UNIX Assignment

These are not expected to be answered in writing, but as plug-ins for

the thorough understanding of the subject.

Have you ever wondered how a system such as UNIX did ever come into

existence?

What is that operating system in the computer systems you have now?

Why Unix has become a popular operating system?

How come Unix is so portable?

Other features of Unix?

Kernel?

How does Unix view the devices?

Types of UNIX Files

Working with files and directories

Default File and Directory permissions – Your umask

Unix Commands?

ls, mkdir, cd, chmod

bc, dc—calculator programs

date and cal—to display time and date

finger, who and id—provide user information

man and help—explains commands

su—"switch user"

du and df—display information about disk contents

Output/Input redirection

What is a process?

In which file is your password stored?

What is the significance of passwd file?

Metacharacters

Regular Expressions

Stream Editors

How do we search for files in a directory structure?

What is a shell?

Bourne Shell

Korn Shell

C Shell

Which is the Shell in your system?

How and why do you program your shell?

What are the popular editors with Unix?

How does Linux differ from Unix?

UNIX Assignment

pwd

When executed this gives the present working directory. Tells where a user is

presently situated. The response obtained is in the form of a path. The path

starts from the root directory.

1. What is meant by Absolute path?

2. What is meant by Relative path?

3. What are the paths that are included by default?

mkdir

To create a directory.

cd

Change Directory

4. What does cd do when used without arguments?

rmdir

To delete or remove a directory.

5. When will rmdir dir1 where dir1 is a directory fail to work?

6. What do the files . and .. mean?

7. Create a tree structure named ‘training’ in which there are 3 subdirectories –

‘level 1’,’ level2’ and ‘cep’. Each one is again further divided into 3. The ‘level

1’ is divided into ‘sdp’, ‘re’ and ‘se’. From the subdirectory ‘se’ how can one

reach the home directory in one step and also how to navigate to the

subdirectory ‘sdp’ in one step? Give the commands, which do the above

actions?

8. Give the commands to remove the directory training.

ls

Lists the contents of a directory.

9. How to give a long list of the files in the directory?

10. What are the different fields in the long listing of the directory?

11. What is the inode number of a file?

UNIX Assignment

12. Give the command to list the files in the reverse alphabetical order.

13. How to get the listing of the directories recursively?

14. What does ‘ls –a’ do?

15. What does ‘ls –h’ do?

touch

Creates a zero byte file. Also used to modify the date and time of creating a file.

16. Create some files with the names ‘quiz’, feedback’, ‘test’ and ‘exam’.

cp

Copies one file onto another

ln

Gives different names to the same file.

17. What is the difference between cp and ln? Explain.

mv

Move the contents of a file onto another. (This is like rename in DOS).

18. What is the difference between cp and mv.

19. How will you copy a directory structure dir1 to dir2 ? (with all the

subdirectories)

rm

Deletes a file. Beware.

20. How to interactively remove a file?

21. How to remove a directory using ‘rm’ command?

chmod

Changing file permissions.

22. What does each digit in chmod signify?

23. Explain about each of the options along with chmod, namely, + - = etc. Try

out all these and compare.

UNIX Assignment

24. What are the default permissions when a file or directory is created?

who

Gives the login details

25. How can you find out the number of users logged in?

26. How can you find out if you have the permission to send a message?

man

Online help for the commands executed.

27. How to get only the options and arguments of a command excluding the

descriptive help?

du

Summarises the disk usage of each file recursively for directories.

28. Find the space occupied ( in Bytes) by the /home directory including all its

subdirectories.

df

Report file system disk space usage.

clear

Clears the screen.

cat

Used to create, display and append to files.

29. How can cat be used to append to a file?

30. How do you get the numbered lines in the output?

more ( less is another pager available on Linux)

Paging output. The screen will handle one page at a time.

31. How do you count the number of logical lines?

32. How do you get the output screen by screen?

UNIX Assignment

type

The location of the program is obtained.

cmp

To check whether two files are identical or not. The output of ‘cmp’, if the files

differ is the first position at which difference occurs.

33. When does ‘cmp’ command not give any response even when the files are

different?

diff

Displays the differences in two files/ directories.

34. What option can be used to get the answer to whether the files are different or

not?

35. When are the changes in cases ignored? ie, with what option?

comm

Finds out what is common in two sorted files.

36. How will you display only the lines common to two files?

umask

When file is created it is given some default permissions. The file permission

mask determines this. The default file permission mask can be found by typing

‘umask’ without any arguments at the command prompt.

37. What is your default file permission mask?

38. What is the mask value you will use if you want the owner of the to have

read/write permissions and everybody else to have nil permissions?

39. Can you create a file if the ‘umask’ is set to 777? Explain your answer

date

date command can be used to print the current date and time. Find out its details

of usage from the man pages and do the following:

40. What is the command for printing the current time in 24-hour format?

41. What is the command for printing the year, month, and date with a horizontal

tab between the fields?

UNIX Assignment

cal

cal command prints the calendar.

42. What is the command for printing the calendar of the current month?

43. What is special about September 1752?

bc

Basic Calculator

44. Do all the basic arithmetic operations using bc.

wc

wc counts characters, words, and lines in a file.

45. What is the command for printing the number of lines in the ‘passwd’ file?

sort

sort sorts information alphabetically.

46. What are the commands used to sort some information in the reverse order

and in the numerical order?

47. What does the command sort –f do?

uniq

uniq is a filter which gives the unrepeated elements in a file.

48. What do the commands uniq –c, uniq –d and uniq –u do?

49. Can uniq be used on an unsorted file?

find

Search for files in a directory hierarchy.

50. Search for a file “india.txt” from the /home directory. It can exist in any of the

subsequent directories.

wild cards

The shell uses certain characters for pattern matching. For example ‘\*’ is used to

match any character including none and ‘?’ matches a single character. Find out

more about these characters and answer the following.

UNIX Assignment

51. First create the following files: chapa, chapb, chapc, chapd, chape, chapA,

chapB, chapC, chapD, chapE, chap01, chap02, chap03, chap04, chap05,

chap11, chap12, chap13, chap14, and chap15.

52. What is the command for listing all files ending in small letters?

53. What is the command for listing all files ending in capitals?

54. What is the command for listing all files whose last but one character is 0?

55. What is the command for listing all files which end in small letters but not ‘a’

and ‘c’?

redirection

Input and output can be redirected from stdin and stdout. ‘<’ symbol is used for

input redirection and ‘>’ symbol is used for output redirection.

56. What is the ‘>>’ symbol used for?

57. What is the command for sending the output to stdout and error messages to

the null device (or bit bucket)?

58. What is the command for sending both the output and the error messages to

the same file?

pipes

‘|’ is called the pipe symbol. With this the output of a command can be connected

to the input of another command. Any number of commands can be connected

this way. To save the output of any command in the middle of a pipe to a file tee

is used.

59. What is the command to print the number of users logged on to a system?

60. How do you connect two commands using tee?

61. Connect the two commands cat part1 part2 > total and lpr total using tee.

62. How to append to a file using tee?

UNIX Assignment

vi

Type vilearn at the command prompt and go through all the lessons. You are

also expected to go through the chapters on editors in the book given to you.

Now answer the following:

63. What are the three modes of vi?

64. What is the command to introduce a new line above the first line in a file?

65. What is the command to replace an entire line?

66. What is the command to insert 50 dashes (‘-‘) in a line?

67. What is the expression you will use to match blank lines?

68. Assume you have deleted 5 lines from a file, 3 lines from another place, and 6

lines from somewhere else. What to do to paste the lines deleted second time

at the end of the file?

69. How do you join two lines?

70. How many named buffers are there in vi?

71. How do you append lines to a named buffer?

72. Where do you store the customization commands for vi?

UNIX Assignment

REGULAR EXPRESSIONS

‘\*’ character is used both as a shell wild card character and in regular

expressions. What is the difference in the interpretations?

73. How is the character ‘^’ used in a regular expression?

74. How is the character ‘$’ used in a regular expression?

grep, sed

grep is used to match patterns in a file.

75. ‘-v’ is one of the options that can used with grep. What does it do and what is

its usefulness?

76. In an organisation one wants to know how many programmers are there. The

employee data is stored in a file called ‘personnel’ with one record per

employee. Every record has field for designation. How can grep be used for

this purpose?

77. When is it necessary to put the string to be matched in quotes in grep?

sed is called stream editor

78. In the organisation mentioned above how can sed be used to print only the

records of all employees who are programmers.

79. In the organisation mentioned above how can sed be used to change the

designation ‘programmer’ to ‘software professional’ every where in the

‘personnel’ file

PROCESSES

80. How do you run a job in the background? How do you ensure that the job

does not get terminated when you logout?

81. Find out about the sleep command and start five jobs in the background, each

one sleeping for 10 minutes.

82. What is the command for printing a list of background jobs?

83. What is the command for bringing the 3rd background job to the foreground?

And how will you put it back in the background?

84. How do you kill the 3rd background job?

UNIX Assignment

85. What will happen if you kill your login shell?

ps

This command makes a process status report.

86. How do you get the status of all the processes running on the system? i.e.

using what option?

87. How to get a long listing of all the process status details?

UNIX Assignment

shell

set

This command makes the shell printout the variables it knows and also to assign

values to the positional parameters.

88. What does the command set seven days in a week do?

export

Allows copies of a variable and its value presented to different processes.

89. Give an example of making a variable available to all the processes using

export. Is it truly becoming available to all?

90. What command will change your prompt to ‘PATNI’?

91. What are contents of the shell variables HOME and PATH?

92. What are the contents of the shell variables PWD and OLDPWD and how do

you use OLDPWD?

93. Is it possible to type more than one command at the shell prompt and how do

you do it?

94. It is possible to concatenate commands with ‘||’ and ‘&&’. What is their

purpose?

95. If you type the following at the shell prompt what happens and why?

cricketer=sachin tendulkar

echo can be used to print the contents of a shell variable or print a string . Find

out more about it from the man page.

96. What is the option to be used if do not want a new line to be output after the

echo?

97. Explain the differences between single quotes (‘), double quotes (“), and back

quotes (`)

98. Type the following at the shell prompt. What happens and how do you modify

it to get the desired output?

echo ‘the number of files in /bin directory are `ls –l /bin|wc –l` ‘

99. Enter the above statement in to a file called ‘script.sh’. What do you have to

do execute it?

UNIX Assignment

100. How to execute the shell script in the same shell as it is operating from

and not in its child shell?

101. Write a script which accepts the word “PATNI” as an answer regardless of

whether upper-case or lower-case letters are used anywhere in the word?

102. Write a script that asks for the capital of INDIA and repeats the question

until the user gets it right.

103. Write a script to print the number of command line arguments?

104. Write a script to print the command line arguments?

105. Modify the above script to test if the number of command line arguments

is zero and print a message if so?

106. Write a script which will take up to nine arguments, tests if each argument

is a readable file and prints a message?

107. How will you modify the above script if the number of arguments is greater

than nine?

Shell Programming Examples

1. Write a shell program, which when executed will ask you to enter your

name, once you give the input it comes up with the out "Hello < your name

>" on the screen.

2. Write a shell script, which will take two strings as input, compares them

and comes up with the output "Both are same" or "Both are different"

based on the inputs. It should also check for null string inputs and should

give corresponding error message.

3. Write a shell program, to print even & odd numbers from 1-50. The output

should look as follows:

ODD EVEN

===== ======

1 2

3 4

... ...

49 50

4. Write a shell program, to print prime numbers from 1-50. Don't worry

about the efficiency of the algorithm.

UNIX Assignment

5. Write a shell program, which when executed will display following

messages base on the current system time:

For System Time Message To Be Displayed

============= ===================

06:00 - 11:59 a m Good Morning

12:00 Good Noon

12:01 - 03:00 p m Good Afternoon

03:01 - 10:00 p m Good Evening

10:01 - 05:59 a m Time To take rest , Good Night.

NOTE: You can add this to your ".profile" to get the message when

ever you login.

6. Without using UNIX line count command (e.g. wc ), write a shell script

which will print the no. of lines of a non-binary file.

7. Write a script to print the amount disk space used, in MBs (mega bytes) of

a current directory including all its sub-directory hierarchy. (Hint: Use "du"

command, du outputs no. of 512 byte blocks occupied by current

directory ).

8. Write a shell program, to send a message of text to all users who have

logged in as your user-id, without using UNIX communication commands.

(e.g.: if you have logged in as "user1", this script will send message all the

other sessions who are currently logged in as "user1". In fact this is a

restriction also that without using UNIX communication commands you

can't send message to any other users who have logged as different userid,

why?

Hint: Strength of UNIX security feature. )

UNIX Assignment

APPENDIX A

How to write a shell script

Introduction

A shell is a command line interpretor. It takes commands and executes them. As such, it

implements a programming language. The Bourne shell is used to create shell scripts -- ie.

programs that are interpreted/executed by the shell. You can write shell scripts with the C-shell;

however, this is not covered here.

Creating a Script

Suppose you often type the command

find . -name file -print

and you'd rather type a simple command, say

sfind file

Create a shell script

% cd ~/bin

% emacs sfind

% page sfind

find . -name $1 -print

% chmod a+x sfind

% rehash

% cd /usr/local/bin

% sfind tcsh

./shells/tcsh

Observations

This quick example is far from adequate but some observations:

1. Shell scripts are simple text files created with an editor.

2. Shell scripts are marked as executeable

%chmod a+x sfind

3. Should be located in your search path and ~/bin should be in your search path.

4. You likely need to rehash if you're a Csh (tcsh) user (but not again when you login).

5. Arguments are passed from the command line and referenced. For example, as $1.

#!/bin/sh

All Bourne Shell scripts should begin with the sequence

#!/bin/sh

From the man page for exec(2):

"On the first line of an interpreter script, following the "#!", is the name of a program which

should be used to interpret the contents of the file. For instance, if the first line contains "#!

/bin/sh", then the con- tents of the file are executed as a shell script."

You can get away without this, but you shouldn't. All good scripts state the interpretor explicitly.

Long ago there was just one (the Bourne Shell) but these days there are many interpretors -- Csh,

Ksh, Bash, and others.

Comments

Comments are any text beginning with the pound (#) sign. A comment can start anywhere on a

line and continue until the end of the line.

UNIX Assignment

Search Path

All shell scripts should include a search path specifica- tion:

PATH=/usr/ucb:/usr/bin:/bin; export PATH

A PATH specification is recommended -- often times a script will fail for some people because

they have a different or incomplete search path.

The Bourne Shell does not export environment variables to children unless explicitly instructed to

do so by using the export command.

Argument Checking

A good shell script should verify that the arguments sup- plied (if any) are correct.

if [ $# -ne 3 ]; then

echo 1>&2 Usage: $0 19 Oct 91

exit 127

fi

This script requires three arguments and gripes accordingly.

Exit status

All Unix utilities should return an exit status.

# is the year out of range for me?

if [ $year -lt 1901 -o $year -gt 2099 ]; then

echo 1>&2 Year \"$year\" out of range

exit 127

fi

etc...

# All done, exit ok

exit 0

A non-zero exit status indicates an error condition of some sort while a zero exit status indicates

things worked as expected.

On BSD systems there's been an attempt to categorize some of the more common exit status

codes. See /usr/include/sysexits.h.

Using exit status

Exit codes are important for those who use your code. Many constructs test on the exit status of a

command.

The conditional construct is:

if command; then

command

fi

For example,

if tty -s; then

echo Enter text end with \^D

fi

Your code should be written with the expectation that others will use it. Making sure you return a

meaningful exit status will help.

UNIX Assignment

Stdin, Stdout, Stderr

Standard input, output, and error are file descriptors 0, 1, and 2. Each has a particular role and

should be used accordingly:

# is the year out of range for me?

if [ $year -lt 1901 -o $year -gt 2099 ]; then

echo 1>&2 Year \"$year\" out of my range

exit 127

fi

etc...

# ok, you have the number of days since Jan 1, ...

case `expr $days % 7` in

0)

echo Mon;;

1)

echo Tue;;

etc...

Error messages should appear on stderr not on stdout! Output should appear on stdout. As for

input/output dialogue:

# give the fellow a chance to quit

if tty -s ; then

echo This will remove all files in $\* since ...

echo $n Ok to procede? $c; read ans

case "$ans" in

n\*|N\*)

echo File purge abandoned;

exit 0 ;;

esac

RM="rm -rfi"

else

RM="rm -rf"

fi

Note: this code behaves differently if there's a user to communicate with (ie. if the standard input

is a tty rather than a pipe, or file, or etc. See tty(1)).

Language Constructs

For loop iteration

Substitute values for variable and perform task:

for variable in word ...

do

command

done

For example:

for i in `cat $LOGS`

do

mv $i $i.$TODAY

cp /dev/null $i

chmod 664 $i

UNIX Assignment

done

Alternatively you may see:

for variable in word ...; do command; done

Case

Switch to statements depending on pattern match

case word in

[ pattern [ | pattern ... ] )

command ;; ] ...

esac

For example:

case "$year" in

[0-9][0-9])

year=19${year}

years=`expr $year - 1901`

;;

[0-9][0-9][0-9][0-9])

years=`expr $year - 1901`

;;

\*)

echo 1>&2 Year \"$year\" out of range ...

exit 127

;;

esac

Conditional Execution

Test exit status of command and branch

if command

then

command

[ else

command ]

fi

For example:

if [ $# -ne 3 ]; then

echo 1>&2 Usage: $0 19 Oct 91

exit 127

fi

Alternatively you may see:

if command; then command; [ else command; ] fi

While/Until Iteration

Repeat task while command returns good exit status.

{while | until} command

do

command

done

For example:

# for each argument mentioned, purge that directory

while [ $# -ge 1 ]; do

\_purge $1

shift

done

Alternatively you may see:

while command; do command; done

UNIX Assignment

Variables

Variables are sequences of letters, digits, or underscores beginning with a letter or

underscore. To get the contents of a variable you must prepend the name with a $.

Numeric variables (eg. like $1, etc.) are positional vari- ables for argument

communication.

Variable Assignment

Assign a value to a variable by variable=value. For example:

PATH=/usr/ucb:/usr/bin:/bin; export PATH

or

TODAY=`(set \`date\`; echo $1)`

Exporting Variables

Variables are not exported to children unless explicitly marked.

# We MUST have a DISPLAY environment variable

if [ "$DISPLAY" = "" ]; then

if tty -s ; then

echo "DISPLAY (`hostname`:0.0)? \c";

read DISPLAY

fi

if [ "$DISPLAY" = "" ]; then

DISPLAY=`hostname`:0.0

fi

export DISPLAY

fi

Likewise, for variables like the PRINTER which you want hon- ored by lpr(1).

From a user's .profile:

PRINTER=PostScript; export PRINTER

Note: that the Cshell exports all environment variables.

Referencing Variables

Use $variable (or, if necessary, ${variable}) to reference the value.

# Most user's have a /bin of their own

if [ "$USER" != "root" ]; then

PATH=$HOME/bin:$PATH

else

PATH=/etc:/usr/etc:$PATH

fi

The braces are required for concatenation constructs.

$p\_01

The value of the variable "p\_01".

${p}\_01

The value of the variable "p" with "\_01" pasted onto the end.

Conditional Reference

${variable-word}

If the variable has been set, use it's value, else use word.

POSTSCRIPT=${POSTSCRIPT-PostScript};

export POSTSCRIPT

${variable:-word}

If the variable has been set and is not null, use it's value, else use word.

These are useful constructions for honoring the user envi- ronment. Ie. the user of

the script can override variable assignments. Cf. programs like lpr(1) honor the

PRINTER environment variable, you can do the same trick with your shell

scripts.

UNIX Assignment

${variable:?word}

If variable is set use it's value, else print out word and exit. Useful for bailing out.

Arguments

Command line arguments to shell scripts are positional vari- ables:

$0, $1, ...

The command and arguments. With $0 the command and the rest the arguments.

$#

The number of arguments.

$\*, $@

All the arguments as a blank separated string. Watch out for "$\*" vs. "$@".

And, some commands:

shift

Shift the postional variables down one and decrement number of arguments.

set arg arg ...

Set the positional variables to the argument list.

Command line parsing uses shift:

# parse argument list

while [ $# -ge 1 ]; do

case $1 in

process arguments...

esac

shift

done

A use of the set command:

# figure out what day it is

TODAY=`(set \`date\`; echo $1)`

cd $SPOOL

for i in `cat $LOGS`

do

mv $i $i.$TODAY

cp /dev/null $i

chmod 664 $i

done

Special Variables

$$

Current process id. This is very useful for constructing temporary files.

tmp=/tmp/cal0$$

trap "rm -f $tmp /tmp/cal1$$ /tmp/cal2$$"

trap exit 1 2 13 15

/usr/lib/calprog >$tmp

$?

The exit status of the last command.

$command

# Run target file if no errors and ...

if [ $? -eq 0 ]

then

etc...

fi

UNIX Assignment

Quotes/Special Characters

Special characters to terminate words:

; & ( ) | ^ < > new-line space tab

These are for command sequences, background jobs, etc. To quote any of these use a

backslash (\) or bracket with quote marks ("" or '').

Single Quotes

Within single quotes all characters are quoted -- including the backslash. The result is one

word.

grep :${gid}: /etc/group | awk -F: '{print $1}'

Double Quotes

Within double quotes you have variable subsitution (ie. the dollar sign is interpreted) but

no file name generation (ie. \* and ? are quoted). The result is one word.

if [ ! "${parent}" ]; then

parent=${people}/${group}/${user}

fi

Back Quotes

Back quotes mean run the command and substitute the output.

if [ "`echo -n`" = "-n" ]; then

n=""

c="\c"

else

n="-n"

c=""

fi

and

TODAY=`(set \`date\`; echo $1)`

Functions

Functions are a powerful feature that aren't used often enough. Syntax is

name ()

{

commands

}

For example:

# Purge a directory

\_purge()

{

# there had better be a directory

if [ ! -d $1 ]; then

echo $1: No such directory 1>&2

return

fi

etc...

}

Within a function the positional parmeters $0, $1, etc. are the arguments to the function

(not the arguments to the script).

Within a function use return instead of exit.

Functions are good for encapsulations. You can pipe, redi- rect input, etc. to functions.

For example:

# deal with a file, add people one at a time

UNIX Assignment

do\_file()

{

while parse\_one

etc...

}

etc...

# take standard input (or a specified file) and do it.

if [ "$1" != "" ]; then

cat $1 | do\_file

else

do\_file

fi

Sourcing commands

You can execute shell scripts from within shell scripts. A couple of choices:

sh command

This runs the shell script as a separate shell. For example, on Sun machines in /etc/rc:

sh /etc/rc.local

. command

This runs the shell script from within the current shell script. For example:

# Read in configuration information

. /etc/hostconfig

What are the virtues of each? What's the difference? The second form is useful for

configuration files where environment variable are set for the script. For example:

for HOST in $HOSTS; do

# is there a config file for this host?

if [ -r ${BACKUPHOME}/${HOST} ]; then

. ${BACKUPHOME}/${HOST}

fi

etc...

Using configuration files in this manner makes it possible to write scripts that are

automatically tailored for differ- ent situations.

Some Tricks

Test

The most powerful command is test(1).

if test expression; then

etc...

and (note the matching bracket argument)

if [ expression ]; then

etc...

On System V machines this is a builtin (check out the com- mand /bin/test).

On BSD systems (like the Suns) compare the command /usr/bin/test with /usr/bin/[.

Useful expressions are:

test { -w, -r, -x, -s, ... } filename

is file writeable, readable, executeable, empty, etc?

UNIX Assignment

test n1 { -eq, -ne, -gt, ... } n2

are numbers equal, not equal, greater than, etc.?

test s1 { =, != } s2

Are strings the same or different?

test cond1 { -o, -a } cond2

Binary or; binary and; use ! for unary negation.

For example

if [ $year -lt 1901 -o $year -gt 2099 ]; then

echo 1>&2 Year \"$year\" out of range

exit 127

fi

Learn this command inside out! It does a lot for you.

String matching

The test command provides limited string matching tests. A more powerful trick is to

match strings with the case switch.

# parse argument list

while [ $# -ge 1 ]; do

case $1 in

-c\*) rate=`echo $1 | cut -c3-`;;

-c) shift; rate=$1 ;;

-p\*) prefix=`echo $1 | cut -c3-`;;

-p) shift; prefix=$1 ;;

-\*) echo $Usage; exit 1 ;;

\*) disks=$\*; break ;;

esac

shift

done

Of course getopt would work much better.

SysV vs BSD echo

On BSD systems to get a prompt you'd say:

echo -n Ok to procede?; read ans

On SysV systems you'd say:

echo Ok to procede? \c; read ans

In an effort to produce portable code we've been using:

# figure out what kind of echo to use

if [ "`echo -n`" = "-n" ]; then

n=""; c="\c"

else

n="-n"; c=""

fi

etc...

echo $n Ok to procede? $c; read ans

Is there a person?

The Unix tradition is that programs should execute as qui- etly as possible. Especially for

pipelines, cron jobs, etc.

User prompts aren't required if there's no user.

# If there's a person out there, prod him a bit.

if tty -s; then

UNIX Assignment

echo Enter text end with \^D

fi

The tradition also extends to output.

# If the output is to a terminal, be verbose

if tty -s <&1; then

verbose=true

else

verbose=false

fi

Beware: just because stdin is a tty that doesn't mean that stdout is too. User prompts

should be directed to the user terminal.

# If there's a person out there, prod him a bit.

if tty -s; then

echo Enter text end with \^D >&0

fi

Have you ever had a program stop waiting for keyboard input when the output is directed

elsewhere?

Creating Input

We're familiar with redirecting input. For example:

# take standard input (or a specified file) and do it.

if [ "$1" != "" ]; then

cat $1 | do\_file

else

do\_file

fi

alternatively, redirection from a file:

# take standard input (or a specified file) and do it.

if [ "$1" != "" ]; then

do\_file < $1

else

do\_file

fi

You can also construct files on the fly.

rmail bsmtp <

rcpt to:

data

from: <$1@newshost.uwo.ca>

to:

Subject: Signon $2

subscribe $2 Usenet Feeder at UWO

.

quit

EOF

Note: that variables are expanded in the input.

String Manipulations

One of the more common things you'll need to do is parse strings. Some tricks

TIME=`date | cut -c12-19`

TIME=`date | sed 's/.\* .\* .\* \(.\*\) .\* .\*/\1/'`

UNIX Assignment

TIME=`date | awk '{print $4}'`

TIME=`set \`date\`; echo $4`

TIME=`date | (read u v w x y z; echo $x)`

With some care, redefining the input field separators can help.

#!/bin/sh

# convert IP number to in-addr.arpa name

name()

{ set `IFS=".";echo $1`

echo $4.$3.$2.$1.in-addr.arpa

}

if [ $# -ne 1 ]; then

echo 1>&2 Usage: bynum IP-address

exit 127

fi

add=`name $1`

nslookup < < EOF | grep "$add" | sed 's/.\*= //'

set type=any

$add

EOF

Debugging

The shell has a number of flags that make debugging easier:

sh -n command

Read the shell script but don't execute the commands. IE. check syntax.

sh -x command

Display commands and arguments as they're executed. In a lot of my shell scripts you'll

see

# Uncomment the next line for testing

# set -x