

## Linux OS



Operating Systems Administration  
by  
Prof. Ed Carr

## Systems Administration



### Policy and Procedures:

- Installation
- Configuration
- Security
- User management
- Customization
- Performance tuning
- Troubleshooting



## Systems Administration



Setup, Configuration and Customization requires modifying many types of scripts (setup, configuration, customization, etc...)

To edit these files a basic editor application is required:

**vi** very common editor found on most Unix operating systems as well as Linux versions

**nano** a free version of **pico**. Very simple and easy to use interface.

## Scripts



Mainstay of Systems Administration

Purpose of scripts to automate administrative tasks.

Modification of common shell scripts. Example of common user script `.bashrc`

## Typical User .bashrc



```
# .bashrc

# User specific aliases and functions
alias startx='echo not available'
alias pine='echo not available'
alias mail='echo not available'
alias mutt='echo not available'
alias lpr='echo not available'
alias pico=/usr/bin/nano

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

## Bash (Bourne Again Shell)



- **Bash** is a free software Unix shell
- Written for the GNU Project.
- Its name is an acronym which stands for *Bourne-again shell*.
- The name is a pun on the name of the Bourne shell (sh), an early and important Unix shell written by Stephen Bourne and distributed with Version 7 Unix circa 1978, and "born again".
- Bash was created in 1987 by Brian Fox.
- Bash is the shell for the GNU operating system from the GNU Project.
- It is the default shell on most systems built on top of the Linux kernel as well as on Mac OS X and Darwin.

## Bash Features



Bash can perform:

- Integer calculations
- I/O redirection
- Brace expansion

```
# This is a bash-specific feature echo
a{p,c}e # ape ace
```

## First Example



```
#!/bin/bash
# declare STRING variable
STRING="Hello World"
#print variable on a screen
echo $STRING
```

## shebang:"#!"



Every bash shell script in these notes starts with **shebang:"#!"** which is not read as a comment. First line is also a place where you put your interpreter which is in this case: `/bin/bash`

## Run your scripts



After placing the **shebang:"#! "** in your code, we will make them executable by typing the following command:

```
[user@linux ~]$ chmod +x <script name>.sh
```

To execute the command we simply type from the command line:

```
[user@linux ~]$ ./<script name>.sh
```

## Simple Backup



```
#!/bin/bash
```

```
tar -czf myhome_directory.tar.gz /home/linuxconfig
```

*“tar has to be the worst utility in the world.”*

Ken Williams

## Bash Comparisons



- Arithmetic Comparisons

```
-lt <
```

```
-gt >
```

```
-le <=
```

```
-ge >=
```

```
-eq ==
```

```
-ne !=
```

## Comparisons Continued



```
#!/bin/bash
# declare integers
NUM1=2
NUM2=2
if [ $NUM1 -eq $NUM2 ]; then
echo "Both Values are equal"
else
echo "Values are NOT equal"
fi
```

## Bash while loop



```
#!/bin/bash
COUNT=6
# bash while loop
while [ $COUNT -gt 0 ]; do
echo Value of count is: $COUNT
let COUNT=COUNT-1
done
```

## Bash for loop



```
#!/bin/bash

# bash for loop
for f in $(ls ~); do
echo $f
done
```

## Arrays



```
#!/bin/bash
#Declare array with 4 elements
ARRAY=( 'Debian Linux' 'Redhat Linux'
Ubuntu Linux )
# get number of elements in the array
ELEMENTS=${#ARRAY[@]}

# echo each element in array
# for loop
for (( i=0;i<$ELEMENTS;i++)); do
echo ${ARRAY[$i]}
done
```

## Z Shell



The **Z shell (zsh)** is a Unix shell that can be used as an interactive login shell and as a powerful command interpreter for shell scripting. Zsh can be thought of as an extended Bourne shell with a large number of improvements, including some of the most useful features of bash, ksh, and tcsh.

## Z Shell Example



```
#!/bin/sh
while read first last email
do
    echo $first $last 'Hello
COMP 590' | mail $email
done < 590mail.txt
```

## Automated Administration



The typical system administrator spends a lot of time doing repetitive tasks. At least they will if they don't have a task scheduling system that automatically runs various tasks for them at suitable points in time.

## Scheduling one-time execution



- The cron system handles all of the time-based scheduling of commands and provides two different solutions for running commands at a specific time.
- The `at` command schedules work for a specific time to be executed once.
- The `crontab` system enables you to specify a schedule for the execution of the command, either at specified times, on specific days, or a combination of the two.

## Scheduling regular executions



- Regular execution is handled by setting up a cron table (called a `crontab`) that defines the interval and sequence for each command. The format of the file is a single line for each command (with six fields):

```
- minute hour day month dayofweek command
```

## time specification



- Minute: 0-59
- Hour: 0-23
- Day: 1-31
- Month: 1-12
- Day: 0-6 (where 0 is Sunday)