MARUDHARKESARIJAINCOLLEGEFORWOMEN,VANIYAMBADI PG

DEPARTMENT OF COMPUTER APPLICATIONS

Subjectcode:23USCA14

Class : I-MCA

SUBJECT:LINUXANDSHELLPROGRAMMING

UNIT-

VSYLLABUS

Workingwithalternativeshells:Understandingthedash shells-Programminginthe dash shells-introducing the ZSH shell-writing script for ZSH.writing simple script utilities:Automatingbackups–Managinguseraccounts-watchingdiskspace producing scriptfordatabase,webeandemail;writingdatabaseshellscript-Emailingreportsfrom script.UsingpythonAsabashscriptingAlternative:technicalrequriments-Python language-Helloworldthepython way-Pythonicarguments-supplying arguments-supplying arguments-counting arguments-significant white space-Reading user input-Using python to write to files-String Manipulation.

WhatisthedashShell?

The Debi and dash shell has had an interesting past. It’s a direct descendant of the ash shell, a simple copy of the original Bourne shell available on Unix systems Kenneth Almquistcreated a small-scale version of the Bourne shell for Unix systems and called it the Almquist shell,which was then shortened to *ash*. This original version of the ash shell was extremely small andfast but without many advanced features, suchas command line editingor history features, making itdifficulttouseasaninteractiveshell.

The NetBSD Unix operating system adopted the ash shell and still uses it today as the default shell.The NetBSD developers customized the ash shell by adding severalnew features, makingitclosertothe Bourneshell. The new features include command line editing using both emacs and vi editorcommands and a history command to recall previously entered commands. This version of the ashshellisalsousedbytheFreeBSDoperatingsystemasthedefaultloginshell.

The Debi a Linux distribution created its own version of the ash shell (called Debi anash, or *dash*) forinclusionin its version ofLinux. For the most part, dashcopies the features of the NetBSDversion of the ash shell, providing advanced command-line editing capabilities.

Everyshellscript muststartwithalinethat declarestheshell used forthe script. Inourbashshellscripts, we’vebeenusingthis:

#!/bin/bash

This tells the shell to use the shell program located at /bin/bash to execute the script. In the UNIX world, the default shell was always /bin/sh. Many shell script programmers familiar with the UNIX environment copy this into their Linux shell scripts:

#!/bin/shunmostLinuxdistributions;the/bin/shefiles isasymbolic link

ThedashShellFeatures:

Althoughboththe bashshelland the dashshellare modeled after the Bourne shell, they have some differences. This section walks you through the features found in the Debiandash shell to acquaint you with how the dash shell works beforethe shellscripting features.

Thedashcommandlineparameters

Thedashshellusescommandlineparameterstocontrolitsbehavior.The liststhe command line parameters and describes what each one does.

ThedashCommandLineParameters
Parameter Description:
-a-Exportsallvariablesassignedtotheshell
-c-Readscommandsfromaspecifiedcommandstring
-e-Ifnotinteractive,exitsimmediatelyifanyuntestedcommandfails
-f-Displayspathnamewildcardcharacters
-n-Ifnotinteractive,readscommandsbutdoesn'texecutethem
-u-WritesanerrormessagetoSTDERRwhenattemptingtoexpandavariablethat is not set
-v-WritesinputtoSTDERRasitis read
-x-WriteseachcommandtoSTDERRasitisexecuted
-I-IgnoresEOFcharactersfromtheinputwhenininteractivemode
-i-Forcestheshelltooperateininteractive mode
-m-Turnsonjobcontrol(enabledbydefaultininteractivemode)
-s-Readscommands fromSTDIN(thedefaultbehaviorifno fileargumentsare Present)
-E-Enablestheemacscommandlineeditor
-V-EnablestheVIcommandlineeditor

Debi an added a few additional command line parameters to the original ash shell command
Lineparameterlist.
The-Eand-Vcommandlineparametersenablethespecialcommand

Thedashenvironmentvariables:
The dash shell uses quite a few default environment variables uses to track information,and you can create your own environment variables as well. This section describes theenvironment variables and how dash handles them.

Defaultenvironmentvariables
The dashenvironment variables are verysimilar to the environment variables used in bash.This is not by accident. Remember that both the dash and bash shells are extensionsoftheBourneshell,sotheybothincorporatemanyofitsfeatures.
However,becauseofitsgoalof simplicity,thedashshell containssignificantly fewer environment variables than the bash shell. You need to take this into consideration when creating shellscripts in a dash shell environment.
Thedashshellusesthesetcommandtodisplayenvironmentvariables:
Positionalparameters:
In addition to the default environment variables, the dash shell also assigns special variablesto any parameters defined in the command line. Here are the positional parametervariables available for use in the dash shell:
■ $0:Thenameoftheshell
■ $n:Thenthpositionparameter

■ $*:Asingle valuewiththecontentsofalltheparameters,separatedbythefirst character inthe IFS environment variable, or a space ifIFS isn't defined
■ $@Expandstomultipleargumentsconsisting of all thecommandline parameters
■ $#:Thenumberofpositionalparameters
■ $?:Theexitstatusofthemostrecentcommand
■ $-:Thecurrentoption flags
■ $$:TheprocessID(PID)ofthecurrentshell
■ $!:TheprocessID(PID)ofthemostrecentbackgroundcommand

All the dash positional parameters mimic the same positional parameters available in the Bash shell. You can use each of the positional parameters in yourshell scripts just as you
wouldbeinthebashshell.

User-definedenvironmentvariable:

Thedashshellalsoallowsyoutosetyourownenvironmentvariables.Aswith bash, you can define a new environment variable on the command line by using theassignment
statement:

```
$testing=10;exporttesting
$echo$testing
10
$
```

Without the export command, user-defined environment variables are visible only in the current shell or process.

Thedashbuilt-incommands:

Justaswiththebashshell, thedashshellcontainsasetof built-incommandsthat it recognizes.
Youcanusethesecommandsdirectlyfromthecommand line interface, oryoucan incorporate them in your shell scripts.

ThedashShellBuilt-InCommands

CommandDescription

Alias-Createsanaliasstringtorepresentatextstring Bg-
Continues specified job in background mode
cd-Switchestothespecifieddirectory
echo-Displaysatextstringandenvironmentvariables
eval- Concatenates all arguments with a space
Exec-Replacestheshellprocesswiththespecifiedcommand
Exit- Terminates the shell process
Export-Exportsthespecifiedenvironmentvariable foruseinallchildshells fg -
Continues specified job in foreground mode
getopts-Obtainsoptionsandargumentsfroma listofparameters
hashMaintainsandretrievesahashtableofrecentcommandsandtheirlocations pwd--
Displays the value of the current working directory

read ---Reads a line from STDIN and assigns the value to a variable
read-only--Readsaline fromSTDINtoavariablethat can’tbechanged
printf-- Displays text and variables using a formatted string
set-Listsorsetsoptionflagsandenvironmentvariables
shift-Shiftsthepositionalparametersaspecified numberoftimes test
-Evaluates anexpression and returns 0 iftrue or 1 if false
times-Displaystheaccumulated userandsystemtimes fortheshellandallshell processes
trap-Parsesandexecutesanactionwhentheshellreceivesaspecifiedsignal
type-Interpretsthespecified nameanddisplaystheresolution(alias, built-in, command,
keyword)
ulimit-Queriesorsetslimitsonprocesses
umask-Setsthevalueofthedefault fileanddirectorypermissions
unalias- Removes the specified alias
unset-Removesthespecified variableoroptionflagfromtheexported variables
wait- Waits for the specified job to complete and returns the exit status

Scripting in dash:

Unfortunately,thedashshelldoesn’trecognizeallthescriptingfeaturesofthebash shell.

- Shellscripts writtenfor the bashenvironment often failwhenrun inthe dash shell,causing all sorts of grief for shell script programmers. This section describes the differences
- you’ll need to be aware of to get your shell scripts to run properly in a dash shell environment.

Creatingdashscripts:

You probably guessed by now that creating shell scripts for the dash shell is pretty similarto creatingshellscripts for the bash shell. Youshould always specify which shell you wantto use in your script to ensure that the script runs with the proper shell.

Youdothisonthefirstlineoftheshell:

#!/bin/dash

You can also specify a shell command line parameter on this line, as was documented earlier in“Thedashcommandlineparameters”section. Things that don’t work

Unfortunately, because the dash shell is only a subset of the Bourne shell features, somethings in the bash shell scripts don’t work in the dash shell. These are often called bashisms.

This section is a quick summaryof bashshell features you maybe used to using in your bashshellscriptsthatdon’tworkifyou’reinadashshellenvironment.

Usingarithmetic

Itshowedthreewaystoexpressamathematicaloperationinthebashshellscript:

■ Usingtheexprcommand:exproperation

- Usingsquarebrackets:$[operation]
- Usingdoubleparentheses:$((operation))

The dashshellsupports the exprcommand and the double parentheses method but doesn’t supportthesquarebracketmethod.Thiscanbeaproblemifyouhavelotsof mathematicaloperations that use square brackets.

Theproper format forperforming mathematicaloperationsindashshellscriptsis to use the double parentheses method:

```
$cattest5b
#!/bin/dash
#testingmathematicaloperations
value1=10
value2=15
value3=$(($value1*$value2)) echo
"The answer is $value3"
$ ./test5b
Theansweris150
$
```

Nowtheshellcanperformthecalculationproperly.

Thetestcommand:

The bashshelltest command allows youto use the double equalsign(==) to test if two strings are equal. Thisis an add-on to accommodate programmers familiar with using this format in other programming languages.

However, the test command available in the dash shell doesn’t recognize the == symbol for text comparisons. Instead, it only recognizes the = symbol. If you use the==symbolinyourbashscripts, youneedtochangethetextcomparisonsymbol to just a single equal sign:

```
$ cat test7
#!/bin/dash
#testingthe=comparison
test1=abcdef
test2=abcdef
if[$test1=$test2] then
echo"They'rethesame!"else
echo"They'redifferent"f
i
$ ./test7
They'rethesame!
$
```

Thislittlebashismisresponsible formanyhoursoffrustrationforshell programmers!

ThefunctionCommand:

Itshowed youhowtodefine yourownfunctionsinyourshellscripts.Thebash shell supports two methods for defining functions:

■ Usingthefunction()statement

■ Usingthefunctionname only

Thedashshelldoesn’tsupportthefunctionstatement.Instead,inthedashshell you mustdefineafunctionusingthefunctionnamewithparentheses.

Ifyou’rewriting shell scriptsthatmay beusedin thedash environment,always definefunctions using the function name and not the function() statement:

```
$cattest10
#!/bin/dash
#testingfunctions
func1() {
echo"Thisisanexampleofafunction"
}
count=1
while[$count-le5] do
func1
count=$(($count+1))
done
echo"Thisistheendoftheloop"func1
echo"Thisistheendofthescript"
$ ./test10
Thisisanexampleofa     function
Thisisanexampleofa     function
Thisisanexampleofa     function
Thisisanexampleofa     function
Thisisanexampleofa     function
This is the end of the loop
Thisisanexampleofa     function
Thisisanexampleofa     function
This is the end of the script
$
```

Now the dashshellrecognizes the function defined inthe script just fine and uses it withinthe script.

ThezshShell:

Another popular shell that you may run into is the Z shell (called zsh). Thezsh shell is an open-source Unix shell developed by Paul Falstad. It takesideasfrom all the existing shells and adds many unique features to create a full-blown advanced shell designed for programmers.

Thefollowingaresomeofthe featuresthatmakethezshshellunique:

■ Improvedshelloptionhandling

■ Shellcompatibilitymodes

■ Loadablemodules

Of all these features, a loadable module is the most advanced feature in shell design. As you've seen in the bash and dash shells, each shell contains a set of built-incommands thatare available withoutthe need forexternalutilityprograms. The benefit of built-in commands is execution speed. The shell doesn't haveto load a utility program into memory before running it; the built-in commands are already in the shell memory, ready to go.

- The zshshellprovides a coresetofbuilt-in commands, plus the capabilityto add more command modules. Each command module provides a set of additional built-in commands for specific circumstances, such as network support and advanced math functions. You can add only the modules you think you need for your specific situation.
- This feature provides a great way to limit the size of the zsh shell for situations thatrequire a small shell size and few commands or expand the number of available built-in commands for situations that require faster execution speeds.

PartsofthezshShell:

The built-in command that are available (or can be added byinstalling modules), as well as the command line parameters and environment variables used by the zsh shell.

Shelloptions:

The most shells use command line parameters to define the behavior of the shell. The zshshelluses a few command line parameters to define the operation of the shell, but mostly it uses options to customize the behavior of the shell. You can set shell options either on the command line or within the shell itself using the set command

ParameterDescription:

-cExecutesonlythespecifiedcommandandexits

-iStartsasaninteractiveshell,providingacommandline interfaceprompt

-sForcestheshelltoreadcommandsfromSTDIN

-oSpecifiescommandlineoptions

Although this may seem like a small set of command line parameters, the -o parameter is somewhat misleading. It allows you to set shell options that define features within the shell.

By far, the zsh shell is the most customizable shell available. You can alter lots of features for your shell environment.

Thedifferentoptionsfitintoseveralgeneralcategories:

- Changingdirectories:Optionsthatcontrolhowthecdanddirscommands handle directory changes
- Completion:Optionsthatcontrolcommandcompletionfeatures
- Expansionandglobbing:Optionsthatcontrolfileexpansionincommands
- History:Optionsthatcontrolcommandhistoryrecall
- Initialization:Optionsthatcontrolhowtheshellhandlesvariablesandstartup files when started
- Input/output:Optionsthatcontrolcommandhandling
- Jobcontrol:Optionsthatdictatehowtheshellhandlesandstartsjobs
- Prompting:Optionsthatdefine howtheshellworkswithcommand lineprompts
- Scriptsand functions:Optionsthatcontrolhowtheshellprocessesshellscripts and defines shell functions
- Shellemulation:Optionsthatallow youtosetthebehaviorofthezshshellto mimic the behavior of other shell types
- Shellstate:Optionsthatdefinewhattypeofshelltostart
- zle:Optionsforcontrollingthezshlineeditor(zle) feature
- Optionaliases:Specialoptions that canbe used as aliases forotheroption names

With this many different categories of shell options, you can imagine just how many actual optionsthezshshellsupport.

Built-incommands:

The zsh shell is unique in that it allows you to expand the built-in commands available in the shell. This provides for a wealth of speedy utilities at your fingertips for a host of different applications.This section describes the core built-incommands, along withthe various modules available at the time ofthis writing.

Corebuilt-incommands:

CommandDescription

Alias Defines an alternate name for a command and arguments

autoload Preloads a shell function into memory for quicker access

bg Executes a job in background mode

bindkeyBindskeyboardcombinationstocommands

builtinExecutes thespecifiedbuilt-incommandinsteadofanexecutablefileof

thesamename
bye Thesameasexit
cd Changesthecurrentworkingdirectory
chdir Changesthecurrentworkingdirectory
command Executesthespecifiedcommandasanexternalfile insteadofa
function orbuilt-in command
declare Sets thedatatypeofavariable(sameastypeset)
dirs. Displays the contents ofthe directory stack
disable Temporarilydisablesthespecifiedhashtableelements
disown Removes the specified job from the job table
echo Displaysvariablesandtext
emulate Setszshtoemulateanothershell,suchastheBourne,Korn,orC shellsenable
Enables the specified hash table elements
evalExecutes thespecifiedcommandandarguments inthecurrentshell
process
exec Executesthespecifiedcommandandargumentsreplacingthecurrent
shell process
exit Exitstheshellwiththespecifiedexitstatus.Ifnonespecified,usethe status
of the last command
export Allowsthespecifiedenvironmentvariablenamesandvaluestobeusedin child
shell processes false Returns an exit status of 1

The zsh shell is no slouch when it comes to providing built-in commands! You should recognize most of these commands from their bash counterparts. The most important features of the zsh shell built-in commands are modules.

Add-inmodules:

There's a long list of modules that provide additional built-in commandsfor the zsh shell, and the list continues to grow as resourceful programmers create new modules. It shows some of the more popular modules available.

ModuleDescription:

zsh/datetime-Additionaldateandtimecommandsandvariables
zsh/files- Commands for basic file handling
zsh/mapfile-Accesstoexternalfilesviaassociativearrays
zsh/math func - Additional scientific functions
zsh/pcre-Theextendedregularexpressionlibrary
zsh/net/socket -Unix domain socket support
zsh/stat-Accesstothestatsystemcalltoprovidesystemstatistics
zsh/system Interface- for various low-level system features
zsh/net/tcp -Access to TCP sockets

zsh/zftp-AspecializedFTPclientcommand
zsh/zselect-Blocksandreturnswhenfiledescriptorsareready
zsh/zutil- Various shell utilities

The zsh shell modules cover a wide range of topics, from providing simple command line editing features to advanced networking functions. The idea behind the zsh shell is to provide a basic minimum shell environment and let you add on the pieces you need to accomplish your programming job.

Viewing,adding,andremovingmodules:

The zmodload command is the interface to the zsh modules. You use thiscommand to view, add, and remove modules fromthe zsh shell session. Usingthe mod load command without any command line parameters displays the currentlyinstalled modules in your zsh shell:

```
% zmodload
zsh/zutil
zsh/complete
zsh/main
zsh/terminfo
zsh/zle
zsh/parameter
```

Different zshshellimplementations include different modules bydefault. Toadd a new module, just specifythe module name onthe zmodload command line:

```
%zmodloadzsh/zftp
%
```

Nothingindicatesthatthemoduleloaded.Youcanperformanotherzmodload command, and the new module should appear inthe list of installed modules.

After you load a module, the commands associated with the module are available as built-in commands:

```
%zftpopenmyhost.comrichtesting1 Welcome
to the myhost FTP server.
%zftpcodtest
%zftpdir
01-21-1111:21PM120823test1
01-21-1111:23PM118432test2
%zftpgettest1>test1.txt
%zftp close
%
```

The zftp command allows you to conduct a complete FTP session directly from your zsh shell command line! You can incorporate these commands into your zsh shell scripts to perform file transfers directly from your scripts.

Toremoveaninstalledmodule,usethe -uparameter,alongwiththe modulename:

```
%zmodload-uzsh/zftp
%zftp
zsh:commandnotfound:zftp
%
```

Scriptingwithzsh:

The main purpose of the zsh shell was to provide an advanced programming environment for shell programmers. With that in mind, it’s no surprise that the zsh shell offers many features that make shell scripting easier.

Mathematicaloperations:

As you would expect, the zsh shell allows you to perform mathematical functions with ease. In the past, the Korn shell has led the way in supporting mathematical operations by providing support for floating-point numbers. The zsh shell has full support for floatingpoint
numbersinallitsmathematicaloperations! Performing
calculations
Thezshshellsupportstwomethodsforperformingmathematicaloperations:

- Theletcommand
- Doubleparentheses

Whenyouusethe letcommand, youshouldenclosetheoperationindouble
quotation
markstoallowforspaces:

```
%letvalue1="4*5.1/3.2"
%echo$value1
6.3750000000
%
```

Be careful,using floating pointnumbersmay introducea precision problem.To solve this,
it’salwaysagoodideatousetheprintfcommandandtospecifythedecimal precision
neededtocorrectlydisplaytheanswer:

```
%printf"%6.3f\n"$value1
6.375
%
```

Nowthat’smuchbetter!

Thesecond method istousethedoubleparentheses.This method incorporatestwo techniques fordefiningthemathematicaloperation:

```
%value1=$((4*5.1))
%((value2= 4*5.1))
%printf"%6.3f\n"$value1$value2
20.400
20.400
%
```

Notice that you can place the double parentheses either around just the operation (preceded by a dollar sign) or around the entire assignment statement. Both methods produce the same results.

Mathematicalfunctions

With the zsh shell, built-in mathematical functions are either feast or famine. The default zshshelldoesn’tincludeanyspecial mathematicalfunction. However, ifyou install the zsh/mathfunc module, you have more math functions than you’ll most likely ever need:

```
%value1=$((sqrt(9)))
zsh:unknownfunction:sqrt
%zmodloadzsh/mathfunc
%value1=$((sqrt(9)))
%echo$value1
3.
%
```

That was simple!Now youhave anentire mathlibraryoffunctions atyour fingertips.

Mathematicalfunctions:

With the zsh shell, built-in mathematical functions are either feast or famine. The default zsh shell doesn’t include any special mathematical function. However, if you install the zsh/math fun module, you have more math functions than you’ll most likely ever need:

```
%value1=$((sqrt(9)))
zsh:unknownfunction:sqrt
%zmodloadzsh/mathfun
%value1=$((sqrt(9)))
%echo$value1
```

```
3.
%
```

That was simple!Now youhave anentire mathlibraryoffunctions atyour fingertips.

Structuredcommands:

Thezshshellprovidesthe usualsetofstructuredcommandsforyourshellscripts:

- if-then-elsestatements
- Forloops(includingtheC-style)
- whileloops
- untilloops
- selectstatements
- casestatements

The zsh shell uses the same syntax for each of these structured commands that you’re used to from the bash shell. The zsh shell also includes adifferent structured command called repeat. The repeat command uses this format:

```
repeatparam
do
commands
done
```

The paramparameter must be a number or a mathematical operation that evaluates toa number. The repeat command then performs the specified commands that number of times:

```
%cattest1
#!/bin/zsh
#usingtherepeatcommand
value1=$(( 10 / 2 ))
repeat$value1
do
echo"Thisisatest"done
$./test1
Thisisatest
Thisisatest
Thisisatest
Thisisatest
Thisisatest
%
```

Thiscommandallowsyoutorepeatsectionsofcodeforasetnumberoftimes based on a calculation.

Functions:

The zshshellsupports the creationof your own functions either usingthe function command or by defining the function name with parentheses:

```
%functionfunctest1{
>echo"Thisisthetest1function"
}
%functest2()
{
>echo"Thisisthetest2function"
}
%functest1
Thisisthetest1 function
%functest2
Thisisthetest2 function
%
```

As with bash shell functions (see Chapter 17), you can definefunctions within your shell script and then either use global variables or pass parameters to your functions.

WritingSimpleScriptUtilities:

Automatingbackups:

The responsible for a Linux system in a business environment or just using it at home, the loss of data can be catastrophic. To help prevent bad things from happening, it’s always a good idea to perform regular backups (or archives).
However, what’s a good idea and what’s practical are often two separate things. Trying to arrangea backup schedule to store importantfiles can be a challenge. This is another place where shell scripts often come to the rescue.It demonstrates two methods for usingshellscripts to archive data on your Linux system.

Archivingdatafiles:

If you’re using your Linux systemto work on an important project, you can createa shellscript that automatically takes snapshots ofspecific directories. Designating these directories in a configuration file allows you to change them when particular project changes. This helps avoid a timeconsuming restore process from your main archive files

Obtainingtherequiredfunctions:

TheworkhorseforarchivingdataintheLinuxworldisthetarcommand.
The tar command is used to archive entire directories into a single file. Here’s anexample ofcreatinganarchive file ofa workingdirectory usingthe tar command:

```
$tar-cfarchive.tar/home/Christine/Project/*.*
tar: Removing leading '/' from member names
$
```

```
$ls-larchive.tar
-Raw-raw-r--.1Christine51200Aug2710:51archive.tar
```

Insteadofmodifyingorcreatinga newarchivescript foreachnewdirectoryor ifle you want to back up, you can use a configuration if le. The configuration if le should contain
Eachdirectoryorifleyouwanttobeincludedinthe archive.

```
$catFiles_To_Backup
/home/Christine/Project
/home/Christine/Downloads
/home/Does_not_exist
/home/Christine/Documents
```

Creatingadailyarchivelocation:

Ifyouarejustbackingupa few fi les,it’sfineto keepthearchive inyourpersonal directory.
However,if several directoriesarebeingbackedup,itisbesttocreateacentral repository archive directory:

```
$sudomkdir/archive
[sudo]passwordforChristine:
$
$Less-LD/archive
Drawer-or-x.2rootroot4096Aug2714:10/archive
$
```

After you have your central repository archive directory created, you need to grant access to it for certain users. If you do not do this, trying to create if les in this directory fails, as shown here:

```
$MyFiles_To_Backup/archive/
My: cannot move 'Files_To_Backup' to
'/archive/Files_To_Backup': Permission denied
$
```

You could grant theusers needing to createif les in this directory permission via sudo or create a user group. Inthis case, a special user group is created, Archives:

```
$Sudogrouped Archives
$
$Sudochirp Archives /archive
$
$Less-LD/archive
```

Runningthedailyarchivescript:

Beforeyouattempttotestthescript,rememberthatyouneedtochange permissions
on

thescriptifle(seeChapter11).Thefile’sownermustbegivenexecute(x) privilege before thescriptcanberun:

```
$ls-lDaily_Archive.sh
-Raw-raw-r--.1Christine1994Aug2815:58Daily_Archive.sh
$
$chmodu+xDaily_Archive.sh
$
$ls-lDaily_Archive.sh
-rwxrw-r--.1ChristineChristine1994Aug2815:58Daily_Archive.sh
$
```

TestingtheDaily_Archive.shscriptisstraightforward:

```
$./Daily_Archive.sh
/home/Does_not_exist, does not exist.
Obviously,Iwillnotincludeitinthisarchive. It is
listed on line 3 of the config file.
Continuingtobuildarchivelist...
Starting archive...
Archivecompleted
Resultingarchivefileis:/archive/archive140828.tar.gz
$ls/archive
archive140828.tar.gzFiles_To_Backup
$
```

You can see that the script caught one directory that does not exist, /home/Does_not_exist. It lets you know what line number in the configuration file this erroneous directory is on and continues making a list and archiving the data. Your datais now safely archived in a tarball file.

Creatinganhourlyarchivescript:

Thehigh-volumeproductionenvironmentwherefilesarechangingrapidly,a daily archive might not be good enough. If you want to increase the archiving frequency to hourly, you need to take another item into consideration.When backing up files hourly and trying to use the date command to timestamp each tarball, things can get pretty ugly pretty quickly. Sifting through a directory of tarballs with filenames looking like this is tedious:

```
archive010211110233.tar.gz
```

- Instead of placing all the archivefilesin the samefolder,you can create a directory hierarchy for your archived files.

- The archive directory contains directories for each month of the year, using the month number as the directory name. Each month’s directory in turn contains folders for each day of the month (using the day’s numerical value as the directory name). This allows you to just timestamp the individual tarballs and place them inthe appropriate directory for the dayand month.
- First, the new directory /archive/hourly must be created, along with the appropriatepermissions set upon it. Rememberfromearly inthis chapterthat members of the archivers group are granted permission to create archives in this directory area. Thus, the newly created directory must have its primary group and group permissions changed:

```
$ sudo  mkdir  /archive/hourly
[sudo]passwordforChristine:
  $
$sudochgrpArchivers/archive/hourly
$
$ls-ld/archive/hourly/
Drawer-or-x.2rootArchivers4096Sep209:24/archive/hourly/
$
$sudochmod775/archive/hourly
$
$ls -ld/archive/hourly
drwxrwxr-x.2rootArchivers4096Sep209:24/archive/hourly
$
```

After the new directory is set up, the Files_To_Backup configuration file for the hourly archives can be moved to the new directory:

```
$catFiles_To_Backup
/us/local/Production/MachineErrors
/home/Development/SimulationLogs
$
$MyFiles_To_Backup/archive/hourly/
$
```

Now,there isa newchallengetosolve.The script mustcreatethe individualmonth and day directories automatically. If these directories already exist, and the script tries to create them, anerror is generated. This is not a desirable outcome!

ManagingUseraccounts:

Managing user accounts is much more than just adding, modifying, and deleting accounts. You must also consider security issues, the need to preserve work, andthe accurate managementofthe accounts. This canbe a time-consuming task. Here is another instance whenwriting script utilities is a realtimesaver!

Obtainingtherequiredfunctions

Deletinganaccountisthemorecomplicatedaccountsmanagementtask.When deleting an account, at least four separate actions are required:

1. Obtainthecorrectuseraccountnametodelete.
2. Killanyprocessescurrentlyrunningonthesystemthatbelongstothataccount.
3. Determinealliflesonthesystembelongingtotheaccount.
4. Removetheuseraccount.

It’s easy to miss a step. The shell script utility in this section helps you avoid making such mistakes.

Gettingthecorrectaccountname:

The first step in the account deletion process is the most important: obtaining the correct user account name to delete. Because this is an interactive script, you can use the read command (see Chapter 14) to obtain the account name. If the script user walks away and leaves the question hanging, you can use the -t option on the read command and timeout after giving the script user 60 seconds to answer the question:

```
Echo"Pleaseenterthe usernameoftheuser”
Echo-e"account youwishtodelete fromsystem: \c"
read -t 60 ANSWER
```

Because interruptionsarepartof life,it’sbesttogive usersthreechancestoanswer the question. This is accomplished by using a while loop (Chapter 13) with the -z option, to test whether the ANSWER variable is empty. The ANSWER variable is empty when the script first enters the while loop on purpose. The question to fill the ANSWER variable is at the end of the loop:

```
while[-z"$ANSWER"]
Do
[...]
Echo"Pleaseenterthe usernameoftheuser”
Echo-e"account youwishtodelete fromsystem: \c"
Read -t 60 ANSWER
Done
```

Creatingafunctiontogetthecorrectaccount name:

The first thing you need to do is declare the function’s name, get answer. Next, clear out any previous answers to questions your script user gave using the unset command the code to do these two items looks like this:

```
Functiongetsanswer{ #
unsetANSWER
```

The other original code item you need to change is the question to the script user. The script doesn’t ask the same question each time, so two new variables are created, LINE1 andLINE2,tohandlequestionlines:

```
echo$LINE1
echo-e$LINE2"\c"
```

statement(seeChapter12)assistswiththis problem.The functiontestsifLINE2is empty andonlyusesLINE1ifitis:

```
if[-n"$LINE2"]
then
echo$LINE1
echo-e$LINE2"\c"else
echo-e$LINE1"\c"fi
```

Finally, thefunction needs to clean up after itself by clearing out theLINE1 andLINE2 variables.Thus,thefunctionnowlookslikethis:

```
function get_answer {
#
unsetANSWER
ASK_COUNT=0
#
while[-z"$ANSWER"] do
ASK_COUNT=$[$ASK_COUNT+1] #
case$ASK_COUNTin
2)
echo
[...]
esac
#
echo
if[-n"$LINE2"]
then#Print2lines
echo $LINE1
echo-e$LINE2"\c"
else #Print 1 line
echo-e$LINE1"\c"fi
#
```

```
read-t60ANSWER
done
#
unsetLINE1
unsetLINE2
#
} #End of get_answer function
```

Verifyingtheenteredaccountname

Because of potential typographical errors, the user account name that was entered should beverified.Thisiseasy becausethecodeisalreadyin placetohandleasking a question:

```
LINE1="Is $USER_ACCOUNT the user account "
LINE2="youwishtodelete
fromthesystem?[y/n]"get_answer
```

After the question is asked, the script must process the answer. The variable ANSWER again carries the script user’s answer to the question. If the user answered “yes,” the correct user account to delete has been entered and the script can continue. A case statement processes the answer. The case statement must be coded so it checks for themultiple ways the answer “yes” can be entered.

```
case $ANSWER in
y|Y|YES|yes|Yes|yEs|yeS|YEs|yES )
#
;;
*)
echo
echo "Because the account, $USER_ACCOUNT, is not "
echo"theoneyouwishtodelete,weareleavingthescript..."echo
exit
;;
esac
```

Sometimes, this script needs to handle a yes/no answer fromthe user. Thus, again, it makes sense to create a function to handle this task. Only a few changes need to be made to the preceding code.

Determiningwhethertheaccountexists:

The user has given us the name of the account to delete and has verified it. Now is a good time to double-check that the user account really exists on the system.Also,it isa goodideato show thefull account record to the script user to checkone moretimethatthis is theaccounttodelete.Toaccomplishtheseitems,a variable, USER_ACCOUNT_RECORD, is settotheoutcomeofa grepsearchforthe accountthroughthe/etc/ passwdfile.The-woptionallowsanexactwordmatchforthisparticularuser account:

```
USER_ACCOUNT_RECORD=$(cat/etc/passwd|grep-w$USER_ACCOUNT)
```

Theexitstatusofthe grepcommand helps here.Ifthe account record is not found, the ?variable is setto1:

```
if[$?-eq1] then
echo
echo"Account,$USER_ACCOUNT,notfound." echo
"Leaving the script..."
echo
exit
fi
```

Ifthe record was found, youstill need to verifywiththe script user that this is the correct account. Here is where allthe work to set up the functions reallypays off!

Removinganyaccountprocesses:

So far, the script has obtained and verified the correct name of the user account to be deleted. In order to remove the user account from the system, the account cannot ownanyprocesses currently running. Thus, the next step is to find and kill off those processes. This is goingto get a little complicated!

Finding the user processes is the easy part. Here the script can use the ps command and the -u option to locate any running processes owned by the account. Byredirecting the output to /dev/null, the user doesn't see any display. This is handy,
because ifthere are noprocesses, the pscommand onlyshows a header, whichcan be confusing to the script user:

```
ps-u$USER_ACCOUNT>/dev/null#Areuserprocessesrunning?
```

The ps command's exit status and a case structure are used to determine the nextstep
to take:

```
case$?in
1)#NoprocessesrunningforthisUserAccount #
echo"Therearenoprocessesforthisaccountcurrentlyrunning."
```

```
echo
;;
0)#ProcessesrunningforthisUserAccount.
#AskScriptUserifwants ustokilltheprocesses. #
echo"$USER_ACCOUNThasthefollowingprocessesrunning:"echo
ps-u$USER_ACCOUNT
#
LINE1="Wouldyoulikemetokilltheprocess(es)?[y/n]"get_ans
wer
#
[...]
esac
```

If the ps command's exit status returns a 1, there are no processes running on the system that belong to the user account. However, if the exit status returns a 0, processes owned the script does is to call the process_answer functionUnfortunately, the next item is too complicated for process_answer. Anothercase statement mustbe embedded toprocess the script user's answer. The first part of the case statement looks very similar to the process_answer function:

```
case $ANSWER in y|Y|YES|yes|Yes|yEs|yeS|YEs|yES ) # If user answers "yes",
#kill User Account processes.
[...]
;;
*)#Ifuseranswersanythingbut"yes",donotkill. echo
echo"Willnotkilltheprocess(es)"ech
o
;;
esac
```

Findingaccountfiles:

When a user account is deleted from the system, it is a good practice to archive all the files that belonged to that account. Alongwiththat practice, it is also important to remove the files or assigntheirownership to another account. Ifthe account you delete has a User ID

MonitoringDiskSpace:

One ofthe biggest problems with multi-user Linux systems is the amount of available diskspace. Insomesituations, suchas ina file-sharingserver, diskspace can fill up almost immediately just because ofone careless user.

This shell script utility helps you determine the top ten disk space consumers for designated directories. It produces a date-stamped report that allows disk space consumption trends to be monitored.

Obtainingtherequiredfunctions:

The firsttoolyouneedto use istheducommand.Thiscommanddisplaysthe disk usage for individual files and directories. The -s option lets you summarize totalsat the directory level. This comes in handy when calculating the total disk space used by an individual user. Here’s what it looks like to use the du command to summarize each user’s $HOME directory for the /home directory contents:

```
$sudodu-s /home/*
[sudo]passwordforChristine:

4204/home/Christine
56/home/Consultant
52/home/Development
4/home/NoSuchUser
96/home/Samantha
36/home/Timothy
1024/home/user1
$
```

The -s option works well for users’ $HOME directories, but what if we wanted to view diskconsumption in a system directory such as /var/log?

```
$sudodu-s /var/log/*
4/var/log/anaconda.ifcfg.log
20/var/log/anaconda.log
32   /var/log/anaconda.program.log
108   /var/log/anaconda.storage.log
40/var/log/anaconda.syslog
56/var/log/anaconda.xlog
116/var/log/anaconda.yum.log
4392/var/log/audit
4/var/log/boot.log
[...]
$
```

The listingquicklybecomes too detailed. The -S (capitalS) optionworks better for our purposes here, providing a total for each directory and subdirectory individually. This allows you to pinpoint problemareas quickly:

```
$sudodu-S/var/log/  4
/var/log/ppp
4/var/log/sssd
```

```
3020/var/log/sa
80/var/log/prelink
4/var/log/samba/old
4/var/log/samba
4/var/log/ntpstats
4/var/log/cups
4392/var/log/audit
420/var/log/gdm
4/var/log/httpd
152/var/log/ConsoleKit
2976/var/log/
$
```

Because we are interested in the directories consuming the biggest chunks of disk space,
thesortcommandisusedonthelistingproducedbydu:

```
$Sudodu-S/var/log/ |sort-rn 4392
/var/log/audit
```

Creatingthescript:

To save time and effort, the script creates a report for multiple designated directories.Variableto accomplish thiscalledCHECK_DIRECTORIESisused. For our purposes here, the variable is set tojust two directories:

```
CHECK_DIRECTORIES=" /var/log /home"
```

The script contains a for loop to performthe du command on eachdirectory listed in the variable. This technique is used to read and process values in a list. Each Time the for loop iterates through the list of values in the variable CHECK_DIRECTORIES,
ItassignstotheDIR_CHECKvariablethenextvalue inthe list: For

```
DIR_CHECK in $CHECK_DIRECTORIES
Do
[...]
Du-S$DIR_CHECK
[...]
Done
```

Toallowquickidentifyaction,adatestampisaddedtothereport’sifrename, using the date Command.Usingtheexeccommand(seeChapter15)thescriptredirectsitsoutput to the Date stamped report if le:

```
DATE=$(date'+%madly')
```

exec>disk_space_$DATE.rpt

Nowtoproducea nicelyformattedreport, thescript uses the echocommand toput in a fewreporttitles:

```
echo"TopTenDiskSpaceUsage"
echo"for$CHECK_DIRECTORIESDirectories"
```

Solet'sseewhatthisscriptlookslikeallputtogether:

```
#!/bin/bash
#
# Big Users - Find big disk space users in various directories
###############################################################
#Parameters forScript
```

ProducingScriptsforDatabase,Web,andE-Mail:

Writingdatabaseshellscripts:

- IT stores all the information you want in your shell script variables, but at the end of the script, the variables just go away. Sometimes, you'd like for your scripts to be able to store data that you can use later.
- In the old days, storing and retrieve data from a shell scriptrequired creating a file, reading data from the file, parsing the data, and then saving the data back into the if le.

Searchingfordata inthe file meansreadingeveryrecord inthe fileto look for your data. Nowadays withdatabases beingallthe rage, it's a snap to interface your shell scripts with professional-quality open-source databases. Currently, themost popular open-source database used in the Linux world is Myself. Its popularity has grown as a part of the Linux-Apache-Myself-PHP (LAMP) server EnvironmentwhichmanyInternetwebservers use for hostingonline stores, blogs, and applications.

Connectingtotheserver:

The myself client program allows you to connect to any myself database server anywhere onthe network, usingany user account and password. Bydefault, ifyouenterthe meprogramonacommand linewithoutanyparameters, itattempts to connect to a myself

Serverrunningonthesame Linuxsystem, usingthe Linuxloginusername. Type 'help;'or '\h' for help. Type '\c'to clear the current input statement.

Myself>

The -p parameter tells theI program to prompt for a password to use with the user account to log in. Enter the password that you assigned to the root user account, either during the installation process, or using the mysqladmin utility. After you're logged in to the server, you can start entering commands.

TheIcommand

TheIprogramusestwodifferenttypesofcommands:

■ Specialmysqlcommands

■ StandardSQLstatements

The mysql program uses its own set of commands that let you easily control the environment and retrieve information about the MySQL server. The mysql commands use either a full name (suchas status) or a shortcut (suchas \s).

You can use either the full command or the shortcut command directly from the mysql
commandprompt:

```
mysql>\s
__________-
mysqlVer14.14Distrib5.5.38, fordebian-linux-gnu(i686)usingreadline6.3
Connection id:
```

The mysqlprogramimplementsallthestandardStructuredQueryLanguage (SQL) commands supported by the MySQL server. One uncommon SQL command that the mysql program implements is the SHOW command. Using this command, you can extract information about the MySQL server, such as the databases and tables created:

```
mysql>SHOWDATABASES;
+_______________+
|Database|
+_______________+
|information_schema|
|mysql|
+_______________+
2rowsinset(0.04sec)
mysql> USE mysql;
Databasechanged
mysql>SHOWTABLES;
+____________________+
|Tables_in_mysql|
+____________________+
|columns_priv|
|db|
|func|
|help_category|
|help_keyword|
|help_relation|
|help_topic|
```

Creatingadatabase:

The MySQL server organizes data into databases. A database usually holds the data for a single application, separating it from other applications that use the database server.Creating a separate database for each shell script application helps eliminateconfusionanddata mix-ups.Here’stheSQLstatementrequiredtocreatea new database:

```
	CREATEDATABASEname;
```

That’s pretty simple. Of course, you must have the proper privileges to create new databases on the MySQL server. The easiest way to do that is to log in as the root user account:

```
$mysql-uroot–p
Enter password:
WelcometotheMySQLmonitor.Commandsendwith;or \g. Your
MySQL connection id is 42
Serverversion:5.5.38-0ubuntu0.14.04.1
```

The test.*entrydefines the database and tables to whichthe privileges apply. This is specified in the following format:

```
database.table
```

As youcansee fromthis example, you’re allowed to use wildcard characters when specifying the database and tables. This format applies the specificed privileges to all the tablescontained in the database named test.Finally, you specify the user account(s) to which the privileges apply. The neat thing aboutthe grant command is that ifthe user account doesn’t exist, it creates it

Youcantestthenewuseraccountdirectlyfromthemysqlprogram:

```
$mysqlmytest-utest–p Enter
password:
WelcometotheMySQLmonitor.Commandsendwith;or \g. Your
MySQL connection id is 42
```

Creatingatable

The MySQL server is considered a relational database. In arelational database, data is organized by data fi elds, records, and tables. A data fi eld is a single piece of information, such as an employee’s last name or a salary. A record isa collection of related data fi elds, such asthe employee ID number, last name, fi rst name, address, and salary. Each record indicatesone set of the data fields.ss table contains all the records that hold the related data. Thus, you’ll have a table called Employees that holds the records for each employee.

To create a new table in the database, you need to use the CREATE TABLE SQL command:

```
$mysqlmytest-uroot-p Enter
password:
```

```
mysql>CREATETABLEemployees(
->empidintnotnull,
->lastnamevarchar(30),
->firstnamevarchar(30),
->salaryfloat,
->primarykey(empid));
QueryOK,0rowsaffected(0.14sec)
mysql>
```

First, notice that to create the new table, we needed to log in to MySQL using the root user account because the test user doesn't have privileges to create anew table. Next, notice that we specified the mytest database on the mysql program command line. If we hadn'the done that, we would need to use the USE SQL command to connect to the test databaseMySQL Data Types

DataTypeDescription
char --A fixed-length string value
varchar--Avariable-lengthstringvalue
int ---An integer value
float--- A floating-point value
boolean---ABooleantrue/falsevalue
date--AdatevalueinYYYY-MM-DDformat
time--- A time value in HH:mm:ss format
timestamp-- A date and time value together
text--- A long string value
BLOB--Alargebinaryvalue,suchasanimageorvideoclip

empid data field also specifies a data constraint. A data constraint restricts what type of data you can enter to create a valid record. The not nulldata constraint indicates thatevery record must have an empid value specified.

Finally, the primary key defines a data field that uniquely identifies each individualrecord. This meansthateachdata record must haveauniqueempid value in the table.

Aftercreatingthe new table, youcanuse the appropriate commandtoensure that it's created.

Inmysql,it'stheshowtablescommand: mysql>

```
show tables;
+___________-+
|Tables_in_test |
+___________-+
|employees|
+___________-+
1rowinset(0.00sec)
```

Insertinganddeletingdata:

Not surprisingly, you use the INSERT SQL command to insert new data records into the table. Each INSERT command must specify the data field values for the MySQL server to accept the record.

Here's the formatoftheINSERTSQLcommand:

```
INSERT INTO table VALUES (...)
```

Thevaluesareinacomma-separatedlistofthedatavaluesforeachdata field:

```
$mysqlmytest-utest-p Enter
password:
mysql>INSERTINTOemployeesVALUES(1,'Blum','Rich',25000.00); Query
OK, 1 row affected (0.35 sec)
```

The exampleusesthe–u commandline prompt tolog in asthe test user account that wascreated in MySQL.

The INSERT command pushes the data values you specify into the data fields inthe table.If youattempt to add another record that duplicates the empid data field value, you get an error message:

```
mysql>INSERTINTOemployeesVALUES(1,'Blum','Barbara',45000.00);
ERROR 1062 (23000): Duplicate entry '1' for key 1
```

Queryingdata:

After you have all your data in your database, it's time to start running reports to extract information.The workhorse for all your querying is the SQL SELECT command. The SELECT command isextremely versatile, but with versatility comes complexity.

Here'sthebasic formatofa *SELECT*statement:

```
SELECTdatafieldsFROMtable
```

The data fields parameter is a comma-separated list of the data field names you want the query to return. If you want to receive all the data field values, you canuseanasteriskas awildcardcharacter. You mustalsospecifythespecifictable you want the query to search. To get meaningful results,youmustmatchyourquerydatafieldswiththepropertable.

By default, the SELECT command returns all the data records in the specified table:

```
mysql>SELECT*FROMemployees;
```

Usingthedatabaseinyourscripts:

Now that you have a workingdatabase going, it's finallytime to turnour attention back to the shell scripting world. This section describes what you need to do to interact with your databases using shell scripts.

Loggingintotheserver:

If you’ve created a special user account in MySQL for your shell scripts, you need to use it.To log in with the mysql command. There are a couple ways to do that. One method is to include the password on the command line using the -p parameter:

```
mysqlmytest-utest–ptest
```

This, however, is not a good idea. Anyone who has access to yourscript willknow the user account and password for your database.

Tosetthedefaultpasswordinthisfile,justcreatethe following:

```
$ cat .my.cnf
[client]
password=test
$chmod400.my.cnf
$
```

The chmod commandisused to restrict the .my.cnf fi le so only you can view it.You can test this now from the command line:

```
$ mysqlmytest-utest
Readingtable informationforcompletionoftableandcolumnnames You
can turn off this feature to get a quicker startup with -A Welcome to
the MySQL monitor. Commands end with ; or \g.
YourMySQLconnectionidis 44
Serverversion:5.5.38-0ubuntu0.14.04.1(Ubuntu)
Copyright(c)2000,2014,Oracleand/oritsaffiliates.Allrightsreserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks oftheir respectiveowners.
Type'help;'or'\h'forhelp.Type '\c'toclearthecurrent inputstatement. mysql>
```

Perfect!Now youdon’t have to include the password onthe command line in your shellscripts.

Sendingcommandstotheserver:

After establishing the connection to the server, you’ll want to send commands tointeract with your database. There are two methods to do this:

- Sendasinglecommandandexit.
- Sendmultiplecommands.

To sendasinglecommand,youmustincludethecommandaspartof themysql command line. For the mysqlcommand, youdo this usingthe -e parameter:

```
$catmtest1
#!/bin/bash
#sendacommandtotheMySQLserver
MYSQL=$(which mysql)
```

```
$MYSQLmytest-utest-e'select*fromemployees'
```

Thisis anexampleofdefininganendoffilestring, withdatainit:

```
$catmtest2
#!/bin/bash
#sendingmultiplecommandstoMySQL
MYSQL=$(which mysql)
$MYSQLmytest-utest<<EOF show
tables;
select*fromemployeeswheresalary>40000; EOF
$ ./mtest2
Tables_in_test
employees
empidlastnamefirstnamesalary 2
Blum Barbara 45000
4BlumJessica52340
$
```

The shell redirects everything with the EOF delimiters to the mysql command, which executes the lines as if you typed them yourself at the prompt. Using this method, youcan send as many commands to the MySQL server as you need. You’ll notice, however,that there’s no separation between the output from each command. In the next section,
“Formattingdata,”you’llsee howto fixthisproblem

Formatting data

The standard output from the mysql command doesn’t lend itself to data retrieval. If youneed to actually do something with the data you retrieve,you need to do some fancy data manipulation. This section describes some of the tricks you can use to help extract datafrom your database reports.The first stepin trying to capture database data is to redirect the output from the mysql and psql commands in an environment variable. This allows you to use the output information in other commands. Here’s an example:

```
$catmtest4
#!/bin/bash
#redirectingSQLoutputtoavariable
MYSQL=$(which mysql)
dbs=$($MYSQLmytest-utest-Bse'showdatabases') for
db in $dbs
do
echo$db
done
```

```
$ ./mtest4
information_schema
test
$
```

Usingthe Web

Oftenwhenyouthinkofshellscriptprogramming, the lastthing youthink ofis the Internet. The command line world often seems foreign to thefancy, graphical world of theInternet. There are, however, several different utilities you can easily use in your shell scripts to gain access to data content on the web, as well as on other network devices.

Almost as old as the Internet itself, the Lynx program was created in 1992 by students at the University of Kansas as a text-based browser. Because it’s text-based, the Lynx program allows you to browse websites directly from a terminal session, replacing the fancy graphics on web pages with HTML text tags. This allows you to surf the Internet fromjust about any type of Linux terminal.