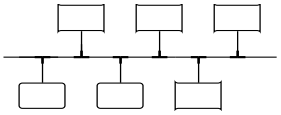


Lab Exercise
Operating Systems (Linux)

Treasure Hunt with “find”

Use the “find” command to search for the following files within the /usr directory of the Wary Puppy 5.3 file system.

1. Find a file called “newbieguide.html”, when was it last modified?
   (Hint: use the “-ls” switch to get more detail about each file or directory found.)

2. Apart from the file above which other file contains “guide” in the file name?

3. Find the file cpu.png, then find all the “.png” files in /usr/local that are newer than it.

4. Find all the index.html files, how many are there?

   a. Repeat the find command above but modify it so that the content of all the files is displayed.
      (Hint: use find, -exec and cat)

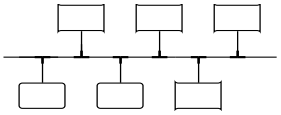
   b. Modify the command so that only the first 3 lines of each file are displayed.
      (Hint: use “head -3”)

   c. Repeat the find command above but modify it so that only lines in the found files containing the string “title” are displayed.

5. Find all files belonging to user “fido” note their name(s).

6. Find a file which has access permissions rwx r-x ---, note it’s name.

Answers are at the end of this document.



Word Search with GREP

grep filters its input based on string matching, this can be used to search for specific lines from a text file or to filter the output from another command.

grep uses regular expressions to specify the strings to be matched.

For experimenting with grep, it is useful to have a large file with many different string patterns to search. Luckily many computer file systems contain a large dictionary which is used for spell checking.

The Puppy Linux distribution has a dictionary with many thousand words, one per line, located at:-

/usr/share/myspell/dicts/en-US.dic

Navigate to the directory and use grep to search the word list for the following:-

1. Find a word which contains the text “linu”.

2. Find the name of a place which contains the text “cs” followed by an “n”.

3. What is the last word in the dictionary which contains the letters “c”, “s” and ”n” in order?

4. What is the first word in the dictionary which starts with “c” followed by “s” and then ”n”?
   (Hint: use “head” to see just the first line of output)

5. Find a word which starts with an “s” and contains every vowel character “aeiou” in order.

Answers are at the end of this document.

7.

Shell Script exercise

Background

The "shell" is the utility which interprets command lines submitted by the user. Thus the user interacts with the shell and the shell protects the core (kernel) of the OS from the user (or vice versa) hence the name.

Shells can also interpret and execute batches of commands from a file, these files can be written by the user to perform some set sequence of actions. These files are termed "shell scripts".

The user's default shell pathname can be found from a variable called "SHELL".

```
# echo $SHELL
```

Note that the "$" indicates a substitution, "SHELL" is a variable and whenever $SHELL is used it is replaced by the current value of that variable.

There are a number of different shells and shell scripts must be written to suit the syntax of the particular shell in use.(e.g. sh, csh, tcsh, bash or ksh).

The default shell in Linux is Bash[1] which is an open source alternative to the original (sh) shell referred to as the Bourne shell.

[1] Note "Bash" stands for Bourne Again SHell

Shell Primer

1) Simple Shell Script

a) Creating a working directory (Puppy Linux and ROX-Filer).

i) Starting the ROX-Filer.

Click the desktop "**file"** icon.
Right Click in the "**~**" window
Choose **New ► Directory**
Enter "/root/**scripts"** as the directory name.
Click on **[Create]**

Click on the icon for the new **scripts** directory, this will open the empty directory.

b) Making a blank script file.

Right click in the ~/scripts directory window.
Choose **New ► Script**
Enter "/root/scripts/**script1"** as the script name.
Remember that file names in Linux are case-sensitive.
(i.e. script1 ≠ Script1)

Click on **[Create]**

c) Opening the shell script

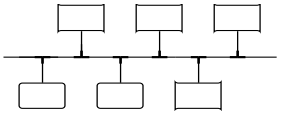
Right click on the **script1** icon
Choose **File 'script1' ► Open As Text**

This will open the script file in the editor.

Note that the first line has already been entered.

```
#!/bin/sh
```

This indicates that the Bourne shell (/bin/sh) will always be used to run this particular script. (i.e. it is not a "csh" nor "ksh" script)



d) Writing the script commands.
Enter the lines below into your shell script file exactly as shown.
Note that, if your system has not yet been configured for UK keyboards (the default is US), you can use the **£** key for **#** and **¬** for a **~.**

```
#!/bin/sh
#My first script
echo Today is
date
echo
echo I am
echo $USER
echo Disk Usage of my-documents is
du –k –s ~/my-documents
echo Bye Bye
```

Notes:
A line beginning with a **"#"** is a comment line.
The "**echo**" command echoes text as screen output, it is useful in job control or scripting.
"**du**" shows the disk usage of a particular directory tree, the **-k** and **-s** switch options give the results in kilobytes as an overall summary

e) Saving the script
Click on **Save**

f) Running the script

The script can be run just by entering its pathname at the prompt.

Open a console window
Run
 # scripts/script1

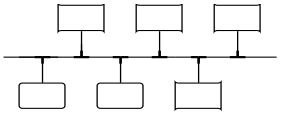
The script should give some screen output, correct errors if any appear.

g) Debugging scripts

If you would like to see the actions and variable substitutions performed by your script as it runs, you can run the script in debug mode using the "-x" switch.

 # /bin/sh –x scripts/script1

or alternatively, the first line of the script can be changed to
`#!/bin/sh -x`



2) Shell Script Control Structures

Like other programming environments, the shell interpreter supports control structures (iteration/decisions).
However the syntax of this scripted language is very critical and error messages are not very helpful. Therefore you need to be very careful and precise with your typing; spaces and capitalisation are important.

Try this example of shell script iteration.

a) Create a blank script file in the **~/scripts** directory as before.
Call this script file "loop1".

b) Enter the following script exactly as shown below.

```
#!/bin/sh
for x in 9 10 11 12
  do
    cal $x 2013 > month$x
done
```

Any commands between "**do**" and "**done**" are repeated once for each of the values listed after the "**in**" keyword.

"**x**" is a variable and whenever **$x** is used it is replaced by the current value of this variable.

In the first iteration "**x**" has the value "**9**" then "**10**" and so on until the last iteration when it finishes with x="**12**".

This script should create 4 monthly calendars in separate files.

Can you tell what the names of these files will be?

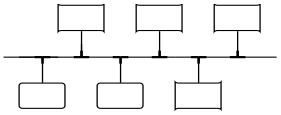
c) Run the script.

```
# scripts/loop1
```

Use the file manager to view the files which have been created.

d) Now modify the script so it will delete all the files that you have just created but leave the calendar file for December.
Your modified script will not use "cal" nor need ">" redirection but will simply use the remove "rm" within the loop to delete certain files.

e) Run the modified script, use the file manager to check that the all the calendar files except December have been removed.



3) Designing your own script.

a) Create a new script called “loop2” which will use a for-do loop to create 4 copies of the remaining (December) calendar file called
plan_tb
plan_gb
plan_as
plan_nc

b) Run the script and check that it created the desired copies with the correct file names.

4) Getting user input
Another way to use variables is to prompt the user to enter a value for a variable.

a) Edit your “script1” script and prepend the following lines.

```
echo Please enter a new value for x
read x
echo You typed $x
```

b) Save and run the script and try entering some values.

Are numbers, strings or sentences accepted?

5) Command Line ”parameter” input

   a) An alternative way of inputting values to a script is by way of command line parameters.

      Example:
      ```
      # myscript param1 param2
      ```

      In the above case the values following the “myscript” command can be used as variable values inside the script itself.
      “$1” is substituted with the first parameter value supplied, “$2” the second and so on.

      Edit your script1 script by adding the following two lines

      ```
      echo Param1 is $1
      echo Param2 is $2
      ```

6) Now run the script giving it two parameters on the command line.

   # **scripts/script1 dog cat**

   This should provide output which includes the lines
   ```
   Param1 is dog
   Param2 is cat
   ```

7) Making Decisions (Branching)

   Scripts can also test conditions and perform differing sets of instructions depending on the result of the test. These tests can compare variable values, check the existence of a file or directory and many other conditions.

   a) if-then-else

      The if-then-else construct allows for a choice of action depending on the result of a test.

      Syntax:
      ```
      if TEST-COMMAND ; then
            echo The test result was true
      else
            echo The test result was false
      fi
      ```

      Notes:
      As the keyword “then” is on the same line as the “if”, it must be preceded by a semi-colon.
      The “else” is optional if there were no commands to execute for a false result.
      The keyword “fi” is used to end the “if” construct.

b) Testing Conditions

The utility which provides for testing various conditions is a utility called "**test**" (/usr/bin/test), it has a very large number of options for choosing comparisons.
For example **"-d"** tests if a directory exists.

To check the result of a test you can put **"&&"** after the test, the next command will only be run if the result of the test was true.

Try this:

```
# test -d my-documents && echo true it exists

# test -d my-docs && echo true it exists
```

This test condition could be built into an if-then-else construct within a script.

Example:

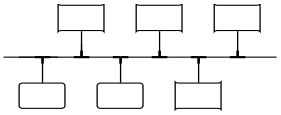
```
#!/bin/sh

if test –d my-documents ; then
      echo There IS a directory called my-documents
else
      echo There is NO directory called my-documents
fi
```

c) Using the above example as a starting point, create a script which will prompt the user to enter a directory name and then report if that directory actually exists.

d) Develop your script so that, if the user supplied name is not a directory, the script then checks to see if it is a file.

(Hint: you can use nested if-then-else constructs)



Common conditions for the **test** utility

Condition	**True means that:-**
-d dname	dname exists and is a directory.
-f fname	fname exists and is a file
-r file	the file exists and is readable.
-w file	the file exists and is writable.
-x file	the file exists and is executable.
file1 **–nt** file2	file1 is newer, by modification date, than file2.
file1 **-ot** file2	file1 is older than file2.
-z string	the length of the string is 0.
-n string	the length of the string is non-zero.
string1 = string2	the strings are equal
string1 **!=** string2	the strings are not equal
int1 **–eq** int2	the integers are equal
int1 **-gt** int2	int1 is greater than int2
int1 **-lt** int2	int1 is less than int2
int1 **-ge** int2	int1 is greater than or equal to int2
int1 **-le** int2	int1 is less than or equal to int2
int1 **-ne** int2	the integers are not equal

Answers to the “find” exercise.

1. $ find /usr –name newbieguide.html –ls
   Jun 7 2003

2. $ find /usr –name *guide*
   /usr/share/inkscape/guide_dialog.png

3. $ find /usr –name cpu.png
   /usr/share/icons/devices/cpu.png
   $ find /usr –name *.png –newer /usr/share/icons/devices/cpu.png
   /usr/local/pmusic/pmusic20.png
   /usr/local/pburn/themes/Mini/gtk/pburn20.png
   /usr/local/pburn/themes/Mini/splash.png
   OR just
   find /usr ` find /usr –name cpu.png ` –name *.png –newer

4. There are 10 index.html files in /usr

   a. $ find /usr –name index.html –exec cat {} \;

   b. $ find /usr –name index.html –exec head -3 {} \;

   c. $ find /usr –name index.html –exec grep title {} \;

5. $ find /usr –user fido
   /usr/sbin/askpass
   /usr/sbin/loginmanager

6. $ find /usr –type f –perm 750
   /usr/local/sys-freedos/sys-freedos.pl

Answers to the grep exercise

1. botulinus

2. Tucson

3. yachtswomen

4. cabstand

5. sacrilegious