Laboratory Experiment VI

SHELL SCRIPTING WITH ARGUMENTS

by Alex Shaw III

Unedited Version

May 24, 2005

TABLE OF CONTENTS

INTRODUCTION

This experiment involves scripting with arguments. An argument is part of a command that is used for a specific purpose, such as an operation. For example, the **cp** command utilizes two arguments—a source file and a destination file.

Syntax of the **cp** command is shown in **Diagram 1-1**.

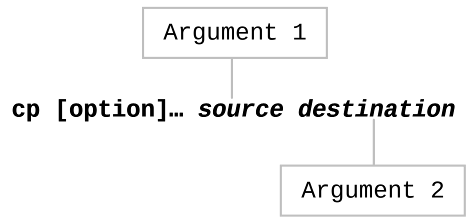


Diagram 1-1 General syntax of the **cp** command; however, there are other formats. The main purpose is to understand what an argument is and where its placement.

The programs created in this experiment are—**twice**, **home**, **suffix**, and **rnfile**.[1] A complete description of each program is in the Source Code section.

[1] All programs in this experiment are presented in lowercase even if it starts a sentence because that is the way they were created in LINUX. It is important to preserve the integrity of the program, particularly since LINUX is case-sensitive.

1
LABORATORY SHEETS

LABORATORY EXPERIMENT VI

LEHMAN COLLEGE
Of the City University of New York

Experiment #6 Shell Script Programming

Objective: To Enhance Shell Script Programming Proficency using Arguments

1. Write a program called "twice" that takes a single integer argument and Doubles its value i.e. $ twice 15
   30

   What happens if a noninteger value is typed? What if the argument is omitted?

2. Describe the program in detail

3. Write a program called "home" that takes the name of a user as an argument and prints their home directory i.e. $ home steve gives /export/home/steve

   Describe all details.

4. Write a program called "suffix" that renames a file by adding the character given as the second argument to the end of the name of the file given as the first argument i.e. $ suffix memo1 sv renames memo1 to memo1.sv

   Modify the program to change the name altogether.

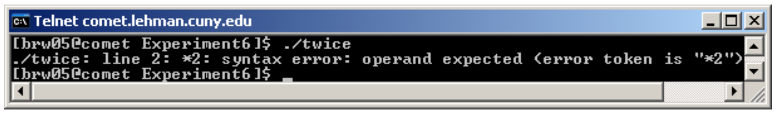
   Describe all details.

Sheet 1-1 Tasks for Experiment 6.

Tasks

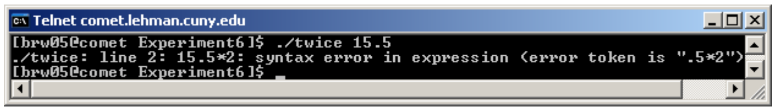
Sheet 1-1 provides the tasks to complete during this experiment.

2
DATA SHEETS



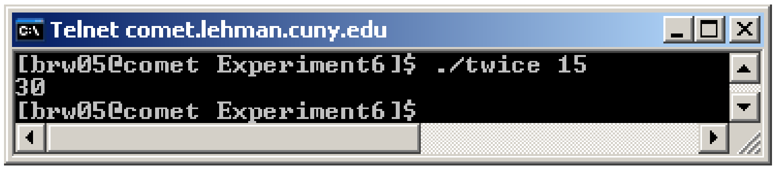
```
[brw05@comet Experiment6]$ ./twice
./twice: line 2: *2: syntax error: operand expected (error token is "*2")
[brw05@comet Experiment6]$ _
```

Output 2-1 Result of the **twice** program when run without an argument, which produces an error statement. A line in the script utilizes an argument to perform an operation.



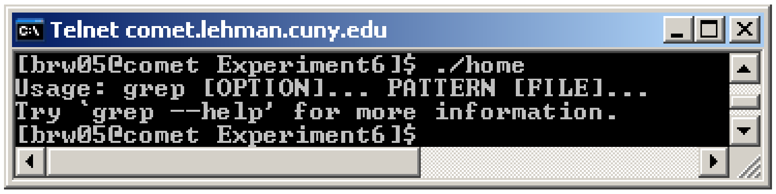
```
[brw05@comet Experiment6]$ ./twice 15.5
./twice: line 2: 15.5*2: syntax error in expression (error token is ".5*2")
[brw05@comet Experiment6]$ _
```

Output 2-2 Result of the **twice** program when run with a non-integer, which produces an error statement. Only integers are acceptable with this script.



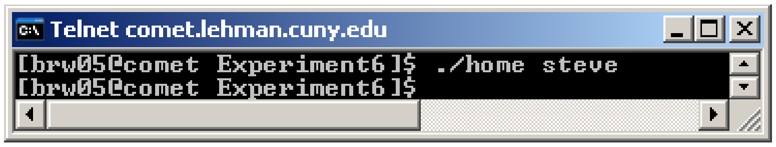
```
[brw05@comet Experiment6]$ ./twice 15
30
[brw05@comet Experiment6]$
```

Output 2-3 Result of the **twice** program when run with an integer. In this case, it produces 30.



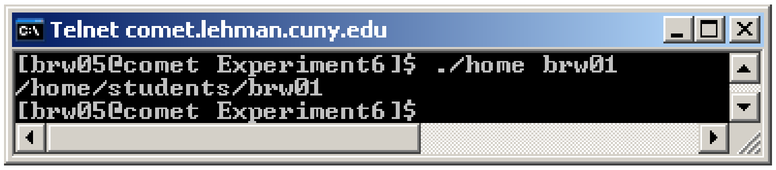
```
[brw05@comet Experiment6]$ ./home
Usage: grep [OPTION]... PATTERN [FILE]...
Try `grep --help' for more information.
[brw05@comet Experiment6]$
```

Output 2-4 Result of the **home** program when run without an argument, which produces an error statement based on the usage of **grep**. In the script, **grep** searches for the argument through the piped result of the **cut** command.



```
[brw05@comet Experiment6]$ ./home steve
[brw05@comet Experiment6]$
```

Output 2-5 Result of the **home** program when run with an argument. In this case, *steve* was not found as a user in the */etc/passwd* file.



```
[brw05@comet Experiment6]$ ./home brw01
/home/students/brw01
[brw05@comet Experiment6]$
```

Output 2-6 Result of the **home** program when run with an argument. In this case, *brw01* was found as a user in the */etc/passwd* file and their home directory is displayed.

Preliminary Results

Preliminary results of **twice**, **home** and **suffix** are presented in this section, along with **rnfile**, an additional command.

twice

twice produces an error when no argument is issued (**Output 2-1**).

twice produces an error when a non-integer argument is issued (**Output 2-2**).

Output 2-3 displays the result of a **twice** run with an integer.

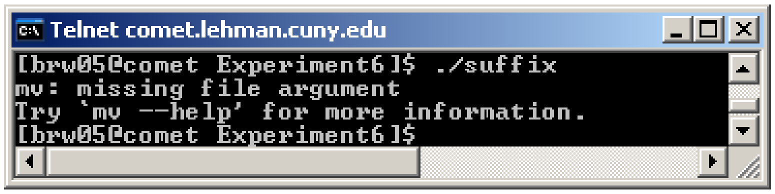
home

home produces an error when no argument is issued (**Output 2-4**).

In **Output 2-5**, **home** seems to work because no error statement or result message appears. Actually, it works fine because the user **steve**[1], in this example, does not exist; therefore, **steve** has no home directory and nothing is displayed.

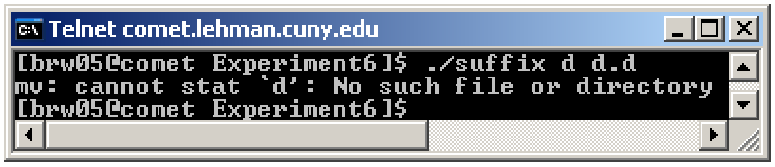
Output 2-6 shows the result of a perfectly executed command. Since **brw01** is a user, it displays their home directory.

[1] Although **steve** is a proper name, the case reflects how he was entered as an argument.



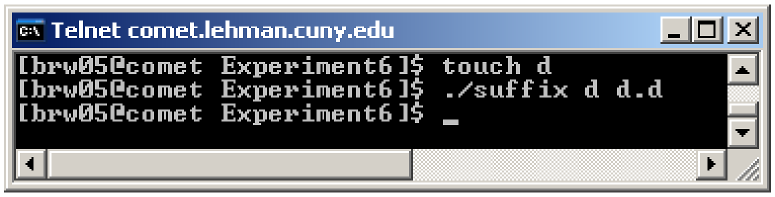
```
[brw05@comet Experiment6]$ ./suffix
mv: missing file argument
Try `mv --help' for more information.
[brw05@comet Experiment6]$
```

Output 2-7 Result of the **suffix** program when run without an argument, which produces an error statement on the usage of the **mv** command. The **suffix** command requires two arguments, the file to change and the changing file.



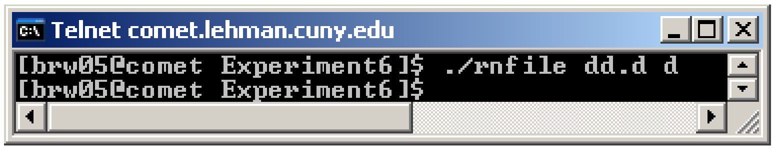
```
[brw05@comet Experiment6]$ ./suffix d d.d
mv: cannot stat `d': No such file or directory
[brw05@comet Experiment6]$
```

Output 2-8 Result of the **suffix** program when run with arguments. In this case, an error statement occurs because there is no such *d* file.



```
[brw05@comet Experiment6]$ touch d
[brw05@comet Experiment6]$ ./suffix d d.d
[brw05@comet Experiment6]$ _
```

Output 2-9 After the **touch** command created a blank *d* file, **suffix** appends *d.d* to d, which renames d to *dd.d*.



```
[brw05@comet Experiment6]$ ./rnfile dd.d d
[brw05@comet Experiment6]$
```

Output 2-10 Instead of appending a suffix to a file, the **rnfile** program renames two files. In this case, *dd.d* is returned to *d*, its original name. On errors, it produces similar results as in **suffix**.

suffix

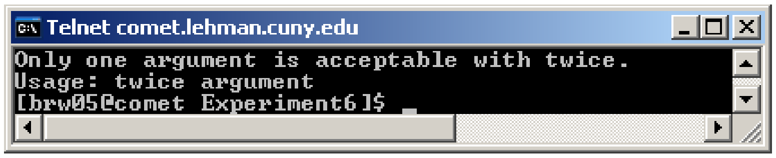
suffix produces an error when run without an argument (**Output 2-7**).

suffix produces an error when it attempts to append *d.d* to *d*. It is suppose to rename *d* to *dd.d* (**Output 2-8**).

In **Output 2-9**, the **touch** command creates the *d* file, enable **suffix** to perform its necessary operation.

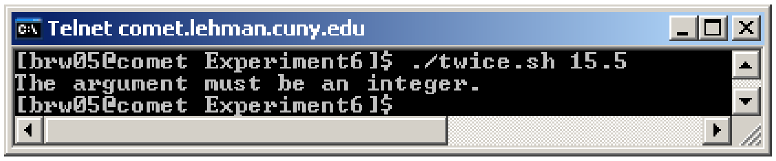
rnfile

Results of the **rnfile** command (**Output 2-10**) is similar to the results of **suffix** (**Output 2-9**). Instead of appending to a file, however, it renames it completely, from *dd.d* to *d*.



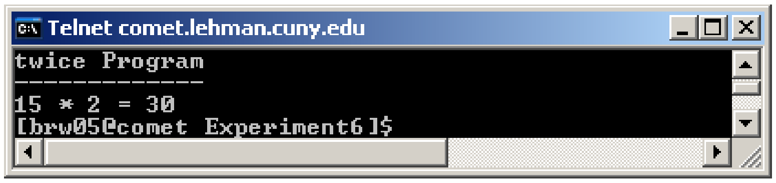
```
Only one argument is acceptable with twice.
Usage: twice argument
[brw05@comet Experiment6]$ _
```

Output 2-11 Result of the **twice.sh** program when run without an argument, which produces a custom error statement. A line in the script utilizes an argument to perform an operation.



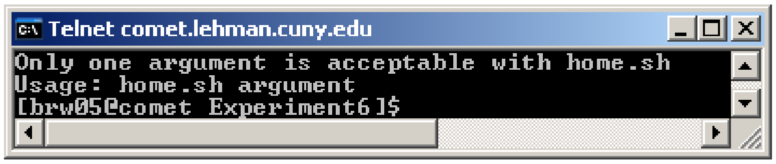
```
[brw05@comet Experiment6]$ ./twice.sh 15.5
The argument must be an integer.
[brw05@comet Experiment6]$
```

Output 2-12 Result of the **twice.sh** program when run with a non-integer, which produces a custom error statement. Only integers are acceptable with this script.



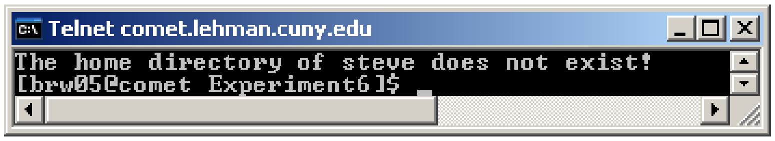
```
twice Program
-------------
15 * 2 = 30
[brw05@comet Experiment6]$
```

Output 2-13 Result of the **twice.sh** program when run with an integer. In this case, it produces 30.



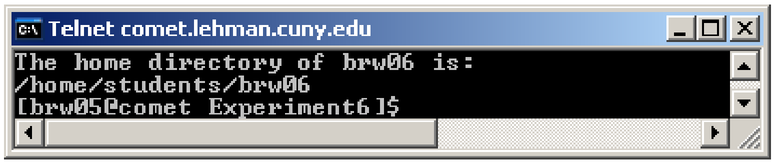
```
Only one argument is acceptable with home.sh
Usage: home.sh argument
[brw05@comet Experiment6]$
```

Output 2-14 Result of the **home.sh** program when run without an argument, which produces a custom error statement.



```
The home directory of steve does not exist!
[brw05@comet Experiment6]$ _
```

Output 2-15 Result of the **home.sh** program when run with an argument. In this case, *steve* was not found as a user in the */etc/passwd* file, which produces a custom error statement.



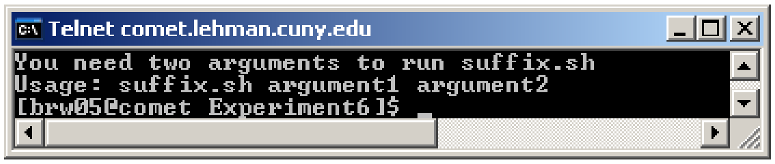
```
The home directory of brw06 is:
/home/students/brw06
[brw05@comet Experiment6]$
```

Output 2-16 Result of the **home.sh** program when run with an argument. In this case, *brw06* was found as a user in the */etc/passwd* file and their home directory is displayed.

Final Results

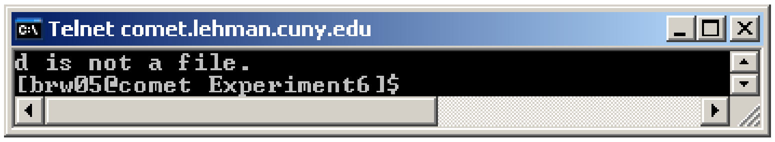
Corrections to the programs in the previous section provide a more user-friendly approach to accomplishing the same tasks.

The windows (**Output 2-11** to **Output 2-20**) display results of what happens when we try to run the same arguments on the revised programs, as in **Output 2-1** to **Output 2-10** in the preliminary results.



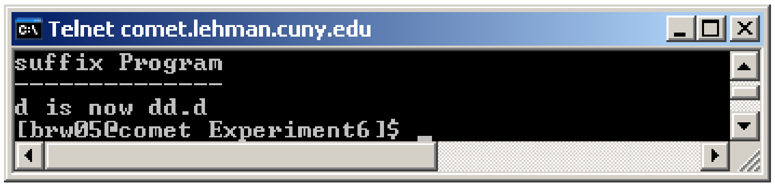
```
You need two arguments to run suffix.sh
Usage: suffix.sh argument1 argument2
[brw05@comet Experiment6]$ _
```

Output 2-17 Result of the **suffix.sh** program when run without an argument, which produces an error statement on the usage of the **mv** command. The **suffix** command requires two arguments, the file to change and the changing file.



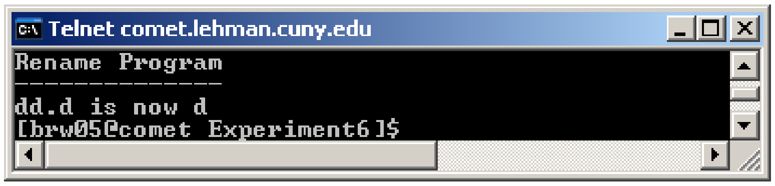
```
d is not a file.
[brw05@comet Experiment6]$
```

Output 2-18 Result of the **suffix.sh** program when run with arguments. In this case, an error statement occurs because there is no such *d* file.



```
suffix Program
--------------
d is now dd.d
[brw05@comet Experiment6]$ _
```

Output 2-19 After the **touch** command created a blank *d* file, **suffix.sh** appends *d.d* to d, which renames d to *dd.d*.



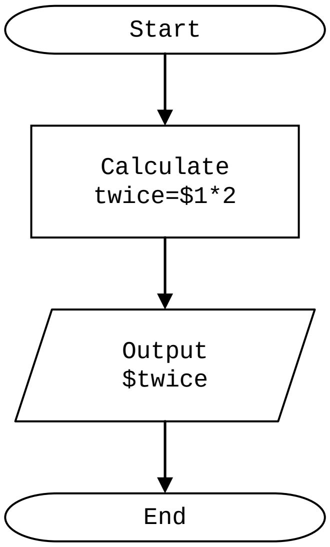
```
Rename Program
--------------
dd.d is now d
[brw05@comet Experiment6]$
```

Output 2-20 Instead of appending a suffix to a file, the **rnfile.sh** program renames two files. In this case, *dd.d* is returned to *d*, its original name. On errors, it produces similar results as in **suffix.sh**.

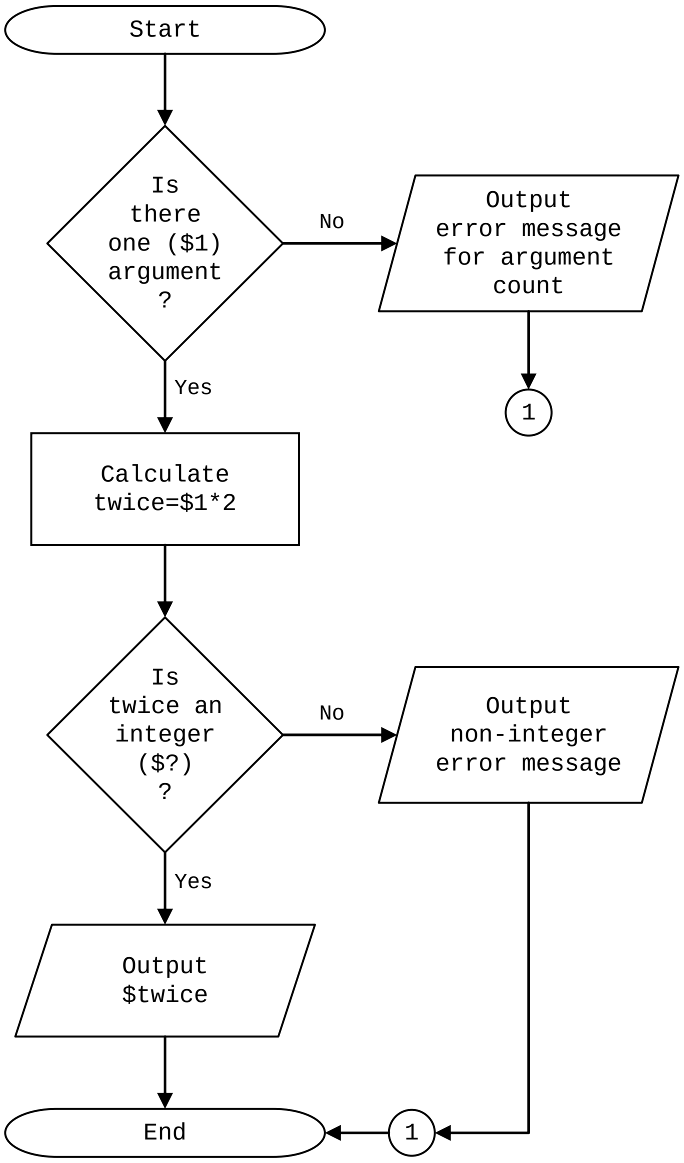
3 DIAGRAMS

Flowcharts

twice

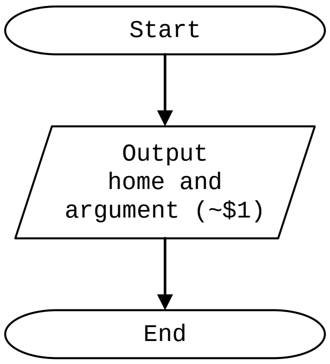


Flowchart 3-1 Semantics of the **twice** program.

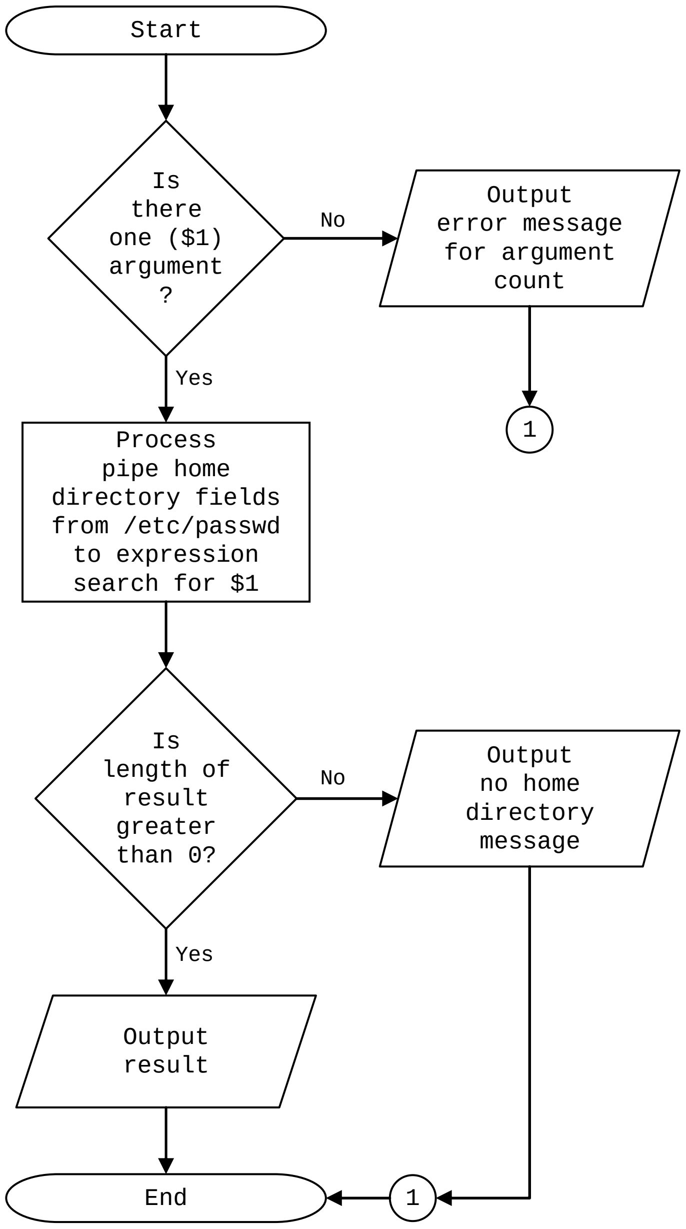

Start
Is there one ($1) argument ?
No
Output error message for argument count
1
Yes
Calculate twice=$1*2
Is twice an integer ($?) ?
No
Output non-integer error message
Yes
Output $twice
End
1

Flowchart 3-2 Semantics of the **twice** program (revised).

home

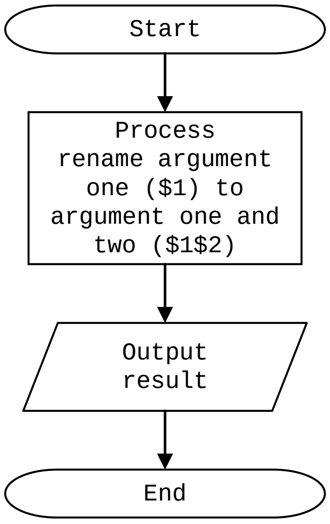


Flowchart 3-3 Semantics of the **home** program.

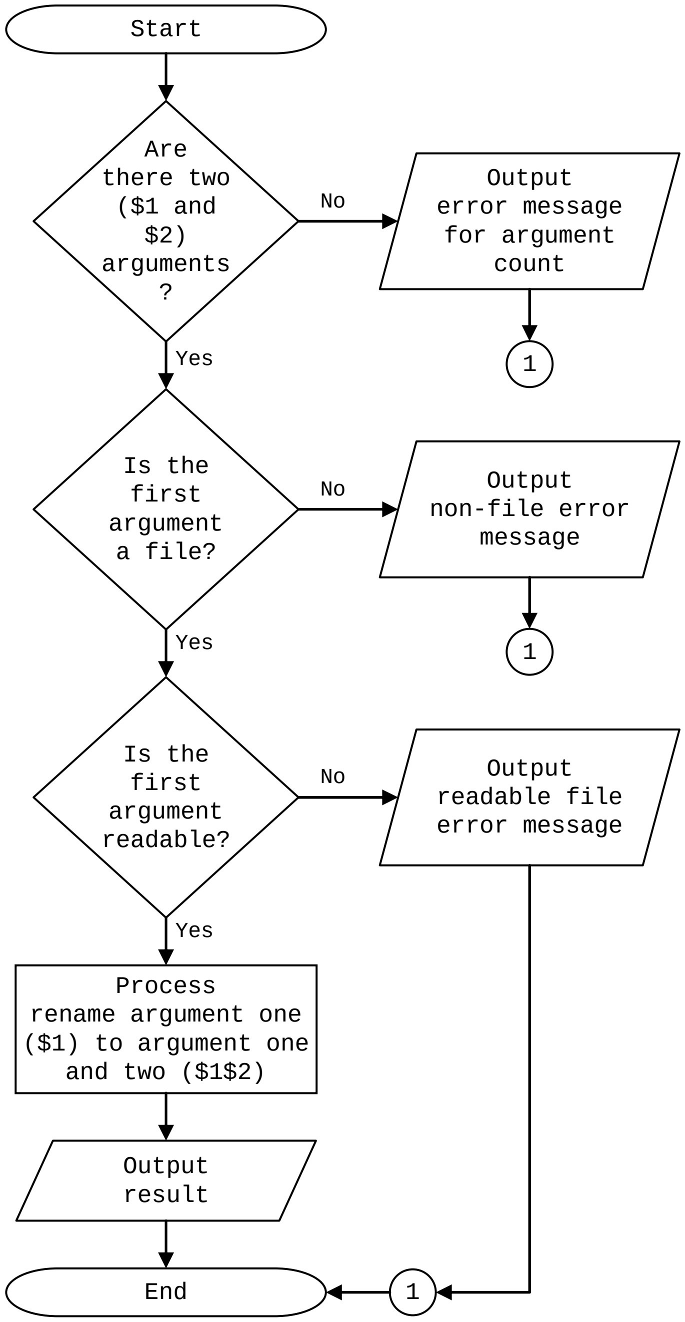

Start
Is there one ($1) argument ?
No
Output error message for argument count
1
Yes
Process pipe home directory fields from /etc/passwd to expression search for $1
Is length of result greater than 0?
No
Output no home directory message
Yes
Output result
End
1

Flowchart 3-4 Semantics of the **home** program (revised).

suffix


Start
Process rename argument one ($1) to argument one and two ($1$2)
Output result
End

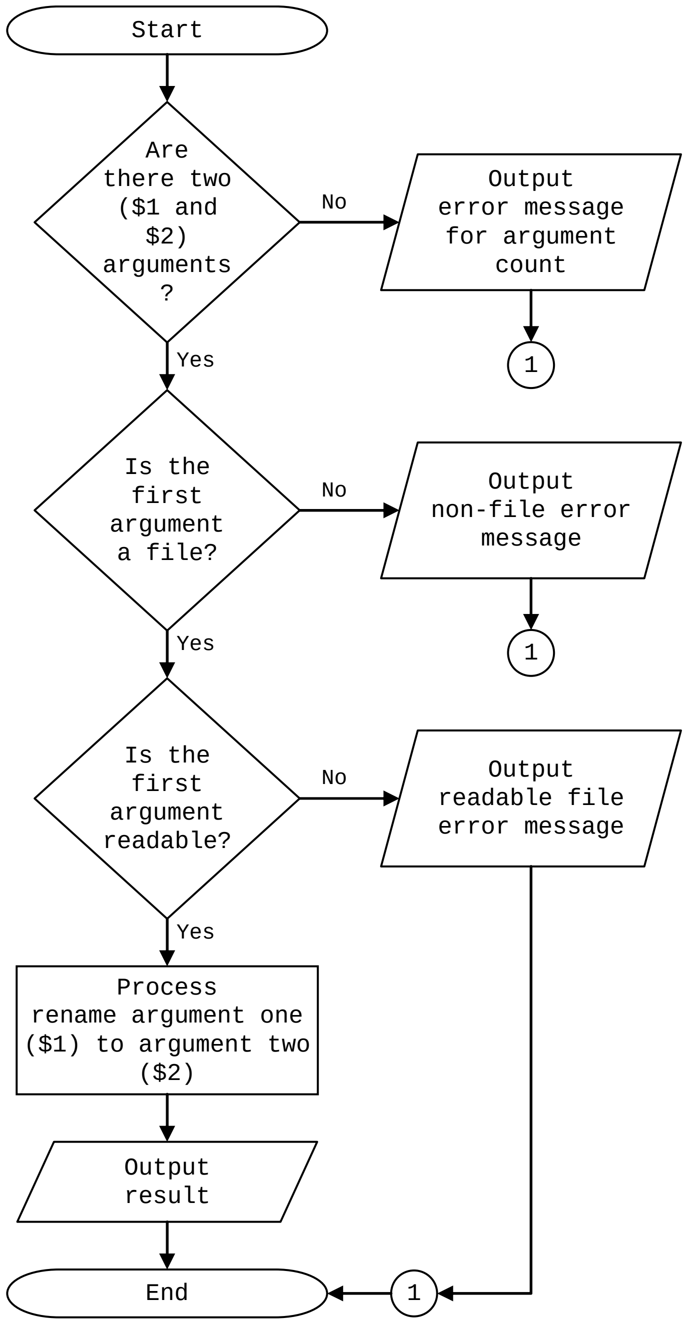
Flowchart 3-5 Semantics of the **suffix** program.


Start
Are there two ($1 and $2) arguments ?
No
Output error message for argument count
1
Yes
Is the first argument a file?
No
Output non-file error message
1
Yes
Is the first argument readable?
No
Output readable file error message
Yes
Process rename argument one ($1) to argument one and two ($1$2)
Output result
End
1

Flowchart 3-6 Semantics of the **suffix** program (revised).

➯ *continue on next page…*

rnfile.sh


Start
Are there two ($1 and $2) arguments ?
No
Output error message for argument count
1
Yes
Is the first argument a file?
No
Output non-file error message
1
Yes
Is the first argument readable?
No
Output readable file error message
Yes
Process rename argument one ($1) to argument two ($2)
Output result
End
1

Flowchart 3-7 Semantics of the **rnfile.sh** program.

4 SOURCE CODE

```
#!/bin/bash
echo $(($1*2))
```

Code 4-1 twice listing.

```
#!/bin/bash
#
#*********************************************************************
# Filename:      twice.sh
# Author:        Alex Shaw III
# Date created:  May 23, 2005
# Last modified: May 23, 2005
#
# Purpose: Doubles an integer
#
# Description:
# twice.sh takes one argument from the user and doubles it.
# An error statement occurs if no argument is present or if
# the argument is an integer.
#*********************************************************************

if [ $# == 1 ]; then                             #If one argument exists
   let "twice=$1*2"                              #   perform arithmetic

   if [ $? == 0 ]; then                          #If exit status is true
      clear                                      #   then the arithmetic
      echo "twice Program"                       #   was successful
      echo "-------------"
      echo "$1 * 2 = $twice"
      exit 0
   else                                          #If exit status is false
      clear                                      #   then argument was
      echo "The argument must be an integer." #   not an integer and
      exit 1                                     #   display error
   fi                                            #   message

   exit 0
else                                                     #Display error
   clear                                                 #   message
   echo "Only one argument is acceptable with twice." #   for the
   echo "Usage: twice argument"                          #   argument
   exit 1
fi
```

Code 4-2 twice.sh listing.

Code Listing

The main goal of this experiment is to resolve the tasks presented on the lab sheets and document the process.

Two versions of **twice**, **home**, and **suffix** enable you to complete a task and enhance it.

twice and twice.sh

The **twice** satisfies the task, while **twice.sh** enhances it.

Code 4-2 provides a complete listing of **twice.sh**, which includes the purpose and description.

```
#!/bin/bash
cut -f6 -d: /etc/passwd | grep $1
```

Code 4-3 home listing.

```
#!/bin/bash
#
#*********************************************************************
# Filename:      home.sh
# Author:        Alex Shaw III
# Date created:  May 23, 2005
# Last modified: May 23, 2005
#
# Purpose: Displays the home directory of a user
#
# Description:
# home.sh takes an argument from the user and displays their home
# directory.  If the user does not exist a custom error statement is
# display.  A custom error statement also displayed if no argument
# is entered.
#
# The cut command cuts fields 1 and 6 of the /etc/passwd file that
# contains the argument entered by the user.  If the length of the
# result is zero, then the user does not exist; otherwise, the home
# directory is displayed.
#*********************************************************************

if [ $# == 1 ]; then
   homedir=`cut -f1,6 -d: /etc/passwd | grep $1`

   # The cut command works on files and directories.
   # However, you can pipe a result to the command, as shown below:

   homedir=`echo $homedir | cut -f2 -d:`

   clear
   if [ -z $homedir ]; then
      echo "The home directory of $1 does not exist!"
      exit 0
   else
      echo "The home directory of $1 is:"
      echo $homedir
      exit 1
   fi

   exit 0
else
   clear
   echo "Only one argument is acceptable with home.sh"
   echo "Usage: home.sh argument"
   exit 1
fi
```

Code 4-4 home.sh listing.

home and home.sh

The **home** satisfies the task, while **home.sh** enhances it.

Code 4-4 provides a complete listing of **home.sh**, which includes the purpose and description.

```
#!/bin/bash
mv $1 $1$2
```

Code 4-5 suffix listing.

```
#!/bin/bash
#
#*********************************************************************
# Filename:      suffix.sh
# Author:        Alex Shaw III
# Date created:  May 24, 2005
# Last modified: May 24, 2005
#
# Purpose: Adds a suffix to a file
#
# Description:
# Accepts two arguments from the users and appends the second
# argument to the first argument. If the argument count is not two,
# then an error statement is displayed.
#
# Argument one must be a file to make the change.
#*********************************************************************

clear
if [ $# == 2 ]; then                             #If argument count is 2
   if [ -f $1 ]; then                            #   check file existence
      newfile=$1$2                               #   for argument one

      mv $1 $newfile

      if [ -f $newfile ]; then                   #If newly created file
         echo "suffix Program"                   #   exists after change,
         echo "--------------"                   #   display result
         echo "$1 is now $1$2"

         exit 0
      else                                       #New file creation was
         echo "Adding $2 to $1 was unsuccessful!" #   unsuccessful
         exit 1
      fi

      exit 0
   else                                          #Argument 1 is not a
      echo "$1 is not a file."                   #   file
      exit 1
   fi

   exit 0
else                                             #The user did not enter
   echo "You need two arguments to run suffix.sh" #   two arguments
   echo "Usage: suffix.sh argument1 argument2"
   exit 1
fi
```

Code 4-6 suffix.sh listing.

suffix and suffix.sh

The **suffix** satisfies the task, while **suffix.sh** enhances it.

Code 4-6 provides a complete listing of **suffix.sh**, which includes the purpose and description.

```
#!/bin/bash
mv $1 $2
```

Code 4-7 rnfile listing.

```
#!/bin/bash
#
#*********************************************************************
# Filename:      rnfile.sh
# Author:        Alex Shaw III
# Date created:  May 24, 2005
# Last modified: May 24, 2005
#
# Purpose: Renames a file
#
# Description:
# Accepts two arguments from the users and renames the second
# argument to the first argument. If the argument count is not two,
# then an error statement is displayed.
#
# Argument one must be a file to make the change.
#*********************************************************************

clear
if [ $# == 2 ]; then                              #If argument count is 2
   if [ -f $1 ]; then                             #   check file existence
      mv $1 $2                                    #   for argument one

      if [ -f $2 ]; then                          #If renamed file exists,
         echo "Rename Program"                    #   display result
         echo "--------------"
         echo "$1 is now $2"

         exit 0
      else                                          #New file creation was
         echo "Renaming $1 to $2 was unsuccessful!" #   unsuccessful
         exit 1
      fi

      exit 0
   else                                           #Argument 1 is not a
      echo "$1 is not a file."                    #   file
      exit 1
   fi

   exit 0
else                                              #The user did not enter
   echo "You need two arguments to run suffix.sh" #   two arguments
   echo "Usage: suffix.sh argument1 argument2"
   exit 1
fi
```

Code 4-8 rnfile.sh listing.

rnfile and rnfile.sh

The **rnfile** satisfies the task, while **rnfile.sh** enhances it.

Code 4-8 provides a complete listing of **rnfile.sh**, which includes the purpose and description.

5 ANALYSIS

Revisions

Once the coding started, problems occurred that forced a revision process to occur in some programs.

home.sh

If you use **echo ~brw05** at the bash prompt, it displays the home directory of user **brw05**. However, if you use **echo ~$1** in a script, it does not return the home directory of **$1**. It returns ~ and the name of argument 1 (**$1**). It was even difficult to assign the instruction to a variable.

In bash, **~"brw05"** delivers the same result that **~$1** does in a script. Therefore, it was realized that **$1** behaved as a string with ~.

Therefore, the **cut** command processed the */etc/passwd* file to cut the user name and home directory fields.

Below is the command line:
cut -f1,6 -d: ***/etc/passwd***

After that, **grep** did a pattern search on **$1** based on **cut** results. Those results were piped to another cut command, which separated the home directory from the user name.

Below is the script's instruction:
echo $homedir | cut -f2 -d:

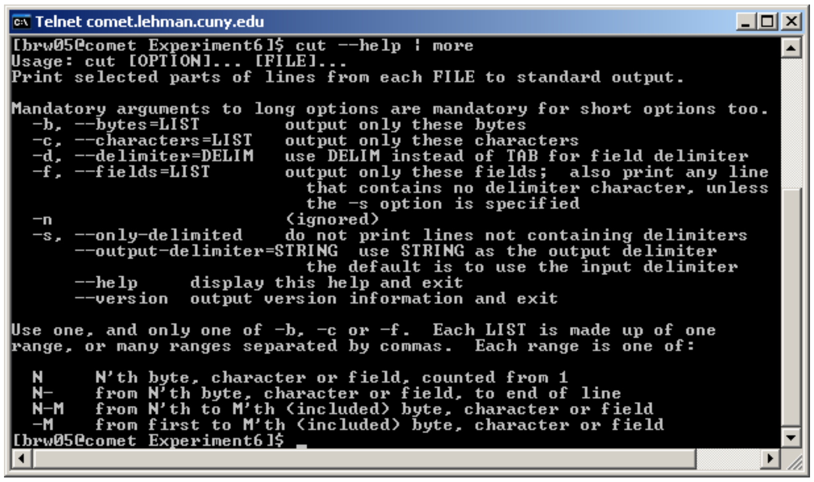
Originally, the script's instruction involved one line that cut the home directory based on $1.

Below is that instruction:
cut -f6 -d: ***/etc/passwd*** **| grep $1**

Field 6 pertains to the home directory; however, does the home directory need to include the user's name? Although the above instruction worked, it did not satisfy the question.

suffix.sh

The revised flowchart for suffix requires a test for file readability; however, the **mv** command renamed a non-readable file. Therefore, the readability test was not implemented in the code.



```
[brw05@comet Experiment6]$ cut --help | more
Usage: cut [OPTION]... [FILE]...
Print selected parts of lines from each FILE to standard output.

Mandatory arguments to long options are mandatory for short options too.
  -b, --bytes=LIST        output only these bytes
  -c, --characters=LIST   output only these characters
  -d, --delimiter=DELIM   use DELIM instead of TAB for field delimiter
  -f, --fields=LIST       output only these fields;  also print any line
                            that contains no delimiter character, unless
                            the -s option is specified
  -n                      (ignored)
  -s, --only-delimited    do not print lines not containing delimiters
      --output-delimiter=STRING  use STRING as the output delimiter
                            the default is to use the input delimiter
      --help     display this help and exit
      --version  output version information and exit

Use one, and only one of -b, -c or -f.  Each LIST is made up of one
range, or many ranges separated by commas.  Each range is one of:

  N     N'th byte, character or field, counted from 1
  N-    from N'th byte, character or field, to end of line
  N-M   from N'th to M'th (included) byte, character or field
  -M    from first to M'th (included) byte, character or field
[brw05@comet Experiment6]$ _
```

Output 5-1 Manual page for the **cut** command.

New Commands

Although a variety of scripting instructions were introduced in this experiment, the **cut** command was vital in accomplishing a vital task.

cut

Output 5-1 displays the man page for **cut**.

Below is the command's usage in the **home.sh** program:

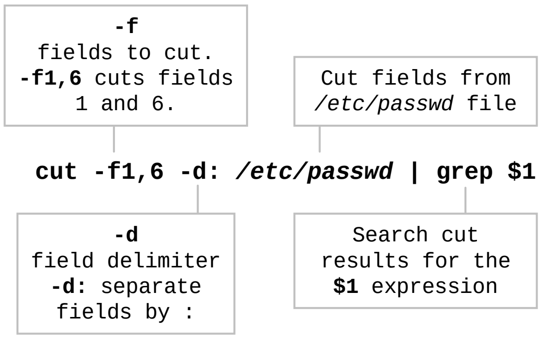

-f
fields to cut.
-f1,6 cuts fields 1 and 6.
Cut fields from /etc/passwd file
cut -f1,6 -d: /etc/passwd | grep $1
-d
field delimiter
-d: separate fields by :
Search cut results for the $1 expression

Diagram 5-1 Syntax of the **cut** command as it is used in the **home.sh** program.

6

CONCLUSION

Overall, no program is full proof, meaning that there can be an error somewhere in the application. Most likely, that error may occur during a daily operation of the program; or, a coworker may mention, upon reading your code, that you may want to check for this or improve that.

Although the programs in this experiment involved relatively simple processes, it still involved a great deal of thinking, tests, and error correction.

Most programming languages allow you to miss spaces and other constructs. It may even fix them for you. However, scripting sometimes require you to follow a strict format. It could be confusing in certain instances; but for the most part, they provide similar functionality.

One thing that seemed to go right for all programs is that they all produced errors based on an omitted argument or an out-of-range argument.

7
REFERENCES

References

No particular source was actually taken from any of the references below; however, they provided some useful examples to work from.

1. Sarwar, Syed Mansoor, Robert Koretsky, Syed Aqeel Sarwar. Linux: The Textbook. Boston: Addison Wesley Longman Inc., 2002.
2. Cooper, M. Advanced Bash-Scripting Guide: http://www.tldp.org/LDP/abs/html/