Shell Programming

Activity Guide

D85153GC10
Edition 1.0
May 2014
D86728

ORACLE

Author

Pardeep Kumar Sharma

Technical Contributors and Reviewers

Joel Goodman, Harald Van Breederode, Vijetha Malkai, Pranamya Jain, Uma Sannasi

This book was published using: Oracle Tutor

Table of Contents

Practices for Lesson 1: Introduction

Chapter 1

Practices for Lesson 1: Introduction

Practices Overview

This practice provides an introduction to your course assignments and the infrastructure that you will use for performing the practices.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
	Introduction
	UNIX Shells
	Shell Scripting
	Shell Environment
	Pattern Matching
	The `sed` Editor
	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practices Infrastructure

This section presents the architectural overview of the infrastructure required for the practices. Your practice environment is based on the Oracle VM VirtualBox virtualization software. The VirtualBox is a cross-platform virtualization application. It provides multiple virtual machines (VMs) that are configured on a private internal network (`192.168.10`). Each VM can communicate with other VMs on the same private network (see Figure 1). Internet access is not configured for these VMs.

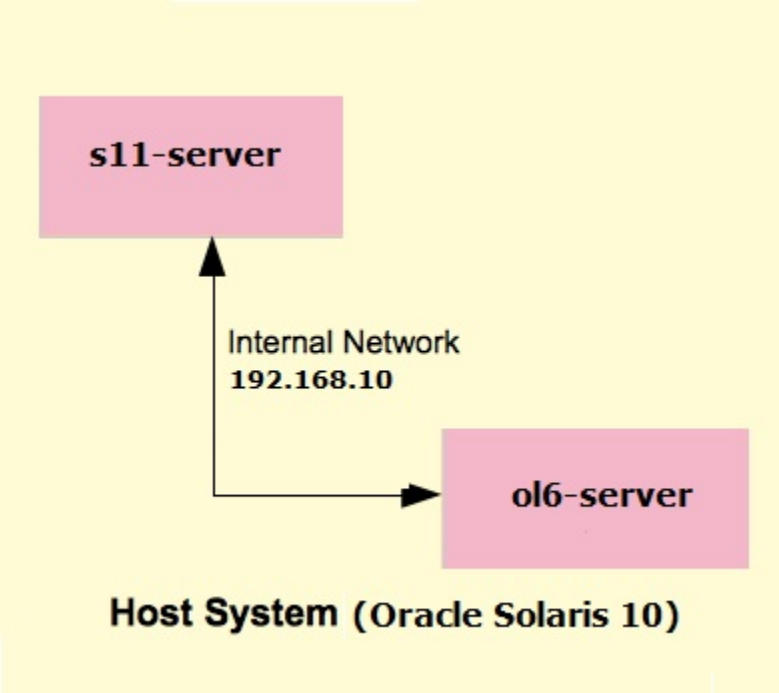


Figure 1: Virtual Pod Network Schema

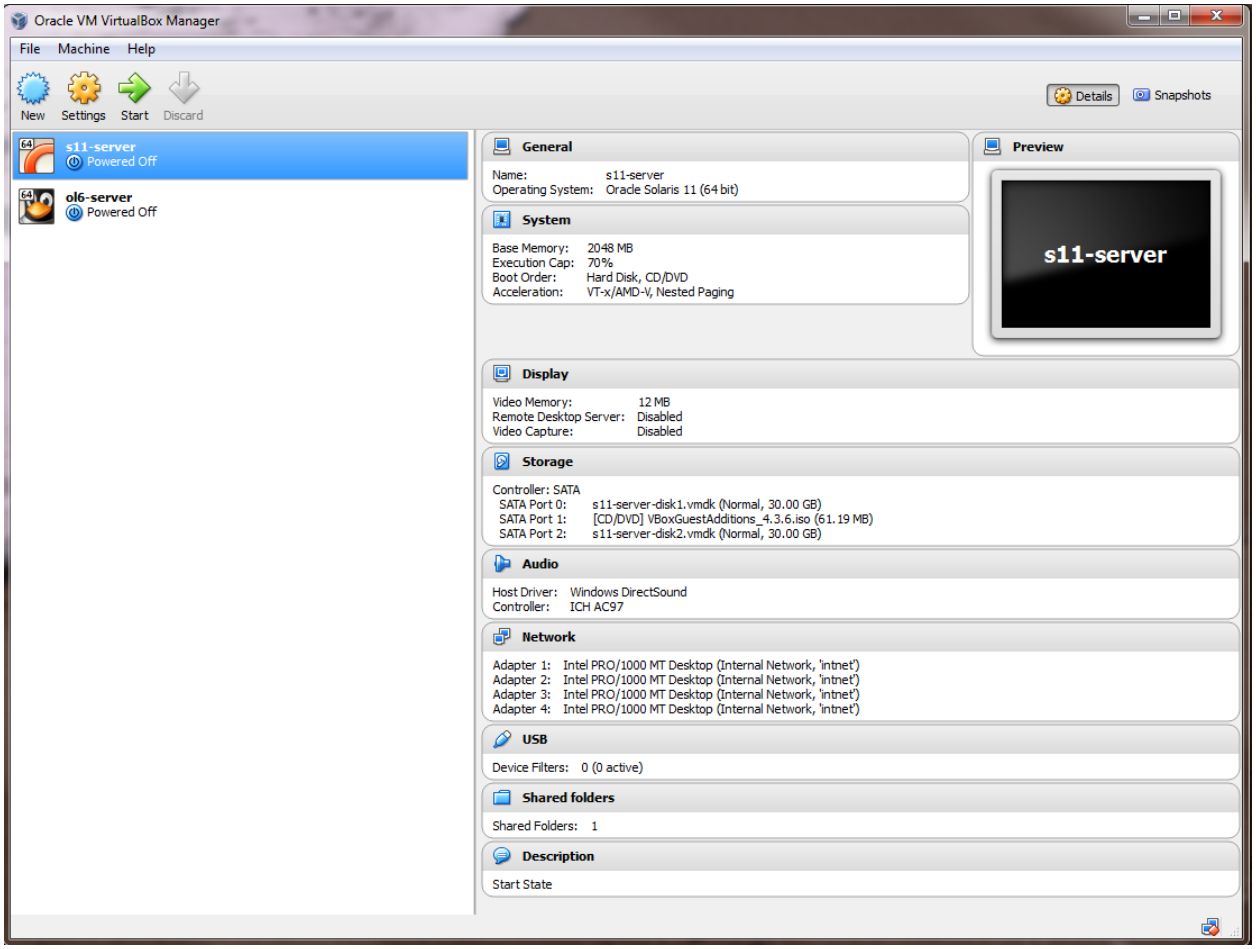


Figure 2: Configured Oracle VirtualBox VMs

Figure 2 shows the configured virtual machines. The VirtualBox environment consists of the following VMs:

Name of the VM	Description
s11-server	This VM provides an Oracle Solaris 11.1 guest OS image where the student performs practices related to Oracle Solaris 11 OS. This is the primary VM.
ol6-server	This VM provides an Oracle Linux 6.5 guest OS image, which you can use as an alternative to perform the practice tasks.

On the VMs, the `/opt/ora` directory contains labs files that you will use to perform practices. The following table describes the contents of the `/opt/ora` directory:

Resource Name	Location	Description
Top level directory	`/opt/ora`	Contains various course files
Lab files	`/opt/ora/labs`	Contains files required to perform the practices

User Credentials

VMs	Credentials
s11-server	• Username: **oracle** • Password: **oracle1** **Note**: As `oracle` user, use `su` to switch to the primary administrator (`root`) role. The password is `oracle1`. The `root` is configured as a role by default in Oracle Solaris 11. The first username created on the system during the installation is the initial privileged user who can assume the primary administrator role. This can be verified in the `/etc/user_attr` file.
ol6-server	• Username: **oracle** • Password: **oracle1** For administrative access, switch to the `root` user by using the `su` command as and when instructed by the instructor. • Username: **root** • Password: **oracle1**

Practice 1-1: Getting Familiar with Your Practice Environment

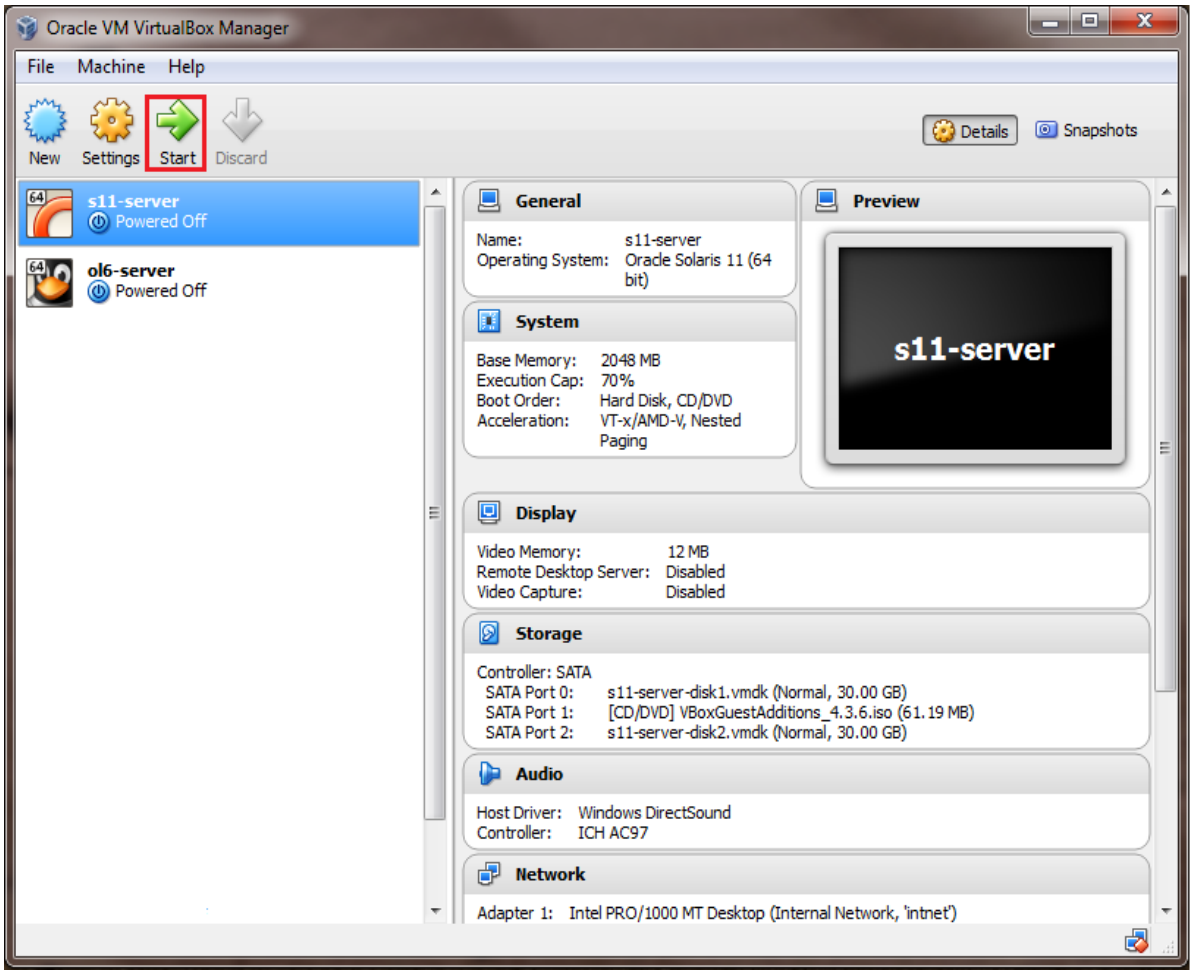
Overview

In this practice, you will get familiarized with the lab environment that you will use to perform practices throughout this course.

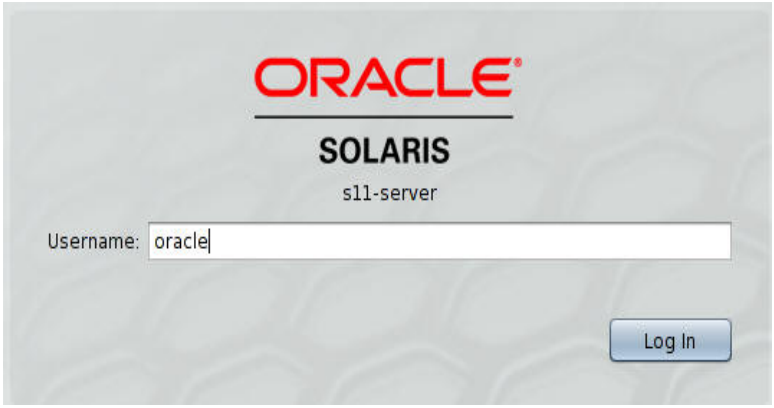
Tasks:

1. On your host system, start the Oracle VM Virtual Box Manager by double-clicking its icon on your desktop.

2. In the Oracle VM VirtualBox Manager window, double-click the s11-server VM to start it. Alternatively, you can select the s11-server VM and click the **Start** button.



3. After the s11-server VM is powered ON, log in with the username `oracle` and password `oracle1`.



4. Similarly, start the ol6-server VM and log in to the virtual machine with the username `oracle` and password `oracle1`.

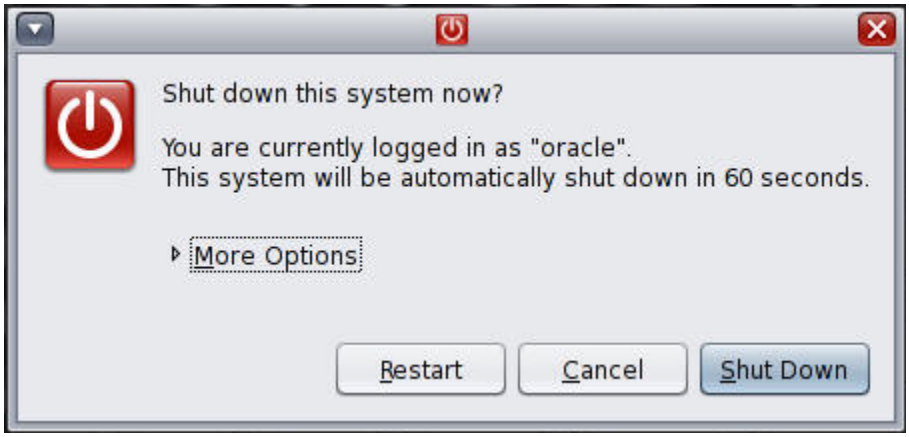
5. After successfully logging in to the VMs, right-click the desktop and select the Open Terminal option in the Oracle Solaris VM or the Open in Terminal option in the Oracle Linux VM.

   Note: In Oracle Linux VM, the default directory will be `/home/oracle/Desktop`. You would have to switch to the `/home/oracle` directory in order to perform practice tasks. Your instructor will guide you through this process as you proceed to later practices.

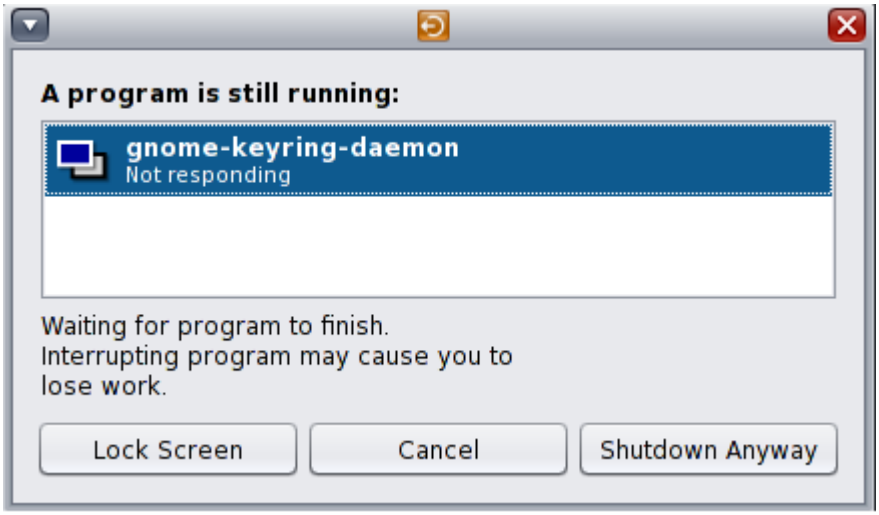
6. In the terminal window, enter the `su` command to assume primary administrator privileges. Enter `oracle1` when prompted for the password.

```
oracle@s11-server:~$ su
Password: oracle1
root@s11-server:~#
```

7. Close a VM by selecting the **Shut Down** option from the **System** menu. This will launch a dialog box as shown below:



Click the **Shut Down** button. This may launch another dialog box. Select the option **Shutdown Anyway**.



8. The above action will initiate the Shut Down procedure.

9. For Oracle Linux VM, ol6-server, select the **Shut Down** option from the **System** menu. This will launch a dialog box as shown below.

10. Click the **Shut Down** button. This will initiate the shut down procedure.

11. Verify that no VMs are running at this time, by viewing the status of the VMs in the Oracle VM VirtualBox Manager window. The status of the VMs should read Powered Off.

Special Instructions

- The practice tasks in this course are written for Oracle Solaris 11.1 environment (s11-server), but can also be performed on Oracle Linux 6.5 environment (ol6-server). Therefore, it is recommended that one system be used at a time, preferably Oracle Solaris (s11-server).
- Ignore any warning messages while you launch the VM.
- You will perform the exercises in the `/home/oracle` directory. For Oracle Linux, change the directory from `Desktop` to the `/home/oracle` directory. Your instructor will help you with changing the directory.
- You will perform most of the tasks as `oracle` user, except where mentioned otherwise.
- You will save your work in `/opt/ora/labs/lab*/solutions` directory (where `lab*` indicates `lab1`, `lab2` ... `lab13`)
- Use the `vi` editor with the `-n` option to avoid creation of swap files while creating or editing a new file.
- The command output may vary from system to system.
- Shut down the VMs when not required. This releases system resources for the primary VM.
- Follow the instructions in the practices with diligence for a smooth learning experience.

Practices for Lesson 2: UNIX Shells

Chapter 2

Practices for Lesson 2: UNIX Shells

Practices Overview

In this practice, you will test your knowledge of various UNIX shells and their functions.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
	UNIX Shells
	Shell Scripting
	Shell Environment
	Pattern Matching
	The `sed` Editor
	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 2-1: Reviewing UNIX Shells

Overview

In this practice, you will test your knowledge of different UNIX shells supported in Oracle Solaris and Oracle Linux operating systems.

Tasks

1. What are the functions of UNIX shells?

   They provide:

   - *An interactive user environment*
   - *A command-line interpretation*
   - *A programming language*
   - *Input-output redirection*
   - *Variable and filename substitution*

2. List and describe the standard shells found in the Oracle Solaris 11 OS.

   - ***Bash shell:*** *Bash is a UNIX shell, written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell (*`sh`*). It has been distributed widely as the shell for the GNU operating system and as the default shell on Oracle Solaris 11, Oracle Linux 6u5, and Mac OS X. This is the most popular shell, described as Bourne-again shell as an acronym. This shell has all the useful features of Bourne shell (*`sh`*), C shell (*`csh`*), and Korn shell (*`ksh`*) included.*
   - ***Bourne shell:*** *The original UNIX shell, written at AT&T Bell Labs, is the preferred shell for scripting because of its compactness and speed. However, it lacks features often expected by users for frequent interactive use.*
   - ***C shell:*** *Written at the University of California at Berkeley. The programming constructs were borrowed from the C language, thus the name of the shell. It includes the history and alias capabilities frequently used by users and programmers.*
   - ***Korn shell:*** *Written at AT&T Bell Labs. This shell uses the programming constructs of the Bourne shell and the history and alias capabilities of the C shell, Additionally, it provides features such as integer arithmetic, arrays, and string manipulation.*

Practices for Lesson 3: Shell Scripting

Chapter 3

Practices for Lesson 3: Shell Scripting

Practices Overview

In these practices, you will perform the following tasks:

- Review shell script.
- Write a simple shell script and debug it.

The following checklist shows your progress through the practices

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
	Shell Scripting
	Shell Environment
	Pattern Matching
	The `sed` Editor
	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 3-1: Reviewing Shell Scripts

Overview

In this practice, you will review the components of a shell script and list some shell commands, which you will use in Practice 3-2 to create a simple shell script.

Assumptions

The command output may vary from system to system.

Tasks

1. Name the main components of a shell script.

   Answer: The main components are:

   - *Comments*
   - *Information displays (output)*
   - *Conditional testing*
   - *Loops*
   - *Arithmetic*
   - *String manipulation*
   - *Variable manipulation and testing*
   - *Argument and option handling*

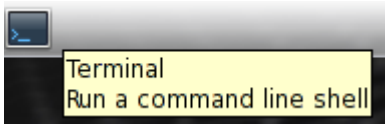
Note: Most of these components will be discussed in later lessons.

2. On plain paper or in a text editor in your system, write an ordered list of shell commands to yield the following:

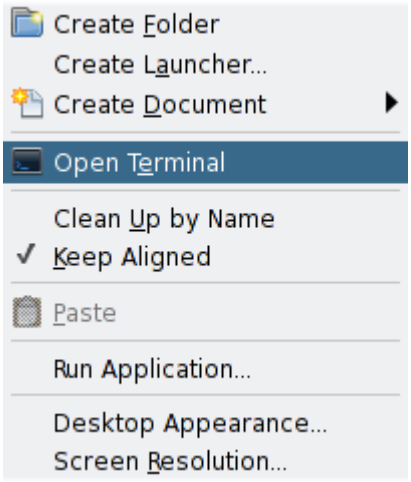
   - The system name (**Hint:** Use `uname`.)
   - The hardware platform (for example, i86pc) (**Hint:** Use `uname`.)
   - Processor-specific information including clock speed (**Hint:** Use `psrinfo` for Oracle Solaris 11 system and `lscpu` for Oracle Linux 6u5 system.)
   - The total number of processes running on the system (**Hint:** Use `ps` and `wc`.)

   Note: You can log in to s11-server VM or ol6-server vm as an `oracle` user with password `oracle1` to execute the respective commands. Refer to Practice 1 to understand how to start and log in to the VMs.

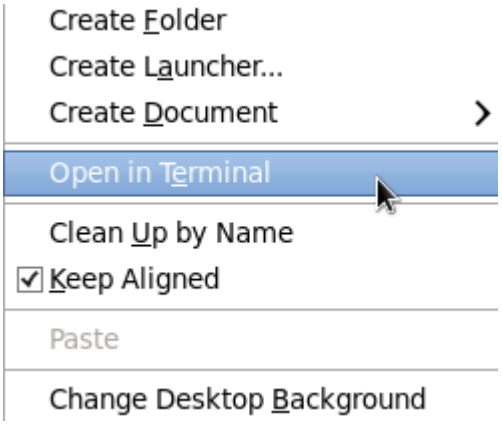
   a. For Oracle Solaris 11: Open a terminal window by clicking the terminal icon as shown below:



or right-clicking and selecting **Open Terminal**.



b. For Oracle Linux OS: Open a terminal window by right-clicking anywhere on desktop and selecting **Open in Terminal**.



The commands to produce the output should resemble the following:

Individual commands with output in Oracle Solaris 11.1 OS:

```
$ uname -n
s11-server

$ uname -i
i86pc
$ psrinfo -pv
The physical processor has 1 virtual processor (0)
  x86 (GenuineIntel 306A9 family 6 model 58 step 9 clock 2568
MHz)
     Intel(r) Core(tm) i5-3320M CPU @ 2.60GHz

$ ps -ef | tail +2 | wc -l
     104
```

Individual commands with output in Oracle Linux 6.5 OS:

```
$ uname -n
ol6-server

$ uname -i
x86_64

$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 58
Stepping:              9
CPU MHz:               2567.858
BogoMIPS:              5135.71
L1d cache:             32K
L1d cache:             32K
L2d cache:             6144K
NUMA node0 CPU(s):     0

$ ps aux | wc -l
141
```

Note: In Practice 3-2, you will enter these commands into a script using an editor, and then modify the script and debug it.

Practice 3-2: Writing and Debugging Shell Scripts

Overview

In this practice, you will create and execute a simple shell script. The script will be modified with the insertion of deliberate errors to showcase benefits of debugging options such as `–x`, `-v`, and `-f`.

Note: The scripts developed and used in this course are available under the `/opt/ora/labs` directory in your respective system. Switch to the `/opt/ora/labs/lab3/solutions` directory before beginning this practice.

Tasks

1. Close all the terminal windows and open a new terminal window.
2. Before beginning the practice, change to the `/opt/ora/labs/lab3/solutions` directory.

```
$ cd /opt/ora/labs/lab3/solutions
$ pwd
/opt/ora/labs/lab3/solutions
```

3. By using the `vi` editor, create a script named `echoscript.sh` and enter the following:

   Note: You will use the same set of commands that you identified in the Practice 3-1.

   Based upon the topics discussed in the current lesson, make sure that you:

   - *Include a first line that calls the Bash shell as the interpreter*
   - *Add a comment to state the script's purpose*
   - *Add any other comments that you deem necessary*

```
$ vi echoscript.sh
#!/bin/bash

# This script will print useful information about the current
# machine. It will print the machine name, model type, specific
# processor information, and the current number of processes
# running.

echo
echo "The name of the machine is \c"
uname -n # uname -n gives the hostname

echo
echo "The machine platform type is \c"
uname -i # uname -i gives the hardware platform

echo
echo "Specific processor information for this machine is:"
psrinfo -v
```

```
echo
echo "Currently the total number of processes "
echo " running on this machine is \c"
ps -ef | tail +2 | wc -l # wc -l gives number of lines

:wq
```

Note: For Oracle Linux, replace commands, i) `ps -ef | tail +2 | wc -l` with `ps aux | wc –l` and ii) `psrinfo –pv` with `lscpu`.

4. Change the permissions for `echoscript.sh` so that you can execute the script as a command.

```
$ chmod 777 echoscript.sh
```

5. Execute the `echoscript.sh` script.

```
$ ./echoscript.sh
The name of the machine is s11-server

The machine platform type is i86pc

Specific processor information for this machine is:
Status of virtual processor 0 as of: 03/17/2014 12:17:45
  on-line since 03/14/2014 14:54:32.
  The i386 processor operates at 2568 MHz,
    and has an i387 compatible floating point processor.

Currently the total number of processes
  running on this machine is      105
```

Observation: In the above steps, you identified the commands to be used in the shell script followed by the creation of the script and its execution.

6. In the next few steps, you will use an existing script to perform debugging operations in it. To start with, copy the `debugscript.sh` script from `lab3` directory to the `debug1.sh` file in the `solutions` directory and read `debug1.sh`.

   Note that the different debug options are entered in the comment statements. This makes it easier to execute the script and test the option that gives you the most useful information.

```
$ cd..
$ pwd
/opt/ora/labs/lab3/
$ cp debugscript.sh solutions/debug1.sh

$ cd solutions
$ cat debug1.sh
```

```
#!/bin/bash

# To make it easier to test debugging options, uncomment one of
the following.
# set -v
# set -x
# set -f

echo "Your terminal type is set to: $TERM"
echo

echo "Your login name is: $LOGNAME"
echo

echo "Now we will list the contents of the /etc/security
directory"
ls /etc/security
echo
```

7. Change file permission for the `debug1.sh` script. Execute the `debug1.sh` script.

```
$ chmod 777 debug1.sh
$ ./debug1.sh
Your terminal type is set to: xterm

Your login name is: oracle

Now we will list the contents of the /etc/security directory
audit_class    auth_attr.d    exec_attr      pam_policy
prof_attr.d    audit_event    crypt.conf     exec_attr.d
policy.conf    tcsd.conf      audit_warn     dev
extra_privs    priv_names     auth_attr      device_policy
kmfpolicy.xml  prof_attr

Does it appear to execute correctly?
```

Note: Yes, the above script executed correctly with the expected output.

8. Now, add a bug deliberately to this script by misspelling one of the `echo` statements near the middle of the script. Change the statement in `debug1.sh` that prints out the value of the `logname` variable instead of `LOGNAME`.

```
echo "Your login name is: $LOGNAME"
```

to:

```
echo "Your login name is: $logname"
```

```
$ vi debug1.sh

#!/bin/bash

# To make it easier to test debugging options, uncomment one of
the following.
# set -v
# set -x
# set -f

echo "Your terminal type is set to: $TERM"
echo

echo "Your login name is: $logname"
echo

echo "Now we will list the contents of the /etc/security
directory"
ls /etc/security
echo
~
~
~
:wq
```

9. Re-execute `debug1.sh`, and write down the output given for the modified line.

```
$ ./debug1.sh
Your terminal type is set to: xterm

Your login name is:

Now we will list the contents of the /etc/security directory
audit_class     crypt.conf      extra_privs     prof_attr
audit_event     dev             kmfpolicy.xml   prof_attr.d
audit_warn      device_policy   pam_policy      tcsd.conf
auth_attr       exec_attr       policy.conf
auth_attr.d     exec_attr.d     priv_names
```

The output for the given line is changed to:

```
Your login name is:
```

Note: The above output fails to recognize the login name.

10. Now, to identify the error/bug in the script, you will turn ON each of the three debug options (one at a time) and re-execute the `debug1.sh` script. Record whether each option gives you any help in determining the problem.

    Hint: Debugging options are `set -v`, `set -x`, `set -f`. Uncomment these options, one at a time and view the output.

```
$ vi debug1.sh

#!/bin/bash

# To make it easier to test debugging options, uncomment one of
the following.
set -v
# set -x
# set -f

echo "Your terminal type is set to: $TERM"
echo

echo "Your login name is: $logname"
echo

echo "Now we will list the contents of the /etc/security
directory"
ls /etc/security
echo
```

The following is the output obtained with each of the debugging options turned on separately.

```
set -v
Output:
echo "Your login name is: $logname"
Your login name is:
You can see the misspelled variable in the echo statement.

set -x
Output:
+ echo Your login name is:
Your login name is:
```

```
+ echo
```

You cannot see the name of the variable because it is was interpreted and replaced by its value (the default value is a null string).

```
set -f
```

Output:

```
Your login name is:
```

You cannot see the name of the variable because it is was interpreted and replaced by its value (the default value is a null string).

Observation: Which debug options gave you information to help point to the variable name problem? *The* `-v` *option was best because it showed the misspelling of the variable as part of the debug output.*

11. Copy the `debug1.sh` script to `debug2.sh`.

```
$ cp debug1.sh debug2.sh
```

12. Update the permissions for the `debug2.sh` file and then modify the script to make necessary corrections and comment the debugging options again:

```
$ chmod 777 debugg2.sh
$ vi debug2.sh

#!/bin/bash

# To make it easier to test debugging options, uncomment one of
the following.
#set -v
#set -x
#set -f

echo "Your terminal type is set to: $TERM"
echo

echo "Your login name is: $LOGNAME"
echo

echo "Now we will list the contents of the /etc/security
directory"
ls /etc/security
echo
```

13. Execute the script.

```
$ ./debug2.sh
```

```
Your terminal type is set to: xterm

Your login name is: oracle

Now we will list the contents of the /etc/security directory
audit_class     auth_attr.d     exec_attr       pam_policy
prof_attr.d
audit_event     crypt.conf      exec_attr.d     policy.conf
tcsd.conf
audit_warn      dev             extra_privs     priv_names
auth_attr       device_policy   kmfpolicy.xml   prof_attr
```

Observation: Note that the changes suggested by debugging options ensured that the script provides the correct output.

14. In this step, you will create a template shell script that contains basic information, which should be included in each script that you write.

    Note: The template should provide documentation, in the form of comments, for each script you write in the future. Later when creating a new script, copy the template, fill in the information, and proceed to enter the statements into the script. The template should include the following:

    - The shell interpreter
    - The name of the script
    - The author of the script
    - The date when the script was written
    - A brief description of the purpose of the script

```
$ cat template.sh

#!/bin/bash
# Script name: template
# Author: Student name
# Date written: xx/xx/xx
#
# Purpose: This script serves as a documentation template for
future scripts
#
```

Note: You will be using the `template.sh` template for writing scripts later.

This completes your practice.

Summary: In this practice, you reviewed the components of a basic shell script, while creating, modifying, and debugging a simple shell script.

Practices for Lesson 4: Shell Environment

Chapter 4

Practices for Lesson 4: Shell Environment

Practices Overview

In this practice, you will create and manipulate shell variables and environment variables in a shell environment. The following are the tasks that you will perform:

- Use shell and environment variables
- Configure user profiles

Assumptions

The default shell in these practices is the bash shell. The output between Oracle Solaris 11.1 and Oracle Linux 6.5 may vary slightly.

Note: The default shell in both Oracle Solaris 11.x and Oracle Linux 6.x operating systems is bash shell.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
	Shell Environment
	Pattern Matching
	The `sed` Editor
	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 4-1: Using the Shell and Environment Variables

Overview

In this practice, you will create, manipulate, and identify local and environment shell variables.

Assumptions

The output may vary on Oracle Solaris 11.1 and Oracle Linux 6.5 operating systems .

Tasks

1. Close all the terminal windows, if required, and open a new terminal window.

2. Before beginning the practice, change to the `/opt/ora/labs/lab4/solutions` directory.

```
$ cd /opt/ora/labs/lab4/solutions
```

Note: This is where you will save your work.

3. Create three variables and display their values:
   a. Create the `var1` shell variable and initialize it to `twenty`.
   b. Create the `var2` shell variable and initialize it to `21`.
   c. Create the `var3` shell variable and initialize it to `temp`.
   d. Display the values of all three variables.

```
$ var1=twenty
$ var2=21
$ var3=temp
$ echo $var1 $var2 $var3
twenty 21 temp
```

4. Display the list of local shell variables using the `set` command and determine whether `var1`, `var2`, or `var3` is in the listing.

```
$ set | grep var
BASHOPTS=cmdhist:expand_aliases:extquote:force_fignore:hostcompl
ete:interactive_comments:progcomp:promptvars:sourcepath:xpg_echo
GNOME_KEYRING_CONTROL=/var/tmp/keyring-J15C3X
MAIL=/var/mail/oracle
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
var1=twenty
var2=21
var3=temp
```

Observation: `var1`, `var2`, and `var3` are listed as local shell variables.

5. Display the list of environmental variables by using the `env` command, and determine whether `var1`, `var2`, or `var3` is in the listing.

```
$ env | grep var
```

```
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
GNOME_KEYRING_CONTROL=/var/tmp/keyring-J15C3X
MAIL=/var/mail/oracle
```

Observation: `var1`, `var2`, and `var3` are not listed as environment or global variables.

6. Use the `export` command to make `var1` and `var3` environmental variables and then display the list of environment variables.

```
$ export var1 var3
$ env | grep var
var1=twenty
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
var3=temp
GNOME_KEYRING_CONTROL=/var/tmp/keyring-J15C3X
MAIL=/var/mail/oracle
```

Observation: `var1` and `var3` are now listed as environment variables.

7. Remove the value of the `var3` variable using the `unset` command and then display the list of local shell variables to determine if `var1`, `var2`, or `var3` is in the listing.

```
$ unset var3
$ echo $var3

$ set | grep var
BASHOPTS=cmdhist:expand_aliases:extquote:force_fignore:hostcompl
ete:interactive_comments:progcomp:promptvars:sourcepath:xpg_echo
GNOME_KEYRING_CONTROL=/var/tmp/keyring-J15C3X
MAIL=/var/mail/oracle
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
var1=twenty
var2=21
```

Observation: `var3` is removed from the list, whereas `var1` and `var2` still exist.

8. Display the list of environmental variables and determine whether `var1`, `var2`, or `var3` is in the listing.

```
$ env | grep var
var1=twenty
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
GNOME_KEYRING_CONTROL=/var/tmp/keyring-J15C3X
MAIL=/var/mail/oracle
```

Observation: `var1` is listed as environment variable, as exported earlier, whereas `var3` is removed.

9. Open a new terminal window by executing the `gnome-terminal` command. In this window, display the list of local shell variables, and determine whether either `var1` or `var2` is in the listing.

Terminal 1

```
$ gnome-terminal

$ set | grep var
BASHOPTS=cmdhist:expand_aliases:extquote:force_fignore:hostcompl
ete:interactive_comments:progcomp:promptvars:sourcepath:xpg_echo
GNOME_KEYRING_CONTROL=/var/tmp/keyring-J15C3X
MAIL=/var/mail/oracle
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
var1=twenty
var2=21
```

Terminal 2 (new terminal)

10. Display the list of local variables and determine whether either `var1` or `var2` is in the listing.

```
$ set | grep var
BASHOPTS=cmdhist:expand_aliases:extquote:force_fignore:hostcompl
ete:interactive_comments:progcomp:promptvars:sourcepath:xpg_echo
GNOME_KEYRING_CONTROL=/var/tmp/keyring-J15C3X
MAIL=/var/mail/oracle
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
var1=twenty
```

Observation: `var2` is not available in the new terminal, because it is exclusive to the shell in the terminal1.

11. Display the list of local variables and determine whether either `var1` or `var2` is in the listing. Close the terminal session by using the `exit` command.

```
$ env | grep var
var1=twenty
ORBIT_SOCKETDIR=/var/tmp/orbit-oracle
GNOME_KEYRING_CONTROL=/var/tmp/keyring-WZNiTz
MAIL=/var/mail/oracle
$ exit
```

Observation: The variable exported in Terminal 1 is still available in Terminal 2. This means that variables declared once in a terminal are available to the shell in a new terminal.

Close all the terminals.

Practice 4-2: Configuring User Profiles

Overview

In this practice, you will edit the system-wide profile file `/etc/profile` to include default shell and text editor information for all users in the system.

Note: The `/etc/profile` file contains the environment variables for all users on the system.

You will also edit the user script file (`.profile`) that contains the system configuration information for an individual user. If this file is unavailable, you will create it and customize it according to your requirements.

You will perform the following tasks in this practice:

- Modify the `/etc/profile` file to set the default shell and default text editor to all users.
- Configure a user profile (`.profile`) file to include user-specific information.

Task 1: Modifying the `/etc/profile` File to Set the Default Shell and Text Editor to All Users

1. To add `bash` as the default shell and `vi` as the default editor for all users, include the following in the `/etc/profile` file:

```
ENV=$home/.bashrc
```

Note: This command will ensure that default shell for the users on this system is set to `bash`.

```
EDITOR=vi
export ENV EDITOR
```

Note: These commands will ensure that the default editor for the users on this system is set to `vi`.

2. Open a new terminal and log in as the `root` user to perform the above mentioned step.

```
$ su -
Password: oracle1
Oracle Corporation   SunOS 5.11     11.1   September 2012
You have new mail.
#

# vi /etc/profile

#
# Copyright (c) 1989, 2012, Oracle and/or its affiliates. All
rights reserved.
#
```

```
# The profile that all logins get before using their own
.profile.

ENV=$home/.bashrc
EDITOR=vi
export ENV EDITOR

trap ""  2 3
export LOGNAME PATH

if [ "$TERM" = "" ]
then
        if /bin/i386
        then
                TERM=sun-color
        else
                TERM=sun
        fi
        export TERM
fi

#       Login and -su shells get /etc/profile services.
#       -rsh is given its environment in its .profile.

case "$0" in
-sh | -ksh | -ksh93 | -jsh | -bash | -zsh)

        if [ ! -f .hushlogin ]
        then
                /usr/sbin/quota
                #       Allow the user to break the Message-Of-
The-Day only.
                trap "trap '' 2"  2
                /bin/cat -s /etc/motd
                                trap "" 2

                /bin/mail -E
                case $? in
                0)
                        echo "You have new mail."
                        ;;
                2)
```

```
                        echo "You have mail."
                        ;;
                esac
        fi
esac

umask 022
trap  2 3

:wq
```

3. Create or modify the `.profile` file for the user `oracle` and set the korn shell as the login shell. This file defines the user's profile at login.

   Note: Every user home directory usually contains a `.profile` file. If the file does not exist, you can create it by using the `vi` editor and adding the following text to the `.profile` file:

```
SHELL=/usr/bin/ksh
export SHELL
/usr/bin/ksh
```

   a. Exit from the `root` role into the `oracle` user account and verify your home directory.

```
# exit
logout
$ pwd
/opt/ora/labs/lab4/solutions
```

   b. Switch to the user home directory and check the current default shell by using the `echo` command.

```
$ cd

$ pwd
/home/oracle

$ echo $SHELL
/usr/bin/bash
```

4. Open the `.profile` file using `vi` editor and include the highlighted text to set the korn shell as the login shell.

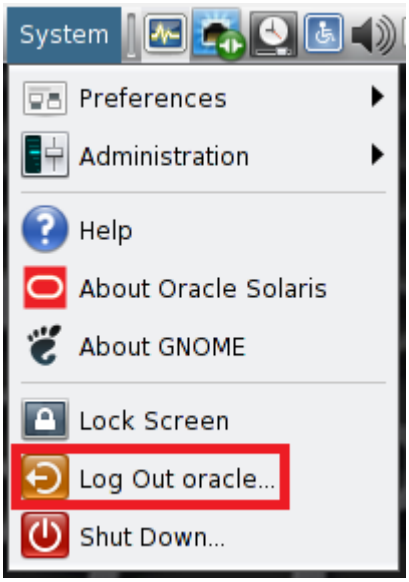
```
$ vi .profile

#
# Simple profile places /usr/bin at front, followed by
# /usr/sbin.
#
# Use less(1) or more(1) as the default pager for the man(1)
```

```
# command.
#
export PATH=/usr/bin:/usr/sbin

if [ -f /usr/bin/less ]; then
    export PAGER="/usr/bin/less -ins"
elif [ -f /usr/bin/more ]; then
    export PAGER="/usr/bin/more -s"
fi

# Apply korn shell as login shell
SHELL=/usr/bin/ksh
export SHELL
/usr/bin/ksh

#
# Define default prompt to <username>@<hostname>:<path><"($|
#) ">
# and print '#' for user "root" and '$' for normal users.
#
# Currently this is only done for bash/pfbash(1).
#
case ${SHELL} in
 *bash)
    typeset +x PS1="\u@\h:\w\\$ "
    ;;
esac

:wq
```

5. Close the terminal window by using the `exit` command.

```
$ exit
```

6. Log out by clicking **System** and then select **Log Out oracle…**



Note: You will be prompted to log out again. Click to continue logging out.

7. Log in again as the `oracle` user and confirm that the default shell is replaced by the korn shell (`ksh`), by using the `echo` command.

```
$ echo $SHELL
/usr/bin/ksh
```

8. Now, to switch back to the default `bash` shell, remove the korn shell entries from the `.profile` file.

   Note: This is a do-it-yourself step, which means that you will not find complete commands to perform this step.

   a. Remove the korn shell entries from the `.profile` file (`vi .profile`) and enable bash shell entries.

   b. Close the terminal window and log out and log in again as the `oracle` user by repeating the substeps from step 3 (a and b).

```
$ exit
```

   c. Log in again as the `oracle` user and confirm that the default shell `bash` is active.

```
$ echo $SHELL
/usr/bin/bash
$
```

9. For the `root` role, set the prompt to the `Test Prompt for root` followed by `#`. (**Hint:** Switch to root role and use the `vi` command to modify the `/.profile` file.):

```
PS1="`Test Prompt for root`# "
export PS1
```

Note: Comment out any existing settings for the default prompt (*old prompt*) temporarily and add the above-mentioned information (*new prompt*).

```
$ su -
Password: oracle1
Oracle Corporation   SunOS 5.11       11.1  September 2012
You have new mail.
# vi .profile

#
# Simple profile places /usr/bin at front, followed by
/usr/sbin.
#
# Use less(1) or more(1) as the default pager for the man(1)
command.
#
export PATH=/usr/bin:/usr/sbin

if [ -f /usr/bin/less ]; then
    export PAGER="/usr/bin/less -ins"
elif [ -f /usr/bin/more ]; then
    export PAGER="/usr/bin/more -s"
fi

#
# Define default prompt to <username>@<hostname>:<path><"($|#)
">
# and print '#' for user "root" and '$' for normal users.
#
# Currently this is only done for bash/pfbash(1).
#
# Old prompt
# case ${SHELL} in
# *bash)
#    typeset +x PS1="\u@\h:\w\\$ "
#    ;;
# esac

# New prompt
```

```
PS1="`Test Prompt for root`# "
export PS1

:wq
```

10. To verify the changes, exit from the `root` role and log in to the `root` role again.

```
root@s11-server:~# exit
logout
oracle@s11-server:~$ su -
Password: oracle1
Oracle Corporation   SunOS 5.11       11.1  September 2012
You have new mail.
`Test Prompt for root`#
```

Observation: The default prompt, `root@s11-server:~#` is replaced by the new prompt `` `Test Prompt for root`# ``.

11. To maintain consistency, you will continue using the default prompt (old prompt) `root@s11-server:~#` for the rest of the practices. Therefore, uncomment the default settings and remove the New Prompt settings as shown below.

```
# vi .profile

#
# Simple profile places /usr/bin at front, followed by
/usr/sbin.
#
# Use less(1) or more(1) as the default pager for the man(1)
command.
#
export PATH=/usr/bin:/usr/sbin

if [ -f /usr/bin/less ]; then
    export PAGER="/usr/bin/less -ins"
elif [ -f /usr/bin/more ]; then
    export PAGER="/usr/bin/more -s"
fi

#
# Define default prompt to <username>@<hostname>:<path><"($|#)
">
# and print '#' for user "root" and '$' for normal users.
#
# Currently this is only done for bash/pfbash(1).
#
```

```
# Old prompt
 case ${SHELL} in
 *bash)
    typeset +x PS1="\u@\h:\w\\$ "
    ;;
 esac

:wq
```

12. To confirm the availability of the default prompt, `exit` from the `root` role and log in to the `root` role again.

```
# exit
logout

$ su -
Password: oracle1
Oracle Corporation   SunOS 5.11 11.1  September 2012
#
```

Task 2: Configuring a User Profile (`.profile`) File to Include User-Specific Information

1. Close all the terminal windows, if required, and open a new terminal window and switch to the `root` role.

```
$ su -
Password:
Oracle Corporation   SunOS 5.11 11.1  September 2012
You have new mail.
#
```

2. Create or edit the `~/.bashrc` file. Add some bash shell commands that you would find useful. For example:
   - `set -o vi`
   - `alias ll='ls -l'`
   - `alias pss='ps -ef | sort +1n | more'`
   - `alias c='clear'`

```
# vi ~/.bashrc

#
```

```
# Define default prompt to <username>@<hostname>:<path><"($|#)
">
# and print '#' for user "root" and '$' for normal users.
#

typeset +x PS1="\u@\h:\w\\$ "

# Useful commands
set -o vi
alias ll='ls -l'
alias pss='ps -ef | sort +1n | more'
alias c='clear'

:wq
```

3. Run the `.` or `source` command to read the file and execute it in the current shell.

```
# . ~/.bashrc
```

4. Test the commands and confirm that they work correctly by running them in the shell.
   - `ll` must generate output equivalent to the `ls -l` command.
   - `pss` must generate output equivalent to the `ps -ef | sort +1n | more` command.
   - `c` must generate the output equivalent to the `clear` command.

```
# ll
total 0

# pss
     UID   PID  PPID   C    STIME TTY         TIME CMD
    root     0     0   0   Mar 14 ?           0:01 sched
    root     1     0   0   Mar 14 ?           0:00
/usr/sbin/init
    root     2     0   0   Mar 14 ?           0:00 pageout
    root     3     0   0   Mar 14 ?           0:55 fsflush
    <Output omitted>
# c

#
```

5. Edit the default `/etc/skel/local.profile` file to introduce new values to the environment variables in a shell.
   a. Append the `/usr/dt/bin` directory to the `PATH` variable.

```
# vi /etc/skel/local.profile

#
# Copyright (c) 1991, 2012, Oracle and/or its affiliates. All
rights reserved.
#
stty istrip
PATH=/usr/bin:/usr/sbin:/usr/dt/bin
export PATH
```

b. Create a `bash` shell user by using the command `useradd`. Use the information provided in the following table.

User Name:	user2
User ID:	<next available>
Primary Group:	10
Login Shell:	/usr/bin/bash
Password:	oracle1
Home Directory Path:	/export/home/user2

```
# useradd -d /export/home/user2 -m -s /usr/bin/bash -g 10 -c
"User 2" user2
    80 blocks
# passwd user2
New Password: oracle1
Re-enter new Password: oracle1
passwd: password successfully changed for user2
#
```

6. Log in as `user2`, and verify that the changes made to `/etc/skel/local.profile` were copied into the user's `$HOME/local.profile`.

```
# su - user2
user2@s11-server:~$ pwd
/export/home/user2
$ cat local.profile
#
# Copyright (c) 1991, 2012, Oracle and/or its affiliates. All
rights reserved.
#
stty istrip
PATH=/usr/bin:/usr/sbin:/usr/dt/bin
export PATH
```

Note: The complete prompt name is retained in the above step to provide clarity for the new user `user2`.

7. Using the `vi` editor, write a script named `multiply.sh` in which you assign an integer number to a variable. The output from the script should be the number, and the value of that number multiplied by itself.
   a. Close the terminal window and start a new terminal. Switch to the `solutions` directory under the `/opt/ora/labs/lab4` directory.

```
$ pwd
/home/oracle

$ cd /opt/ora/labs/lab4/solutions
```

   b. Create the script `multiply.sh`.

```
$ vi multiply.sh

#!/bin/bash

#Script Name: multiply.sh
# Purpose: To take a value from the parameter list and multiple
it by itself.
#
num=8

print "The number is: $num"
(( result = num * num ))
print "The number multiplied by itself is: $result"

:wq
```

   c. Change permissions to execute the script.

```
$ chmod 777 multiply.sh
```

   d. Execute the script.

```
$ ./multiply.sh

The number is: 8
The number multiplied by itself is: 64
```

This completes your practice. Close all the terminal windows.

Summary: In this practice, you performed tasks to manipulate local and environment variables in a shell.

Practices for Lesson 5: Pattern Matching

Chapter 5

Practices for Lesson 5: Pattern Matching

Practices Overview

In these practices, you will use regular expressions to search for a pattern within text files or a block of text, and to manipulate strings in the shell script. You will use the `grep` command with regular expressions that return text strings or lines matching an expression.

You will perform the following tasks in this practice:

- Use regular expressions to search for a pattern
- Use the `grep` command to search for specific string within text files

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
	Pattern Matching
	The `sed` Editor
	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 5-1: Using Regular Expressions and the `grep` Command

Overview

In this practice, you will write `grep` commands to search for the specified strings within text files.

Assumptions

The output of commands may vary from system to system. You will save your work in the `/opt/ora/labs/lab5/solutions` directory. The output may vary from system to system.

Tasks

1. Open a new terminal window by right-clicking anywhere on the desktop and selecting Open Terminal option.

2. Change the directory to `/opt/ora/labs/lab5`.

```
$ cd /opt/ora/labs/lab5
```

3. Print all the entries from `/etc/passwd` for users who have `bash` as their login shell.

```
$ grep 'bash$' /etc/passwd
root:x:0:0:Super-User:/root:/usr/bin/bash
oracle:x:100:10:oracle:/home/oracle:/usr/bin/bash
user2:x:101:10:User 2:/export/home/user2:/usr/bin/bash
```

4. Print all entries from `/etc/passwd` for users who have `ksh` as their login shell.

```
$ grep '/ksh$' /etc/passwd
upnp:x:52:52:UPnP Server Reserved UID:/var/coherence:/bin/ksh
```

The output indicates that the standard daemon (`upnp`) entry exists in the `/etc/passwd` file for `ksh`.

5. Print the lines from `/etc/group` that contains the pattern `root` preceded by the line number.

```
$ grep -n 'root' /etc/group
1:root::0:
2:other::1:root
3:bin::2:root,daemon
4:sys::3:root,bin,adm
5:adm::4:root,daemon
6:uucp::5:root
7:mail::6:root
8:tty::7:root,adm
9:lp::8:root,adm
10:nuucp::9:root
12:daemon::12:root
```

6. Print the lines from `/etc/group` that do not contain the pattern `root`.

```
$ grep -v 'root' /etc/group
staff::10:
sysadmin::14:
games::20:
smmsp::25:
gdm::50:
upnp::52:
xvm::60:
netadm::65:
mysql::70:
openldap::75:
webservd::80:
postgres::90:
slocate::95:
unknown::96:
nobody::60001:
noaccess::60002:
nogroup::65534:
aiuser::61:
pkg5srv::97:
vboxsf::100:
```

7. Determine the package name associated with the installed Java package on your machine. (**Hint:** Search using the `pkg list` command for Oracle Solaris and `rpm -qa` command for Oracle Linux.)

```
$ pkg list | grep -i 'java'
consolidation/java/java-incorporation           0.5.11-0.175.1.0.0.24.0    i--
consolidation/ub_javavm/ub_javavm-incorporation 0.5.11-0.175.1.0.0.24.1    i--
library/java/javahelp                           0.5.11-0.175.1.0.0.11.0    i--
runtime/java                                    1.6.0.35-0.175.1.0.0.24.1  i--
runtime/java/jre-6                              1.6.0.35-0.175.1.0.0.24.1  i--
runtime/java/jre-7                              1.7.0.7-0.175.1.0.0.24.0   i--
system/management/rad/client/rad-java           0.5.11-0.175.1.0.0.24.2    i--
web/browser/firefox/plugin/firefox-java         1.0-0.175.0.0.0.0.0        i--
```

Note: The `-i` option is used to ignore case, because `grep` is case-sensitive.

For Oracle Linux Users:

```
$ rpm -qa | grep -i 'java'
java-1.7.0-openjdk-devel-1.7.0.45-2.4.3.3.0.1.el6.x86_64
java-1.6.0-openjdk-devel-1.6.0.0-1.66.1.13.0.el6.x86_64
```

```
tzdata-java-2013g-1.el6.noarch
java-1.6.0-openjdk-1.6.0.0-1.66.1.13.0.el6.x86_64
java-1.7.0-openjdk-1.7.0.45-2.4.3.3.0.1.el6.x86_64
```

The following task is valid for Oracle Solaris users only:

8. View the contents of the Apache web server configuration file, `httpd.conf`.

```
$ cat /etc/apache2/2.2/httpd.conf
#
# This is the main Apache HTTP server configuration file.  It
contains the
# configuration directives that give the server its
instructions.
# See <URL:http://httpd.apache.org/docs/2.2> for detailed
information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.2/mod/directives.html>
# for a discussion of each configuration directive.
#
# Solaris Quick Configuration Information
#
# 1. Set ServerName if necessary (default is 127.0.0.1)
# 2. Set ServerAdmin to a valid email address
#
#
# Do NOT simply read the instructions in here without
understanding
# what they do.  They're here only as hints or reminders.  If
you are unsure
# consult the online docs. You have been warned.
#
# Configuration and logfile names: If the filenames you specify
for many
# of the server's control files begin with "/" (or "drive:/" for
Win32), the
# server will use that explicit path.  If the filenames do *not*
begin
# with "/", the value of ServerRoot is prepended -- so
"/var/apache2/2.2/logs/foo_log"
# with ServerRoot set to "/usr/apache2/2.2" will be interpreted
by the
# server as "/usr/apache2/2.2//var/apache2/2.2/logs/foo_log".


#
```

```
# ServerRoot: The top of the directory tree under which the
server's
# configuration, error, and log files are kept.
#
# Do not add a slash at the end of the directory path.  If you
point
# ServerRoot at a non-local disk, be sure to point the LockFile
directive
# at a local disk.  If you wish to share the same ServerRoot for
multiple
# httpd daemons, you will need to change at least LockFile and
PidFile.
#
ServerRoot "/usr/apache2/2.2"


#
# Listen: Allows you to bind Apache to specific IP addresses
and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below
to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 80


#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was
built as a DSO you
# have to place corresponding `LoadModule' lines within the
appropriate
# (32-bit or 64-bit module) /etc/apache2/2.2/conf.d/modules-
*.load file so that
# the directives contained in it are actually available _before_
they are used.
#
<IfDefine 64bit>
Include /etc/apache2/2.2/conf.d/modules-64.load
</IfDefine>
<IfDefine !64bit>
```

```
Include /etc/apache2/2.2/conf.d/modules-32.load
</IfDefine>


<IfModule !mpm_netware_module>
<IfModule !mpm_winnt_module>
….
….
….
….
….
# Note: The following must be present to support
#       starting without SSL on platforms with no /dev/random
equivalent
#       but a statically compiled-in mod_ssl.
#
<IfModule ssl_module>
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
</IfModule>
```

9. Now, use `grep` to search and filter information from the configuration file of Apache web server in your system. Filter all the comments from the `httpd.conf` file.

```
$ grep -v "#" /etc/apache2/2.2/httpd.conf

ServerRoot "/usr/apache2/2.2"

Listen 80

<IfDefine 64bit>
Include /etc/apache2/2.2/conf.d/modules-64.load
</IfDefine>
<IfDefine !64bit>
Include /etc/apache2/2.2/conf.d/modules-32.load
</IfDefine>


<IfModule !mpm_netware_module>
<IfModule !mpm_winnt_module>
User webservd
Group webservd
```

```
</IfModule>
</IfModule>


ServerAdmin you@yourhost.com

ServerName 127.0.0.1

DocumentRoot "/var/apache2/2.2/htdocs"

<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>


<Directory "/var/apache2/2.2/htdocs">
    Options Indexes FollowSymLinks

    AllowOverride None

    Order allow,deny
    Allow from all

</Directory>

<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>

<FilesMatch "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy All
</FilesMatch>

ErrorLog "/var/apache2/2.2/logs/error_log"

LogLevel warn
```

```
<IfModule log_config_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\"
\"%{User-Agent}i\"" combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common

    <IfModule logio_module>
      LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\"
\"%{User-Agent}i\" %I %O" combinedio
    </IfModule>

    CustomLog "/var/apache2/2.2/logs/access_log" common

</IfModule>

<IfModule alias_module>


    ScriptAlias /cgi-bin/ "/var/apache2/2.2/cgi-bin/"

</IfModule>

<IfModule cgid_module>
</IfModule>

<Directory "/var/apache2/2.2/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>

DefaultType text/plain

<IfModule mime_module>
    TypesConfig /etc/apache2/2.2/mime.types

    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz
```

```
    AddType application/vnd.pkg5.info .p5i
</IfModule>



Include /etc/apache2/2.2/conf.d/*.conf

<IfModule ssl_module>
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
</IfModule>
```

Observation: All the comment entries from the Apache configuration files are removed from the output. The `-v` option informs the `grep` command to invert its output, which means that instead of printing all matching lines, it prints all lines that don't match the expression. In this case, the # commented line.

For Oracle Linux 6 users, follow this task:

View the contents of LDAP server configuration file, `ldap.conf`, and use `grep` to search and filter information from the configuration file of the LDAP server in your system. Filter all the comments from the `ldap.conf` file.

```
$ cat /etc/openldap/ldap.conf
#
# LDAP Defaults
#

# See ldap.conf(5) for details
# This file should be world readable but not world writable.

#BASE     dc=example,dc=com
#URI ldap://ldap.example.com ldap://ldap-master.example.com:666

#SIZELIMIT     12
#TIMELIMIT     15
#DEREF         never
TLS_CACERTDIR  /etc/openldap/certs
```

```
$ grep -v "#" /etc/openldap/ldap.conf

TLS_CACERTDIR  /etc/openldap/certs
```

10. Continue from step 9, on Oracle Solaris VM and list all the entries of local disk devices from `/etc/vfstab`.

    Note: For Oracle Linux 6.5, use the `/etc/fstab` file.

```
$ grep '^/dev' /etc/vfstab
/devices  -          /devices   devfs -     no    -
/dev/zvol/dsk/rpool/swap  -          -          swap  -     no
    -
```

11. Switch to the `solutions` directory and create a shell script that places two `grep` commands from steps 7 and 8 in it. Change the file permission and execute the script.

    Note: For Oracle Linux, replace the following references: the `/etc/vfstab` file reference with the `/etc/fstab` file and `pkg list | grep -i 'java'` with the `rpm -qa | grep -i 'java'` command in the below script.

```
$ cd solutions
$ pwd
/opt/ora/labs/lab5/solutions

$ vi twogreps.sh
#!/bin/bash
# Script Name: twogreps.sh

echo "Here is the list of Java installation files in this
system"
pkg list | grep -i 'java'
echo
echo "Now a listing of local disk devices from /etc/vfstab"
grep '^/dev' /etc/vfstab

$ chmod 777 twogreps.sh
$ ./twogreps.sh
Here is the list of Java installation files in this system
consolidation/java/java-incorporation             0.5.11-
0.175.1.0.0.24.0    i--
consolidation/ub_javavm/ub_javavm-incorporation   0.5.11-
0.175.1.0.0.24.1    i--
library/java/javahelp                             0.5.11-
0.175.1.0.0.11.0    i--
runtime/java                                      1.6.0.35-
0.175.1.0.0.24.1  i--
runtime/java/jre-6                                1.6.0.35-
0.175.1.0.0.24.1  i--
runtime/java/jre-7                                1.7.0.7-
0.175.1.0.0.24.0   i--
```

```
system/management/rad/client/rad-java              0.5.11-
0.175.1.0.0.24.2    i--
web/browser/firefox/plugin/firefox-java            1.0-
0.175.0.0.0.0.0        i—
Now a listing of local disk devices from /etc/vfstab
/devices   -           /devices    devfs -     no     -
/dev/zvol/dsk/rpool/swap   -           -           swap  -     no
     -
```

Observation: Shell script help combines the results of two commands.

12. Copy the `adduser_sh.template` file from the `/opt/ora/labs/lab5` directory to create a script with the name `adduser.sh` in the `solutions` directory. Copy another file `mypasswd` from the `lab5` directory to the `solutions` directory.

```
$ cp adduser_sh.template solutions/adduser.sh
$ cp mypasswd solutions/
$ cd solutions
$ ls
adduser.sh mypasswd
```

13. Now update the file permissions and edit the `adduser.sh` script file to add a single `grep` command that allows you to determine whether the username (stored in the name variable) already exists in the `mypasswd` file. Test all the values for the name that appeared in the template file. Change the file name in the script code as displayed below. (You will build on this script in later practices as you encounter new concepts and constructs.)

```
$ chmod 777 adduser.sh
$ vi adduser.sh

#!/bin/bash
# Purpose: To write a script to add user to the system.
# Name: adduser.sh
### Set a shell variable to test the grep command.
name=nuucp # Should match a single entry
#name=uucp # Should match a single entry
#name=root # Should match a single entry
#name=rot # Should not match any entry
grep "^$name:" mypasswd
# End of lab

:wq
```

14. Change the file permission and execute the script to test the value for the user account, `nuucp`.

```
$ chmod 777 adduser.sh
$ ./adduser.sh
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
```

Note: The `nuucp` user account is used for the UNIX-to-UNIX copy program.

This completes your practice. Close all the terminal windows.

Summary: In this practice, you used regular expressions to search for a pattern within text files or a block of text, and to manipulate strings in the shell script. You also used the `grep` command with regular expressions that returns the text string or lines matching an expression.

Practices for Lesson 6: The sed Editor

Chapter 6

Practices for Lesson 6: The `sed` Editor

Practices Overview

The `sed` or Stream Editor is used to perform transformations on the text read from the source. In these practices, you will write `sed` commands to:

- Delete, write, search, and substitute text patterns using regular expressions
- Print lines from the input file to the standard output or to a specified file

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
	The `sed` Editor
	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 6-1: Using the `sed` Editor

Overview

In this practice, you will write `sed` commands to delete, write, search, and substitute, as well as print lines from the input file to standard output or to a specified file.

Assumptions

The output may vary from system to system. You will save your work in the `/opt/ora/labs/lab6/solutions` directory.

Tasks

1. Open a new terminal. Before beginning the practice, change to the `/opt/ora/labs/lab6` directory.

```
$ cd /opt/ora/labs/lab6
$
```

Note: You will use the `passwd2` file provided in the `lab6` directory for the duration of this lab.

2. Print the lines from `passwd2` file that end in the pattern `sh` to the standard output (screen).

```
$ sed -n '/sh$/p' passwd2
root:x:0:1:Super-User:/:/sbin/sh
user1:x:1001:10::/export/home/user1:/bin/sh
user2:x:1002:10::/export/home/user2:/bin/ksh
```

Note: Ensure that you execute this command from the `lab6` subdirectory as mentioned in step 1.

3. Print only the lines from the `passwd2` file that have the letter `o` as the second character.

```
$ sed -n '/^.o/p' passwd2
root:x:0:1:Super-User:/:/sbin/sh
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
```

4. Hide the first three characters of each line of the `passwd2` file and print the remaining output.

```
$ sed 's/^...//' passwd2
t:x:0:1:Super-User:/:/sbin/sh
mon:x:1:1::/:
:x:2:2::/usr/bin:
:x:3:3::/:
:x:4:4:Admin:/var/adm:
x:71:8:Line Printer Admin:/usr/spool/lp:
```

```
p:x:5:5:uucp Admin:/usr/lib/uucp:
cp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
ten:x:37:4:Network Admin:/usr/net/nls:
ody:x:60001:60001:Nobody:/:
ccess:x:60002:60002:No Access User:/:
ody4:x:65534:65534:SunOS 4.x Nobody:/:
r1:x:1001:10::/export/home/user1:/bin/sh
r2:x:1002:10::/export/home/user2:/bin/ksh
```

5. Print all the lines except those containing the pattern `uu` from the `passwd2` file.

```
$ sed '/uu/d' passwd2
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
user1:x:1001:10::/export/home/user1:/bin/sh
user2:x:1002:10::/export/home/user2:/bin/ksh
```

6. Print only the lines containing the pattern `oo` in the `passwd2` file.

```
$ sed -n '/oo/p' passwd2
root:x:0:1:Super-User:/:/sbin/sh
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
```

7. Now, copy `passwd2` and `hostwithspaces` files to the solutions directory. Change the permissions for all the files in the `solutions` directory.

   Note: You will use these files in the remaining steps.

```
$ cp passwd2 hostwithspaces solutions/
$ chmod 777 solutions/*
```

8. Create a file in the `/opt/ora/labs/lab6/solutions` directory consisting of the following:
   a. The `sed` commands from steps 6 and 7
   b. A command to display the contents of the `/etc/hosts` file after displaying the contents of the `passwd2` file in the output

```
$ cd solutions
$ vi labscript1.sed

/uu/d
/oo/p
$r /etc/hosts

:wq
```

9. Change the file permission to execute the `labscript1.sed` script by passing the `passwd2` file as the input file.

   Note: Remember to use the `-n` option while executing the script to replace the pattern space with the next line of input.

```
$ chmod 777 labscript1.sed
$ sed -n -f labscript1.sed passwd2
root:x:0:1:Super-User:/:/sbin/sh
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Internet host table
#
::1 s11-server localhost
127.0.0.1 localhost loghost
192.168.10.10  s11-server #Oracle Solaris server
192.168.10.20  ol6-server #Oracle Linux Server
```

10. Create a file consisting of `sed` commands to perform the following:
    - If the first letter on the line is an `l`, print the entire line appended with `**** SYM LINK`.
    - If the first letter on the line is a `-`, print the entire line appended with `**** PLAIN FILE`.

```
$ vi labscript2.sed

s/^l.*/& **** SYM LINK/
s/^-.*/& **** PLAIN FILE/

:wq
```

11. Change the file permission and execute the `labscript2.sed` script by sending the input of a long listing of the `/` (root) directory to the `sed` command through a `|` (pipe).

```
$ chmod 777 labscript2.sed
$ ls -l / | sed -f labscript2.sed
total 1036
lrwxrwxrwx   1 root     root           9 Jan 21 16:29 bin ->
./usr/bin **** SYM LINK
drwxr-xr-x   5 root     sys            8 Jan 21 16:29 boot
drwxr-xr-x   2 root     root           4 Mar 14 09:27 cdrom
drwxr-xr-x 261 root     sys          264 Mar 14 14:54 dev
drwxr-xr-x   4 root     sys            4 Mar 14 14:54 devices
drwxr-xr-x  90 root     sys          180 Mar 24 12:41 etc
drwxr-xr-x   3 root     staff          3 Jan 21 16:44 export
dr-xr-xr-x   2 root     root           2 Mar 14 09:27 home
drwxr-xr-x  19 root     sys           19 Jan 21 16:17 kernel
drwxr-xr-x  14 root     bin          302 Jan 21 16:17 lib
drwxr-xr-x   3 root     root           5 Mar 24 12:49 media
drwxr-xr-x   2 root     sys            2 Jan 21 16:29 mnt
dr-xr-xr-x   1 root     root           1 Mar 24 10:49 net
dr-xr-xr-x   1 root     root           1 Mar 24 10:49 nfs4
drwxr-xr-x   4 root     sys            4 Jan 21 11:42 opt
drwxr-xr-x   5 root     sys            5 Sep 20  2012 platform
dr-xr-xr-x 110 root     root      480032 Mar 24 16:30 proc
drwx------   2 root     root           7 Mar 19 15:43 root
drwxr-xr-x   3 root     root           3 Jan 21 16:31 rpool
lrwxrwxrwx   1 root     root          10 Jan 21 16:29 sbin ->
./usr/sbin **** SYM LINK
drwxr-xr-x   5 root     root           5 Sep 20  2012 system
drwxrwxrwt  11 root     sys         1124 Mar 24 13:13 tmp
drwxr-xr-x  33 root     sys           45 Jan 21 16:28 usr
drwxr-xr-x  39 root     sys           46 Jan 21 16:29 var
```

12. Create a `sed` script named `notrailingspaces.sed` to perform the following on the `hostwithspaces` input file:
    - Delete trailing spaces at the end of the line.

```
s/ *$//
```

    - Write all lines to a `hostsnospaces` output file.

```
w hostsnospaces
```

    Hint: You can use the `:set` option of the `vi` editor to view the trailing spaces.

```
$ vi notrailingspaces.sed

s/ *$//
w hostsnospaces

:wq
```

13. Change the file permission to execute the `notrailingspaces.sed` script and verify the results by viewing the `hostsnospaces` file's contents.

```
$ chmod 777 notrailingspaces.sed
$ sed -nf notrailingspaces.sed hostswithspaces

$ cat hostsnospaces
#
# Internet host table
#
127.0.0.1 localhost
192.9.200.111 ultrabear loghost
192.9.200.121 ladybear
192.9.200.122 littlebear
192.9.200.123 poppabear
192.9.200.124 newbear
192.9.200.125 sparcbear
```

This completes your practice. Close all the terminal windows.

Summary: In this practice, you used the `sed` editor to delete, write, search, and substitute text patterns using regular expressions and print lines from the input file to the standard output or to a specified file.

Practices for Lesson 7: The nawk Programming Language

Chapter 7

Practices for Lesson 7: The `nawk` Programming Language

Practices Overview

The `nawk` programming language allows you to develop programs within a script, with the help of data filtering, numerical and text processing, and reports generation.

In this practice, you perform the following tasks:

- Use `nawk` and regular expressions to process text.
- Use `nawk` to create a simple report.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
✓	The `sed` Editor
	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 7-1: Using `nawk` and Regular Expressions

Overview

In this practice, you use `nawk` commands to process text files and generate specified output.

Assumptions

Output may vary between systems. You save your work in the `/opt/ora/labs/lab7/solutions` directory.

Tasks

1. Close all terminal windows and open a new terminal window.

2. Before starting the practice, change to the `/opt/ora/labs/lab7` directory.

   Note: During the course of this practice, you will use `data.file` and a few other files as mentioned in the next step. These files are provided in the `lab7` directory.

3. Copy the following files to the solutions directory:
   - `currentuid`
   - `mypasswd`
   - `data.file`
   - `raggeddata.file`

   Note: You use these files in this practice.

```
$ cp currentuid mypasswd data.file raggeddata.file solutions
$ cd solutions/
$ ls
currentuid       data.file        mypasswd       raggeddata.file
$ chmod 777 *
```

   Note: Change file permissions after the copy operation as shown in the preceding step. You use these files in the rest of the practice.

For every line of the `data.file` file, print the following:

- A string `Name:`,
- The person's first name
- The person's last name
- A tab space
- The string `Region:`
- The abbreviation of the region

```
$ cat data.file
northwest  NW    Joel Craig   3.0 .98    3     4
western    WE    Sharon Kelly 5.3 .97    5     23
southwest  SW    Chris Foster 2.7 .8     2     18
southern   SO    May Chin     5.1 .95    4     15
southeast  SE    Derek Johnson     5.0 .70     4     17
```

```
eastern    EA    Susan Beal        4.4 .8      5     20
northeast  NE    TJ Nichols        5.1 .94     3     13
north      NO    Val Shultz        4.5 .89     5     9
central    CT    Sheri Watson      5.7 .94     5     13

$ nawk '{ print "Name:", $3, $4, "\t" "Region:", $2 }' data.file
Name: Joel Craig       Region: NW
Name: Sharon Kelly     Region: WE
Name: Chris Foster     Region: SW
Name: May Chin         Region: SO
Name: Derek Johnson    Region: SE
Name: Susan Beal       Region: EA
Name: TJ Nichols       Region: NE
Name: Val Shultz       Region: NO
Name: Sheri Watson     Region: CT
```

4. For every line of the `data.file` file that matches the pattern of a `W`, followed or preceded by another capital letter, print fields 2, 3, 4, and 1.

```
$ nawk '/[A-Z]W/ { print $2, $3, $4, $1 } /W[A-Z]/ { print $2,
$3, $4, $1 }' data.file
NW NW Craig northwest
WE Sharon Kelly western
SW SW Foster southwest
```

5. For the output of step 4, add a leading title to the report called `Developers in the Western Regions`.

```
$ nawk 'BEGIN { print "Developers in the Western Regions" } /[A-
Z]W/ { print $2, $3, $4, $1 } /W[A-Z]/ { print $2, $3, $4, $1 }'
data.file
Developers in the Western Regions
NW Joel Craig northwest
WE Sharon Kelly western
SW Chris Foster southwest
```

6. To the output of step 4, add a trailing footer to the report called `End of Report on the Western Regions`.

```
$ nawk 'BEGIN { print "Developers in the Western Regions" } /[A-
Z]W/ { print $2, $3, $4, $1 } /W[A-Z]/ { print $2, $3, $4, $1 }
END { print "End of Report on the Western Regions" }' data.file
Developers in the Western Regions
NW Joel Craig northwest
WE Sharon Kelly western
SW Chris Foster southwest
End of Report on the Western Regions
```

7. For every line of the `data.file` file, print the following:
   - A line number
   - A colon
   - A tab space
   - The entire record

```
$ nawk '{ print NR, ":\t" $0 }' data.file
1 :  northwest  NW   Joel Craig      3.0 .98 3  4
2 :  western    WE   Sharon Kelly    5.3 .97 5  23
3 :  southwest  SW   Chris Foster    2.7 .8  2  18
4 :  southern   SO   May Chin        5.1 .95 4  15
5 :  southeast  SE   Derek Johnson   5.0 .70 4  17
6 :  eastern    EA   Susan Beal      4.4 .8  5  20
7 :  northeast  NE   TJ Nichols      5.1 .94 3  13
8 :  north      NO   Val Shultz      4.5 .89 5  9
9 :  central    CT   Sheri Watson    5.7 .94 5  13
```

8. For every line of the `data.file` file, set the output field separator to a tab space and print the following:
   - An initial heading for the report as `****** Regional Report ******`
   - A line number
   - The person's last name
   - The region name

```
$ nawk 'BEGIN { OFS="\t" } BEGIN { print "****** Regional Report
******" } { print NR, $4, $1 }' data.file
****** Regional Report ******
1    Craig      northwest
2    Kelly      western
3    Foster     southwest
4    Chin       southern
5    Johnson    southeast
6    Beal       eastern
7    Nichols    northeast
8    Shultz     north
9    Watson     central
```

Practice 7-2: Using `nawk` to Create a Report

Overview

In this practice, you create a report on the top three users of disk space in a file system. You use the `nawk` command from the command line to obtain and manipulate data. After you are certain of the commands and command sequence, you place the commands in a shell script for execution.

Tasks

1. Write a `nawk` program called `northsouthreport.nawk` that does the following:
   - Reads the `data.file` file
   - Prints a report of all records that contain north or south in the region name
   - Counts how many records are listed for both the north and south regions
   - Prints the total count at the end of the report

   The report must look like the following:

Record #	Region Name	Employee	Region Count
1	northwest	Craig	1
3	southwest	Foster	1
4	southern	Chin	2
5	southeast	Johnson	3
7	northeast	Nichols	2
8	north	Shultz	3

   Total of North region is: 3

   Total of South region is: 3

```
$ vi northsouthreport.nawk
BEGIN { Ncount = 0; Scount=0 }
BEGIN { printf "%s %15s %15s %15s \n", "Record #", "Region
Name", "Employee", "Region Count" }
/north/ { printf "%6d %15s %15s %15d \n", NR, $1, $4, Ncount =
Ncount + 1 }
/south/ { printf "%6d %15s %15s %15d \n", NR, $1, $4, Scount =
Scount + 1 }
END { print "Total of North region is:", Ncount }
END { print "Total of South region is:", Scount }
~
~
~
:wq
```

2. Execute the `northsouthreport.nawk` program to create a report as expected in the description of the step 1.

```
$ nawk -f northsouthreport.nawk data.file
```

```
Record #     Region Name         Employee    Region Count
     1       northwest           Craig              1
     3       southwest           Foster             1
     4        southern             Chin              2
     5       southeast          Johnson             3
     7       northeast          Nichols             2
     8           north           Shultz             3
Total of North region is: 3
Total of South region is: 3
```

3. Write a `nawk` script called `raggedcount.nawk` that counts the number of fields in the `raggeddata.file` file.

```
$ vi raggedcount.nawk
BEGIN { print "This program is counting the number of fields in"
}
BEGIN { print "the file raggeddata.file, please wait." }

{ totalfields = totalfields + NF }

END { print "==> The total number of fields is:", totalfields }

:wq
```

4. Execute the `raggedcount.nawk` script to count the number of fields in the `raggeddata.file` file.

```
$ nawk -f raggedcount.nawk raggeddata.file
This program is counting the number of fields in
the file raggeddata.file, please wait.
==> The total number of fields is: 58
```

5. Write a script called `bashcounter.nawk` that counts and lists the `bash` shell user entries in the `mypasswd` file. Update the file permission to allow the script to be executed.

   Note: Use the `mypasswd` file that is found in the `/opt/ora/labs/lab7` directory for the tasks that follow.

```
$ vi bashcounter.nawk
BEGIN { print "\nHere is the listing of Bash shell users \n" }

/\/bash$/ { print $0; bashtotal = bashtotal + 1 }

END { print "==> Total number of Bash shell users:", bashtotal }

:wq
```

6. Change the file permission to enable the execution of the script.

```
$ chmod 777 bashcounter.nawk
```

7. Write a `nawk` script called `korncounter.nawk` that counts and lists the Korn shell user entries in the `/etc/passwd` file.

```
$ vi korncounter.nawk
BEGIN { print "\nHere is the listing of Korn shell users \n" }

/\/ksh$/ { print $0; korncounter = korncounter + 1 }

END { print "==> Total number of Korn shell users:", korncounter
}

:wq
```

8. Change the file permission to enable the execution of the script.

```
$ chmod 777 korncounter.nawk
```

9. Write a shell script called `shshscript.sh` that calls both the `bashcounter.nawk` and `korncounter.nawk` scripts that count users in the `/etc/passwd` file based on their default shell.

```
$ vi shshscript.sh

#!/bin/bash

echo "This is a report on default shells for the local users"

nawk -f bashcounter.nawk /etc/passwd

nawk -f korncounter.nawk /etc/passwd
```

10. Change the file permission to execute the `shshscript.sh` script to view the report on the default shells of the users.

```
$ chmod 777 shshscript.sh
$ ./shshscript.sh
This is a report on default shells for the local users

Here is the listing of Bash shell users

root:x:0:0:Super-User:/root:/usr/bin/bash
oracle:x:100:10:oracle:/home/oracle:/usr/bin/bash
user2:x:101:10:User 2:/export/home/user2:/usr/bin/bash
==> Total number of Bash shell users: 3

Here is the listing of Korn shell users

upnp:x:52:52:UPnP Server Reserved UID:/var/coherence:/bin/ksh
==> Total number of Korn shell users: 1
```

Note: The preceding output may vary from system to system.

11. Use the `nawk` program to extract all the UIDs from the `mypasswd` file into a file called `currentuid`.

Note: Use the `mypasswd` file that is found in the `/opt/ora/labs/lab7` directory for the tasks that follow.

```
$ nawk -F: '{print $3}' mypasswd > currentuid

$ cat currentuid
0
1
2
3
4
71
5
9
37
60001
60002
65534
1001
1002
```

12. Use the `sed` editor to remove all the system UID numbers (that is, all the one-, two-, and three-digit UIDs) from the `currentuid` file and write the remaining UIDs to a file named `currentuid2`.

```
$ sed -e '/^.$/d' -e '/^..$/d' -e '/^...$/d' currentuid >
currentuid2
$ cat currentuid2
60001
60002
65534
1001
1002
```

13. Use the `sed` editor to remove the `nobody` (`60001`), `noaccess` (`60002`), and `nobody4` (`65534`) UIDs and write the remaining UIDs to a file named `currentuid3`.

```
$ sed -e '/^6000[12]$/d' -e '/^65534$/d' currentuid2 >
currentuid3
$ cat currentuid3
1001
1002
```

14. Close all the terminal windows.
    This completes the practice.

Summary: In this practice, you processed text data and created reports by using the `nawk` command and regular expressions. You also identified that the `nawk` command helped you to extract specific information in combination with the `sed` command.

Practices for Lesson 8: Interactive Scripts

Chapter 8

Practices for Lesson 8: Interactive Scripts

Practices Overview

Interactive scripts are those scripts that require input from the user or give output to the user as the script is running. These scripts are more flexible and allow customization while running.

In this practice, you perform the following tasks:

- Write a backup script that accepts a file system as a command-line argument.
- Modify the `adduser` script to prompt for an advanced user's information.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
✓	The `sed` Editor
✓	The `nawk` Programming Language
	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 8-1: Writing a ZFS File System Backup Script

Overview

In this practice, you write a backup script named `zfsbackup` that accepts a file system as a command-line argument. This script prompts the user to enter the file system that should be specified in the command.

Assumptions

The output of some commands may vary between systems. You save your work in the `/opt/ora/labs/lab8/solutions` directory.

Tasks

1. Close all terminal windows if required and open a new terminal window. Switch to the `root` role.

```
$ su -
Password: oracle1
Oracle Corporation   SunOS 5.11       11.1  September 2012
You have new mail.
#
```

2. Before starting the practice, change to the `/opt/ora/labs/lab8` directory.

3. List the commands that you plan to include while in the ZFS file system backup (snapshot) script, which will automate the `zfs snapshot` commands to create a file system backup.

   Note: The output for these commands may vary from system to system.

   a. List the ZFS file systems that are available on the system:

```
# zfs list
NAME                          USED  AVAIL  REFER  MOUNTPOINT
rpool                        6.14G  22.9G  4.90M  /rpool
rpool/ROOT                   4.07G  22.9G    31K  legacy
rpool/ROOT/solaris           4.07G  22.9G  3.77G  /
rpool/ROOT/solaris/var        200M  22.9G   195M  /var
rpool/VARSHARE               57.5K  22.9G  57.5K  /var/share
rpool/dump                   1.03G  22.9G  1.00G  -
rpool/export                  973K  22.9G    32K  /export
rpool/export/home             941K  22.9G    33K  /export/home
rpool/export/home/oracle      872K  22.9G   872K  /export/home/oracle
rpool/export/home/user2        36K  22.9G    36K  /export/home/user2
rpool/swap                   1.03G  22.9G  1.00G  -
```

b. Create a recursive ZFS snapshot named `archive` of the `/export/home` ZFS pool, in the format `zfs snapshot -r <dataset name>@<archive name>`.

```
# zfs snapshot -r rpool/export/home@archive
```

c. Verify that the `archive` snapshot was created.

```
# zfs list rpool/export/home@archive
NAME                          USED  AVAIL  REFER  MOUNTPOINT
rpool/export/home@archive        0      -    33K  -
```

d. Create a local copy of the `archive` in compressed format and confirm its availability by using the `ls` command.

```
# zfs send -Rv rpool/export/home@archive | gzip >
/opt/ora/labs/lab8/solutions/sample.zfs.gz
sending from @ to rpool/export/home@archive
sending from @ to rpool/export/home/oracle@archive
sending from @ to rpool/export/home/user2@archive
# cd solutions
# ls
sample.zfs.gz
```

4. Destroy the snapshot that was created in step 3 by running the `zfs destroy` command and confirm its non-availability.

   Note: Run this step only if you have not modified the script to destroy the snapshot as mentioned in the note for step 3.

```
# zfs destroy -r rpool/export/home@archive
# zfs list -t snapshot
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool/ROOT/solaris@install          105M      -  3.52G  -
rpool/ROOT/solaris/var@install     5.42M      -   189M  -
```

   Observation: Note that the recently created snapshot `home@archive` does not exist anymore.

5. Confirm whether the current working directory is `solutions` under `lab8` and write a script named `zfsbackup` that automates the `zfs snapshot` commands to take file system backup. Use the commands tested in step 3. Ensure that you provide information about each command and step as comments in the script.

```
# pwd
/opt/ora/labs/lab8/solutions

# vi zfsbackup.sh

#!/bin/bash

# Script Name: zfsbackup.sh

# Following is the list of ZFS file system available for backup:
```

```
zfs list
echo "Type the zfs file system that you want to backup [Enter]: "
 read rpool

# Note: If you press Enter and continue above, it will create snapshot
# of /export/home zfs pool by default.

# The following command creates a recursive snapshot named archive
# of the /export/home pool.
zfs snapshot -r rpool/export/home@archive

# The following command creates a local copy of the archive in
compressed format
zfs send -Rv rpool/export/home@archive | gzip > /opt/ora/sample.zfs.gz

# List the local snapshot
zfs list rpool/export/home@archive

# Destroy the local snapshots as the recovery archive are created
# zfs destroy -r rpool/export/home@archive
```

Note: You can modify the `zfsbackup` script per your requirements. Test the script by enabling the last command `zfs destroy`, which will destroy the snapshot.

6. Change the file permission to execute the `zfsbackup.sh` script.

```
# chmod 777 zfsbackup.sh
# ./zfsbackup.sh
NAME                          USED  AVAIL  REFER  MOUNTPOINT
rpool                        6.14G  22.9G  4.90M  /rpool
rpool/ROOT                   4.07G  22.9G    31K  legacy
rpool/ROOT/solaris           4.07G  22.9G  3.77G  /
rpool/ROOT/solaris/var        201M  22.9G   195M  /var
rpool/VARSHARE                 82K  22.9G    82K  /var/share
rpool/dump                   1.03G  22.9G  1.00G  -
rpool/export                  980K  22.9G    32K  /export
rpool/export/home             948K  22.9G    33K  /export/home
rpool/export/home/oracle      878K  22.9G   878K
/export/home/oracle
rpool/export/home/user2        37K  22.9G    37K
/export/home/user2
rpool/swap                   1.03G  22.9G  1.00G  -
Type the zfs file system that you want to backup [Enter]:
/export/home # type /export/home
sending from @ to rpool/export/home@archive
sending from @ to rpool/export/home/oracle@archive
```

```
sending from @ to rpool/export/home/user2@archive
NAME                          USED  AVAIL  REFER  MOUNTPOINT
rpool/export/home@archive        0      -    33K  -
```

Practice 8-2: Modifying the `adduser` Script to Prompt for User Information

Overview

In this practice, you edit a copy of the `adduser` script to prompt for a user's primary group, login shell, and comment information.

Assumptions:

- In this practice, you modify an existing script, the `adduser.sh` script, and the `mypasswd` file, to prompt for more information about the user.
- A copy of the `adduser.sh` script and the `mypasswd` file, including a few other dependent files (see step 2 below), is available in the `/opt/ora/labs/lab8` directory.

Tasks

1. Close all terminal windows and open a new terminal window.

2. Switch to the `/opt/ora/labs/lab8` directory and copy the following files to the `solutions` directory:
   - `adduser.sh`
   - `mypasswd`
   - `currentuid`
   - `currentuid2`
   - `currentuid3`
   - `currentuid4`

   ```
   $ cd /opt/ora/labs/lab8
   $ cp adduser.sh mypasswd currentuid currentuid2 currentuid3
   currentuid4 solutions/
   $ chmod 777 solutions/*
   ```

   Note: Change file permissions after the copy operation as shown in the preceding step. You use these files in the rest of the practice.

3. Switch to the `solutions` directory and confirm whether the files mentioned in the preceding list are copied.

   ```
   $ cd solutions
   $ ls
   adduser.sh   currentuid   currentuid2  currentuid3  currentuid4
   mypasswd
   ```

   Note: You might observe a few other files in this directory—these were created in the previous practice.

4. List the command constructs to perform the following (do not perform them directly from the command line):
   a. Prompt the user to enter a group name and verify that the name entered exists in the `/etc/group` file.

```
echo "What primary group should $1 belong to?"
echo "Please specify the group name: \c"
read group junk
```

   b. If the group name is not in the `/etc/group` file, inform the user that this is not a valid group and exit the script.

```
grep "^$group:" /etc/group > /dev/null  2>&1
if  [ $? -ne 0 ]
then
        echo "The group $group is not valid."
        echo
        exit 2
fi
```

   c. Prompt the user to select the login shell for the new user.

   Note: You can also list the ones that you will accept with a *here* document.

```
$ echo "Good, now select a login shell for user $1"
Good, now select a login shell for user
$ cat << LOGINSHELLS
> bash
> csh
> ksh
> sh
> LOGINSHELLS
bash
csh
ksh
sh
$
```

   d. Read the requested shell choice.

```
read ushell
```

   e. If the shell choice is acceptable:

```
if [ $ushell = "bash" –o $ushell = "csh" -o  $ushell = "ksh" -o
$ushell = "sh" ]
```

      1) Prompt for the comment field information

```
echo "Please enter the comment information for the user:"
```

      2) Read the comment information

```
read comment
```

Note: Assume that the login directory for the user is the `/export/home/<username>` directory.

f. Print an entry for the new user by using colon delimiters between the seven fields.

```
echo "$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell"
```

g. Append the new user entry to the `mypasswd` file.

```
echo "$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell" >>
./mypasswd
```

h. If the shell requested is not the one you expected, inform the user that the shell is not a valid shell and exit the script.

Hint: You can run the solution to see the script flow. When doing this, add some invalid information if you want to test error messages.

```
else
       echo "Sorry, that is not a valid shell, exiting..."
       exit 3
```

5. *Now, consolidate all the information from step 3 and update the `adduser.sh` script as highlighted in red in the following output:*

```
$ vi adduser.sh

#!/bin/bash
# Purpose: To write a script to add user to the system.
# Name: adduser.sh

# Use nawk to extract all the UIDs into the temp file currentuid
nawk -F: '{print $3}' ./mypasswd > ./currentuid

# Use sed to remove all 1, 2, and 3 digit UIDs from currentuid,
# and place the output in the temp file currentuid2
sed -e '/^.$/d' -e '/^..$/d' -e '/^...$/d'  ./currentuid >
./currentuid2



# Use sed to remove UIDs 60001, 60002, and 65534 from
# currentuid2,and place the output in the temp file currentuid3
sed -e '/^6000[12]$/d' -e '/^65534$/d'  ./currentuid2 >
./currentuid3

# Prompt for a group name and verify whether it exists in
/etc/group

echo "What primary group should $1 belong to? "
echo "Please specify the group name:"
read group junk
```

```
grep "^$group:" /etc/group > /dev/null  2>&1
if  [ $? -ne 0 ]
then
        echo "The group $group is not valid."
        echo
        exit 2
fi

echo "Good, now select a login shell for user $1"
cat << LOGINSHELLS
> bash
> csh
> ksh
> sh
LOGINSHELLS

read ushell

if [ $ushell = "bash" -o $ushell = "csh" -o  $ushell = "ksh" -o
$ushell = "sh" ]
then
        echo "Please enter the comment information for the
user:"
        read comment

        echo "The new user entry looks like this:"
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell"
        echo " "
        echo "New user $1 has been added to the ./mypasswd file"
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell" >>
./mypasswd
else
       echo "Sorry, that is not a valid shell, exiting..."
exit 3
fi
```

6. To test the script, add a new user `user3`.

```
$ ./adduser.sh user3
The uid for user3 is: 1003
What primary group should user3 belong to?
Please specify the group name: nobody
Good, now select a login shell for user user3
> bash
> csh
> ksh
> sh
bash
Please enter the comment information for the user:
This is a test user
The new user entry looks like this:
user3:x:1003:nobody:This is a test user:/export/home/user3:/bin/bash

New user user3 has been added to the ./mypasswd file
```

7. View the `mypasswd` file to confirm whether the new user, `user3`, is added with the necessary profile information.

```
$ cat mypasswd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
user1:x:1001:10::/export/home/user1:/bin/sh
user2:x:1002:10::/export/home/user2:/bin/ksh
user3:x:1003:nobody:This is a test user:/export/home/user3:/bin/bash
```

Observe that the new user, `user3`, has been added to the `./mypasswd` file.

8. Close all the terminal windows.
   This completes the practice.

Summary: In this practice, you wrote a `zfsbackup` script that creates a snapshot (backup) of the ZFS file system as a command-line argument. You also modified an existing `adduser` script to prompt for user information.

Practices for Lesson 9: Variables and Positional Parameters

Chapter 9

Practices for Lesson 9: Variables and Positional Parameters

Practices Overview

Variables are containers that hold values that are used in scripts. Positional parameters reference each word on the command line by its position on the command line.

In this practice, you write scripts that manipulate the positional parameters, use a loop construct, and perform integer arithmetic.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
✓	The `sed` Editor
✓	The `nawk` Programming Language
✓	Interactive Scripts
	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 9-1: Using Advanced Variables, Parameters, and Argument Lists

Overview

In this practice, you write scripts that manipulate the positional parameters, use a loop construct, and perform integer arithmetic.

Assumptions

For Oracle Linux, replace the `print` command with `echo` in the following tasks.

Tasks

1. Close all terminal windows if required and open a new terminal window.

2. Before starting the practice, change to the `/opt/ora/labs/lab9` directory. Switch to the `solutions` directory.

```
$ cd /opt/ora/labs/lab9
$ cd solutions
$ pwd
/opt/ora/labs/lab9/solutions
```

3. List the command constructs to perform the following tasks (do not perform them directly from the command line):
   a. Display the name of the script that is being executed (`$0`).

```
print "The name of the script is: $0"
```

   b. Set the values of the positional parameters by using the `set` statement. Pass the values `a`, `b`, and `c` into the script and assign the positional parameters `$1`, `$2`, and `$3`, respectively. Print the values of the positional parameters by using the `echo` statement.

```
print "Executing script $0 \n"
print "$1 $2 $3"

set uno duo tres
print "One two three in Latin is:"
print "$1"
print "$2"
print "$3 \n"
```

c. The variable `textline` is assigned a value in the form of a string. Assign new values to the positional parameters by using the `set` statement.

```
textline="name phone address birthdate salary"
set $textline
print "$*"
print 'At this time $1 =' $1 'and $4 =' $4 "\n"
```

Note: Because `$textline` is replaced by its value (a string of several words), the words become the positional parameter values.

d. Print the positional parameter values by using the `print $*` statement and shift the positional parameters by using the `shift` statement.

```
shift 3
print "$* \n"
shift --
print "$0 $*"
```

4. Now write a script called `pospara2.sh` to include all the command constructs listed in step 3.

```
$ vi pospara2.sh

#!/bin/bash

# Script name: pospara2.sh

print "Executing script $0 \n"
print "$1 $2 $3"

set uno duo tres
print "One two three in Latin is:"
print "$1"
print "$2"
print "$3 \n"

textline="name phone address birthdate salary"
set $textline
print "$*"
print 'At this time $1 =' $1 'and $4 =' $4 "\n"

shift 3
print "$* \n"

shift --
print "$0 $*"
```

5. Change file permissions to execute the `pospara2.sh` script and display the results.

```
$ chmod 777 pospara2.sh
$ ./pospara2.sh a b c
Executing script ./pospara2.sh

a b c
One two three in Latin is:
uno
duo
tres

name phone address birthdate salary
At this time $1 = name and $4 = birthdate

birthdate salary

./pospara2.sh salary
```

Observation: The script displays the values of positional parameters as required.
This completes the current task list.

6. Write a script called `parsepath.sh` that parses information from a long path name.

Note: Assume that the path name is the only item passed in on the command line.

```
$ vi parsepath.sh
#!/bin/bash

# Purpose: To parse a long pathname.
#
# Name: parsepath.sh

if [ $# -ne 1 ]; then
   echo "Usage: parsepath.sh /pathname"
   exit 1
fi

name=$1

print
print "The pathname is: $name"


while [[ -n $name ]]
do
```

```
   print $name

   name=${name%/*}
done

print
```

7. Change file permissions to execute the `parsepath.sh` script to parse information from a long path name.

   A good path name is: `/usr/share/lib/terminfo/s/sun`

   An example execution is:

```
$ chmod 777 parsepath.sh
$ ./parsepath.sh /usr/share/lib/terminfo/s/sun
The pathname is: /usr/share/lib/terminfo/s/sun
/usr/share/lib/terminfo/s/sun
/usr/share/lib/terminfo/s
/usr/share/lib/terminfo
/usr/share/lib
/usr/share
/usr
```

8. Close all the terminal windows.

   This completes the practice.

Summary: In this practice, you wrote two scripts that manipulate the positional parameters, use a loop construct, and perform integer arithmetic.

Practices for Lesson 10: Conditionals

Chapter 10

Practices for Lesson 10: Conditionals

Practices Overview

In a shell script, you need to accommodate different courses of action, depending on the success or failure of a command. Conditionals provide this ability to the shell script.

In this practice, you perform the following tasks:

- Answer questions about usage of conditionals and create a script that determines whether the path name is a directory or a file.
- Modify the existing `adduser` script to add more functionality with the use of conditionals to create a new user.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
✓	The `sed` Editor
✓	The `nawk` Programming Language
✓	Interactive Scripts
✓	Variables and Positional Parameters
	Conditionals
	Loops
	Functions
	Traps

Practice 10-1: Using Conditionals

Overview

In this practice, you are provided with code snippets to analyze their purpose. This is followed by the creation of a script named `filetype`, which uses conditionals to determine logic.

Tasks

1. Close all terminal windows if required and open a new terminal window.

2. Before starting the practice, change to the `/opt/ora/labs/lab10` directory. Switch to the `solutions` directory.

```
$ cd /opt/ora/labs/lab10
$ cd solutions
$ pwd
/opt/ora/labs/lab10/solutions
```

Note: This is where you will save your work.

3. Read through the following code snippet, and answer the questions that follow.

```
startnfsd=0
if [ -f /etc/dfs/dfstab ] && /usr/bin/egrep -v '^[       ]*(#|$)' \
    /etc/dfs/dfstab >/dev/null 2>&1; then

    /usr/sbin/shareall -F nfs
```

The preceding lines of code determine when the machine should attempt to share all the directories listed in the `dfstab` file. Given this information, answer the following questions:

a. What does the following command do?

```
 [ -f /etc/dfs/dfstab ]
```

Answer: *It determines whether the `/etc/dfs/dfstab` file exists, and whether it is a regular file. The exit status is `0` if the file exists and `1` if it is non-existent.*

b. What does the following command do?

```
/usr/bin/egrep -v '^[       ]*(#|$)' /etc/dfs/dfstab >/dev/null 2>&1
```

Answer: *The `egrep` command finds all lines that do not begin with zero or more white spaces, followed by a `#` or end of line (`$`) in the `/etc/dfs/dfstab` file. The output and error messages are sent to the `/dev/null` file. `egrep` then returns `0` if it successfully finds any line with the pattern, or it returns `1` if it failed to find the specified pattern.*

c. What does the `if` statement evaluate?

Answer: *The `if` statement evaluates the results of the Boolean (`&&` operator) of the `-f` command and the results of the `egrep` command. The `&&` operator returns a `0` exit status if both commands succeed, or a nonzero exit status if one or both commands fail.*

- *If the `&&` operator returns a `0` exit status, the `if` statement returns true. In this situation, the `if` statement executes the statements that follow the `then` statement.*

– *If the `&&` operator returns a nonzero exit status, the `if` statement returns false. In this situation, the `if` statement executes the statements that follow the `else` statement (if an `else` statement exists), or the `fi` statement.*

4. List the command constructs to be included in a script named `filetype`, which accepts exactly one command-line argument and assumes that the command-line argument is a path name to a file or directory. The command constructs should perform the following:
   a. Output a `Usage` message and exit the script immediately when the script is executed with more or less than one command-line argument.

```
if [ $# -ne 1 ]; then
   echo "Usage: filetype pathname"
   exit 1
fi
```

   b. If the path name is a directory:
      1) List the directory name
      2) List the contents of the directory
      3) List the file space consumed by the directory and all the files and subdirectories of the directory, in kilobytes

   Note: Remember that this command must perform a calculation before the number is displayed. Therefore, the script will have to wait if you select a directory that is deep or large.)

```
if [ -d $1 ]; then
   echo "$1 is a directory."
   echo "Following is a listing of the $1 directory."
   ls $1
   echo " "
   echo "The number of kilobytes used by directory $1 is: \c"
   du -s -k $1 | nawk '{print $1}'
else
```

   c. If the path name is not a directory, assume that the path name is a file and perform the following:
      1) List the file name.
      2) List the file size in bytes.
      3) Report whether the file is any of the following:
         - Block-special file
         - Character-special file
         - `set-user-id` bit file

   Hint: To validate the character-special and blocked-special tests, use the physical device name associated with a slice of your local disk. Following are some examples of the physical device name:

   - Block-special device

```
/devices/pci@1f,0/pci@1,1/ide@3/dad@0,0:a
```

   - Character-special device

```
/devices/pci@1f,0/pci@1,1/ide@3/dad@0,0:a,raw
```

Note: The `format` command will list the devices information.

- File with the `set-user-id` permission

```
/usr/bin/crontab
```

```
if [ -d $1 ]; then
...
else
        echo "$1 is a file."
        echo "The size of $1 in bytes is: \c"
        ls -l $1 | nawk '{print $5}'
        echo
        if [ -b $1 ]; then
                echo "$1 is a block special file."
        elif  [ -c $1 ]; then
                echo "$1 is a character special file."
        elif  [ -u $1 ]; then
                echo "$1 has the set-user-id permission."
        else

fi
```

5. Write a script named `filetype.sh` by using the `vi` editor.

```
$ vi filetype.sh
#!/bin/bash

# Purpose: This file will determine if the pathname passed on
#          the command line is a file or a directory. Then it
#          will print information about the pathname.
#
# Scriptname: filetype.sh

if [ $# -ne 1 ]; then
    echo "Usage: filetype pathname"
    exit 1
fi

if [ -d $1 ]; then
    echo "$1 is a directory."
    echo "Following is a listing of the $1 directory."
    ls $1
    echo " "
    echo "The number of kilobytes used by directory $1 is: \c"
    du -s -k $1 | nawk '{print $1}'
```

```
else
    echo "$1 is a file."
    echo "The size of $1 in bytes is: \c"
    ls -l $1 | nawk '{print $5}'
    echo
    if [ -b $1 ]; then
            echo "$1 is a block special file."
    elif  [ -c $1 ]; then
            echo "$1 is a character special file."
    elif  [ -u $1 ]; then
            echo "$1 has the set-user-id permission."
    else
            echo "There is nothing special to say about file
$1."
    fi
fi

echo "Finished"
```

Note: Be careful about the usage of apostrophes or single quotes in this script; sometimes, they are represented differently on a system.

6. Change the file permission and test the `filetype.sh` script by executing it with a path name. In the following example, the script is executed for the `/usr/bin/crontab` file path.

```
$ chmod 777 filetype.sh
$ ./filetype.sh /usr/bin/crontab
/usr/bin/crontab is a file.
The size of /usr/bin/crontab in bytes is: 27496

/usr/bin/crontab has the set-user-id permission.
Finished
```

Practice 10-2: Modifying the `adduser` Script by Using Conditionals

Overview

In this practice, you modify the `adduser` script from Practice 8-2 titled "Modifying the `adduser` Script to Prompt for User Information" to create a new user based on multiple conditions.

Tasks

1. Close all terminal windows if required and open a new terminal window.

2. Switch to the `/opt/ora/labs/lab10` directory and copy the following files to the `solutions` directory:
   - `adduser.sh`
   - `mypasswd`
   - `currentuid`
   - `currentuid2`
   - `currentuid3`
   - `currentuid4`

```
$ cd /opt/ora/labs/lab10
$ cp adduser.sh mypasswd currentuid currentuid2 currentuid3
currentuid4 solutions/
$ chmod 777 solutions/*
```

   Note: Change file permissions after the copy operation as shown in the preceding step. You use these five files in the rest of the practice.

3. Before modifying the `adduser` script, list the command constructs to perform the following:
   a. Accept exactly one command-line argument, which is assumed to be the new user's name. Output a `Usage` message and exit the script immediately when the script is executed with more or less than one command-line argument.

```
if [ $# -ne 1 ]
then
        echo "Usage: adduser newusername"
        exit 1
fi
```

   b. Either assign the value of `$1` to the `name` variable or change the `grep` command to search for `$1`.

```
name=$1
```

   c. If the username already exists in the `mypasswd` file, print a message, print the actual `mypasswd` entry for the user, and then exit the script.

```
if grep "^$name:" ./mypasswd > /dev/null 2>&1
then
        echo "The username $1 is already in use. Here "
        echo "is the entry from the ./mypasswd file."
        echo
```

```
        grep "^$name:" ./mypasswd
        exit 2
fi
```

d. Numerically sort the contents of the `currentuid3` file and store the output in a file named `currentuid4`.

Note: This file is already copied to the `solutions` directory in the step 2.

```
sort -n ./currentuid3 > ./currentuid4
```

e. The last line in `currentuid4` has the maximum UID number that is currently in use. Increment that number by 1 for the new user.

```
lastuid=`sed -n '$p' ./currentuid4`
uid=`expr $lastuid + 1`
```

f. Print the new username and UID.

```
echo "The uid for $name is: $uid"
```

4. *Switch to the `solutions` directory and modify the `adduser.sh` file by using the `vi` command to include the command constructs listed in step 3, as highlighted in the following code:*

```
#!/bin/bash
# Purpose: To write a script to add user to the system.
# Name: adduser.sh

# Check for only a single command line argument
if [ $# -ne 1 ]
then
        echo "Usage: adduser newusername"
        exit 1
fi


# Assign the value of $1 to the variable name
name=$1

if grep "^$name:" ./mypasswd > /dev/null 2>&1
then
        echo "The username $1 is already in use. Here "
        echo "is the entry from the ./mypasswd file."
        echo
        grep "^$name:" ./mypasswd
        exit 2
fi


# Use nawk to extract all the UIDs into the temp file currentuid
```

```
nawk -F: '{print $3}' ./mypasswd > ./currentuid

# Use sed to remove all 1, 2, and 3 digit UIDs from currentuid,
and
# place the output in the temp file currentuid2
sed -e '/^.$/d' -e '/^..$/d' -e '/^...$/d'  ./currentuid >
./currentuid2
# Use sed to remove UIDs 60001, 60002, and 65534 from
# currentuid2,and place the output in the temp file currentuid3
sed -e '/^6000[12]$/d' -e '/^65534$/d'  ./currentuid2 >
./currentuid3



# Prompt for a group name and verify whether it exists in
/etc/group

echo "What primary group should $1 belong to? "
echo "Please specify the group name: \c"
read group junk



grep "^$group:" /etc/group > /dev/null  2>&1
if  [ $? -ne 0 ]
then
        echo "The group $group is not valid."
        echo
        exit 2
fi

echo "Good, now select a login shell for user $1"
cat << LOGINSHELLS
> bash
> csh
> ksh
> sh
LOGINSHELLS

read ushell

if [ $ushell = "bash" -o $ushell = "csh" -o  $ushell = "ksh" -o
$ushell = "sh" ]
then
        echo "Please enter the comment information for the
user:"
```

```
        read comment

        echo "The new user entry looks like this:"
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell"
        echo " "
        echo "New user $1 has been added to the ./mypasswd file"
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell" >>
./mypasswd
else
       echo "Sorry, that is not a valid shell, exiting..."
       exit 3

# Sort the UIDs in currentuid3 numerically
sort -n ./currentuid3 > ./currentuid4

# Calculate the next available UID
lastuid=`sed -n '$p' ./currentuid4`

uid=`expr $lastuid + 1`
echo "The uid for $name is: $uid"
```

5. Change file permissions to execute the `adduser.sh` script for a new user.

```
$ chmod 777 adduser.sh
$ ./adduser.sh user7
What primary group should belong to?
Please specify the group name: staff
Good, now select a login shell for user user7
> bash
> csh
> ksh
> sh
bash
Please enter the comment information for the user:
`
The new user entry looks like this:
user7:x::staff:test user:/export/home/user7:/bin/bash

New user user7 has been added to the ./mypasswd file
The uid for user7 is: 1005
```

Note: Observe that the script can create a new user, `user7`, and append the new user ID by 1. This value may vary between systems.

6. Close all the terminal windows.
   This completes the practice.

Summary: In this practice, you answered a few questions about the use of conditionals in a shell script. This was followed by the creation of a script named `filetype` that determined whether the path name is a directory or a file. In the subsequent task, you modified the existing `adduser` script to add more capabilities with the use of conditionals to create a new user.

Practices for Lesson 11: Loops

Chapter 11

Practices for Lesson 11: Loops

Practices Overview

In several scenarios, you may have a requirement to run a shell script repeatedly to execute a statement or group of statements until some specified condition is met. In addition, you may want to execute a group of statements for each element in a certain list. To address such scenarios, the shell provides the following looping constructs:

- `for` construct
- `while` construct
- `until` construct

In this practice, you perform the following tasks:

- Write a script with a `loop` construct.
- Modify scripts to use loops and menus

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
✓	The `sed` Editor
✓	The `nawk` Programming Language
✓	Interactive Scripts
✓	Variables and Positional Parameters
✓	Conditionals
	Loops
	Functions
	Traps

Practice 11-1: Using `for` Loops

Overview

In this practice, you write a script by using `for` loops that list all the device scripts from a specified `devices` directory.

Tasks

1. Close all terminal windows if required, open a new terminal window, and switch to the `root` role.

```
$ su -
Password:
Oracle Corporation   SunOS 5.11       11.1  September 2012
You have new mail.
#
```

2. Before starting the practice, change to the `/opt/ora/labs/lab11` directory.

```
# cd /opt/ora/labs/lab11

# cd solutions/
# pwd
/opt/ora/labs/lab11/solutions
```

Note: This is where you save your work.

3. List the command constructs to perform the following (**for Oracle Solaris users**):
   a. Output a `Usage` message and exit the script immediately when the script is executed without any command-line argument.

```
if [ $# -ne 0 ]; then
   echo "Usage: devscripts.sh "
   exit 1
fi
```

   b. Print all device scripts in the `/etc/devices` directory:

```
for dsript in /etc/devices/d*
do
```

   c. Print the script name and the inode number of the script.

```
print "The script name is $dscript"
print -n "The inode number is "
inode=`ls -i $dscript | nawk '{print $1}'`
print "$inode \n"
```

   d. Print all other files under `/etc` that have the same inode number.

```
print "Other files under /etc with the same inode number are: "
find /etc -inum $inode
```

4. Write a script name `devscripts.sh` by using the `vi` editor, to include all the commands listed in step 3.

```
# vi devscripts.sh

#!/bin/bash

# Purpose: This script will list all the device scripts in the
# /etc/devices directory. Additionally, it will report the inode
# number for each script, and then list all files under /etc
# with the same inode number.
#
# Scriptname: devscripts.sh

if [ $# -ne 0 ]; then
        echo "Usage: devscripts.sh "
        exit 1
fi

for dscript in /etc/devices/d*
do
        print "The script name is $dscript"
        print -n "The inode number is "
        inode=`ls -i $dscript | nawk '{print $1}'`
        print "$inode \n"
        print "Other files under /etc with the same inode number
are: "
        find /etc -inum $inode
        print "\n\n"
done

echo "Finished"
```

5. Change the file permissions and execute the script to:
   - List all the device scripts in the `/etc/devices` directory
   - Report the inode number of each script

- List all files under `/etc` with the same inode number

```
# chmod 777 devscripts.sh
# ./devscripts.sh
The script name is /etc/devices/devid_cache
The inode number is 147514

Other files under /etc with the same inode number are:
/etc/devices/devid_cache

The script name is /etc/devices/devname_cache
The inode number is 148255

Other files under /etc with the same inode number are:
/etc/devices/devname_cache

Finished
```

For Oracle Linux users:

6. Create a script that lists all the `start` scripts in the `/etc/init.d` directory. Report the inode number for each script, and then list all the files under `/etc` with the same inode number.

   Open the `vi` editor and enter the script as follows:

```
# vi rcscripts.sh

#!/bin/bash
# Purpose: This script will list all the start scripts in the
/etc/init.d
# directory. Additionally it will report the inode number for
# each script, and then list all files under /etc with the same
inode
# number.
#
# Scriptname: rcscripts.sh
# Output a usage message
if [ $# -ne 0 ]; then
echo "Usage: rcscripts.sh "
exit 1
fi
for rcscript in /etc/init.d/s*
do
echo "The script name is $rcscript"
echo -n "The inode number is "
```

```
        inode=`ls -i $rcscript | nawk '{print $1}'`
        echo "$inode \n"
        echo "Other files under /etc with the same inode number
are: "
        find /etc -inum $inode
        echo "\n\n"
done

echo "Finished"
```

7. Change the file permissions and execute the script to:
   - List all the `start` scripts in the `/etc/init.d` directory
   - Report the inode number of each script
   - List all the files under `/etc` with the same inode number

```
# chmod 777 rcscripts.sh
# ./rcscripts.sh
The script name is /etc/init.d/sandbox
The inode number is 396698 \n
Other files under /etc with the same inode number are:
/etc/rc.d/init.d/sandbox
\n\n
The script name is /etc/init.d/saslauthd
The inode number is 394955 \n
Other files under /etc with the same inode number are:
/etc/rc.d/init.d/saslauthd
\n\n
The script name is /etc/init.d/sblim-sfcb
The inode number is 400979 \n
Other files under /etc with the same inode number are:
/etc/rc.d/init.d/sblim-sfcb
…
…
…

<Output Omitted>
```

This completes this practice. Close all terminal windows.

Practice 11-2: Using Loops and Menus

Overview

In this practice, you write scripts that use loops and menus by modifying the `filetype` and `adduser` scripts that were created in the previous practices.

Tasks

1. Close all terminal windows if required and open a new terminal window.

2. Before beginning the practice, change to the `/opt/ora/labs/lab11` directory.

```
$ cd /opt/ora/labs/lab11
```

3. Copy the following files from the `/opt/ora/labs/lab11` directory to the `solutions` directory:
   - `filetype.sh`
   - `adduser.sh`
   - `currentuid`
   - `currentuid2`
   - `currentuid3`
   - `currentuid4`
   - `mypasswd`

```
$ cp filetype.sh adduser.sh currentuid currentuid2 currentuid3
currentuid4 mypasswd solutions/
$ chmod 777 solutions/*
```

   Note: Change file permissions after the copy operation as shown in the preceding step. You use these files in the rest of the practice.

4. *Modify the `filetype` script to introduce the `if` and `while` loop constructs.*

   Create a copy of the `filetype` script as `filetype2`. Replace the code from the previous version of the `filetype.sh` file as follows. List the command instructs to include the following options:

   a. Modify the code so that it accepts one or more path names on the command line and prints a `Usage` message if it receives no arguments.

```
if [ $# -eq 0 ]; then
        echo "Usage: filetype2.sh pathname [pathname2...]"
        exit 1
fi
```

b. Use a `loop` construct to test each path name entered on the command line to determine whether it is a file or a directory.

```
while [ $# -ne 0 ]
do
   if [ -d $1 ]; then
...
   else
      echo "$1 is a file."
...
shift

echo "\n\n"
done

echo "Finished"
```

5. *Write the complete solution by modifying the `filetype2.sh` file with the highlighted code. Remove extra code from the previous practice as follows:*

```
$ cd solutions
$ pwd
/opt/ora/labs/lab11/solutions
$ cp filetype.sh filetype2.sh

$ vi filetype2.sh

#!/bin/bash
# Purpose: This script will determine whether each pathname
# passed on the command line is a file or directory. Then it
# will print information about the pathname.
#
# Scriptname: filetype2.sh

if [ $# -eq 0 ]; then
        echo "Usage: filetype2.sh pathname [pathname2...]"
        exit 1
fi
while [ $# -ne 0 ]
do
        if [ -d $1 ]; then
                echo "$1 is a directory."
```

```
                echo "Following is a listing of the $1
directory."
                ls $1
                echo " "
                echo "The number of kilobytes used by directory
$1 is: \c"
                du -s -k $1 | nawk '{print $1}'
        else
                echo "$1 is a file."
                echo "The size of $1 in bytes is: \c"
                ls -l $1 | nawk '{print $5}'
                echo
                if [ -b $1 ]; then
                        echo "$1 is a block special file."
                elif  [ -c $1 ]; then
                        echo "$1 is a character special file."
                elif  [ -u $1 ]; then
                        echo "$1 has the set-user-id
permission."
                else
                        echo "There is nothing special to say
about file $1
."
                fi
        fi
        shift
        echo "\n\n"
done

echo "Finished"
```

Note: The modified code is highlighted in red.

6. Change the file permission and execute the script to determine whether the path names passed on the command line are a file or a directory and print information about the path name.

```
$ chmod 777 filetype2.sh
$ ./filetype2.sh /usr/bin/crontab /usr/bin/grep
/usr/bin/crontab is a file.
The size of /usr/bin/crontab in bytes is: 27496

/usr/bin/crontab has the set-user-id permission.
```

```
/usr/bin/grep is a file.
The size of /usr/bin/grep in bytes is: 13248

There is nothing special to say about file /usr/bin/grep.



Finished
```

7. Modify the `adduser` script to introduce menu options.

   List the command constructs to modify the script and perform the following:

   Note: For Oracle Linux, replace the `print` command with the `echo` command.

   a. Create a menu for the list of available shells that you want to offer as login shells.

```
select ushell in bash csh ksh sh
```

   b. When the user selects a value that is not in the menu, print a message stating that it is an unknown choice, and then force the user to re-enter a value until the user enters a correct menu choice.

```
*)
        print "Unknown choice, try again."
        ;;
esac
```

   c. When the user selects a valid shell from the menu, print a verification of the selected shell. Then, allow the program to continue prompting the user for the comment-field information (which was written in Practice 11-1: Using `for` Loops).

```
select ushell in bash csh ksh sh
do
       case $ushell in
       bash)
                print "$ushell was selected."
                break
                ;;

       csh)
                print "$ushell was selected."
                break
                ;;

        ksh)
                print "$ushell was selected."
                break
                ;;
```

```
        sh)
                print "$ushell was selected."
                break
                ;;

        *)
                print "Unknown choice, try again."
                ;;
        esac
done
```

8. *Using the `vi` editor, modify the script `adduser.sh` to include the command constructs listed in step 6 as highlighted in the following code:*

```
$ vi adduser.sh
#!/bin/bash
# Purpose: To write a script to add user to the system.
# Name: adduser.sh

# Check for only a single command line argument
if [ $# -ne 1 ]
then
        echo "Usage: adduser newusername"
        exit 1
fi


# Assign the value of $1 to the variable name
name=$1

if grep "^$name:" ./mypasswd > /dev/null 2>&1
then
        echo "The username $1 is already in use. Here "
        echo "is the entry from the ./mypasswd file."
        echo
        grep "^$name:" ./mypasswd
        exit 2
fi

# Use nawk to extract all the UIDs into the temp file currentuid
nawk -F: '{print $3}' ./mypasswd > ./currentuid
```

```
# Use sed to remove all 1, 2, and 3 digit UIDs from currentuid,
and
# place the output in the temp file currentuid2
sed -e '/^.$/d' -e '/^..$/d' -e '/^...$/d'  ./currentuid >
./currentuid2


# Use sed to remove UIDs 60001, 60002, and 65534 from
currentuid2,
# and place the output in the temp file currentuid3
sed -e '/^6000[12]$/d' -e '/^65534$/d'  ./currentuid2 >
./currentuid3


# Prompt for a group name and verify whether it exists in
/etc/group

echo "What primary group should $1 belong to? "
echo "Please specify the group name: \c"
read group junk


grep "^$group:" /etc/group > /dev/null  2>&1
if  [ $? -ne 0 ]
then
        echo "The group $group is not valid."
        echo
        exit 2
fi

echo "Good, now select a login shell for user $1"

PS3="Enter the number: "
select ushell in bash csh ksh sh
do
        case $ushell in
        bash)
                print "$ushell was selected."
                break
                ;;

        csh)
```

```
                print "$ushell was selected."
                break
                ;;

        ksh)
print "$ushell was selected."
                break
                ;;

        sh)
                print "$ushell was selected."
                break
                ;;

        *)
                print "Unknown choice, try again."
                ;;
        esac
done

# Note: comment these entries
# echo "Good, now select a login shell for user $1"
# cat << LOGINSHELLS
# > bash
# > csh
# > ksh
# > sh
# LOGINSHELLS

# read ushell

 if [ $ushell = "bash" -o $ushell = "csh" -o  $ushell = "ksh" -o
$ushell = "sh" ]
 then
        echo "Please enter the comment information for the
user:"
read comment

        echo "The new user entry looks like this:"
```

```
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell"
        echo " "
        echo "New user $1 has been added to the ./mypasswd file"
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell" >>
./mypasswd
else
       echo "Sorry, that is not a valid shell, exiting..."
       exit 3
fi

# Sort the UIDs in currentuid3 numerically
sort -n ./currentuid3 > ./currentuid4

# Calculate the next available UID
lastuid=`sed -n '$p' ./currentuid4`

uid=`expr $lastuid + 1`
echo "The uid for $name is: $uid"
```

Note: The comment entries are in highlighted and bold.

9. Change file permissions and execute the `adduser.sh` script to confirm whether the new user selection by using menu options is available.

```
$ chmod 777 adduser.sh
$ ./adduser.sh user10
What primary group should user10 belong to?
Please specify the group name: staff
Good, now select a login shell for user user10
1) bash
2) csh
3) ksh
4) sh
Enter the number: 1
bash was selected.
Please enter the comment information for the user:
This is a test user
The new user entry looks like this:
user10:x::staff:This is a test
user:/export/home/user10:/bin/bash

New user user10 has been added to the ./mypasswd file
The uid for user10 is: 1004
```

10. View the `mypasswd` file to confirm user registration.

```
$ cat mypasswd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp
Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
user1:x:1001:10::/export/home/user1:/bin/sh
user2:x:1002:10::/export/home/user2:/bin/ksh
user3:x:1003:nobody:This is a test
user:/export/home/user3:/bin/bash
user3:x::nobody:this is a test user:/export/home/user3:/bin/bash
user3:x::nobody:test user n:/export/home/user3:/bin/bash
user4:x::staff:This is another test
user:/export/home/user4:/bin/bash
user10:x::staff:This is a test user:/export/home/user10:/bin/bash
```

Note: The preceding output will vary between systems.

11. Use the `filetype2` script and show usage of the `while` loop with the `until` loop by modifying it with a new name `filetype3.sh`.
    a. Copy the `filetype2` script to `filetype3` and modify the `filetype3` script as follows:
        1) Where it uses a `while` loop, change the code to use an `until` loop.
        2) Where it uses an `until` loop, change the code to use a `while` loop.

    Change:

```
while [ $# -ne 0 ]
To:
until [ $# = 0 ]
```

12. *Copy the file `filetype2.sh` and create a new file `filetype3.sh`. Using the `vi` editor, modify the script `adduser.sh` to include the command constructs listed in step 10 as highlighted in the following code:*

```
$ cp filetype2.sh filetype3.sh
$ vi filetype3.sh
```

```
#!/bin/bash

# Purpose: This script will determine whether each pathname
# passed on the command line is a file or a directory. Then it
will print information about the pathname.
#
# Scriptname: filetype3.sh
if [ $# -eq 0 ]; then
     echo "Usage: filetype3.sh pathname [pathname2...]"
     exit 1
fi
until [ $# = 0 ]
do
     if [ -d $1 ]; then
     echo "$1 is a directory."
     echo "Following is a listing of the $1 directory."
     ls $1
     echo " "
     echo "The number of kilobytes used by directory $1 is:
\c"
du -s -k $1 | nawk '{print $1}'
     else
     echo "$1 is a file."
     echo "The size of $1 in bytes is: \c"
     ls -l $1 | nawk '{print $5}'
     echo
if [ -b $1 ]; then
     echo "$1 is s block special file."
     elif [ -c $1 ]; then
     echo "$1 is s character special file."
elif [ -u $1 ]; then
     echo "$1 has the set-user-id permission."
else
     echo "There is nothing special to say about file
$1."
fi
fi
     shift
     echo "\n\n"
done
echo "Finished"
```

```
:wq
```

13. Change file permissions and execute the script to determine whether the path names passed on the command line are a file or a directory and print information about the path name.

```
$ chmod 777 filetype3.sh
$ ./filetype3.sh /usr/bin/crontab /usr/bin/grep
/usr/bin/crontab is a file.
The size of /usr/bin/crontab in bytes is:
/usr/bin/crontab has the set-user-id permission.



/usr/bin/grep is a file.
The size of /usr/bin/grep in bytes is:
There is nothing special to say about file
/usr/bin/grep.



Finished
```

Observation: You can use replace the `while` loop with the `until` loop with similar results. This completes this practice.

Summary: In this practice, you developed advanced scripts by using loops and menus.

Practices for Lesson 12: Functions

Chapter 12

Practices for Lesson 12: Functions

Practices Overview

In a shell script, a function is a set of one or more statements that act as a complete routine. Each function must have a unique name within that shell or shell script.

In this practice, you create functions and use them in shell scripts.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
✓	The `sed` Editor
✓	The `nawk` Programming Language
✓	Interactive Scripts
✓	Variables and Positional Parameters
✓	Conditionals
✓	Loops
	Functions
	Traps

Practice 12-1: Using Functions

Overview

In this practice, you write shell scripts that include or use functions.

Tasks

1. Close all terminal windows if required and open a new terminal window.

2. Before starting the practice, change to the `/opt/ora/labs/lab12` directory. Switch to the `solutions` directory.

```
$ cd /opt/ora/labs/lab12
$ cd solutions
$ pwd
/opt/ora/labs/lab12/solutions
```

   Note: This is where you save your work.

3. Create a directory named `funcs`.

```
$ mkdir funcs
```

   Note: This directory stores the functions that are used in the script.

4. Create a function file named `parsepath` and save it in the `funcs` directory. This function does the parsing, and prints the path names.

```
$ cd funcs
$ vi parsepath
function parsepath
{
   name=$1

   echo
   echo "The pathname is: $name"

   while [[ -n $name ]]
   do
      echo $name
      name=${name%/*}
   done

   echo
}

:wq
```

5. Switch back to the `solutions` directory and create a script file called `firstfunc.sh`. Include the function definition from step 4. List the programming constructs to include the following options in the script file:

a. Outputs a `Usage` message if any arguments are received and exits the script immediately if the script is executed without any command-line argument

```
# Purpose: To parse a long pathname.
#
# Name: firstfunc.sh

if [ $# -ne 0 ]; then
   echo "Usage: firstfunc.sh"
   exit 1
fi
```

b. Prompts the user for a path name, reads the path name, calls the function, and passes the path name to the function. Perform this task at least two or three times by using a `loop` construct.

```
echo -n "Enter a pathname and I will parse it: [Control-D to
quit]: "
read pathname junk

while [[ -n $pathname ]]
do
   parsepath $pathname

   # Enter a control-D to quit
   echo -n "Enter a pathname and I will parse it: [Control-D to
quit]: "

   read pathname junk
done
```

6. *Now write the shell script, `firstfunc.sh`, in the `solutions` directory by using the `vi` editor and include the information from step 5.*

```
$ cd ..
$ pwd
/opt/ora/labs/lab12/solutions



$ vi firstfunc.sh

#!/bin/bash

# Purpose: To parse a long pathname.
#
# Name: firstfunc.sh

# parsepath function definition
```

```
function parsepath
{
   name=$1

   echo
   echo "The pathname is: $name"

   while [[ -n $name ]]
   do
      echo $name
      name=${name%/*}
   done

   echo
}

# Output a Usage message
if [ $# -ne 0 ]; then
       echo "Usage: firstfunc.sh"
     exit 1
 fi

# Prompts the user for a path name, reads the path name, calls
# the function, and passes the path name to the function.

echo -n "Enter a pathname and I will parse it: [Control-D to
quit]: "
read pathname junk

while [[ -n $pathname ]]
do
   parsepath $pathname

   # Enter a control-D to quit
   echo -n "Enter a pathname and I will parse it: [Control-D to
quit]: "

   read pathname junk
done
```

7. Change file permissions and execute the `firstfunc.sh` script.

```
$ chmod 777 firstfunc.sh
$ ./firstfunc.sh
```

```
Enter a pathname and I will parse it: [Control-D to quit]:
/opt/ora/labs

The pathname is: /opt/ora/labs
/opt/ora/labs
/opt/ora
/opt

Enter a pathname and I will parse it: [Control-D to quit]: ^D
```

8. Export the function `parsepath` in the `secondfunc.sh` script.
   a. Create a copy of the `firstfunc.sh` script with the name `secondfunc.sh`.

```
$ pwd
/opt/ora/labs/lab12/solutions
$ cp firstfunc.sh secondfunc.sh
```

   b. In the script, `secondfunc.sh`, export the function `parsepath` with the following declaration:

```
export -f parsepath
```

   c. *Comment the "Output a Usage message" section in the code.*

```
# Output a Usage message
# if [ $# -ne 0 ]; then
#   echo "Usage: firstfunc.sh"
#   exit 1
# fi
```

9. *Edit the `secondfunc.sh` script by using the `vi` editor to perform the tasks listed in step 8.*
   ***Note:** Updated entries are highlighted in red.*

```
$ vi secondfunc.sh

#!/bin/bash
# Purpose: To parse a long pathname.
#
# Name: secondfunc.sh

# parsepath function definition
function parsepath
{
   name=$1

   echo
   echo "The pathname is: $name"

   while [[ -n $name ]]
```

```
   do
      echo $name
      name=${name%/*}
   done

   echo
}

# bash uses export -f command to export functions.
export -f parsepath

# Output a Usage message
# if [ $# -ne 0 ]; then
#    echo "Usage: firstfunc.sh"
#    exit 1
# fi

# Prompts the user for a path name, reads the path name, calls
# the function, and passes the path name to the function.

echo -n "Enter a pathname and I will parse it: [Control-D to
quit]: "
read pathname junk

while [[ -n $pathname ]]
do
   parsepath $pathname

   # Enter a control-D to quit
   echo -n "Enter a pathname and I will parse it: [Control-D to
quit]: "

   read pathname junk
done

:wq
```

10. Change file permissions and execute the `secondfunc.sh` script so that the script reads the specified directory. Access the `parsepath` function by using `export` command.

    Note: You can use any directory path of your choice. The following example uses the `/opt/ora/labs` and `/etc/passwd` directories.

```
$ chmod 777 secondfunc.sh
$ ./secondfunc.sh
Enter a pathname and I will parse it: [Control-D to quit]:
/opt/ora/labs

The pathname is: /opt/ora/labs
/opt/ora/labs
/opt/ora
/opt

Enter a pathname and I will parse it: [Control-D to quit]:
/etc/passwd

The pathname is: /etc/passwd
/etc/passwd
/etc

Enter a pathname and I will parse it: [Control-D to quit]:
oracle@s11-server:/opt/ora/labs/lab12/solutions$
```

This completes this practice.

Summary: In this practice, you created functions and exported them in shell scripts.

Practices for Lesson 13: Traps

Chapter 13

Practices for Lesson 13: Traps

Practices Overview

In UNIX, a signal is a message to inform that some abnormal event has taken place or to request another process to perform a task. The signal is sent from one process to another process. You use the `kill` command to send signals to processes.

In this practice, you write a script to prevent any user from immediately terminating a process by using `Control-C`, `Control-\`, or `Control-Z` because the script may still have some pending tasks to perform. For example, the script may need to do some clean-up before exiting. To avoid having the script exit before the clean-up is performed, you use the `trap` statement in your script to catch these signals.

In this practice, you modify a shell script to trap the `Control-C` signal.

The following checklist shows your progress through the practices:

	Shell Programming Checklist
✓	Introduction
✓	UNIX Shells
✓	Shell Scripting
✓	Shell Environment
✓	Pattern Matching
✓	The `sed` Editor
✓	The `nawk` Programming Language
✓	Interactive Scripts
✓	Variables and Positional Parameters
✓	Conditionals
✓	Loops
✓	Functions
	Traps

Practice 13-1: Using Traps

Overview

In this practice, you modify the `adduser` shell script, which you modified in Practice 11-2 titled "Using Loops and Menus," to trap the `Control-C` signal in this practice.

Tasks

1. Close all terminal windows if required and open a new terminal window.
   Switch to the `/opt/ora/labs/lab13` directory and copy the following files to the `solutions` directory:
   - `adduser.sh`
   - `mypasswd`
   - `currentuid`
   - `currentuid2`
   - `currentuid3`
   - `currentuid4`

```
$ cd /opt/ora/labs/lab13
$ cp adduser.sh mypasswd currentuid currentuid2 currentuid3
currentuid4 solutions/
$ chmod 777 solutions/*
```

   Note: Change file permissions after the copy operation as shown in the preceding step. You use these files in the rest of the practice.

2. Change to the `solution` directory and execute the `adduser.sh` script with a new username. Press `Control-C` at the shell menu prompt.

```
$ cd solutions/
$ pwd
/opt/ora/labs/lab13/solutions

$ ./adduser.sh michael
What primary group should michael belong to?
Please specify the group name: staff
Good, now select a login shell for user michael
1) bash
2) csh
3) ksh
4) sh
Enter the number: 1
bash was selected.
Please enter the comment information for the user:
^C
```

   Observation: The `kill` signal sent by `CTRL-C` (`^C`) exits you from the script and the prompt returns.

3. *Now, add the following `trap` statement to the `adduser.sh` script by using the `vi` editor to catch the INT signal, so that the user cannot terminate with that signal. The `trap` should notify the user that `CTRL-C` is not available. Place the `trap` statement after the test for the command-line arguments.*

```
trap 'banner "No Control-C"' 2
```

```
$ vi adduser.sh

#!/bin/bash

# Purpose: To write a script to add user to the system.
# Name: adduser.sh

# Check for only a single command line argument
if [ $# -ne 1 ]
then
        echo "Usage: adduser newusername"
        exit 1
fi

# Add a trap for Control-C
trap 'banner "No Control-C" ' 2

# Assign the value of $1 to the variable name
name=$1

if grep "^$name:" ./mypasswd > /dev/null 2>&1
then
        echo "The username $1 is already in use. Here "
        echo "is the entry from the ./mypasswd file."
        echo
        grep "^$name:" ./mypasswd
        exit 2
fi

# Use nawk to extract all the UIDs into the temp file currentuid
nawk -F: '{print $3}' ./mypasswd > ./currentuid

# Use sed to remove all 1, 2, and 3 digit UIDs from currentuid,
and
# place the output in the temp file currentuid2
```

```
sed -e '/^.$/d' -e '/^..$/d' -e '/^...$/d'  ./currentuid >
./currentuid2



# Use sed to remove UIDs 60001, 60002, and 65534 from
currentuid2,
# and place the output in the temp file currentuid3
sed -e '/^6000[12]$/d' -e '/^65534$/d'  ./currentuid2 >
./currentuid3



# Prompt for a group name and verify whether it exists in
/etc/group

echo "What primary group should $1 belong to? "
echo "Please specify the group name: \c"
read group junk



grep "^$group:" /etc/group > /dev/null  2>&1
if  [ $? -ne 0 ]
then
        echo "The group $group is not valid."
        echo
        exit 2
fi

echo "Good, now select a login shell for user $1"

PS3="Enter the number: "
select ushell in bash csh ksh sh
do
        case $ushell in
        bash)
                print "$ushell was selected."
                break
                ;;

        csh)
                print "$ushell was selected."
                break
                ;;
```

```
        ksh)
print "$ushell was selected."
                break
                ;;

        sh)
                print "$ushell was selected."
                break
                ;;

        *)
                print "Unknown choice, try again."
                ;;
        esac
done

# Note: comment these entries
# echo "Good, now select a login shell for user $1"
# cat << LOGINSHELLS
# > bash
# > csh
# > ksh
# > sh
# LOGINSHELLS

# read ushell

 if [ $ushell = "bash" -o $ushell = "csh" -o  $ushell = "ksh" -o
$ushell = "sh" ]
 then
        echo "Please enter the comment information for the
user:"
read comment

        echo "The new user entry looks like this:"
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell"
        echo " "
        echo "New user $1 has been added to the ./mypasswd file"
        echo
"$1:x:$uid:$group:$comment:/export/home/$1:/bin/$ushell" >>
./mypasswd
else
```

```
        echo "Sorry, that is not a valid shell, exiting..."
        exit 3
fi

# Sort the UIDs in currentuid3 numerically
sort -n ./currentuid3 > ./currentuid4

# Calculate the next available UID
lastuid=`sed -n '$p' ./currentuid4`

uid=`expr $lastuid + 1`
echo "The uid for $name is: $uid"
```

4. Change the file permission to execute the `adduser` script with a new username and press `CTRL-C` at the shell menu prompt, which triggers a notification that `CTRL-C` is disabled.

```
$ ./adduser.sh michael
What primary group should michael belong to?
Please specify the group name: staff
1) bash
2) csh
3) ksh
4) sh
Enter the number: ^C#      #                #####
##    #   ####         #     #   ####   #    #   #####  #####    ####   #
# #   #  #    #        #        #    #  ##   #     #    #    #  #    #  #
#  #  #  #    #        #        #    #  # #  #     #    #    #  #    #  #
#   # #  #    #        #        #    #  #  # #     #    #####   #    #  #
#    ##  #    #        #     #  #    #  #   ##     #    #   #   #    #  #
#     #   ####          #####    ####   #    #     #    #    #   ####   ######
```

Note: Press `CTRL-Z` to come out of the script execution.

```
Enter the number: ^Z
[1]+  Stopped                 ./adduser.sh Michael
```

Observation: The `trap` statement that was added to the `adduser.sh` script ensures that the `kill` signal sent by `CTRL-C` is not executed and instead displays a banner as NO CONTROL. The banner message indicates that `CTRL-C` cannot kill the script because the termination signal is trapped.

This completes this practice.

Summary: In this practice, you modified an existing shell script to trap the termination signal.