Introduction to Open Source, Linux and OpenOffice.org

Jigish Gohil
CyberOrg Info
cyberorg@opensuse.org

License

Copyright (C) 2009 CyberOrg Info

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

Introduction

Agenda

- What are Open Source softwares?
- Advantages of using open source softwares including Openoffice.org.
- Components of Open Source desktop machine.
- Introduction to Linux Operating system with CLI and GUI.
- Some basic commands of Linux related to directory, files, users etc.
- Use of an Editor - VI editor.
- Simple shell scripting.

© CyberOrg Info

Open Source Software

Agenda

- Overview of Open Source
  - what is Open Source?
  - What it isn't
  - Examples
  - Licenses
  - The Community
- History of Linux
  - Birth and Lineage
  - The Kernel and key components
  - Common Distributions

© CyberOrg Info

Feel the *Force* of Open Source!



- 72% of firms interviewed plan to use more open source
- Cost is the primary benefit for 68% of interviewees
- 80% say that the SCO lawsuit has no impact on their plans

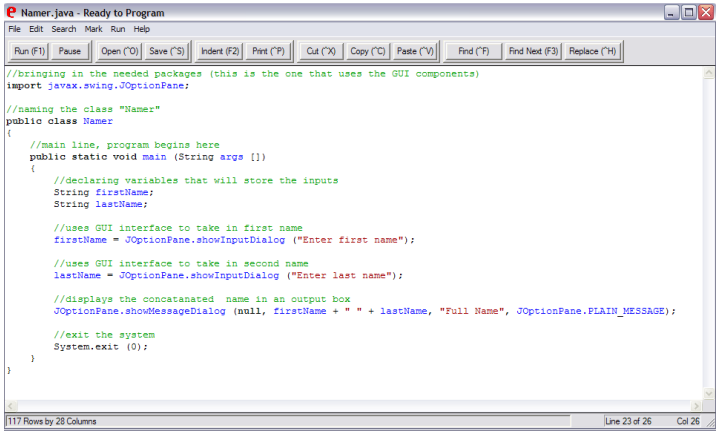
Source: http://www.forrester.com/ER/Research/Report/Summary/0,1338,17096,00.html

© CyberOrg Info

How Does Software Get Created?

The term **'software'** refers to two different but related things:

Source code - a set of human readable and understandable instructions that comprise the 'recipe' from which an executable program can be made



Object code - the actual executable program which is compiled of machine readable source code. It is fed into a computer's microprocessor to perform various operations



© CyberOrg Info

What is Open Source Software?



Open Source Software (OSS) is any software whose code is available for users to look at and modify freely

- Web definition

Linux is the best known example of Open Source Software



© CyberOrg Info

OSS is not a “what,” but a “how”

Open Source Software (OSS) is a methodology, a process:

- Allows contributions from anywhere and anyone
- Thorough reviews by skilled developers/programmers
- Bug corrections and patches released frequently
- The stability and reliability of these products are **due to** the “chaotic” environment in which they are produced

OSS is NOT:

- Linux
- A license
- Sourceforge.org

© CyberOrg Info

Open Source
the Basic Idea

"The basic idea behind open source is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing."



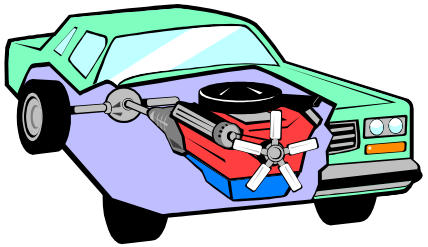
Open Source Initiative organization

(OSI – http://www.opensource.org)

© CyberOrg Info

Proprietary Software vs. Open Source

Software What's the difference?



Open Source Software:

- Free and open discussion
- Results are free to the entire community
- Voluntary contributions through code/documentation/testing
- Improvements and new functionalities are delivered to users

A commonly used analogy:

"Proprietary software is like buying a car with the hood welded shut……
…whereas Open Source Software allows you to play with the engine!"

© CyberOrg Info

Open Source Software



Advantages:

- Access to Source code
- Quick evolution
- Rapid development
- Security
- Openness and trust
- Standards compliance
- Low Cost
- Great community spirit
- Peer review = higher quality



Disadvantages:

- Intellectual property issues (patents on algorithms...)
- Difficult to gather information about a product
- Potential lack of support and consulting
- Too much choice?
- Challenging to manage so much change so fast

© CyberOrg Info

Examples of OS Projects











- **OpenOffice:** Multi-platform office productivity suite compatible with all major file formats
- **Apache:** enormous foundation supported projects with many sub-projects and extensive corporate involvement
- **MySQL:** Actively developed by a company specializing in it, making money by offering support services
- **MONO:** Open source Implementation of .Net on Linux, UNIX & Windows
- **GNOME:** Open source graphical user interface for UNIX, Linux and other free systems

© CyberOrg Info

Open Source
Does Not Always Mean “Free”

“**Open Source**” and “**Free**”

- Often used in the same sentence
- The term “free” means free(dom) to modify, free(dom) to re-distribute
- All Open Source projects are governed by some type of end-user license agreement
- Free(dom) to distribute software together with the source code
- Concurrent development around the globe and quick improvement/bug fixing (evolution)

© CyberOrg Info

License Agreements



- All Open Source code is protected by some type of license agreement
- Defines the condition under which the source code may be used and obligations with respect to enhancements and derivative works
- Some allow free and open use of the software source code – others are much more restrictive
- Most commonly used – GPL, LGPL, BSD



List of approved license agreements http://www.opensource.org/licenses/

© CyberOrg Info

Open Source License Agreements



GNU General Public License



GNU Lesser General Public License



Berkeley Software Distribution

List of approved license agreements http://www.opensource.org/licenses/

© CyberOrg Info

Rates of Kernel Change

- first six months of 2.4 devel: -220,000 lines, +600,000 lines
- first six months of 2.6 devel: -600,000 lines, +900,000 lines

  1.5M lines changed in a 6.2M line tree

  64 MB diff in six months - and that's the stable kernel
- Current 2.6.11 -> 2.6.12-rc4 (10 weeks): 729 K lines, 22 MB diff
- Current 2.6.12-rc4-mm1 patchset: 414 K lines, 13 MB diff

© CyberOrg Info

The Cathedral and the Bazaar

Well-known paper by Eric S. Raymond about software development models:

- ***Cathedral:*** Traditional style, small teams, clear design & functionality before development starts, few release cycles, not much external feedback
- ***Bazaar:*** Maintainer releases functionality, frequent release cycles, a lot of feedback from the beginning of the process, anyone/anywhere can contribute

http://catb.org/~esr/writings/cathedral-bazaar/

© CyberOrg Info

Open Source Community



Richard Stallman: Founder, Free Software Foundation



Linus Torvalds: Creator/ Maintainer of Linux kernel



Brad Nicholes: Novell Senior Software Engineer and Member of the Apache Software Foundation



Bruce Perens: Independent Consultant (author of The Open Source Definition)



Miguel de Icaza: Founder of Mono project and key Gnome developer

Unwashed 

developers!



Eric Raymond: Activist & President, Open Source Institute



Monty Widenius: Founder and key developer of MySQL



Nat Friedman: Co-founder of Ximian and key Gnome developer



Jon "Mad Dog" Hall: Founder Executive Director of Linux International

© CyberOrg Info

Attributes of OS Community

Open Source Community



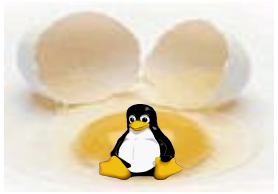
- Highly skilled and experienced in creating stable, quality applications
- 58%* of them are professional IT programmers or administrators
- 11 years of professional experience
- Motivated by need to rapidly respond to problems at work
- Position in community based on merit
- Anti-proprietary attitude

* Study conducted by the Boston Consulting Group

© CyberOrg Info

History of Linux

The Birth of the Kernel





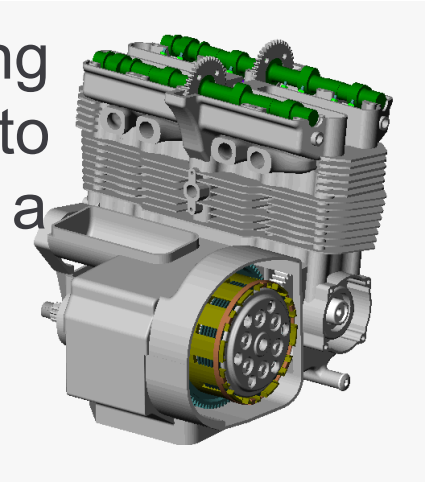


- (1991) Linus Torvalds begins writing an operating system similar to UNIX.
- (Six months later…) Linus posts his work on the Internet for anyone else interested in the work he was doing
- After 3 years of patches, testing version 1.0 of the Linux kernel is released
- Present Day: the latest stable version is at 2.6.16

© CyberOrg Info

What is a Kernel?

A Kernel is the central core of the operating system (OS) that provides basic services to other parts of the system. Linux began as a kernel, not a complete OS.



- Linux kernels with even numbers are stable; Linux kernels with odd numbers are beta
- The current full-featured version is 2.4.32 and development continues on an on-going, worldwide basis.
- The next stable version of Linux is 2.6.16

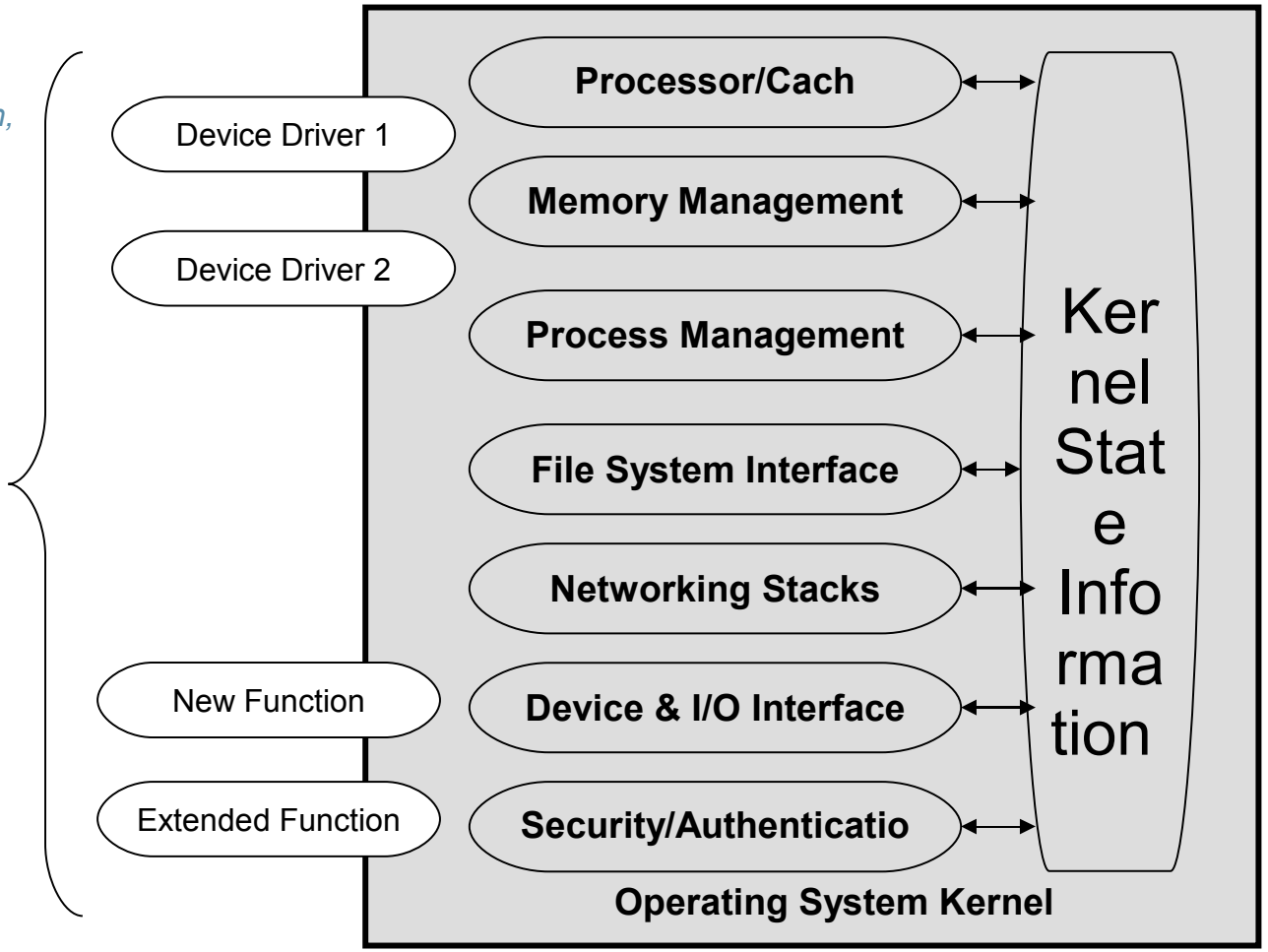
© CyberOrg Info

Key Components of a Kernel

Linux is a monolithic kernel

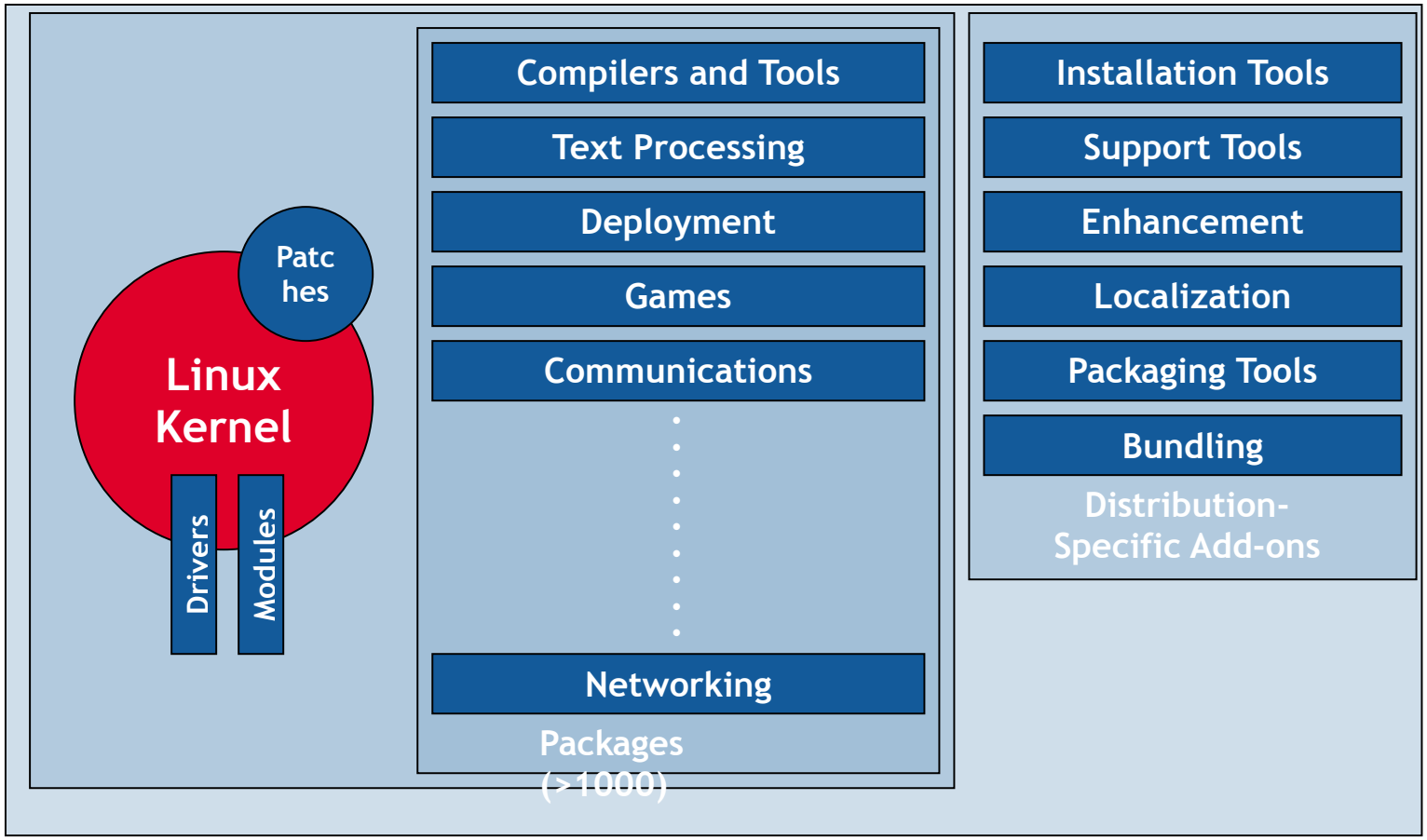
Think of device drivers as steering wheel, navigation, stereo, etc. of a car.

Kernel Modules



© CyberOrg Info

Example of a Linux Distribution



Source: The Business and Economics of Linux and Open Source - *Martin Fink*

© CyberOrg Info

Common Distributions



Founded 1992

Strongest in Europe

YaST configuration tool



Founded 1994

Widely deployed

RPM package manager



Founded 1998

Focus on desktop

Easy to install/ use

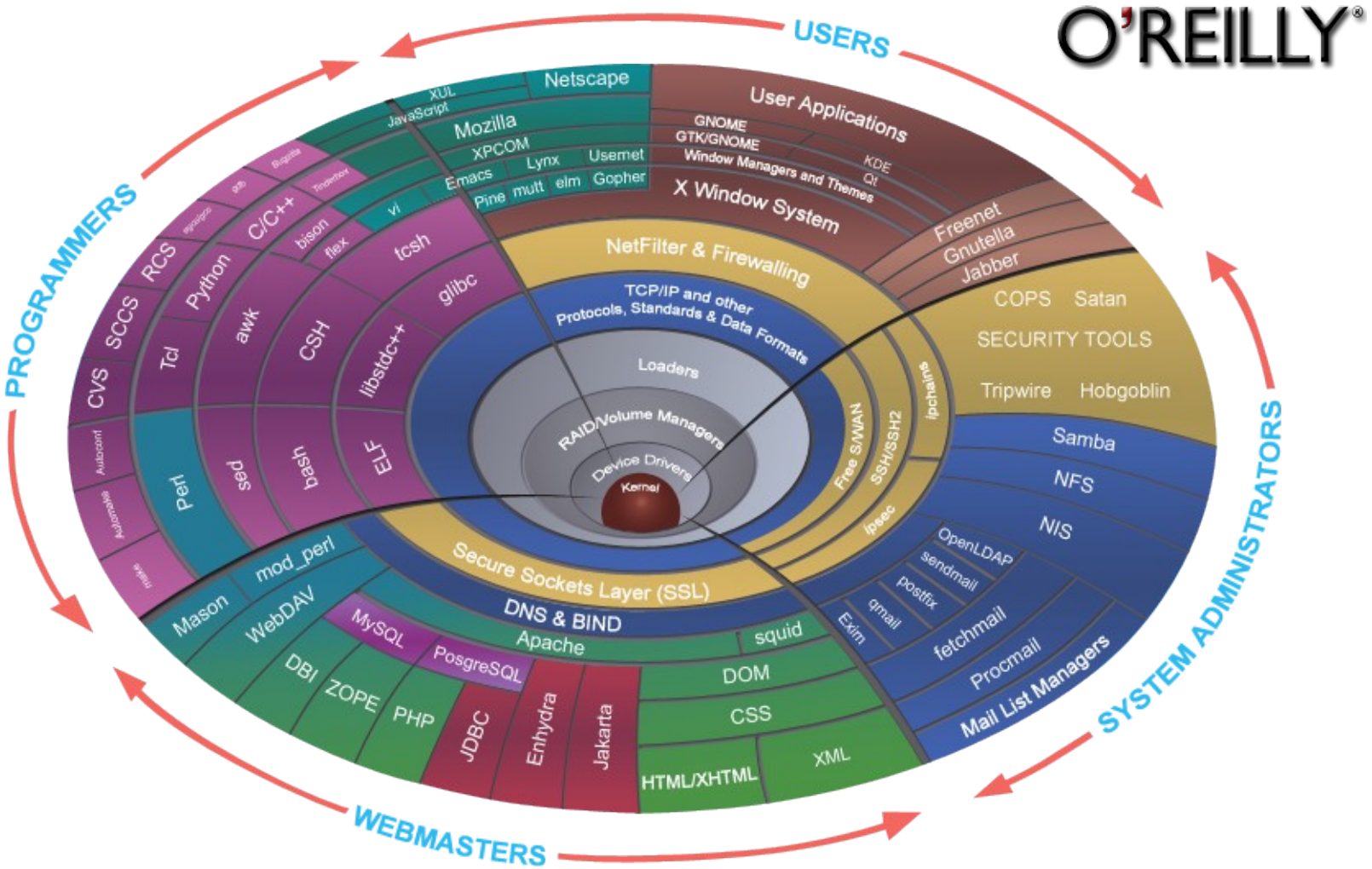


Founded 1993

DPKG package management system

© CyberOrg Info

From Kernel to Distribution



© CyberOrg Info

Linux Milestones



1991: Linus Torvalds initiates development of Linux

1992: Receives vendor package treatment

1994: Version 1.0 of the Linux kernel released

1998: IBM DB2 – Oracle – Informix – Sybase – port to Linux

1999: First SAP system on Linux productive

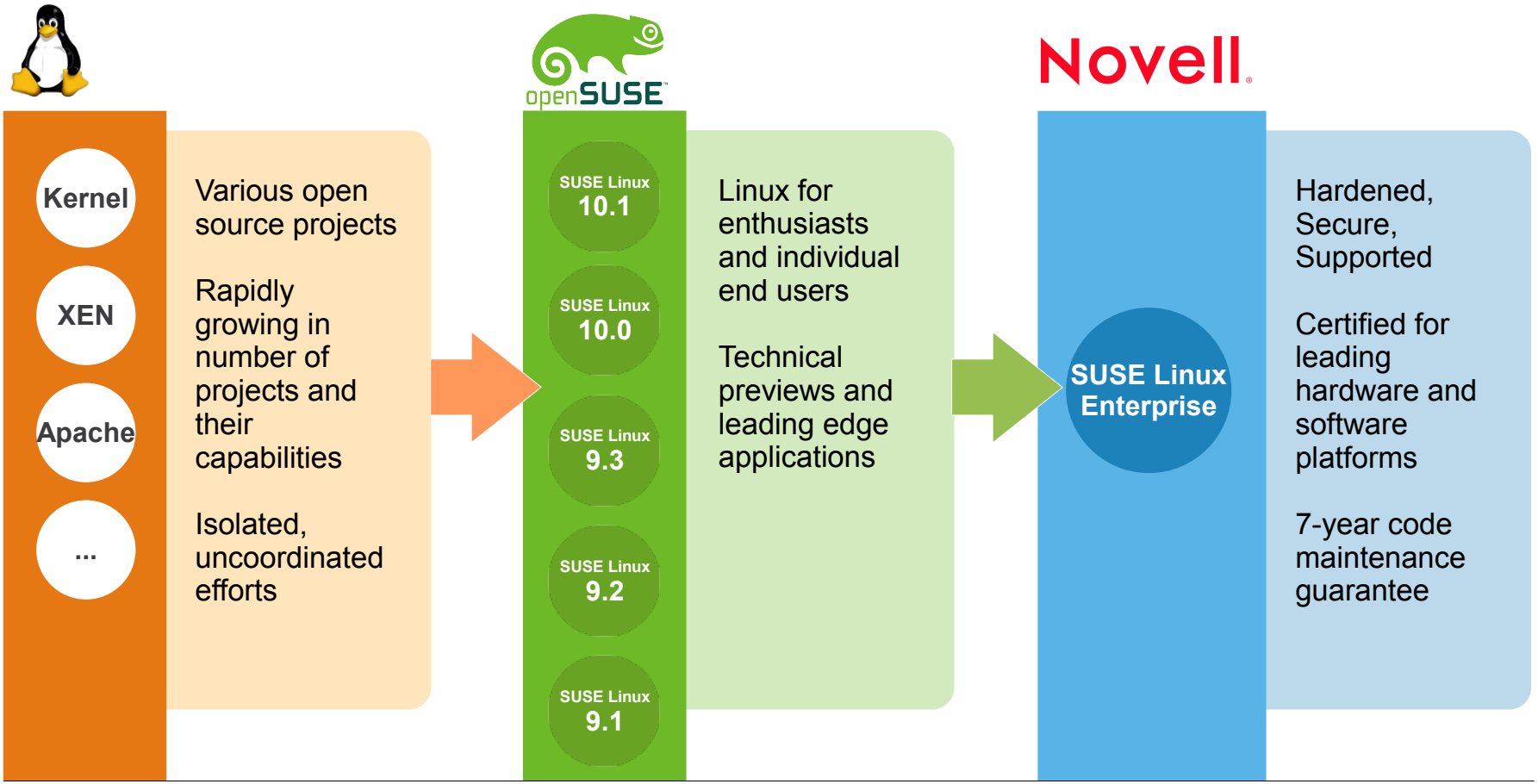
99 – 2001: Growth of Linux applications 92.7%

(Source ISIS Linux Report)

2003: Fastest growing OS, cross-platform, supported by most major vendors

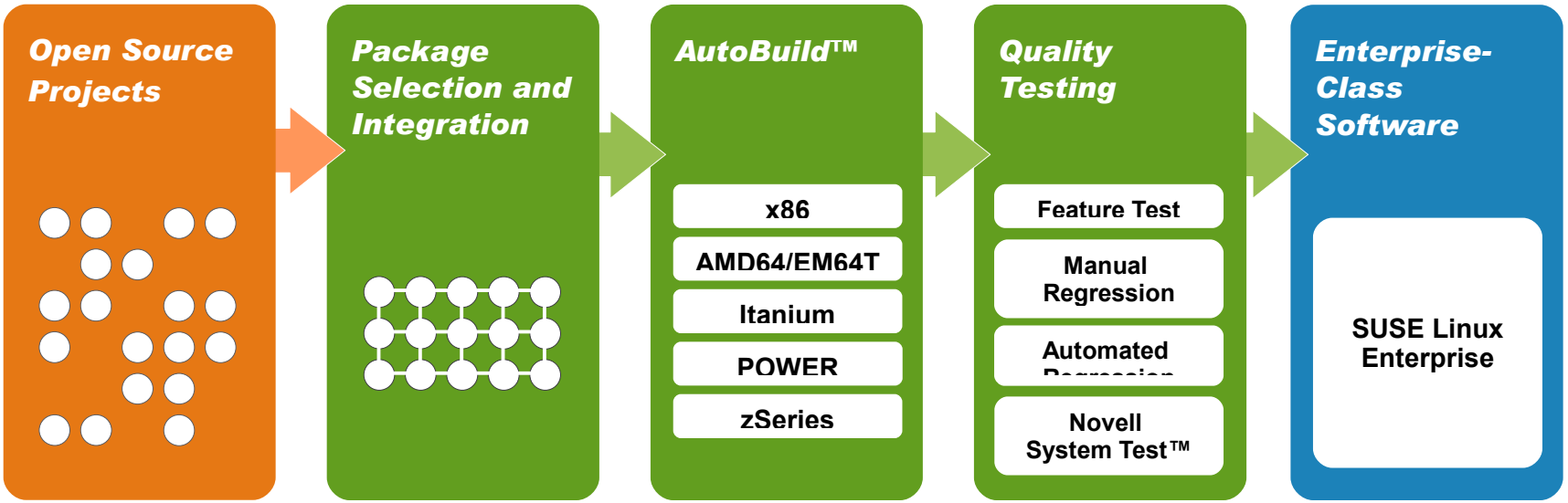
© CyberOrg Info

Novell's Linux Development Process



© CyberOrg Info

The AutoBuild™ Advantage



- Reduces production problems
- Consolidates IT skills across disparate systems
- Delivers critical updates in hours – not days or weeks

© CyberOrg Info

Linux CLI and GUI

Linux Shell and Command Line

The shell serves as an interface between a user and an OS.

Linux uses the Bourne again shell (**/bin/bash**) as the default.

Distinction between the 2 types of shells:

- Login Shells
- Non-login Shells

The following files are read when starting a login shell:

- /etc/profile
- ~/.bash_profile
- ~/.bash_login
- ~/.profile

The following file is read when starting a non-login shell:

- ~/.bashrc

© CyberOrg Info

Bash History and Switch User

History commands stored in ***~/.bash_history***

Ctrl +R to search history

To become root, you can enter the following:

- **su –**

To become another user, you can enter the following:

- **su –** <username>

© CyberOrg Info

CLI Tasks

echo $variable: Shows system or shell environment variable

alias/unalias : Defining aliases lets you create shortcuts for commands and their options or to create commands with entirely different names. Systemwide storage ***/etc/bash.bashrc***

type: will tell you whether it is a built-in shell command, a regular command, a function, or an alias.

cat <filename>: Displays the content of a file.

© CyberOrg Info

Piping and Redirection

Linux has three standard data channels:

- **0**: Standard input (stdin)
- **1**: Standard output (stdout)
- **2**: Standard error (stderr)

Each channel can be redirected:

- **<**: Redirects standard input.
- **>, 1>** or **>>**: Redirects standard output.
- **2>**: Redirects standard error output.

Using the **pipe ("|")**, the output from one command can be used as the input for another command.

The command **tee** can be used to split the standard output.

© CyberOrg Info

File System Hierarchy Standard (FHS)

Important directories:

- Root directory “/”
- Essential binaries (/bin/)
- System binaries (/sbin/)
- Libraries (/lib/)
- Configuration files (/etc/)
- Device files (/dev/)
- The hierarchy below /usr/
- Directory of applications (/opt/)
- Temporary area (/tmp/)
- Variable files (/var/)
- User directories (/home/)

© CyberOrg Info

File Types in the Linux System

The six file types in Linux:

- Normal files
- Directories
- Links
- Device files
- Sockets
- FIFOs

© CyberOrg Info

The Linux Desktop

Objectives

1. Overview of the Linux Desktop
   - The X Window System consists of a server component (X server) and client applications.
2. Use the KDE and GNOME Desktop Environment
   - Booting, log in and log out, navigate, customize and shutdown
3. Access the Command Line Interface From the Desktop
   - Six virtual terminals: Ctrl+Alt+F1 to Ctrl+Alt+F6. GUI on Ctrl+Alt+F7.

© CyberOrg Info

Text Editors

Most of the services of a Linux computer are configured by editing an ASCII file. For this reason, you need a text editor. There are 2 types of editors:

- command line editors (**vi, emac** etc)
- graphical editors (**kedit, jedit, kate** etc)

© CyberOrg Info

VI or VIM CLI Editor

The following is a summary of the available modes:

- **Command mode**: When vi starts, it is automatically in this mode. In command mode, vi can be given commands. The command **i** puts it into insert mode and the command **:** switches it to command-line mode.
- **Insert mode**: In this mode (keyword **--INSERT--**), **vi** accepts all input as text. Return to command mode with **Esc**.
- **Command-line mode**: In this mode (keyword **:**), **vi** accepts commands from the command line. Pressing **Enter** causes the command to be executed and automatically switches back to the command mode.

© CyberOrg Info

VI or VIM CLI Editor

Command and their meanings

i or **Insert** Switches vi to insert mode.

x or **Delete** Deletes the character where the cursor is.

dd Deletes the line in which the cursor is located and copies it to the buffer.

D Deletes the rest of the current line from the cursor position.

yy Copies the line in which the cursor is located to the buffer.

p,P Pastes the contents of the buffer after/before current cursor position.

zz Saves and exits vi.

u Undoes the last operation.

/pattern Searches forward from the cursor position for pattern.

?*pattern* Searches backward from the cursor position for pattern.

n Repeats the search in the same direction.

N Repeats the search in the opposite direction.

© CyberOrg Info

VI or VIM CLI Editor

:q Ends vi (if no changes were made).

:q! Ends vi without saving changes in the file.

:wq or :**x** Saves the current file and ends vi.

:w Saves the current file.

:w *file* Saves as. (Note: You continue editing the original file, not the new file.)

:syn on brings on syntax highlighting.

VIM preferences can be stored in ~/.vimrc add **set backspace=2** to allow back space delete function.

© CyberOrg Info

Working with files and user management using CLI

Objective

File manipulation under Linux

1. copy, rename, delete and move
2. directory listing
3. file handling and IO redirection

Users and Groups

1. Concept of users and groups
2. Owner creator
3. Primary and secondary group
4. types of file and directory permissions

© CyberOrg Info

Change Directories and List Directory Contents

The current directory is shown in the prompt of a shell terminal: tux@da10:~.

Some commands

ls, lists the current directory

pwd, prints the working directory

tree, prints the tree view of the directory

© CyberOrg Info

Create and View File

touch <newfilename> creates a new file.

cat <filename> Displays the contents of a file.

less (-#) <filename> Displays the content page by page. Number of lines to be displayed can be specified with -<number> switch.

head (-#) <filename> Displays first 10 lines. Number of lines to be displayed can be specified with -<number> switch.

tail (-f) <filename> Displays last 10 lines. To continuously display last lines of changing file use **-f** switch.

© CyberOrg Info

Manage Files and Directories

mv moves one or more files to another directory or

renames a file.

cp copies the file overwriting existing files, -r switch copies recursively.

mkdir makes directory, option **-p** allows you to create a

complete path.

rmdir removes the empty directory.

rm -r removes the directory with all the files inside. Files that are deleted with this command cannot be restored.

ln creates a hard link, files must be on the same filesystem as they share the same inode number.

ln -s creates a symbolic link

© CyberOrg Info

Finding Files on Linux

Kfind – GUI find programme.

find *path criterion action* to find files from shell terminal.

find -name filename, searches for files with the given name.

locate searches through a database previously created for this purpose, thus making it much faster.

whereis command returns the binaries (option -b), manual pages (option -m), and the source code (option -s) of the specified command.

which command searches all paths listed in the variable PATH, for the specified command and returns the full path of the command.

© CyberOrg Info

Search File Content

grep and its variant egrep are used to search files for certain patterns, according to this syntax: **grep** *search_pattern filename*

egrep command (or **grep -E**) accepts extended regular expressions for complex searches.

Metacharacters:

- ^: beginning of the line
- $: end of the line
- \<: beginning of the word
- \>: end of the word
- [abc]: one character from the set
- [a-z]: any one from the range
- [^xyz]: none of the characters
- .: any single characters

© CyberOrg Info

Manage File Permissions and Ownership

Besides using letters (rwx), you can also use the octal way of representing the permission letters with groups of numbers:

Owner	Group	Others
rwx	rwx	rwx
421	421	421

By using number equivalents, you can add the numbers together, as in the following:

r w x r w -

r - x

4 2 1 4+2+1=7 4 2 - 4+2=6

4 - 1 4+1=5

© CyberOrg Info

Manage User Accounts

useradd <username> is used to add users to the system.

useradd -m <username> to create home directory.

passwd <username> to change password.

usermod to modify the existing user.

userdel <username> to delete the user.

userdel -r <username> to delete the user with home folder.

Important files

- /etc/passwd
- /etc/group
- /etc/shadow
- /etc/default/useradd
- /etc/login.defs

© CyberOrg Info

Manage the Network Configuration Information from YaST

The following details are needed to integrate the network device into an existing network:

- Method of network setup
- Static IP address
- Network mask
- Host name
- Name server
- Routing (gateway)

ifconfig to verify the configuration of the network cards

© CyberOrg Info

Package Management using CLI

Manage RPM Software Packages

Introduction

- There are several package formats available for Linux
- The most common format is RPM Package Manager (**RPM**)
  - Formerly Red Hat Package Manager
- Installing software in **RPM** format can be done with **YaST** or using the command line utility **rpm**
  - **YaST** ensures the automatic resolution of dependencies
  - With **rpm** you must resolve them manually

© CyberOrg Info

Manage RPM Software Packages

Install, Update and Uninstall Packages

- **Install a RPM Package**
  - Usually use **rpm -i *package_name*.rpm**
    - Use **-v** (verbose) for more information
    - Use **-h** (hash) for a progress bar
    - Copies executable programs, documentation files and start scripts to the correct place in the file system
  - Only installs if no dependency conflicts or file problems
    - If not, lists dependencies or problems to be resolved
    - Could use options to ignore errors but may not work
      - Use **––nodeps** to ignore dependencies

© CyberOrg Info

Manage RPM Software Packages

Install, Update and Uninstall Packages

- **Uninstall a RPM Package**
  - Syntax **rpm -e *package_name***
  - All files except modified configuration files are removed
    - Uses the **RPM** database to figure this out
    - Ensures a clean uninstall
  - Package is deleted only if does not break other packages
    - If dependencies get error message
  - Could force deletion using the **–-nodeps** option but will break other software

© CyberOrg Info

Manage RPM Software Packages

Query the RPM Database and RPM Archives

- Use the **-q** option to query the **RPM** database and add the option **-p** to inspect the **RPM** archives that are not installed yet
  - Some common options

-a - List all install packages

-i - List package information

-l - Display a file list

-f *file* - Find out which package the file belongs to

- Must specify the full path

-d - List only documentation files (implies **-l**)

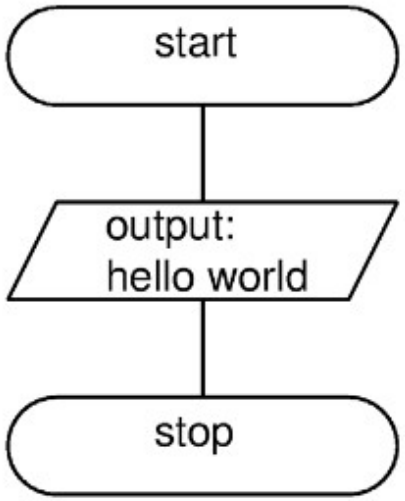
-c - List only configuration files (implies **-l**)

© CyberOrg Info

Shell Scripting

Simple Shell Scripting

The following is the flow chart for the script:



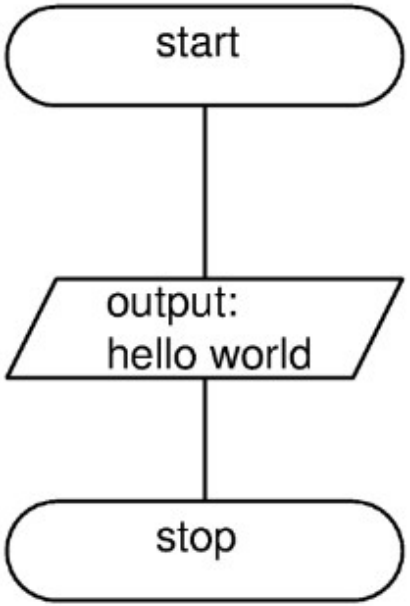
The script consists of three elements:

- The program start
- The action to print out “Hello world”
- The program stop

© CyberOrg Info

Shell Scripting

The following illustrates the 3 elements with the corresponding script code on the right:

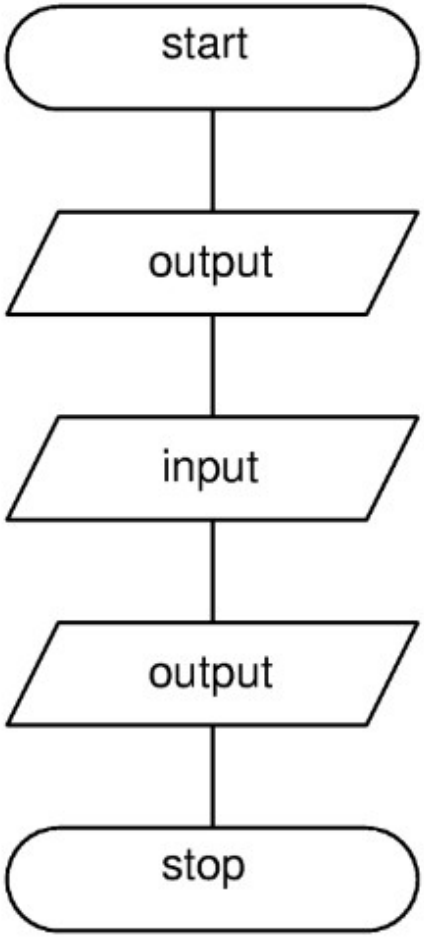



```
#!/bin/bash

# This script ...

echo "Hello world!"

exit 0
```


© CyberOrg Info

Shell Scripting - Input/Output

The following flow chart illustrates the structure of a script that reads user input:




```
Please enter ... :
```

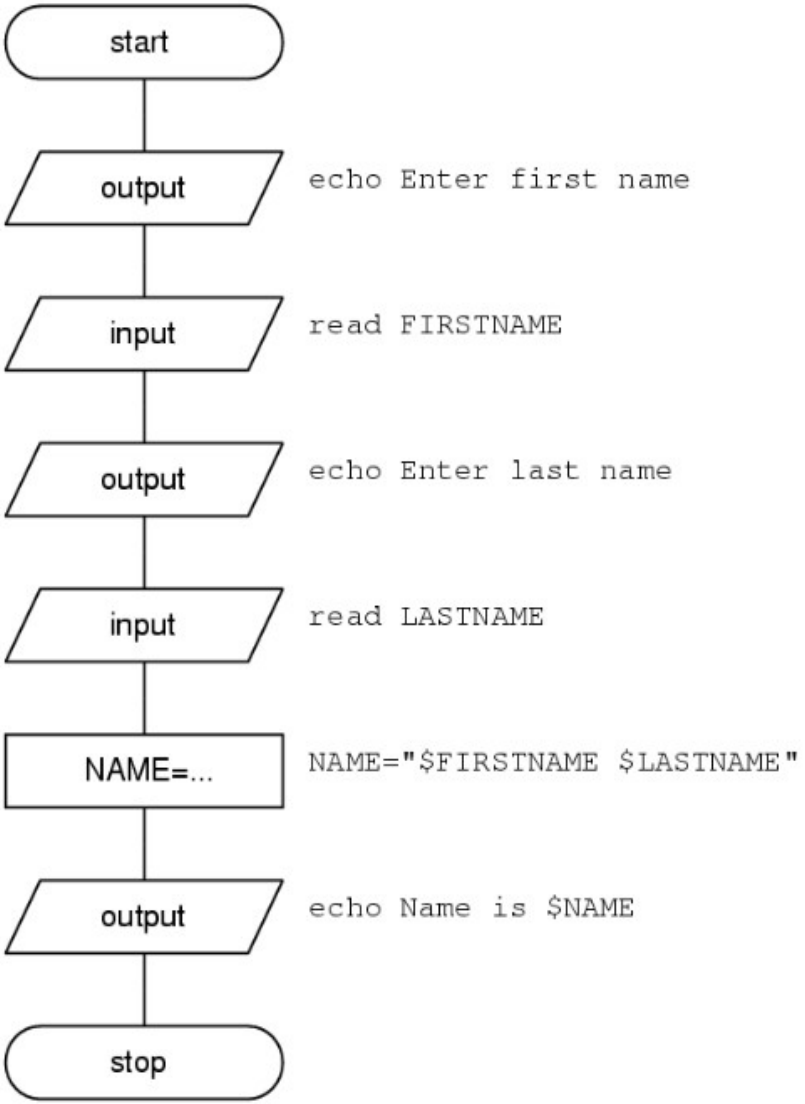
```
read VARIABLE
```

```
echo $VARIABLE
```


© CyberOrg Info

Shell Scripting - Variables

The following flowchart and script show how a string value can be assigned to a variable:



© CyberOrg Info

Shell Scripting – Command Substitution

How to Use Command Substitution

The term *command substitution* basically means that the output of a command is used in a shell command line or a shell script.

In the following example, the output of the command date is used to generate the output of the current date:

```
#!/bin/bash

echo "Today is `date +%m/%d/%Y`"
```

An important thing to remember is that the command date +%m/%d/%Y is included in backticks (` ... `).

Instead of printing the output of a command to the screen with echo, it can also be assigned to a variable, as in the following:

```
#!/bin/bash

TODAY=`date +%m/%d/%Y`
echo "Today is $TODAY"
```

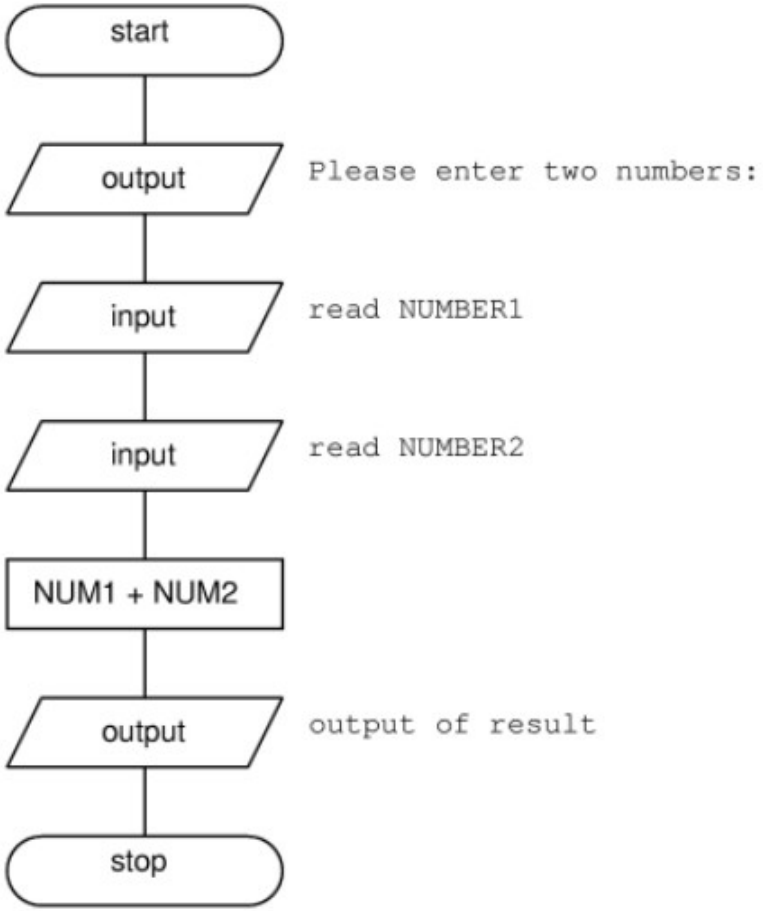

© CyberOrg Info

Shell Scripting – Arithmetic Operations

Use Arithmetic Operations

Do the following:

1. Review the following flowchart:



© CyberOrg Info

Shell Scripting – Variable Substitution

The following variable substitutions are possible:

Substitution Operator	Description
${variable-value}	Returns value if the variable does not exist.
${variable=value}	Assigns value to the variable and returns value if the variable does not exist.
${variable+value}	Returns value if the variable exists.
${#variable}	Returns the number of characters in the value of variable.
${variable#pattern}	Deletes the shortest part matched by pattern from the beginning of the variable's value and returns the rest.
${variable##pattern}	Deletes the longest part matched by pattern from the beginning of the variable's value and returns the rest.
${variable%pattern}	Deletes the shortest part matched by pattern from the end of the variable's value and returns the rest.

© CyberOrg Info

Shell Scripting

The following are some examples of how to use the substitution operators:

```
tux@DA1:~> echo $VAR

tux@DA1:~> echo ${VAR-value}
value

tux@DA1:~> echo $VAR

tux@DA1:~> echo ${VAR=value}
value

tux@DA1:~> echo $VAR
value

tux@DA1:~> VAR=
tux@DA1:~> echo ${VAR=value}

tux@DA1:~> echo ${VAR:=value}
value

tux@DA1:~> echo $VAR
value

tux@DA1:~> echo ${VAR+VaLue}
VaLue
```


© CyberOrg Info

Shell Scripting – if-else branches

Create Basic Branches With the if Command

You can use the if command to perform certain actions in your script that depend on a condition.

The following is the basic usage of the if command:

```
if condition
then
  commands
fi
```

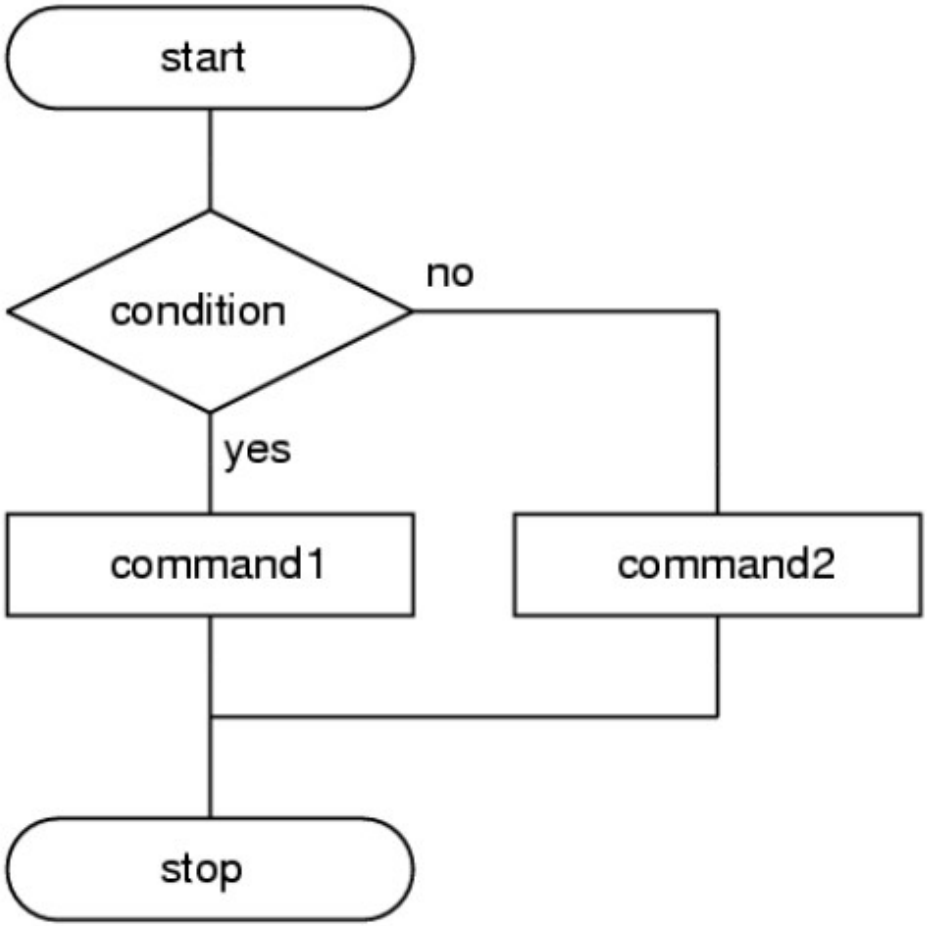
The if statement can be extended with an optional else statement, as in the following:

```
if condition
then
  command1
else
  command2
fi
```


© CyberOrg Info

Shell Scripting

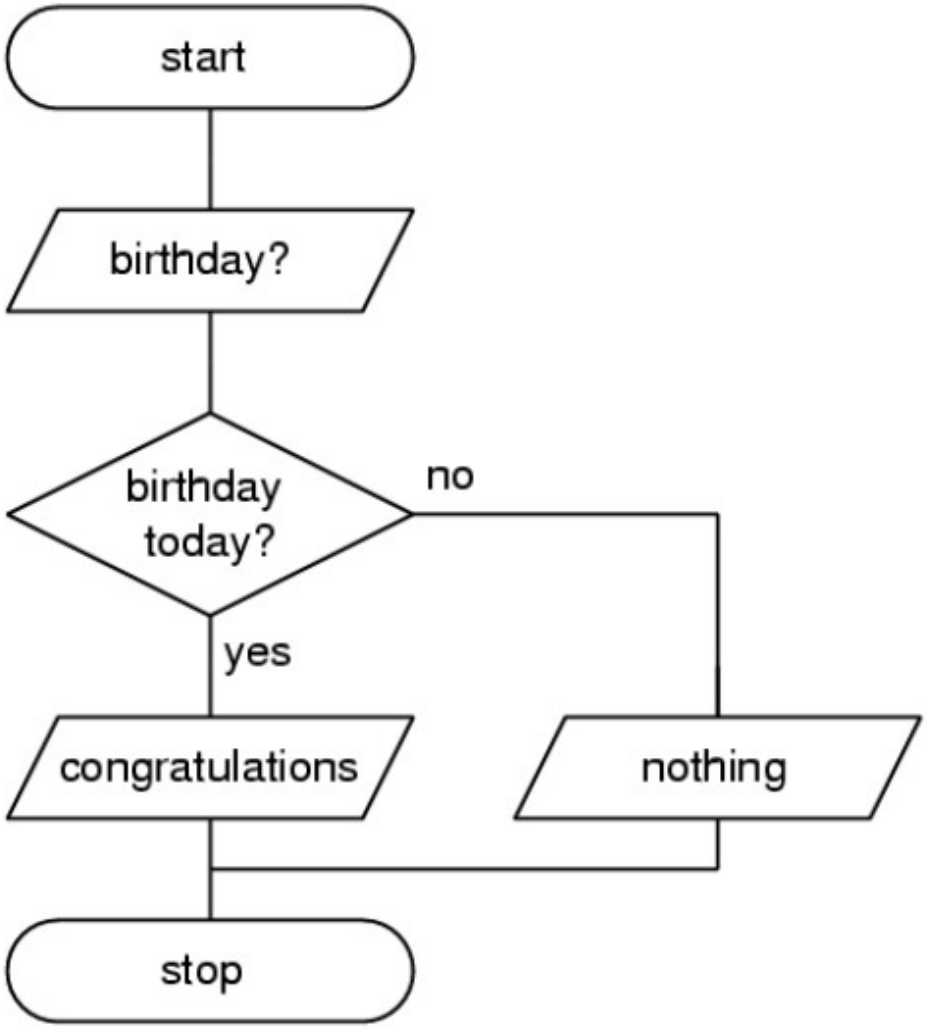
In a program flow chart, a branch created with an if statement can be represented like the following:



© CyberOrg Info

Shell Scripting

The following example uses a sample script to explain how an if branch works:



© CyberOrg Info

Shell Scripting - elif

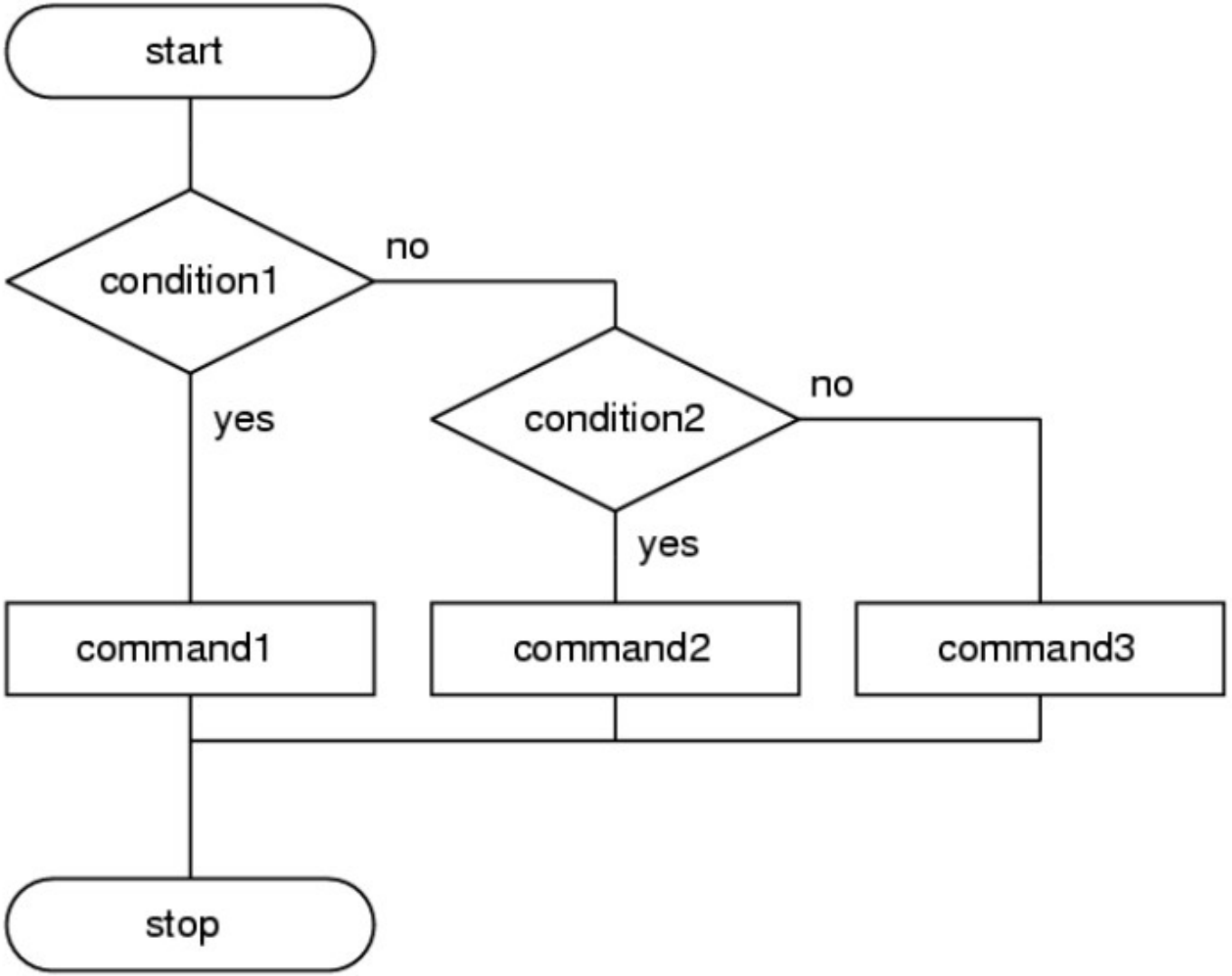
Several if branches are often nested in each other. The command elif, which represents a more compact way of writing the command sequence else if, is useful for the following kind of structures:

```
if condition1
then
  command1
elif condition2
  command2
else
  command3
fi
```


© CyberOrg Info

Shell Scripting

The elif command is illustrated in the following:



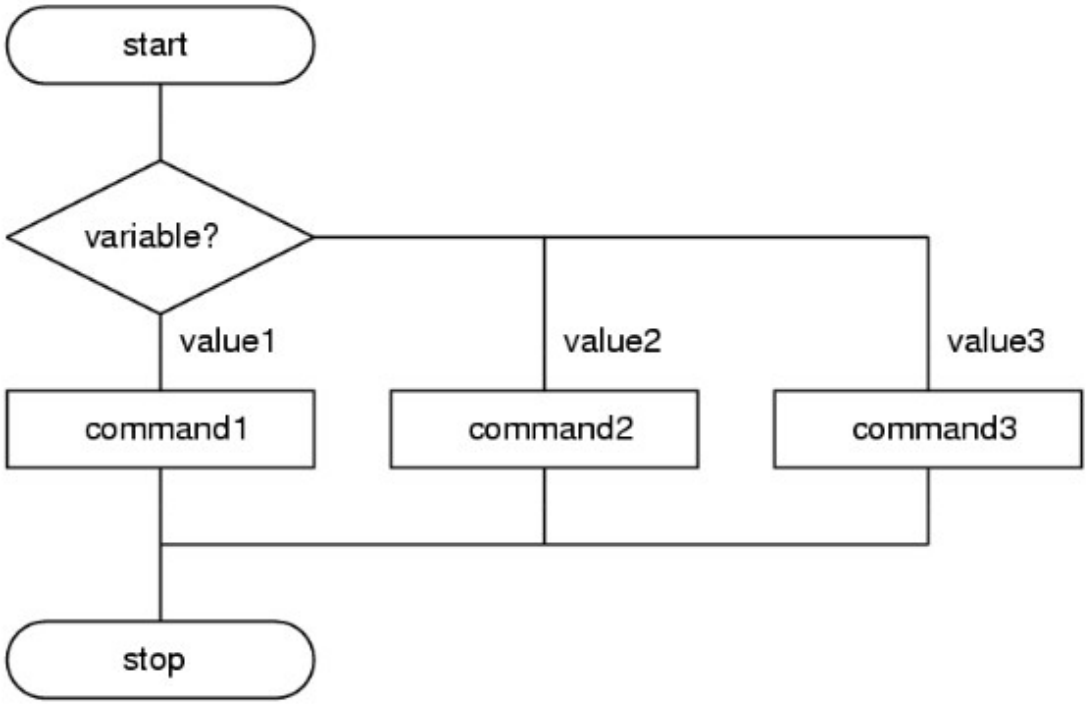
© CyberOrg Info

Shell Scripting – Case statements

A case statement has the following structure:

```
case $variable in
  expression1) command1;;
  expression2) command2;;
esac
```

In a flow chart, multiple branching with case looks similar to a simple branch created with if:



© CyberOrg Info

Shell Scripting – Case example

The following is an example of how a multiple branch works:

```
#!/bin/bash
cat << EOF
  Name me an animal and I will tell you how many legs it
has!
EOF

read CREATURE
case "$CREATURE" in
  dog | cat | mouse ) echo "A $CREATURE has 4 legs."
  ;;
  bird | human | monkey ) echo "A $CREATURE has 2 legs."
  ;;
  spider ) echo "A $CREATURE has 8 legs."
  ;;
  fly ) echo "A $CREATURE has 6 legs."
  ;;
  * ) echo "I haven t the faintest idea how many
      legs a(n) $CREATURE has."
  ;;

esac
exit 0
```


© CyberOrg Info

Shell Scripting – Loops: while, until

Create Loops Using the while and until Commands

The purpose of a loop is to test a certain condition and to execute a given command while the condition is true (while loop) or until the condition becomes true (until loop).

The following is the structure of a while loop:

```
while condition
do
  commands
done
```

The following is a structure of until loop:

```
until condition
do
 commands
done
```


© CyberOrg Info

Shell Scripting – Loop: for

Process Lists with the for Loop

The purpose of a for loop is to process a list of elements. It has the following syntax:

```
for variable in element1 element2 element3
do
  commands
done
```

```
for i in 1 2 3 4 5 6 7 8
do
  ping -c1 DA$i
done
```

If you want to use a range of numbers in your for loop, you can use the following C-Style syntax:

```
LIMIT=10

for ((a=1; a <= LIMIT ; a++))
do
  echo -n "$a "
done
```


© CyberOrg Info

Shell Scripting - Functions

The following is a simple function that creates a directory and then changes to that directory:

```
# mcd: mkdir + cd; creates a new directory and
# changes into that new directory right away

mcd (){
  mkdir $1
  cd $1
}
```

After having been created, this function can be called in a shell scripts, as in the following:

```
...
mcd directory
...
```


© CyberOrg Info

Shell Scripting – Function example

```
# Prompt the user to answer with "yes" or "no.
# The question itself is supplied as an argument
# when calling the function, for example:
# "yesno Do you want to continue?"

yesno (){
  while true
  do
    echo "$*"
    echo "Please answer by entering (y)es or (n)o:"
    read ANSWER
    case "$ANSWER" in
      [yY] | [yY][eE][sS] )
         return 0
         ;;
      [nN] | [nN][oO] )
         return 1
         ;;
      * )
         echo "I cannot understand you over here."
         ;;
     esac
  done }
```


© CyberOrg Info

Shell Scripting - getopts

```
while getopts abc: variable
do
  case $variable in
    a ) echo "The option -a was used." ;;
    b ) echo "The option -b was used." ;;
    c ) option_c="$OPTARG"
        echo "Option c has been set." ;;
  esac
done

echo $option_c
```


© CyberOrg Info

Shell Scripting – Common commands

- You can use external commands in Shell scripts to perform certain tasks.
- The following is a list of commonly-used commands:
  - cat
  - cut
  - date
  - echo
  - grep and egrep
  - sed
  - test
  - tr

Read “man” for more information and examples to use

© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This script prints a "Hello world" greeting
# Author: Tux Penguin
# Created: 8/22/2005

echo -e "\aHello\nworld"
exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This script reads the users first and last name
# and then prints a greeting with the full name.
# Author: Tux Penguin
# Created: 8/22/2005

echo "Please enter your first name:"

# first name gets assigned to variable FIRSTNAME
read FIRSTNAME

echo "Please enter your last name:"

# last name gets assigned to variable LASTNAME
read LASTNAME

#Now print the greeting:
echo "Welcome to the club, $FIRSTNAME $LASTNAME"
exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This scripts reads the users first and last name
# and then prints a greeting with this full name.
# Author: Tux Penguin
# Created: 8/22/2005

echo "Please enter your first name:"

# first name gets assigned to variable FIRSTNAME
read FIRSTNAME

echo "Please enter your last name:"

# last name gets assigned to variable LASTNAME
read LASTNAME

# create a new NAME variable
NAME="$FIRSTNAME $LASTNAME"

# Now print the greeting:
echo "Welcome back home, $NAME"

exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This script prints information about
# the current login
# and the current working directory.
# Author: Tux Penguin
# Created: 8/22/2005

login=`whoami`
path=`pwd`

echo "The current login is: $login"
echo "The current path is: $path"
exit 0
```


© CyberOrg Info

Shell Scripting

```
#!/bin/bash
# This script lets the user specify two whole
# numbers and then adds them together. All kinds of
# arithmetic formats that are possible
# under Bash are used, one after another.
# Author: Tux Penguin
# Created: 8/22/2005

declare -i INTEGER1
declare -i INTEGER2
declare -i SUM

# read first integer
echo "Please enter first integer: "
read INTEGER1

# read second integer
echo "Please enter second integer: "
read INTEGER2

# this uses  expr  for Bourne shell compatibility:
RESULT=`expr $INTEGER1 + $INTEGER2`
echo "The  expr  command returns the result: $RESULT."

# this uses the Bash built-in  let :
let RESULT="$INTEGER1 + $INTEGER2"
echo "The  let  built-in returns the result: $RESULT."

# this uses a Bash-specific arithmetic expression:
RESULT=$[$INTEGER1 + $INTEGER2]
#or:
#RESULT=$(($INTEGER1 + $INTEGER2))

echo "Using an arithmetic expression in Bash, the result
is: $RESULT."

# this one uses the variables declared as integers #above:
SUM=INTEGER1+INTEGER2
echo "Using the variables declared as integers, the sum is:
$SUM."

exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This script searches for files in the current
# directory.
# The user is prompted to enter a filename;
# if no name is entered, we search for the default
# value anyway, which is set to "*.bak"
# Author: Tux Penguin
# Created: 8/22/2005

echo "Please enter the file to be searched for (default is:
*.bak):"
read FILE
find . -name "${FILE:="*.bak"}"
exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This script checks whether a file exists and if
# its executable
# Author: Tux Penguin
# Created: 8/22/2005

echo "Please enter a filename: "

read FILENAME

if test -e $FILENAME
then
  if test -x $FILENAME
  then
    echo "The file exists and is executable."
  else
    echo "The file exists but is not executable."
  fi
else
    echo "The file does not exist."
fi

exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# A script to iterate over a simple "while" loop 100
# times.
# Author: Tux Penguin
# Created: 8/22/2005

declare -i COUNTER=1

while  test $COUNTER -le 100
do
  echo "The counter stands at $COUNTER."
  COUNTER=COUNTER+1
  sleep 1
done

exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This script renames all files in the current
# directory so that they have all lowercase file
# names.
# Author: Tux Penguin
# Created: 8/22/2005

for     FILE in `find . -type f -maxdepth 1`
do
  NEWFILE=`echo $FILE | tr [A-Z] [a-z]`
  if test $FILE != $NEWFILE
  then
    echo mv $FILE $NEWFILE
  fi
done

exit 0
```


© CyberOrg Info

Shell Scripting - Examples

```
#!/bin/bash
# This script renames all files in the current
# directory so that they have all-lowercase file
# names.
# 2nd version: Now we also check whether the file
# already exists with lowercase lettering.
# Author: Tux Penguin
# Created: 8/22/2005

for FILE in `find . -type f -maxdepth 1`
do
  NEWFILE=`echo $FILE | tr [A-Z] [a-z]`
  if test $FILE != $NEWFILE
  then
    if test -e $NEWFILE
    then
      echo "There is already a file
            with the name $NEWFILE."
      echo "$FILE will not be renamed."

# Skip the rest and begin next loop iteration:
      continue
    fi
    echo mv $FILE $NEWFILE
  fi
done
exit 0
```


© CyberOrg Info

OpenOffice.Org

OpenOffice.Org and full Linux curriculum

Get the presentations from here:
http://www.cospa-project.org/download.html

Get the complete course wares for teaching in lab and instructors guides:
- Intermediate School Level Curriculum
- High School Level Curriculum

Disclaimer: Contents of many slides generously borrowed from various sources, copyrights for those contents belong to the original authors.

© CyberOrg Info