**Using Arguments**

**Example 10. Shell Script Arguments**

|  |
| --- |
| #!/bin/bash  # example of using arguments to a script  echo "My first name is $1"  echo "My surname is $2"  echo "Total number of arguments is $#" |

Save this file as name.sh, set execute permission on that file by typing **chmod a+x name.sh** and then execute the file like this: **./name.sh**.

|  |
| --- |
| $ chmod a+x name.sh  $ ./name.sh Hans-Wolfgang Loidl  My first name is Hans-Wolfgang  My surname is Loidl  Total number of arguments is 2 |

**Version 1: Line count example**

The first example simply counts the number of lines in an input file. It does so by iterating over all lines of a file using a **while** loop, performing a **read** operation in the loop header. While there is a line to process, the loop body will be executed in this case simply increasing a counter by **((counter++))**. Additionally the current line is written into a file, whose name is specified by the variable file, by echoing the value of the variable line and redirecting the standard output of the variable to **$file**. the current line to file. The latter is not needed for the line count, of course, but demonstrates how to check for success of an operation: the special variable **$?** will contain the return code from the previous command (the redirected **echo**). By Unix convention, success is indicated by a return code of 0, all other values are error code with application specific meaning.

Another important issue to consider is that the integer variable, over which iteration is performed should always *count down* so that the analysis can find a bound. This might require some restructuring of the code, as in the following example, where an explicit counter z is introduced for this purpose. After the loop, the line count and the contents of the last line are printed, using **echo**. Of course, there is a Linux command that already implements line-count functionality: **wc** (for word-count) prints, when called with option **-l**, the number of lines in the file. We use this to check wether our line count is correct, demonstrating numeric operations on the way.

|  |
| --- |
| #!/bin/bash  # Simple line count example, using bash  #  # Bash tutorial: http://linuxconfig.org/Bash\_scripting\_Tutorial#8-2-read-file-into-bash-array  # My scripting link: http://www.macs.hw.ac.uk/~hwloidl/docs/index.html#scripting  #  # Usage: ./line\_count.sh file  # -----------------------------------------------------------------------------  # Link filedescriptor 10 with stdin  exec 10<&0  # stdin replaced with a file supplied as a first argument  exec < $1  # remember the name of the input file  in=$1  # init  file="current\_line.txt"  let count=0  # this while loop iterates over all lines of the file  while read LINE  do  # increase line counter  ((count++))  # write current line to a tmp file with name $file (not needed for counting)  echo $LINE > $file  # this checks the return code of echo (not needed for writing; just for demo)  if [ $? -ne 0 ]  then echo "Error in writing to file ${file}; check its permissions!"  fi  done  echo "Number of lines: $count"  echo "The last line of the file is: `cat ${file}`"  # Note: You can achieve the same by just using the tool wc like this  echo "Expected number of lines: `wc -l $in`"  # restore stdin from filedescriptor 10  # and close filedescriptor 10  exec 0<&10 10<&- |

As documented at the start of the script, it is called like this (you must have a file text\_file.txt in your current directory):

|  |
| --- |
| $ ./line\_count.sh text\_file.txt |

**Several versions of line counting across a set of files**

This section develops several shell scripts, each counting the total number of lines across a set of files. These examples elaborate specific shell features. For counting the number of lines in one file we use **wc -l**. As a simple exercise you can replace this command with a call to the line counting script above.

**Version 1: Explicit For loop**

We use a for-loop to iterate over all files provided as arguments to the script. We can access all arguments through the variable $\*. The sed command matches the line count, and replaces the entire line with just the line count, using the back reference to the first substring (\1). In the for-loop, the shell variable n is a counter for the number of files, and s is the total line count so far.

|  |
| --- |
| #!/bin/bash  # Counting the number of lines in a list of files  # for loop over arguments  if [ $# -lt 1 ]  then  echo "Usage: $0 file ..."  exit 1  fi  echo "$0 counts the lines of code"  l=0  n=0  s=0  for f in $\*  do  l=`wc -l $f | sed 's/^\([0-9]\*\).\*$/\1/'`  echo "$f: $l"  n=$[ $n + 1 ]  s=$[ $s + $l ]  done  echo "$n files in total, with $s lines in total" |

**Version 2: Using a Shell Function**

In this version we define a function **count\_lines** that counts the number of lines in the file provided as argument. Inside the function the value of the argument is retrieved by accessing the variable $1.

|  |
| --- |
| #!/bin/bash  # Counting the number of lines in a list of files  # function version  count\_lines () {  local f=$1  # this is the return value, i.e. non local  l=`wc -l $f | sed 's/^\([0-9]\*\).\*$/\1/'`  }  if [ $# -lt 1 ]  then  echo "Usage: $0 file ..."  exit 1  fi  echo "$0 counts the lines of code"  l=0  n=0  s=0  while [ "$\*" != "" ]  do  count\_lines $1  echo "$1: $l"  n=$[ $n + 1 ]  s=$[ $s + $l ]  shift  done  echo "$n files in total, with $s lines in total" |

**Version 3: Using a return code in a function**

This version tries to use the return value of the function to return the line count. However, this fails on files with more than 255 lines. The return value is intended to just provide a return code, e.g. 0 for success, 1 for failure, but not for returning proper values.

|  |
| --- |
| #!/bin/bash  # Counting the number of lines in a list of files  # function version using return code  # WRONG version: the return code is limited to 0-255  # so this script will run, but print wrong values for  # files with more than 255 lines  count\_lines () {  local f=$1  local m  m=`wc -l $f | sed 's/^\([0-9]\*\).\*$/\1/'`  return $m  }  if [ $# -lt 1 ]  then  echo "Usage: $0 file ..."  exit 1  fi  echo "$0 counts the lines of code"  l=0  n=0  s=0  while [ "$\*" != "" ]  do  count\_lines $1  l=$?  echo "$1: $l"  n=$[ $n + 1 ]  s=$[ $s + $l ]  shift  done  echo "$n files in total, with $s lines in total" |

**Version 4: Generating the file list in a shell function**

|  |
| --- |
| #!/bin/bash  # Counting the number of lines in a list of files  # function version  # function storing list of all files in variable files  get\_files () {  files="`ls \*.[ch]`"  }  # function counting the number of lines in a file  count\_lines () {  local f=$1 # 1st argument is filename  l=`wc -l $f | sed 's/^\([0-9]\*\).\*$/\1/'` # number of lines  }  # the script should be called without arguments  if [ $# -ge 1 ]  then  echo "Usage: $0 "  exit 1  fi  # split by newline  IFS=$'\012'  echo "$0 counts the lines of code"  # don't forget to initialise!  l=0  n=0  s=0  # call a function to get a list of files  get\_files  # iterate over this list  for f in $files  do  # call a function to count the lines  count\_lines $f  echo "$f: $l"  # increase counter  n=$[ $n + 1 ]  # increase sum of all lines  s=$[ $s + $l ]  done  echo "$n files in total, with $s lines in total" |

**Version 5: Using an array to store all line counts**

The example below uses shell arrays to store all filenames (file) and its number of lines (line). The elements in an array are referred to using the usual [ ] notation, e.g. file[1] refers to the first element in the array file. Note, that bash only supports 1-dimensional arrays with integers as indizes.

See [the section on arrays in the Advanced Bash-Scripting Guide:](http://tldp.org/LDP/abs/html/arrays.html).

|  |
| --- |
| #!/bin/bash  # Counting the number of lines in a list of files  # function version  # function storing list of all files in variable files  get\_files () {  files="`ls \*.[ch]`"  }  # function counting the number of lines in a file  count\_lines () {  f=$1 # 1st argument is filename  l=`wc -l $f | sed 's/^\([0-9]\*\).\*$/\1/'` # number of lines  }  # the script should be called without arguments  if [ $# -ge 1 ]  then  echo "Usage: $0 "  exit 1  fi  # split by newline  IFS=$'\012'  echo "$0 counts the lines of code"  # don't forget to initialise!  l=0  n=0  s=0  # call a function to get a list of files  get\_files  # iterate over this list  for f in $files  do  # call a function to count the lines  count\_lines $f  echo "$f: $l"loc  # store filename in an array  file[$n]=$f  # store number of lines in an array  lines[$n]=$l  # increase counter  n=$[ $n + 1 ]  # increase sum of all lines  s=$[ $s + $l ]  done  echo "$n files in total, with $s lines in total"  i=5  echo "The $i-th file was ${file[$i]} with ${lines[$i]} lines" |

**Version 6: Count only files we own**

|  |
| --- |
| #!/bin/bash  # Counting the number of lines in a list of files  # for loop over arguments  # count only those files I am owner of  if [ $# -lt 1 ]  then  echo "Usage: $0 file ..."  exit 1  fi  echo "$0 counts the lines of code"  l=0  n=0  s=0  for f in $\*  do  if [ -O $f ] # checks whether file owner is running the script  then  l=`wc -l $f | sed 's/^\([0-9]\*\).\*$/\1/'`  echo "$f: $l"  n=$[ $n + 1 ]  s=$[ $s + $l ]  else  continue  fi  done  echo "$n files in total, with $s lines in total" |

**Version 7: Line count over several files**

The final example supports options that can be passed from the command-line, e.g. by **./loc7.sh -d 1 loc7.sh**. The getopts shell function is used to iterate over all options (given in the following string) and assigning the current option to variable name. Typically it is used in a while loop, to set shell variables that will be used later. We use a pipe of **cat** and **awk** to print the header of this file, up to the first empty line, if the help option is chosen. The main part of the script is a for loop over all non-option command-line arguments. In each iteration, $f contains the name of the file to process. If the date options are used to narrow the scope of files to process, we use the **date** and an if-statement, to compare whether the modification time of the file is within the specified interval. Only in this case do we count the number of lines as before. After the loop, we print the total number of lines and the number of files that have been processed.

**Example 11. Version 7: Line count over several files**

|  |
| --- |
| #!/bin/bash  ############################################################################  #  # Usage: loc7.sh [options] file ...  #  # Count the number of lines in a given list of files.  # Uses a for loop over all arguments.  #  # Options:  # -h ... help message  # -d n ... consider only files modified within the last n days  # -w n ... consider only files modified within the last n weeks  #  # Limitations:  # . only one option should be given; a second one overrides  #  ############################################################################  help=0  verb=0  weeks=0  # defaults  days=0  m=1  str="days"  getopts "hvd:w:" name  while [ "$name" != "?" ] ; do  case $name in  h) help=1;;  v) verb=1;;  d) days=$OPTARG  m=$OPTARG  str="days";;  w) weeks=$OPTARG  m=$OPTARG  str="weeks";;  esac  getopts "hvd:w:" name  done  if [ $help -eq 1 ]  then no\_of\_lines=`cat $0 | awk 'BEGIN { n = 0; } \  /^$/ { print n; \  exit; } \  { n++; }'`  echo "`head -$no\_of\_lines $0`"  exit  fi  shift $[ $OPTIND - 1 ]  if [ $# -lt 1 ]  then  echo "Usage: $0 file ..."  exit 1  fi  if [ $verb -eq 1 ]  then echo "$0 counts the lines of code"  fi  l=0  n=0  s=0  for f in $\*  do  x=`stat -c "%y" $f`  # modification date  d=`date --date="$x" +%y%m%d`  # date of $m days/weeks ago  e=`date --date="$m $str ago" +%y%m%d`  # now  z=`date +%y%m%d`  #echo "Stat: $x; Now: $z; File: $d; $m $str ago: $e"  # checks whether file is more recent then req  if [ $d -ge $e -a $d -le $z ] # ToDo: fix year wrap-arounds  then  # be verbose if we found a recent file  if [ $verb -eq 1 ]  then echo "$f: modified (mmdd) $d"  fi  # do the line count  l=`wc -l $f | sed 's/^\([0-9]\*\).\*$/\1/'`  echo "$f: $l"  # increase the counters  n=$[ $n + 1 ]  s=$[ $s + $l ]  else  # not strictly necessary, because it's the end of the loop  continue  fi  done  echo "$n files in total, with $s lines in total |