PassQueen
a.k.a dumpsqueen.com

Arista Linux Essentials Exam

Arista ACE-P-ALE1.04

Version Demo

Total Demo Questions: 12

Total Premium Questions: 120

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Linux Fundamentals	86
Topic 2, Linux System Administration	23
Topic 3, Linux Networking	2
Topic 4, Linux Security	9
Total	120

QUESTION NO: 1

Which two commands can be used to locate the executable that would run when a command name is entered in Bash? (choose two)

A. command -v

B. type

C. locate

D. find

ANSWER: A B

Explanation:

The `command -v` and `type` built-ins can identify how Bash resolves a command name. They can report whether the name refers to an alias, shell function, builtin, or executable found through `PATH`. For example, `command -v python3` commonly returns the pathname that Bash would execute, while `type cd` identifies `cd` as a shell builtin. See the GNU Bash builtins documentation.

QUESTION NO: 2

Which two commands can be used to change the ownership of a file? (choose two)

A. chown

B. chgrp

C. chmod

D. umask

ANSWER: A B

Explanation:

`chown` changes the owning user of a file and can also set its group. `chgrp` changes only the owning group. Both commands modify ownership metadata and commonly require appropriate privileges when changing ownership to another user or group. In contrast, `chmod` changes permission bits rather than ownership. See the GNU Coreutils chown documentation and the GNU Coreutils chgrp documentation.

QUESTION NO: 3

Which two statements are true about the `PATH` environment variable? (choose two)

A. It is normally a colon-separated list of directories

B. It is searched for commands entered without a pathname

C. It defines the current working directory

D. It stores the permissions applied to new files

ANSWER: A B

Explanation:

The `PATH` variable is a colon-separated list of directories that the shell searches when a command is entered without a pathname. Its value can be displayed with `echo "$PATH"`, and a directory can be added by assigning a revised value, such as `PATH="$PATH:/opt/tools"`. The shell does not search `PATH` when an explicit pathname such as `./script.sh` or `/usr/bin/ls` is supplied. See the GNU Bash documentation for shell variables and the GNU Bash command search documentation.

QUESTION NO: 4

What free optional software package allows the Linux kernel to support proprietary filesystems such as ExFAT and NTFS?

A. VFS

B. inodes

C. FUSE

D. UFS

ANSWER: C

Explanation:

FUSE is correct because it provides the Filesystem in Userspace framework, which enables filesystem implementations to run as ordinary user-space processes while presenting mounted filesystems through the standard Linux filesystem interface. The kernel includes a FUSE module that communicates with a user-space filesystem daemon, allowing applications to access files using normal paths, permissions, and mount points without requiring every filesystem driver to be implemented entirely in the kernel.

This architecture has historically been especially useful for optional filesystem support, including user-space implementations for formats such as NTFS and exFAT. For example, NTFS-3G is a well-known NTFS driver built on FUSE. The approach lets Linux systems add filesystem capabilities through installable packages and user-space tools, rather than modifying or rebuilding the running kernel. In modern Linux releases, some filesystems may also have native in-kernel drivers, but FUSE remains the general framework described by the question and is widely used for third-party, networked, encrypted, virtual, and compatibility filesystem mounts.

The Linux kernel documentation describes the FUSE kernel/userspace interface at Linux Kernel FUSE documentation. Additional project documentation is available from libfuse.

QUESTION NO: 5

Which statements about exported shell variables are true? (choose two)

A. An exported variable is inherited by child processes

B. The export built-in marks a variable for the environment

C. All shell variables are automatically inherited by child processes

D. A child process can permanently change its parent shell's variables

ANSWER: A B

Explanation:

An exported shell variable is placed in the environment of commands started by the shell. The `export` built-in marks a variable for this inheritance, allowing child processes such as scripts and utilities to receive its value.

Ordinary shell variables are not automatically inherited by child processes. Also, a child process cannot directly modify the parent shell's environment, so changing an exported variable in a script does not alter the invoking shell after the script exits. See the GNU Bash export built-in documentation and the GNU Bash environment documentation.

QUESTION NO: 6

In regular expressions, to what does greedy vs. lazy refer?

A. How many lines in a file the pattern will match

B. How many times the pattern will match on a line

C. How strict the pattern is required to be

D. Whether the pattern quantifiers will stop at whitespace

E. Whether quantifiers prefer to match as much text as possible or as little text as possible

ANSWER: E

Explanation:

Greedy versus lazy refers to how a quantified part of a regular expression chooses the amount of text it consumes when more than one match length can satisfy the overall expression. A greedy quantifier, such as `*`, `+`, `?`, or `{m,n}`, initially consumes as much input as possible while still allowing the expression to match. A lazy (also called reluctant or non-greedy) quantifier uses the corresponding quantifier followed by `?`, such as `*?` or `+?`, and initially consumes the smallest amount of input that permits a match. For example, when matching text enclosed in delimiters, a greedy repetition can extend through later delimiters, whereas a lazy repetition stops at the earliest delimiter that allows the rest of the expression to succeed. This behavior is about the match length selected by quantifiers, not about lines, whitespace, or the number of independent matches attempted. Regex engines may backtrack if needed to satisfy subsequent portions of the pattern, but greediness establishes the initial preference for the longest viable match and laziness establishes the initial preference for the shortest viable match. See the Python regular-expression documentation on quantifiers at https://docs.python.org/3/library/re.html and the detailed quantifier reference at https://www.regular-expressions.info/repeat.html.

QUESTION NO: 7

To negate a set in a regular expression pattern, which character is used?

A. !

B. ^

C. ?

D. -

ANSWER: B

Explanation:

The correct character is `^`, when it appears immediately after the opening square bracket of a character class. A character class normally matches one character from the listed set, such as `[abc]`. Placing the caret first inside the brackets negates that set: `[^abc]` matches one character that is not `a`, `b`, or `c`. This syntax is widely used in regular-expression implementations available in Linux tools and programming languages, including POSIX-style regular expressions. Its position is essential: within a bracket expression, the caret has this complement meaning only when it is placed directly after the opening bracket. For example, `[^0-9]` matches a single non-digit character, which is useful for matching separators, validating input, or excluding unwanted characters. The POSIX regular-expression specification describes bracket expressions and specifies that a circumflex at the beginning of the list complements the matching list. See the POSIX Regular Expressions specification and the GNU grep documentation on bracket expressions.

QUESTION NO: 8

Which two characters are required at the top of a script to inform the kernel on which interpreter to use? (choose two)

A. !!

B. !

C. #

D. #!

ANSWER: B C

Explanation:

The required characters are `#` followed immediately by `!`. Together, they form the *shebang* prefix, `#!`, which must be the first two bytes of a script file. When a script is executed, the kernel recognizes this prefix and reads the interpreter path that follows it, such as `#!/bin/bash` or `#!/usr/bin/env python3`. The `#` begins the special interpreter directive and the `!` distinguishes it from an ordinary shell comment. The interpreter pathname and any permitted optional argument are then supplied after these two characters. For example, `#!/bin/sh` tells the kernel to invoke the POSIX shell to process the script. This mechanism is implemented by the Linux kernel's script binary-format handler, commonly called `binfmt_script`. The two individual characters are therefore `#` and `!`, conventionally written together as `#!`. See the Linux manual page for executable-script handling in execve(2) and the Linux kernel source for the binfmt_script handler.

QUESTION NO: 9

How much swap space is available on Arista switches?

A. 4GB

B. 0 bytes

C. 1GB

D. 640kB

ANSWER: B

Explanation:

0 bytes is correct. Arista EOS switches are designed to operate without configured swap space. The EOS control plane runs on a Linux-based environment, but its normal storage and memory design does not provide a disk-backed swap device or swap file for general use. Therefore, utilities that report system memory and swap status, such as `free` or inspection of `/proc/meminfo`, show the total available swap as zero.

This is an important operational distinction when troubleshooting memory use on a switch. Physical RAM is the memory resource available to EOS processes; swap is not an additional capacity pool that can be expected to absorb memory pressure. Operators should consequently monitor RAM consumption, investigate abnormal process memory growth, and follow the applicable EOS operational guidance rather than attempting to enable or depend on swap. The Linux kernel documents that the `SwapTotal` field represents total swap space, so a value of zero indicates that no swap capacity is configured. Arista's EOS documentation describes EOS as a Linux-based network operating system while retaining an appliance-oriented architecture for switch operation.

References: Arista EOS Overview; Linux proc_meminfo(5) documentation.

QUESTION NO: 10

Which two statements are true about SSH public-key authentication? (choose two)

A. The public key can be installed in authorized_keys on the remote host

B. The private key should remain on the client system

C. The private key must be copied to every remote host

D. Public-key authentication requires sharing one private key among users

ANSWER: A B

Explanation:

With SSH public-key authentication, the private key remains under the control of the client user, while the corresponding public key is installed on the remote system, commonly in `~/.ssh/authorized_keys`. The server uses the public key to verify a signature created by the client, so the private key does not need to be copied to the remote host. Protecting private-key files with restrictive permissions remains essential. See the OpenSSH authorized_keys documentation and the OpenSSH ssh-keygen manual.

QUESTION NO: 11

In file globbing, which symbol matches any single character?

A. *

B. ?

C. []

D. .

ANSWER: B

Explanation:

In shell file globbing, `?` matches exactly one arbitrary character in a filename, except that it does not match a leading period in a name unless the pattern itself explicitly begins with a period. For example, the pattern `file?.txt` can match `file1.txt` or `fileA.txt`, because precisely one character occupies the position represented by `?`. This is useful when a filename has a known structure but one character varies. Glob patterns are expanded by the shell before a command receives its arguments, allowing commands to operate on groups of matching pathnames. The POSIX shell specification defines `?` as matching any single character, while shell documentation describes this behavior as pathname expansion. See the POSIX Shell Command Language: Pattern Matching Notation and the GNU Bash manual on pattern matching.

QUESTION NO: 12

Why is it recommended for root's homedir to be in a separate location from other users' homedirs?

A. Because root doesn't want to hang out with pesky users

B. Because users are annoying

C. Because root's homedir permissions prevent user access

D. Because user homedirs are sometimes a separate filesystem that may not be mounted in single user mode

ANSWER: D

Explanation:

Because user home directories are sometimes on a separate filesystem that may not be mounted in single-user mode is correct. The root account must remain usable during maintenance, recovery, and troubleshooting, including situations where only the root filesystem has been mounted. If root's home directory were placed beneath `/home` and `/home` were a separate filesystem, root could lose access to its home directory, configuration files, administrative scripts, SSH material, and other essential resources until that filesystem is mounted. Keeping root's home at `/root`, on the root filesystem, avoids this

dependency and provides a reliable administrative environment when repairing storage, filesystem mounts, boot configuration, or user-home filesystems. The Filesystem Hierarchy Standard specifically defines `/root` as the home directory for the root user, reflecting this operational need. Rescue and single-user style environments are intended for system repair and may start with a minimal set of filesystems available, so root's home should not rely on a separately mounted user-data filesystem. See the Filesystem Hierarchy Standard entry for /root and the systemd rescue.target documentation.