Contents

[**0.0.1 Course Introduction** 2](#_Toc177489824)

[**Basic Linux File Management** 4](#_Toc177489825)

[**Linux File System Basics** 4](#_Toc177489826)

[**Create, Copy, and Move a Linux Directory** 6](#_Toc177489827)

[**Remove a Linux Directory** 7](#_Toc177489828)

[**File Management Commands** 7](#_Toc177489829)

[**The nano Editor** 9](#_Toc177489830)

[**The vi Editor** 10](#_Toc177489831)

[Section 1.1Linux Introduction 12](#_Toc177489832)

[**1.1.3Linux Introduction Facts** 13](#_Toc177489833)

[Linux Distributions 17](#_Toc177489834)

[Other Linux Implementations 17](#_Toc177489835)

[**1.1.4Server Roles Facts** 19](#_Toc177489836)

[**1.1.5Practice Questions** 24](#_Toc177489837)

[**Question 1 of 10** 24](#_Toc177489838)

[**Question 2 of 10** 25](#_Toc177489839)

[**Question 3 of 10** 25](#_Toc177489840)

[**Question 4 of 10** 26](#_Toc177489841)

[**Question 5 of 10** 27](#_Toc177489842)

[**Question 6 of 10** 28](#_Toc177489843)

[**Question 7 of 10** 29](#_Toc177489844)

[**Question 8 of 10** 30](#_Toc177489845)

[**Question 9 of 10** 31](#_Toc177489846)

[**Question 10 of 10** 32](#_Toc177489847)

[**Section 2.1The Linux Shell** 33](#_Toc177489848)

[**2.1.2Linux Shell Facts** 35](#_Toc177489849)

[**2.1.6Linux Shell Command Facts** 37](#_Toc177489850)

[**2.1.7Practice Questions** 41](#_Toc177489851)

[**Section 2.2 Linux Help** 47](#_Toc177489852)

[**2.2.3Help Facts** 48](#_Toc177489853)

[**2.2.4Get Help** 52](#_Toc177489854)

[**2.2.5Practice Questions** 53](#_Toc177489855)

[**Section 2.3Text Editors** 59](#_Toc177489856)

[**2.3.4Text Editor Facts** 61](#_Toc177489857)

[**2.3.5Create a New File** 65](#_Toc177489858)

[**2.3.8Practice Questions** 65](#_Toc177489859)

[**Section 2.4Aliases** 72](#_Toc177489860)

[**2.4.3Alias Facts** 73](#_Toc177489861)

[**2.4.4Practice Questions** 74](#_Toc177489862)

[Question 5 of 10 76](#_Toc177489863)

[Explanation 76](#_Toc177489864)

[**Section 2.5Environment Variables** 79](#_Toc177489865)

[**2.5.3Environment Variable Facts** 80](#_Toc177489866)

[**2.5.5Practice Questions** 86](#_Toc177489867)

[**Section 2.6Shell Configuration Files** 92](#_Toc177489868)

[**2.6.3Shell Configuration Facts** 93](#_Toc177489869)

[**2.6.4Practice Questions** 95](#_Toc177489870)

# **0.0.1 Course Introduction**

This course is designed to prepare you to pass the TestOut Linux Pro and CompTIA Linux+ Certifications.

As you study this section, answer the following questions:

* What are the course prerequisites?
* Which major topics are covered in the course?
* Which certifications does this course prepare you for?

**Course Prerequisites**

Before you take this course, you should have a basic understanding of computers. You should have:

* Computer systems and networking knowledge
* 9 to 12 months IT experience

Although not a prerequisite, it's helpful if you've had some basic Linux exposure in both command line and GUI environments.

It's also helpful to have some additional certifications. Optional certifications could include the:

* TestOut PC Pro Certification
* TestOut Network Pro Certification

If you don't have this knowledge and experience, you can still take the course. Just be prepared to have a slightly steeper learning curve.

**TestOut Linux Pro Certification**

The TestOut Linux Pro Certification is part of the TestOut Pro Certification. This certification measures not only what you know, but also what you can do. It measures your ability to install, configure, manage, and troubleshoot the Linux operating system.

The TestOut Linux Pro Certification addresses the following knowledge domains:

* System administration and configuration
* Storage and file system management
* Networking and printing
* Security and access control

**CompTIA Linux+ Certification**

The CompTIA Linux+ Certification validates the competency required for system administrators to manage and maintain Linux, including an essential knowledge of the command line interface.

The CompTIA Linux+ Certification addresses the following knowledge domains:

* Hardware and system configuration
* Systems operation and maintenance
* Security
* Linux troubleshooting and diagnostics
* Automation and scripting

**Real-World Skills**

In addition to covering things you need to know to become certified, this course is designed to help you gain real-world skills that you'll use every day as a Linux technician. By the time you complete this course, you should be able to do the following:

* Install a new Linux system
* Install software on a Linux system
* Create and manage Linux users and group
* Install and manage Linux storage devices and file systems
* Monitor Linux systems
* Set up and manage Linux networks, including configuring routers and firewalls
* Install and manage Linux systems in the cloud and in virtualized environments
* Understand the basics of scripting and automation
* Manage security, including network and VPN security
* Troubleshoot various aspects of Linux, including users, storage, networking, and SELinux

# **Basic Linux File Management**

As a Linux administrator, much of your work is done from a command line interface. If you are new to Linux, you may find it difficult to navigate the file system and make changes to text files. To help you make a quick transition to Linux, and to help you be successful in this course, this lesson introduces a few basic command line interface pointers.

This lesson covers the following topics:

* Linux file system basics
* Create, copy, and move a Linux directory
* Remove a Linux directory
* File management commands
* The vi editor

**Linux File System Basics**

The container that holds Linux files is called a directory. The following rules apply to files and directories:

* File and directory names are case sensitive. The filenames testout.txt, TestOut.txt, and TESTOUT.txt are three different files.
* File and directory names can be made up of upper and lowercase letters, numbers, the dot (.), and underscore (\_) symbols.
* The dot (.) in a filename is not necessary. However, it can be helpful to use a dot based filename extension to identify file type.
* A directory can hold files and other directories.
* Linux directories are arranged hierarchically as parent and child directories.
* The root directory does not have a parent directory and is represented by a single forward slash (/).
* The forward slash is also used as a delimiter when showing the directory path from the root directory to a file or directory. For example: /home/rtracy/bin/myaddresses.dat.

The following table describes basic commands you use to navigate within a Linux directory.

| **Command** | **Description** | **Examples** |
| --- | --- | --- |
| **pwd** | Displays the current working directory. | * If a user named Fred is currently in his home directory and types **pwd** at the shell prompt, **/home/Fred** is displayed. |
| **cd** | Changes the present working directory.   * **cd ..** changes to the parent directory. * **cd ../..** changes two levels up in the directory. * **cd /** changes to the root directory. | * **cd directory1** changes to a directory named **directory1** within the current working directory. (This is a *relative* path.) * **cd /home/Fred/directory1** switches to **directory1** in Fred's home directory, regardless of the current working directory. (This is an absolute path.) |
| **ls** | Displays the contents of a directory. Options include:   * **-a** displays all directory contents, including hidden content. * **-l** displays extended information, including the owner, modified date, size, and permissions. * **-R** displays the contents of a directory and all of its subdirectories. * **-d** displays directories but not files. * **-r** reverses the sort order. | * **ls -al** displays a long listing of all the contents in the current working directory, including hidden content. * **ls -d** displays only directories within the current directory. * **ls -R /etc** displays the contents of the /etc directory and all of its subdirectories. |

**Create, Copy, and Move a Linux Directory**

The following table describes basic commands you use to manage a Linux directory:

| **Command** | **Description** | **Examples** |
| --- | --- | --- |
| **mkdir** | Creates a new directory. Use the **-p** option to create all directories within the specified path when that path does not already exist. | * **mkdir work\_files** creates a directory named *work\_files* in the current working directory. * **mkdir /home/Fred/work\_files** creates a directory named *work\_files* within the specified path. |
| **cp** | Copies directories. Copying leaves the source contents (directories and files) intact. Use the **-r** or -R option to recursively copy subdirectories and files within the directory. | * **cp -r /temp /home/user** copies the entire **/temp** directory (with all of its files, subdirectories, and files in the subdirectories) to the **/home/user** directory. |
| **mv** | Moves or renames directories (and files). Moving directories removes the source directory and places it in the destination. Options include:   * **-f** overwrites a directory that already exist in the destination directory without prompting. * **-i** prompts before overwriting a directory in the destination directory. * **-n** never overwrites files in the destination directory. | * **mv /temp/station ~/doc/** moves **station** from the **/temp** directory to the **~/doc** directory. * **mv /current /previous** renames the directory **current** to **previous** |

**Remove a Linux Directory**

The following table describes basic commands you use to remove a Linux directory:

| **Command** | **Description** | **Examples** |
| --- | --- | --- |
| **rmdir** | Deletes an empty directory. | * **rmdir ~/Fred/work\_files** deletes the **work\_files** directory if it is empty. |
| **rm** | Removes the directory and file information from the file system, making the directories and files inaccessible. Options include:   * **-i** prompts before removing. * **-r** removes directories, subdirectories, and files within them. * **-f** eliminates prompt for read-only files and avoids an exit code error if a file doesn't exist. | * **rm -rf /home/user/temp** deletes the temp directory with all its subdirectories and files without prompting. * **rm -r /home/user/\*** deletes all directories and files in the **/home/user** directory. |

**File Management Commands**

The following table describes Linux commands that can be used to manage files.

| **Command** | **Function** | **Examples** |
| --- | --- | --- |
| **touch** | If the file does not exist, **touch** creates a blank version of the file. If the file does exist, this command updates the file's modification and last accessed time. | * **touch myfile** makes a blank file named **myfile** . |
| **cp** | Copies files. Copying leaves the source file intact.   * **-f** overwrites files that already exist in the destination directory. * **-i** prompts before overwriting a file in the destination directory. | * **cp /temp/document\_ab.txt ~/doc/document.txt** copies **document\_ab.txt** from the **/temp** directory to the **~/doc** directory and renames the file to **document.txt** . * **cp /temp/\*.txt ~/doc** copies all text files from the **/temp** directory to the **~/doc** directory. |
| **mv** | Moves or renames files (and directories). Moving files erases the source file and moves it to the destination.   * **-f** overwrites files that already exist in the destination directory. * **-i** prompts before overwriting a file in the destination directory. * **-n** never overwrites files in the destination directory. | * **mv /temp/document.txt ~/doc/document.txt** moves **document.txt** from the **/temp** directory to the **~/doc** directory. * **mv /temp/\*.txt ~/doc/\*.txt** copies all text files from the **/temp** directory to the **~/doc** directory. |
| **rm** | Removes a file or directory. Use the **-f** option to delete with a prompt.  The **rm** command deletes a file or directory's inode, but it does not actually delete the data. To permanently remove data, use the **shred** command. | * **rm myfile** deletes a file in the current directory named **myfile** . * **rm /home/user/myfile** deletes **myfile** from the **/home/user** directory regardless of the current directory. * **rm -f /home/user/temp/\*** deletes all files in the temp directory with prompts. |

**The nano Editor**

The nano editor is included in most Linux distributions.

* To start the editor, type **nano** at the shell prompt.
* The nano editor is simpler to use than the vi editor.
  + Common keystroke shortcuts are listed at the bottom of the nano interface.
    - A caret (^) in the shortcut means press and hold the Ctrl key.
    - 'M-' listed in the shortcut means press and hold the Meta key. On modern keyboards, the Alt or Esc key is substituted for the Meta key.
  + Ctrl+G (listed as ^G) displays the full help text, which includes a full list of shortcuts.
  + Prompts are displayed when user input is needed.
  + The Home, End, PgUp, PgDn, and arrow keys move the cursor in the same way as other common editors.

The table below lists some common nano shortcuts:

| **Shortcut** | **Function** |
| --- | --- |
| ^G (Ctrl+G) | Displays the help text, which includes a list of all keyboard shortcuts. |
| ^X (Ctrl+X) | Closes the current buffer or exits from nano. |
| ^O (Ctrl+O) | Writes the current buffer (or the marked region) to disk. |
| M-Space (Alt+Space or Esc+Space) | Goes back one word. |
| ^Space (Ctrl+Space) | Goes forward one word. |
| M-A (Alt+A or Esc+A) | Marks text starting from the cursor position. |
| M-6 (Alt+6 or Esc+6) | Copies current line (or marked region) and stores it in cutbuffer. |
| ^K (Ctrl+K) | Cuts current line (or marked region) and stores it in cutbuffer. |
| ^U (Ctrl+U) | Uncuts (paste) from the cutbuffer into the current line. |
| ^W (Ctrl+W) | Searches forward for a string or a regular expression. |
| ^\ (Ctrl+\) | Replaces a string or a regular expression. |

**The vi Editor**

The vi editor is a utility that creates and modifies text files. It is the defacto command line text editor included with most Linux distributions.

The vi editor uses the following operational modes:

* Command mode is the initial mode vi uses when started. It provides commands that can cut and replace text. It is also the mode from which you access the other vi modes.
* Command line mode is used to load files, and to save files after editing them in the file system.
* Edit mode is the mode that vi uses to write and edit text in the file. It has two operation modes:
  + Insert mode adds text between the preceding and subsequent text.
  + Replace mode overwrites subsequent text.

The table below lists some of the most common vi commands:

| **Command** | **Function** | **Mode** |
| --- | --- | --- |
| **vi** | Starts vi. Type the command at the shell prompt. | N/A |
| **vi** **[file\_name]** | Starts vi and immediately begins working on the named file (either a new file or an existing file). Type the **vi** command at the shell prompt. | N/A |
| **Insert** key **i s** | Enters insert mode from command mode. | Command |
| **Esc** key | Enters command mode from edit mode. | Insert/Replace |
| **Delete** key | Deletes text. | Insert/Replace |
| **Insert** key | Toggles between the insert and replace modes while in edit mode. | Insert/Replace |
| **z** | Exits without saving. | Command |
| **:** | Enters command line mode from command mode. | Command |
| **w** | Saves the current document. | Command liwne |
| **w** **[file\_name]** | Names and saves the file. | Command line |
| **w![file\_name]** | Overwrites the file. | Command line |
| **q** | Exits vi. This produces an error if the text was modified. | Command line |
| **q!** | Exits vi without saving. | Command line |
| **wq** or **exit** | Saves the document and exits vi. | Command line |

# Section 1.1Linux Introduction

As you study this section, answer the following questions:

* What year was Linux created?
* Who originally created Linux?
* What are the most common server roles for Linux servers?
* What is the development cycle for Linux?
* What is a Linux kernel?
* What are Linux distributions?
* What is a Linux user interface? What are the two most common types of user interfaces?

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| Linux kernel | The core of the Linux operating system. It is, in fact, the actual operating system itself. |
| Graphical user interface (GUI) | A graphical user interface similar to the ones seen in other operating systems such as Windows. When a user wants to complete some task, they can click on buttons or navigate through menus to accomplish the desired task. |
| Distribution | A unique compilation of the Linux kernel (free and open to all) with utilities, desktop environments, applications, and more. |

This section helps you prepare for the following certification exam objectives:

| **Exam** | **Objective** |
| --- | --- |
| CompTIA Linux+ XK0-005 | 2.1 Summarize the purpose and use of security best practices in a Linux environment   * Manage public key infrastructure (PKI) certificates   + Certificate authorities |

# **1.1.3Linux Introduction Facts**

Linux has become a major player in computer networking and plays a vital role for servers and desktops—it even works with mobile devices. Understanding the origin of Linux and how it has evolved to its prominent place in the industry is an important part of your training.

This lesson covers the following topics:

* Linux development history
* Linux expansion and growth
* Linux operating system overview
* Linux key components
* Linux distributions
* Other Linux implementations

**Linux Development History**

The Linux operating system had its start in 1991 when, as a graduate student at Finland's University of Helsinki, Linus Torvalds began a project that later became the Linux kernel. Linus based his version of Linux on a Unix-like system named MINIX, which was released by Andrew S. Tanenbaum. Linux version 0.02 was released in October of 1991 and consisted of the Linux kernel and its three basic utilities. These were:

* A Bash shell providing a command line interface.
* An update utility used for flushing the file system buffers.
* A GCC (GNU Compiler Collection) compiler system allowing an individual to write their own programs.

The source code for the Linux operating system was shared as freeware on the internet, and others were encouraged to enhance it and make it better. At this point, Linux took on a life of its own, and it became a worldwide collaborative development project with no secrecy or tightly guarded copyrights. Access to the source code was open to anyone who wanted it. This collaborative development project on Linux continued for several years until 1994, when Linux version 1.0 was released.

The Linux kernel is licensed under the GNU General Public License (GPL), which requires the source code to remain freely available to anybody who wants it. GNU is a recursive acronym for "GNU's Not Unix!"

**Linux Expansion and Growth**

When Linux was first released, it was considered an experimental operating system. So it was something that you might experiment within a lab, but probably never consider putting into a production environment. Since that time, things have changed dramatically, and Linux is now a mainstay operating system, especially in server rooms. Using the wide variety of network services that are now available for the Linux operating system, you can configure it to perform almost any networking role that any competing server operating system can perform, including as a:

* File server
* Print server
* Database server
* Web server
* Email server

Linux is also slowly becoming more popular as a desktop operating system due to the many applications that are currently available. Many of these applications are free.

A few desktop application examples include:

| **Application** | **Description** |
| --- | --- |
| LibreOffice | A free office software suite for word processing, spreadsheets, and presentations. |
| Apache OpenOffice | A free office software suite for word processing, spreadsheets, and presentations. |
| GIMP | GIMP is an acronym for GNU Image Manipulation Program. It is a free and open-source image editor, similar to Photoshop. |
| LightWorks | An editing tool available in free and for-purchase versions. |

**Linux Operating System Overview**

As an operating system, Linux provides the following key functions of a computer:

| **Function** | **Definition** |
| --- | --- |
| Application platform | The operating system provides a platform where applications can run. |
| Hardware interface | A key function of the operating system is to ensure that one application running on the system does not try to use an area in memory that's already in use by another application. It is also responsible for ensuring that a given application running on the system does not monopolize the CPU time so that other applications running on the system cannot use the CPU. |
| Data storage | The operating system is responsible for providing an efficient and reliable means for storing data. This is usually done using some type of storage device, like a hard disk drive formatted with a particular file system. The file system's job is to organize the information on the hard disk in an easily retrievable format. |
| Security | The operating system is responsible for providing some degree of security for the data that's stored on its storage devices. For example, the system administrator can create rules and assign permissions that determine who can access what information on the system. |
| Network connectivity | The operating system provides some type of connectivity between computer systems over a network connection. They can do this using a variety of different network media and interfaces, such as an Ethernet connection between computer systems. There are other standards that can be used to create network connections, such as mobile broadband wireless or Wi-Fi wireless. |

**Linux Key Components**

| **Component** | **Description** |
| --- | --- |
| Linux kernel (operating system) | The Linux kernel is the core of the Linux operating system. It is the actual operating system itself. Linux also provides libraries. Libraries contain pre-written code elements that programmers can use, such as how to interface with a hard disk. For example, when a programmer needs to write data to a hard disk, he or she does not need to know whether the machine has a SATA, IDE, or SCSI drive installed. Instead, the programmer simply calls the appropriate library and tells the operating system that it needs to write data to whatever storage device is installed in the system, and then the operating system takes care of the rest using its libraries. |
| Utilities | The Linux operating system includes a wide variety of utilities that can complete operating system management tasks, such as creating files and maintaining file systems, editing text files, managing the applications that are running on the system, installing new applications on the system, etc. |
| User interfaces | The Linux operating system provides the end user with a means of interacting with the operating system, the user interface.  Linux provides two different user interfaces:   * Graphical user interface: a Linux graphical user interface (GUI) is similar to the GUIs used in other operating systems, such as Windows. When a user wants to complete some task, they can click on buttons or navigate through menus to accomplish the desired task. * A Linux graphical user interface * Text-based command line interfaces: a text-based interface (often referred to as a terminal or shell) provides a place where the user can type commands. This is similar to Windows Command Prompt and PowerShell. Linux system administrator needs to know how to perform tasks from the text-based interface because most Linux servers disable the graphical user interface to better utilize the systems memory and processor.  A Linux text-based command line interface or terminal window |

# Linux Distributions

Simply put, a Linux distribution (also known as a distro) is a unique compilation of the Linux kernel, utilities, desktop environments, applications, and more.

Since the Linux operating system is not produced by a single organization, different organizations combine the desired components they want to use, sometimes creating their own unique features. Then they compile them into their own flavor of a Linux operating system or distribution. This distribution is then often made available at no cost or, in some cases (usually for server versions of Linux), for a fee. Individuals can also create their own distribution, but the process of compiling the software can be time consuming, and it is difficult to make all of the different programs work together properly.

There are hundreds of distributions available. Some of the most popular include:

* Mint
* Ubuntu
* Debian
* Fedora
* openSUSE
* Red Hat Enterprise Linux
* Oracle
* CentOS

# Other Linux Implementations

Linux is also useful in the following implementations:

| **Implementation** | **Description** |
| --- | --- |
| Linux on mobile devices | Linux has nearly taken over the mobile device market in the form of the Android operating system. The current Android operating system is a specialized Linux distribution created by Google. It was designed primarily for touchscreen mobile devices, such as smartphones and tablet computers.  Android benefits include:   * Cost—since Android is based on the Linux kernel, it is much less expensive than other mobile device operating systems, like iOS or Windows RT. * Performance—android performs extremely well on mobile devices. * Application support—there are many apps available for Android devices. In most cases, these apps allow Android devices the ability to provide the same functionality as the more expensive devices from Apple and Microsoft. |
| Linux virtualization | Virtualization is the ability to install and run multiple operating systems concurrently on a single physical machine. This is typically accomplished using a hypervisor. A hypervisor is a thin layer of software that resides between the guest operating system and the hardware. A hypervisor allows virtual machines to interact with the hardware without going through the host operating system. The Linux operating system can be virtualized.  A key benefit of virtualization is a more efficient use of system resources. All of the available computing capacity of the system hardware is allocated and distributed among all the virtual machines running on the system. Another benefit of virtualization is the ability to run multiple platforms at the same time. For example, you can run Windows at the same time that you are running Linux. This can be a real benefit for Linux software developers and testers. It also makes it much easier to test how an application being developed will perform on different platforms or different versions of a given operating system. |
| Linux and cloud computing | In cloud computing, the hardware, software, and/or network resources that have historically been implemented on-site are moved offsite. When a new Linux system is required, you can use an internet cloud provider to deploy the new Linux virtual machine using a hypervisor at their site. You then pay that provider a fee to access this virtual machine through your organization's network connection. This process is referred to as Infrastructure as a Service (IaaS).  Other cloud computing options for Linux include:   * Software as a Service (SaaS), which provides access to software and data through the cloud. * Network as a Service (NaaS), which provides network connectivity through the cloud. * Storage as a Service (STaaS), which provides access to storage devices through the cloud. |
| Embedded Linux | Embedded Linux is the process of embedding Linux within intelligent devices, such as automation and control equipment, smart TVs, smartphones, and tablets. To accomplish this, the operating system is customized so that it only provides the functions required by that particular device, and all the remaining unnecessary elements of the Linux kernel are removed. Once that's done, the kernel itself is embedded in flash memory chips on the given device. |

# **1.1.4Server Roles Facts**

This lesson covers Linux server roles.

**Linux Server Roles**

As a result of the continued popularity, growth, and development of Linux, it can now fulfill many server roles. In most cases, a server role is an application or process installed on a Linux server. In some cases, a Linux server may be dedicated to run or fulfill a single server role. While at other times, several server roles may be functioning on the same server.

The following table summarizes and explains several of the Linux server roles.

| **Role** | **Description** |
| --- | --- |
| NTP | The Network Time Protocol (NTP) is used to synchronize the time on your Linux system with a centralized NTP server. A local NTP server on the network can be synchronized with an external timing source to keep all the servers in your organization in sync with an accurate time. NTP uses a hierarchy of clocks and computers for synchronizing the current time.  NTP uses stepping to quickly make large adjustments to close wide time discrepancies, usually about once every 60 seconds. For example, if there's a big differential between the provider's time and the time on your local system, NTP will adjust the time on your local system in small increments until the time eventually becomes synchronized. |
| SSH | SSH (Secure Shell or Secure Socket Shell) is a protocol used to securely log onto remote systems using encryption. SSH is the most common way to access a remote Linux system. OpenSSH is an open source implementation of the Secure Shell (SSH) protocol and implemented by default on most Linux distributions.  Two major components include the SSH client and the SSH server. The SSH client is a program that is typically only run as needed. Once installed, the SSH server is a daemon that constantly runs in the background. |
| Web server | A web server is the program responsible for accepting HTTP (Hypertext Transfer Protocol) requests from web browsers or clients and, in turn, sending the clients the files that form webpages. For example, webpages often consist of HTML (Hypertext Markup Language) documents and linked objects, such as images. A machine that has been dedicated to performing this role is also called a web server.  A few examples of Linux web server implementations include:   * Apache server * Nginx * Lighttpd * Apache Tomcat * Monkey HTTP Daemon |
| Certificate authority | A digital certificate is an electronic document that can be used as proof of identification. For example, digital certificates are used between an end user and a bank to establish a trusted connection. As an end user, we trust digital certificates because they are created by trusted entities, called certificate authorities (CAs). A few of the most public certificate authorities include GeoTrust, Comodo, Digicert, Thawte, Verisign and GoDaddy. These CAs require the person or company applying for a certificate (such as your bank) to provide documents and information that proves they are who they claim to be.  At times, you may find that using digital certificates within your own organization can be beneficial. For example, when using VPNs, you could use a digital certificate for authentication instead of a pre-shared key. Digital certificates could also be useful for development and staging systems. Rather than paying a public certificate authority for digital certificates for your internal needs, you can configure a Linux system to be a certification authority. One method of doing this is to use OpenSSL, a free open-source library. |
| Name server | A name server resolves (or maps) the fully qualified domain names (FQDNs), such as www.TestOut.com, to their respective IP addresses, and IP addresses to their respective FQDNs. For example, this lets a user access the TestOut site from their web browser by entering https://www.TestOut.com instead of something like https://104.16.32.53.  In many cases, you need to download and install a name resolver software on your Linux system to enable the name server features. The Berkeley Internet Name Domain (BIND) software is an example of one of the most widely used DNS software on the internet. |
| DHCP | The Dynamic Host Configuration protocol (DHCP) centralizes IP address assignment management by allowing a server (such as a Linux server) to dynamically assign IP addresses to clients. DHCP also allows users who move from network to network to easily obtain an IP address appropriate for the subnet they are connected to. The DHCP server and the client use broadcasts to communicate with each other. In many cases, you need to download and install the DHCP server software. For example, for an Ubuntu server, enter: **$ sudo apt install isc-dhcp-server** at the command prompt.Configuration of the server can then be completed as needed. |
| SNMP | The Simple Network Management Protocol (SNMP) is designed for managing complex networks, and is used to communicate with and monitor network devices, servers, and more by means of the IP protocol. SNMP lets network hosts exchange configuration and status information. For example, SNMP can be used to remotely retrieve the operational statistics of a router or firewall. On a Linux machine, SNMP runs as a daemon. In many cases, you need to download and install SNMP. For example, to install SNMP on a CentOS system ,enter: **yum -y install net-snmp net-snmp-utils** at the command prompt. |
| File servers | A Linux file server is a machine that has been set up and configured to let other machines store and retrieve files to and from a central location. In addition, using a file server can simplify backups and security. Using SMB shares and a variety of programs such as Samba or Network File System, a Linux file server can share files with other Linux systems, as well as with non-Linux systems, such as Windows and Mac. |
| Authentication server | Most enterprise networks require centralized user authentication and access controls for all system resources. This is not only convenient for users, but also allows an administrator to monitor and audit user types and the type of access they have on each machine. It also makes provisioning and disabling user accounts easier.  Linux centralized authentication (an authentication server) can be accomplished in many ways, depending on the Linux distribution being used. Some options include installing and using OpenLDAP (Lightweight Directory Access Protocol) or purchasing programs that aid in the installation and management of centralized authentication, such as FreeIPA Identity & Access Manager. |
| Proxy | A proxy is a computer that provides indirect internet access to the computers in your network. In most cases, a proxy server is installed on the same computer as the firewall. Proxy servers provide increased performance and security by blocking direct access between two networks, such as the corporate network and the internet. Proxy can be configured in a variety of ways, such as using SSH tunneling or installing an app on a system that has been configured as a web server. |
| Logging | An important Linux role is the ability to capture a timeline of events that have taken place on the computer in the form of a file, which is referred to as a log file. The process of creating these logs is known as logging. Logging is enabled by default, and logs are often captured for such things as services, the Linux operating system, and applications. Logging is useful for troubleshooting, security, and evaluating server performance. You can configure a centralized logging server, making it easier to evaluate and use the logs created on many systems. Although log files can be stored in a variety of places, most are stored in the /var/log directory or a subdirectory thereof. |
| Containers | Linux containers give you the ability to run an application (with all of the necessary libraries, dependencies, and files) in an isolated environment known as an image or container. Due to this isolation, multiple containers can be run on the same host without affecting each other or the main operating system. All containers utilize and share the same operating system kernel of the host machine, making them very lightweight and fast.  Containers are highly portable. When you move or copy a container from one host to another, all of the files and changes necessary to run the applications within the container are moved or copied with it. Moving a container to a new host does not impact the host operating system. Although Linux containers are extremely portable, they must be compatible with the underlying system. For example, x86 Linux systems run x86 containers while ARM Linux systems only run ARM Linux containers. However, an x86 Linux system cannot run an ARM Linux container. |
| VPN | A VPN (Virtual Private Network) can be installed on a Linux host and is a type of network that uses encryption to allow IP traffic to travel securely over the TCP/IP network. A VPN is primarily used to support secure communications over an untrusted network (for example, connecting two remote site by means of the internet). |
| Monitoring | Monitoring refers to the process of monitoring the essential Linux services, including operating system metrics, process state, logs, service state, and file system usage. It also refers to monitoring servers' availability. Depending on the Linux distribution, monitoring information can often be gathered manually using command line monitoring tools, such as **top**, **lsof**, **tcdump**, and **vmstat**. Web-based utilities (such as Monit and Nagios) can also be installed, which usually provides some type of user interface that makes seeing and analyzing the information easier. |
| Database | A database is a structured set of data held in a computer, especially one that is accessible in various ways. In simpler terms, a database is an organized collection of various forms of data. The information stored in a database is typically organized into rows, columns, and tables. Database information is also indexed to make it easier to find the information required. Many open-source databases are available for Linux, which allow you to manage large chunks of data in a secure way with high performance abilities. Many versions of Linux databases can be installed on your Linux system, such as MySQL, Apache Derby, and PostgreSQL. |
| Print server | A print server is used when a company wants to make a printer available to multiple users over a network. Print servers accept the print jobs from the users and stores them in a queue. When the appropriate printer is available, the job is sent from the queue to the printer. In addition, a print server makes printer queue and status information available to end users and network administrators.  The Common UNIX Printing System, or CUPS, is the most common Linux printing system in use today. CUPS manages print jobs and queues and provides network printing using the standard Internet Printing Protocol (IPP). |
| Mail server | A mail server is a computer that sends, receives, and stores email for users.  When a user creates an email, he or she does so using a mail user agent (MUA) (such as Evolution, Mozilla Thunderbird, or Mutt). The MUA must be configured to send and receive mail by means of a mail server or a Linux system where the mail transfer agent (MTA) has been installed. It is the MTA’s responsibility to then either save the message so it can be downloaded by another local user or, using the internet, send the email to the destination MTA where it will be stored for download by the intended user.  Some Linux distribution may have a default MTA that can be configured and used. If one does not exist or you want to use a different email system, other MTAs can be downloaded and installed. A few common MTAs include Postfix and Qmail. |
| Load balancer | When a company has back-end servers that receive a significant amount of traffic (such as Netfilx, Hulu, or Airbnb), response time to these servers can be increased through load balancers by distributing the workload across the available servers.  Although load balancers can be purchased as a hardware appliance, software can be installed on a Linux server, making it a load balancer. Three common Linux load balancers include Linux Virtual Server (a free and open-source project), Nginx, and HAProxy, all of which run on top of Linux. Some of the load balancer software is free, and some must be purchased. |
| Clustering | With clustering, two or more servers are grouped together in a way to make them work like one.  Clustering is often used to create a failover system, a load balance system, or a parallel processing unit. A failover cluster means that if one system fails, the other servers will take over the load, giving end-users uninterrupted access to the desired data. There are many options for building a Linux cluster, including using free open-source software (such as OpenHPC) or purchasing a commercial product. |

# **1.1.5Practice Questions**

## **Question 1 of 10**

Linux can be implemented in many different ways.

Drag the implementation type on the left to the definition on the right that BEST matches.

Keyboard Instructions

Running Linux and Windows on the same physical computer.

Linux virtualization

Correct Answer:

Infrastructure as a Service (IaaS).

Linux and Cloud Computing

Correct Answer:

Used by Google on many of the physical products it sells.

Linux on mobile devices

Correct Answer:

Manages intelligent devices, such as automation and control equipment.

Embedded Linux

Correct Answer:

**Explanation**

Linux on mobile devices  
Linux has nearly taken over the mobile device market in the form of the Android operating system. The current Android operating system is a specialized Linux distribution created by Google. It was designed primarily for touch screen mobile devices, such as smart phones and tablet computers.

Linux Virtualization  
Virtualization is the ability to install and run multiple operating systems concurrently on a single physical machine. The Linux operating system can be virtualized.

Embedded Linux  
Embedded Linux is the process of embedding Linux within intelligent devices, such as automation and control equipment, smart TVs, smart phones, and tablets. To accomplish this, the operating system is reworked and customized in such a way that it provides only the functions required by that particular device and all the remaining unnecessary elements of the Linux kernel are removed. Once that's done, the kernel itself is embedded in flash memory chips on the given device.

Linux and Cloud Computing  
In cloud computing, the hardware, software, and/or network resources that have historically been implemented onsite are moved offsite. When a new Linux system is required, you can use an internet cloud provider to deploy the new Linux virtual machine using a hypervisor at their site. You then pay that provider a fee to access this virtual machine through your organization's network connection. This process is referred to as Infrastructure as a Service (IaaS).

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_linux\_intro\_facts\_implementation\_type\_def\_lp6\_01.question.fex

## **Question 2 of 10**

When working with Linux, it is important to understand what distributions are and how they are used.

Which of the following BEST describes a Linux distribution?

###answer###

Correct Answer:

A unique bundling of the Linux kernel.

The kernel used to build a Linux operating system.

A pre-written code element programmers can use within programs.

An application capable of running on Linux.

**Explanation**

A Linux distribution (also known as a distro) is a unique bundling of the Linux kernel, plus other applications and services.

Because the Linux operating system is not produced by a single organization, different organizations combine the desired components they want to use, sometimes creating their own unique features, and then compile them into their own flavor of a Linux operating system or distribution.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_linux\_intro\_facts\_linux\_distribution\_def\_lp6\_01.question.fex

## **Question 3 of 10**

Your company develops applications to run on Linux systems. You currently have four development teams, each working on a different aspect of the same application.

Which of the following server roles would give you the BEST method for testing all team members' code without affecting your part of the project or your operating system and personal files?

###answer###

Correct Answer:

Containers

Load balancer

Monitoring

Clustering

Database

**Explanation**

Linux containers give you the ability to run an application (with all of the necessary libraries, dependencies, and files) in an isolated environment known as an image or container. Due to this isolation, multiple containers can run on the same host without affecting each other or the main operating system. All containers utilize and share the same operating system kernel of the host machine, making them very lightweight and fast.  
  
Containers are highly portable. When you move or copy a container from one host to another, all of the files and changes necessary to run the applications within the container are moved or copied with it. Moving a container to a new host does not impact the host operating system. Although Linux containers are extremely portable, they must be compatible with the underlying system. For example, x86 Linux systems will run x86 containers, while ARM Linux systems only run ARM Linux containers. However, an x86 Linux system cannot run an ARM Linux container.

With clustering, two or more servers are grouped together to make them work like one. Clustering is often used to create a failover system, a load balance system, or a parallel processing unit.

Load balancers distribute workload across available servers, which increases response time.

Monitoring refers to the process of monitoring the essential Linux services, including operating system metrics, process state, logs, service state, and file system usage. It also refers to monitoring servers' availability.

A database is a structured set of data held in a computer, especially one that is accessible in various ways. In simpler terms, a database is an organized collection of various forms of data.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_server\_roles\_containers\_sol\_02\_lp6.question.fex

## **Question 4 of 10**

Alex, a webmaster, is implementing an order processing system on the company's website.

Which of the following server roles should Alex implement with the order processing application?

###answer###

Correct Answer:

Database

Clustering

Monitoring

VPN

**Explanation**

A database server should be implemented with the order processing application to store the data gathered by the application.

Monitoring refers to the process of monitoring essential Linux services, including operating system metrics, process state, logs, service state, and file system usage.

Clustering is often used to create a failover system, a load balance system, or a parallel processing unit. A failover cluster means that if one system fails, the other servers will take over the load, giving end users uninterrupted access to the desired data.

A VPN (Virtual Private Network) is a type of network that uses encryption to allow IP traffic to travel securely over the TCP/IP network.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_server\_roles\_database\_lp6.question.fex

## **Question 5 of 10**

Drag the server role on the left to its proper description on the right.

Keyboard Instructions

A protocol used to communicate with and monitor network devices and servers.

SNMP

Correct Answer:

A protocol used to securely log on to remote systems using encryption.

SSH

Correct Answer:

Resolves (or maps) the fully qualified domain names (FQDNs) to IP addresses.

Name server

Correct Answer:

A program responsible for accepting HTTP (Hypertext Transfer Protocol) requests from clients.

Web server

Correct Answer:

**Explanation**

SSH (Secure Shell or Secure Socket Shell) is a protocol used to securely log onto remote systems using encryption. SSH is the most common way to access a remote Linux system. OpenSSH is an open-source implementation of the Secure Shell (SSH) protocol and is implemented on most Linux distributions by default.

A web server is responsible for accepting HTTP (Hypertext Transfer Protocol) requests from web browsers or clients and, in turn, sending the clients the files that form webpages. For example, webpages often consist of HTML (Hypertext Markup Language) documents and linked objects, such as images. A machine that has been dedicated to perform this role is also called a web server.

A name server resolves (or maps) the fully qualified domain names (FQDNs), such as www.TestOut.com, to their respective IP addresses and IP addresses to their respective FQDNs. This would let a user access the TestOut site from a web browser by entering https://www.TestOut.com instead of something like https://104/16/32/53.

The Simple Network Management Protocol (SNMP) is a protocol designed for managing complex networks and is used to communicate with and monitor network devices, servers, and other devices through the IP protocol. SNMP lets network hosts exchange configuration and status information. For example, SNMP can be used to remotely retrieve the operational statistics of a router or a firewall. On a Linux machine, SNMP runs as a daemon.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_server\_roles\_descriptions\_01\_lp6.question.fex

## **Question 6 of 10**

Drag the server role on the left to its proper description on the right.

Keyboard Instructions

Capturing a timeline of events that have taken place on the computer in the form of a file.

Logging

Correct Answer:

A type of network that uses encryption to allow IP traffic to travel securely over the TCP/IP network.

VPN

Correct Answer:

Increases response time to back-end servers by distributing the workload across the available servers.

Load balancer

Correct Answer:

A computer that provides indirect internet access to the computers in your network.

Proxy

Correct Answer:

**Explanation**

Proxy: A proxy is a computer that provides indirect internet access to the computers in your network. In most cases, a proxy server is installed on the same computer as the firewall. Proxy servers provide increased performance and security by blocking direct access between two networks, such as the corporate network and the internet. Proxy can be configured in a variety of ways, such as using SSH tunneling or installing an app on a system that has been configured as a web server.

Logging: An important Linux role is the ability to capture a timeline of events that have taken place on the computer in the form of a file, referred to as a log file. The process of creating these logs is known as logging. Logging is enabled by default and logs are often captured for such things as services, the Linux operating system, and applications. Logging is useful for such things as troubleshooting, security, and evaluating server performance. If desired, you can configure a centralized logging server making it easier to evaluate and use the logs created on many systems. Although log files can be stored in a variety of places, most logs are stored in the /var/log directory or a subdirectory thereof.

VPN: A VPN (Virtual Private Network) can be installed on a Linux host and is a type of network that uses encryption to allow IP traffic to travel securely over the TCP/IP network. A VPN is used primarily to support secure communications over an untrusted network. For example, connecting two remote sites by means of the internet.

Load balancer: When a company has back-end servers that receive a significant amount of traffic (such as Netfilx, Hulu and Airbnb), response time to these servers can be increased through load balancers by distributing the workload across the available servers. Although load balancers can be purchased as a hardware appliance, software can be installed on a Linux server, making it a load balancer. Three common Linux load balancers include Linux Virtual Server (a free and open-source project), Nginx, and HAProxy, all of which run on top of Linux. Some of the load balancer software is free and some are for pay.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_server\_roles\_descriptions\_02\_lp6.question.fex

## **Question 7 of 10**

Your company uses both Linux desktops and Windows desktops.

Which of the following server roles should you use to provide a central location for users of both operating systems to share files?

###answer###

Database

Authentication server

Correct Answer:

File servers

Proxy

**Explanation**

A Linux file server is a machine that has been set up and configured to let other machines store and retrieve files to and from a central location. In addition, using a file server can simplify backups and security. Using SMB shares and a variety of programs such as Samba or Network File System, a Linux file server can share files with other Linux systems, as well as with non-Linux systems such as Windows and Mac.

Linux centralized authentication (an authentication server) can be accomplished in many ways, depending on the Linux distribution being used.

A proxy is a computer that provides indirect internet access to the computers in your network. In most cases, a proxy server is installed on the same computer as the firewall.

A database is a structured set of data held in a computer, especially one that is accessible in various ways. In simpler terms, a database is an organized collection of various forms of data.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_server\_roles\_file\_servers\_lp6.question.fex

## **Question 8 of 10**

Your company has been expanding the number of servers in the company's data center, and there is an increased need to gather metrics, watch process states, work with logs, watch services states and file system usage.

Which of the following sever roles should be installed to provide this functionality?

###answer###

Containers

Database

Logging

Correct Answer:

Monitoring

**Explanation**

Monitoring refers to the process of monitoring the essential Linux services, including such things as operating system metrics, process state, logs, service state, and file system usage. It also refers to monitoring servers' availability.

Depending on the Linux distribution, monitoring information can often be gathered manually using command line monitoring tools, such as top , lsof , tcdump , and vmstat . Web-based utilities (such as Monit and Nagios) can also be installed, which usually provides some type of user interface that makes seeing and analyzing the information easier.

A database is a structured set of data held in a computer, especially one that is accessible in various ways. In simpler terms, a database is an organized collection of various forms of data.

An important Linux role is the ability to capture a timeline of events that have taken place on the computer in the form of a file, which is referred to as a log file.

Linux containers give you the ability to run an application (with all of the necessary libraries, dependencies, and files) in an isolated environment known as an image or container.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_server\_roles\_monitoring\_lp6.question.fex

## **Question 9 of 10**

Which of the following is the primary role of a mail transfer agent (MTA)?

###answer###

Provide redundant online storage for the mail sever.

Transfer mail to a print server queue.

Correct Answer:

Store messages so they can be downloaded or send email to a destination MTA.

Control the bandwidth used by mail user agent (MUA).

**Explanation**

After a mail transfer agent (MTA) has been installed, it's the MTA's responsibility to then either save the message to be downloaded by another local user or, using the internet, send the email to the destination MTA, where it will be stored for download by the intended user.

The MUA is the email client, such as Evolution, Mozilla Thunderbird, or Mutt.

Redundant online storage and transferring email to a print server are not performed by the MTA.

**References**

TestOut Linux Pro 6.0 - 1.1 Linux Introduction

q\_server\_roles\_mta\_lp6.question.fex

## **Question 10 of 10**

An IT technician has been given a work order to install the Apache web server on a system configured with a YUM repository.

Which of the following commands will install the web server?

###answer###

rpm -ivh apache2

dnf install httpd

yum install apache2

Correct Answer:

yum install httpd

**Explanation**

yum install httpd is used to install Apache on a system using a YUM repository.

dnf install httpd would work on systems where dnf is used instead of YUM.

yum install apache2 will return *No package apache2 available*.

rpm -ivh apache2 will return *No such file or directory*. The rpm command needs the full .rpm file name.

# **Section 2.1The Linux Shell**

As you study this section, answer the following questions:

* What is a shell?
* What are the basic differences between common shells?
* Which common commands are used to navigate through shells?
* Which items can you use the Tab key to complete after beginning to type the name?
* What does the tilde symbol ( **~** ) in the prompt indicate?

In this lesson, you will learn to:

* Enter shell commands
* Work with the Linux shell

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| Linux shell | A command line interface (CLI) or text user interface (TUI) for the Linux operating system. |
| Shell prompt | A character or set of characters at the start of the command line that indicates that the shell is ready to receive commands. |
| PATH environment variable | An shell environment variable that contains the set of directories that are searched when you type an executable file at the shell prompt. |
| Tab complete | A shell feature that attempts to complete a command, file, or directory after pressing the Tab key when the first few characters of a command, file, or directory are typed at the shell prompt. |
| Command history | A shell feature that stores shell commands and displays them when you press the Up and Down arrow keys. |

This section helps you prepare for the following certification exam objectives:

| **Exam** | **Objective** |
| --- | --- |
| TestOut Linux Pro | 1.2 Configure and use Linux shell environments   * Manage environment variables * View available shells |
| CompTIA Linux+ XK0-005 | 1.2 Given a scenario, manage files and directories   * File and directory operations   + ls   + pwd   + cd   + ~   + .   2.2 Given a scenario, implement identity management   * Account creation and deletion   + Default shell   + Configuration files     - /etc/passwd   2.4 Given a scenario, configure and execute remote connectivity for system management   * Commands   + su -   2.5 Given a scenario, apply the appropriate access controls   * Command-line utilities   + ls   3.1 Given a scenario, create simple shell scripts to automate common tasks.   * Shell script elements   + Shell built-in commands     - echo * Environment variables   + $PATH   + $SHELL |

# **2.1.2Linux Shell Facts**

The Linux shell is often described as a Command Line Interface (CLI) or Text User Interface (TUI) to the Linux Operating System.

This lesson covers the following topics:

* Linux shell definition and use
* Linux shell access methods
* Linux shell types
* Linux shell common characteristics

**Linux Shell Definition and Use**

A Linux shell is a program that traditionally provides the text user interface (TUI) for Linux. The term shell is descriptive since it is considered an outer layer of the operating system. The shell is the interface between you, a user or administrator, and the internal parts of the operating system, including its very core, which is the kernel. The shell's main function is to read and parse your commands and then execute them through interactions with the Linux kernel.

Most Linux distributions offer a workstation version that includes a graphical user interface (GUI), and many administrative tasks can be completed within the graphical environment. Since a GUI can diminish server performance, many Linux distributions offer a server version where the GUI is either disabled or is not installed. Whether you are managing a Linux workstation version or a server version, you perform much of your administrative tasks using shell commands. In addition, you will find that graphical environments and tools may vary between distributions, but shell commands are more likely to be consistent.

**Linux Shell Access Methods**

The Linux shell can be accessed:

| **Access Method** | **Description** |
| --- | --- |
| From the Console (The keyboard and monitor attached to a computer running Linux) | Press the following key combinations:   * **Ctrl+Alt+F1** (in some Linux distributions, **Alt+F1** ) will start or switch to the first Linux shell session. * **Ctrl+Alt+F2** through **Ctrl+Alt+F6** (in some Linux distributions, **Alt+F2** through **Alt+F6** ) will switch to the second through sixth shell sessions.   While Linux distributions will vary, in most cases, when a GUI is installed, it will use the first console session. Use **Ctrl+Alt+F1** to switch to a graphical login screen. If you are already logged in, **Ctrl+Alt+F7** will switch to the desktop GUI. |
| From a desktop GUI | Open a terminal session from the applications menu. |

**Linux Shell Types**

The following table describes many common shell types:

| **Shell Type** | **Description** |
| --- | --- |
| sh | The Bourne shell is the oldest Linux shell but is not widely used. This shell was developed for UNIX in the 1970s. |
| Bash | The Bourne-Again shell (Bash) is the default shell used by most Linux distributions. It uses commands similar to a UNIX shell. The Bash includes:   * Command and file name completion when pressing the Tab key. * Command history. |
| zsh | The Z shell (zsh) is an improved version of Bash and is available on many Linux distributions. |
| shh | The Bourne shell (sh) is an earlier version of Bash and is similar in many ways. The sh shell was originally created by Steve Bourne. |
| ksh | The Korn shell (ksh) provides scripting features not found in Bash. Ksh was developed by David Korn. |
| csh | The C shell (csh) uses syntax similar to the syntax used in the C programming language. |
| tcsh | The tcsh shell is an improved version of csh. It offers command line editing and completion features that are not available in csh. |

If a shell type is installed, it will be listed in the **/etc/shells** file. You can switch to any shell by typing the shell's name. For example, type **ksh** within any shell to switch to the Korn shell.

The following commands can be used to determine the default shell type and the current shell type.

| **Command** | **Description** |
| --- | --- |
| **echo $SHELL** | The **echo $SHELL** command returns the default or preferred shell. The environment **$SHELL** holds the user's preferred shell, which is typically set in **/etc/passwd** . For example, if Bash is the user's preferred shell, **$SHELL** will hold the value **/bin/bash** . The preferred shell does not change when you switch shell types. |
| **echo $0** | The **echo $0** command can be used to return the current shell type. The special variable **$0** normally holds the command used to start a script. A shell session is initiated by a script. Consequently, **$0** will hold the name of the script, which is consistent with the shell type. For example, **$0** in a console session running bash might hold the value **-bash** , $0 in a terminal application opened from a desktop menu might hold the value **bash** . When you type **ksh** , the **ksh** script is run, a Korn shell session is opened, and **$0** will hold the value **ksh** . |

**Linux Shell Common Characteristics**

Despite their differences, all shells share some common characteristics.

* The shell provides a command line interface that allows the user to interact with the Linux kernel.
* A Linux system can run multiple shell sessions at the same time.
* One shell session can run within another shell session. This may be done interactively, such as when a user starts a second shell from the first shell's command line or automatically, by scripts or programs.
* Shells use configuration files to establish their operating environments.

# **2.1.6Linux Shell Command Facts**

The Linux shell is a text user interface that provides a command line interface (CLI). Users employ the shell to interact with the Linux kernel by typing commands at the shell prompt.

This lesson covers the following topics:

* Bash shell command line prompt
* $PATH Linux shell environment variable
* Running an executable file
* Command history feature
* Tab complete feature
* Common Linux shell commands

**Bash Shell Command Line Prompt**

Each Linux shell type has a customizable command line prompt. For the Bash shell, the default command line prompt varies with each Linux distribution but is generally displayed in the following format:

* The username of the current user
* The @ symbol
* The hostname
* A space
* The base name of the current working directory

If the current working directory is the home directory (the default directory when the user first logs in), the tilde symbol ( **~** ) is displayed instead.

* A character that indicates the type of user:
  + The # character typically indicates that the current user is the root user.
  + The $ character typically indicates that the current user is a normal user.

**$PATH Linux Shell Environment Variable**

The $PATH environment variable contains the set of directories that are searched when you type an executable file at the shell prompt.

* The shell does not look (by default) in the current working directory for the executable file.
* To see the list of directories, type **echo $PATH** at the shell prompt.
* To add a directory to a path, type **PATH=$PATH:[directory\_path]** and then type **export PATH** .

**Running an Executable File**

The following table describes how to run executable files.

| **File Location** | **Run Action** |
| --- | --- |
| Resides in a directory that is included in the $PATH environment variable. | Type the filename at the shell prompt. |
| Resides in the current working directory, and the current directory is not included within the path environment variable. | Type **./** followed by the filename. |
| Does not reside in the current working directory, and its directory is not included within the path environment variable. | Type the full or relative path to the executable file. |

File names and paths are case-sensitive.

**Command History Feature**

The Linux shell keeps a history of commands you type at the shell prompt.

* The commands are stored as a history queue within the hidden .bash\_history file in your home directory.
* Press the **Up** and **Down** keys to scroll through your previously typed commands.
  + You can edit the command or re-run it as is by pressing **Enter**.
* Type **history** to display the commands stored in the history queue.
* Type **history -c** to clear the history queue.
* Using a **space** before a command prevents the command from appearing in the history.

**Tab Complete Feature**

**Tab**

* After typing the beginning of a command, file, or directory, press **Tab** to complete it.
* If pressing Tab does not fully complete the command, file, or directory, press **Tab** again.
  + If there is no matching command, file, or directory name, your command entry will not change.
  + If there is more than one matching command, file, or directory name, a list of all matches will be displayed to help you resolve your command entry.

**Common Linux Shell Commands**

The following table describes several common commands used at the shell prompt.

| **Command** | **Function** |
| --- | --- |
| **pwd** | Shows the present working directory. |
| **whoami** | Displays the current username. |
| **uname** | Prints system information. The **uname** command has the following options:   * **-a** prints all system information. * **-o** prints the operating system. * **-p** prints the processor's architecture type. |
| **su** | Switches users in the shell prompt. The **su** command has the following options:   * **su -l** **[username]** switches to the specified user and creates a new login shell. * **su** **[username]** (without the dash, but with the username) switches to the user in the current shell. * **su -** **[username]** (with the dash and the username) switches to the user and loads that user's environmental variables. * **su -** (with the dash but without the username) switches to the root user and loads the root user's environmental variables.   + The root user account is the Linux system superuser.   + The root user can perform any task. Some utilities do not work if the administrator is not logged in as the root user. * **su** (no dash or username) switches to the root user but does not load the root user's environmental variables.   **su** requires the password of the user except when switching from root to a normal user. |
| **exit** | Exits the current shell (which may close the login shell) or to go back to the original user after using the **su** command. |
| **exec** | Executes an executable to replace the shell process with the new process created by the executable file. |
| **cd** | Changes directories. For example, when the **/usr** directory is the current directory:   * **cd bin** changes to the **bin** directory in the current directory. * **cd /usr/bin** changes to the **/usr/bin** directory from anywhere in the file system. |
| **ls** | Shows names of files and directories in the current directory. The **ls** command has the following options:   * **-a** shows all files and directories, including hidden files. * **-l** shows extended information about files, including size, permissions, owner, and modified date. * **-d** displays only directories. * **-s** sorts files by size. * **-X** sorts by extension.   Many distributions use a color scheme to identify different file types as follows:   * Directories are blue. * Text files are white. * Links are cyan. * Executable files are green. * Compressed files are red. |
| **history** | Shows all the commands in the history queue. The **-c** option clears the history list.  History command queues are separate for each user. For example, a command typed as one user cannot be used after using the **su** command to switch to another user. |
| **clear** | Clears the shell screen. |
| **chsh** | Changes the default shell. The **chsh** command has the following options:   * **-s** changes to a different installed shell. The command prompts for a password. * **-l** lists all installed shells.   For example, **chsh -s /bin/ksh [username]** changes the default shell for the user to the Korn shell if it is installed on the computer. |

# **2.1.7Practice Questions**

**Question 1 of 10**

**Next**

Listen to exam instructions

You are at a terminal command prompt in a Linux GUI interface that someone else has been using and want to find out which shell type is currently being used by the terminal.

What command can you enter to find out the current shell type?

answer

Correct Answer:

echo $0

echo $PATH

Incorrect answer:

echo $SHELL

echo $UID

**Explanation**

You can enter the echo $0 command to find out the current shell type. The variable $0 normally holds the command used to start a script. A shell session is initiated by a script. Consequently, $0 will hold the name of the script which is consistent with the shell type. For example, $0 in a console session running bash might hold the value -bash.

The echo $SHELL command returns the default or preferred shell. However, this value may not reflect the current shell type if someone else has switch from the preferred shell to another shell type.

The echo $PATH command returns a list of directories in which the system looks for executable files.

The echo $UID command returns the current user's unique ID.

**Question 2 of 10**

**BackNext**

Listen to exam instructions

In which of the following files is the user's preferred shell normally set?

answer

Correct Answer:

/etc/passwd

/etc/shadow

/etc/profile

/etc/group

**Explanation**

The user's preferred shell is normally set in the /etc/passwd file.

The /etc/group file defines the groups to which users belong.

The /etc/shadow file stores encrypted user passwords.

The /etc/profile file normally defines the PATH variable, user limits, and other settings.

**Question 3 of 10**

**BackNext**

Listen to exam instructions

Which of the following is the standard shell for MOST Linux computers?

answer

Bourne shell

C-shell

Correct Answer:

Bourne again shell (bash)

tcsh

Korn

**Explanation**

The Bourne again shell (Bash) is the standard shell used in most Linux computers. It uses commands similar to a UNIX shell. Bash includes features such as:

* Command completion when pressing the tab key
* Command history
* Improved arithmetic functions

The Bourne shell (Sh) is an earlier version of the Bash shell. It is similar in many ways. Sh is the original shell created by Steve Bourne.

The Korn shell (ksh) was developed by David Korn. Ksh has scripting features not found in bash.

The C-shell (csh) uses syntax similar to syntax used in the C programming language.

The tcsh shell is an improved version of csh. It offers command line editing and completion features not available with csh.

**Question 4 of 10**

**BackNext**

Listen to exam instructions

What would you enter at a command prompt to start a new Bourne again shell (Bash) session?

correct answer selected

**Explanation**

The bash command opens a Bourne again shell (Bash) session. Bash is the standard shell used in most Linux computers. It uses commands similar to a UNIX shell. Bash includes features such as:

* Command completion when pressing the tab key
* Command history
* Improved arithmetic functions

**Question 5 of 10**

You are managing a Linux server with a console (without a GUI installed).

Which of the following key combinations would you press to switch to the fourth shell session?

answer

Ctrl+4

Ctrl+F4

Ctrl+Shift+4

Correct Answer:

Ctrl+Alt+F4

**Explanation**

Ctrl+Alt+F1 through Ctrl+Alt+F6 (in some Linux distributions Alt+F1 through Alt+F6) will switch to the first through the sixth shell sessions.

**Question 6 of 10**

**BackNext**

Listen to exam instructions

A Linux user has an executable file named *ni*that can save a snapshot of network information with the date and time in a log file. The executable ni file is in the /root directory, and /root is the current working directory.

Which of the following commands would run the executable file? (Select two.)

answer

exec ni

ni

source ni

Correct Answer:

/root/ni

Correct Answer:

./ni

**Explanation**

To run an executable, you can either change to the directory where the script is stored and type ./ni or type the absolute path (in this case, /root/ni) to run the script from any directory.

Typing just the file name, ni, will not work because the current working directory, /root, is not typically contained in the PATH variable.

The source ni command is typically used within a shell script to read and execute commands within the ni file. In this case, the ni file would not be found, since the current working directory, /root, is typically not contained in the PATH variable.

The exec ni command is used to execute a command that completely replaces the current process. In this case, the ni file would not be found since the current working directory, /root, is typically not contained in the PATH variable.

**Question 7 of 10**

As a Linux user, you have access to an executable file named myapp. It's found in the current directory, but not in the command path.

What would you enter at the command prompt to start the myapp file and replace the shell with myapp process?

correct answer selected

**Explanation**

Use exec ./myapp to start the myapp executable file and replace the shell with myapp process.

The exec command executes an executable not found in the command path. It also replaces the shell with the new process created by the executable file. ./ indicates that the executable is in the current directory.

**Question 8 of 10**

**BackNext**

Listen to exam instructions

Which of the following commands should a Linux user enter to see a list of all the commands the user recently ran at the command prompt?

answer

clear

Correct Answer:

history

uname

chsh

**Explanation**

Use history to see all commands in the history queue. The -c option clears the history list. History command queues are separate for each user. A command typed as one user cannot be used after using the su command to switch to another user.

The clear command clears the shell screen, but does not clear the command history.

The chsh command changes the default shell. For example, chsh -s /bin/ksh changes the default shell for the user to the Korn shell if it is installed on the computer.

The uname command prints system information.

**Question 9 of 10**

**BackNext**

Listen to exam instructions

What would you enter at the command prompt to display the current working directory?

correct answer selected

**Explanation**

Use the pwd command to display the current working directory.

**Question 10 of 10**

**BackNext**

Listen to exam instructions

Which of the following is displayed when the uname -a command is run?

answer

The current username

The current working directory

Correct Answer:

All system information

The names of files and directories in the current directory

**Explanation**

The uname -a command displays all system information.

The pwd command displays the present working directory.

The whoami command displays the current username.

The ls command displays names of files and directories in the current directory.

# **Section 2.2 Linux Help**

As you study this section, answer the following questions:

* What are the differences between the **man** utility and the **info** utility?
* What types of files are stored in the **/usr/share/man** directory?
* How can you access on-screen help for each command?

In this lesson, you will learn to:

* Access help resources

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| Man page | A text-based help file for a specific command, service, or configuration file that shows a command's syntax, options, related files, and commands. |
| Info node | Text-based help information similar to a man page but more verbose and emphasizes how to use a Linux command or utility. |
| Whatis database | A database containing short man page descriptions. |

This section helps you prepare for the following certification exam objectives:

| **Exam** | **Objective** |
| --- | --- |
| TestOut Linux Pro | 1.1 Use command-line utilities   * Get help with Linux commands |

# **2.2.3Help Facts**

There are several resources available to you that provide help with Linux commands, utilities, and configuration files.

This lesson covers the following topics:

* Man interface
* Info reader
* On-screen help and the help command
* The /usr/share/doc directory
* Whatis database

**Man Interface**

The man interface displays manual (man) pages, which are text-based help files for a specific command, service, or configuration file. A man page shows a command's syntax, options, and related files and commands.

* You can open the man interface to view a man page using **man *command or file*** .
  + Use the **-k** option to search all man pages for a specific search term.
  + Example: **man userdel** shows the man pages for the **userdel** command.
  + Example: **man -k user** lists all the commands with *user* in the command name or description.
* Man pages are typically stored in the **/usr/man** or **/usr/share/man** directory. Subdirectories store man pages for different types of commands or languages.
  + Some distributions use the MANPATH environment variable to identify the location where man pages are stored. Type **echo $MANPATH** to see the value of this variable.
  + Some distributions use the MANPATH\_MAP directive in the /etc/man\_db.conf or /etc/manpath.config file to specify the directories where man pages are stored.
* A man page usually consists of several elements that are used to organize the information.
  + The TITLE section is the first line of the man page. It lists the name of the command or file followed by the section number.
  + The NAME section displays the name of the command or file and a short summary that describes what it does.
  + The SYNOPSIS section reviews the syntax for using a command or a brief overview of what a configuration file is used for.
  + The DESCRIPTION section describes how the command or configuration file works. It also provides a list of options that can be used.
  + The AUTHOR section displays the name of the programmer who wrote the command or service referenced in the man page.
  + The REPORTING BUGS section provides contact information to report any bugs you discover.
  + The COPYRIGHT section identifies who owns the copyright to the command or service referenced in the man page.
  + The SEE ALSO section lists man pages or other resources that are related to the current man page.
  + The last line of a man page displays the version number and revision date of the command or service.
* Use the following keystrokes to navigate within man pages.
  + Use the **Up arrow** key to move up one line.
  + Use the **Down arrow** key to move down one line.
  + Use the **PgUp** key to move up one display page.
  + Use the **PgDn** key or the space bar to move down one display page.
  + Use the **Home** key to go to the beginning of the man page.
  + Use the **End** key to go to the end of the man page.
  + Use the **/** key to search for text within the man page and the n key to move to the next occurrence of the searched-for text.
  + Use the **q** key to exit the man page.

**Info Reader**

The Info reader displays Info pages (which are called Info nodes). Info nodes are similar to man pages but are more verbose and emphasize how to use Linux commands and utilities. Info nodes use hypertext links to navigate between nodes.

* You can open the Info reader to view an Info node command using **info *command or file*** .
  + Example: **info mkdir** opens the Info page for the **mkdir** command.
* The top line of the display shows the next and previous Info nodes.
* Use the following keystrokes to navigate within Info nodes.
  + Use the **Up** **arrow** key to move up one line.
  + Use the **Down** **arrow** key to move down one line.
  + Use the **PgUp** key or the Del key to move up one display page.
  + Use the **PgDn** key or the space bar to move down one display page.
  + Use the **h** key to show the Info help node.
  + Use the **Tab** key to move to the next hyperlink within the Info nodes.
  + Use the **Home** key to move to the beginning of a node.
  + Use the **Enter** key to follow the selected hyperlink.
  + Use the **n** key to move to the next Info node.
  + Use the **p** key to move to the previous Info node.
  + Use the **q** key to exit the Info reader.

**On-Screen Help and the help Command**

Many commands include on-screen help as one of the options for the command.

* Typically, to view the help available for a command, type *:*
  + **[command] --help**
    - Example: **jobs --help** displays help on the **jobs** command.
  + **[** **command]** **-h**
    - Example: **echo -h** displays help on the **echo** command.
* If the command is typed incorrectly, many commands display the help information automatically.
* In most cases, help is an abbreviated list that shows the command syntax and available options with brief descriptions.
* Although some commands support both switches, the **--help** switch is more common.
* If necessary, pipe the command through **more** using the command **[command] --help | more** to scroll through the on-screen help.

**help**

* To view information about a built-in command that matches a pattern of characters, type:
  + **help [pattern]**
    - Example: **help exec** displays help on the exec command.
    - Example: **help ex** displays help on three commands: **exec** , **exit** , and **export** .
* The help command has three options:
  + **help -d** outputs a short description for each topic.
  + **help -m** displays command usage in a pseudo-man page format.
  + **help -s** outputs only a short usage synopsis for each matching topic.

**The /usr/share/doc Directory**

The /usr/share/doc directory contains documentation resources for many of the different components of the Linux system.

* Within the /usr/share/doc directory are subdirectories for many components.
* Typical files in each subdirectory include:
  + NEWS
  + README
  + AUTHOR
  + OVERVIEW
  + COPYING
* Use the **cat** command to display the contents of the files in each subdirectory.

**Whatis Database**

**whatis**

* The **whatis [keyword]** command displays one-line man page descriptions for commands that match the keyword exactly.
  + The **whatis** results are intended to give you a general idea of what a command or component will do.
  + Example: **whatis grep**
  + If the whatis database has not been created, run the **/usr/sbin/makewhatis** command.
  + Use **makewhatis -u** to update the database after changes have been made to the man pages.
* The **apropos [keyword]** command displays one-line man page descriptions for any command or man page description that contains the keyword.
  + The **apropos** results are intended to help you learn what command or program to use when performing a certain task.
  + Example: **apropos declare**
* Both the **whatis** and **apropos** commands use the whatis database.
  + If the whatis database has not been created, run the **/usr/sbin/makewhatis** command.
  + Use **makewhatis -u** to update the database after changes have been made to the man pages.

# **2.2.4Get Help**

You are a student learning how to use Linux. Your teacher has given you the assignment to learn how to use a few basic commands. Complete this lab from the Terminal.

In this lab, your task is to:

* Learn the options available for the **killall** command.
  + Answer Question 1.
* Learn the options available for the **useradd** command.
  + Answer Question 2.
* Learn the options available for the **usermod** command.
  + Answer Question 3.
* Start Lab

*Q1*Which option can be added to the **killall** command to kill a process group instead of just a process?

Your answer:-g

Correct answer:-g

*Q2*What does the **-M** switch do when used with the **useradd** command?

Your answer:The home directory is created, regardless of system settings.

Correct answer:The home directory is not created, regardless of system settings.

*Q3*What is the result of adding **-G** to the **usermod** command as a single option?

Your answer:The user is assigned to a new list of supplementary groups.

Correct answer:The user is assigned to a new list of supplementary groups.

**EXPLANATION**

Complete this lab as follows:

1. Use the **--help** option to learn more about the switches that can be used for the **killall** command.
   1. From the Favorites bar, select **Terminal**.
   2. From the prompt, type **killall --help** and press **Enter**to view the help information for this command.

Help for Linux commands can also be seen using the **-h** option. Example: **killall -h**

* 1. From the top right, select **Answer Questions**.
  2. Answer Question 1.

1. Use the **--help** option to learn more about the switches for the **useradd** command.
   1. From the prompt, type **useradd --help** and press **Enter** to view the help information for this command.
   2. Answer Question 2.
2. Use the **--help** option to learn more about the switches for the **usermod** command.
   1. From the prompt, type **usermod --help** and press **Enter** to view the help information for this command.
   2. Answer Question 3.
   3. From the Lab Questions dialog, select **Score Lab**.

# **2.2.5Practice Questions**

You have just entered the apropos grep command at the command prompt. What can you expect to be displayed?

answer

The appropriate use of the grep command in your Linux system.

Correct Answer:

A one-line man page description for the grep command.

The apropos information in the grep command man pages.

The syntax available for the grep command.

Explanation

When you enter the apropos grep command, a one-line man page description is displayed for the command.

You would enter help -s grep to display the syntax for the command.

There is no specific "apropos" information in the grep command pages.

There is no specific category of help information that outlines the appropriate use of the grep command.

References

TestOut Linux Pro 6.0 - 2.2 Linux Help

q\_help\_appropos\_command\_lp6.question.fex

**Question 2 of 10**

**BackNext**

Listen to exam instructions

Which of the following man page sections shows a list of options available for a Linux command and explains what the options do?

answer

TITLE

SYNOPSIS

NAME

Correct Answer:

DESCRIPTION

**Explanation**

The DESCRIPTION section typically lists the options that are available for the command and explains their purpose and use.

The SYNOPSIS section shows a brief description of the syntax for using the command.

The NAME section gives the name of the command and a short description of what the command does.

The TITLE section shows the title of the man page and the section of the manual it is found in.

**References**

TestOut Linux Pro 6.0 - 2.2 Linux Help

q\_help\_description\_section\_lp6.question.fex

**Question 3 of 10**

You have forgotten what information is provided by the whoami command, and simply want a short description of the command.

Which of the following help commands could you enter to view that short description?

answer

Correct Answer:

help -d whoami

Incorrect answer:

help -s whoami

help whoami

help -m whoami

**Explanation**

Entering help -d whoami would provide a short description of the whoami command.

Entering help whoami would provide all help information about the whoami command.

Entering help -s whoami would provide the syntax for the whoami command.

Entering help -m whoami would display all the help information in a man page format.

**Question 4 of 10**

**BackNext**

Listen to exam instructions

If you are viewing the contents of a man page, which key can you press to get back to the beginning of the page?

incorrect answer. Correct Answer:Home

**Explanation**

Pressing the Home key moves you back to the beginning of the man page.

**Question 5 of 10**

**BackNext**

Listen to exam instructions

Which of the following commands would you enter to view info node information about the sysctrl command?

answer

info -k sysctrl

info "systctrl"

info -w sysctrl

Correct Answer:

info sysctrl

**Explanation**

You would enter the info sysctrl command to view the info node pages for the sysctrl command.

The info -k sysctrl command would searches all info indices for any mention of sysctrl.

The info "sysctrl" command is not a valid info command string.

The info -w sysctrl command would display the location of the sysctrl info file page in your Linux system.

**Question 6 of 10**

**BackNext**

Listen to exam instructions

Which of the following are characteristics make info nodes different from man pages? (Select two.)

answer

Correct Answer:

They are more verbose and emphasize how to use commands and utilities.

You can use the End key to go to the end of a page.

Correct Answer:

They contain hypertext links to navigate between nodes.

You can use the PgUp and PgDn keys to move one page at a time.

They contain a SYNOPSIS section for reviewing the syntax of a command.

**Explanation**

The following are characteristics of info nodes that make them different from man pages:

* They are more verbose and emphasize how to use commands and utilities.
* They contain hypertext links to navigate between nodes.

You can use the PgUp and PgDn keys to move one page at a time for both man pages and info nodes.

Only man pages contain a standard SYNOPSIS section.

You use the End key to go to the end of page for man pages.

**Question 7 of 10**

**BackNext**

Listen to exam instructions

You want to search all the man pages for information available for the Bluetooth protocol.

Which of the following commands would you enter to display this information?

answer

Correct Answer:

man -k bluetooth

man -w bluetooth

man -f bluetooth

man bluetooth

**Explanation**

The man -k bluetooth command searches all man pages for any mention of bluetooth.

The man bluetooth command displays all the man pages for the bluetooth command only.

The man -f bluetooth command looks for the sections in which a bluetooth command is present.

The man -w bluetooth command displays the location of the bluetooth man page in your Linux system.

**Question 8 of 10**

**BackNext**

Listen to exam instructions

You are working on a Linux system and need more information about the uname command. What would you enter at the command prompt to learn about the uname command syntax and options?

correct answer selected

**Explanation**

To learn about the uname command syntax and options, type any of the following at the command prompt:

* man uname displays the man page for the uname command. A man page shows the command's syntax, options, and related files and commands.
* info uname displays the info page for the uname command. For GNU software, info pages are the primary documentation source. Info pages use hypertext links to navigate the pages.
* uname --help displays the on-screen help for the uname command. (For some commands, the -h option displays on-screen help as well.) In most cases, help displays an abbreviated list that shows of the command syntax and available options with brief descriptions. If the command is typed incorrectly, many commands display the help information automatically.

**Question 9 of 10**

**BackNext**

Listen to exam instructions

You want to view the Linux help information for the mkdir command. While viewing the information, you want to be able to scroll through the onscreen help.

What is the command you would use to display and be able to scroll through the mkdir help information?

incorrect answer. Correct Answer:mkdir --help | more

**Explanation**

Depending on your Linux system, you would enter mkdir --help | more or mkdir -h | more to let you scroll through the onscreen mkdir help.

**Question 10 of 10**

**BackNext**

Listen to exam instructions

Which of the following commands searches man pages for a specific keyword? (Select three.)

answer

Correct Answer:

whatis

slocate

find

help

Correct Answer:

man -k

export

Correct Answer:

apropos

**Explanation**

To search the man pages, use the man -k, apropos, or whatis commands. The whatis database must be created prior to using these commands. Use the makewhatis command to build the database.

The find command searches for files based on file name, size, and other file attributes.

The slocate command searches for files based on file name, size, and other file attributes.

The help command displays information about the builtin command, but not from man pages.

The export command exports a shell variables for use by a child process.

# **Section 2.3Text Editors**

As you study this section, answer the following questions:

* Why is it important to know how to run a text editor on a Linux system?
* What are the four different modes available in the vi editor, and how can you tell the current mode?
* Which vi mode would you use to overwrite existing text?
* How can you list all the keystroke shortcuts in the nano text editor?
* Which text editor, vi or nano, should you use?

In this lesson, you will learn to:

* Create a new file
* Modify an existing file
* Use the nano editor

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| vi or vim | A utility that creates and modifies text files. Due to the *vim* interface being virtually identical to *vi* , the names *vi* and *vim* have become interchangeable, and the shorter name *vi* is used when describing both editors. |
| vi operating modes | Four different modes that the vi text editor operates under are Command Mode, Command Line Mode, Insert Mode, and Replace Mode. |
| vi commands | Keystroke sequences that control the vi editor, including cursor movement, cutting, copying and pasting text, finding and replacing text, and writing the buffer to a file. |
| nano | A utility that creates and modifies text files. |

This section helps you prepare for the following certification exam objectives:

| **Exam** | **Objective** |
| --- | --- |
| TestOut Linux Pro | 1.1 Use command-line utilities   * View, search, and edit the content of text files   2.2 Manage the file system   * Create, copy, move, and delete files in the file system |
| CompTIA Linux+ XK0-005 | 1.2 Given a scenario, manage files and directories   * File editing   + nano   + vi(m) |

# **2.3.4Text Editor Facts**

Traditionally, many Linux system configurations are made by editing text files. Therefore, knowing how to use a text editor is an important Linux skill. This is especially important when configuring Linux servers that are not configured with a GUI.

This lesson covers the following topics:

* The vi editor
* The nano editor
* Choosing a text editor

**The vi Editor**

The vi or vim editor is a utility that creates and modifies text files. It is the de facto command line text editor included with most Linux distributions.

Most Linux distributions actually include the vim editor instead of vi. The name vim is an acronym for "Vi Improved," which indicates that vim extends the vi functionality. Due to the vim interface being virtually identical to vi, the names vi and vim have become interchangeable, and the shorter name vi is used when describing both editors.

The vi editor uses the following operational modes:

* Command mode is the initial mode vi uses when started. It provides commands that can cut and replace text. It is also the mode from which you access the other vi modes.
* Command line mode is used to load files and save files after editing them in the file system.
* Edit mode is the mode that vi uses to write and edit text in the file. It has two operation modes:
  + Insert mode adds text between the preceding and subsequent text.
  + Replace mode overwrites subsequent text.

The table below lists some of the most common vi commands:

| **Command** | **Function** | **Mode** |
| --- | --- | --- |
| **vi** | Starts vi. Type the command at the shell prompt. | N/A |
| **vi** **[file\_name]** | Starts vi and immediately begins working on the named file (either a new file or an existing file). Type the **vi** command at the shell prompt. | N/A |
| **Insert** key **i s** | Enters insert mode from command mode. | Command |
| **Esc** key | Enters command mode from edit mode. | Insert/Replace |
| **Delete** key | Deletes text. | Insert/Replace |
| **Insert** key | Toggles between the insert and replace modes while in edit mode. | Insert/Replace |
| ***#* [line\_number]** | Goes to a specific line in the document while in command mode. For example, **#94** moves the cursor to line 94. | Command |
| **dw** | Cuts a whole word and trailing space. | Command |
| **de** | Cuts a whole word but omits the trailing space. | Command |
| **d$** or **D** | Cuts all text following the cursor to the end of the line. | Command |
| **dd** | Cuts a line from the text. | Command |
| **p** | Places text in memory into the document. | Command |
| **u** | Undoes the last action. | Command |
| **O** | Opens a new line above the current line. | Command |
| **o** | Opens a new line below the current line. | Command |
| **Ctrl+g** | Displays the file name, the total number of lines in the file, and the cursor position. | Command |
| **/[term]** | Searches forward for all instances of a term. Press n to go to the next term and N to go to the previous term. | Command |
| **?[term]** | Searches backward for all instances of a term. Press n to go to the previous term and N to go to the next term. | Command |
| **yy** | Copies a line of text into memory. | Command |
| **a** | Appends text after the cursor. | Command |
| **A** | Appends text after the current line. | Command |
| **C** | Changes text from the current cursor position to the end of the line. | Command |
| **cc** | Changes text of the entire line. | Command |
| **ZZ** | Saves the current file and exits vi. | Command |
| **h** | Moves the cursor one space to the left. | Command |
| **j** | Moves the cursor down a line. | Command |
| **k** | Moves the cursor up a line. | Command |
| **l** | Moves the cursor one space to the right. | Command |
| **z** | Exits without saving. | Command |
| **:** | Enters command line mode from command mode. | Command |
| **w** | Saves the current document. | Command line |
| **w** **[file\_name]** | Names and saves the file. | Command line |
| **w![file\_name]** | Overwrites the file. | Command line |
| **q** | Exits vi. This produces an error if the text was modified. | Command line |
| **q!** | Exits vi without saving. | Command line |
| **wq** or **exit** | Saves the document and exits vi. | Command line |
| **e!** | Reloads the file from the last saved version. This discards all edits and reloads the last saved version of the file into vi. | Command line |

**The nano Editor**

The nano editor is included in most Linux distributions.

* To start the editor, type **nano** at the shell prompt.
* The nano editor is simpler to use than the vi editor.
  + Common keystroke shortcuts are listed at the bottom of the nano interface.
    - A caret (^) in the shortcut means press and hold the Ctrl key.
    - 'M-' listed in the shortcut means press and hold the Meta key. On modern keyboards, the Alt or Esc key is substituted for the Meta key.
  + Ctrl+G (listed as ^G) displays the full help text, which includes a full list of shortcuts.
  + Prompts are displayed when user input is needed.
  + The Home, End, PgUp, PgDn, and arrow keys move the cursor in the same way as other common editors.

The table below lists some common nano shortcuts:

| **Shortcut** | **Function** |
| --- | --- |
| ^G (Ctrl+G) | Displays the help text, which includes a list of all keyboard shortcuts. |
| ^X (Ctrl+X) | Closes the current buffer or exits from nano. |
| ^O (Ctrl+O) | Writes the current buffer (or the marked region) to disk. |
| M-Space (Alt+Space or Esc+Space) | Goes back one word. |
| ^Space (Ctrl+Space) | Goes forward one word. |
| M-A (Alt+A or Esc+A) | Marks text starting from the cursor position. |
| M-6 (Alt+6 or Esc+6) | Copies current line (or marked region) and stores it in cutbuffer. |
| ^K (Ctrl+K) | Cuts current line (or marked region) and stores it in cutbuffer. |
| ^U (Ctrl+U) | Uncuts (paste) from the cutbuffer into the current line. |
| ^W (Ctrl+W) | Searches forward for a string or a regular expression. |
| ^\ (Ctrl+\) | Replaces a string or a regular expression. |

**Choosing a Text Editor**

There are a host of text editors available for the Linux platform. The vi and nano editors have proven to be the most popular, largely due to their availability. A highlight of the differences between vi and nano may help when choosing a text editor.

| **Attribute** | **The vi Editor** | **The nano Editor** |
| --- | --- | --- |
| Availability | Included in virtually every Linux distribution. | Included in most Linux distributions. |
| Licensing | BSD License or CDDL (free and open source software). | GNU General Public License (GPL) (free software license). |
| Interface Complexity | Non-intuitive, especially due to the different operational modes. | Intuitive, mostly due to the onscreen display of keystroke shortcuts. |
| Feature set | Full and complex feature set. | Basic feature set. |
| Learning curve | May take a prolonged time to learn due to its complex feature set. | Usually learned quickly, especially for users having experience with other editors. |

# **2.3.5Create a New File**

You need to create a script file to change and export the SHELL environmental variable as the C shell. You have decided to use the vim editor to do this.

In this lab, your task is to:

* Use **vim** to open a new file named **/etc/pref\_shell**.
* Add the following lines to the new file:  
  **SHELL=/bin/csh  
  export SHELL**
* Save and close the file.

# **2.3.8Practice Questions**

**Question 1 of 10**

Which of the following nano editor keyboard shortcuts displays help text, including a list of all keyboard shortcuts?

answer

^O (Ctrl+O)

M-A (Alt+A or Esc+A)

M-Space (Alt+Space or Esc+Space)

Correct Answer:

^G (Ctrl+G)

**Explanation**

The ^G (Ctrl+G) keyboard shortcut displays help text which includes a list of all keyboard shortcuts.

The ^O (Ctrl+O) keyboard shortcut writes the current buffer (or the marked region) to disk.

The M-A (Alt+A or Esc+A) keyboard shortcut is used to mark text that is to be cut or copied from the cursor position.

The M-Space (Alt+Space or Esc+Space) keyboard shortcut moves the cursor forward one word.

**Question 2 of 10**

**BackNext**

Listen to exam instructions

Which of the following nano text editor features makes it easier for beginners to learn than the vi text editor?

answer

It is included in most Linux distributions.

Correct Answer:

Keyboard shortcuts are displayed at the bottom of the editor.

The software falls under the GNU General Public License.

There are multiple operational modes.

**Explanation**

The nano text editor is considered by most beginning users to be easier to learn because the keyboard shortcuts are displayed at the bottom of the editor.

While the nano text editor falls under the GNU General Public License, which makes it essential free software, the vi text editor also falls under the BSD License of CDDL, which makes it free and open source software.

The nano text editor is included in most Linux distributions. The vi text editor is included in virtually every Linux distribution. However, this availability does not make one editor easier for beginners to learn.

The vi text editor has multiple operational modes. The nano editor does not. The vi editor interface is considered to be more complex due to these operational modes.

**Question 3 of 10**

**BackNext**

Listen to exam instructions

Match the nano shortcut on the left to the correct shortcut definition on the right.

Keyboard Instructions

Cuts the current line (or selected text) and stores it in the cutbuffer.

^K (Ctrl+K)

Correct Answer:

Closes the current buffer or exits from nano.

^X (Ctrl+X)

Correct Answer:

Uncuts (pastes) the text from the cutbuffer into the current line.

^U (Ctrl+U)

Correct Answer:

Moves forward one word in the current line.

^Space (Ctrl+Space)

Correct Answer:

Displays the help text, which includes a list of all keyboard shortcuts.

^G (Ctrl+G)

Correct Answer:

**Explanation**

The following are the correct definitions for the listed shortcuts:

* ^G (Ctrl+G) - Displays the help text, which includes a list of all keyboard shortcuts.
* ^X (Ctrl+X) - Closes the current buffer or exits from nano.
* ^Space (Ctrl+Space) - Moves forward one word in the current line.
* ^K (Ctrl+K) - Cuts the current line (or selected text) and stores it in the cutbuffer.
* ^U (Ctrl+U) - Uncuts (pastes) the text from the cutbuffer into the current line.

**Question 4 of 10**

**BackNext**

Listen to exam instructions

Match the vi command on the left with the correct description of the cursor movement the on the right.

Keyboard Instructions

Move the cursor up a line.

k

Correct Answer:

Move the cursor one space to the left.

h

Correct Answer:

Move the cursor one space to the right.

l

Correct Answer:

Move the cursor down a line.

j

Correct Answer:

**Explanation**

To navigate while in vi Command Mode, use the following:

* h moves the cursor one space to the left.
* j moves the cursor down a line.
* k moves the cursor up a line.
* l moves the cursor one space to the right.

**Question 5 of 10**

**BackNext**

Listen to exam instructions

As an IT administrator, you are tasked with configuring an Apache web server by modifying the /etc/apache2/apache2.conf file. You open the file in vi.

As you're looking at the file, you knock your spill-proof drink container onto the keyboard. It rolls over several keys as you try to pick it up. Now there are extra characters everywhere in the text buffer.

Which of the following commands can you use to exit vi without saving any changes that were made to the text buffer?

answer

:x

:wq

Correct Answer:

:q!

:q

**Explanation**

You can use the :q! command to exit without saving the changes made to a file.

The :q command exits vi, but only if there are no changes.

The :wq command saves the file and quits vi.

The :x command saves the file if it has changed and quits vi.

**Question 6 of 10**

**BackNext**

Listen to exam instructions

What command would you enter while in vi Command Mode to find the word *Sam*?

correct answer selected

**Explanation**

You can use either of the following to search for the word *Sam* while in vi Command Mode:

* /Sam searches forward for all instances of a term.
* ?Sam searches backward for all instances of a term.

**Question 7 of 10**

**BackNext**

Listen to exam instructions

A friend sends you a shell script file that is 117 lines long. He says that he wants you to check the code on lines 82 through 87.

What command would you enter while in vi command mode to go directly to line 82?

correct answer selected

**Explanation**

You would use #82 to go to line 82 of the document while in command mode. Typing a colon (:) enters command line mode from command mode.

**Question 8 of 10**

**BackNext**

Listen to exam instructions

Match the vi mode on the left to the correct definition on the right.

Keyboard Instructions

The initial mode used when vi is started. It has commands that cut and replace text, and it is the mode vi uses to enter the other modes.

Command Mode

Correct Answer:

The vi mode that works with the file system. Use it to save files after editing them.

Command Line Mode

Correct Answer:

The vi mode used to write and edit text in the file.

Edit Mode

Correct Answer:

**Explanation**

The vi editor provides the following modes:

* *Command Mode* is the initial mode vi uses when started. It has commands that cut and replace text, and it is the mode vi uses to enter the other modes.
* *Command Line Mode* is the mode that works with the file system. Use it to save files after editing them.
* *Edit Mode* is the mode that vi uses to write and edit text in the file. It has two operation modes:
  + *Insert Mode* adds text between the preceding and subsequent text.
  + *Replace Mode* overwrites subsequent text.

**Question 9 of 10**

**BackNext**

Listen to exam instructions

While in vi Command Mode, you copy a whole line of text to the general buffer. You then navigate to a different location in the file.

Which of the following commands will let you paste the copied text?

answer

dw

Correct Answer:

p

cc

u

**Explanation**

Each of the following commands must be done in Command Mode. To enter Command Mode from Edit Mode, press the <Esc> key.

* You can use the p command to paste text from the text buffer into the document while in command mode.
* The dw command deletes a whole word and trailing space and places the deleted text in the general buffer.
* The u command performs an undo of the last action.
* The cc command deletes the entire line of text at the cursor location and places the deleted text in the general buffer.

**Question 10 of 10**

**BackNext**

Listen to exam instructions

Which of the following vi key combinations should you press while in Insert Mode to save the file you are working on and exit the editor?

answer

<Esc>S

Correct Answer:

<Esc>:wq

<Esc>s

<Esc>:q

**Explanation**

From Insert Mode, press the Esc key to enter the command mode. Enter a colon (:) for command line mode. Press w to write the file and press q to exit the editor (quit).

While vi is in Insert Mode, press Esc to enter the command mode. The s command substitutes one character with a string.

While vi is in Insert Mode, press Esc to enter the command mode. The S command substitutes the current line with a string.

The <Esc>:q key combination enters the command mode and then attempts to exit the editor (quit). The file will not be automatically saved and, if the text buffer has changed, a warning will be displayed.

# **Section 2.4Aliases**

As you study this section, answer the following questions:

* What is the purpose of an alias?
* When would it be important to make an alias persistent?

In this section, you will learn to:

* Use aliases

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| Alias | A shortcut stored in memory that runs a command on your Linux system. |
| Persistent alias | An alias that persists after logging out or rebooting. |

# **2.4.3Alias Facts**

This lesson covers aliases.

**Alias**

An alias is a shortcut stored in memory that runs a command on your Linux system. Most distributions include predefined aliases that are created at system startup. However, a custom alias can be defined from the shell prompt. Be aware of the following:

* Aliases defined with the **alias** command are not persistent across reboots.
* To make aliases persistent across reboots, add the alias definitions to **/etc/profile** or **/home/ *user* /.bashrc** .

The following table describes the commands that create and remove aliases:

| **Command** | **Function** | **Example** |
| --- | --- | --- |
| **alias** | Displays the currently defined aliases on the system. | [rtracy@fs5 ~]$ **alias**  alias egrep='egrep --color=auto'  alias fgrep='fgrep --color=auto'  alias grep='grep --color=auto'  alias l.='ls -d .\* --color=auto'  alias ll='ls -l --color=auto'  alias ls='ls --color=auto'  alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde' |
| **alias [name='command']** | Creates a custom alias that runs an existing command. A single alias can be defined to run multiple commands.  When creating the alias, encapsulate the command(s) with quotation marks or apostrophes. | **alias securebackup='cp ./\*.\* /dev/st0/\*.\*;shred -fuvz ./\*'** creates an alias that copies all files in the current directory to a tape backup device and then shreds the original files.  **alias forcelogout="killall /usr/bin/Xorg"** creates a shortcut that kills all Xserver processes. |
| **unalias [name]** | Removes an alias. | **unalias securebackup** removes the alias specified for the **securebackup** command.  **unalias forcelogout** deletes the forcelogout alias. |

# **2.4.4Practice Questions**

**Question 1 of 10**

**Next**

Listen to exam instructions

What command displays a list of the currently defined aliases on the system?

correct answer selected

**Explanation**

Entering the alias command at the shell prompt displays a list of the currently defined aliases on the system.

**Question 2 of 10**

**BackNext**

Listen to exam instructions

You want to make an alias persistent across all reboots of your Linux server.

In which of the following files do you need to place the alias command to make the alias persistent?

answer

Correct Answer:

/home/*user*/.bashrc

/home/*user*/.bash\_profile

/etc/resolv.conf

Correct Answer:

/etc/profile

/etc/securetty

**Explanation**

To make an alias persistent across reboots, add the alias to the /etc/profile or /home/*user*/.bashrc file.

The /home/*user*/.bash\_profile file saves user settings for the shell.

The /etc/resolv.conf file contains a list of domain name servers (DNS) used by the local machine.

The /etc/securetty file contains a list of terminals where root can log in.

**Question 3 of 10**

**BackNext**

Listen to exam instructions

Which of the following Linux features lets you store a shortcut in memory or as part of the shell configuration file that runs a specific command?

answer

Shell

Background service

Kernel

Correct Answer:

Alias

**Explanation**

An alias is a shortcut stored in memory or as part of the shell configuration file that runs a command on your Linux system.

A background service is a process that runs in the background without any user intervention.

The Linux shell is a command-line interpreter that provides an interface between you and the kernel and runs commands you enter at the shell prompt.

The Linux kernel is the main component of the Linux operating system and provides an interface between the computer hardware and processes.

**Question 4 of 10**

**BackNext**

Listen to exam instructions

You want to create a grepcolor alias that provides a lime green (36) background color to the search terms when using the grep command. You want this command to be persistent when rebooting your Linux server.

What command would you enter into the shell configuration file to create this persistent alias?

correct answer selected

**Explanation**

You would enter one of the following commands in the shell configuration file:

* alias grepcolor="grep --color=36"
* alias grepcolor='grep --color=36'

## Question 5 of 10

**BackNext**

Listen to exam instructions

What commonly predefined alias is configured to run the **ls -l** command?

correct answer selected

### Explanation

The **ll** command is a commonly predefined alias that runs the **ls -l** command, which lists the contents of a directory in long form.

**Question 6 of 10**

**BackNext**

Listen to exam instructions

Mary Brown is a Linux user with the username *mbrown*. Mary has a directory named *logs* in her home directory that is regularly updated with new log files when certain system events occur. She runs the following commands several times a week to check this directory:

* cd /home/mbrown/logs
* ls -al

She wants a persistent alias named *logcheck* to run these two commands. What command would Mary enter into the shell configuration file to create this persistent alias?

correct answer selected

**Explanation**

Mary would enter the following command in the shell configuration file:

alias logcheck="cd /home/mbrown/logs;ls -al"

The two commands need to be inside double quotation marks (") or single quotation marks (') and separated by a semi-colon (;).

**References**

**Question 7 of 10**

**BackNext**

Listen to exam instructions

Which of the following actions defines a persistent alias?

answer

Using the alias command at the shell prompt with the -P option.

Adding the command defining the alias to the $ALIAS environment variable.

Adding the command defining the alias to the /etc/default/alias.conf file.

Correct Answer:

Adding the command defining the alias to the appropriate shell configuration file.

**Explanation**

To make an alias persistent, you need to add the command defining the alias to the appropriate shell configuration file. The name of the shell configuration file varies across Linux distributions. For example, in the Fedora distribution, the shell configuration file is the .bashrc hidden file found in each user's home directory. Another popular location to add aliases is the .profile file.

While the alias command has a -p option to print all defined aliases, there are no options to the alias command that makes the alias persistent.

Creating an ALIAS environment variable has no effect on aliases.

The /etc/profile file is run during all user logins, as are scripts in the /etc/profile.d/ directory. Running the alias command in these scripts creates persistent aliases. The /etc/default/alias.conf files has no effect on aliases.

**Question 8 of 10**

**BackNext**

Listen to exam instructions

Which of the following commands creates a shortcut that can be used to run the tail -f /var/log/messages command?

answer

export alias="tail -f /var/log/messages"

export sysmesg="tail -f /var/log/messages"

env alias="tail -f /var/log/messages"

Correct Answer:

alias sysmesg="tail -f /var/log/messages"

**Explanation**

The alias sysmesg="tail -f /var/log/messages" command creates an alias named *sysmesg* that, when typed at the shell prompt, executes the tail -f /var/log/messages command.

The export command allows an environment variable to be used by a child process.

The env command displays a list of the current environment variables and their values.

**Question 9 of 10**

**BackNext**

Listen to exam instructions

You have created a listgroups alias that you now want to remove.

Which of the following commands will remove the alias?

answer

Correct Answer:

unalias listgroups

unalias "listgroups"

unalias -a listgroups

unalias -a "listgroups"

**Explanation**

The unalias listgroups command will remove the listgroups alias.

No quotation marks are needed when entering the alias name with the unalias command.

The unalias -a command removes all aliases from the current session instead of a specific alias.

**Question 10 of 10**

**BackNext**

Listen to exam instructions

When creating a custom alias, how many commands can you add to the alias?

answer

1

Correct Answer:

1 or more

3 or more

2 or more

**Explanation**

You can add one (1) or more commands to a custom alias.

# **Section 2.5Environment Variables**

As you study this section, answer the following questions:

* What is the role of environment variables?
* What is the standard syntax for environment variable identifiers?
* How do you preserve the current values of the PATH environment variable when adding a new value to it?
* How do you make a new value assigned to an environment variable persistent across all shell sessions?

In this section, you will learn to:

* Manage environment variables
* View environment variables

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| Variable | A data object that has a name and one or more assigned values. |
| Environment variable | A Linux variable that provides settings and configurations across different applications, processes, and services and is inherited by child shells and processes. |
| Shell variable | A Linux variable that is not inherited by child shells and processes. |
| User variable | A Linux shell variable that is typically defined at the shell prompt or added when scripts that are defined in a user's profile are run. |
| Local variable | A Linux shell variable that is typically defined in scripts and in script functions and generally does not exist after the script or function is run. |

This section helps you prepare for the following certification exam objectives:

| **Exam** | **Objective** |
| --- | --- |
| TestOut Linux Pro | 1.2 Configure and use Linux shell environments   * Manage environment variables |
| CompTIA Linux+ XK0-005 | 2.2 Given a scenario, implement identity management   * Account creation and deletion   + Configuration files     - /etc/profile     - .bash\_profile     - .bashrc   3.1 Given a scenario, create simple shell scripts to automate common tasks.   * Common script utilities   + Environment variables     - $PATH     - $SHELL     - $? |

# **2.5.3Environment Variable Facts**

Like most variables, an environment variable is a data object that has a name and a value. In Linux, environment variables provide a way to share settings and configurations across different applications, processes, and services.

This lesson covers the following topics:

* Environment variables vs. shell, user, and local variables
* Create and work with environment variables
* Create persistent environment variables
* Common environment variables

**Environment Variables vs. Shell, User, and Local Variables**

When a shell session is spawned, environment information is gathered from a variety of files and settings on the system and is made available to the shell process. This environment is implemented as key-value pairs or environment variables. Each environment variable has a name or variable identifier, which is conventionally uppercase. Environment variables can be assigned one value or multiple values. If the shell session spawns a new shell or runs a program, this environment (that is, this set of environment variables) is inherited or passed on to the child process.

Shell variables may appear to be environment variables since they also have uppercase names. However, these variables are not inherited from a parent process, but are generated by scripts as the shell is initiated. For example, HISTFILESIZE is created during a Bash shell initialization and determines the number of commands that are stored in a history file. If this Bash shell spawns a C shell as a child process, HISTFILESIZE won't be defined. It isn't inherited from the Bash shell and is not generated when the C shell is initiated. This confusion between environment variables and shell variables is amplified by the fact that users generally work with only one shell type, and the shell variables are initiated across all shell sessions in the same way. This consistency makes shell variables appear to be inherited from a parent shell when, in reality, they are generated during the shell initialization. A further reason for confusion is that many environment variables available in a shell are created and exported when the shell initializes.

All Linux variables can be classified as one of two types: environment variables or shell variables, with the main difference being that environment variables are inherited by child processes, but shell variables aren't.

Both user and local variables are strictly shell variables. If there is any distinction between them, it is in how they are created and used. User variables are typically defined at the shell prompt or added when scripts that are defined in a user's profile are run. Variables defined in scripts and in script functions are usually defined as local variables since they generally do not exist after the script or function is run. By convention, user and local variables are given lowercase names to avoid overwriting environment variables and shell variables.

The table below summarizes these ideas.

| **Variable Type** | **Typical Source** | **Naming Convention** | **Inherited by Child Processes** |
| --- | --- | --- | --- |
| Environment | Inherited from a parent process. It may be the system process that passes environment information gathered from a variety of system files and settings. | Uppercase | Yes |
| Shell | Generated by shell startup scripts. | Uppercase | No |
| User | Created by a user at the shell prompt or added when scripts that are defined in a user's profile are run. | Lowercase | No |
| Local | Defined in scripts and in script functions. | Lowercase | No |

**Create and Work with Environment Variables**

A shell variable becomes an environment variable when the shell variable is exported. Essentially, this changes a variable's export attribute. Be aware of the following when working with environment variables:

* By convention, environment variable names (variable identifiers) are defined and referenced using uppercase characters, making them less likely to be overridden by lowercase shell variable identifiers.
* If an environment variable has multiple values, they are separated by a colon (:) character.
* The default values assigned to environment variables can be overridden to customize the user's computing environment.
  + An overridden environment variable will only apply to the current shell session and any subsequent child processes.

The following table lists command that are used when creating and working with environment variables.

| **Command** | **Function** | **Example** |
| --- | --- | --- |
| **[name]=[value]** | Creates a new shell variable with an assigned value or changes the value of an existing variable.  To append information to a variable instead of replacing it, include the current variable in the command. (See example.) Changing the value of an environment variable will change it for the current shell session and any subsequent child processes. | **TRAINING="TestOut" PATH=$PATH:/bin/additionalpath** |
| **export [name]=[value]** | Creates a new environment variable for the current shell session and any subsequent child processes. | **export TRAINING="TestOut"** |
| **export [name]** | Exports an existing shell variable to make it an environment variable for the current shell session and any subsequent child processes. | **TRAINING="TestOut" export TRAINING** |
| **declare -x [name]=[value]** | Creates a new environment variable for the current shell session and any subsequent child processes (functionally equivalent to **export [name]=[value]** ). | **declare -x TRAINING="TestOut"** |
| **declare -x [name]** | Exports an existing shell variable to make it an environment variable for the current shell session and any subsequent child processes (functionally equivalent to **export [name]** ). | **TRAINING="TestOut" declare -x TRAINING** |
| **export -n [name]** | Removes the export property of an environment variable, making it a shell variable. | **export -n TRAINING** |
| **declare +x [name]** | Removes the export property of an environment variable, making it a shell variable (functionally equivalent to **export -n [name]** ). | **declare +x TRAINING** |
| **echo $[name]** | Displays the contents of an environment variable (or any variable). | **echo $TRAINING** |
| **printenv** | Displays a list of the current environment variables. | **printenv** |
| **env** | Displays a list of the current environment variables.  You can also use the **env** command to run a command using temporarily manipulated environment variables. | **env** |
| **export -p** | Displays a list of all exported variables and functions. | **export -p** |
| **set** | Displays a list of all environment variables, shell variables, local variables, and shell functions.  You can also use the **set** command to set or unset values of shell options and positional parameters. | **set** |
| **unset [name]** | Removes the variable and its value independent of whether the variable is an environment variable or a shell variable. | **unset TRAINING** |

**Create Persistent Environment Variables**

The above commands can be added to the following system and bash configuration files to create environment variables that persist across system reboots and Bash shell startups:

| **File(s)** | **Description** |
| --- | --- |
| /etc/environment | The /etc/environment file is a system-wide configuration file for all users. Environment variables defined in this file are available to all shells and processes.  Changes to this file do not take affect until after a reboot. |
| /etc/profile | This file is run for all users, but only during interactive logins (when a username and password is required to log in). While environment variable can be defined in this file, the preferred method is to define them in a separate script file saved in the /etc/profile.d directory.  An interactive login occurs when the user must supply a username and password. An example is when the user connects using SSH, when using Ctrl+Alt+F2 to log in to a virtual terminal, or when using the **su** (switch user) command. |
| /etc/profile.d/\*.sh | These script files are run for all users, but only during interactive logins. You can add environment variable definitions to these files. These files are run by the /etc/profile file. |
| ~/.bash\_profile ~/.bash\_login ~/.profile | These hidden files are located in a user's home directory and are only run for that specific user and only during interactive logins. A search is made for these files in the order that is listed. Only the first file found will be run. Environment variable definitions can be added to these files. |
| /etc/bashrc | This file is run for all users during a Bash shell initialization. It is run during both interactive logins and interactive non-logins. Environment variable definitions can be added to them.  An interactive non-login occurs when the user is not required to enter a username and password to open a shell. An example is when the user opens a graphical terminal window in gnome or a new shell using the **bash** command. |
| ~/.bashrc | This hidden file is located in a users home directory and is only run for that specific user during both interactive logins and non-interactive logins. You can add environment variable definitions to this file. |

**Common Environment Variables**

The table below lists common environment variables.

A few of these environment variables are exported during shell initialization and aren't inherited from the system process. They're still considered environment variables because they're inherited by any child processes.

| **Variable** | **Description** |
| --- | --- |
| SHELL | The user's login shell. |
| DISPLAY | The location where Windows output is displayed. |
| ENV | The location of the configuration file for the current shell. |
| HISTSIZE | The number of lines or commands that are stored in a history list while your shell session is ongoing. |
| HOME | The absolute path of the user's home directory. |
| HOSTNAME | HOSTNAME is identical to HOST, but is only used on certain distributions. |
| LOGNAME | The current user's username. |
| MAIL | The path to the current user's mailbox file. |
| OLDPWD | The directory the user was in prior to switching to the current directory. |
| PATH | The directory paths used to search for programs and files.   * Use a colon to separate entries in the PATH variable. * Do not include a period (.) in the PATH variable. A period indicates that the working directory is in the path and poses a security risk. |
| PWD | The path of the current working directory. |
| LANG | The language the operating system uses. |
| TERM | The type of terminal to emulate when running the shell. |
| USER | The current logged in user. |

The table below lists useful Bash shell variables:

These shell variables are not inherited by child shells and processes. However, when the child shell is created, these shell variables will be initialized.

| **Variable** | **Description** |
| --- | --- |
| BASH | The location of the Bash executable file. |
| BASHOPTS | The list of options that were used when Bash was executed. |
| BASH\_VERSION | The Bash version being executed in a readable form. |
| BASH\_VERSINFO | The Bash version in machine-readable output. |
| EUID | The current user's ID number. |
| HISTFILE | The filename where past commands are stored. |
| HISTFILESIZE | The number of past commands that HISTFILE stores for multiple sessions. |
| OSTYPE | The type of operating system (usually Linux). |
| PS1 | The characters that the shell uses to define what the shell prompt looks like. |
| COLUMNS | The number of columns being used to draw output on the screen. |
| LINES | The number of lines being used to draw output on the screen. |
| UID | The current user's user ID. |

# **2.5.5Practice Questions**

**Question 1 of 10**

**Next**

Listen to exam instructions

Anna, a system administrator, wants to set an environment variable that can be used each time she logs in. Currently, Anna has to set the variable each time a terminal is opened.

Which of the following files would Anna need to modify to make the variable a global environment variable? (Select two.)

answer

~./bash\_history

~./bash\_logout

Correct Answer:

~./bash\_profile

~/.bashrc

Correct Answer:

/etc/profile

**Explanation**

Anna can modify the ~./bash\_profile and /etc/profile files to add the variable as a global environment variable. Each time Anna logs on, the variable will be set.

~./bash\_history stores the command history.

~./bash\_logout is used to perform cleanup when exiting a bash shell.

~/.bashrc is a hidden file located in a users home directory and is only run for that specific user during both interactive logins and interactive non-logins.

**Question 2 of 10**

**BackNext**

Listen to exam instructions

You want to view the number of commands your bash shell is set to save by examining the current HISTSIZE environment variable. You don't want to have to scroll through all the environment variables.

Which of the following commands is the BEST way to determine the current value of the HISTSIZE variable?

answer

Correct Answer:

echo $HISTSIZE

cat /etc/profile | grep "HISTSIZE"

HISTSIZE=

cat $HISTSIZE

**Explanation**

The echo $HISTSIZE command displays the value for the HISTSIZE environment variable.

You can use the HISTSIZE=[new value] command to change the HISTSIZE environment variable value.

While the cat /etc/profile | grep "HISTSIZE" command displays the default HISTSIZE value, the value may have changed after it was set when the /etc/profile file was run.

Typically, the value of the HISTSIZE variable is 1000. If $HISTSIZE is replace with 1000 in cat $HISTSIZE, the command executed would be cat 1000. If there is a file named 1000, the contents of the file are displayed. Otherwise, the cat command returns a "No such file or directory" message.

**Question 3 of 10**

**BackNext**

Listen to exam instructions

Which of the following commands displays the value of the LANG environmental variable currently set for the language the operating system uses?

answer

echo LANG

Correct Answer:

echo $LANG

echo %LANG

echo %LANG%

**Explanation**

The echo command displays the results of an expression. An expression formed with a dollar sign ($) followed by a variable name results in the assigned value of the variable.

The echo LANG command displays "LANG" without providing the assigned value of the variable.

The echo %LANG command displays "%LANG" without providing the assigned value of the variable. The percent (%) character is used for substitutions in Windows command line scripts.

The echo %LANG% command displays "%LANG%" without providing the assigned value of the variable. The percent (%) character is used for substitutions in Windows command-line scripts.

**Question 4 of 10**

**BackNext**

Listen to exam instructions

You need to set the COMP variable to the value 1745.

Which of the following commands sets the variable, so it is inherited by subsequent child shells?

answer

Correct Answer:

export COMP=1745

COMP=1745

set COMP to 1745

set COMP=1745

**Explanation**

The export COMP=1745 command creates the COMP variable, assigns it the value of 1745, and sets the export attribute to make it an environment variable that will be inherited by subsequent child shells.

The set COMP to 745 command sets the value of the $1 positional variable to "COMP," the value of the $2 positional variable to "to," and the value of the $3 positional variable to "1745."

The set COMP=1745 command sets the value of the $1 positional variable to "COMP=1745."

The COMP=1745 command simply creates a shell variable named COMP and sets its value to 1745. This shell variable is not inherited by subsequent child shells.

**Question 5 of 10**

**BackNext**

Listen to exam instructions

You recently used the HOST=FS4 command.

Which of the following commands should you use to change the HOST variable to an environment variable that will be inherited by subsequent child shells and processes?

answer

Correct Answer:

export HOST

env HOST

set HOST

unset HOST

**Explanation**

The export HOST command changes the HOST variable to an environment variable.

The set command with no options or arguments displays the names and values of all variables and functions. Otherwise, the set command is used to manipulate shell options and positional parameters that are mostly used in shell scripting. The set HOST command essentially sets the first positional parameter $1 to HOST, which can be verified by subsequently running the echo $1 command.

The env HOST command attempts to run the HOST command. Normally, environment variables (not present in this command) are temporarily set before the command is run. Consequently, this command will likely give the message "No such file or directory" since HOST is not a command and there is no executable file named HOST.

The unset HOST command attempts to remove the HOST variable. If the variable exists, it is removed. If not, no warning message is displayed.

**Question 6 of 10**

**BackNext**

Listen to exam instructions

Tim, a technician, creates a local variable named *val* and sets it to 5000 at the bash prompt. Tim wants to use the variable in a script. But when the script is executed, the value of *val* is not set to 5000.

Which of the following commands would allow Tim to set *val* as an environment variable that would be available to the script?

answer

echo $val=5000

Correct Answer:

export val=5000

exec val

declare val

**Explanation**

export val=5000 will create an environment variable named *val*and set the value to 5000. The variable *val*will be available to any scripts or applications run in that terminal session.

declare val is used to declare a variable. In this case, no value would be assigned. The scope of the variable is not global.

echo $val=5000 will display the value of $val, which is empty, and then displays =5000 as the output.

exec val will display the error "bash: exec: val: not found."

**Question 7 of 10**

**BackNext**

Listen to exam instructions

Which of the following commands configures the shell to retain 300 recently used commands in the ~/.bash\_history file for multiple shell sessions?

answer

BASH=300

HISTSIZE=300

HISTFILE=300

Correct Answer:

HISTFILESIZE=300

**Explanation**

The HISTFILESIZE=300 command sets the number of past commands remembered between multiple sessions and stored in the ~/.bash\_history file.

The HISTSIZE environment variable specifies the number of past commands remembered for the current shell session.

The HISTFILE shell variable specifies the filename where past commands are stored, which is ~/.bash\_history by default.

The BASH environment variable specifies the location of the bash executable file, which is normally /user/bin/bash.

**Question 8 of 10**

**BackNext**

Listen to exam instructions

You want the directory /sbin/special to be include in the PATH environment variable. You also want to keep all the current directory entries currently in the PATH variable.

Which of the following commands would you use?

answer

PATH=$PATH:$/sbin/special

PATH=PATH&/sbin/special

PATH=/sbin/special

Correct Answer:

PATH=$PATH:/sbin/special

**Explanation**

A colon (:) separates entries in the PATH statement. The PATH=$PATH:/sbin/special command appends a colon and the /sbin/special directory to the existing PATH ($PATH).

The PATH=$PATH:$/sbin/special command corrupts the PATH variable by appending a dollar sign ($) and the /sbin/special directory.

The PATH=/sbin/special command replaces the current PATH with only the /sbin/special directory.

The ampersand (&) in the PATH=PATH&/sbin/special command causes the PATH=PATH command to be run in a child process and then the /sbin/special command to run. While this command is interpreted by the bash shell, it does not modify the PATH variable.

**Question 9 of 10**

**BackNext**

Listen to exam instructions

Which of the following statements BEST describes the PATH environment variable?

answer

It contains the path of the current working directory.

It specifies the characters the shell uses to indicate normal user ($), root user (#), and similar items.

It specifies the filename where past commands are stored.

Correct Answer:

It contains the directory prefixes used to search for programs and files.

**Explanation**

The PATH environment variable contains the directory prefixes used to search for programs and files. Use a colon (:) to separate multiple directories in the PATH variable.

The HISTFILE shell variable specifies the filename where past commands are stored.

The PS1 shell variable specifies the characters in the shell prompt that indicate either a normal user ($) or root user (#).

The PWD environment variable contains the path of the current (or present) working directory.

**Question 10 of 10**

**BackNext**

Listen to exam instructions

Which command displays all the environment variables defined in the shell?

correct answer selected

**Explanation**

Both the printenv and env commands display the values for environment variables defined in the shell.

# **Section 2.6Shell Configuration Files**

As you study this section, answer the following questions:

* What is a shell configuration file?
* In what order do configuration scripts execute?
* What is the difference between a login shell and a non-login shell?
* What does the **su -l** command do?

In this section, you will learn to:

* View shell configuration files

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| Login shell | A shell that runs after the user successfully logs in using a user ID and password. |
| Non-login shell | A shell that is opened without requiring the user to authenticate with a user ID and password. |
| Shell configuration file | A special file containing a script that is run during a shell startup. |

This section helps you prepare for the following certification exam objectives:

| **Exam** | **Objective** |
| --- | --- |
| TestOut Linux Pro | 1.2 Configure and use Linux shell environments   * Manage environment variables |
| CompTIA Linux+ XK0-005 | 2.2 Given a scenario, implement identity management   * Account creation and deletion   + Configuration files     - .bashrc     - .bash\_profile     - /etc/profile |

# **2.6.3Shell Configuration Facts**

Shell configuration files are scripts that execute when a shell starts.

This lesson covers the following topics:

* Shell types
* Shell configuration files
* Login shell configuration script precedence

**Shell Types**

There are two shell types that determine which shell configuration files are executed.

* Login shells run after the user successfully logs in using a user ID and password. For example:
  + When the user logs in using the GUI.

A login shell runs in the background when a user first logs in using the GUI, so the effects from any configuration file are not easily seen.

* + When the Linux system does not have a GUI and the user logs in using the text-based interface.
  + When the Linux system has a GUI, but a user logs into the text-based console using the Ctrl + Alt+ F2 key combination.
  + When a user runs the **su -l <user ID>** (or just the **su -** , which implies the -l) command and enters the correct password.
  + When a user runs the **sudo -i <user ID>** command and enters the correct password.
  + When a user logs in from a remote system using the SSH protocol.
* Non-login shells run when a user opens a shell after first authenticating with a user ID and password. For example:
  + When a user opens a shell session (terminal application) from within the GUI.
  + When a user runs the **su <user ID>** command without the **-l** or **-** options.
  + When a user runs the **sudo <user ID>** command without the **-i** option.
  + When a user executes a script from within a shell.
  + When a user opens a new shell from a shell prompt.

One way to determine the shell type is to run the **echo $0** command. If this command returns a dash followed by the name of the shell (for example, **-bash** ), then it is a login shell. If the command returns the name of the shell without the dash or returns the name of the command that created the shell (for example, **bash** or **su** ), then it is a non-login shell.

**Shell Configuration Files**

Shell configuration files are scripts that are run during shell startup. The following configuration files are run when a shell starts.

| **Configuration File** | **Run by** | **Shell Type** |
| --- | --- | --- |
| **/etc/bashrc** or **/etc/bash.bashrc** | All users | Non-login |
| **~/.bashrc** | The specified user only | Non-login  On most Linux distributions, this file is also called by login shell configuration files. |
| **/etc/profile** | All users | Login |
| **\*. sh files in the /etc/profile.d/ directory** | All users | Login  On most Linux distributions, this file is also called by non-login shell configuration files. |
| **~/.bash\_profile** | The specified user only | Login |
| **~/.bash\_login** | The specified user only | Login |
| **~/.profile** | The specified user only | Login |
| **~/.bash\_logout** | The specified user only | Login  This file is only run as the user logs out (runs the **exit** command). |

**Login Shell Configuration Script Precedence**

Login shells execute the configuration scripts they use in the following order:

1. **/etc/profile**
2. **~/.bash\_profile** (if this file is found, the shell does not look for additional configuration script files).
3. **~/.bash\_login** (if this file is found, the shell does not look for additional configuration script files).
4. **~/.profile** (this file executes only in the absence of the two preceding configuration script files).

# **2.6.4Practice Questions**

**Question 1 of 10**

**Next**

Listen to exam instructions

A user starts a bash session and the script in /etc/profile. If all of the following files exist, which will be run next?

answer

~/.profile

~/.bash\_logout

~/.bash\_login

Correct Answer:

~/.bash\_profile

**Explanation**

If the ~/.bash\_profile file exists, it is run after the /etc/profile file. No additional configuration script files will be run.

If the ~/.bash\_login file exists, it will only be run in the absence of the ~/.bash\_profile file. No additional configuration script files will be run.

If the ~/.profile file exists, it will only be run in the absence of the ~/.bash\_profile file; and the ~/.bash\_login file.

The ~/.bash\_logout file is only run when the user logs out.

**Question 2 of 10**

**BackNext**

Listen to exam instructions

All users at your site are using the bash shell. You want to set a variable that will apply to every user and always have the same value.

Which of the following shell configuration files should you place this variable in?

answer

Correct Answer:

/etc/profile

.bashrc

.bash\_profile

.bash\_login

**Explanation**

The /etc/profile file runs for all users. Variables in this file are set for every user.

The .bash\_profile file is located in a user's home directory and runs only for that specified user.

The .login file is located in a user's home directory and runs only for that specified user.

The .bashrc file is located in a user's home directory and runs only for that specified user.

**Question 3 of 10**

**BackNext**

Listen to exam instructions

From the drop-down list below, select the file that is called by the non-login shell configuration file.

~/.bashrc

correct answer selected

**Explanation**

On most Linux distributions, the ~/.bashrc file is called by a non-login shell configuration file.

On most Linux distributions, the \*.sh files in the /etc/profile.d/ directory are called by login shell configuration files.

~/.bash\_login and ~/.bash\_logout files are for a specified user only, and logout files are only run as the user logs out (runs the exit command).

**Question 4 of 10**

**BackNext**

Listen to exam instructions

Which of the following are examples of a user running a login shell? (Select two.)

answer

When a user opens a shell session (terminal application) from within the GUI.

Correct Answer:

When a user runs the sudo -i <user ID> command and enters the correct password.

When a user runs the sudo <user ID> command without the -i option.

When a user opens a new shell from a shell prompt.

Correct Answer:

When the Linux system does not have a GUI and the user logs in using the text-based interface.

**Explanation**

The following are examples of a user running a login shell:

* When the Linux system does not have a GUI and the user logs in using the text-based interface.
* When a user runs the sudo -i <user ID> command and enters the correct password.

The following are examples of a user running a non-login shell:

* When a user opens a shell session (terminal application) from within the GUI.
* When a user runs the sudo <user ID> command without the -i option.
* When a user opens a new shell from a shell prompt.

## Question 5 of 10

**BackNext**

Listen to exam instructions

What type of shell runs after the user successfully logs in using an ID and password? (Type the answer in the blank field.)

correct answer selected

### Explanation

A login shell runs after the user successfully logs in using a user ID and password.

**Question 6 of 10**

**BackNext**

Listen to exam instructions

Select the file type from the drop-down list below that is also called by non-login shell configuration files.

\*.sh files in /etc/profile.d

correct answer selected

**Explanation**

On most Linux distributions, the \*.sh files in the /etc/profile.d/ directory are also called by non-login shell configuration files.

On most Linux distributions, the ~/.bashrc file is also called by non-login shell configuration files.

The ~/.bash\_login and ~/.bash\_logout files are for a specified user only, and the logout file is only run as the user logs out (runs the exit command).

**Question 7 of 10**

**BackNext**

Listen to exam instructions

Which of the following are examples of a user running a non-login shell? (Select two.)

answer

When the Linux system does not have a GUI and the user logs in using the text-based interface.

Correct Answer:

When a user opens a shell session (terminal application) from within the GUI.

When a user logs in from a remote system using the SSH protocol.

Correct Answer:

When a user runs the sudo <user ID> command without the -i option.

When a user runs the sudo -i <user ID> command and enters the correct password.

**Explanation**

The following are examples of a user running a non-login shell:

* When a user opens a shell session (terminal application) from within the GUI.
* When a user runs the sudo <user ID> command without the -i option.

The following are examples of a user running a login shell:

* When the Linux system does not have a GUI and the user logs in using the text-based interface.
* When a user runs the sudo -i <user ID> command and enters the correct password.
* When a user logs in from a remote system using the SSH protocol.

**Question 8 of 10**

**BackNext**

Listen to exam instructions

What type of shell runs when a user opens a shell after first authenticating with a user ID and password? (Type the answer in the blank field.)

correct answer selected

**Explanation**

Non-login shells run when a user opens a shell after first authenticating with a user ID and password.

**Question 9 of 10**

**BackNext**

Listen to exam instructions

Which of the following files could you use to create aliases that are applied when a specific user starts a bash session? (Select two.)

answer

~/.bash\_logout

/etc/profile

/etc/bashrc

Correct Answer:

~/.profile

Correct Answer:

~/.bashrc

**Explanation**

The ~/.profile file runs for individual users.

The ~/.bashrc file runs for individual users.

The /etc/profile runs for all users.

The ~/.bash\_logout file runs for an individual user but only runs when the user logs out.

The /etc/bashrc runs for all users.

**Question 10 of 10**

**BackNext**

Listen to exam instructions

Each time you present credentials to log in, you want a particular entry to be written to a log file. This action will only apply to yourself. You are using the bash shell.

Which of the following configuration files is the BEST to modify to enable this action?

answer

.bashrc

/etc/profile

Correct Answer:

.profile

.sh

**Explanation**

The .profile file exists within the user's home directory and is executed upon each login. Modifying this script will enable this action for only yourself.

The .sh file is a custom hidden script file but would not be executed during user login.

While typically the .bashrc file is run by a login script, Linux distributions don't guarantee to run this file during user logins.

The /etc/profile file will run for all users instead of one specific user.

# **Section 2.7Redirection, Piping and Command Substitution**

As you study this section, answer the following questions:

* What are the three default file descriptors that Linux uses to classify information for a command?
* What is the difference between redirection and piping?
* When might you choose to redirect the input of a command?
* How can you overcome the 128 KB shell command size restriction?

In this section, you will learn to:

* Use redirection
* Use piping

Key terms for this section include the following:

| **Term** | **Definition** |
| --- | --- |
| Standard stream | Preconnected input and output communication channels available to Linux shells and processes. |
| stdin | A standard stream that provides data that is typically streamed from the keyboard. |
| stdout | A standard stream that accepts normal output information to be streamed to the console screen or shell window. |
| stderr | A standard stream that accepts normal error information to be streamed to the console screen or shell window. |
| Redirection | The process of modifying a shell command to divert the standard input, output, and error streams to locations other than the default. |
| Piping | The process of redirecting the output from one command to be the input of another command. |
| Here documents | A block of text that is redirected as input to a command. |
| Command substitution | A feature of the bash shell that substitutes the output of one shell command as the arguments for another shell command. |

This section helps you prepare for the following certification exam objectives:

| **Exam** | **Objective** |
| --- | --- |
| CompTIA Linux+ XK0-005 | 1.1 Summarize Linux fundamentals   * Device types in /dev   + Special character devices     - /dev/null     - /dev/zero     - /dev/urandom   3.1 Given a scenario, create simple shell scripts to automate common tasks.   * Shell script elements   + Standard stream redirection     - |     - ||     - >     - >>     - <     - <<     - &     - &&     - Redirecting       * stderr       * stdout   + Here documents   + Common script utilities     - xargs |

# **2.7.5Redirection and Piping Facts**

Redirection and piping are two Linux shell features that allow the input and output of a command to be a file or another command.

This lesson covers the following topics:

* Standard streams and bash shell file descriptors
* Redirection
* Piping
* The tee command
* Here documents
* Device files often used with redirection and piping

**Standard Streams and Bash Shell File Descriptors**

The bash shell maintains three standard data streams, which are pre-connected input and output communication channels. Unless configured otherwise, bash commands use the following standard streams:

* Standard input (stdin) is data that is typically streamed from the keyboard. If a bash command accepts input, by default, it is gathered from stdin.
* Standard output (stdout) is data that is typically streamed to the console screen. By default, bash commands send their output to stdout.
* Standard error (stderr) is data that is also typically streamed to the console screen. If a command needs to output an error messages or give diagnostics, by default, it sends this output to stderr.

The bash shell assigns a file descriptor to each of the standard streams. A file descriptor is a handle or number that identifies an open file or other data source and how that resource can be accessed. The following table summarizes these ideas.

| **Standard Stream** | **File Descriptor** | **Associated Device** |
| --- | --- | --- |
| stdin | 0 | Keyboard |
| stdout | 1 | Console screen or graphical shell window |
| stderr | 2 | Console screen or graphical shell window |

**Redirection**

Linux commands can be modified to divert the standard input, output, and error streams to locations other than the default. This process is called redirection and is implemented using the following command operators:

| **Command Operator** | **Description** |
| --- | --- |
| **>** and **1>** | Redirects command output that is normally sent to stdout to the file name that follows the operator. The 1 is implied so that **>** and **1>** are functionally identical.   * If the file that follows the operator exists, it is overwritten. * If the file doesn't exist, it is created. * If there is no output generated by the command, the file will be empty. |
| **2>** | Redirects command output that is normally sent to stderr to the file name that follows the operator. The **2** is mandatory.   * If the file that follows the operator exists, it is overwritten. * If the file doesn't exist, it is created. * If there is no error generated by the command, the file will be empty. |
| **>>** , **1>>** and **2>>** | The **>>** operator functions in the same way as the **>** operator except that any output/errors are appended to the file that follows the operator.   * The **>>** and **1>>** operators appends the output sent to stdout. * The **2>>** operator appends the output sent to stderr. * If the file that follows the operator exists, it is appended with the output/error. * If the file doesn't exist, it is created. * The file will be not be appended if there is no output/errors generated by the command. |
| **&>** | Redirects both command output that is normally sent to stdout and command errors that are normally sent to stderr to the file name that follows the operator.   * As part of a command, **&> myfile.txt** is equivalent to **> myfile.txt 2> &1** or **> myfile.txt 2> myfile.txt** . * File creation follows the rules for the **>** operator. |
| **&>>** | The **&>>** operator functions in the same way as the **&>** operator except that both output and errors are appended to the file that follows the operator.   * As part of a command, **&>> myfile.txt** is equivalent to **>> myfile.txt 2>> &1** (where **&** indicates that what follows is a file descriptor and not a filename) or **>> myfile.txt 2>> myfile.txt** . * File creation follows the rules for the **>** operator. |
| **<** and **0<** | Redirects command input that is normally read from stdin so that it is read from the file name that follows the operator. The 0 is implied so that **<** and **0<** are functionally identical.   * If the file that follows the operator exists, it is used as input. * If the file doesn't exist, an error is shown. * If there is no input needed by the command, the file is ignored. |

The following examples demonstrate redirection concepts.

| **Example** | **Result** |
| --- | --- |
| **ls /usr > /tmp/deleteme** | Writes the list of files in the /usr directory to a file named /tmp/deleteme. |
| **ls /nonesuch > /tmp/deleteme** | Creates an empty /tmp/deleteme file (or overwrites it as an empty file if it exists) and displays the error message */nonesuch not found* (because the /nonesuch directory does not exist). |
| **ls /nonesuch 2> /tmp/deleteme** | Writes the error message */nonesuch not found* (because the /nonesuch directory does not exist) to a file named /tmp/deleteme. The /tmp/deleteme file will be overwritten if it already exists. |
| **ls /bin /nonesuch > /tmp/deleteme** | Writes a listing of the files and directories within the /bin directory to the /tmp/deleteme file and displays the error message */nonesuch not found* to the console screen or shell window. The /tmp/deleteme file will be overwritten if it already exists. |
| **ls /bin /nonesuch > /tmp/deleteme 2>&1** | Writes a listing of the files and directories within the /bin directory to the /tmp/deleteme file and also writes the error message */nonesuch not found* to the /tmp/deleteme file. Both the list of files and directories within the /bin directory and the error message are written to the same file. |
| **ls /bin >> /tmp/deleteme** | Appends the list of files from the /usr directory to the /tmp/deleteme file. The /tmp/deleteme file will be created if it does not exist. |
| **sort < unordered.txt > ordered.txt** | Uses the contents of the unordered.txt as input to the **sort** command, and then writes the sorted contents to the file named ordered.txt. If the ordered.txt file already exists, it will be overwritten. |

**Piping**

Piping redirects the output from one command to be the input of another command.

* A Linux pipe is represented by a vertical bar ( **|** ).
* The pipe functionality is similar to that of using stdout redirection to write the output of one command to an intermediate file, that is then used as input to a second command, using stdin redirection.
* A plumbing pipe, where water enters from one end and exits the other, can be used as a mnemonic to help explain how the pipe (|) operator works.

The following examples demonstrate piping concepts.

| **Example** | **Result** |
| --- | --- |
| **ls /bin | sort | mail jdoe** | Takes the list of the contents of /bin directory, sorts it, and then mails the sorted contents to jdoe. |
| **cat /usr/wordlist1 /usr/wordlist2 | sort > sortedwordlist** | Pipes the combined contents of the wordlist1 and wordlist2 files to sort and then redirect the sorted output to the sortedwordlist file. |

**The tee Command**

There may be times when you want to view the output of a command as it is normally sent to the console screen (stdout), but you also want the same output to be saved in a file. This can be done using the tee command.

* The output from a command is piped to the tee command.
* The file used to store the output is added as a tee command argument.
* A plumbing tee, where water flow is divided from one pipe to two separate pipes, can help you understand how the tee command works.

The following examples demonstrate tee command concepts.

| **Example** | **Result** |
| --- | --- |
| **ls /bin | tee binfiles.txt** | Displays the files and directories contained in the /bin directory on the console screen (or shell window) and writes the same information to the binfiles.txt file. |
| **ls -1 \*.txt | wc -l | tee count.txt** | Pipes a one-column list of files that end with *.txt* in the current directory to the wc (word count) command and then take that output, which gives the number of files in the list, and pipes that to the tee command that displays this number on the console screen (or shell window) and writes the same number to the count.txt file. |

**Here Documents**

A here document is a block of text that is redirected as input to a command. Here documents are often used in shell scripts.

* A command is followed by the **<<** operator, which is then followed by a marker, which is traditionally an uppercase word.

The term *here document* may have origins in the practice of using the word *HERE* as the marker.

* Lines of text are included in the block.
* The end of the block of text is indicated by the same marker that follows the **<<** operator.
* The shell passes the block of text to the command as input.

The following examples demonstrate here document concepts.

| **Example** | **Explanation** |
| --- | --- |
| **cat << HERE > Today, we hope you are > learning a great deal > about Linux redirection > and piping from TestOut. >HERE** | The following lines are displayed on the console screen or shell window:  Today, we hope you are learning a great deal about Linux redirection and piping from TestOut.  This may be useful when creating shell scripts that present explanations and documentation to users. |
| **lftp machine -uUser,Passwd <<END cd your\_dir get your\_file bye END** | Multiple commands are entered during an ftp session. These commands are managed using the lftp program. This may be helpful when automating a long list of ftp commands. |

**Device Files Often Used with Redirection and Piping**

Device files are file-like access points to hardware devices. There are two device files that are often used with redirection and piping: /dev/tty and /dev/null

| **Device File** | **Description** |
| --- | --- |
| **/dev/tty** | The first terminals were Teletype (abbreviated as *tty* ), which can be compared to a remote controlled typewriter. The /dev/tty device file is associated with the computer's controlling terminal or the shell's window.   * Data can be both written to and read from this file. * Text written to this file is displayed on the console monitor's screen or shell window. * Text read from this file originates from the console's keyboard. * The **/dev/tty** device file is similar to a combination of stdin and stdout. Both stdin and stdout are accessed as data streams, whereas **/dev/tty** is accessed like a file. |
| **/dev/null** | The /dev/null device file is associate with a null device. A null device is commonly used for disposing unwanted output streams.   * While a command can read from **/dev/null** , commands typically write unwanted output or unwanted error messages to **/dev/null** . * A slang word for the **/dev/nul** device file is bit bucket. |
| **/dev/zero** | Similar to /dev/null, /dev/zero discards any input. It also returns a "0" for however many times it is accessed. It is most commonly used for:   * Initializing a new block device * Overwriting existing data |
| **/dev/urandom** | Returns a pseudo-random number. Frequently used when performing cryptographic (encryption) tasks. |

**/dev/tty**

| **Example** | **Explanation** |
| --- | --- |
| **echo "test" > /dev/tty** | Writes the word *test* to the console or to a shell windows. This is redundant, since the echo command by itself performs the same action. |
| **sort < /dev/tty > sortkeyboard.txt** | The text entered using the keyboard is sorted and written to the sortkeyboard.txt file.  The **< /dev/tty** operation will continue to accept keyboard input until the user enters an end-of-file (EOF) sequence using the Ctrl+D key combination. |
| **rm deleteme.txt 2> /dev/null** | Deletes the **deleteme.txt** file if it exists. If it doesn't exist, don't display an error message.  This logic is often used in shell scripts to suppress error messages that are not important to the script's overall purpose. |

# **2.7.7Command Substitution Facts**

Command substitution is a feature of the Bash shell that substitutes the output of one shell command as the arguments for another shell command.

This lesson covers the following topics:

* Implement command substitution
* The xargs command

**Implement Command Substitution**

**$(<command>)**

1. The shell creates a child process that runs the specified command.
2. The stdout from the first command is redirected back to the shell.
3. The shell parses the output from the first command into words separated by white space.

If the command substitution operator **$()** appears within single quotes (' '), word parsing is not performed on the output.

1. The shell creates a new command by substituting the parsed output from the first command in place of the **$(< *command* >)** operator.
2. The shell creates another child process that runs the second command.

The backtick ( **`** ) operator also performs command substitution. Enclose a command within backticks ( **`** ) to achieve command substitution in the same way as the **$()** operator. On newer keyboards, the backtick ( **`** ) is on the same key as the tilde ( **~** ).

The examples below demonstrate these command substitution concepts.

| **Example** | **Explanation** |
| --- | --- |
| **printf "The date and time is: $(date)\n"** | Command substitution occurs when the shell encounters **$(date)** . The process is as follows:   1. The shell creates a child process and runs the **$(date)** command. 2. The output from the child process is redirected back to the shell, but is not parsed. 3. The **$(date)** operator in the original command is replaced with the output from the child process. 4. Another new process is created that runs the **printf** command with the replaced text.   The **printf** command replaces the **\n** in double quotes with a newline character. |
| **echo -e "List of logged on users and what they are doing:\n $(w)"** | Command substitution occurs when the shell encounters **$(w)** . The process is as follows:   1. The shell creates a child process and runs the **w** command.   The **w** command is short for "who" and returns a summary of logged-on users.   1. The output from the child process is redirected back to the shell, but is not parsed. 2. The **$(w)** operator in the original command is replaced with the output from the child process. 3. Another new process is created that runs the **echo** command with the replaced text.   The **-e** option with the **echo** command causes the **\n** in double quotes to be replaced with a newline character. |

**The xargs Command**

**xargs**

**xargs**

| **Example** | **Explanation** |
| --- | --- |
| **find /home -name \*~ | xargs rm** | This command deletes all files in all subdirectories of the /home directory that end with the tilde (~) character.   1. The **find /home -name \*~** command returns a list of all the files in all the subdirectories under the /home directory that end with the tilde (~) character. 2. The file list is piped as a stream to the **xargs** command. 3. The **xargs** command collects the first 128-KB chunk from the stream. 4. The **rm** command is run using the 128-KB chunk as an argument. 5. The **xargs** command continues to collect 128-KB chunks from the stream and continues to run the **rm** command using the chunks until the end of the stream. |
| **ls -S \*.txt | xargs wc** | When there are large number of \*.txt files in a directory, this command displays a list of all the files along with the number of lines/words/characters in each. The list is sorted by size, with the largest files shown first.   1. The **ls -S \*.txt** command returns the list of files sorted from largest to smallest. 2. The file list is piped as a stream to the **xargs** command. 3. The **xargs** command collects the first 128-KB chunk from the stream. 4. The **wc** command is run using the 128-KB chunk as an argument. 5. The **xargs** command continues to collect 128-KB chunks from the stream and continues to run the **wc** command using the chunks until the end of the stream. |

# **2.7.8Practice Questions**