**Examples of Features of the Shell**

**Pipe**

$ who |sort -r

& **Background**

$ sort file &

; **Separator**

$ date ;who

**()** **Grouping**

$ (date ; who) & 1s



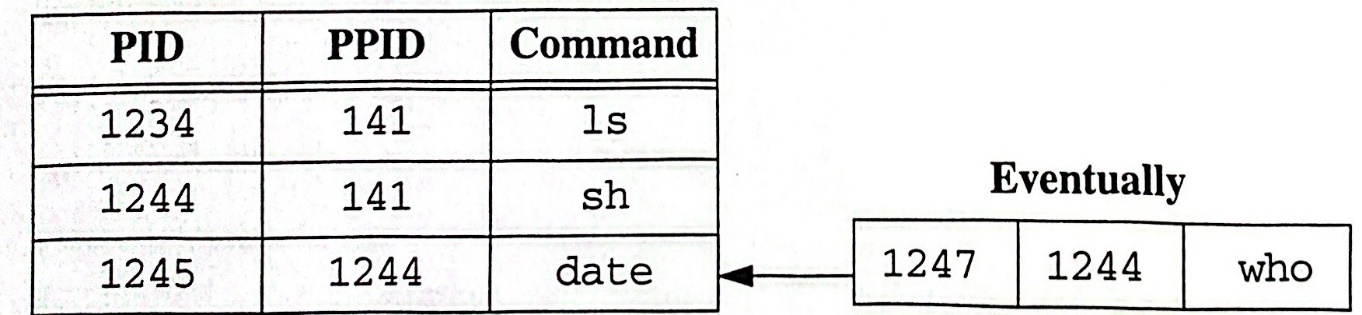
**Notes**

The Shell uses a pipe to connect the standard output of one program to the standard input of another.

The Shell executes the command associated with the symbol & in the background. The Shell prompt returns immediately.

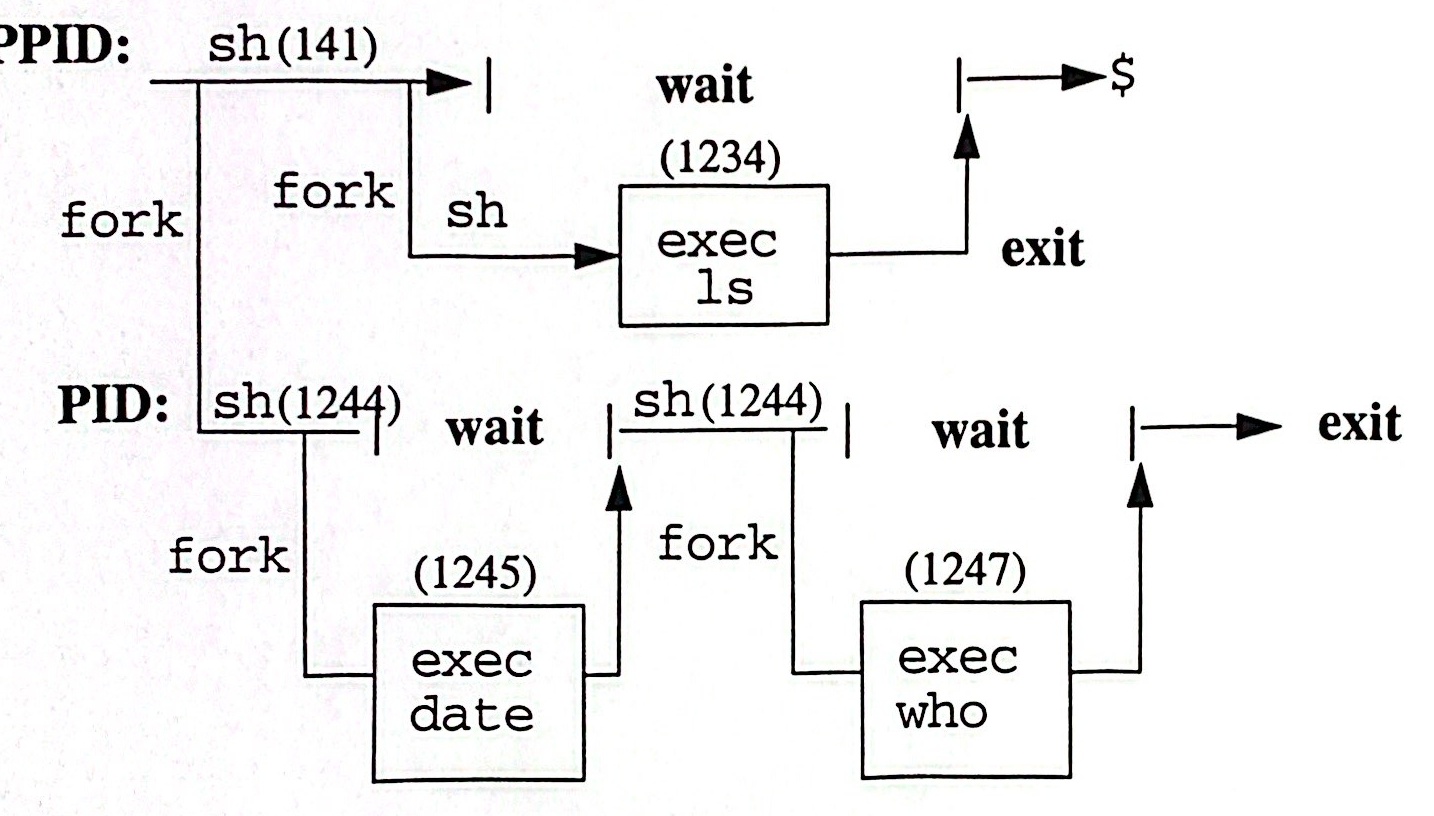
The semicolon can be used to insert one or more Shell commands on the same line. The commands are executed sequentially.

Each group of commands enclosed in parentheses is considered a job, executed by a subshell.



**Summary**

$ (date;who)&ls



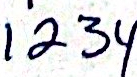
|  |  |  |
| --- | --- | --- |
| 1247 | 1244 | who |

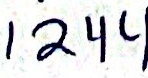
**Process Table**

|  |  |  |
| --- | --- | --- |
| PID | PPID | Command |
| 1234 | 141 | 1s |
| 1244 | 141 | sh |
| 1245 | 1244 | date |

**Notes**









The Shell as a Process

**Summary**

· The shell is an ordinary user program that **reads** **and** **executes commands.**

There are several UNIX shells, including the **Bourne,** Korn, **and** **C** **shells.**

Commands are executed by firstcreating **a** **new** **process with** fork **and** **then** executing the new program with exec.

The shell manages

- File name expansion

I/O redirection

- Pipelines

- Background processing

Command execution

Notes

1-23

UNIX Shell Programming

© Copyright 1996-2000 GTRC,All Rights Reserved

**Lab 1**

**Objective:**

In this exercise you will look at the process table and identify your process and its parent process. Also you will locate background processes in the process table and those process with the same process-ID, same group-ID,and same parent process-ID.

**Exercises:**

1. Use the ps command with appropriate options to get information about running processes.

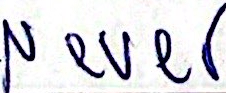
2.Create a subshell using sh. Again execute ps to see your current Shell.

3. Trace your shell back to its ancestor process init which has a process-ID of 1.

4. Now execute the command exec 1s and again look at the process table using ps.

5. Try exec 1s now. Explain the result.

6. Under what circumstances can two processes have the same:

· process-ID?  

·parent process-ID?

7. Locate two processes in the process table with the same parent.



**2**

**Built-In Com** **mands**

**Objectives**

**Recognize Shell Built-Ins**

**·Explain the difference between built-in commands** versusothercommands

**·Assign values to Shell variables and display them**



**Notes**

When the user enters a command for execution the Shel1 searches for the command on the disk and requests that the Kernel execute the command. However, some of the commands the She11 can execute itself. They are called built-in commands.

**Built-In Commands**

Some of the commands are internal to the Shell.

The commands are accessed faster and they are more efficient than external commands.

These commands do not require the Shell to fork in order to execute the command.

·These commands cannot be redefined by the user.

**Notes**

· In the chapter, we are building a list of built-in commands.



Built-In Commands

**Built-In Commands**

The built-in commands execute in **the** **current environment**

·The shell does not create a separate **process for** **the** built-in commands

· Output from the built-in **command** **goes** **to** **standard** **output** (the screen)

· Built-in commands can be used **in** **pipelines**

**Notes**

The built-in command returns an exit value of false(1) if the following conditions occur:

An invalid option

Incorrect number of arguments

- Incorrect argument

Invalid I/O redirection

Invalid variable assignment

2-4

UNIX Shell Programming

©Copyright 1996-2000 GTRC,All Rights Reserved

**Built-In Commands**

**Command:** **:**

**Effect:** **The null command does nothing, (0) exit status is returned**

**Example:** x=backup

if false

then:

else echo $x

fi

**Command:** **· file**

**Effect:** **Reads and executes commands in the file as input to the** **current shell and returns**

**Example:** $ echo $TERM

vt102

$· **change**

TERM=vt100

$ echo $TERM

vt100

**Notes**

Use the null command : when you want the command to do nothing. : returns an exit status of 0.

The search path spcified by PATH is used to find the directory containing file or charge.The commands in file or change are executed and any variables that are set within the scripts remain in effect after execution.

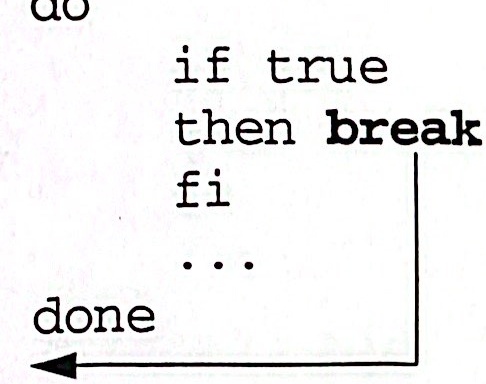
.is used to change variables in your environment.

**Built-In Commands**

**Command:** **break [n]**

**Effect:** **Allows premature loop termination. If** **n** is specified,it will **break out n levels.**

**Example:** for i in \*



**Command:** **continue [n]**

**Effect:** **Resumes the next iteration of a loop.**

**Example:** for i in \*

do

if true

then **continue**

fi

done



Notes

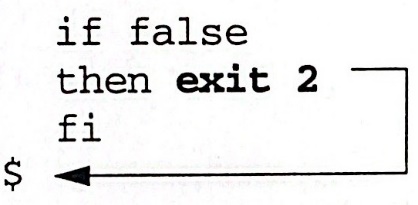
n is an integer≥1.

break goes to the bottom of a loop and cont inue goes to the top of the loop.

At a break execution continues after the loop. With continue the rest of the commands in the loop are skipped.

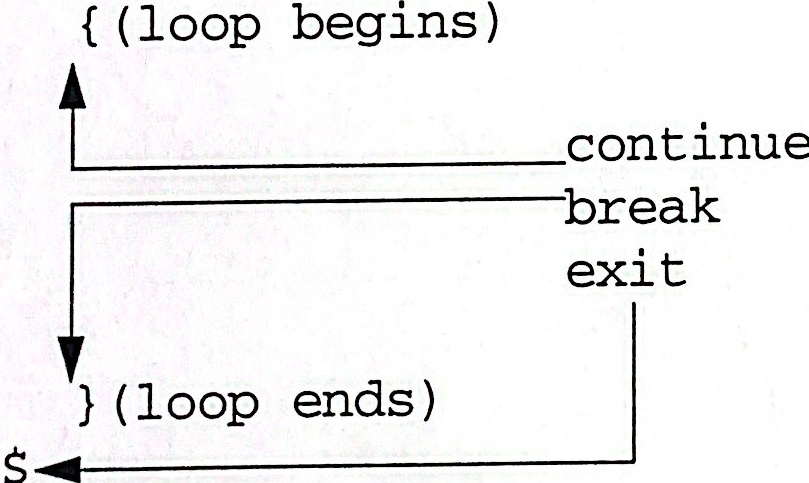
**Built-In Commands**

**Command:** **exit [n]**

**Effect:** **Terminates the subshell.If n is specified, then the exitstatus** **is n, otherwise it is that of the last command executed.**

**Example:**

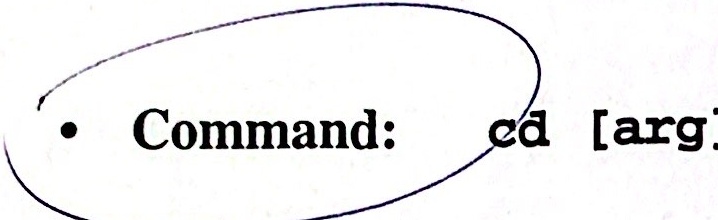
**break, continue,and exit**



**Notes**

exit is used to terminate your program.

**Built-In Commands**



**Effect:** **Changes directory**

**Example:** $ **cd /usr/bin**

$ pwd

/usr/bin

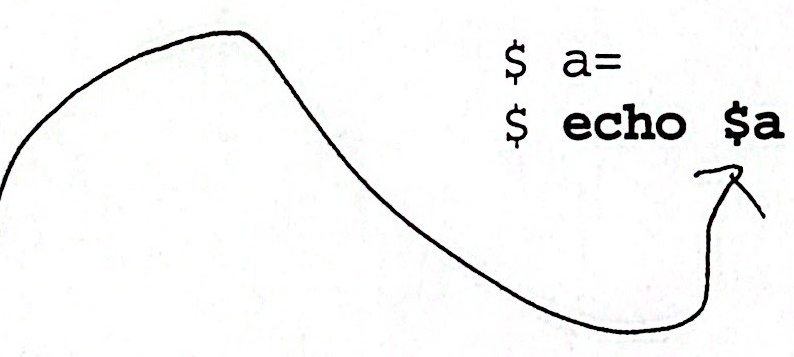
**Command:** **echo [arg]**

**Effect:** **Retrieves stored variables**

**Example:** $ a="Keep going"

$ **echo $a**

Keep going





**Notes**

The shell parameter CDPATH defines the search path for the directory containing arg.

cd will fail if you don't have execute permission on the directory you are moving to.

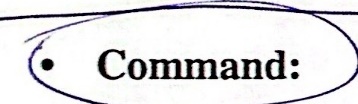
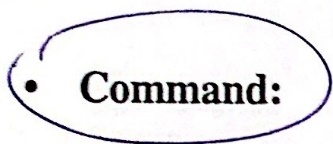
A dollar sign tells the Shell to evaluate the following Shell variable. If the variable is not set,the Shell returns nothing but your prompt.

Certain characters preceeded with a backslash have special meaning to echo. For example,\b is backslash, \007 is bell,etc.

**Built-In Commands**

**eval [command line]**

**Effect**: **Arguments are read as input to the shell and the resulting** **command(s) executed.**

**Example:** pipe="|"

**eval** ps $pipe 1p

**1st pass:** **and 1p are passed to ps as arguments**

**2nd pass:** **is recognized as the pipe symbol**

**exec [program]**

**Effect:** **Overlays existing Shell or processes with a new program.**

**The command executes without forking.**

**Example**: $ **exec** cat /etc/passwd

Thelogin:**message appears.**

$ sh -c **'exec** cat /etc/passwd'

**What is the difference?**

**Notes**

With eval the shell scans the command line twice before evaluating it.

sh -c indicates commands should be read from string.

**Built-In Commands**

**Command:** **export**

**Effect:**

**Exports variable to all subsequent subshells. All subshells** **have a copy of the exported variable. Variables are passed** **downward.**

**Example:** **Right** 

$ a=hi $

$ **export a**

$ sh $ sh

$ echo $a $echo $a

hi <nothing>



Notes



Built-In Commands

**Built-In Commands**

·Command:**hash [name]**

Effect:

**For each name,the location in thesearch path of the** **command specified by** name **is determined and remembered** **by the shell.**

Example:

$ **hash** **date** 1s # add date and 1s to hash list

$ **hash**

# print hash list

date=/usr/bin/date

1s=/usr/bin/ls

**$ hash -r**

# remove all commands from hash

**Command:**

**newgrp group**

Effect:

**Your real group-ID is changed to** group.If newgrpis **executed without an argument you are changed to your** **default group (specified** in/etc/passwd)

Example:

$ **newgrp admin**

# change to admin group

$ **newgrp**

# change back to default group

Command: pwd

Effect:

Print **your working directory**

Example:

$ **pwd**

/home

Notes

UNIX Shell Programming

2-11

©Copyright 1996-2000 GTRC,All Rights Reserved



Built-In Commands

**Built-In Commands**

**·** Command:

**read [variable]**

Effect:

**The** readcommand **takes a line** ofinput **and assigns** **consecutive words to the named variables.Any excess words** **are assigned to the last variable.**

**The read** commandallows **the user to enter data into the** **program** during **execution.**

**Examples:**

$ cat/r

**read** abc

echo Sa

echo $b

echo $c

$ cat read.var

echo 'Read in a string'

**read** mystring

echo $mystring

$ cat count

echo "Enter a line of text"

**read** text1 text2

echo "This is the first word: $text1"

echo "This is the rest of the line: $text2"

echo "This is a count of words and characters"

echo $text1 $text2 | wc

**Notes**

Words are delimited with spaces, tabs, and newlines.

$ read a b c

there are more than three words

S echo sa

there

$ echo $b

are

$ echo $c

more than three words

Words are delimited with spaces,tabs,and newlines.

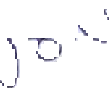
UNIX Shell Programming

©Copyright 1996-2000 GTRC,All Rights *Reserved*

2-12

**Built-In Commands**



**· Command:readonly [variable]**

**Effect:**

readonly **on** **a** **line** **by** **itself** shows the currently set readonly **variables.**

**It can also be used to make a variable** readonly.

Example: $ a=hi

$ **readonly a**

$ a=hello

a: is read only

**Notes**

Specified variables are read only and cannot be changed. To change values, the current Shell must be terminated.

·If you ty to change a readonly variable, you will get an error message.

readonly can be used to set your PATH, HOME, and other variables you do not want to change. 



Built-In Commands

**Built-In Commands**

**·Command:**

**set**

**Effect:**

Displays **Shell Variables**

Useset **for debugging shell scripts**

set -x **Prints only executable lines of a Shell script and** **executes.**

set -v **Prints every line of a Shell script and executes.**set -n **Reads commands of a Shell script but does not** **execute.**

**Example:**

$ **set**

HOME=/

MAIL=/usr/mail/root

PATH=:/bin:/usr/bin:/etc

PS1=#

TERM=vt100

**Notes**

parameters.The set command is used to display shell variables, set shell options, and assign positional

A Shell variable is a place holder for a sring of characters.

Using set you can determine currently defined Shell variables.

keys and cursor motion keys.

1. Change your terminal using the command.

$ TERM=vt52

Use set to verify assignment.

**$ set**

2. Assign a the value hello there.

$ a="hello there"

Use set toverify assignment.

**$ set**

When assigning variables, no spaces should surround the equal sign.

Q:How do you eliminate a variable?

A: You can set the variable to a null variable using a=""(no space between the double are erased.quotes). unset variable can be used on some systems.When you logout all set variables

line.The debugging tools can be used within Shell scripts,or they can be entered on the command

To turn the shell options on preceed the option with a dash, -.To turn the option use a plus, +.

UNIX Shell Programming

©Copyright 1996-2000 GTRC,All Rights Reserved

2-14



Built-In Commands

**Built-In Commands**

**· Command:** **shift [n]**

**Effect:**

**Provides a way to process or left shift positional parameters.**

**Example:**

$ set a b c

#set positional

#parameters 1, 2, and 3

$ echo $1 $2 $3

# display parameters

$ **shift**

# move positions to the

# 1eft and drops $1

$ echo $1 $2 $3

$ **shift 2**

$ echo $1 $2 $3

# what is the output?

**Command:**

**test <expr>**

**Effect:**

**Return the value of the expression**

**Example**:

a="hi"

**test $a = "hi "**

# returns false

**Notes**

Arguments to a command script are assigned numbers and are called positional parameters.

· By using an argument with shift,you can shift the positional parameters more than one position.

test is used with the if,while,and until commands.

A return value of O is true and otherwise the value is false.

An alternate version of test <expr> is [ expr ].The spaces around the brackets are required.

test can be used to check file types and file permissions, to compare strings, and to compare arithmetic expressions.

UNIX Shell Programming

©Copyright 1996-2000 GTRC,All Rights Reserved

2-15



Built-In Commands

**Built-In Commands**

Command:

**times**

**Effect:**

**Prints the accumulated user and system time for the process** **being executed from the current Shell.**

**Example:**

**$ times**

2m6s 1m3s

#2 minutes, 6 seconds

\*user time

# 1 minute, 3 seconds

#system time

**Command:**

**trap <cmd> <signal(s)>**

**Effect:**

trapcanbeused **in a script or application program such** **that during the execution of the program if the specified** **signal is received,then the corresponding command is** **executed.**

**Example:**

**trap** "rm /tmp/r\*" 2

**Remove the files in /tmp that start with r when an interrupt** **signal(2)is received**

**Notes**

The time is displayed in hundredths of a second.

The output of times varies greatly from one implementation to another.

timex.·There are three versions of times depending on your system; try times, time,and

Common signals and action:

Signal

Action

0

Exit from shell

1

Hangup

2

Delete/Interrupt

15

Kill

Quotes must be used around the command(s) if more than one command is used.

trap can be used to ignore signals within a program.

UNIX Shell Programming

©Copyright 1996-2000 GTRC,All Rights Reserved

2-16

**Built-In Commands**

Command: **type <cmd(s)>**

**Effect:** **Prints information about the indicated command(s).**

**Example:** $ **type** cd 1s

cd is a shell built-in

1s is hashed (/bin/1s)

Command**:** **ulimit <size>**

**Effect:** **Sets the maximum size (in blocks) of files that can be written** **by the child process.**

**Example**: $ **ulimit**

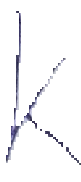
0 # unlimited size

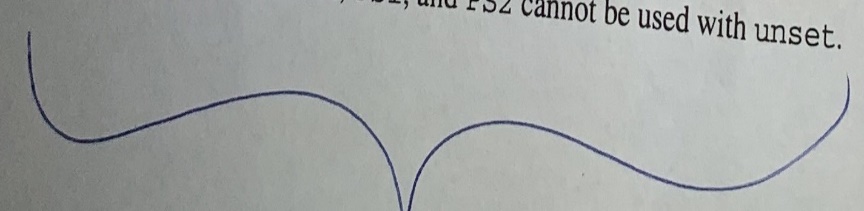
$ **ulimit 100**

$ ulimit

100 # size is set to 100

**Notes**



Built-In Commands

**Built-In Commands**

**· Command:**

**umask <mask>**

**Effect:**

**Sets the default file creation mask to** mask

**Example:**

$ **umask**

0002

**$ umask 066**

# remove read and write

# permission from group

# and other

$ touch memo

$ 1s -1 memo

-rwx---

memo

**· Command:**

**unset <name(s)>**

**Effect:**

**Erases definitions of previously set variables or functions.**

**Example:**

**unset** ac

# a and c are now undefined

**Notes**

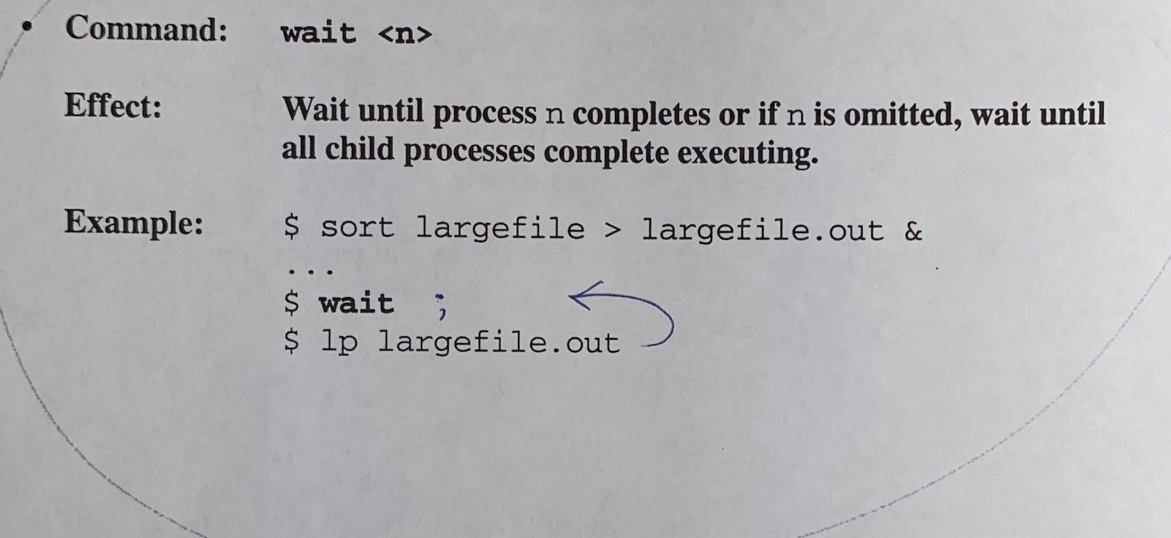
file.As files are created,the permissions are ANDed with the mask to determine the mode of the

IFS,MAILCHECK, PATH,PS1,and PS2 cannot be used with unset.

UNIX Shell Programming

©Copyright 1996-2000 GTRC,All Rights Reserved

2-18



Built-In Commands

**Built-In Commands**

**Notes**

wait can be used to wait on processes that have been sent to the background.

n above refers to the process-ID.

UNIX Shell Programming

2-19

©Copyright 1996-2000 GTRC,All Rights Reserved



Built-In Commands

**Advantages/Disadvantages of Built-in**

**Commands**

Advantages:

Command can change the shell's environment.

Shell does not have to fork a new process.

Does not have to search through directories looking for named executable program.

Therefore,program execution is faster.

Disadvantages:

Makes it difficult to execute custom program with same name as built-in.

Must recompile the shell to add new built-ins.

Notes

UNIX Shell Programming

2-20

©Copyright 1996-2000 GTRC,All Rights Reserved

**Built-In Commands Summary**

|  |  |
| --- | --- |
| Build-In  Commands | Effect |
| : | The null command does nothing, (0) exit status is returned. |
| .file | Reads and executes commands in the file. |
| break | Allows premature loop termination. |
| cd | Changes directory. |
| continue | Resumes the next iteration of a loop. |
| echo | Retrieves stored variables |
| eval | Arguments are read and the resulting command(s) executed. |
| exec | Overlays existing Shell or processes with a new process. |
| exit | Terminates the subshell. |
| export | Exports variable to all subsequent subshells. |
| hash | For each name, the location in the search path of the command specified by name is determined and remembered by the shell. |
| newgrp | Changes group-ID. |
| pwd | Prints your working directory. |
|  | The read command takes a line of input and assigns consecutive words to the named variables. |
| readonly | Shows the currenty set readonly variables. |
| set | Displays Shell Variables and sets debugging tools |
| shift | Provides a way to process or left shift positional parameters. |
| test | Returns the value of the expression |
| times | Prints the accumulated user and system time. |
| trap | If the specified signal is received, then the corresponding command is executed. |
| type | Prints information about the indicated command(s). |
| ulimit | Sets the maximum size of files that can be written by the child process. |
| umask | Sets the default file creation mask. |
| unset | Erases definitions of previously set variables or functions. |
| wait | Waits until process completes. |

2-21



Built-In Commands

**Lab 2**

**Objectives:**

·Now you will experiment with the built-in commands. At the present time you may not realize the full power of these commands, but many of the commands will be explored in more detail in later chapters. At the present time try to execute the indicated command to see how it reacts and how the displayed output changes.

**Exercises:**

1. Type in the following sequence of steps and explain the result.

if grep the \*

then :

else echo "not there"

fi

2. Put the following entry in your .profi le file then execute the commands in the file using .profile.

New Entry:

PS1="'hostname':\$PWD> "

3. Enter the following commands on the command line.

for i inx

do

echo $i

break

done

What happened?

4. In the example above change the word break to continue.

What happened?

5. Now change the word break to exit 2.

Explain the result

UNIX Shell Programming

2-22

©Copyright 1996-2000 GTRC,All Rights Reserved

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Task | Command | Result (1) | Result (2) |
| I | define a | a=hi |  |  |
| II | verify definition | echo $a |
| III | create a new shell | sh |  |  |
| IV | verify definition | echo $a |
| V | exit shell | exit |  |  |
| VI | verify definition | echo sa |



Built-In Commands

6. In your HOME directory make a subdirectory called secret.Assign permission 400 to secret.

Try to move to secret.

Explain the result

7. Find the value assigned to your terminal or window by using the echo command.

8. Define pipe in the following manner: pipe="|" (the double quotes are important)Use eval and spipe to send the output of ps to more.

9. Explain the difference between the following two commands

exec cat /etc/passwd sh -c 'exec cat /etc/passwd'

10. This is a short exercise in exporting

Now repeat the steps above, but after (I) export a.

Explain the difference after exporting a.

11.Execute some of your favorite commands,such as who, date,1s,etc. Repeat some of the commands.

UNIX Shell Programming

2-23

©Copyright 1996-2000 GTRC,All Rights Reserved

|  |  |
| --- | --- |
| Variable | Setting |
| MAIL |  |
| EDITOR |  |
| SHELL |  |
| TERM |  |
| PATH |  |
| a |  |



Built-In Commands

Run the hash command to print the hash list.

Remove all commands from your hash list.

12.Use the read command to assign variables.

Enter the read command followed by two arguments, say a and b.

The system hangs waiting for your input. Enter three words.

Display the value of your variables.

echo $a

echo $b

Are you surprised at the result?

Explain

13. Use the variable a that you defined in exercise 12.

Verify its definition using echo $a.

Make a a readonl y variable.

Try to assign a different value to a.

a=change

The only way to change a is to terminate your current shell.

14. Use the set command to display your current settings. Enter the value for the listed variables.

Add a new directory to your path.

UNIX Shell Programming

2-24

©Copyright 1996-2000 GTRC,All Rights Reserved

Define a new variable b.

Use set to display your new settings.

15. Use set as a debugging tool.

Turn on debugging using set -x.

Run several commands.

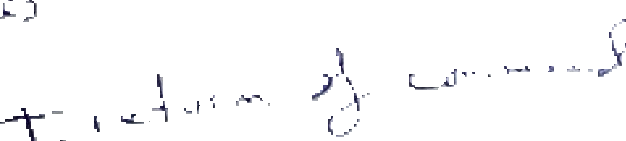
Turn of debugging using set +x.

16. This is an exercise combining set and shi ft. Peform the ollowing commands and explain the results.

|  |  |
| --- | --- |
| Command | Results |
| $ set all cows eat grass |  |
| $ echo $# |  |
|  |  |
| $ shift |  |
| $ echo $1 |  |
| $ shift 2 |  |
| $ echo $1 |  |

17.Look in all the files in your current directory for the word the.

Use the test command to verify the accuracy of your search.



grep the \*

test $? -eq 0

echo $? #display a 0 if the test is true

18. Use the times command to print the accumulated user and system times for the shell and for processes run from the shell.

19.Set your trap to stop when the Interrupt or Signal 2 is received.

**HP,IBM,Solaris:** s trap "banner stop" 2

**Linux,SunOS:** $ trap "echo STOP" 2

UNIX Shell Programming 2-25



Built-In Commands

Now use find to locate all files on the system having rc as part of their name, pipe to output to more.

$ find / -name "\*rc\*" -print | more

Press the Interrupt key or CTRL-C.

What happened?

20. What is the maximum size (in blocks) of files that can be written by your child processes?

Can you change this limit?

UNIX Shell Programming

2-26

©Copyright 1996-2000 GTRC,All Rights Reserved