[ !( $cprompt =~ ^(Y|y)) ]]  then  echo "Aborting ls script."  exit 0  fi  fi  echo "Doing a recursive listing of $target\_directory"  start\_time=$(date +%s%N)  if ls -R $target\_directory > ${directory\_listing} 2> ${error\_log}  then  echo "Done. Took $((($(date +%s%N) - start\_time)/1000000)) ms."  else  echo "Could not list all directories."  fi |

Comments:

* -i)
  + Turns on our custom interactive mode, where we’ll accept input from the user to confirm the execution of the script on the desired directory
  + We create a variable called “interactive\_mode” and set it to 1
    - This is really just a flag
* if [[ interactive\_mode -eq 1 ]]
  + Check our flag to see if we turned it on during runtime
  + If we did, prompt the user for feedback via the **read** command
  + Check the value in $cprompt with =~
    - This checks the value of the LHS with the regular expression on the RHS
    - If so, echo feedback and exit the script via exit 0
* Read is straightforward....for awk and sed, maybe look at http://tldp.org/LDP/abs/html/sedawk.html if we have TIME? Otherwise just man pages. Awk is a full-featured text processing language. awk deals with fields - pattern action.

## Sed and Awk

sed is a stream editor.

awk is a field-oriented pattern processing language with a C-Style syntax

sed s/pattern/replace is very, very useful

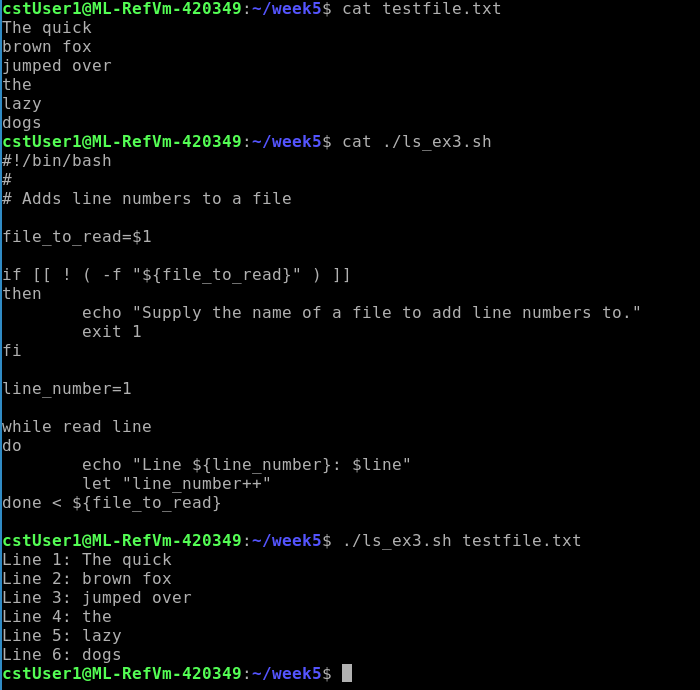
echo "Some te3xt here." | sed 's/te3xt/text/'

ps aux | sed 's/^root/not\_root/'

ps aux | awk '/^insacc/ {print $11 " (" $2 ")"}' | sed 's/.\*\///'

## Reading from a file

We can read from a file by redirecting a file into the **read** command. It’s often done in a loop:



|  |
| --- |
| #!/bin/bash  #  # Adds line numbers to a file  file\_to\_read=$1  if [[ ! ( -f "${file\_to\_read}" ) ]]  then  echo "Supply the name of a file to add line numbers to."  exit 1  fi  line\_number=1  while read line  do  echo "Line ${line\_number}: $line"  let "line\_number++"  done < ${file\_to\_read} |

Comments:

* file\_to\_read=$1
  + capture our first argument and assign it to the variable “file\_to\_read”
* if [[ ! ( -f “${file\_to\_read}” ) ]]
  + Tests if the filename in “file\_to\_read” is an actual file
  + If not, exit the script
* While loop
  + We redirect the input of our read from the file we specified
  + We do some string manip to print line by line with line numbers
  + The let command will execute our arithmetic expression (line\_number++)