Ultimate Linux Command Guide

Please note: these commands are meant to aid in studying for the CompTIA Linux+ learning objectives which encompasses a massive amount of material. However, this command guide is still helpful as a reference for anyone interested in Linux as I tried to include only the most popular and/or useful commands and flags that you will need day-to-day.

Shell Scripting

- `function_name() { #commands go here }` = This is the syntax for creating a function in Bash. The first part can be named whatever you want. Inside the {} brackets, you actually write your function parameters. In bash, functions do not declare explicit parameter names in their signature (like function_name(arg1, arg2) in Python). Instead, functions accept arguments using positional parameters identical to script arguments:
    - $1, $2, $3: Refers to the 1st, 2nd, and 3rd arguments passed specifically to the function when you call on it (ex. function_name “syslog” “var/log”)
    - $#: Total number of positional parameters passed to the function.
    - $* / $@: All parameters passed to the function.
- `()` = (Subshell): Commands placed inside parentheses `(foo)` run in a separate subshell environment. Environment changes inside a subshell NEVER affect the parent shell. FYI, the term foo is a placeholder name for a command. When Bash encounters commands inside standard parentheses (foo):
    - The parent shell spawns a separate child process (subshell).
    - The subshell inherits the environment (variables, working directory, open file descriptors) of the parent shell.
    - The commands inside (foo) execute sequentially inside that child process.
    - When execution finishes, the child process exits and terminates.
- `#` = (Comment/Shebang): Used for comments. When it appears as `#!` at the very first line, it is a Shebang, telling the OS which interpreter to use (e.g., `#!/bin/bash`).
- `$` = (Variable Expansion): Used to access the value stored in a variable (e.g., `$USER`).

- **${ }** = (Parameter Expansion): A more robust way to expand variables, allowing for string manipulation or protecting the variable name from surrounding text.
    - **${var}** = Parameter expansion is the process where Bash evaluates a variable expression and replaces it with its resolved string value before executing a command. While `$var` is simple, using the explicit braces `${var}` unlocks powerful, built-in string manipulation, default assignment, and evaluation techniques without invoking external utilities. The primary technical reason to use explicit curly braces `${var}` rather than simple `$var` is to separate the variable name from adjacent text or characters that would otherwise alter the variable name evaluation. With braces {}, bash evaluates the variable. (ex. echo ${PREFIX}_log Safely evaluates $PREFIX followed by the literal string "_log")
- **$( )** = (Command Substitution): Command substitution allows the output of a command (what is written to `stdout`) to replace the command itself inside a script or string. When Bash encounters $(command):
    - It executes command in a **subshell** environment.
    - It captures whatever the command prints to standard output (stdout).
    - It strips any trailing newline characters.
    - It places that output text directly into the command line where the $(...) expression was located.
    - **`foo`** = the older, POSIX-standard backtick syntax for command substitution, functionally equivalent to `$(foo)` for simple cases. It's considered legacy because it's harder to read, can't be nested cleanly (nesting requires escaping inner backticks with `\`), and its interaction with escape characters is less consistent than `$()`. `$()` is universally recommended for new scripts, but recognizing backticks in older/legacy scripts is still expected.
- **!** = Bang command. (Logical NOT): In scripts, it represents "NOT" (logical negation).
- **=** = variable assignment. Assigns a value to a variable. (ex. ROLE=”admin”)
- **@** = (All Arguments): Usually seen as $@, it represents all positional parameters passed to a script.
- **"** = (Double Quote): Weak quoting. Most special characters are treated as literal text, but variables ($) and command substitutions still expand.
- **'** = (Single Quote): Strong quoting. Everything inside is treated literally; no expansions occur.
- **** = (Escape Character): Tells Bash to treat the *very next* character as literal text, "escaping" its special meaning.

- `*` = (Asterisk): Matches zero or more characters in a filename.
- **%** = percent sign can be used for the following uses:
    - `%string` (The Prefix Search): Matches a job that starts with the specified string. (ex. fg %python)
    - Job Control with the job ID prefix (e.g., `%1`).
    - Arithmetic Modulo: In shell math, `%` calculates the remainder.
        - `echo $(( 10 % 3 ))` outputs `1`, because 3 goes into 10 three times with a remainder of 1.
    - Variable Parameter Expansion: This is a "pro-level" scripting trick. Using `${variable%pattern}` removes the shortest matching pattern from the end of a string.
        - Example: If `FILE="image.jpg"`, then `${FILE%.jpg}` results in just `image`.
    - The User Prompt: Conventionally, a `%` at the end of your command prompt (instead of `$`) often indicates you are using the Zsh shell as a standard user.
        - Vim (Current Filename): If you are inside the Vim editor, `%` represents the file you are currently editing.
        - Command: `:!gcc %` tells Vim to run the compiler on "this current file.
    - Systemd Specifiers: In unit files, `%u` stands for the user name and `%H` stands for the hostname.
    - Job IDs are identified by %<ID> (e.g., `%1`, `%2`).
- **%?** = search for a job containing a string (ex. fg %?backup)
- **?** = (Question Mark): Matches exactly one character in a filename.
- **$?** = exit status variable. Tells you if the last command succeeded. (ex. echo $?) This specific variable returns the exit status of the last executed command. A `0` means success, while anything else (usually `1-255`) indicates an error
- **+** and `-` = Used in arithmetic expressions or as indicators for command-line flags (e.g., `ls -l`). In comparisons, they appear in strings like `-eq` (equal) or `-gt` (greater than).
- **IFS** = Input field separator. A special shell environment variable that determines how Bash recognizes word boundaries when splitting strings into separate tokens during word expansion and read operations. By default, `IFS` is set to whitespace characters: **Space, Tab, and Newline** (`\t\n`). In system administration tasks, data is often separated by colons (e.g., `/etc/passwd`), commas (CSV), or newlines. You can override `IFS` to change how Bash splits

lines. Always modify `IFS` **locally** on a single command (as shown below) or save and restore the original `IFS` value. Changing `IFS` globally across a script can break standard command execution, loops, and parameter expansion!

```
# Example: Reading /etc/passwd entries separated by colons (:)
LINE="root:x:0:0:root:/root:/bin/bash"

# Temporarily change IFS to a colon for the read command
IFS=":" read -r username password uid gid gecos home shell <<< "$LINE"

echo "User: $username" # Output: User: root
echo "Shell: $shell"    # Output: Shell: /bin/bash
```

- **OFS** = Output field separator. While `IFS` is used by the Bash shell for reading/splitting inputs, **`OFS`** is a built-in variable in text-processing utilities like **`awk`** used for setting the character inserted between output fields when printing. It defines the string or character printed between fields when `awk` outputs data using commas.

```
# Input: Space-separated input string
# Goal: Reformat to CSV (comma-separated output)
echo "server01 192.168.1.50 active" | awk 'BEGIN {OFS=","} {print $1, $2, $3}'
# Output: server01,192.168.1.50,active
```

- **`if`** = controls execution flow in scripts based on whether commands or comparisons succeed or fail (True or False statements). **Exit Status `0`:** Represents **Success** (evaluated as true). **Exit Status `1` to `255`:** Represents **Failure** or errors (evaluated as false).

```
if command_or_condition; then
   # Runs ONLY if command returned exit code 0
elif alternative_condition; then
   # Runs ONLY if first condition failed and this one returns exit code 0
else
   # Runs if ALL conditions above fail (non-zero exit codes)
fi
```

    - Conditional logic is evaluated using one of three brackets:

        - [ ] = (POSIX / Standard Test). Calls the `test` binary command. Requires spaces around brackets (e.g., `[ "$a" = "$b" ]`). Requiring quotes around variables to avoid word-splitting errors on empty strings.
        - [[ ]] = (Extended Bash Test — Recommended). Supports **Pattern Matching** and **Regex Operations** (=~). Handles unquoted variables safely without crashing on empty strings. Supports logical operators like && (AND) and || (OR) inside the brackets.
        - (( )) = (Arithmetic Evaluation). Used strictly for **numerical calculations** and C-style integer comparisons (e.g., `(( count > 5 ))`).
    - **elif** = Runs ONLY if the first condition failed and this one returns exit code 0. Same syntax as **if**
        - **-f /path/to/file** True if the file exists AND is a regular file.
        - **-d /path/to/dir** True if the path exists AND is a directory.
        - **-e /path/to/item** True if the file/directory exists regardless of type.
        - **-r /path/to/file** True if the current user has read permissions.
        - **-w /path/to/file** True if the current user has write permissions.
        - **-x /path/to/file** True if the current user has execute permissions.
        - **-s /path/to/file** True if the file exists and is **not empty** (size > 0 bytes).
        - **-z "$var"** True if string is **zero length** (empty or unset).
        - **-n "$var"** True if string is **non-zero length** (has text).
        - **"$a" == "$b"** True if string $a equals string $b.
        - **"$a" != "$b"** True if string $a does not equal string
        - **$b.[["$var" =~ regex]]** True if string matches the specified regular expression.
        - **-eq** Equal to ==
        - **-ne** Not equal to !=
        - **-gt** Greater than >
        - **-ge** Greater than or equal to >=
        - **-lt** Less than <
        - **-le** Less than or equal to <=
    - **else** = Runs if ALL conditions above fail (non-zero exit codes) Same syntax as **if**
    - **fi** = marks the end (finish) of an if block
- **case** = provides a clean, efficient way to handle multi-way branching based on pattern matching. It is best used for Evaluating a single variable against fixed patterns or string values. Instead of writing long, nested `if...elif...else` chains when evaluating a single variable against multiple options, `case` compares a test string against several defined patterns sequentially. Just as `if`

ends with `fi`, `case` ends with `esac` ("case" spelled backward). Closing Parenthesis `)`: Every pattern must end with a closing parenthesis `)`. Double Semicolons `;;`: Each command block must end with `;;`. This acts like a `break` statement in languages like C/Python, preventing execution from falling through into the next pattern block. The Wildcard Asterisk `*)`: Acts as the `else` or `default` block. Because Bash matches patterns from top to bottom, `*)` must always be placed at the very end.

```
case "$VARIABLE" in
   pattern1)
      # Commands executed if $VARIABLE matches pattern1
      ;;
   pattern2|pattern3)
      # Commands executed if $VARIABLE matches pattern2 OR pattern3
      ;;
   *)
      # Default fallback (matches anything else)
      ;;
esac
```

    - `*`: Matches any string of zero or more characters.
    - `?`: Matches exactly one character.
    - `[...]`: Matches any single character within the specified set (e.g., `[yY]`).
    - `|`: Acts as an "OR" operator between multiple patterns.
- **Looping statements** = Bash offers four looping constructs that repeat a block of commands. All are closed with `done`.
    - **`for var in list; do ... done`** = iterates over each item in a list (words, filenames, or command output), running the loop body once per item with `var` set to the current item.

```
for user in alice bob carol; do
   echo "Creating home dir for $user"
   mkdir -p /home/$user
done
```

    - **`for (( init; condition; increment )); do ... done`** = the C-style variant, used when you need explicit numeric control (start, stop, step).

```
for (( i=0; i<10; i++ )); do
```

```
    echo "Counter: $i"
done
```

- **`while condition; do ... done`** = repeats the loop body for as long as `condition` evaluates to true (exit status 0), checking before each iteration.

```
count=1
while [ $count -le 5 ]; do
    echo "Count is $count"
    ((count++))
done
```

- **`until condition; do ... done`** = the inverse of `while` — repeats for as long as `condition` is false, checking before each iteration.

```
count=1
until [ $count -gt 5 ]; do
    echo "Count is $count"
    ((count++))
done
```

- **`continue`** = skips the rest of the current iteration and jumps to the next one. This works similarly to break, but continues the loop.
- **`break`** = Exits the loop immediately — no further iterations, and skips straight to code after done. This works similarly to continue, but breaks out of the entire loop.

```
for i in {1..10}; do
    if [ "$i" -eq 5 ]; then
        break
    fi
    echo "$i"
done
# Output: 1 2 3 4
```

- Comparisons = compares multiple values. Must be conducted within [ ] single brackets after an if statement.
    - **`[ ]`** (POSIX Test): Calls `/usr/bin/test` standard. Requires variable quoting and flag-based operators (`-eq`, `-gt`). Redirection characters (`<`, `>`) break unless escaped. Use **`[ ]`** when you need strict POSIX

compatibility (`/bin/sh`) using numeric flags (`-eq`, `-gt`) or string equality (`=`).

- `[[ ]]` (Bash Keyword): Shell-native engine supporting rich operators (`==` globs, `=~` regex, ASCII ordering) without word-splitting bugs. Use `[[ ]]` when you are working with Text, Wildcards (`*`), or Regex (`=~`).
- `(( ))` (Arithmetic Context): C-style evaluation environment strictly for integer math (`<`, `>`, `==`, `<=`, `>=`). Use `(( ))` when you are doing C-style Integer Math.
- Numerical = used for comparing numbers/integers mathematically. Numerical operators were created specifically as flag parameters for the `test` binary. Because they are plain flags (e.g., `test "$a" -eq "$b"`), the shell can parse them inside `[ ]` without syntax conflicts.
  - -eq = equal to, evaluates a numerical comparison mathematically. This is different than == which evaluates two items as text, not mathematically
  - -ge = greater than or equal to, mathematical comparison. Evaluates if the item on the left is greater than or equal to the item on the right
  - -gt = greater than, mathematical comparison. Evaluates if the item on the left is greater than the item on the right
  - -le = less than or equal to, mathematical comparison. Evaluates if the item on the left is less than or equal to the item on the right
  - -lt = less than, mathematical comparison. Evaluates if the item on the left is less than the item on the right
  - -ne = not equal to, mathematical comparison. Evaluates if the items are not equal
- String = used for comparing text/string values. must be used in [ ] or [[ ]] after an if statement. All of these operators can also be used for arithmetic comparisons if you use the (( ))... confusing I know…
  - > = greater than, evaluates whether a string of text comes later alphabetically. Only the first character of the comparison is evaluated. Requires [[ ]] double brackets to work properly
  - < = less than, evaluates whether a string of text comes earlier alphabetically. Only the first character of the comparison is evaluated. Requires [[ ]] double brackets to work properly
  - == = equal to, evaluates a text based comparison for character by character string matching. different than -eq

which compares numerical value. Requires [[ ]] double brackets to work properly. One key advantage == has over = is that with == bash treats the string on the right-hand side as a pattern rather than a strict literal string which means you can use wildcards * if you don't know the entire name

- = = equal to for POSIX standard, evaluates a text based comparison for character by character string matching. same as == but can be used inside [ ] single brackets. Under double brackets [[ ]], = and == are completely interchangeable… very confusing I know… basically this one is just more portable to different shells and can be used with [ ]
- =~ = Regular Expression (Regex) Matching Operator in Bash. Unlike = or == (which check for exact string equality or glob pattern matching), =~ tests whether a string matches a POSIX Extended Regular Expression (ERE) which matches any substring by default (unless anchored). The =~ operator MUST be used inside double brackets `[[ ]]`. See example below

```
ip="192.168.1.50"
if [[ "$ip" =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then
    echo "Valid IP syntax."
fi
```

- != = not equal to, evaluates a text based comparison for character by character string matching. different than -nq which compares numerical value. Can also use wildcards but needs [[ ]] to work with those.
- <= = less than or equal to, evaluates whether a string of text comes earlier or the same alphabetically. Only the first character of the comparison is evaluated. different than -le which compares numerical value. Can also use wildcards but needs [[ ]] to work with those.
- >= = greater than or equal to, evaluates whether a string of text comes later or the same alphabetically. Only the first character of the comparison is evaluated. different than -ge

which compares numerical value. Can also use wildcards but needs [[ ]] to work with those.

- test = true (0) or false (1) operator. test can also be used with the [ ] single brackets.

```
#identical test syntax
if test -f /etc/passwd; then echo "Found"; fi

if [ -f /etc/passwd ]; then echo "Found"; fi
```

    - `-n <string>` = True if the length of `<string>` is non-zero (not empty).
    - `-z <string>` = True if the length of `<string>` is zero (empty).
    - -d <path> = True if <path> exists and is a directory.
    - -f <file> = True if <file> exists and is a regular file.
    - ! = logical not. negates the outcome.
    - `-r <path>`: True if read permission is granted.
    - `-w <path>`: True if write permission is granted.
    - `-x <path>`: True if execute permission is granted.
    - `-e <path>`: True if `<path>` exists (regardless of type).
    - `-s <file>`: True if `<file>` exists and has a size greater than zero (is not empty).
    - -eq: Equal to ([$num1 -eq $num2]).
    - -ne: Not equal to.
    - -gt: Greater than.
    - -ge: Greater than or equal to.
    - -lt: Less than.
    - -le: Less than or equal to.
- Variables = a labeled storage location in system memory (RAM) that holds a value—such as a text string, a number, or a path—so that programs and scripts can reuse that data without hardcoding it. Normal variables are defined within the respective shell session. Local variables are specified within functions to ensure the variable doesn't leak out into the shell session. Environmental variables or global variables are "global" only within its own process tree (the current shell and its subshells, scripts, or programs).

- local = stops a function variable from leaking out into the parent shell, otherwise variables will leak out of the function into the shell session.
- export = exports a variable as a global variable. Scoped to the entire process tree. This is required for inheritance of variables in subshells.
  - `export VAR=value`: Sets and exports in a single step.
  - `-p`: Lists all exported variables in the current environment.
- Environment variables:
  - PATH: A colon-separated (:) list of directories where the shell searches for executable commands when a full path is not provided.
  - USER: Stores the username of the currently logged-in user.
  - HOME: Stores the absolute path to the current user's home directory (e.g., /home/username).
  - SHELL: Points to the absolute path of the user's default login shell (e.g., /bin/bash).
  - DISPLAY: Specifies the X11 server display address used by graphical applications (e.g., :0).
  - PS1: Defines the layout and appearance of the primary command prompt string (e.g., [\u@\h \W]\$).
- Arguments = (also called a positional parameter in shell scripting) is any piece of dynamic data you pass into a command, utility, or script when you invoke it from the terminal. (for example, echo "Target Server: $1")
  - $0: Holds the name of the script or command as it was executed (e.g., ./script.sh).
  - $1, $2, $3 ... $N: Holds individual positional arguments in order.For arguments above 9, you must use curly braces (e.g., ${10}) to prevent Bash from interpreting $10 as $1 followed by a literal 0.
  - $#: Holds the total count of positional arguments passed to the script (excluding $0).
  - $@: Holds all positional arguments passed to the script as individual, space-separated words ("$1" "$2" "$3"). This is the safest array-like expansion to use in loops.

        - $*: Holds all positional arguments combined into a single single-string variable separated by the first character of IFS ("$1 $2 $3").
        - $?: Holds the exit status code of the last executed command or function (0 = success, non-zero = error).
    - set = Displays all shell variables, environment variables, and shell functions currently defined.
    - unset = Removes a variable (shell or environment) from memory.
    - alias = While a variable is used to store data strings, paths, or numerical values in memory, an alias is a shortcut for commands, flags, and pipelines within a user's session. Unlike variables which fail for pipelines, redirections, and quotes, aliases do work for these. (Study Tip: Variables hold data that you pass into commands. Aliases replace commands)

```
# Using an Alias (WORKS):
alias mylog='cat /var/log/syslog | grep error'
mylog
# Using a Variable (FAILS):
MYLOG="cat /var/log/syslog | grep error"
$MYLOG
```

    - unalias = removes a previously defined alias from the active shell's memory.

Everyday Commands

Files & Directories

- **pwd** = print working directory
    - `-L` = Logical. This is the default behavior. It displays the path including any symbolic links you may have traversed to get there.
    - `-P` = Physical This flag tells the command to avoid symbolic links and instead display the "real," physical path on the storage device.
- **ls** = list contents in directory Syntax: ls [flags] [options] [file or directory]
    - -a = all file and subdirectory name, even hidden items
    - -d = directory metadata, not content

    - -t = sort by time
    - -S = sort by size
    - -r = sort reverse order
    - -F = classify filetype by indicator code (ex. *, /, =, >, @, |)
    - -i = inode index # of file
    - -R = recursive directory content, shows directories and subdirectory content
    - ll or ls -l = list long format of the file and info about file
    - -h = human readable format
    - -Z = primary tool for inspecting SELinux security contexts on files and directories. If `ls -l` shows you standard permissions, `ls -Z` acts as your "security-enhanced" version of that command.
- **touch** = create file with a given extension or update file timestamp to show file has been accessed (ex. touch test.txt)
    - `-a` (Access time only): Updates only the atime (last time the file was read).
    - `-m` (Modification time only): Updates only the mtime (last time the content was changed).
    - `-c` or `--no-create`: Updates the timestamps only if the file already exists. It will not create a new, empty file if the target is missing.
    - `-t [[CC]YY]MMDDhhmm[.ss]`: Allows you to set a very specific date and time for the file rather than using the current system time.
    - `-d <string>`: Allows you to use a human-readable date string (e.g., `touch -d "yesterday" file.txt`) to set the timestamp.
    - `-r <reference_file>`: Uses the timestamps of a specific "reference file" and applies them to your target file. This is useful for "resetting" a file's metadata to match a known good version.
- **mkdir** = make directory
    - -p = creates parent folders if they don't exist (ex. mkdir -p testfolder/testfolder2/)
    - -m = mode, allows you to set the file permissions in octal format (e.g., `mkdir -m 755 new_dir`)
    - -v = verbose
    - -Z = sets default SELinux security for the new directory
- **cd** = change directory
    - .. = one directory back on the directory tree
    - - = go back to the last used directory
    - ../.. = two directories back on the directory tree
    - **~** = (Home Directory): A shortcut representing the current user's HOME directory.

    - / = (Root Directory): A shortcut representing the system root directory.
    - -L = show logical links as real directory
    - -P = physical link, resolve symbolic links
- **cp** = plain and simple Linux copy tool (cp -r testdir testdir1)
    - -a = perform a recursive archive copy and keep original file attributes from original file
    - -f = force overwrite pre-existing destination files with same name as the file you are copying
    - -i = interactive, prompts before overwriting anything (USE THIS IF YOU ARE WORRIED OF LOSING ANYTHING)
    - -n = do NOT overwrite destination file if the name matches
    - -s = symbolic link instead of actually copying
    - -l, –link = hard link instead of actually copying
    - -R, -r = recursively copy all source subdirectories and files (USE THIS IF YOU ARE COPYING DIRECTORY TREE)
    - -u = update destination files with same name as DEST if source file is newer
    - -v = verbose detailed log of what is happening on the screen
    - -p = preserver file metadata with copy
- **install** = acts like a high-powered `cp` command, but it sets ownership, permissions, and directory structures all in a single step. Usually you would run this after you curl a binary file from the internet and make it executable on your system with chmod +x
    - **`-m`** (`--mode`): Sets file permissions (e.g., `-m 755` for executable, `-m 644` for standard file).
    - **`-o`** (`--owner`): Sets the file owner (requires root/sudo privileges).
    - **`-g`** (`--group`): Sets the file group ownership.
    - **`-d`** (`--directory`): Creates missing target directories if they don't exist (like `mkdir -p`).
    - **`-v`** (`--verbose`): Displays each file as it gets copied and configured.
- **mv** = move or rename a file (ex. mv -v testdir/testdoc testdocnew)
    - -f = force overwrite pre-existing destination file with same name as DEST
    - -i = interactive, ask before overwriting anything (USE THIS IF YOU ARE WORRIED OF LOSING ANYTHING)
    - -n = no clobber, do NOT overwrite destination file with the same name as DEST (skip anything that already has the name)
    - -v = verbose details
    - -u = update, move only if source is newer than destination
    - –backup = take a backup of destination file

- **rm** = remove/delete command
  - -d = deletes an empty directory
  - -f = force deletion even if some files don't exist in source
  - -i = interactive. ask for approval to delete (USE THIS EVERY TIME)
  - -I = bulk interactive, ask for approval when deleting over 3 files at a time
  - -R, -r = recursively deletes subdirectories and directories
  - -v = verbose logging
  - NEVER RUN **rm -rf /** OR IT WILL DELETE YOUR ENTIRE ROOT FOLDER
- **rmdir** = remove directory only if it is empty
  - -v = verbose logging
  - -p = parent, deletes directories and empty subdirectory trees
- **tree** = like ls command but lists all subdirectories within the directories in a visual tree diagram
  - -a = all files even hidden ones
  - -d = directories only
  - -f = full path
  - -l = show symbolic links too
  - -L = establish maximum display depth of directory tree
  - -P = pattern
- **ln** = hard link a file to another file name pointing to the same inode
  - -s = soft/symbolic link a file which essentially creates a shortcut (IF YOU DELETE THE ORIGINAL, THE SYMBOLIC LINK IS BROKEN)
  - -i = display inode
  - -o = display owner
  - -g = display group
- **file** = basic info about file type
  - `-b` (Brief mode): Suppresses the filename in the output. This is extremely useful in Objective 4.2: Shell Scripting when you only want the file type string (e.g., "ASCII text") and not the filename.
  - `-i` (MIME type): Instead of a human-readable description, it outputs the MIME type (e.g., `text/plain; charset=us-ascii`). This is critical for web server administration or automated mail processing.
  - `-L` (Dereference): By default, if you run `file` on a symbolic link, it just tells you it's a symbolic link. Using `-L` tells the command to "follow" the link and identify the file it actually points to.
  - `-z` (Look inside compressed files): Attempts to look inside compressed files (like `.gz` or `.bz2`) to identify the type of the contained file.

    - `-f <file>` (Read from list): Allows you to provide a text file containing a list of filenames to be processed. This is much faster than running the command 100 separate times.
    - `-s` (Special files): Used to read block or character special files (like those in `/dev`). Normally, `file` won't try to read these to avoid side effects; `-s` forces it to identify them.
- **stat** = detailed status and info of a file or filesystem pulled directly from the inode.
    - -f (Filesystem status): Instead of showing information about the specific file, it displays the status of the filesystem the file resides on, including total blocks and available inodes.
    - -L (Dereference): Tells the command to follow symbolic links and display the information for the target file rather than the link itself.
    - -t (Terse): Outputs the data in a compact, one-line format, which is ideal for integration into shell scripts for automated monitoring.
    - -c (Format): Allows you to use a specific format string to print only the data you need, such as octal permissions (%a) or the owner's UID (%u).
- **diff** = tell me the difference between files
    - -u = unified format (USE THIS MOST OF THE TIME)Displays differences in a "unified" format. This is the industry standard for creating patches because it provides context lines around changes, making it easier for the patch command to apply updates.
        - \+ = added line
        - \- = removed line
    - -e = create a script that makes the first file compare like the second file compared
    - -q = question, are the files different? Yes or no?
    - -s = are files the same? Yes or no?
    - -r = recursively compares subdirectories and the contents within them
    - -W n = width max for output
    - -y (Side-by-side): Produces a two-column output so you can visually compare files next to each other. This flag provides functionality almost identical to the sdiff command.
    - -i (Ignore case): Ignores case differences. This is vital when comparing configuration files where parameter names might have inconsistent capitalization but the same meaning.
    - -w (Ignore all white space): Ignores tabs and spaces when comparing lines. This helps filter out "noise" if one file uses tabs and the other uses spaces for indentation.
    - Output of Diff Explained:
        - a = add, lines from second file need to be added to first

        - c = change, lines from first need to be replaced by lines in second
        - d = delete, line in first file need to be removed to match second file
        - < = indicates this line is in the first file
        - > = indicates this line is in the second file
- **sdiff** = While the standard `diff` command provides a top-down list of changes, `sdiff` performs a **side-by-side** comparison, making it much easier for a human to visually inspect differences between two files in a terminal.
    - -s / --suppress-common-lines: This is perhaps the most useful flag. It hides all lines that are identical, showing you only the differences. This is a lifesaver when comparing large configuration files.
    - -o <file> / --output: This turns sdiff into an interactive merging tool. It will show you the differences and ask you which version (left or right) to save into a new output file.
    - -w <number> / --width: Sets the column width (default is usually 130 characters). If you are on a narrow terminal or a serial console, you may need to reduce this to see the output correctly.
    - -l / --left-column: If the lines are identical, it only prints the left column to save space.
    - -i / --ignore-case: Ignores case differences. This is vital for Objective 5.2 when troubleshooting service configurations where case sensitivity might be the root cause of an issue.
    - -W / --ignore-all-space: Ignores all white space when comparing. This helps you focus on the actual code or parameters rather than indentation changes.
    - Output of sdiff explained: When you run `sdiff file1 file2`, the output uses specific symbols in the middle column to show you how the files relate:
        - `|` **(Pipe):** The lines are different between the two files.
        - **<** **(Less than):** The line exists only in the first file (left side).
        - **>** **(Greater than):** The line exists only in the second file (right side).
        - **Blank space:** The lines are identical.
- **locate** = locate a file on the system (updated once a day via a default cron automation. If you need to manually update the locate database, type updateb in the cli)
    - -A = filenames that match all patterns
    - -b = only filenames that match a pattern, not directory names
    - -c = count the number of files that match
    - -i = case insensitive
    - -q = quiet, no displaying errors
    - -r = regular expression instead of pattern match

    - -w = filenames and directory names that match pattern
- **find** = locate files based on metadata starting in your current directory (real-time tool, but slower than locate)
    - -cmin 2 = displays file whose status changed 2 minutes ago
    - -empty = displays empty files
    - -gid 2 = groupID number is 2
    - -group groupname = displays files whose group is groupname
    - -inum 2 = displays files whose inode number is 2
    - -maxdepth 2 = go into directory 2 levels deep
    - -mmin 2 = displays files whose data changed 2 minutes ago
    - -name pattern = display files whose names match pattern
    - -iname pattern = display files whose names match pattern, case insensitive
    - -nogroup = display files that have no group assignment
    - -type pattern = display file type
    - -nouser = display files with no user assigned
    - -perm mode = display files whose permission match mode
    - -size +-n = display files that are n size or greater or less than n
    - -user name = files whose user is name
    - -mtime +-n = modified user or over n days ago

User & Account Management

- **adduser** = adding a new user to a Linux system. On debian-based distros, adduser is a completely different command than useradd. It is usually run without flags to trigger the interactive mode in debian distros. In RHEL, adduser is a symbolic link to useradd.
    - --home <DIR>: Specifies a custom path for the user's home directory instead of the default /home/username.
    - --shell <SHELL>: Sets the login shell for the user (e.g., /bin/bash). This is useful for creating service accounts that require /sbin/nologin.
    - --uid <ID>: Forces the use of a specific Unique Identifier (UID) rather than accepting the next available number in the standard range.
    - --ingroup <GROUP>: Adds the new user to an existing primary group instead of creating a new group with the same name as the user.
    - --disabled-password: Creates the user account but does not set a password, effectively preventing the user from logging in until an administrator runs the passwd command.
    - --gecos <STRING>: Allows you to pre-fill the "General Electric Comprehensive Operating System" (GECOS) field, which contains descriptive information like the user's full name or office number.

    - --system: Creates a system account. These accounts usually have UIDs in a lower range (often 1–999) and are typically created without a home directory or a login shell.
    - --no-create-home: Overrides the default behavior and prevents the creation of a home directory for the user.
    - When using `adduser`, the system automatically updates three critical configuration files that you must know
        - **`/etc/passwd`**: Stores basic user account information (UID, GID, Home, Shell).
        - **`/etc/shadow`**: Stores the encrypted password and account expiration details.
        - **`/etc/group`**: Updated if a new primary group is created for the user.
- **deluser** = deleting a user from a Linux system. On debian-based distros, deluser is a completely different command than userdel. It is usually run without flags to trigger the interactive mode in debian distros. In RHEL, deluser is NOT a command.
    - **--remove-home**: Deletes the user's home directory and their mail spool along with the account.
    - **--remove-all-files**: A powerful and potentially DANGEROUS flag that searches the entire filesystem for any files owned by the user and deletes them.
    - **--backup**: Creates a backup (usually a .tar.gz file) of the user's home directory and mail spool before deleting them.
    - **--backup-to <DIR>**: Specifies a specific directory where the backup file should be placed.
    - **--system**: Ensures that the user is only deleted if they are a **system account** (typically having a UID below 1000).
    - **--group**: When used with a group name instead of a username, it removes a group from the system (provided no users are currently using it as their primary group).
- **useradd** = adding a new user to a Linux system. Low-level, high-speed tool for account creation made for no-frills and commonly used in automation. Also comes as the default for RHEL distros and requires that you provide all details from the get-go via flags.
    - -m (Create home): Automatically creates the user's home directory if it doesn't exist and populates it with files from the /etc/skel template.
    - -s <shell> (Shell): Defines the user's login shell (e.g., /bin/bash) or prevents login for service accounts by using /sbin/nologin.

        - Can also be used to restrict login by adding to the /sbin/nologin shell.
        - Can also be used to restrict an account with guard rail by adding to /bin/rbash shell.
    - -u <UID> (User ID): Manually assigns a specific Unique Identifier (UID). This is critical when you need to match UIDs across different servers for consistent file permissions.
    - -g <group> (Primary Group): Assigns the user's primary Group Identifier (GID) by name or number.
    - -G <groups> (Secondary Groups): Adds the user to a comma-separated list of supplementary groups, such as the wheel or sudo groups for privilege escalation.
    - -c <comment> (Comment/GECOS): Adds text to the comment field in /etc/passwd, typically used for the user's full name or contact info.
    - -e <YYYY-MM-DD> (Expire date): Sets a specific date for when the account will be automatically disabled—a key requirement for Objective 3.4: Account hardening.
    - -r (System account): Creates a system account with a UID in the lower range (typically below 1000) and usually without a home directory.
    - -d <path> (Home directory): Specifies a custom path for the home directory if you aren't using the standard /home/username.
    - -D (Defaults): When used alone, this flag displays the current default values for useradd. When used with other flags, it *changes* those defaults for all future users.
- **userdel** = deleting a user from a Linux system. Low-level, high-speed tool for account deletion made for no-frills and commonly used in automation. It is the standard binary for user deletion across all major distributions, including **RHEL** and **Debian**. Also requires that you provide all details from the get-go via flags.
    - `-r` (Recursive/Remove): This is the most important flag for the exam. It removes the user's home directory and their mail spool along with the account. Without this flag, the user's files remain on the disk.
    - `-f` (Force): Forces the removal of the user account even if the user is still logged in. It also forces `userdel` to remove the user's home directory even if another user uses the same home directory. Use this with caution in Objective 5.2 (Troubleshooting).
    - `-Z` (SELinux): Removes any SELinux user mapping for the user. This is a common requirement in RHEL-based environments to ensure security policies are fully cleaned up.
- **usermod** = used to change almost any attribute of an *existing* user..

    - `-a -G` (Append Secondary Group): This is the most tested flag combo. The -G flag sets the user's secondary groups. If you don't use -a (append), it will remove the user from all other secondary groups they belong to.
        - *Example:* `usermod -aG sudo jdoe` adds `jdoe` to the `sudo` group without kicking them out of others
    - `-c <comment>`: Modifies the GECOS/Comment field (usually the user's full name) in `/etc/passwd`.
    - `-d <path>`: Changes the user's home directory path.
    - `-m` (Move contents): Usually used with `-d`. It physically moves the contents of the old home directory to the new location.
    - `-e <YYYY-MM-DD>`: Sets or changes the account expiration date—critical for account hardening.
    - `-g <group>`: Changes the user's Primary Group ID (GID) or name.
    - `-l <new_name>`: Changes the user's login name (renames the user). Note that this does *not* automatically rename the home directory.
    - `-L` (Lock): Locks the user's password by putting a `!` in front of the encrypted string in `/etc/shadow`. The user can no longer log in via password, however, they might still be able to login with an SSH key
    - `-U` (Unlock): Removes the lock (the `!`) from the password in `/etc/shadow`.
    - `-s <shell>`: Changes the user's login shell (e.g., changing from `/bin/sh` to `/bin/bash`).
        - Can also be used to restrict login by adding to the /sbin/nologin shell.
        - Can also be used to restrict an account with guard rail by adding to /bin/rbash shell.
    - `-u <UID>`: Changes the numerical User ID. Like `groupmod`, this can "orphan" files owned by the old UID.
- **groupadd** = creates new groups in Linux. This command is the primary way to organize users for the purpose of granting shared permissions to files, directories, or system resources
    - `-g <GID>` (Group ID): Allows you to manually assign a specific Group Identifier (GID) instead of letting the system choose the next available one.
    - `-r` (System group): Creates a system group. These typically have GIDs in the lower range (usually below 1000) and are used for background services rather than human users.

    - `-f` (Force): Tells the command to exit with a "success" status even if the group already exists. It also forces the command to proceed if you try to use a GID that is already assigned (canceling the effect of `-g` if the GID exists).
    - `-o` (Non-unique): Used in conjunction with `-g` to allow the creation of a group with a duplicate GID. This is generally avoided for security reasons but is sometimes necessary for legacy compatibility.
    - `-p <password>`: Allows you to set an encrypted password for the group (stored in `/etc/gshadow`). This is rarely used in modern environments but may appear in theoretical security questions.
- **groupdel** = deletes a group in a Linux system. Don't usually use flags.
    - `-f` / `--force`: This flag forces the deletion of the group, even if it is the primary group of a user. Warning: This is generally considered dangerous because it leaves users with a "ghost" GID that no longer exists in `/etc/group`, leading to permission issues.
    - `-R` / `--root CHROOT_DIR`: Applies the changes in a `chroot` environment. This is common when performing system recovery or managing a disk image from a different mount point.
    - `-P` / `--prefix PREFIX_DIR`: Similar to `-R`, this allows the command to look for the `/etc/group` file in a specific directory prefix rather than the system default.
    - `-h` / `--help`: Displays the usage summary (common for all shadow-utils).
- **groupmod** = modify a group
    - `-g <GID>` (Group ID): Changes the numerical Group ID of the specified group.
        - Crucial Exam Note: Changing a GID does *not* automatically update the files owned by that group. Files previously owned by the old GID will now appear as owned by a number (the old GID) rather than a name. You must manually fix this using `chown` or `find`.
    - `-n <NEW_NAME>` (New Name): Renames a group. This is a common administrative task when a department changes names.
    - `-o` (Non-unique): Used in combination with the `-g` flag to allow the group to share a GID that is already assigned to another group.
    - `-p <PASSWORD>`: Changes the encrypted password for the group (stored in `/etc/gshadow`). This is rarely used in modern environments but is technically a feature of the command.
    - `-R <CHROOT_DIR>`: Applies the changes within a specific `chroot` directory, useful for system recovery or container management.

- **groups** = a simple but essential tool for identifying group memberships.
    - `groups [username] =` Displays the groups a specific user belongs to.
        - Usage: `groups devuser`. If no username is provided, it defaults to the current user. The first group listed is typically the user's primary group, followed by any secondary (supplementary) groups.
- **id** = used to display the Real and Effective User IDs (UID) and Group IDs (GID) for a specific user or the current user. When you run `id` without any arguments, it provides a snapshot of the current user's identity, including: **uid**: The numeric user ID and the username. **gid**: The numeric primary group ID and the group name. **groups**: A comma-separated list of all supplementary groups the user belongs to. **context**: If SELinux is enabled, it will also show the security context
    - `-u` (--user): Displays only the numeric User ID (UID).
    - `-g` (--group): Displays only the numeric Primary Group ID (GID).
    - `-G` (--groups): Displays all numeric Supplementary Group IDs the user belongs to.
    - `-n` (--name): Used in conjunction with the flags above (e.g., `-un` or `-gn`) to display the name instead of the numeric ID.
    - `-Z` (--context): Displays only the SELinux security context of the current user.
    - `-r` (--real): Displays the real ID instead of the effective ID (useful when dealing with `setuid` or `sudo` scenarios)
- **chsh** = change shell. Used to change a user's default login shell (defaults to bash unless specified)
    - `-s <SHELL>` (shell): Specifies the absolute path to the new login shell (e.g., `/bin/zsh`). This is the most common flag used in scripts.
    - `-l` (list-shells): Displays the list of "authorized" shells currently installed on the system. This list is pulled directly from the `/etc/shells` configuration file.
    - `-u` (help): Displays a brief usage summary.
    - `-v` (version): Displays the version information of the utility.
- **passwd** = lightweight tool that changes the password for a user account, locks accounts, and manages password policies
    - -l / -u = locks or unlocks the users password, however, they might still be able to login with an SSH key when you lock the password this way.
    - -e = expires user's password to make them change it
    - `-d` (Delete/Empty): Removes a user's password entirely. DANGEROUS: This makes the account passwordless, which can be a massive security risk, though some configurations allow this for quick internal logins.

    - `-S` (Status): Displays the current status of the user's password (locked, set, or empty) and the date it was last changed.
    - `-n <days>` (Minimum days): Sets the minimum number of days that must pass before a user is allowed to change their password again.
    - `-x <days>` (Maximum days): Sets the maximum number of days a password remains valid before it must be changed.
    - `-w <days>` (Warning): Sets the number of days the user will see a warning message before their password expires.
    - `-i <days>` (Inactive): Sets the number of days an account can remain "expired" before it is automatically disabled by the system.
- **chage** = change age, unlike passwd, chage is a power tool that manages account AND password expiration and aging policies. chage only works on a per-user basis. It does not affect system-side paramenters. For that, look to /etc/login.defs (aging and expiration) and /etc/security/pwquality.conf (complexity, length, history, and reuse) files.
    - `-l` (List): Displays the current aging information for a specific user. This is your first step in Objective 5.2 (Troubleshooting) when a user can't log in despite having the right password.
    - `-m <days>` (Min days): Sets the minimum number of days required between password changes. Setting this to a non-zero value prevents users from changing their password and immediately changing it back to the original.
    - `-M <days>` (Max days): Sets the maximum number of days a password is valid. This is the core of "password rotation" policies.
    - `-W <days>` (Warning): Sets how many days of warning a user receives before their password expires.
    - `-I <days>` (Inactive): Sets the number of days an account can remain with an expired password before the account is automatically locked. Think of this as a "grace period."
    - `-E <YYYY-MM-DD>` (Expire): Sets a specific date for the account itself to expire. This is perfect for temporary contractors or seasonal employees. It also ensures a user cannot login with either a password or an SSH key which passwd -l and usermod -L cannot do
    - `-d <days or date>` (Last Day): Manually sets the date the password was last changed.
        - Pro-Tip: Running `chage -d 0 <username>` is the standard way to force a user to change their password immediately upon their next login.

- **getent** = get entries, used to retrieve entries from important file or databases that the Name Service Switch (NSS) configures. Tip: while you might be used to using `cat /etc/passwd` to view user accounts, `getent passwd` is the superior "pro" method because it queries the databases configured in `/etc/nsswitch.conf`.
    - `getent passwd [username] =` Queries information for a specific user.
        - Usage: Instead of piping to `grep`, you can run `getent passwd jsmith`. If the user exists (locally or via network), it returns the standard colon-separated string. If not, it returns nothing and an exit code of 2.
    - `getent passwd [UID] =` Queries information by the User ID number.
        - Usage: `getent passwd 1001`. This is useful for troubleshooting file ownership issues where only a UID is visible (e.g., in `ls -l` output).
    - `getent group =` While not `passwd`, you must know that `getent` works with other databases. `getent group` displays all groups (local and network) just as `passwd` does for users.
    - `getent shadow =` Retrieves password hash information (requires root/sudo).
        - Used to verify if password aging or hashing is visible through the service switch.
    - `getent -s [service] passwd` (The `-s` flag) = Overrides `/etc/nsswitch.conf` to query a specific service.
        - Usage: `getent -s files passwd` will *only* look at the local `/etc/passwd` file, ignoring LDAP or SSSD. This is a critical troubleshooting step if you suspect network authentication is failing.
- **last** = used to list the history of all logins and logouts. It is essential for auditing system access, tracking user activity, and investigating potential security breaches.
    - -n [number] (or -[number]): Limits the output to a specific number of lines (e.g., last -n 5 shows the 5 most recent logins).
    - -f [file]: Tells the command to read from a specific file instead of the default /var/log/wtmp. This is useful for analyzing older, rotated logs.
    - -F (--fulltimes): Displays the full login and logout times and dates, which is critical for precise forensic timelines.
    - -i (--ip): Displays the IP address of the remote host instead of just the hostname, aiding in network-based troubleshooting

    - -a (--hostlast): Moves the hostname column to the end of the output to prevent long names from being truncated.
    - -x: Displays system shutdown entries and run-level changes, helping you determine if a system crash or a manual reboot occurred
- **lastlog** = While the `last` command shows a history of all logins and logouts , `lastlog` is used to examine the **most recent** login for each user.
    - `-u` (or `--user`): Displays the last login information for a specific user or range of users only.
    - `-t` (or `--time`): Filters the output to show only those logins more recent than a specified number of days.
    - `-b` (or `--before`): Displays only the users who have not logged in since a specified number of days (excellent for finding stale accounts that should be locked or deleted).
    - `-S` (or `--set`): Used to manually set or update the last login record for a user (less common on the exam, but useful for administrative "cleanup").
    - `-C` (or `--clear`): Clears the last login record for a specified user.
- **w** = essentially a "three-in-one" tool that provides a summary of system status, who is currently logged in, and what processes they are currently running.
    - -h (or --no-header): Suppresses the header (the uptime and column title lines), which is useful if you are piping the output into a script or automation tool.
    - -s (or --short): Provides a "short" format by omitting the login time, JCPU, and PCPU times to keep the display clean.
    - -f (or --from): Toggles the display of the FROM (remote hostname or IP address) field.
    - -i (or --ip-addr): Displays the IP address instead of the hostname in the "FROM" field, which is vital for network troubleshooting and security auditing.
    - -u (or --no-current): Ignores the username while figuring out the current process and CPU times.
- **who** = used to List account information. While the `w` command provides a summary of system activity and what users are doing, `who` is a more focused tool for identifying who is currently logged into the system and which terminal (TTY) they are using
    - `-a` (or `--all`): Provides a comprehensive output that includes information about the last boot, dead processes, and more—essentially combining several other flags.
    - `-b` (or `--boot`): Displays the date and time of the last system boot.

    - `-H` (or `--heading`): Adds a header row to the output, making it easier to read columns like NAME, LINE, and TIME.
    - `-m`: Displays only information about the current terminal session (equivalent to the command `who am i`).
    - `-q` (or `--count`): A "quick" mode that shows only the login names and the total count of users currently logged in.
    - `-r` (or `--runlevel`): Displays the current system runlevel (mostly relevant for legacy systems, but still part of the standard utility).
    - `-u` (or `--users`): Lists users who are currently logged in and includes their idle time and Process ID (PID).
- **whoami** = does exactly one thing: it prints the effective username associated with the current shell session. (Useful if you want to make sure you are root)

Process & Job Management

- **ulimit** = displays or sets per-process resource limits for the current shell session (e.g., max open file descriptors, max processes, max memory). (ex. `ulimit -n 4096` raises the open-file-descriptor limit for the session)
    - **`-a`** = lists every current soft limit.
    - **`-n`** = max number of open file descriptors.
    - **`-u`** = max number of processes a user can run.
    - **`-Hn / -Sn`** = view or set the hard (`-H`) vs. soft (`-S`) limit for a given resource.
- **`bg`** = resume a suspended job and keep it running in the background. You must first "pause" the task with **`Ctrl+Z`**, then type `bg`. You use `bg` when you started a command in the foreground, realized it's taking way longer than expected, and now you want your terminal back without killing the task.
    - `bg`: Resumes the most recent job you suspended.
    - `bg %1`: Resumes Job ID #1 (the number you see when you run the `jobs` command).
    - **`bg %name`**: Resumes a job that starts with a specific string (e.g., `bg %find`).
- **`fg`** = foreground command. Opposite of the bg command, the fg command is a "reclaiming" tool. If the bg command hides a task, the fg command brings it back to center stage so you can interact with it.
    - `fg` (no arguments): Brings the current job (the one marked with a `+` in the `jobs` output) to the foreground. This is usually the most recently suspended or backgrounded task.

    - `fg %[job_id]`: Pulls a specific job by its ID. For example, `fg %2` brings job number 2 to the foreground.
    - `fg %[string]`: Brings the job that starts with a specific command name. For example, `fg %vim` would pull your suspended text editor back up.
    - `fg %?string`: Brings the job that contains a specific string anywhere in its command line.
- **&** = (Background): Placing this at the end of a command runs that process in the background, allowing the shell to continue immediately. You use `&` when you know before you hit Enter that a command is going to take a long time and you don't want it to lock up your terminal.
- **jobs** = lists all processes and background jobs being managed by your specific shell session. While ps and top show you everything happening on the computer, jobs only shows you what you started in that specific terminal window-specifically things you've backgrounded with & or paused with ctrl + z.
    - `-l`: (Long format) Displays the Process ID (PID) alongside the Job ID and status. This is the most important flag to know because you often need the PID to use tools like `strace` or `renice`.
    - `-p`: Displays only the PIDs of the jobs. This is incredibly useful in shell scripting when you want to pass a list of background PIDs to another command.
    - `-r`: Filters the list to show only Running jobs.
    - `-s`: Filters the list to show only Stopped (suspended) jobs.
    - `-n`: Only displays jobs that have changed status since the shell last notified you (e.g., a job that just finished).
    - Output explained:
        - **`[1]`**: The **Job ID**. This is what you use with `fg %1` or `kill %1`.
        - **`+` (Plus sign)**: This is the **"current" job**. If you type `fg` without a number, this is the one that will come to the front.
        - **`-` (Minus sign)**: This is the **"previous" job**. It will become the "current" job if the one with the `+` finishes.
        - **Status**: Usually `Running`, `Stopped`, or `Done`.
- **kill** = force terminate a specific program. It can also be used to pause or resume jobs too. By default, it sends SIGTERM (15) to terminate a specific PID.
    - `-l` (List): Displays a list of all available signal names and their corresponding numbers. This is a lifesaver if you forget a signal number during a lab.
    - `-s [signal]`: Explicitly specifies the signal to be sent (e.g., `kill -s SIGKILL 1234`).

    - `-[signal_number]`: A shorthand way to send a signal (e.g., `kill -9 1234`).
    - `-[signal_name]`: Another shorthand using the name instead of the number (e.g., `kill -SIGTERM 1234`).
    - `-a`: Tells `kill` to process names even if they don't belong to the current user (though you'll still need `sudo` for this to actually work).
    - -SIGTERM (or -15) = graceful termination, asks program to save data and shutdown
    - -SIGKILL (or -9) = Force kill
    - -SIGHUP (or -1) = hangup, often used to tell daemon to reload config files without fully stopping
    - -SIGINT (or -2) = terminate, just like pressing ctrl + c
    - -SIGSTOP (or -19) = pauses a process, just like pressing ctrl + z
    - -SIGCONT (or -18) = resumes or continues after a program has been paused by -SIGSTOP
- **killall** = utility used to send a signal to every process running under a specific name. (ex. killall chrome) By default, it sends SIGTERM (15) to terminate all programs with a certain name
    - `-i` (interactive): The "safety" flag. It asks you for confirmation before killing each individual process. Highly recommended if you aren't 100% sure what you're about to terminate.
    - `-l` (list): Just like the `kill` command, this lists all the known signal names so you don't have to guess.
    - `-q` (quiet): If you try to kill "firefox" but no firefox processes are running, `killall` usually complains with an error message. `-q` tells it to stay silent.
    - `-s` (signal): Specifies the signal to send (e.g., `killall -s 9 chrome`). You can also just use the shorthand `-9`.
    - `-u [user]`: Only kills processes that match the name and are owned by a specific user. This is great for an admin who needs to kick a specific user's stuck apps without affecting other people.
    - `-v` (verbose): Reports whether the signal was successfully sent to the processes.
    - `-w` (wait): Tells `killall` to wait for all the processes to actually die before the command finishes and gives you back your prompt. This is useful in scripts to ensure a "clean slate."
    - -SIGTERM (or -15) = graceful termination, asks program to save data and shutdown

    - -SIGKILL (or -9) = Force kill
    - -SIGHUP (or -1) = hangup, often used to tell daemon to reload config files without fully stopping
    - -SIGINT (or -2) = terminate, just like pressing ctrl + c
    - -SIGSTOP (or -19) = pauses a process, just like pressing ctrl + z
    - -SIGCONT (or -18) = resumes or continues after a program has been paused by -SIGSTOP
- **pkill** = utility used to send a signal to every process running under a specific pattern match (regular expression). By default, it sends SIGTERM (15) to terminate all programs with a certain name
    - `-f` (Full): (High Value) By default, `pkill` only looks at the process name. With `-f`, it looks at the full command line, including arguments. This is how you distinguish between two different Python scripts (e.g., `python script_A.py` vs `python script_B.py`).
    - `-u [user]`: Only kills processes owned by the specified Effective User ID.
    - `-U [user]`: Only kills processes owned by the specified Real User ID.
    - `-t [tty]`: Targets processes associated with a specific terminal (e.g., `pkill -t pts/0`). This is useful for kicking a "hung" user session.
    - `-n` (Newest): Signals only the most recently started process that matches the pattern.
    - `-o` (Oldest): Signals only the first process started that matches the pattern.
    - `-v` (Inverse): Signals every process except those that match the pattern. Use this with extreme caution!
    - `-x` (Exact): Forces the pattern to match the process name exactly (essentially making it behave like `killall`).
    - `-[signal]`: Just like `kill`, you can specify the signal by name or number (e.g., `pkill -9` or `pkill -SIGTERM`).
    - -SIGTERM (or -15) = graceful termination, asks program to save data and shutdown
    - -SIGKILL (or -9) = Force kill
    - -SIGHUP (or -1) = hangup, often used to tell daemon to reload config files without fully stopping
    - -SIGINT (or -2) = terminate, just like pressing ctrl + c
    - -SIGSTOP (or -19) = pauses a process, just like pressing ctrl + z
    - -SIGCONT (or -18) = resumes or continues after a program has been paused by -SIGSTOP

- **nohup** = no hang up. Utility that allows a process to continue running even after the user who started it has logged out or closed their terminal.
    - The `&` (Ampersand): While not a flag of `nohup`, the exam expects you to know that `nohup` is almost always used with `&` at the end (e.g., `nohup ./script.sh &`). Without the `&`, the command runs in the foreground and still locks up your current terminal.
- **ctrl + c** = signal interrupt (SIGINT 2). Terminates a process and kills it immediately. It is essentially the cancel button for a running task. The process will be in the terminated state after you run ctrl + c
    - Signal Number: **2**: You must memorize that `Ctrl + C` corresponds to signal number **2**. You might see a question asking what `kill -2 [PID]` does; it's the same as hitting `Ctrl + C`.
- **ctrl + z** = signal terminal stop (SIGTSTP 15). It is essentially the pause button for a running task.It allows you to freeze a process in its tracks without losing its current state or data. The process will be in the stopped state when you run ctrl + z and it can be resumed by running `bg`. A process suspended by `Ctrl + Z` still resides in RAM but consumes **0% CPU** because the scheduler stops giving it cycles
- **ctrl + d** = End of input session (EOF). Unlike `Ctrl + C` (which kills a process) or `Ctrl + Z` (which pauses it), `Ctrl + D` tells the system: *"I'm done sending data; you can stop listening now."*
    - EOF (End of File): It sends a special character that signifies the end of a transmission. It tells the reading process that there is no more data coming.
    - Shell Logout: If you are at a bare Bash prompt, pressing `Ctrl + D` tells the shell that its input stream (your typing) is finished. This causes the shell to exit, effectively logging you out of that session or closing that terminal window.
- **exec** = used to replace the current shell process with a new program . It is the "transformer" of the Linux world: it doesn't start a new process alongside the current one; it overwrites the current shell with a new command. In normal Linux life, when you run a command (like `ls`), the shell clones itself (called "forking"), and the clone runs the command while the original shell waits for it to finish. `exec` is different. It is a "Hostile Takeover." It doesn't clone anything. It tells the current process to delete itself and let the new command take over its physical space (the Process ID).

    - `-a [name]`: Sets argv[0] (the name of the process). This allows you to start a process and give it a different name in the process list (`ps`), which is occasionally used for simple obfuscation or script identification.
    - `-c`: Executes the command with an empty environment. This is used in security or testing scenarios where you want to ensure no environment variables (like `$PATH` or `$USER`) are passed from the shell to the new process.
    - `-l`: Tells the shell to behave as if it were a login shell. It places a dash (`-`) at the beginning of the process name (e.g., `-bash`), signaling the program to read login profile files.
    - `exec > [file]` (Redirection): This is the "hidden flag" of `exec`. Using `exec` without a command but with a redirection symbol allows you to redirect the Standard Output (stdout) or Standard Error (stderr) for the entire remainder of that shell session.
    - `exec < [file]`: Redirects Standard Input (stdin) for the shell, often used in scripts to read a file line-by-line without using a loop structure.
- **lsof** = list open files. In the Linux philosophy, "everything is a file"—this includes regular files, directories, shared libraries, character special files, and even network sockets.
    - -i (Internet files): Displays all open network sockets. This is a favorite for security audits and troubleshooting network services. *Tip:* lsof -i :80 will show you exactly which process is listening on port 80.
    - -u <user> (User): Filters the list to show only files opened by a specific user. This is helpful when investigating a specific user's impact on system resources.
    - -p <PID> (Process ID): Lists all files opened by a specific process ID. If a process is behaving strangely, this shows you exactly what it's touching.
    - +D <directory> (Directory): Recursively searches a directory and lists all open files within it. *Tip:* If you get a "device is busy" error while trying to umount /mnt/data, you would run lsof +D /mnt/data to find the culprit processes.
    - -t (Terse): Returns only the Process IDs (PIDs). This is perfect for Objective 4.2 (Shell Scripting) because you can pipe the output directly into a kill command: kill -9 $(lsof -t /path/to/file).
    - -c <command> (Command): Lists files opened by processes starting with a specific name (e.g., lsof -c apache).
    - -n (No DNS): Inhibits the conversion of network numbers to hostnames. This makes the command run much faster because it skips DNS lookups.

- **crontab** = automation scheduler (cron is the scheduler which uses crond as the background service.) Crontab is the cron table that you can add tasks to.
    - -e = edit (USE THIS MOST OF THE TIME)
    - -l = list the contents of your current crontab
    - -r = deletes your crontab file (BEWARE WHEN USING THIS)
    - -u user = allows root to view or edit another user's crontab (ex. sudo crontab -u bob -e)
    - `[min] [hour] [day] [month] [weekday] [USER] [command]`
    - `(* * * * *` user command file.sh)
    - `*  *  *  *  *`
    - `|  |  |  |  |`
    - `|  |  |  |  +----- Day of Week (0-6) (0 is Sunday, or use names)`
    - `|  |  |  +---------- Month (1-12)`
    - `|  |  +-------------- Day of Month (1-31)`
    - `|  +------------------- Hour (0-23)`
    - `+------------------------ Minute (0-59)`
    - `*` (Asterisk): Matches all values (e.g., an asterisk in the hour field means "every hour").
    - `,` (Comma): Defines a list of discrete values (e.g., `1,3,5` in the day of week field).
    - `-` (Hyphen): Defines a range of values (e.g., `1-5` for Monday through Friday).
    - `/` (Slash): Defines step/interval values (e.g., `*/15` in the minute field means "every 15 minutes").
- **anacron** = If a scheduled task is missed because the computer was powered off during its designated runtime, `anacron` catches it. When the system boots back up, `anacron` detects that the execution window was missed and runs the job after a brief, safe delay.
    - `-f` (Force): Overrides timestamps and forces `anacron` to execute all jobs instantly, regardless of whether they have already been run during the specified period.
    - `-u` (Update): Updates the timestamps of all jobs in `/var/spool/anacron/` to the current date without actually executing the underlying commands. This effectively "skips" the current execution cycle.
    - `-s` (Serialize): Executes scheduled jobs sequentially rather than concurrently. `anacron` will wait for one job to completely finish before triggering the next one in the queue.

    - `-n` (Now): Instructs `anacron` to run all jobs immediately, ignoring any "delay" parameters configured in the `/etc/anacrontab` file. This is frequently combined with sequential execution (e.g., `anacron -sn`).
    - `-d` (Don't Fork / Debug): Keeps `anacron` running in the foreground as an active terminal process instead of letting it split off into a background daemon. It prints all execution and debugging messages directly to standard error (`stderr`), making it a vital troubleshooting tool.
    - `-t <config-file>` (Template/Alternative Configuration): Commands `anacron` to bypass the standard `/etc/anacrontab` file and execute jobs using a custom, user-specified configuration template instead.
    - [period_in_days] [delay_in_minutes] [job-identifier] [command]
    - * * * *
    - | | | |
    - | | | +---------- Command/script
    - | | +------------- Job ID
    - | +---------------------- Delay in minutes after bootup before anacron triggers
    - +----------------------- Period in days (1-31)
- **at** = While `cron` and `anacron` are used for recurring tasks , The `at` command is used to schedule a command or script to execute exactly once at a specific time in the future. When you use `at`, it opens an interactive prompt where you type the commands you want to execute, ending the input with `Ctrl + d`. (ex. at 23:45). Alternatively, you can pipe or direct scripts into it. (ex. echo "/usr/local/bin/clean_tmp.sh" | at now + 3 hours)
    - `-f <file>` (File): Reads the commands from a specific script or file instead of forcing you to type them into the standard interactive terminal prompt.
    - `-m` (Mail): Sends an email via the local mail transfer agent to the user when the job completes, even if the command produced no standard output or errors.
    - `-l` (List): Lists the pending jobs for the currently authenticated user. This is a direct built-in alias for running the standalone `atq` command.
    - `-d <job_id>` (Delete): Deletes a scheduled job from the queue using its specific job number. This is a direct built-in alias for running the standalone `atrm` command.
    - `-c <job_id>` (Cat): Outputs the full contents of the specified job script to standard output (`stdout`). This is incredibly useful on the exam for troubleshooting exactly what environment variables and commands a scheduled job is going to run.

    - `-q <queue>` (Queue): Specifies a distinct queue character (a single letter from `a` to `z` or `A` to `Z`). Lower letters have lower "niceness" (higher priority). The default queue for `at` is `a`.
    - `-v` (Version / Time Verification): When entering a job, it displays the exact calculated time the job will run before you write the execution steps, preventing scheduling mathematical errors.
    - `-t <time_arg>` (Time Specification): Allows you to specify the execution time using the traditional POSIX timestamp format (`[[CC]YY]MMDDhhmm[.ss]`) rather than natural language strings like "noon" or "tomorrow".
- **top** = real-time, dynamic and interactive display showing task manager-like interface of all running programs (usually used for showing overall system health and performance)
    - Command-Line Flags (at launch):
        - -d [seconds]: Sets the delay interval between screen updates.
        - -u [username]: Filters the display to show only processes owned by a specific user.
        - -p [PID]: Monitors only specific Process Identification Numbers.
        - -n [number]: Tells top to update the screen a specific number of times and then exit (useful for scripting).
    - Interactive Keys (while running):
        - h: Displays the help screen, showing all available commands.
        - k: Allows you to kill a process by entering its PID.
        - r: Allows you to renice a process, changing its priority while it is running.
        - M: Sorts the process list by Memory usage.
        - P: Sorts the process list by CPU usage (the default view).
        - q: Quits the top utility.
        - 1: (The number one) Toggles the display to show individual CPU cores rather than a single combined average.
- **htop** = human-readable top, user-friendly, interactive and color-coded version of top
    - The "Management" Keys (Critical) These keys perform actions listed as specific exam objectives:
        - F9 (Kill): Opens the signal menu to terminate a process. You'll need to know which signal to send—usually 9 (SIGKILL) or 15 (SIGTERM).
        - F7 (Nice -): Increases the priority of a process by decreasing its nice value.

        - F8 (Nice +): Decreases the priority of a process by increasing its nice value.
    - The "Verification" Keys (Essential) These keys help you analyze processes as required by Objective 2.3:
        - F5 (Tree View): Organizes processes into a parent-child hierarchy, serving the same purpose as the pstree command.
        - u (Filter by User): Allows you to show only processes owned by a specific account, which is a core part of Local Account Management.
        - F3 (Search): Lets you type the name of a specific binary or service to find its PID.
        - F4 (Filter): Similar to search, but it hides all other processes except those matching your string.
    - The "Navigation" Keys
        - F6 (Sort): Allows you to choose the sorting criteria, such as %CPU, %MEM, or PID.
        - F2 (Setup): Used to customize the interface, such as adding meters for different CPU cores or storage I/O.
        - F10 (Quit): Safely exits the utility.
- **atop** = advanced top, an advanced power-tool for system and process monitor that provides a holistic, color-coded view of hardware resources—CPU, memory, disk, and network—and the processes utilizing them. It is the only top-like command that includes network and disk I/O activity per process
    - Flags to know (At Launch)
        - -r [file]: Reads and displays data from a raw log file. atop often runs as a background daemon, and this flag allows you to perform historical analysis of a system crash or performance spike.
        - -m: Displays memory-related information, such as resident memory, virtual memory size, and swap usage.
        - -d: Displays disk utilization details, including read/write transfers and the percentage of time the disk was busy.
        - -n: Displays network-related information (note: this often requires an additional kernel module/daemon to be fully functional).
        - -u: Summarizes resource consumption per user, which is vital for identifying a specific user causing a system slowdown.
        - -c: Shows the full command line for each process rather than just the process name.
        - -y: Breaks down the display into individual threads within processes.

    - Resource-Specific Views These keys change the bottom half of the screen to focus on specific hardware components:
        - m: Switches to Memory mode. This is vital for spotting memory leaks or heavy swap usage.
        - d: Switches to Disk mode. Use this to troubleshoot high I/O wait times or disk latency.
        - n: Switches to Network mode (requires the netatop kernel module to be active).
        - v: Shows Variable/Status info, which is great for seeing process start times and specific process states (like Zombie or Interrupted) .
    - Sorting and Identification When you need to find the "noisy neighbor" on your system, use these:
        - C: Sorts by CPU consumption (this is the default).
        - M: Sorts by Memory usage.
        - D: Sorts by Disk activity.
        - u: Provides a User-based summary. This is a "bare minimum" skill for Objective 2.2 (Local Account Management) to see which specific account is hogging resources.
        - c: Toggles between the process name and the full command line.
    - Navigation and Utility
        - t: Switches to a threaded view, showing individual threads rather than just the parent process.
        - h: The "save me" key—opens the Help menu.
        - q: Quits the utility.
- **mpstat** = multi=processor statistics, performance monitoring tool specifically for CPU performance stats. It is the go-to tool for identifying if a performance bottleneck is caused by a single core being overwhelmed or a system-wide CPU saturation.
    - `-P {cpu | ALL}`: Specifies which processor to report on. Using `-P ALL` is the most common exam scenario, as it breaks down every individual core.
    - `[interval] [count]`: Not a flag with a letter, but a syntax requirement. For example, `mpstat 2 5` tells the system to take a report every 2 seconds, a total of 5 times.
    - `-u`: Reports CPU utilization (this is the default report if no other flags are used).
    - `-A`: An "all-in-one" flag that displays all statistics that `mpstat` can provide, including interrupts and utilization.

    - `-I {keyword}`: Reports interrupt statistics. This is useful for troubleshooting hardware issues (Objective 5.2).
    - `-V`: Displays the version of the utility.
- **pidstat** = specialized performance monitoring tool used for process verification. Unlike `top`, which gives a system-wide overview, `pidstat` is often used to "zoom in" on a specific **Process ID (PID)** to see exactly how much CPU, memory, or disk I/O that single process is consuming over a set interval.
    - **`-p [PID]`**: Filters the output to show statistics for a specific **Process Identification Number**.
    - **`-u`**: Displays **CPU utilization** for the specified process (this is the default view).
    - **`-r`**: Reports **Memory utilization**, including page faults and memory usage.
    - **`-d`**: Reports **Disk I/O statistics**, showing read and write rates for the process.
    - **`-w`**: Displays **Task switching** activity, which is vital for diagnosing **High context switching** symptoms.
    - **`-t`**: Shows statistics for **threads** associated with the selected tasks, moving beyond just the parent process.
    - **`[interval] [count]`**: Not a letter flag, but a standard syntax (e.g., `pidstat 1 5`) to report statistics at a specific interval for a specific number of times.
- **ps** = processes list, point in time snapshot of current processes running on a system (usually used for finding a specific PID)
    - aux (BSD Style - no dash):
        - a: Displays processes for all users, not just the current user.
        - u: Displays a user-oriented format, including CPU and memory usage percentages.
        - x: Includes processes that do not have a controlling terminal, such as background daemons.
    - -ef (UNIX/Standard Style - with dash):
        - -e: Displays every process on the system.
        - -f: Provides a full-format listing, which is critical because it displays the Parent Process ID (PPID).
        - -u [user]: Filters the list to show only processes owned by a specific user.
        - -p [PID]: Filters the output to show information for a specific Process Identification Number.

        - -o [format]: Allows for a custom output format, where you can specify exactly which columns to display (e.g., ps -o pid,comm,nice).
        - --forest: Displays a visual tree of processes, showing the relationship between parents and children.
- **pstree** = process tree. While commands like `ps` or `top` give you lists, `pstree` visually organizes processes into a tree-like hierarchy.
    - **`-p`**: Displays the **PID** for every process name in the tree, which is critical when you need to identify which specific child process to kill.
    - **`-a`**: Shows the **command-line arguments** used to start each process, helping you distinguish between multiple instances of the same service.
    - **`-u`**: Shows **uid transitions**; if a process was started by a user other than the parent, the new username is displayed in parentheses.
    - **`-h`**: Highlights the **current process** and its ancestors (useful if you want to see where your own shell lives in the hierarchy).
    - **`-n`**: Sorts processes with the same ancestor by **PID** instead of alphabetically.
    - **`-A`**: Uses **ASCII characters** to draw the tree, ensuring it displays correctly even on older terminals or over limited remote connections.
    - **`-s`**: Shows the **parentage** of a specific PID; it "works backward" from a child to the root process (usually `systemd`).
- **strace** = system call tracer. While `top` tells you a process is running and `ps` tells you who owns it, `strace` shows you exactly what the process is saying to the Linux Kernel.
    - **`-p [PID]`**: Attaches `strace` to an **already running process**. This is the most common use case for troubleshooting a service that has "hung."
    - **`-o [file]`**: Redirects the output to a **text file**. `strace` output is sent to *stderr*, so a simple `>` redirect doesn't always work as expected; this flag is the reliable way to save logs for analysis.
    - **`-e [trace=syscall]`**: Filters for **specific system calls**. For example, `-e trace=open,connect` will only show you which files the process tries to open and which network connections it tries to make.
    - **`-f`**: Tells `strace` to **follow forks**. If the process starts child processes (which most modern apps do), this flag ensures you see the syscalls for the children as well.
    - **`-c`**: Provides a **summary/count** of all system calls made. This is excellent for **Performance Troubleshooting** (Objective 5.5) to see which syscall is taking the most time or failing the most often.

    - **`-v`**: Verbose mode. It prevents `strace` from abbreviating the contents of structures, which is useful when you need to see the exact data being passed to the kernel.
    - **`-s [size]`**: Specifies the maximum string size to print. By default, it's small (32 bytes); increasing this helps you read full file paths or data buffers.
- **nice** = process priority management utility. In Linux, "priority" is measured by a "niceness" value. A process that is "nicer" is more willing to give up CPU time to other processes. Conversely, a process that is "less nice" is more aggressive and demands more CPU time. Standard Users: Can only *increase* the niceness value (e.g., move from 0 to 10). They can only make their processes "nicer" (lower priority). Root (Superuser): Can set any value, including negative values to give a process "High Priority."
    - Niceness Scale:
        - Range: `-20` to `19`.
        - -20: The Highest Priority (least nice).
        - 19: The Lowest Priority (most nice).
        - 0: The Default priority for most user-started processes.
    - `-n [value]`: This is the primary flag used to set the increment of niceness.
        - If you don't specify the `-n` flag but just provide a number (e.g., `nice -5 command`), Linux interprets the `5` as the nice value. For negative values, you would type `nice --5`
    - No flags: If you run `nice` without any arguments or values, it simply displays the current niceness of the shell you are in (usually `0`).
    - Default behavior: If you run `nice command` without specifying a value, the system defaults to a niceness of 10.
- **renice** = If `nice` is how you set expectations before a process starts, **`renice`** is how you change the nice value for a process that's already running.
    - `-n [priority]`: (Optional but recommended for clarity) Specifies the new absolute nice value.
        - *Note:* Unlike `nice`, which adds to the current value, `renice` sets it to exactly what you type.
    - `-p [PID]`: The most common flag. It targets a specific Process ID. You can list multiple PIDs separated by spaces.
    - `-u [username/UID]`: Targets every single process owned by a specific user. This is a powerful "clean up" tool if a specific user is bogging down the system.

    - `-g [group/GID]`: Targets every process owned by a specific process group.
    - `-v`: Displays the version of the utility.

Shell Basics & Text Utilities

- **.bashrc** = a per-user script that runs every time a new interactive, non-login Bash shell is opened (e.g., opening a new terminal tab). It's the standard place to define aliases, shell functions, and prompt customizations (PS1) that you want in every session.
- **.bash_profile** = a per-user script that runs once, only for a login shell (e.g., an SSH session or a fresh console login). It typically sets environment variables and then sources `.bashrc` so interactive settings still apply.
- **.profile** = a POSIX-generic login-shell startup file, read by `sh` and by Bash when `.bash_profile` doesn't exist. It's the portable fallback used across shells other than Bash.
- **/etc/profile** = the system-wide login shell configuration file, read before any user's `.bash_profile` or `.profile`. Administrators use it to set baseline environment variables (like `PATH`) for every user on the system.
- **~** = (Home Directory): A shortcut representing the current user's HOME directory.
- / = (Root Directory): A shortcut representing the system root directory.
- **>** = overwrite, don't print the results to the screen. Instead, send them to a file instead as stdout. If that file exists already, overwrite it. (ex. ls /etc > list.txt)
- **>>** = append, don't print the results to the screen. Instead, send them to a file instead as stdout. If that file exists already, append this output to the end of the file (ex. date >> log.txt)
- **<** = redirect input. feeds the contents of a file into a command as if you had typed it as stdin. This is useful to feed config files into a cli utility.(ex. mysql -u user -p database_name < backup.sql)
- **<<** = the “Here-Document” Allows you to type multiple lines of input from a multi-line block directly into a script or command as stdin until a specific "end word" (usually `EOF` or `STOP`) is typed. (ex. cat << EOF > greeting.txt)
- **<<<** = the “Here-String”, pass a single string or variable as stdin directly into a command (ex. bc <<< "10 + 20")
- **|** = pipe, connects two commands so the output of the first becomes the input of the second (ex. cat /etc/passwd | grep "tux")
- **;** = command separator. Run the next command regardless of whether the previous one succeeded or failed.
- **&&** = Run the next command only if the previous one succeeded (Exit 0).

- **stdin** = Standard input is data fed into a command (can be what you typed or a file). Represented by 0
- **stdout** = Standard output is normal command output (can go to your screen or to a file); Represented by 1
- **stderr** = Standard error is where error messages go (usually your screen or a log file); Represented by 2
    - Breaking Down `2>&1` Think of this as a "merging" instruction for the shell.
        - `2`: Refers to Standard Error.
        - `>`: This is the redirection operator.
        - `&`: This tells the shell, "The next character is a File Descriptor number, not a filename." (Without the `&`, it would create a file literally named `1`). In this context, it is different then using the & operator in a command. Instead of moving a command to the background, & in this case tells the shell that the next character is a file descriptor number.
        - `1`: Refers to Standard Output.
- **!** = Bang command. (Logical NOT / History): In scripts, it represents "NOT" (logical negation). In an interactive shell, `!!` repeats the last command bang command, used for history expansion and repeating commands from the command history
    - !! = repeats the very last command you typed in
        - `!!:$` : The **last argument** of the previous command. (Highly useful!)
            - *Example*: `mkdir /var/www/html` followed by `cd !!:$`
        - `!!:^` : The **first argument** of the previous command.
        - `!!:*` : **All arguments** of the previous command (excludes the command itself).
        - `!!:n` : The *n-th* word of the previous command (where 0 is the command name).
    - !n = repeats command # n from the history list (ex. !452)
    - !-n = runs the command n # of steps back
    - !string = start match, runs the most recent command starting with the word “string”. You can use whatever starting word you want. (ex. !yum)
    - !?string? = contain match, runs the most recent command containing the word string (ex. !?conf? runs the latest command containing “conf”)
    - `:p` : **Print** the command but don't execute it. (Safe way to check what `!string` will do).

    - `:h` : **Head** (removes the trailing filename, leaving just the path).
    - `:t` : **Tail** (removes all leading path components, leaving just the filename).
    - `:r` : **Remove** the file extension.
- **history** = shows previously executed commands
    - -c = clears the history
    - -d [offset] = deletes a specific entry at a given position (offset)
    - -a = append the current session's history to the history file
    - -w = overwrites the current history list to the history file, overwriting its contents
    - -r = reads the history file and appends its content to the current session's history list
- **alias** = a string that is substituted for a command when it is typed into the shell. This allows you to create custom commands with options so you don't have to type it out every time.
    - alias name='command –options' : create an alias or update an existing alias
    - alias = lists all aliases
    - alias name = shows a specific alias
    - unalias name = remove an alias
    - unalias -a = removes all aliases
- **awk** = data extraction and text processing programming language (ex. awk 'pattern {action }' file). Unlike grep which just finds a matching criteria or cut which is used for simple character delimiters in well-structured files, awk reaches into complex lines within a dataset and grabs the piece of data you want.
    - -F = field separator. tells that columns are separated by something other than whitespace
        - $0 = the entire line
        - $1, $2, $n = first, second, or nth column
        - $NF = number of fields, last column in the line
        - NR = number of records, current line number
        - FS = field separator (input)
        - OFS = output field separator
    - -v = assign a variable, passes a shell value into the awk script
    - `BEGIN`: Executed *before* any lines are read (useful for headers or setting variables).
    - `END`: Executed *after* all lines are processed (useful for totals/summaries).

      ```
      awk 'BEGIN {print "STARTING"} {print $1} END {print "DONE"}' file.txt
      ```

- **cut** = used to extract specific simple sections (columns, fields, or character ranges) from each line of a file or piped input
    - -d = delimiter, defines the character used to separate fields (default is Tab) (ex. cut -d: /etc/passwd) < this changes to a colon instead of tab
    - -f = fields, specifies which fields to extract (ex. -f1 for the first column)
    - -c = characters, extracts based on character position rather than delimiters (ex. -c1 -5)
    - -b = bytes, extracts based on byte position
    - –compelent = displays all except the fields you specified (very useful)
- **bc** = basic calculator for precision floating point math
    - -l = standard math library. It loads the math library and sets default scale (decimal places) to 20
        - scale will need to be set or else the default is to not have decimals, so 3/2=1 instead of 1.5.
    - -q = quiet mode, doesn't display unnecessary info
- **nano** = simple, user-friendly text editor on Linux. Very useful for quick edits because it is modeless (you don't have to swap to insert mode to edit the text)
    - -w = disables hard-wrapping of long lines
    - -B = create a backup of the previous version of the file before saving
    - -v = opens the file in view/read-only mode
    - -c = constant cursor, always shows the cursor position (line and column number) at the bottom of the screen
    - internal navigation within the nano editor
        - `Ctrl+O` (Write Out): Saves the changes to the file.
        - `Ctrl+X` (Exit): Closes the editor. If you have unsaved changes, it will prompt you to save.
        - `Ctrl+K` (Cut Text): Deletes the entire current line (and stores it in the "cutbuffer").
        - `Ctrl+U` (Uncut Text): Pastes the line currently in the cutbuffer.
        - `Ctrl+W` (Where Is): Opens the search function to find a specific string.
        - `Ctrl+G` (Get Help): Opens the internal documentation.
- **vi / vim** = visual editor used to edit files. This is the powertool for text editing files in Linux. vim stands for vi improved. vim is the successor to vi. vim is modal, meaning the keys on your keyboard perform different functions depending on which "mode" you are in. The three modes are listed below
    - The three modes are listed below:
        - i = insert mode. This is the actual mode used to edit text in the file.
        - Esc = default mode. This mode is used for navigation, deleting, and copying

                - **`x`**: Deletes the character under the cursor.
                - **`dd`**: Deletes (cuts) the **entire current line**.
                - **`yy`**: "Yanks" (copies) the current line.
                - **`p`**: "Puts" (pastes) the copied or deleted text after the cursor.
                - **`u`**: **Undo** the last action.
                - **`/pattern`**: Searches forward for a specific string of text.
            - : = command line mode used for saving, quitting, and advanced functions
                - :w = saves changes to the file
                - :q = quits out of editor (fails if there are unsaved changes)
                - :wq = save and quit (use this most of the time)
                - :q! = force quit and discard changes
                - :x = same as :wq except it only writes if changes were made
    - -R = opens the file in read-only mode to present accidental changes
    - +n = go to line n
    - -r = recovers a file from a swap file after a system crash (very useful for recovering information)
    - -b = binary mode (used for editing non-text files)
- **screen** = session manager. One of the most useful commands in Linux. Detaches your running session from the terminal window and runs the session in the background. screen can give you session persistence which is a must-have for large tasks. It establishes tabs within a single shell so you can run multiple sessions at a time, allows multiple users to connect to the screen simultaneously, and works as a terminal emulator. It is a powerhouse of a command. Syntax: screen [options] [program]
    - -S <session_name> = Create a Named Session: Instead of letting the utility randomly assign a cryptic numeric Process ID (PID) to your session, this forces a custom, human-readable identifier. Always use this to keep your terminal workspaces organized. *Example:* screen -S db-migration
    - `-ls` (or `-list`) = List Running Sessions: Scans the local system socket directory and prints a clear table of all active background screen instances, their PIDs, creation timestamps, and active connection statuses (`Attached` or `Detached`). *Example:* `screen -ls`
    - -r [PID.session_name] = Reattach to a Session: Pulls a backgrounded, detached session back into your foreground terminal display view. If you only have one session running, running screen -r without naming it will auto-connect. *Example:* screen -r db-migration
    - `-x` = Multi-Display Connection: Attaches to a session that is already attached somewhere else. Instead of hijacking or blocking the original

viewer, it mirrors the output to both terminals simultaneously. This is ideal for dual-admin troubleshooting or training environments. *Example:* `screen -x training-session`

- `-d -r` = Force Detach and Reattach: If your SSH connection drops violently, your session might get frozen in an `Attached` state. If you try to run standard `screen -r`, the utility will refuse to connect, claiming the terminal is already occupied. Using `-d -r` tells the host to forcefully kick out the phantom connection and instantly bridge the screen back to you. *Example:* `screen -d -r 12345.db-migration`
- `-d -m` = Start in Detached Mode: Spins up a completely brand-new screen session, initializes your designated background command, but remains in the background without switching your current terminal display context. This is highly useful for daemonizing shell scripts or automating background workers. *Example:* `screen -d -m -S monitor-script ./run_monitoring.sh`
- `-R` = Reattach or Build First Match: Attempts to locate and reattach to the first detached session it finds. If no background sessions are found running on the OS, it dynamically switches gears and creates a fresh session for you on the fly. *Example:* `screen -R`
- -L = Turn on Automatic Output Logging: Instructs screen to record every single string of standard output (stdout) and errors generated inside the virtual window and stream it directly into a persistent capture file on disk (usually named screenlog.0). *Example:* screen -L -S critical-update ./apply_patches.sh
- -h <number> = Adjust Scrollback History Buffer: Dictates the explicit number of text lines saved in your scrollback cache history buffer. Increasing this value gives you a deeper buffer window when scrolling backward through massive compilation trackers. *Example:* screen -h 5000
- -c <filename> = Specify Custom Config File: Overrides the system default lookups for the global setup or user configurations (~/.screenrc) and initializes the environment using the parameters written inside a targeted plain-text configuration file. *Example:* screen -c /etc/custom_screen_profile
- -v (or --version) = Display Version Information: Prints the version string and compilation metadata of the local screen binary tool suite.
- Shortcuts you need to know:
  - `Ctrl + A` then `d` — Detach Session (Most useful) Disconnects your active terminal from the screen environment, backgrounding everything safely. This is the exact shortcut you use before logging

out of a server to ensure your backup or update script finishes running.
        - `Ctrl + A` then `c` — Create Window Spawns a brand-new virtual shell.
        - `Ctrl + A` then `n` / `p` — Next / Previous Window Cycles forward (`n`) or backward (`p`) through your open virtual shells.
        - `Ctrl + A` then `[` — Scrollback / Copy Mode Freezes the live terminal output so you can use your arrow keys or `Page Up`/`Page Down` to read through logs that scrolled off the top of the screen. (You press `Esc` to exit this mode).
        - `Ctrl + A` then `"` — Window List Pops up an interactive visual menu showing the index and names of all active shells so you can hop directly to a specific one.
- **cat** = concatenate files to screen OR join files together (ex. cat file1.txt file2.txt > combined_file.txt)
    - -A = showall
    - -n = number the output lines
    - -s = squeeze multiple consecutive empty lines in a a single empty line
    - -E = displays a $ at the end of every line
    - -v = show invisible characters
    - > = create or overwrite (ex. echo “hello” > test1.txt)
    - >> = append to a file (ex. echo “hello again” >> test1.txt)
- **pr** = print output to screen in printable format
    - -n = files in column format in n # of columns
    - -l n = change default 66 line page length to n # of lines
    - -m = display multiple files in parallel columns
    - -t = don’t display headers or trailers
    - -w n = change 72 character width to n
- **head** = displays first 10 lines of a file
    - -n 3 = displays the first 3 lines in a file (ex. head -n 5 test.txt)
        - -n -3 = display lines except for the first 3
    - -c 3 = displays the first 3 bytes of the file rather than lines
    - -q = quiet never prints filenames when processing multiple files
    - -v = verbose
- **tail** = display last 10 lines of a file
    - -f = follow the output (useful for seeing logs in real time)
    - -F = follow and keep trying to open the file if it is moved or rotated
    - -c 3 = outputs the last 3 bytes of a file rather than the lines
    - -n 3 = displays the last 3 of lines in a file

    - -n -3 = displays lines except the last 3 lines
- **more** = read any files in a semi-interactive way. Cannot scroll backwards through files (replaced by less command) (like nano but can't edit)
    - -n # = number of lines at a time to show
    - -d = display help prompts to show at the bottom of the screen
    - -p = clear the screen before displaying the next page of text
    - -s = squeeze multiples consecutive blank lines to maximize screen space
    - +n = start displaying the file at line n
    - For internal navigation, here are the options you have
        - `Spacebar`: Scroll forward one full screen.
        - `Enter`: Scroll forward one line at a time.
        - `f`: Scroll forward one window (similar to the spacebar).
        - `q`: Quit the pager immediately.
        - `/pattern`: Search forward for a specific string.
        - `h`: Display the help screen for all internal commands.
- **less** = read any files in a semi-interactive way (can do way more than more command) (like nano but can't edit)
    - -N = display line numbers
    - -S = chop long lines rather than wrap them
    - -i = case insensitive when searching for keywords
    - -X = prevents less from clearing the screen when exiting less
    - For internal navigation, here are the options you have
        - q = exit
        - b = move backward one window
        - g = jump to beginning of the file
        - G = jump to the end of the file
        - /pattern = search forward for a specific text
        - ?pattern = search backward for a specific string
        - n = repeat the previous search (forward)
        - N = repeat the previous search (backward)
        - spacebar = go forward a page
        - esc+v = go back a page
        - arrow keys = scroll up and down through file
        - F = forward to the end of the file. Useful for viewing logs
        - 4 g = go to the line #4
- **man** = help manual on certain commands (ex. man grep) (uses less to interact with the manual)
- **which** = finding the file path where a utility resides on the system
    - a = all location where the app is installed (if there are multiple installations)
- **whereis** = finding the file path AND the source code file for a system utility

- **grep** = finds a pattern match from a file (grep -ir “root” /etc/passwd)
    - -i = case insensitive
    - -r = recursive, search subdirectories (USE THIS MOST OF THE TIME)
    - -v = invert match
    - -n = show line numbers
    - -c = count number of matches
    - -l = show filenames only
    - -w = exact word match, this is default
    - -A = show lines after match
    - -E = extended regex, allows use of extended regular expressions
    - -d skip = skip directories. Only search for files
    - ^ = matches the beginning of a line (ex. “^Error”)
    - $ = matches the end of a line (ex. “Success$”)
    - . = matches any single character (ex. “gr.y”)
- **echo** = simple way to write arguments or display a line of text to stdout. You'll see it used primarily for variable expansion, verifying shell environment configurations, and basic scripting automation.
        - echo $USER = displays current logged on user
        - echo $PATH = displays the directories the shell searches for commands
        - echo $SHELL = displays the path to the current shell
        - echo $? = displays the exit status of the last command (0=success)
    - -e = interprets backslash-escape characters like \n or \t
        - `\n`: **New line** (moves text to the next line).
        - `\t`: **Horizontal tab** (indents text).
        - `\\`: **Backslash** (displays a literal backslash).
    - -n = prevents the cursor from jumping to a new line
    - echo $VAR = interprets the variable
    - echo “$VAR” = interprets the variable
    - echo ‘$VAR’ = interprets as literal text
- **printf** = power tool for formatting and printing data to stdout. Offers precise control over how text, numbers, and variables are aligned and displayed (ex. printf “format_string” [arguments])
    - %s = string format
    - %d or %i = integer format
    - %f = floating-point decimal format
    - %x = hex format
    - \n = moves to a new line (essential because printf won’t move to the next line without it)
    - \t = horizontal tab used to align columns of data

    - \\ = literal backslash
- **tee** = shell utility that reads from stdin and writes to BOTH stdout and one or more files simultaneously
    - -a = appends the output to the end of a specific file
    - -i = ignore interrupts during execution
    - -p = provides diagnostic when an error occurs writing to non-pipes
- **tr** = translate characters from stdin and write the results in stdout. Unlike sed or awk, tr CANNOT read directly from a file. it must receive data via a pipe (|) or redirection (<) (ex. echo “linuxplus” | tr ‘a-z’ ‘A-Z’
    - -d = delete the characters specified in the first set form the input
    - -s = squeeze. replaces sequences of repeated characters with a single occurrence of that character (consolidating blank lines to one blank line)
    - -c = complement (opposite) of the first set (everything except what you specify)
    - -t = truncate (shortening by cutting off the end) the first set to the length of the second set before translating
- **uname** = unix name, used to print detailed info about the system, kernel, and hardware architecture
    - -a = prints all available system info in a single line
    - -s = prints the name of the kernel (ex. Linux)
    - -n = nodename, prints the network node hostname
    - -r = prints the kernel release version
    - -v - prints kernel version and build date info
    - -m = machine hardware architecture (ex. x86_64)
    - -p = prints processor type
    - -o = prints OS name
- **sed** = stream editor. Transforming and substituting text. Takes an input stream (a file or piped output), performs operations on it based on your instructions, and spits out the result. Essentially it finds and replaces text
    - -i = in-pace, saves the changes to the file. By default, `sed` prints the result to the screen. `-i` actually saves the changes to the file.
    - -n = quiet/silent mode
    - -e = specify multiple commands in one go (e.g., `sed -e 's/a/b/' -e 's/c/d/'`).
    - -f = file, tells
    - -E or -r = extended regular expressions (ERE) without needing to escape every special character
- **sort** = sort lines of a text file or piped input in a specific order (alphabetic, numerical, etc)
    - -n = numeric sorting

    - -r = reverse order sorting
    - -k = specifies a key column to sort by (ex. -k3)
    - -t = field delimiter character (default is whitespace)
    - -u = unique, remove duplicate lines from the output (like using sort | uniq)
    - -M - sorts by three-letter month abbreviations (JAN, FEB)
    - -h = human readable (ex. 2K, 2G, 5M)
- **uniq** = filter out, remove, or report adjacent duplicate lines in a file or stream
    - -c = counts the # of lines, used for frequency analysis (ex. counting failed login attempts)
    - -d = print only duplicate lines
    - -u = only prints lines that are not duplicated/repeated
    - -i = case insensitive
    - -f [n] = skip fields, avoid comparing the first n # of fields
- **wc** = word count, counts lines, characters, bytes, and words in a file or from piped input
    - -l = lines
    - -w = words
    - -c = bytes
    - -m = characters
    - -c = max line length
- **xargs** = takes a stream of text (like a list of filenames from a pipe) and turns them into arguments for another command that wouldn't normally accept piped data. (Useful for pushing data to commands that will not accept it through a pipe)
    - -0 = null, input items are terminated by a null character instead of whitespace
    - -p = ask the user for confirmation before executing each command
    - -t = verbose. prints the command to stderr before executing it
    - -n [number] = max arguments passed to a single command execution
    - -I [string] = replace, defines a placeholder (often {}) to specify exactly where the argument should go
- **source** = reads and executes commands from a file in the current shell environment. source runs in the current shell, only requires read access and variables and aliases stay in memory once the script exits. source can also be used to force refresh an update for certain files. (ex. source ~/.bashrc)
    - The period (.) symbol is a shorthand form for the source command
    - ./ is slightly different in that it runs in a new subshell, requires (+x) permission and the variables are lost once the script exits. (ex. .~/.bashrc)

Package Management

- **Sandboxed Applications (Flatpak / Snap)** = package formats that bundle an application together with all of its own dependencies and run it in an isolated sandbox, independent of the host distro's package versions.
  - **`flatpak`** = install/run sandboxed desktop apps distributed via remotes like Flathub. (ex. `flatpak install flathub org.videolan.VLC`)
  - **`snap`** = Canonical's competing sandboxed package format, managed by the `snapd` daemon. (ex. `snap install code --classic`)
- **dnf/yum** = high-level pkg manager for RPM-based Linux distros. dnf will install dependencies for packages
  - install = install certain program
  - remove = uninstall package and dependencies. Useful for uninstalling services you don't need or unsecure services like telnet (ex. sudo dnf remove telnet-server)
  - update = updates packages to the latest version
  - check-update = lists available updates (look but don't touch)
  - makecache = download and cache metadata for all currently enabled repositories. It forces dnf to pull the most up-to-date repo updates.
  - search = looks for packages matching a keyword
  - list = Lists packages (e.g., `dnf list installed` or `dnf list available`).
  - provides = find. Tells which pkg contains a specific file
  - -y = automate yes
  - history = list of past dnf activity
  - repolist = shows which software repositories are currently enabled
  - `--enablerepo` / `--disablerepo`: Temporarily turns a repository on or off for a single command.
  - `clean all` = Removes all cached package data and metadata. Use this if `dnf` is behaving strangely or reporting "outdated" information.
  - -x/--exclude=<package name> = bypass an update for a package that you don't want upgraded
  - versionlock = keeps a package from being updated with dnf update
    - add = lock a package to its current version
    - delete = removes a lock
    - clear = clears all locks
  - system-upgrade = Used to upgrade the version of Linux. (Only works for Fedora. For RHEL, use leapp) (ex. sudo dnf system-upgrade download -y --releasever=44 --allowerasing && sudo dnf system-upgrade reboot)
  - dnf-automatic = setting up automatic updates using the systemd timer.
    - /etc/dnf/automatic.conf = configure the update details

        - dnf-automatic.timer = timer that is controlled by the automatic.conf file
- **leapp** = official Red Hat framework used for performing **in-place upgrades** of RHEL (Red Hat Enterprise Linux) systems between major versions (e.g., RHEL 8 to RHEL 9)
    - `preupgrade =` Performs a system analysis and reports issues that would prevent a successful upgrade. This is a mandatory first step in the RHEL upgrade lifecycle.
    - `upgrade =` Initiates the actual upgrade process after all inhibitors from the `preupgrade` phase are resolved.
    - `list-runs =` Displays the history of previous Leapp executions.
    - `help =` Provides documentation for the utility or specific sub-commands.
    - `--target [version]` (ex. `--target 9.4 ) =` Explicitly defines the major/minor version of RHEL you wish to move toward.
    - `--reboot =` Automatically reboots the system once the initial package preparation and initramfs creation are complete. Useful for automation or when you want the process to transition to the actual OS swap immediately.
    - `--no-rhsm =` Tells Leapp not to interact with the Red Hat Subscription Manager. Essential for environments using air-gapped repositories, local mirrors, or Red Hat Satellite.
    - `--enablerepo [repo_id] =` Temporarily enables a specific repository during the upgrade process.
    - `--debug =` Increases the verbosity of the output to provide detailed internal steps. Used for troubleshooting failures during the `preupgrade` or package download stages.
    - `--answerfile [path] =` Points to a specific location for the answerfile if not using the default (`/var/log/leapp/answerfile`).
- **rpm** = red hat package manager (low level package manager that doesn't install dependencies)
    - `-i` (Install): Used to install a new package file.
    - `-U` (Upgrade): Upgrades an existing package or installs it if it's missing.
    - `-F` (Freshen): Upgrades a package only if a previous version is already installed.
    - `-e` (Erase): Uninstalls a package from the system.
    - `-v` (Verbose): Provides a detailed output of what the command is doing.
    - `-h` (Hash): Displays progress using hash marks (`#`)—often paired as `-ivh`.

    - `-qa`: Lists all currently installed packages.
    - `-qi <package>`: Shows detailed information, including the version, description, and architecture.
    - `-ql <package>`: Lists every file installed by that package.
    - `-qc <package>`: Lists only the configuration files.
    - `-qf <file_path>`: Tells you which package owns a specific file (e.g., `rpm -qf /bin/ls`).
    - `-qp <rpm_file>`: Queries a package file that has not been installed yet (requires the full path to the `.rpm` file).
    - `-V <package>`: Verifies a package by comparing its current state on disk to the metadata in the RPM database. It checks for changes in size, checksums, permissions, and ownership.
    - `-Va`: Verifies all installed packages. This is a common security auditing technique to find modified system binaries.
    - `--import <key>`: Imports a GPG key for Signed package verification, ensuring the authenticity of the software source
- **apt** = advanced package tool (high level package manager used in all debian based Linux distros
    - update = refresh the list of latest package versions. Must be run before installing or upgrading
    - install = install specific pkg
    - remove = uninstall pkg.
    - purge = removes pkg and all config files. Useful for uninstalling services you don't need or unsecure services like telnet (ex. sudo apt purge telnetd)
    - upgrade = update packages
    - full-upgrade = performs upgrade but can also remove or install new pkgs if necessary to resolve dependency mismatches
    - show = detailed information about a pkg
    - autoremove = removes pkgs that were installed but are no longer needed
    - -y = automate yes
    - -q = quiet output omitting progress bar
    - -s = simulation/dry-run
- **apt-mark** = pinning a package in debian, or stopping or starting a package from being included in app update && apt upgrade
    - showhold = see whats currently being held
    - hold/unhold <package name> = holds or unholds a package from being updated

- **do-release-upgrade** = high-level utility used primarily on Debian-based systems (specifically Ubuntu) to manage the transition from one distribution release to the next (e.g., upgrading from Ubuntu 22.04 LTS to 24.04 LTS).
    - `-c` / `--check-dist-upgrade-only =` Checks if a new release is available and exits without performing the upgrade. Used for system auditing and verifying update paths without making changes to the system.
    - `-d` / `--devel-release =` Upgrades to the latest development release.Critical for testers or when you need to force an upgrade to a version not yet marked as "stable" for your current track.
    - `-m [mode]` / `--mode=[mode] =` Specifies the mode of the upgrade (e.g., `desktop` or `server`). Important for ensuring the correct package set (GUI vs. Headless) is prioritized during the transition.
    - `-f [frontend]` / `--frontend=[frontend] =` Specifies the frontend to use (e.g., `DistUpgradeViewText`). Used in automation or scripts to force a specific user interface, such as a non-interactive text mode for remote management.
    - `-p` / `--proposed =` Attempts to upgrade using the latest release from the `ubuntu-proposed` repository. Used for testing patches and pre-release software in a staging environment.
- **dpkg** = debian package manager (low level package manager that doesn't install dependencies)
    - -i = installs .deb pkg file
    - -r = uninstalls pkg but keeps config files on the system
    - -P = removes a pkg and all config files
    - -c = lists content contained withing a .deb file that hasn't been installed yet
    - –configure = reconfigures an unpacked pkg. fix broken installs
    - -l (List): Lists all packages matching a pattern. If no pattern is given, it lists every package known to the system.
    - -L (List Files): Shows every file installed on your system by a specific package (e.g., dpkg -L coreutils).
    - -s (Status): Reports the current status of a package (e.g., Installed, Unpacked, or Not-installed).
    - -S (Search): Finds which package "owns" a specific file on your disk (e.g., dpkg -S /bin/ls).
    - -V = checksum verification
- **pip** = Package manager for Python. (Pip installs Packages) Unlike system package managers (`dnf` or `apt`) that manage binaries for the entire operating system, `pip` specifically fetches, installs, updates, and removes code libraries

and modules from the online Python Package Index (PyPI). WARNING: You should **avoid running `pip install` as root or with `sudo`** unless absolutely necessary. Many modern Linux system utilities (such as Red Hat's `dnf` package manager) are written in Python. If you use `sudo pip install`, you risk overwriting a global Python library that the operating system relies on, which can inadvertently break core OS tools.

    - `install <package>`: Downloads and installs a specific Python module from PyPI.
    - `uninstall <package>`: Safely removes an installed Python module and its associated files from the environment.
    - `list`: Displays all currently installed Python packages along with their exact version numbers in a clean columns format.
    - `show <package>`: Outputs detailed metadata about a specific installed package, including its author, version, summary, home directory, and critical dependencies.
    - `freeze`: Outputs all installed packages in the standard `package==version` format. This is heavily used by Linux administrators to clone environments by redirecting the output to a file (e.g., `pip freeze > requirements.txt`).
    - `-r <file>` (Requirements Flag): Paired with the install command (`pip install -r requirements.txt`) to automatically install a massive batch of dependencies listed inside a text file.
    - `--upgrade` (or `-U`): Instructs `pip` to update a specified package (and its dependencies) to the latest available version on PyPI.
    - `--user`: Installs the Python package strictly within the current user's local home directory (`~/.local/bin` and `~/.local/lib`) rather than the global system space.
- **cargo** = the official package manager, artifact repository pipeline, and compilation build system for the **Rust** programming language.
    - `new <project_name>`: Creates a brand new Rust project directory containing a default template and a boilerplate `Cargo.toml` configuration file.
    - `build`: Downloads all necessary dependencies and compiles the local project files into a non-optimized, debugging-friendly binary.
    - `run`: A combined shorthand command that compiles the project code (if changes are detected) and immediately executes the resulting binary file.

    - `check`: Performs a rapid scan of the source code to ensure it complies with syntax constraints and can compile, but completely skips the actual binary code generation step to save time during development.
    - `test`: Automatically discovers and executes any unit tests or integration tests defined within the codebase.
    - `update`: Scans remote repositories to find newer versions of your project dependencies, updating the local configuration layout accordingly.
    - `--release`: A critical compilation optimization flag used alongside other commands (such as `cargo build --release`). It strips debugging symbols and configures the compiler to fully optimize the binary for maximum performance in a production deployment environment.
- **npm** = node (node.js) package manager. Default package manager for the JavaScript runtime environment.
    - `init`: Initializes a brand new JavaScript project, prompting you for metadata to create the foundational tracking file known as `package.json`.
    - `install <package>` (or `npm i`): Downloads and installs a specified package into a local subdirectory called `node_modules/` inside your active project directory. If run by itself without naming a specific package, it reads the local `package.json` file and automatically downloads *every* dependency specified for that project.
    - `uninstall <package>`: Cleanly deletes an installed package and purges its execution footprint from both the `node_modules/` folder and the tracking logs.
    - `update`: Queries the registry to scan for newer versions of your project's active packages, updating them within the safety parameters specified by your project manifest.
    - `list` (or `npm ls`): Generates a tree-like hierarchy on your standard output displaying all locally installed modules along with their specific versions.
    - `-g` (Global Flag): Forces `npm` to install a package globally across the system (typically to paths like `/usr/lib/node_modules/`) rather than locally inside the current folder. This flag is heavily used by Linux administrators to deploy executable command-line server tools system-wide.
    - `--save-dev` (or `-D`): Installs a package specifically as a "development dependency" (like a testing framework or a code linter). This marks it

inside the configuration file as a utility that is only needed for coding, allowing production environments to skip downloading it to save space.

- **update-alternatives=** system utility that creates, removes, and maintains a structured directory of symbolic links, allowing an administrator to safely establish a default system-wide command version while keeping other versions fully accessible.
    - **`--config <name>`**: Launches an interactive menu on the standard output listing all available paths that can provide the specific master command `<name>`. It displays their priority levels and lets you input a selection number to dynamically change the system default.
    - **`--display <name>`**: Outputs the comprehensive current state of the specified alternative group. It reveals whether the group is in automatic or manual mode, shows exactly which path the master link currently points to, and lists all alternative paths alongside their priority metrics.
    - **`--install <link> <name> <path> <priority>`**: Adds a brand-new alternative group or appends a new path option to an existing group.
        - `<link>` specifies the master symlink location (e.g., `/usr/bin/editor`).
        - `<name>` defines the unique master name in the alternatives directory (e.g., `editor`).
        - `<path>` points to the new physical executable binary (e.g., `/usr/bin/vim`).
        - `<priority>` is an integer score; in automatic mode, the option with the highest priority is automatically selected as the system default.
    - **`--remove <name> <path>`**: Removes a targeted executable path option from the specified alternative master group. If the path being removed is the current active system default, the utility automatically falls back to the path with the next highest priority score.
    - **`--auto <name>`**: Restores the specified alternative group back to automatic mode, immediately giving the system permission to switch the current active link to whatever binary carries the highest priority integer.

systemd & Service Control

- **systemctl** = system controller. It inspects and controls systemd units and services. It is the primary system manager used to control processes, query service units, and switch targets. `systemd` (system daemon) is the init system

that initializes the OS and assumes Process ID 1 (PID 1). Because `systemctl` is the direct command-line interface to PID 1, it has unparalleled control over the operating system. This is arguably the most important and most-used command in Linux. If you do not specify an extension, `systemctl` assumes you mean a `.service`. For everything else, you **must** append the unit type extension.

- **start** = Immediately starts a unit (not persistent after reboot).
- `stop =` Immediately stops a running unit.
- `restart =` Stops and then starts a unit; used after configuration changes. (ex. systemctl restart chronyd)
- `reload =` Tells a service to re-read its config file without stopping the process.
- `status =` Shows if a unit is active/running and provides the last few log entries.
- `enable =` Configures a unit to start automatically at boot.
- `disable =` Prevents a unit from starting automatically at boot. Highly useful for disabling unsecure services like telnet or ftp. (ex. sudo systemctl disable telnet.socket)
- `is-active =` Returns a simple "active" or "inactive" status (great for scripting).
- mask = "stronger" than `disable`. It links the unit file to `/dev/null`, making it impossible to start the service manually or via dependencies.
- `unmask =` restores the service's ability to be started.
- `daemon-reload` = You must run this after manually editing a unit file (e.g., in `/etc/systemd/system/`) so systemd recognizes the changes.
- `edit` = This command opens a text editor to create a "drop-in" snippet for a unit, allowing you to override specific settings without modifying the original vendor-supplied file.
- `list-units` = Used to view all units currently loaded in memory. You can filter by type, such as `--type=service` or `--type=timer` .
- reboot = restarts the system
- poweroff = shuts down the system
- suspend = puts the system in a low-power sleep mode
- Unit Types: Everything systemd manages is defined in configuration files called Units
  - Services (`.service`) = Manage standard background daemons and applications.

        - Use standard state triggers (`start`, `stop`, `restart`, `reload`, `status`, `enable`, `disable`). Example: `systemctl restart sshd.service`
- Timers (`.timer`) = Replace legacy cron jobs by triggering events based on time or calendar schedules.
    - You start, stop, or enable timers just like services. To see all active timers on the system, use the dedicated command: `systemctl list-timers`.
    - Example: `systemctl enable backup.timer`
    - OnCalendar = – Used for real-time/calendar events (e.g., daily, weekly, or *-*-* 00:00:00).
    - OnBootSec = – Used for monotonic events relative to when the machine turned on (e.g., 15min).
    - Unit = – Specifies the exact execution service to trigger.
    - Persistent = – Set to true or false. Determines if a missed execution (because the server was turned off) should fire immediately upon the next boot.
- Mounts (`.mount`) = Control filesystem mount points via systemd. You can also use the mount and umount commands to do this same thing.
    - The Critical Catch: The name of the unit file must exactly match the physical path of the mount directory, replacing slashes with hyphens. If you want to manage the mount point `/mnt/data/backup`, the unit file *must* be named `mnt-data-backup.mount`.
    - How to interact:
        - `systemctl start mnt-data.mount` (Mounts the filesystem instantly)
        - `systemctl stop mnt-data.mount` (Unmounts the filesystem)
- Targets (`.target`) = Group other units together to define operating system states (replacing traditional SysV runlevels). Instead of using `start`, you use the `isolate` subcommand to switch the entire operating system into a specific target mode.
    - Example: `systemctl isolate multi-user.target` (Switches the server to non-graphical CLI mode).
- Sockets (`.socket`) = Intercept network traffic or IPC channels. Sockets facilitate "socket activation," meaning systemd will listen on

a port (e.g., port 22) and keep the associated service (SSH) completely turned off until a packet arrives, saving system memory. You manage them like services, but stopping a service while its socket is still running means incoming traffic will just wake the service right back up.
        - Example: `systemctl stop sshd.socket` (Stops systemd from listening on the network port entirely).
    - Devices (`.device`) = Expose hardware devices (tracked by the kernel and `udev`) as systemd units. You cannot start, stop, or edit a device unit because hardware is controlled by physical presence and drivers. However, you use them to check hardware status or create dependencies (e.g., telling a service not to start until a specific network card device is active).
        - Example: `systemctl status dev-sda1.device`
    - Paths (`.path`) = Monitor specific files or directories for real-time changes (like file modification, creation, or deletion) and trigger a corresponding service when a change occurs.
        - How to interact: * `systemctl start watch-config.path`
        - Editing context: When editing a `.path` unit, you define tracking parameters like `PathModified=/etc/nginx/nginx.conf` to declare exactly what target path to monitor.
    - Slices (`.slice`) = Group processes together into resource-management buckets using Linux kernel Control Groups (`cgroups`). This allows you to restrict CPU, memory, and I/O utilization for a whole group of applications at once. You monitor them using `status` to view resource consumption hierarchies.
        - Example: `systemctl status user.slice` (Shows all resources used by active logged-in users).
        - Editing context: To restrict resource allocations, you run `systemctl edit system.slice` and append resource boundaries under the `[Slice]` configuration block.
- **systemd-analyze** = audit boot performance and startup issues. Reads performance metrics directly from systemd
    - `time` (or bare command) = This is the default action if you run the command with no arguments. It prints a single-line summary breaking down the global boot timeline into distinct execution phases.

        - Syntax: `systemd-analyze time`
    - `blame =` Generates an ordered list of every active systemd unit initialized during the boot process, sorted highest to lowest by total initialization time. This is the ultimate tool for optimizing performance. It reveals which specific background application or configuration script is dragging down system startup times.
        - Syntax: `systemd-analyze blame`
    - `critical-chain =` Outputs a text-based dependency tree mapping out the "time-critical chain" of services. It isolates only the services that directly delayed the initialization of the final boot target, showing you exactly which dependent service had to finish before the next could start.
        - Syntax: `systemd-analyze critical-chain`
    - `plot =` Generates an incredibly detailed, vector-based graphical timeline chart of the boot cycle and outputs it as raw SVG code. You can pipe this output straight into an `.svg` file and open it in a web browser to inspect visual bars representing when each service started and stopped loading.
        - Syntax: `systemd-analyze plot > boot_profile.svg`
    - `verify =` Scans your unit file directories (like `/etc/systemd/system/`) and checks for syntax errors, missing dependency links, or typos without actually executing or starting the services. This is highly useful for validation before deploying updates
        - Syntax: `systemd-analyze verify /etc/systemd/system/my-script.service`
- **systemd-cryptenroll** = automatically enter the drive decryption key during boot so you don't have to type it manually (ex. sudo systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=0+7 /dev/nvme0n1p3)
- **hostnamectl** = systemd utility used to query and change the system hostname and related settings. `hostnamectl` lets you safely modify how the local machine identifies itself on the local network and to the kernel.
    - `--static` = This is the traditional Linux hostname. It is stored permanently in the `/etc/hostname` file and is loaded automatically every time the system boots.
    - `--transient` = This is a temporary, runtime hostname maintained strictly by the Linux kernel. If no static hostname is assigned, network services like DHCP or mDNS can dynamically overwrite the transient hostname. It is lost completely upon reboot.
    - `--pretty` = A high-level, human-readable UTF-8 hostname string used for presentation to users (e.g., `Linux+ Test Lab Server #4`). Unlike

static/transient names, it allows spaces, special punctuation, and capital letters. It is saved inside `/etc/machine-info`.
    - `status` – (Default) Dumps a detailed overview of the system identity profile, including all three hostname values, the machine ID, boot ID, virtualization environment (if applicable), operating system kernel architecture, and version string.
    - `set-hostname [NEW_NAME]` – Changes the system's hostname to the specified string.
- **`resolvectl`** = viewing and modifying DNS settings on systems using the `systemd-resolved` daemon.
    - `status` = Shows the global and per-interface DNS settings (current nameservers, domain routing).
    - `query <domain>` = Resolves a domain name (A, AAAA records) utilizing the systemd-resolved stack.
    - `dns <interface> <ip>` = Temporarily sets the DNS server for a specific network interface.
    - `domain <interface> <domain>` = Sets the search domains for a specific interface.
    - `flush-caches` = Clears the local DNS cache (highly useful for troubleshooting stale records).
    - **systemd-resolved** = Modern replacement for /etc/resolv.conf. Manages DNS resolution. On many modern distributions (like Ubuntu or Fedora), `/etc/resolv.conf` is no longer managed manually. In new systems, `/etc/resolv.conf` is often a symbolic link pointing to a file managed by `systemd-resolved`.
- **sysctl** = changes kernel configs. It examines and modifies linux kernel parameters at runtime. While systemctl manages services, timers, mounts etc, sysctl adjusts system resource limits, networking behavior, and security hardening. Contrary to popular belief, sysctl has absolutely nothing to do with systemd and was made before systemd.
    - `-a`, `--all` Dumps every single kernel variable currently available in the system alongside its active value. Because it outputs thousands of lines, it is almost always paired with a pipeline filter (e.g., `sysctl -a | grep net`).
    - `-w`, `--write` Used to write/modify a kernel parameter dynamically in system memory. Requires root/sudo privileges and the format `variable=value` (e.g., `sysctl -w net.ipv4.ip_forward=1`).

    - `-p[FILE]`, `--load[=FILE]` Loads and applies kernel settings from a configuration file. If you do not specify a file path, it defaults to reading the main system configuration file at `/etc/sysctl.conf`.
    - `-n`, `--values` Suppresses the printing of the variable name in the output, returning only the actual value assigned to it. This is highly useful for writing clean automation and bash shell scripts.
    - `-N`, `--names` The direct inverse of `-n`. It prints only the names of the keys/variables and hides their assigned values.
    - `-r`, `--pattern` Allows you to filter variables using a regular expression (regex). Instead of dumping everything, it only evaluates keys that match your string pattern.
    - `-e`, `--ignore` Instructs the command to gracefully ignore errors regarding unknown or unrecognized keys. This is helpful if you are deploying a single configuration script across multiple different Linux kernel versions where some keys might not exist.
    - `-b`, `--binary` Prints the designated value in a raw binary format without appending a trailing newline character.
    - `-q`, `--quiet` Suppresses standard output entirely. When you modify a variable using `-w`, `sysctl` normally echoes the change back to your screen; `-q` silences this behavior.
    - `-V`, `--version` Displays the current version of the `sysctl` binary utility being used by the operating system.
    - `-h`, `--help` Prints a concise help menu detailing standard usage parameters and exit statuses.
- **`timedatectl` =** Modern Linux distributions utilize `timedatectl` (part of the `systemd` ecosystem) as a centralized frontend tool to check and alter the system clock, time zone, and NTP status.
    - **`status`** – (Default) Displays current local time, universal time (UTC), RTC (Real-Time Clock), time zone, and whether NTP synchronization is active.
    - **`set-ntp [true|false]`** – Enables or disables network time synchronization. Turning this on activates the underlying NTP client (like `systemd-timesyncd` or `chronyd`).
    - **`set-time [YYYY-MM-DD HH:MM:SS]`** – Manually adjusts the system clock. *Note: This will fail if NTP synchronization is currently enabled.*
    - **`set-timezone [Zone/Subzone]`** – Changes the system's active time zone (e.g., `America/New_York`).

    - `list-timezones` – Lists all available time zones valid for the system configuration.
- **reboot** = restart immediately
    - -f = force
    - -p = poweroff (same as shutdown command)
- **shutdown** = professional, detailed way to shutdown a Linux system
    - `-h` **(Halt)**: Stops the OS but might not cut hardware power depending on the BIOS/UEFI.
    - `-r` **(Reboot)**: Restarts the system instead of powering off.
    - `-P` **(Poweroff)**: Explicitly powers off the machine (default behavior in many modern distros).
    - `-c` **(Cancel)**: Aborts a pending scheduled shutdown.
    - `+<n>` = delay shutting down by n number of seconds

Networking Basics

- **/etc/nsswitch.conf** = (Name Service Switch) configuration file that tells the system which sources to check, and in what order, when resolving things like hostnames, users, and groups. For example, a `hosts:` line of `files dns` means check `/etc/hosts` first, then fall back to DNS.
- **nm-connection-editor** = lightweight gui-based network config tool for quickly establishing connections or handling credentials for secure networks (wifi or VPN) (ex. nmconnect [OPTIONS] <connection_name | UUID>)
    - `--create`: Launches the editor directly into the "New Connection" wizard. This is useful when you need to quickly add a **Bridge**, **Bond**, or **VLAN** without navigating the main menu.
    - `--edit <UUID or Name>`: Opens the configuration window for a specific, existing network profile. You can find the UUID using `nmcli connection show`.
    - `--show <UUID or Name>`: Displays the details of a connection in a read-only window.
    - `--type <type>`: Filters the "New Connection" list to a specific technology (e.g., `ethernet`, `wifi`, `vpn`).
    - `--help`: Displays all available command-line arguments for the tool.
- **arp** = address resolution protocol, allows you to view and manipulate the ARP cache which maps IP addresses to MAC addresses
    - -a = human readable format
    - -n = numerical order
    - -d <IP address> = deletes specific entry from the ARP cache

    - -v = verbose output
- **curl** = transfer data using web protocols to or from server from a web service or API. can also test web services and APIs
    - -O = saves the downloaded file using its original name from the URL
    - -o <name> = saves the downloaded file with a custom name you specify
    - -v = verbose, shows entire HTTP request and response
    - -L = location, tells curl to follow HTTP redirects
    - `-X [METHOD]` – Specifies the custom HTTP request method to use, such as `GET`, `POST`, `PUT`, or `DELETE`.
    - `-I` (or `--head`) – Fetches only the HTTP response headers rather than downloading the entire page content. This is crucial for verifying HTTP status codes (e.g., 200 OK, 404 Not Found, 500 Internal Server Error).
    - `-k` (or `--insecure`) – Explicitly allows `curl` to connect to an HTTPS site using an unverified or self-signed SSL/TLS certificate without throwing an error.
- **dig** = power tool to query DNS servers (like nslookup command but more details) (ex. dig [@server <domain> [type])
    - **@server**: Allows you to specify a particular nameserver to query (e.g., dig @8.8.8.8 google.com), which is helpful if you suspect your local ISP's DNS is the problem.
    - **-x** (Reverse Lookup): Finds the hostname associated with a specific IP address.
    - **+short**: Strips away the metadata and header information, providing only the answer (IP address or record value).
    - **+trace**: Performs a full, recursive trace from the root nameservers down to the authoritative server, showing every step of the resolution process.
    - Record Types: You can specify which DNS record you want to see:
        - A: IPv4 address.
        - AAAA: IPv6 address.
        - MX: Mail exchange records (where email should be sent).
        - NS: Nameserver records for the domain.
        - ANY: Requests all available records for the domain
    - **/etc/hosts** = provides a way to resolve hostnames to IP addresses WITHOUT a DNS server
    - **/etc/resolv.conf** = configuration file used by the system's resolver library to determine how to resolve hostnames into IP addresses via the Domain Name System (DNS) (Now replaced by **systemd-resolved**) Sometimes, changes to `/etc/resolv.conf` are overwritten by NetworkManager or DHCP upon reboot. Know that you might need to use `chattr +i` to

make the file **immutable** if you want to force manual settings, or better yet, configure the DNS via `nmcli` or **Netplan**
    - nameserver = specifies IP address of the DNS server. (e.g., `nameserver 8.8.8.8`). You can list multiple nameserver lines; the system will try them in order.
    - search = defines default FQDN search list. (e.g., `search example.com`) For example, if you type ping db01, it will actually ping db01.example.com
    - **`domain`**: Similar to `search`, but used for a single local domain name.
    - **`options`**: Used to modify the behavior of the resolver, such as setting timeouts or the number of retries.
- **nslookup** = lightweight tool to query DNS to obtain IP address to MAC address mappings. Non-Interactive Mode: You provide a single command and get a single answer. Great for quick checks. (ex. nslookup <hostname> <dns_server_ip>) Interactive Mode: You enter a dedicated `nslookup` shell to perform multiple queries.
    - **`-type=MX example.com`** = used as a command-line flag before entering interactive mode (command line flag)
    - **`set type=MX`**: Tells the utility to look for Mail Exchange records. (used in interactive menu)
    - **`set type=NS`**: Queries for the authoritative Name Servers of a domain. (used in interactive menu)
    - **`set type=A`**: The default; queries for IPv4 addresses. (used in interactive menu)
    - **`set type=AAAA`**: Queries for IPv6 addresses. (used in interactive menu)
    - **`server <ip>`**: Changes the DNS server you are currently querying while staying inside the interactive session. (used in interactive menu)
- **nmcli** = power tool network manager command line. A high-stakes utility used for controlling network manager and reporting network status. Primary tool for network configs on all RHEL distros. Rather than traditional flags, `nmcli` works with "objects." You should be comfortable with these:
    - **`device`** **(or `d` or `dev`)**: Interacts with the physical or virtual network interfaces (e.g., `eth0`, `wlan0`).
        - **`status`**: Displays the state of all network interfaces (connected, disconnected, unmanaged).
        - **`show`**: Provides detailed information about a specific device (IP, MAC address, MTU).

        - **connect <ifname>**: Tells NetworkManager to find a suitable connection for that hardware.
        - **disconnect <ifname>**: Safely disconnects the hardware and prevents it from auto-reconnecting immediately.
        - **wifi**: Lists available Wi-Fi networks or connects to one.
    - **`connection` (or `c or con`)**: Interacts with "connection profiles," which are the collections of settings (IP, gateway, DNS) applied to a device.
        - **show**: Lists all connection profiles. Use `--active` to see only those currently in use.
        - **up <name>**: Activates a specific connection.
        - **down <name>**: Deactivates a specific connection.
        - **add type <ethernet, wifi, gsm, cdma, bridge, bond, team, vlan, vpn, tun, tap, wireguard> ifname <name>**: Creates a new connection. Requires `type` (e.g., `type ethernet`) and `ifname` (interface name).
        - **`modify` (mod)**: Changes settings within a profile. Common parameters used with `mod` include:
            - `ipv4.addresses`: Sets a static IP.
            - `ipv4.gateway`: Sets the default gateway.
            - `ipv4.dns`: Sets the DNS servers.
            - `ipv4.method`: Set to `manual` for static or `auto` for DHCP.
        - **`delete`**: Removes a connection profile.
        - **`reload`**: Forces NetworkManager to re-read connection files from disk.
    - **`general` (or `g or gen`)**: Shows the overall status of NetworkManager and the system hostname.
        - status
        - hostname
        - permissions
        - logging
    - **`networking` (or `n or net`)**: Used to enable or disable networking entirely.
        - **on / off**: Enables or disables all networking managed by NetworkManager.
        - **`connectivity`**: Checks the state of the connection (Full, Limited, None, or Portal).
    - -P = pretty, human readable output
    - -t = terse, machine-readable output

- **netplan** = critical YAML-based utility for network configuration, primarily used in Debian-based distributions like Ubuntu. Netplan stores its configuration files in a specific directory: **`/etc/netplan/`**. Recognize that enabling DHCP in a netplan schema requires setting `dhcp4: true` inside the YAML file. For DNS servers, you must list our the addresses as addresses: [x.x.x.x, x.x.x.x]
    - **netplan apply** = applies current config to the system
    - **netplan try** = applies the config but reverts back in 120 seconds
    - **netplan status** = summarizes network interfaces and their config status
- **dhclient** = The traditional, universal command-line tool used to manually interact with the local DHCP client daemon to request, release, or renew IP address leases from a DHCP server.
    - `-r` – Releases the current DHCP lease for the specified interface, dropping the assigned IP address and sending a release packet back to the server.
    - `-v` – Enables verbose logging output, which allows you to watch the live DORA (Discover, Offer, Request, Acknowledge) negotiation sequence process in real time.
    - `[interface]` – Targets a specific interface explicitly (e.g., `dhclient eth0`) rather than running across all active network cards.
- **ethtool** = high-level utility used to query and control network driver and hardware settings for wired Ethernet devices. (layer 1 and layer 2 settings)
    - -i = query driver information
    - -S = display statistics on the ethernet link
    - -s = change settings (ex. ethtool -s eth0 speed 100 duplex full autoneg off)
    - -p = identify the physical port by blinking the LED on the back of the NIC
- **ip** = modern replacement of ifconfig, route, and arp. (layer 3 settings)
    - a (or address or addr) = check ipv4/ipv6 address
        - `show`: Displays all IP addresses assigned to all interfaces.
        - `add <IP/mask> dev <interface>`: Manually assigns a static IP (e.g., `ip addr add 192.168.1.50/24 dev eth0`).
        - `del`: Removes an assigned IP from an interface.
    - route (or r) = check the routing table
        - show: Displays the routing table. Use this to troubleshoot "Routing issues" or a missing "Gateway".
        - add default via <IP>: Sets the default gateway for the system.
    - link (or l) = manage physical or virtual network devices (layer 2)
        - show: Shows the status (UP/DOWN) and MAC address of interfaces. Use this to diagnose a "Link down" scenario.

        - set <interface> up/down: Manually enables or disables an interface.
        - set <interface> mtu <value>: Changes the Maximum Transmission Unit. Essential for fixing an "MTU mismatch".
    - neighbor (or neigh or n) = managing ARP/Neighbor cache (mapping IP to MAC)
        - `show` (or just `ip n`): Displays the current neighbor table. It shows the IP address, the network interface, the MAC address, and the state of the entry.
        - `add <IP> lladdr <MAC> dev <interface>`: Manually adds a static ARP entry to the table.
        - `del <IP> dev <interface>`: Removes a specific entry from the cache.
        - `flush dev <interface>`: Clears all neighbor entries for a specific interface. This is a common troubleshooting step for **IP conflicts**.
- **iperf3** = network bandwidth and throughput measurement
    - -s = starts utility in server mode to wait for incoming tests
    - -c <IP> = starts the utility in client mode and connects to a specific server IP.
    - **`-u` (UDP)**: Switches the test from the default TCP to **UDP**. This is critical for measuring **jitter** and **packet loss**, which TCP hides through its retransmission logic.
    - **`-b <n>[K/M/G]` (Bandwidth)**: Sets the target bandwidth. For UDP, this defaults to 1Mbps, so you must often set it higher (e.g., `-b 1G` for a 1Gbps link) to properly stress the network.
    - **`-P <n>` (Parallel)**: Runs multiple parallel streams at once. This is used to fully saturate high-speed links (like 10Gbps or higher) that a single stream might not be able to fill.
    - **`-R` (Reverse)**: Reverses the direction of traffic so the **server sends** and the **client receives**. This is useful for testing "download" speeds from a server.
    - **`-i <n>` (Interval)**: Sets the number of seconds between periodic bandwidth reports during the test (e.g., `-i 1` for a report every second).
    - **`-t <n>` (Time)**: Sets the duration of the test in seconds (default is 10 seconds).
    - **`-f [k/m/g/K/M/G]` (Format)**: Changes the units of the output (e.g., `M` for Mbits, `G` for Gbits).

    - `-J` **(JSON)**: Outputs the results in JSON format. This is a "must-know" for **Objective 4.2 (Shell Scripting)** because it allows scripts to easily parse the results.
    - `-u` = UDP jitter loss. identify if specific types of traffic (like VoIP) will suffer due to network instability.
    - `-O <n>`: Omit flag. Useful to ignore the first few seconds of a test where TCP "slow-start" might skew the results of a high-speed link.
- **mtr** = my trace route. Power user route tracing tool. Unlike traceroute command, mtr stays active, continuously probing routers on the path and updating statistics
    - -r = report mode. runs for a set # of cycles and then prints a report
    - -c <count> = Used with report mode to specify how many packets to send to each hop (default is 10)
    - -n = forces mtr to display IP addresses only. Skips DNS resolution, which makes the command faster
    - -w = wide output shows full hostnames without cutting them off
    - -i <seconds> = changes time interval between pings. Lowering this can help catch bursting packet loss
- **traceroute** = maps the path data takes from your local machine to a remote host, identifying every router (hop) along the way (by default, traceroute uses UDP which is blocked on most firewalls)
    - -n = numeric, only shows IP addresses of hope and does not resolve hostnames which takes more time
    - -I = ICMP ping packets instead of UDP packets. This might be required to get past firewall restrictions on hops
    - -T = TCP SYN packets for probing hops via port 80 or 443. Use when a firewall is blocking both ICMP and UDP.
    - -m <hops> = max hops (default is 30)
    - -w <seconds> = how long to wait for a response from the hop before giving up
    - -p <port> = specifies the port to use for the probe to map the route
- **tracepath** = simple version of traceroute. Tracepath can also discover path MTU (Maximum Transmission Units), which traceroute cannot discover. Tracepath also does not require sudo
    - -n = numeric, only shows IP addresses, skips DNS lookups
    - -p <port> = sets destination port to use (default is 44444)
    - -l <length> = sets the packet length (This is vital for MTU testing)
- **hostname** = displays machine name
    - -a or –alias = displays alias names of the host
    - -d or –domain = displays the name of the DNS domain
    - -f or –fqdn = displays FQDN (ex. server1.example.com)

    - -i or –ip-address = displays the IP address for the hostname
    - -I or –all-ip-addresses = displays all network addresses of the host
    - -s or –short = displays short host name (the part before the first dot)
- **nc** = netcat, the do-it-all swiss army knife tool for networking. It can read from and write to network connections using TCP or UDP (the default is TCP). It can see if s port is reachable, identify the version of a service running on a port, port scan, and it can even send files to a system without SSH or FTP
    - -l = listen for incoming connections (server mode)
    - -p = specifies which port to use (mandatory with -l)
    - -v = verbose, use -vv for even more detail
    - -z = zero IO, used for scanning and connecting to a host but not send any data
    - -u = UDP, switches from TCP to UDP packets
    - -n = numeric, only IP addresses. Skips DNS resolution
    - -w = wait timeout before stop trying to connect
    - **`[hostname/IP] [port]`** – Specifies the target and port (e.g., `nc -v localhost 25`).
- **nmap** = network mapper. Powertool for network port scans. While nc can run a port scan, nmap is the the high-powered purpose built solution for port scanning. (The basic syntax is `nmap [Scan Type] [Options] {target specification}`)
    - -sS = SYN stealth scan. It sends a SYN packet and never fully establishes the connection with SYN ACK which makes it hard for logs to catch it
    - -sT = TCP connect scan. This is the default method. It completes the 3-way handshake and scans for TCP connections
    - -sU = UDP scan . Scans for all UDP services (like DNS, DHCP, or SNMP). It is much slower than TCP scans
    - -p = specifies port range (e.g., `-p 22,80,443` or `-p 1-1024`)
    - -F = fast scan. scans only the top 100 most common ports instead of the top 1000
    - -Pn = skips the initial ping command when starting a port scan. This is useful if a firewall is blocking ICMP
    - -sV = detect the software version that is running on the port
    - -O = OS fingerprinting. Uses TCP/IP stack to guess the OS of the target
    - -A = aggressive combo flag that combines OS fingerprinting, version detection, script scanning, and traceroute all at once
    - `-oN / -oG` (Output): Saves the results to a Normal file or a Grepable file (essential for automation).
- **ping / ping6** = testing reachability using ICMP packets
    - -c 4 = specifies to send 4 ICMP packets

    - -i 4 = specifies to wait 4 seconds between sending each packet
    - -s <size> - change packet size
    - -I <interface> = set source interface
    - -W <timeout> = wait timeout
- **ss** = modern replacement for netstat. Used to see open ports and connections
    - -t /-u = filters for tcp or udp packets
    - -l shows only listening ports
    - -a = all, both listening and established (active) connections
    - -s = summary, displays high level summary of all socket types
    - -n = numeric output, shows raw IP addresses and port numbers and doesn't resolve hostnames
    - -p = shows process using the socket (requires root)
    - -tulpen = show all important info at once
- **tcpdump** = packet capture (like wireshark)
    - -i <interface> = specify interface (e.g., `-i eth0` or `-i any`).
    - -n = no DNS resolution
    - -nn = no port resolution. Shows port # (80) instead of service names (http)
    - -c <count> = packet count. Stop after n packets. Vital for automation/scripts
    - -v, -vv, -vvv = verbose, increases amount of details
    - -w <file.pcap> = write to a file. Writes the output of the packet capture to a file
    - -r <file.pcap> = read from a file. Load a previously captured packet capture
    - -X = shows payload in both Hex and readable (ASCII) text
    - -s <size> = sets how many bytes of each packet to capture

Backup, Archiving & Compression

- **unzip** = extracts files from a `.zip` archive. It's the standalone counterpart to `7z`, `gzip`, `bzip2`, and `xz` — most Linux distros ship it separately since `zip`/`unzip` aren't part of the base coreutils set. (ex. `unzip archive.zip -d /target/dir`)
- **cpio** = create, extract, and copy archive files. Unlike tar, which can take filenames as direct arguments, cpio is designed to read a list of filenames from stdin, which means it almost always needs a pipe (|) or redirection (< or >). It creates files with the .cpio file extension
    - -o = copy-out, creates an archive by reading the list of files from stdin and writes the archive to stdout
    - -i = copy-in, extracts files from an archive. It reads the archive from stdin

    - -p = pass-through. copies files from one directory tree to another without creating an intermediate archive file
    - -d = create directory automatically where needed during extraction
    - -v = verbose
    - -t = table of contents
    - -H [format] = lists the contents of the archive without extracting them
    - -u = unconditional, overwrites existing files without prompting (useful in scripting)
- **tar** = tape archive, most common utility for bundling multiple files into a single archive file called a “tarball” (Usually paired with gzip for compression). It creates files with the .tar file extension. When bundled with compression tools, it creates files with these file extensions: (.tgz, .tbz2, .txz)
    - -c = create a new archive
    - -x = extracts files from an archive
    - -v = verbose output, lists all files being processed
    - -f <filename> = specifies filename of the archive (must be the last flag used in sequence)
    - -z = compress or decompress using gzip
    - -j = compress or decompress using bjip2
- **7z** = 7-zip compression utility for both archiving (bundling) and compression (shrinking) in a single step. The `7z` syntax is a bit different from `tar`. It uses **functions** (commands) and then **switches** (flags). Note that for the main commands, you do **not** use a leading dash. 7z can also handle encryption well and is compatible with almost every archive format.It creates files with the .7z or .zip file extension
    - Essential commands (functions)
        - a = add, creates a new archive or adds files to an existing one
        - x = eXtracts files with their full directory structure (use this for backups)
        - l = lists what is inside the archive without unpacking it
        - d = deletes a specific files from inside an existing archive
    - Essential flags (switches)
        - -p = password encrypts the archive.
        - -t [type] = specifies compression format (-tzip, ttar, t7z)
        - -o [dir] = output, sets the destination directory for extraction
        - -v [size] = volume, splits the archive into chunks (ex. -v100m for 100MB slices)
        - -y = automate yes

- **gzip** = standard fast utility for compressing files but has the lowest compression ratio. It also does not archive files, which is why it is used alongside tar for archiving. It creates files with the .gz extension or the .tgz when archived with tar
    - -d = decompress the file (same as gunzip)
    - -k = keeps original uncompressed file instead of deleting it after compression
    - -f = force overwrite output file if it already exists
    - -v = verbose, shows compression ratio and percentage of space saved
    - -c = writes output to stdout, leaving the original file untouched
    - -l = lists the compressed or uncompressed size and ratio
    - -r = recursive travels through all directory structures to compress every file individually
    - -1 to -9 = Sets compression speed vs. quality. `-1` is fastest; `-9` is smallest. Default is `-6`. Useful for performance-tuning scenarios.
    - -z = archive with tar
- **bzip2** = high-quality data compression tool that is more efficient than gzip but slower to execute and is a good middle ground between gzip and xz for compression ratios. It creates files with the bz2 extension or the tbz2 when archived with tar
    - -d = decompress a .bz2 file
    - -k = keep original. by default, bzip2 deletes the original source file. This flag keeps it.
    - -c = writes output to stdout, leaving the original file untouched
    - -f = force overwrite output file if it already exists
    - -v = verbose, shows compression ratio and percentage of space saved
    - -1 to -9 = Sets compression speed vs. quality. `-1` is fastest; `-9` is smallest. Default is `-6`. Useful for performance-tuning scenarios.
    - -j = archive with tar
- **xz** = heavyweight champion of Linux compression tools. It provides the highest compression ratios of the compression algorithms, but is the slowest to execute and uses a massive amount of CPU and RAM resources.
    - -d = decompress a .xz file
    - -k = keep original. by default, xz deletes the original source file. This flag keeps it.
    - -f = force overwrite output file if it already exists
    - -v = verbose, shows compression ratio and percentage of space saved
    - -l = list info about compressed files (compression ratio, size)
    - -T = Uses multiple CPU threads (ex. `-T 0` uses all cores).

    - -1 to -9 = Sets compression speed vs. quality. `-1` is fastest; `-9` is smallest. Default is `-6`. Useful for performance-tuning scenarios.
    - -J = archive with tar
- **dd** = data duplicator. Used to create forensic bit-by-bit copies of files or entire filesystems. Unlike cp which copies file contents, dd copies data directly from one location to another, making it essential for managing disk images, partitions and boot records. The syntax for `dd` is unique because it uses `option=value` instead of the traditional dash-based flags. `dd` can also be used for data destruction with the following parameters (sudo dd if=dev/urandom of=/dev/sdb)
    - if= : input file, or source of the data that you want copied (ex. /dev/sda)
    - of= : output file or the destination for the data
    - bs= : block size for input and output. Large blocks are faster(e.g., `bs=1M`, `bs=4k`).
    - count= : copy only specific # of input blocks. Useful to backup specific parts of a disk (like MBR)
    - status=progress : displays transfer speed and progress bar
    - conv= : tells dd how to handle data. Mainly used in data recovery scenarios (e.g., `noerror` to ignore read errors).
- **ddrescue** = specialized, smarter version of dd used for data recovery from failing hardware (like bad sectors or physical read errors) The syntax for `ddrescue` is: `ddrescue [options] infile outfile [mapfile]` . The map file is the file tracking the progress and is not needed.
    - -f = force, required when the output is a block device (like a new harddrive). Use this when cloning one disk to another
    - -n = no scraping. don't spend too much time on the hard parts and do a quick pass to get the most data in the shortest time
    - -r [n] = number of retries for reading a bad sector (higher numbers take longer but might recover more data)
    - -d = direct disk access, bypassing the kernel cache. Useful for hardware that is behaving erratically
    - -v = verbose output on recovery progress
- **rsync** = power tool for copying on Linux (Can do delta migrations, and has many other cool features for bulk copies) (ex. rsync -rc –progress testdir/ testdir1)
    - -a = archive mode, includes recursion, symlinks, permissions, timestamps, owner, and group settings (USE THIS EVERY TIME)
    - -v = verbose
    - -e = execute, specifies the remote shell to use (usually `ssh`).
    - -D = retain device and special files
    - -g = retain file group

    - -h = human-readable output
    - -l = copy symbolic links as well as actual files
    - -H = preserves hard links. Without this, rsync treats two hard-linked files as independent files on the destination
    - -o = retain file ownership
    - -P = Shows a progress bar and keeps partially transferred files if the connection breaks.
    - -p = retain file permissions
    - –progress = display file copy progress bar
    - –bwlimit = bandwidth limit
    - –stats = file transfer statistics
    - -r = recursively copy directory and subdirectories
    - -t = retain file modification time
    - -c = checksum to ensure hashes of source and destination are the same
    - -z = compress destination file to a zip
    - –delete = deletes files in destination before copying over source
    - -u = updates, preserves newer file
    - -n = simulation mode, shows what will happen without taking action (USE THIS BEFORE EVERY RSYNC JOB)
- **zcat** = compressed cat, utility used to view the contents of a compressed file without decompressing the file. It only works for .gz files. It will fail on `.bz2` or `.xz` files (use `bzcat` or `xzcat` for those). zcat does exactly the same thing as gunzip -c or gzip -dc
    - -f = force, Tells `zcat` to treat the file as uncompressed if it isn't in gzip format.
- **zgrep** = compressed grep, searches for expressions, patterns, strings within a compressed file. It only works on .gz files.It will fail on `.bz2` or `.xz` files (use `bzgrep` or `xzgrep` for those). (ex. zgrep "192.168.1.1" log1.gz)
    - -i = case insensitive
    - -v = invert match, show everything except the lines that match the pattern
    - -c = count # of lines that match the pattern
    - -n = line # where the match was found
    - -l = lists names of files that contain a match
    - -r = recursive, searches through all compressed files in a directory and subdirectories
- **zless** = compressed less "pager" utility specifically designed to let you view the contents of compressed files (primarily **gzip**) one screen at a time.
    - -N = display line numbers
    - -S = chop long lines rather than wrap them
    - -i = case insensitive when searching for keywords

    - -X = prevents less from clearing the screen when exiting less
    - For internal navigation, here are the options you have
        - q = exit
        - b = move backward one window
        - g = jump to beginning of the file
        - G = jump to the end of the file
        - /pattern = search forward for a specific text
        - ?pattern = search backward for a specific string
        - n = repeat the previous search (forward)
        - N = repeat the previous search (backward)
        - spacebar = go forward a page
        - esc+v = go back a page
        - arrow keys = scroll up and down through file
        - F = forward to the end of the file. Useful for viewing logs
        - 4 g = go to the line #4

Advanced Commands and Programs

Device & Hardware Management

- **nvtop** = interactive monitor for GPU resources
- **Kernel Modules:**
- **depmod** = dependency module, manages kernel modules, and scans files to figure out which modules rely on others to function
    - -a = probes all modules and rebuilds modules.dep by scanning every single module
    - -A = quick compare of file timestamps. Rebuilds modules.dep only if it detects that a kernel module is newer than the existing one.
    - -n = dry run, processes the dependencies but print the output directly to screen instead of overwriting the modules.dep file
    - -v = prints exactly what is being scanned
- **insmod** = (insert module) is a low-level utility used to insert a kernel module into the Linux kernel at runtime. simple tool
    - [filename]
    - [module parameters]
    - -f = force load the module even if the version of running kernel doesn't match version the module was compiled for
    - -v = verbose, shows detailed insertion process
- **lsmod** = (list modules) command shows you exactly which kernel modules are currently loaded into memory.

- **modinfo** = extracts detailed information directly from the Linux kernel module files (usually ending in `.ko`) located on your disk.
    - -a = displays author of the module
    - -d = displays the description of what the module does
    - -l = displays only the license type
    - -n = displays only the absolute path to the module file name on disk
    - -p = displays available parameters you can pass to the module when loading it with modprobe
    - -F = allows you to specify a specific field display
    - -k = specify a different kernel version to query, rather than the one currently running
- **modprobe** = add or remove modules from the Linux kernel. It is considered "intelligent" because it automatically handles **dependency resolution**.
    - -r = remove, unloads the specified module and any associated dependencies. It can also be used to unload a filesystem from memory.
    - -v = verbose, shows what program is doing
    - -n = dry run, shows dependency checks but doesn't load anything
    - -f = force, attempts to load module even if version doesn't match the current kernel
    - -i = ignore, tells modprobe to ignore any install or remove commands in the config files
    - -D = dependencies, lists dependencies for a module w/o actually loading it
- **rmmod** = (remove module) command is a simple utility used to unload a specific module from the Linux kernel
    - -f = force, attempts to remove module even if it is currently in use
    - -v = verbose, gives details on removal
    - -s - syslog, sends errors to system log
    - -w = wait, tells command to wait until module is no longer in use before removal
- **dmesg** = examine kernel ring buffer to see messages produced by kernel when hardware events occur
    - -T /–ctime = human readable timestamps (date/time)
    - -H / –human = human friendly output, combines -T with less pager
    - -w / –follow = live monitoring
    - -l / –level = filter by severity (e.g., `emerg`, `alert`, `crit`, `err`, `warn`, `notice`, `info`, `debug`).
    - -n / –console-level = set console logging level, controls which messages are sent directly to the terminal
    - -c / –read-clear = clear the buffer after printing the output to the screen (THIS DESTROYS THE LOG)

    - -C / –clear = wipes the buffer without printing anything (THIS DESTROYS THE LOG)
- **dmidecode** = Desktop management interface table. Descriptions of hardware as reported by UEFI/BIOS
    - -t / –type = filter by keyword or number (DMI types)
        - **0** `bios` BIOS/UEFI Vendor, version, and release date.
        - **1** `system` Manufacturer, product name (e.g., "Dell PowerEdge"), and serial number.
        - **2** `baseboard` Motherboard model and revision.
        - **3** `chassis` Physical enclosure type (e.g., "Rack Mount Chassis").
        - **4** `processor` CPU family, voltage, max speed, and socket type.
        - **17** `memory` Details for individual RAM sticks (Size, Type, Speed, Manufacturer).
    - -s / –string = filter by a string (ex. system serial #)
    - -u / –dump = hexadecimal dump, displays the DMI table in Hex format
    - -q / –quiet = less verbose.
- **ipmitool** = OOBM for server hardware (another way to manager your server even if it is powered off) It communicates with the **Baseboard Management Controller (BMC)**, a specialized processor on the motherboard that monitors the physical state of the computer.
    - -I = defines the IPMI interface to use (Common values include open or lanplus)
    - -H = IP address or DNS hostname of the remote BMC you wish to manage
    - -U = username required to authenticate with remote BMC
    - -P = password for the specified user
    - -v = verbose logging
- **sensors** = linux monitoring sensors for monitoring temp, voltage, fan speeds, etc…
    - -f = fahrenheit temp
    - -u = raw output without units of measurement
    - -j = JSON format
- **sensors-detect** = probes hardware sensors
- **lscpu** = list detailed info about CPU
    - -e = extended readable output
    - -p = parsable output for easy reading
    - -J = JSON format
    - -C = cache details (L1, L2, L3)
    - -y = physical IDs
- **lshw** = list hardware

    - -short = compact single-line-per-device table
    - -class <class> = classes (processor, memory, display, network, storage, bus)
    - -json = JSON output
    - -html = HTML output
    - -businfo = bus info
    - -sanitize - removes sensitive info (serial #, IP)
- **lsmem** = how the kernel actually sees and manages physical RAM blocks (check RAM block status)
    - -J = JSON
    - -o = specify which column to display (ex. STATE, BLOCK, SIZE)
    - -n = No Header row in output
    - -S = Groups memory blocks based on whether they are online or offline
    - -r = raw output
    - -s = summarize only
- **free** = lists amount RAM free and used (check RAM status)
    - -h = human readable
    - -m = megabytes
    - -g = gigabytes
    - -t = total
    - -w = wide mode, separates buffer and cache
    - -s 10 = refreshes output ever 10 seconds, (crucial for watching a memory leak)
- **lspci** = lists PCI interfaces
    - -v = verbose
    - -k = kernel drivers
    - -n = numeric codes
    - -t = tree diagram
    - -s [[bus]:[slot][.func]] = specifies devices
    - -mm = machine readable
- **lsusb** = lists usb interfaces
    - -v = verbose
    - -t = tree diagram
    - -s [[bus]:][devnum] = select device
    - -d [vendor]:[product] = search by ID
- **dracut** = creates initrd or initramfs, which is a small, temporary filesystem that is loaded into memory by the bootloader with the essential drivers and modules to mount the root filesystem
    - -f / –force = overwrite existing file
    - –kver = specific kernel version

    - –add = include modules (like `network` or `crypt`) that the tool might not have auto-detected.
    - – omit = exclude modules
    - –regenerate-all = rebuild initramfs for every kernel version. Used after major system update
    - -v = verbose logging
- **mkinitrd** = legacy utility for initrd (replaced by dracut in most distros)
    - -f = force overwrite an existing initrd image file
    - -v = verbose logging
    - –with=<module> = add specific kernel module to image
    - –preload=<module> = ensure specific module is loaded first

Storage Management (LVM, RAID & Filesystems)

- **tmpfs** = a temporary, RAM-backed filesystem format. Files written to a `tmpfs` mount live in volatile memory (and swap, if needed) rather than on disk, so they disappear on reboot or unmount. It's commonly used for `/tmp` and `/dev/shm` because it's extremely fast and avoids unnecessary disk wear.
- **mount / umount** = mounts or unmounts filesystems (like attaching a usb drive or secondary hard drive so you can actually interact with it. Syntax: mount [flags] <device/source> <mount_point_directory>
    - `-t <type>`: Specifies the filesystem type (e.g., `ext4`, `xfs`, `cifs`, `nfs`, `vfat`). Use this when Linux cannot automatically detect the file system type.
    - -a = mounts/unmounts all filesystems listed in /etc/fstab (USE THIS MOST OF THE TIME)
    - -o <options> = passes specific mount options (e.g., `mount -o ro` for read-only, or `mount -o remount`).
        - `ro` / `rw`: Manually forces the filesystem into Read-Only or Read-Write mode.
        - `remount`: Alters the permissions of a drive that is already actively mounted without requiring you to unmount it first (e.g., `mount -o remount,rw /`).
        - `noexec`: Blocks any scripts or binaries on that partition from being executed. Frequently used to lock down a public directory like `/tmp`.
        - `nosuid`: Blocks the execution of Set-User-ID bits, preventing users from gaining elevated root privileges from files on that drive.

        - `loop`: Mounts a standalone file (like an `.iso` disk image) onto a folder as if it were a physical disk drive.
    - `-r`: Mounts the filesystem as Read-Only (identical to `-o ro`). Used for safely pulling data off a corrupted drive or performing forensics.
    - `-w`: Mounts the filesystem as Read-Write (Default behavior).
    - `-f`: Forces an unmount. Crucial for severing connections to stuck network shares (NFS or SMB) when a remote server goes offline and freezes your local system.
    - `-l`: Lazy unmount. Instantly unlinks the partition from the directory tree, then cleans up active program dependencies quietly in the background. This is the primary fix for the "Device is busy" error.
- **lvchange** = change the attributes of an existing logical volume (LV) without altering the data inside it or resizing it. toggles LV operational state
    - -a = change availability, deactivates the volume so the kernel can no longer access it
    - -p [r] [rw]= change permissions
    - –refresh = refresh the logical volume
    - -y = automate yes
- **lvcreate** = takes a chunk of space from a storage pool and creates a logical volume with it
    - -n = name the volume
    - -L = absolute size, (ex. -L 50G for 50 gigabytes of storage)
    - -l = specifies size using LVM “extents” (ex. -l 100%FREE to use all remaining disk space)
    - -m = create a mirrored logical volume for redundancy
- **lvdisplay** = display detailed info about a logical volume
    - [Volume Path] = show details on logical volume path only
    - -m = display map of hard drives that map to specific logical volumes
    - -a = show all (this is default)
    - -v = verbose output
- **lvremove** = permanently delete a logical volume, requires the use of umount to unmount the volume first (WARNING: DANGEROUS)
    - -f = force, WARNING: EXTRA DANGEROUS
    - -y = automate yes, WARNING: EXTRA EXTRA DANGEROUS
- **lvresize** = unline lvchange, lvresize actually changes the size of the volume by EITHER shrinking OR expanding the volume
    - -r = resize filesystem, MOST IMPORTANT, Expanding a volume doesn't automatically expand the filesystem (like `ext4` or `xfs`) inside it. Using `-r`

tells LVM to automatically call the appropriate filesystem resizing tool (`resize2fs` or `xfs_growfs`) so you don't have to do it manually.
    - -L = change by size (ex. -L -10G)
    - -l = specifies size using LVM “extents” (ex. -l +100%FREE to use all remaining disk space)
- **lvextend** = increase the size of a LV, replaced by lvresize
    - -r = resize filesystem, **The most critical flag.** Expanding the LVM block device does not automatically expand the **Filesystems** residing on it. Using `-r` forces LVM to automatically call the correct expansion tool for filesystems like **ext4**, **xfs**, or **btrfs** so the OS can actually use the new space.
    - -L = change by size (ex. -L +10G)
    - -l = specifies size using LVM “extents” (ex. -l +100%FREE to use all remaining disk space)
- **lvreduce** = reduce the size of a LV, replaced by lvresize
    - -r = resize filesystem, **The most critical flag.** Expanding the LVM block device does not automatically expand the **Filesystems** residing on it. Using `-r` forces LVM to automatically call the correct expansion tool for filesystems like **ext4**, **xfs**, or **btrfs** so the OS can actually use the new space.
    - -L = change by size (ex. -L -10G)
    - -l = specifies size using LVM “extents” (ex. -l -50%FREE to use half of remaining disk space)
- **lvs** = display logical volume info in column view
    - -o = select only columns
    - -a = show all LVs, even hidden ones
    - –noheadings = strips title row
- **vgchange** = change the attributes of one or more volume groups (VG), toggles pool operational state
    - -a = change availability (Followed by `y` (yes) or `n` (no). For example, `vgchange -a y vg_data` makes the volume group active and accessible to the OS.)
    - -y = automate yes
    - -l = max LVs allowed in LG
    - -p = max PVs allowed in LG
    - -S = select only specific VG that matches criteria
- **vgcreate** = create a volume group
    - -s = set size of physical extents (default is 4MiB), need to increase this for large volume groups
    - -y = automate yes

    - -v = verbose
    - -A = controls whether LVM automatically backs up the VG metadata after a change
- **vgdisplay** = display deep-dive of detailed info about Volume Group
    - [VG_Name] = targeted volume group
    - -v = verbose
    - -s = short summary
    - -A = show active only
- **vgexport** = safely eject an inactive volume group to make it safe to move the physical disk to another system (similar to ejecting a USB drive on a Windows system) YOU MUST RUN UMOUNT AND VGCHANGE BEFORE VGEXPORT
    - [VG_Name] = targeted volume group
    - -a = export all inactive volume groups
    - -v = verbose
    - -y = automate yes
- **vgextend** = add one or more physical volumes to an existing volume group and extends the volume group
    - -y = automates yes
    - -t = test mode, dry run
    - -v = verbose
    - -Z = zero metadata, determines whether to wipe the first 512 bytes of data of the new PV to ensure no old partition data interferes with the merge
- **vgimport** = import a volume group to a system
    - -a = import all unknown or exported volume group the system can find on connected disks
    - -v = verbose
    - -f = force import (DANGEROUS)
- **vgremove** = removes a volume group (DANGEROUS)
    - -f = force remove even if it contains logical volumes (EXTRA DANGEROUS)
    - -y = automate yes (EXTRA DANGEROUS)
    - [VG_NAME] = specifies a volume group
- **vgs** = column display of all volume groups
    - -o = select columns
    - –noheadings = removes headers
    - –units [g] = force output into format like g for gigabytes
    - –separator [char] = changes the space between columns to a character such as a , for generating csv format
    - -a = show all, even hidden volume groups

- **vgscan** = scan the physical disks for unknown or exported volume groups that can be imported.
    - –mknodes = recreates special device filesin the /dev directory
    - -v = verbose
    - -t = test, dry run mode
- **pvcreate** = initialize a disk or partition for use by the LVM
    - -f = forces creation of PV even if the device already contain a a partition table or LVM signature (DANGEROUS)
    - -y = automate yes
    - -u = manually specify the UUID for the device
    - -v = verbose
    - –metadatacopies <0|1|2> = sets the # of metadata copies for redundancy
    - –dataalignment <size> = aligns the start of the data area to a specific size (relevant for SSD performance tuning)
- **pvdisplay** = detailed multi-line attribute list for each physical volume
    - -C = display output in column format
    - <device_path> = display info for specific device only
    - -v = shows verbose details
    - -m = display mapping of physical extents to logical volume
    - –short = concise summary
- **pvmove** = move Physical Extents (PEs) from one Physical Volume (PV) to another within the same Volume Group. This is primarily used when you need to retire a failing disk or migrate data to faster storage without taking the system offline.
    - -n <LV> = moves only the extents belonging to a specific Logical Volume
    - -i <sec> = reports progress every n seconds
    - -b = runs the move in the background
    - –abort = stops the pvmove operation
    - –atomic = ensures the move is completed fully or not at all (relevant for data integrity)
- **pvremove** = used to remove the LVM label (metadata) from a disk or partition, effectively "un-initializing" it so it can be used for other purposes outside of LVM. Order of operations is
    - **Unmount** the filesystem.
    - **`lvremove`**: Delete the Logical Volume.
    - **`vgreduce`**: Remove the Physical Volume from the Volume Group.
    - **`pvremove`**: Remove the LVM metadata from the disk.
    - -f = force remove even if it has metadata (DANGEROUS)
    - -y = automate yes (DANGEROUS)
    - -v = verbose

    - –debug = deep technical output
- **pvresize** = grow or shrink the size of the physical volume
    - –setphysicalvolumesize <size> = sets to a specified size
    - -v = verbose
    - -t = dry run, test mode
- **pvs** = concise, one line summary per device
    - -o = specify which columns to display (ex. pvs -o pv_name,pv_uuid)
    - –noheadings - suppress header row
    - –units <unit> = forces output in specific units (m for MB, g for GB)
    - -a = shows all devices even those not initialized by LVM
- **pvscan** = discover physical volumes on the disk
    - -e = shows PVs that have been exported
    - -n = only shows PVs that are not yes part of any volume group
    - –cache = manually update the LVM metadata for one of more devices
    - -u = displays UUID of discovered physical volumes
    - -v = verbose
- **blkid** = find unique identifiers (UUID) and filesystem types necessary or persistent configuration
    - -s = shows only specific tag (ex. blkid -s UUID)
    - -o <format> = changes output format (ex. (e.g., `value`, `device`, `full`)
    - -p = probes device directly bypassing cache
    - -g = garbage can for blkid cache for device not being used anymore
    - -u <usage> = Restricts the search to specific types (e.g., `filesystem`, `raid`, `crypto`).
- **fdisk** = create/manage MBR partition (ex. fdisk /dev/sdx)
    - -l <device> = lists partition table
    - -s <partition> = displays size of specific partition in blocks
    - p = print partition table
    - n = create new partition
    - d = delete partition
    - t = change partition's SystemID (type)
    - L = list known partition types
    - w = Write table to disk and exit
    - q = quit without saving changes
    - m = help menu
- **gdisk** = create/manage GPT partition (ex. gdisk /dev/sdx)
    - -l <device> = lists partition table
    - -p <file> = probes the device
    - p = print partition table
    - n = create new partition

    - d = delete partition
    - t = change partition's typecode
    - L = list known partition types
    - w = Write table to disk and exit
    - q = quit without saving changes
    - i = information
- **growpart** = used to auto-extend a partition to fill the remaining available space on a block device. It is most commonly used in Cloud (AWS, Azure, GCP) and Virtualized (KVM, VMware) environments where a disk has been expanded at the hypervisor level. Unlike most commands, growpart requires a space between the device and the partition number. (ex. growpart /dev/sda 1).
    - -N = dry run
    - -v = verbose
    - -h = help
- **lsblk** = shows structure of physical disks divided into partitions and how those partitions are consumed by LVM. lsblk is the best tool to see which Physical Volumes (PV) belong to which Volume Groups (VG) and which Logical Volumes (LV) are on top.
    - -f = filesystems
    - -p = full paths
    - -a = all, including hidden or empty partitions
    - -l = flat list view output
    - -o <columns> = customize which columns to show
    - -n = no headings
    - -J = JSON format
- **parted** = partition editor. swiss army knife of linux partitioning tool. Does not require an interactive menu like gdisk
    - -l = list partition layout on all block devices
    - -s = script mode, never prompt for input
    - -m = display output in machine-parseable format (colon-separated)
    - -a <alignment> = sets alignment for new partitions
    - **`print`**: Displays the partition table, disk size, and model.
    - **`mklabel <type>`**: Creates a new disk label (partition table). You'll usually choose `msdos` for MBR or `gpt` for GPT.
    - **`mkpart`**: Creates a new partition. You must specify the type (primary, logical, etc.), filesystem, and the start/end points.
    - **`resizepart`**: Resizes a partition by changing its end point. This is a common **Troubleshooting** task when a partition is full.
    - **`rm <number>`**: Deletes a partition by its number (e.g., `rm 2`).

    - **`unit <unit>`**: Changes the display units to bytes, sectors, GB, or TB.
- **df** = disk free
    - -h = human readable (MOST COMMON)
    - -T = file system type
    - -a = all filesystems
    - -i = inode usage
    - -l = local file systems only
    - -t <type> = Limits the output to a specific filesystem type, such as `df -t ext4`.
    - -x = Excludes specific types, such as `df -x tmpfs`, to focus only on physical disks.
    -
- **du** = disk used
    - -s = summarize total
    - -a = all files
    - -h = human readable (often used with -s as -sh)
    - -c = grand total
    - -d <N> = max depth
    - -x = one file system only
- **fio** = I/O checker. Tells you how fast your storage is using IO measurements
    - --name=<jobname> = defines the name of a test job
    - –filename=<path> : specifies the file or block device to test (ex. /dev/sdb)
    - –rw=<type> = sets the I/O pattern: `read`, `write`, **`randread`**, or `randwrite`.
    - –size=<size> = total amount of data to read or write (eg. 512M or 2G)
    - –direct=1 = uses non-buffered I/O and bypasses OS cache (use this to get a better picture of actual disk speed)
    - –ioengine=<engine> = how the I/O is submitted (libaio or sync)
    - –runtime=<sec> = limits the test to a specific duration
    - -iodepth=<N> = number of I/O units to keep in flight at once
- **fsck** = file system consistency checker. (Like sfc for Linux) Use for repairing corrupt files (NEVER RUN ON A MOUNTED FILESYSTEM. ALWAYS UNMOUNT BEFORE RUNNING FSCK)
    - -A = all filesystems in one go
    - -y = automate yes
    - -n = check only but don’t repair files
    - -C = completion bar
    - -t < =type> = specifies filesystems type to check
    - -f = Force a check even if filesystem is marked as clean
    - -N = dry run

- **mkfs** = make filesystem
  - -t <format> = format with type (xfs, btrfs, ext4)
  - .<format> = format with type (ex. mkfs.ext4)
  - -L <Label> = sets volume label. Labels are vital for **Objective 1.3 (Mounted storage)** because they allow you to mount drives in `/etc/fstab` by name instead of volatile device paths like `/dev/sdb1`
  - -v = verbose
  - -V = Version info
  - -c = checks device for bad blocks before creating filesystem
- /proc/mdstat = a virtual file that shows a status report for RAID (File, not a command)
  - cat /proc/mdadm = display one-time snapshot of RAID status
  - watch -n 1 cat /proc/mdstat = live monitoring of RAID rebuild
- **mdadm** = create, manage, or delete RAID array (stupid simple RAID tool for block-level RAID)
  - -c = create
  - -l = raid level (0,1,5,6,10)
  - -n = # of active devices in the array
  - -D = show detailed info about RAID device
  - -f = manually mark the selected disk as faulty
  - -r = removes a faulty disk from the array
  - -a = adds a new disk to an existing array
  - -S = deactivates a RAID array and releases the resources
- **btrfs** <group> <subcommand> <arguments> = filesystem with a volume manager included (new standard for Linux)
  - mkfs.btrfs = mk filesystem btrfs
    - -d <raidtype> = sets raid level for data. Common values are `single`, `raid0`, `raid1`, `raid10`, `raid5`, and `raid6`.
    - **`-m <profile>`**: Sets the RAID level for **Metadata**. It is a best practice to keep metadata mirrored (`raid1`) even if data is `single` to ensure filesystem integrity.
    - **`-L <label>`**: Assigns a volume label for easier mounting.
  - filesystem = general maintenance and status checks
    - show = display info about filesystem
    - df <path> = disk free
    - resize <size> <path> = Resizes a mounted filesystem. Use `max` to expand to the full partition size.
    - `label <path> [<newlabel>]`: Gets or sets the filesystem label.

        - `defragment <path>`: Manually triggers defragmentation (useful since CoW can cause fragmentation).
    - subvolume = subvolume and snapshot mgmt
        - `create <name>`: Creates a new subvolume.
        - `list <path>`: Lists all subvolumes and snapshots in the filesystem.
        - `delete <name>`: Removes a subvolume or snapshot.
        - `snapshot [-r] <source> <dest>`: Creates a snapshot of a subvolume.
            - **Flag `-r`**: Creates a **read-only** snapshot (essential for backups).
    - device = device mgmt
        - `add <device> <path>`: Adds a new physical device to an existing pool.
        - `delete <device> <path>`: Safely removes a device from the pool (data is migrated off it automatically).
        - `scan`: Scans for Btrfs devices (useful after adding a new disk).
        - `stats <path>`: Displays I/O error statistics for devices in the pool.
    - scrub = maintenance
        - `scrub start <path>`: Reads all data and metadata from the disk and uses checksums to find and fix silent data corruption (bit rot).
    - balance = maintenance
        - `balance start <path>`: Redistributes data chunks across all available devices. Often used after adding/removing a device or to change RAID levels online.
- **ext4** = old reliable filesystem comes default on most Linux Distros
    - mkfs.ext4 = making ext4 filesystem
        - **`-L <label>`**: Assigns a volume label to the filesystem for easier identification.
        - **`-m <percentage>`**: Sets the percentage of blocks reserved for the super-user (default is **5%**).
        - **`-b <block-size>`**: Specifies the block size (typically 1024, 2048, or 4096 bytes).
        - **`-U <UUID>`**: Manually sets a specific Universally Unique Identifier (UUID) for the volume.
    - tune2fs = tuning parameters on ext4 filesystem

        - `-l`: **(Highly important)** Lists the contents of the filesystem superblock, including current mount counts, features, and UUID.
        - `-c <max-mounts>`: Sets the maximum number of times the filesystem can be mounted before a mandatory check is performed.
        - `-i <interval>`: Sets the maximum time interval (days/months/weeks) between filesystem checks.
        - `-m <percentage>`: Adjusts the reserved block percentage after the filesystem has already been created.
        - `-L <label>`: Changes the volume label.
    - resize2fs = expand or shrink filesystems (used for ext4 only. You can only shrink an ext4 filesystem while it is unmounted. Targets a device path (like `/dev/sda1`))
        - -p = progress bar
        - -f = force even if filesystem has issues (DANGEROUS)
        - -M = minimum size (used for compact baseline images)
        - -d = debug mode
        - -S <size> = manually specify new size
    - e2fsck/fsck
        - `-p`: "Preen" mode; automatically repairs any issues that can be fixed safely without human intervention.
        - `-y`: Automatically answers "yes" to all repair prompts (useful for automated scripts).
        - `-n`: Opens the filesystem in **read-only** mode to check for errors without making any changes.
        - `-f`: Forces a check even if the filesystem is marked as "clean."
- **xfs** = Extended filesystem was built for massive scale. It is the default for Red Hat Enterprise Linux (RHEL) for a reason: it's a beast in high-concurrency environments. BEWARE, you can grow xfs but you CANNOT shrink it.
    - mkfs.xfs = create xfs filesystem
        - `-L <label>`: Assigns a volume label to the filesystem.
        - `-f`: Forces the creation of the filesystem, even if a previous filesystem is detected on the device.
        - `-b size=<value>`: Specifies the block size (e.g., `4096`).
        - `-d agcount=<value>`: Sets the number of **Allocation Groups**. XFS uses these to handle parallel I/O, which is a major part of its high performance.
    - xfs_growfs = grows the size of an xfs filesystem (While `resize2fs` typically targets a device path (like `/dev/sda1`), `xfs_growfs` targets the

mount point (like `/` or `/data`). You must use lvextend, fdisk or growpart before xfs_growfs to grow the partition. XFS systems CANNOT be shrunk)
        - -n = dry run
        - -d = data section of filesystem
        - -m = max percentage of fileystem that can be occupied by subvolumes
        - -D <size> = grows filesystem to specific # of block rather than maximize space
    - xfs_repair = repair and recover XFS filesystem (You must unmount filesystem first)
        - -n = dry run
        - -v = verbose
        - -L = force log wipe (clears corrupt logs)
        - -d = direct (DANGEROUS, allows repair to run on a read-only mounted root filesystem for emergencies)
- **/etc/fstab** = configuration file (file system table) that shows what should be mounted by the system. auto-mounts filesystem on boot and unmounts on shutdown. This is where you go to configure which drives you want to mount at startup
- **/etc/mtab** = user space table that is persistent temporarily only in the current session. It's actually a symbolic link pointing to /proc/mounts. It's a configuration file (file system table) that shows what is actually mounted by the system (updates once a day)
- **/proc/mounts** = A virtual file reflecting the kernel's internal mount table. configuration file (file system table) that shows what is actually mounted by the system (generated on the fly and always up-to-date)
- **autofs** = automatically mounts and unmounts filesystems on demand (like when you cd into it or ls its contents) Auto-unmounts after a period of inactivity
    - /etc/auto.master = master config for editing autofs
    - /etc/auto.nfs = config file for adding, editing, or removing specific share
- **mount** = attach a filesystem found on a device to a directory in Linux (called the mount point)
    - -a = mounts all filesystems listed in /etc/fstab
    - -t = specifies filesystem type (ect4, xfs, btrfs… etc)
    - -v = verbose
    - -r = mount filesystem as read-only
    - -o = options
        - ro = read only
        - rw = read-write
        - remount = remount already mounted filesystem

        - nosuid = prevent users from gaining elevated permissions through files on that device by ignoring setuid and setgid
        - noexec = prevents unauthorized software execution or running compiled binaries or scripts from that partition
        - nodev = prevents interpretation of block device and limits attackers ability to interact with hardware at mount point. Prevents direct hardware interaction via device nodes
        - noatime = do not update file access times. Tells the kernel to not record when the last file access was
        - nodiratime = only update directory access times. Tell the kernel to only record directory access
        - nofail = try to mount this, but if it isn't there, just keep booting anyway
        - loop = mounts an image file as if it were a block device
- **umount** = detach a filesystem from the Linux directory hierarchy
    - -a = unmounts all filesystems except the root filesystem
    - -f = force even if the device is unreachable
    - -l = lazy detaches the filesystem from the directory tree immediately and clean up all references to the file system
    - -r = if unmount fails, it attempts to remount the filesystem as read-only
    - -v = verbose
- **NFS** = Network file system (Linux to Linux file sharing)
    - showmount = see what server is offering to be mounted
        - -e = shows export list on a specific host
        - -a = list all client hostname and the directories they have mounted
        - -d = list only directories currently mounted by remote clients
    - exportfs = server side utility used to manage the list of file systems that are shared via NFS
        - -a = exports or unexports all directories listed in /etc/exports
        - -r = re-export, synchronizes /etc/exports with the kernel's export table
        - -u = unexport or stop sharing a directory
        - -v = verbose, displays what is being shared
        - -s = status or current export list with the options applies to each
        - -f = flush = clears the kernels export table entirely (DANGEROUS)
    - /etc/exports = config file on server
    - mount = mount a remote NFS folder
        - `-t nfs`: Tells the system you are using the NFS protocol.
        - `-o soft`: If the server goes down, the client will eventually "give up" and report an error (good for non-essential data).

        - **`-o hard`**: The client will hang and retry forever until the server comes back (best for critical data).
        - **`-o intr`**: Allows you to **interrupt** a hung "hard" mount using `Ctrl+C`.
- **SMB** = server message block (file sharing that is compatible between Linux/Unix and Windows systems). In the Linux world, we use the Samba suite to talk to and host SMB shares
    - smbclient = FTP-like client used to interact with SMB shared on a target server
        - `-L <target>`: Lists the available shares on the target server (e.g., `smbclient -L 192.168.1.50`).
        - `-U <username>`: Specifies the user account to use for the connection (e.g., `smbclient //server/share -U administrator`).
        - `-N`: Used for anonymous/guest access (No password).
        - `-I <IP>`: Manually specifies the IP address of the server if name resolution fails.
        - `-M <NetBIOS_Name>`: Sends a WinPopup message to a machine.
        - `-m <protocol>`: Specifies the max SMB protocol level (e.g., `SMB2`, `SMB3`).
    - mount -t cifs = mounts a remote smb share to a local linux filesystem using the Common Internet File System (CIFS) driver.
        - `-o` (Options): This is where you pass the most important flags:
            - `username=`: The remote user.
            - `password=`: The password (avoid using this in scripts; use `credentials=` instead).
            - `credentials=<path>`: Points to a text file containing the username, password, and domain for better security.
            - `vers=`: Specifies the SMB version (e.g., `vers=3.0` or `vers=2.1`).
            - `_netdev`: A critical flag for `/etc/fstab` that tells the system to wait for the network to be up before attempting to mount the share.
    - smb.conf = smb configuration file
        - `[global]`: Sets settings for the entire server (Workgroup, Security mode, Log levels).
        - `[share_name]`: Defines a specific shared folder.

        - Key Parameters within a share:
            - `path = /srv/data`: The local directory being shared.
            - `valid users = @groupname`: Restricted access to specific users or groups.
            - `read only = no` or `writable = yes`: Controls write access.
            - `browsable = yes`: Whether the share shows up in network listings.
    - testparm = validation tool. A common troubleshooting tool. Use `testparm` to check the `/etc/samba/smb.conf` file for syntax errors before restarting the service.
    - smbpasswd = password utility. Samba maintains its own database of passwords separate from the system `/etc/shadow` file.
        - `-a <user>`: Adds a new user to the Samba database.
        - `-x <user>`: Deletes a user from the Samba database.
        - `-d <user>`: Disables a user account.
        - `-e <user>`: Enables a previously disabled account.

Common Network Services

- **IMAP4 =** (Internet Message Access Protocol 4) the standard protocol mail clients use to read and manage mail that stays stored on the server (as opposed to POP3, which downloads and removes it). Runs on port 143 (or 993 for IMAPS/TLS).
- **`chronyc`:** The command-line interface tool used to monitor and manage `chronyd`. Chrony is the default NTP implementation on modern enterprise distributions (such as RHEL, Rocky Linux, and Ubuntu Enterprise). It excels at syncing time quickly, handling intermittent network connections, and operating as a local time server.
    - **`sources`** – Lists the current NTP servers `chronyd` is connected to, showing their stratum levels, reachability, and clock offsets.
    - **`sourcestats`** – Displays detailed statistical information about clock drift and error rates for each source.
    - **`tracking`** – Shows parameters of the system clock performance, such as how much the system clock is lagging or leading compared to the reference IDs.

    - **`makestep`** – Forces the system clock to instantly "step" (jump) to the correct time if the clock offset is large, rather than slowly adjusting (slewing) it over time.
- **nginx** = Nginx was built specifically to solve the "C10K problem" (handling 10,000 concurrent connections smoothly) that plagues apache. It utilizes an asynchronous, non-blocking, event-driven loop. A small, fixed number of worker processes run continuously, and each worker handles thousands of connections simultaneously within a single thread. The catch: you cannot process dynamic content natively; must pass requests to an external gateway (like `php-fpm`).
    - -t (Test Configuration) = This is the most heavily tested Nginx flag. It parses all configuration files (like /etc/nginx/nginx.conf) and verifies that the syntax is completely correct without actually starting or restarting the live service. = Always run this *before* running a systemctl reload or systemctl restart command to ensure you won't crash a production site.
    - -T (Test and Dump) = Same as -t, but if the configuration is valid, it prints the entire configuration text to stdout.
    - -v (Version short) = Displays the currently installed Nginx version in short summary
    - -V (Version Long)= Displays the version number, the compiler version, and every single configuration parameter/module compiled into the binary.
    - -s = signal control.
        - -s reload = reloads nginx just like systemctl reload nginx
        - -s stop = forces an immediate fast shutdown of nginx processes just like systemctl stop nginx
        - `-s quit` = Triggers a graceful shutdown where Nginx waits for active client connections to finish before stopping.
- **apachectl/httpd/apache2** = Apache web server. apachectl is the cli script-wrapper. httpd is the daemon on RHEL distros. apache2 is the daemon on debian distros. Apache assigns a dedicated operating system thread or process to every single incoming HTTP connection. Apache can process php content natively which means it is excellent at hosting dynamic content. The Catch: While excellent for stability and executing embedded code (like PHP), it eats up significant RAM under heavy, concurrent traffic because creating and switching between hundreds of OS processes consumes heavy system overhead.
    - apachectl configtest = This is the Apache equivalent of nginx -t. It checks the structural syntax of /etc/httpd/conf/httpd.conf and outputs Syntax OK or points directly to the line number containing a broken directive.
    - `apachectl graceful` – The command-line script equivalent of `systemctl reload`. It signals the background binary to reload configurations without disconnecting existing web site visitors.

    - apachectl -t or httpd -t or apache2 -t = Performs the exact same configuration syntax check as apachectl configtest but runs through the primary daemon binary itself instead of the control script.
    - apachectl -M or httpd -M or apache2 -M = Lists all shared and static modules currently loaded into the Apache server engine. This is useful if an explicit feature (like URL rewriting or SSL) isn't working.
    - apachectl -V or httpd -V (or apache2 -V) (Version and Settings) = Displays the Apache server version along with the build parameters and compile settings (such as which Multi-Processing Module (MPM) is actively compiling requests).
- **mailq** = Inspects the mail queue. A monitoring tool that prints a summary of the mail messages sitting in the local outbound queue waiting to be processed by the Mail Transfer Agent (MTA, usually Postfix on modern enterprise systems). There are no flags for mailq.
- **mail/mailx** = mail is the basic mail sender to test if messages can be sent with SMTP. mailx is the more powerful brother command.
    - **`mail`**
        - `-s "Subject Line"` = Specifies the subject header string. This flag is mandatory for practical script usage.
    - **mailx**
        - `-s "Subject Line"` – Sets the email subject string (identical to `mail`).
        - `-a /path/to/file` – The critical differentiator. Attaches a physical file (like a PDF report, compressed archive, or text log) as a MIME attachment to the message body.
        - `-r "sender@example.com"` – Explicitly sets the "From" envelope address header.

Virtualization

- **VM Network Types** = unlike container networking (bridge/host/none/macvlan/ipvlan), hypervisors like KVM/QEMU expose these five virtual-machine network modes:
    - **Bridged** = the VM's virtual NIC is attached to a bridge device connected to the host's physical NIC, so the VM appears as its own device on the physical LAN with its own IP address.
    - **NAT (Network Address Translation)** = the VM shares the host's IP address for outbound traffic through a virtual router; it's the typical default for a workstation hypervisor and requires port forwarding for inbound access.

    - **Host-only/Isolated** = creates a private virtual network shared only between the host and its VMs, with no route to the physical LAN or internet — useful for isolated lab/test environments.
    - **Routed** = the host acts as a router for the VM's own subnet, forwarding packets between the VM network and the physical network without NAT translation.
    - **Open** = no firewall/NAT rules are applied to the VM's traffic at all; the host simply passes packets through, leaving all filtering to be handled elsewhere.
- **QEMU** = quick emulator, generic, open-source, lightweight machine emulator (virtualizer). By default, it uses software-based emulation which is slow, however, if you add the `-enable-kvm` then it can use the Linux kernel and operate at near-native speed. QEMU files are in the .qcow2 file format.
    - **qemu-system**
        - `qemu-system-x86_64`: The standard command for emulating 64-bit PC architectures.
            -
        - `qemu-system-aarch64`: Used for emulating 64-bit ARM architectures.
        - `qemu-system-riscv64`: Used for emulating 64-bit RISC-V architectures.
        - `-m <size>`: Sets the amount of **RAM** for the virtual machine.
        - `-smp <n>`: Configures the number of **CPUs** or cores available to the VM.
        - `-netdev` / `-device`: Used to configure the virtual **Network** adapter and backend type (e.g., bridge or NAT).
        - `-enable-kvm`: Enables **KVM (Kernel-based Virtual Machine)** support for hardware acceleration. (USE THIS EVERY TIME)
        - `-accel <type>`: Explicitly defines the accelerator to use (typically `kvm` for performance or `tcg` for software emulation).
        - **-hda <file>**: Assigns a virtual disk image as the primary hard drive.
        - **-cdrom <file>**: Loads an ISO file into the virtual CD-ROM drive for OS installation.
        - **-boot <order>**: Specifies the boot sequence (e.g., `d` for CD-ROM, `c` for Hard Disk).
        - **-drive**: A more advanced version of -hda that allows you to specify the interface type (e.g., **VirtIO** for optimized drivers).
    - **qemu-img**
        - **convert**: Changes a disk image from one format to another. This can be used to convert vmdk, raw, vdi, ISO, vhd/vhdx to .qcow2

        - **resize**: Adjusts the capacity of a virtual disk image.
        - **info**: Displays the format, size, and properties of a disk image.
        - **create**: Creates a new virtual disk image
        - **check**: Scans a disk image for consistency or corruption
        - `-f <format>`: Specifies the **input** format (e.g., `raw`, `qcow2`).
        - `-b <baseTemplate>`: specifies the backing file (the template)
        - `-O <format>`: Specifies the **output** format (used with `convert`).
        - `-p`: Displays a progress bar (useful for long conversions).
        - `-F qcow2`: Explicitly tells QEMU the format of the backing file for security.
        - **-nographic**: Disables the GUI and redirects the serial console to the terminal—essential for managing remote Linux servers.
        - **-daemonize**: Runs the VM process in the background.
        - **-snapshot**: Prevents QEMU from writing to the disk image file; all changes are lost when the VM is powered off.
- **KVM** = Kernel-based virtual machine, literally transforms the linux kernel into a Type 1 (bare metal) hypervisor allowing the host machine to run multiple isolated VMs. On most distributions, the `kvm` command is actually a **wrapper script** or an alias for `qemu-system-x86_64` that automatically includes the `-enable-kvm` flag. This means that the flags you need to know for `kvm` are identical to the essential QEMU flags used to manage VM resources. Running kvm is the same as running qemu-system-x86_64 -enable-kvm. KVM uses .qcow2 files as well.
    - **`-m <size>`**: Allocates a specific amount of **RAM** to the virtual machine (e.g., `-m 4G`).
    - **`-smp <n>`**: Sets the number of **CPUs** or cores available to the guest OS.
    - **`-name <name>`**: Assigns a display name to the VM instance for easier tracking in process lists.
    - **`-hda <file>`**: Defines the primary virtual **storage** disk image.
    - **`-cdrom <file>`**: Points to an ISO file to act as a virtual CD-ROM, often used for OS installation.
    - **`-boot <order>`**: Specifies the boot sequence, such as `d` for CD-ROM or `c` for the first hard drive.
    - **-netdev <type>**: Defines the **network** backend (e.g., user for NAT or tap for bridged networking).
    - **-device <driver>**: Specifies the virtual network hardware for the guest, often used with **virtio** for high performance.

    - `-nographic`: Completely disables graphical output, redirecting the serial console to your terminal—this is a common requirement for managing headless servers.
    - `-vnc <address>`: Sets up a VNC server for remote graphical access to the VM's display.
- **virsh** = (virtualization shell) The interactive or command-driven shell used to execute **VM operations**. While tools like `qemu-system-x86_64` act as the emulator, `virsh` acts as the management layer that talks to the **libvirtd** daemon to control KVM, QEMU, or other hypervisors. It is the primary CLI for managing VMs and libvirt management toolset. virsh doesn't have flags, it has sub-commands. If libvert stops working, virsh commands will not work.
    - virsh list --all: Displays a list of all defined virtual machines and their current status, such as "running" or "shut off".
    - virsh start <vm>: Powers on a specific virtual machine that is currently in a shut-off state.
    - virsh shutdown <vm>: Initiates a graceful shutdown by sending an ACPI signal to the guest operating system.
    - virsh destroy <vm>: Forcibly stops a running VM immediately, similar to pulling the power cable; use this if a guest is unresponsive.
    - virsh reboot <vm>: Commands the guest operating system to perform a restart.
    - virsh suspend <vm>: Pauses a running VM and saves its current state in RAM.
    - virsh resume <vm>: Transitions a suspended virtual machine back to a running state.
    - virsh edit <vm>: Opens the XML configuration file for the VM, allowing you to manually adjust resources like RAM, CPU, or Network interfaces.
        - <model type='virtio'/> = changes the network model to virtio paravirtualized drivers
        - `<target dev='vda' bus='virtio'/>` = set the target bus to paravirtualized drivers
        - <vcpu placement='static'>X</vcpu> = changes the number of cpu cores
        - `<memory unit='KiB'>XXXX</memory>` = adjusting memory
        - `<boot dev='hd'/>` or `<boot dev='cdrom'/>` = fixing boot priority
    - virsh dominfo <vm>: Provides a summary of the VM's metadata, including image properties, UUID, and resource allocation.

    - virsh autostart <vm>: Configures the specific VM to boot automatically whenever the host's physical server starts up.
    - virsh console <vm>: Provides a direct serial connection to the guest; this is a primary tool when the server is inaccessible or unreachable over the network.
    - virsh snapshot-create-as <vm> <name> = Creates a point-in-time snapshot of the virtual machine's disk and state.
    - virsh snapshot-list <vm> = Displays all available snapshots associated with a specific virtual machine.
    - virsh snapshot-revert <vm> <name> = Restores the virtual machine to the exact state captured in a previously created snapshot.
    - `virsh setmem <vm_name> 4G --config` = (Delegates 4GB of RAM to the named VM).
    - `virsh setvcpus <vm_name> 2 --config` = (Delegates 2 vCPUs to the named VM).
    - `virsh attach-disk <vm_name> <source_file> <target_device> =` add a new hard drive
    - `virsh detach-disk <vm_name> <source_file> <target_device> =` gracefully removes a disk
    - `virsh domblklist <vm_name>` = lists the hard drives attached to the VM
    - virsh dumpxml <vmname> > <vmname.xml> = export metadata to a XML file
    - `virsh define <file.xml>`, = used to register a virtual machine's metadata with the hypervisor to make it persistent, which essentially installs the VM's configuration into the system's memory.
    - `virsh migrate my_vm qemu+ssh://destination_host/system` = live migration of a running VM from one system to another
        - **`--live`**: Keeps the VM running during the move.
        - **`--persistent`**: Ensures the VM is defined on the destination host after the move.
        - **`--undefinesource`**: Removes the VM configuration from the old host once successful.
- **virt-** = While `virsh` is the "shell" for general management, the `virt-` commands are specialized tools for specific VM operations and disk image operations. If libvert stops working, virt- commands will not work.
    - **virt-install** = Purpose: Creates and provisions new virtual machines.
        - Scenario: You need to script the creation of 10 new servers with specific CPU and RAM allocations from the command line.

- **`--name OR -N`**: Sets the human-readable name of the VM.
- **`--vcpus`**: Allocates the number of virtual CPUs.
- **`--cpu`**: Defines the CPU model and features (e.g., `--cpu host-passthrough` for nested virtualization).
- **`--memory`**: Sets the RAM (e.g., `--memory 2048` or `--memory 4G`).
- **`--description`**: Adds a text note about the VM's purpose.
- **`--uuid`**: Manually sets a unique identifier (usually generated automatically).
- **`--disk`**: Defines storage. Common sub-options include:
    - `path=`: Path to the `.qcow2` or `.raw` file.
    - `size=`: Size in GB if creating a new disk.
    - `format=`: Specifies `qcow2` or `raw`.
    - `bus=`: Specifies the controller (e.g., `virtio`, `ide`, `scsi`).
    - `cache=`: Sets the caching mode (e.g., `none`, `writethrough`).
- **`--network`**: Defines the network type (e.g., `bridge=br0` or `network=default`).
    - `bridge=br0`: Connects the VM to a physical bridge on the host.
    - `network=default`: Uses the libvirt default NAT network.
    - `model=virtio`: Uses the high-performance paravirtualized driver.
    - `mac=`: Manually assigns a hardware address.
- **`--os-variant`**: Optimizes the VM for a specific OS (e.g., `ubuntu22.04`). This is critical for performance as it selects the correct **VirtIO** drivers.
- **`--location`** or **`--cdrom`**: Specifies the installation source (ISO file or URL).
- **`--graphics`**: Defines how to view the console (e.g., `vnc` or `spice`). Use `none` for text-only.
- **`--dry-run`**: runs it as a test but doesn't execute
- **`--check`**: Controls validation checks (e.g., `--check all=off` to skip validation).
- **`--location`**: Used for network installs (HTTP/FTP) or points to a kernel/initrd pair.

- **`--cdrom`**: Points to an ISO file or physical optical drive.
- **`--pxe`**: Boot from the network using PXE.
- **`--import`**: Skips the OS installation and builds a VM around an *already existing* disk image.
- **`--boot`**: Defines the boot order (e.g., `--boot cdrom,hd,network`).
- **`--cloud-init`**: Injects **cloud-init** configurations (user data, meta-data) directly into the image at first boot.
- **`--os-variant`**: CRITICAL. This tells libvirt which OS you are installing (e.g., `rhel9`, `ubuntu22.04`). It automatically optimizes the CPU, Disk, and Network drivers for that specific OS.
- **`--virt-type`**: Usually set to `kvm`, but can be set to `qemu` for pure emulation.
- **`--print-xml`**: Prints the generated XML to `stdout` instead of creating the VM (useful for template building).
- **`--graphics`**: Sets the display protocol
  - `vnc`: Standard remote desktop.
  - `spice`: Enhanced protocol with better video/audio.
  - `none`: No graphical display (text-only console).

```
#sample virt install
sudo virt-install \
  --name=test-vm \
  --ram=8092 \
  --vcpus=4 \
  --disk
path=/mnt/secondary_drive/vms/rocky10-test-vm.qcow2,size=80,bus=virtio,format=qcow2 \
  --os-variant=rhel10.0 \
  --network network=default,model=virtio \
  --graphics none \
  --console pty,target_type=serial \
  --location=/path/to/Rocky-9-x86_64-minimal.iso \
  --extra-args="console=ttyS0,115200n8 serial"
```

- **virt-clone** = Purpose: Clones an existing virtual machine.
  - Scenario: You need to duplicate a "Gold Image" or baseline image template to spin up a new production environment quickly.

        - **`--original`** (or **`-o`**): The name of the VM you want to copy.
        - **`--name`** (or **`-n`**): The name for the new cloned VM.
        - **`--file`** (or **`-f`**): The path for the new virtual disk image.
        - **`--auto-clone`**: Automatically generates names and disk paths based on the original.
    - **virt-manager** = Purpose: The primary GUI management tool for virtualization. This is being replaced with cockpit.
        - Scenario: You prefer a visual dashboard to monitor VM states, manage resources, and console into guests without using the CLI.
    - **virt-viewer** = Purpose: Provides a graphical console for a running VM.
        - Scenario: You need to interact with the OS installer or the desktop environment of a guest, but you don't need the full management features of `virt-manager`.
        - **`--connect`**: Specifies the URI of the hypervisor (e.g., `qemu:///system`).
        - **`--wait`**: Waits for the VM to start before opening the window.
        - **`<domain>`**: The name of the VM you want to view.
    - **virt-sysprep** = Purpose: Resets or "cleans" a VM disk image.
        - Scenario: You are preparing a baseline image template and need to remove unique identifiers like SSH host keys, persistent network rules, and machine IDs.
        - **`-d`** (or **`--domain`**): The name of the VM/domain to prep.
        - **`-a`** (or **`--add`**): Points directly to a disk image file if the VM isn't defined in libvirt.
        - **`--operations`**: Specifies which "cleaning" tasks to run (e.g., `ssh-hostkeys,bash-history,logfiles`). By default, it runs most safe operations.
        - **`--enable`**: Similar to operations; allows you to turn on specific cleanup tasks.
    - **virt-df** = Purpose: Displays disk usage statistics from *inside* virtual machine images.
        - Scenario: You want to see how much space is left on the partitions inside a `.qcow2` file without having to actually boot the VM.
        - **`-d`** (or **`--domain`**): Check the disks of a specific named VM.
        - **`-a`** (or **`--add`**): Check a specific disk image file directly.
        - **`-h`** (or **`--human-readable`**): Displays sizes in GB/MB instead of blocks.

    - **virt-builder** = Used for rapid, automated building of virtual machine images without running an interactive installer.
        - **`--output`** (or **`-o`**): The filename for the resulting image.
        - **`--format`**: The disk format (usually `qcow2` or `raw`).
        - **`--root-password`**: Sets the root password for the new image during the build process.
        - **`--edit`**: Allows you to modify files inside the image as it's being built.
    - **virt-viewer** = a lightweight graphical interface tool used to display and interact with the graphical console of a virtual machine. This is basically the Linux equivalent to RDP but just for your VMs on that host. This is a very useful tool (ex. virt-viewer -c qemu:///system <vm_name_or_id>)
        - **`-c qemu:///system`** (or `--connect`) — Connects to local system-level KVM virtual machines. **(Most Important)**
        - **`-f`** (or `--full-screen`) — Opens the VM window directly in full-screen mode.
        - **`-w`** (or `--wait`) — Waits for the VM to boot up if it isn't running yet, launching the window as soon as it powers on.
        - **`-r`** (or `--reconnect`) — Automatically reconnects to the VM display if the guest reboots or drops the display connection.

Containers

- **runC and containerd** = the lower-level container runtimes that sit underneath user-facing tools like Docker and Podman.
    - **`runC`** = the low-level OCI-compliant runtime that actually creates and runs a container (namespaces, cgroups, the works) based on a bundle spec. Docker and Podman both call down to a runtime like this to do the real work.
    - **`containerd`** = a higher-level daemon that manages the full container lifecycle (image pulls, storage, networking, and invoking runC) and exposes an API that Docker and Kubernetes build on top of.
- **Overlay (Docker/Podman network type)** = a multi-host container network driver that lets containers running on different physical/VM hosts communicate as if they were on the same network. It requires a key-value store or Swarm/Kubernetes to coordinate the network across nodes — distinct from the single-host `bridge` driver.
- **docker/podman** = container engines that bundle an application together with all the dependency libraries and run it in an isolated box that uses the same Linux kernel as the host. Docker is a container engine that relies on the dockerd

daemon. Because the daemon runs as `root`, anyone given permission to run a Docker command effectively has full administrative control over the entire host operating system. Podman (pod manager) is the new RHEL, secure container engine that is daemonless which improves security and reliability. In RHEL, docker is a symlink to podman. The same subcommands and flags work for both, which is why I've included them together. Each of these subcommands and flags is a shorthand way to invoke docker container. Syntax: docker [management-command] [subcommand] [flags] [arguments]

- `run` — Spawns and starts a container from a specified image.
  - `-d` (`--detach`): Runs the container in background daemon mode so it doesn't lock up your shell prompt.
  - `--name [string]`: Overrides the default random string name with a clean identifier (e.g., `--name prod-web`).
  - `--rm`: Automatically deletes the container instance and its filesystem footprint the moment it exits or stops.
  - `-p [host_port]:[container_port]` (Port Mapping) = Forwards external traffic from the physical host server into the isolated container. Remember the order: Host First, Container Second. For example, `-p 8080:80` means users hit port 8080 on your server to talk to a web app listening on port 80 inside the box.
  - `-v [host_path]:[container_path]` (Volumes) = Mounts a physical directory from your Linux server into the container. Containers are completely stateless; if you delete them, their data dies. If a question asks how to make sure database tables or web files survive a container crash or upgrade, you must use `-v`.
  - `-it` (Interactive Terminal) = Combines interactive input (`-i`) with a virtual terminal display (`-t`). Use this when you need to run a container and immediately type commands inside it. If a scenario asks you to troubleshoot a container by launching it with a `bash` or `sh` prompt, you must pair it with `-it`.
  - `:Z` (SELinux Volume Flag) = Appended to the end of a volume mount string (e.g., `-v /data:/app:Z`). If a scenario states that a developer deployed a container using Podman and it instantly crashed with a `Permission Denied` error when trying to write to a host folder, look for the missing `:Z` flag. It tells Podman to automatically re-label the host directory's security context so the rootless container process can access it.

- `-e` (`--env`) = Configuration setting injected inside. Injects an Environment Variable (a key-value pair) directly into the container's internal operating environment at startup. (ex. `docker -e APP_ENV=production`)
- `–env-file` = allows you to specify environment variables from a .env file instead of typing them out manually with the -e flag.
- --net/--network = tells docker what network settings to configure. When you launch a container, you choose its networking model using the `--network [driver]` flag. You must know when to deploy each of these five standard drivers:
    - `bridge` (The Default) = Creates a private, internal virtual network on the Linux host (usually named `docker0` or `podman0`). Containers on this bridge get private IP addresses (like `172.17.0.X`) and are completely hidden from your physical corporate network. This is the default choice for standard multi-tier applications. To let outside traffic hit a container on a bridge network, you must use port mapping (`-p`) to punch a hole through the host's firewall.
    - `host` (Zero Isolation) = Strips away the network sandbox entirely. The container shares the exact same network stack, network interfaces, and IP address as the physical Linux server. If a containerized app runs on port `80` using the `host` driver, it binds directly to the physical server's port `80`. Port mapping (`-p`) is ignored. Choose this driver if you need maximum network performance or if the container needs to monitor raw host network traffic.
    - `none` (The Air-Gapped Box) = Completely disconnects the container from the network. It gets a local loopback interface (`127.0.0.1`) but has no external network interface and no route to the internet. Choose this driver for high-security processing tasks—like a batch script calculating sensitive financial tables or hashing passwords—where you want to guarantee zero data exfiltration risks.
    - `macvlan` (Direct Network Assignment) = Makes the container appear as a distinct physical device on your corporate router. It gives the container its own unique hardware MAC address and pulls a real IP address straight from your physical local network DHCP server. Used when

migrating legacy enterprise applications into containers that strictly require their own distinct, un-routed network identity on the corporate LAN.

- `ipvlan` (The Lightweight Alternative) = Very similar to `macvlan`, but with one major twist: all containers share the exact same hardware MAC address as the physical host network card, while maintaining completely distinct IP addresses. Choose `ipvlan` over `macvlan` if you are running thousands of containers on a single host and your physical corporate network switches start crashing because they cannot handle thousands of artificial MAC addresses hitting their routing tables.

- `compose` = a tool used to define, launch, and manage multi-container applications on a single host using a declarative YAML manifest file (typically `compose.yaml` or `docker-compose.yml`).
  - `up` = Creates, configures, and starts all containers, networks, and volumes defined in the Compose manifest.
  - `down` = Stops running containers and removes the containers, networks, and default resources created by `up`.
  - `logs` = Displays log output generated by services in the stack.
  - `ps` = Lists the current status of all containers belonging to the Compose project.
  - `exec` = Executes an interactive or ad-hoc command inside a running service container.
  - `build` = Builds or rebuilds container images defined under the `build:` key in the Compose manifest.
- `ps` = Lists currently active, running containers.
  - `-a` (`--all`): Crucial for troubleshooting. Lists all containers, including those that have crashed, exited, or paused.
  - `-q` (`--quiet`): Returns only the hexadecimal container IDs, which is heavily used in Bash loop automation scripts.
- `exec` = Spawns a brand-new process *inside* an already running container container to see why it is misbehaving.
  - `-it`: Interactive (`-i`) + Pseudo-TTY terminal (`-t`). This combination is used to jump directly into a live terminal shell inside the container for diagnostics (e.g., `docker exec -it prod-web /bin/bash`).

        - `-u` (or `--user`) = Forces the command to run under the security context of a specific user account or User ID (UID) inside the container.
    - `logs` Pulls the stdout and stderr data streams straight from the container application.
        - `-f` (`--follow`): Streams the logs live to your display terminal in real time as events occur.
    - `stop` / `docker start` — Gracefully stops an application container via `SIGTERM` or wakes up an existing, stopped container instance.
    - `rm` — Deletes a stopped container template from system storage.
        - `-f` (`--force`): Forces the deletion of a container even if it is currently running (uses `SIGKILL`).
    - `inspect [name]`**=** This dumps a massive JSON metadata configuration manifest for the container. On the exam, this is the tool you use to hunt down a container's internal IP address, mount paths, environment variables, or port bindings from the host side.
    - `pull` — Fetches an image template from an upstream container registry (e.g., `docker pull alpine:latest`).
    - `build` — Compiles a textual configuration script into a fully operational local image.
        - `-t [name:tag]`: Applies a descriptive moniker tag to the image architecture (e.g., `-t cache-server:v1.2`).
        - `.` (The Trailing Dot): Points the builder to your current working directory to look for the configuration file.
    - tag = a tag is an human-readable alias pointing to the docker image. Helps with version control. (ex. docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG])
    - `rmi` — Removes a cached local image from host memory storage.
    - `image prune` — Sweeps away unreferenced "dangling" image layers. Adding the `-a` flag tells Docker to wipe out all unused images to rapidly recover host drive capacity during a storage crisis.
    - `system prune =` deletes all stopped containers, all unused container networks, and all unused images.
        - `-a` (or `--all`): By default, `system prune` leaves behind named, valid images. Adding `-a` forces it to delete *every single cached image* that isn't actively running a live container this second.

        - `-f` (or `--force`): Bypasses the warning screen. Normally, the system prompts you with `Are you sure? [y/N]`. The `-f` flag makes it completely silent, which is a common requirement in automated Bash or CI/CD pipeline scripts.
    - `container prune =` deletes every container instance that is currently in a `stopped`, `exited`, or `paused` state.
    - `volume` = subcommand for volume management tool
        - `prune =` deletes any dedicated storage volume that is not currently mounted inside an existing container.
        - `create [name]`= Provisions a brand-new, empty named volume wrapper on the host. Run this *before* deploying a database to ensure your dedicated storage pool is ready to go.
        - `ls` = Lists all named volumes currently sitting on the host machine. Used to audit storage sprawl, check for orphaned data, or verify that a volume actually exists.
        - `inspect [name]`= Dumps a detailed JSON metadata manifest showing the configuration of the volume. This is the exact tool you use if an exam question asks how to locate the exact physical path (`Mountpoint`) where volume data lives on the underlying Linux hard drive.
        - `rm [name]`= Manually deletes a single specified volume from the disk. Keep in mind that this command will fail if the volume is still actively attached to a container, even if that container is stopped.
    - `network` = docker network management toolbelt. Just like volumes and containers, networks are managed as standalone objects using the `docker network` (or `podman network`) subcommand framework:
        - `prune` = deletes all custom isolated networks (like user-defined bridge or overlay networks) that don't have at least one container actively joined to their subnet configuration.
        - `ls` = Lists all active virtual networks on the Linux host, revealing which drivers are currently initialized.
        - `create --driver [type] [subnet_name]`= Provisions a custom, isolated virtual switch. (ex. docker network create --driver bridge secure-net) Containers attached to a *custom* user-defined bridge gain automatic DNS service discovery. They can talk to each other using their container names (e.g., `ping database-server`) instead of volatile, shifting IP addresses.

        - `inspect [subnet_name]`= Dumps detailed JSON network maps. Use this to find the specific subnet gateway or the exact internal IP address assigned to a running container.
        - `rm [name]` / `docker network prune` = Deletes a specific network interface, or clears out all unused virtual network allocations to tidy up host system routing loops.
    - `-p [host_port]:[container_port]` — Network Port Mapping. Maps an external port on your physical server to an internal port inside the container block (e.g., `-p 8080:80` allows public users to hitting port 8080 on the server to reach a web server running inside the container on port 80).
    - `-v [host_path]:[container_path]`= Volume Mapping. Binds a physical directory on your local host to an operational directory inside the container's virtual partition. This ensures your application data persists safely even if the container is deleted.
    - --privileged = DANGEROUS, gives container root access to the entire system. Privileged mode essentially gives the guest a master key to the entire building.
    - `--cap-add` and `--cap-drop` = grants standard unprivileged container a specific admin permission it needs to do its job. (principle of least privilege)
    - There are five critical directives used inside a `Dockerfile` to configure a container's assembly template. You must know how to interpret them:
        - `FROM` — The mandatory starting line. Defines the base operating system layout or image template used to seed the container layer (e.g., `FROM rhel:9`).
        - `RUN` — Executes commands *during the creation build phase* to modify the file template (e.g., installing a application via `RUN apt install -y nginx`).
        - `USER` — Sets the default Linux user account context that executes all future commands inside the container, enforcing the security principle of least privilege.
        - `CMD` vs. `ENTRYPOINT` (CompTIA loves to compare these):
            - `ENTRYPOINT`: Specifies the core executable or binary command that the container is built to run (e.g., `ENTRYPOINT ["/usr/sbin/nginx"]`). It cannot be easily overridden by runtime inputs.

- `CMD`: Specifies the *default parameters* passed directly to the `ENTRYPOINT`. Unlike `ENTRYPOINT`, the values in `CMD` can be completely overwritten by typing parameters at the end of a `docker run` execution.

Authentication, Authorization & Accounting

- **realm** = used to manage enrollment in enterprise domains such as Active Directory. Part of the realmd service. By default it uses SSSD, but can be specified to use Winbind. Syntax: realm [subcommand] [domain] [flags]
  - discover = Uses DNS SRV records to locate the Active Directory Domain Controllers, identifies the domain type, and prints out a list of missing system packages required to join. Syntax: `realm discover corp.example.com`
  - join/leave = joins or leaves a domain. Modifies `/etc/sssd/sssd.conf` or Samba files, authenticates/deauthenicates against the network domain controller, creates or removes a machine account, and updates `/etc/nsswitch.conf`. Syntax: `realm join corp.example.com [flags]`
    - `-U` or `--user=USER =` By default, `realm` tries to connect to the domain using the local system username. Because standard Linux users don't have permission to create computer accounts in Active Directory, you use this flag to specify a valid Domain Administrator account. Example: `realm join corp.example.com -U Administrator`
    - `--client-software=SOFTWARE =` If your server has both SSSD and Winbind installed, `realm` will guess which one to use. This flag lets you explicitly dictate the client software stack. Example: `realm join corp.example.com --client-software=winbind` *Exam Note:* The two valid values for this flag are `sssd` (default) or `winbind`.
    - `--install=PATH =` By default, if `realm discover` finds that a package like `sssd` or `samba-common-tools` is missing, `realm join` will automatically run your system package manager (`dnf` or `apt`) to download it. If you are in a secure, air-gapped environment without internet access, you can turn this off. Example: `realm`

`join corp.example.com --install=/` *(or setting it to a specific directory where packages are cached).*
        - `--remove` (Used only with `leave`) = If you run `realm leave`, it stops local domain logins but leaves the computer placeholder object inside the Active Directory database. Adding this flag tells the domain controller to completely shred the computer account from the network directory. Example: `realm leave corp.example.com --remove -U Administrator`
    - `list` = Displays detailed configuration information about the realm the server is currently attached to, including the client software being used and permitted login paths.
    - `deny` = Blocks domain users from logging into the machine.
        - `-a` or `--all =` Locks down the machine. No domain users can log in via SSH or console anymore (local accounts like `root` remain untouched). Example: `realm deny --all`
    - `permit =` Creates an explicit "Allow List" to grant exceptions after a `deny --all` command has been run. Example (Allow a specific user): `realm permit alice@corp.example.com`
        - `-g` or `--groups` = allow a whole group. Example (Allow a whole security group): `realm permit -g sysadmins`
- **kinit**: Fetches a new Ticket-Granting Ticket (TGT) from Kerberos server. It will prompt you for your password.
- **klist**: Shows your current kerberos tickets, their issue times, and their expiration times.
- **kdestroy**: Wipes your local kerberos ticket cache (effectively logs you out of the Kerberos realm).
- **ldapsearch** = queries the LDAP (port 389 or 636 for LDAPS) directory and returns information. *Example:* `ldapsearch -x -b "dc=company,dc=com" '(uid=jdoe)'`
    - `-x` = uses simple authentication (Use most of the time)
    - `-b` = sets the base search path in the directory tree
    - Attributes to know for LDAP:
        - `uid` (User ID): `uid=jdoe` (The specific username)
        - `cn` (Common Name): `cn=John Doe` (The actual name or object name)
        - `ou` (Organizational Unit): `ou=Engineering` (The department folder or group)

        - `dc` (Domain Component): `dc=company, dc=com` (Breaks up the network domain `company.com`)
- **pkexec** = polkit execution. While with sudo, you give a user permission to run the entire `systemctl` binary as root. It's all-or-nothing. With polkit, when a non-root user clicks "Restart" on a desktop menu. Polkit steps in and safely permits *only* that specific reboot action to execute, without giving the user a root shell or full access to other system services. Polkit is most heavily used in the GNOME gui. (ex. pkexec timedatectl set-timezone America/New_York)
- **journalctl** = queries the systemd journal logs.
    - -u <service> = filters logs for a specific unit or service
    - -f = follows the log in real time as new entries are added (USE THIS MOST OF THE TIME)
    - -b = shows logs only from the current boot
    - –since=”time” = filters logs starting from a specific time
- **logger** = command interface to the rsyslog system module, allowing you to add custom entries to log files like `/var/log/syslog` or `/var/log/messages`
    - -p <facility>.<severity>: (High Exam Value) Specifies the priority of the message. *Example:* logger -p auth.notice "Unauthorized access attempt detected"
    - -t <tag>: Marks every log line with a specific string tag so you can easily grep for it later. *Example:* logger -t BACKUP_SCRIPT "Backup completed successfully"
    - -f <file>: Logs the contents of a specific text file rather than a typed message.
    - -i: Automatically logs the Process ID (PID) of the logger process with each line, which is useful for tracking execution scripts.
- **logrotate** = utility for preventing your system hard drives from filling up with massive text logs. It automatically breaks logs up, compresses them, and throws away old ones based on age or file size. The basic syntax is: `logrotate [flags] <config_file>` (usually `/etc/logrotate.conf`), but can also be /etc/logrotate.d/
    - `-d` / `--debug`: (Extremely High Exam Value) Enters debug mode. It processes the log files and spits out exactly what it *would* do on screen, but does not actually modify any files. This is the ultimate tool for testing if a new rotation rule works without messing up current logs.
    - `-f` / `--force`: (High Exam Value) Forces `logrotate` to rotate files immediately, even if the age or size limits specified in the configuration file haven't been met yet. This is perfect for executing changes right after writing a new config rule to verify it works.

    - `-v` / `--verbose`: Turns on verbose mode, displaying detailed progress messages on screen during execution.
    - `-s <statefile>` / `--state <statefile>`: Tells logrotate to use an alternate state file instead of the default `/var/lib/logrotate/status`. The state file is what logrotate uses to remember the exact date and time it last rotated a specific log.
- **auditctl** = used to manage and modify audit rules and view live rules and also to pass a rule directly to the command line. If you run a rule this way, it will get overwritten upon reboot. For permanent rules, you must go to /etc/audit/rules.d/audit.rules and add your rule to this file and you must apply them using the augenrules --load command.
    - -l: (High Priority) Lists all currently active rules loaded in the running kernel.
    - -D: Deletes all active rules from the running kernel memory.
    - -s: Reports the current Status of the audit system (enabled/disabled, failure flags, rate limits).
    - -R <file>: Reads and executes a specific file containing audit rules (temporary execution).
    - `-w <path>`: Specifies the file or directory Watch path.
    - `-p <permissions>`: Filters by Permissions. Triggers are:
        - `r` = read
        - `w` = write
        - `x` = execute
        - `a` = attribute change (permissions, ownership, etc.)
    - `-k <key_name>`: Assigns a custom search Key (string label) to group the logs.
- **augenrules** = compiles all rule changes and additions from the /etc/audit/rules.d/ to the master /etc/audit/audit.rules file which loads them directly into the kernel permanently
    - `--load`: Compiles all separate `.rules` files sitting inside `/etc/audit/rules.d/` into the final master `/etc/audit/audit.rules` file, then immediately reloads them into the kernel.
    - `--check`: Checks if any modifications were made inside `rules.d/` that haven't been compiled into the master rule file yet.
- **ausearch** = The primary tool used to query the raw, messy `/var/log/audit/audit.log` file based on specific criteria. Useful for forensic audits

    - `-k <key_name>`: (High Priority) Filters logs to show only events matching a specific rule Key label.
    - `-m <message_type>`: Filters by audit Message type (e.g., `USER_LOGIN`, `ADD_USER`, `CONFIG_CHANGE`).
    - `-ts <time>`: Filters events by Time Start. Accepts keywords (`today`, `yesterday`, `this-week`) or absolute timestamps (`06/11/2026`).
    - `-te <time>`: Filters events by Time End.
    - `-u <uid_or_username>`: Searches for events triggered by a specific User.
    - `-p <pid>`: Searches for events triggered by a specific Process ID.
    - `-i`: Interprets numeric values (like UIDs, GIDs, and timestamps) into human-readable text (e.g., changing `uid=1000` to `uid=jdoe`).
- **aureport** = reports on audit logs and spits out high level summaries and stats for compliance auditing. It pulls logs from the `/var/log/audit/audit.log` file and gives you an executive summary and high-level overview of them
    - `-k`: Generates a summary report grouped by your custom rule Keys.
    - `-u`: Generates a User authentication report (successful/failed logins, system access).
    - `-f`: Generates a File modification report (which watched files were manipulated).
    - `-p`: Generates a Process execution report.
    - `-x`: Generates an Executable report (identifies which software binaries were run).
    - `-s`: Generates a Syscall event report.
    - `--summary`: Drops tabular details and instead prints a simple mathematical tally of events (e.g., "Total failed logins: 42").

Firewalls

- **firewall-cmd** = command-line frontend tool for the firewalld which is the default firewall management tool for **RHEL** distros. firewall-cmd is zone-based, so changes are applied to zones instead of rule-based ACLs. Unlike older legacy tools like raw `iptables`, which require you to understand complex chain tables and packet filtering rules, `firewall-cmd` simplifies network security using Zones and Services.
    - Zones: groups based on the level of trust you have in the connected network interface.

        - `public`: The default zone for new interfaces. You do not trust other computers on the network; only selected incoming connections are allowed (e.g., SSH).
        - `drop`: Any incoming network packets are dropped immediately with no response sent back. Good for hiding from port scans.
        - `block`: Any incoming connections are rejected with an ICMP "host-prohibited" message sent back.
        - `internal` / `home` / `work`: Used for networks you mostly trust. More services are open by default because you assume other machines on the local network aren't malicious.
        - `trusted`: All network connections are accepted.
    - `--reload`: (High Priority) Necessary after you make a change permanent. Drops the runtime memory cache and reads the permanent XML configuration files from disk, activating all rules made with the `--permanent` flag without dropping active network connections.
    - `--permanent` = (High Priority) Rule will survive reboot. Written to the XML files on disk, but inactive in live memory until the --reload flag is run
        - `--add-service=ssh`: Permanently allows SSH connections (Requires a reload to take effect).
    - `--state`: Checks if the firewall daemon is actively running or stopped.
    - `--get-active-zones`: Shows which network interfaces (like `eth0` or `wlan0`) are assigned to which zones.
    - `--list-all`: (Extremely High Exam Value) Shows every open port, allowed service, and interface bound to the current default zone.
    - `--zone=<zone_name>` = apply to only a specific zone
        - `--list-all`: Shows the rules for a *specific* zone instead of the default zone.
    - `--add-service=http`: Instantly allows web traffic to hit the server (Runtime only).
    - `--remove-service=ftp`: Blocks FTP access.
    - `--remove-port=21/tcp`: Closes port 21 temporarily unless the --permenant flag is run
    - `--add-port=8080/tcp`: Opens TCP port 8080 temporarily.
    - `--add-masquerade`: Turns on outbound NAT
    - --add-forward-port=port=80:proto=tcp:toport=8080:toaddr=192.168.1.50 = adds port forwarding

    - `--add-rich-rule='...'`: The core flag telling the system that an advanced configuration string is inside the single quotes. (ex. sudo firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source address="192.168.10.25" port port="5432" protocol="tcp" accept')
    - --set-default-zone=<zone> = sets the default zone to something else. The default for most distros is public. (I WOULDN'T CHANGE THIS)
- **ufw** = uncomplicated firewall. Default firewall utility on ubuntu/debian distros. It uses a simple, rule-based intuitive syntax designed to let administrators secure a machine in seconds without having to understand complex networking concepts. ufw is a sequential ACL ruleset. It has no "runtime vs permanent" distinction like firewall-cmd.
    - `ufw status`: Checks if the firewall is active or inactive, and lists current rules.
        - `numbered`: (Extremely High Exam Value) Lists all active rules with an assigned index number next to them (e.g., `[ 1]`, `[ 2]`). This is the mandatory first step before deleting rules.
        - `verbose`: Shows additional information, such as the default policies and logging state.
    - `ufw enable`: Turns the firewall on and hooks it into system boot.
    - `ufw disable`: Shuts the firewall down completely.
    - `ufw reset`: Wipes out all custom rules and resets the firewall back to factory defaults. (DANGEROUS)
    - `ufw allow <port or service>`: Opens a port. *Examples:* `ufw allow ssh` or `ufw allow 80/tcp`
    - `ufw deny <port or service>`: Explicitly blocks a port or service. *Example:* `ufw deny 21/tcp`
    - `ufw limit <port or service>`: (High Security Value) Automatically throttles connections if an IP tries to connect too many times in a short window. CompTIA tests this specifically as a defense against SSH brute-force attacks.
    - `ufw delete <number>`: Used right after running `ufw status numbered`. It lets you drop a rule instantly without typing out a long line of syntax. *Example:* `ufw delete 2`
- **nft** = command for interacting with the nftables. While `firewall-cmd` and `ufw` are friendly wrappers, `nft` is the raw, unmasked language of the Linux kernel's packet filter. You will usually use `firewall-cmd` and `ufw` to edit nftables.

- address families: nftables separates different types of network protocols into distinct namespaces called Address Families. You must recognize these three major families on the exam
  - inet = both IPv4 and IPv6
  - ip = IPv4 traffic only
  - ip6: Applies strictly to IPv6 traffic.
  - bridge = traffic passing through a network bridge (Layer 2 switching such as virtualization/containers)
- subcommands:
  - list = lists what you specify
    - ruleset = Ultimate Priority Command: This is the most heavily tested `nft` command on the Linux+. It dumps the entire active firewall configuration out to the terminal screen. If a question asks how to verify the raw, underlying kernel firewall configuration regardless of what frontend is running, this is your answer.
    - tables = Lists only the high-level tables currently active in the kernel, sorted by their address families (e.g., `table ip filter`, `table inet firewalld`).
  - flush ruleset = deletes every single table, change and rule from the kernel memory (DANGEROUS)
  - delete table <family> <name> = removes a specific table and all chains/rules bundled inside it (e.g., `sudo nft delete table inet firewalld`). (DANGEROUS)
  - `add rule`: Instructs the kernel to append a new rule to the end of a specific chain.
  - `delete rule`: Removes a rule from a chain (usually requires referencing the rule's internal handle number).
- Match Selectors (The Criteria); These keywords are used within a rule to identify the specific type of traffic you want to target.
  - `ip saddr`: Matches the Source IP Address (e.g., `ip saddr 192.168.1.50`).
  - `ip daddr`: Matches the Destination IP Address.
  - `tcp dport`: Matches a TCP Destination Port (e.g., `tcp dport 22`).
  - `udp dport`: Matches a UDP Destination Port (e.g., `udp dport 53`).
- Verdicts (Final Action)

        - `accept`: Allows the packet to pass through.
        - `drop`: Silently discards the packet without sending a notification to the sender.
        - `reject`: Blocks the packet and explicitly tells the sender the connection was refused by sending an ICMP packet back.
- **iptables** = ACL for firewall rules in Linux. Replaced by nftables in modern distros. Unlike nftables, iptables can ONLY handle IPv4. It structures its rules inside Tables, which contain Chains, which contain Rules. (ex. sudo iptables -t filter -A INPUT -p tcp --dport 22 -s 10.0.0.5 -j ACCEPT)
    - The 3 Main Tables to Know (Used with the -t flag)
        - `filter`: The default and most common table. If you don't specify a table in your command, `iptables` assumes you are working here. It is used for basic blocking and allowing of traffic.
        - `nat` (Network Address Translation): Used when your server acts as a router, redirecting packets to different networks or altering source/destination IPs (e.g., port forwarding).
        - `mangle`: Used for specialized packet alteration (like modifying Quality of Service [QoS] bits in the packet header)
    - Chains are the specific traffic lanes where packets are caught:
        - `INPUT`: Evaluates packets heading *into* the server from the network.
        - `OUTPUT`: Evaluates packets created *by* the server heading out to the network.
        - `FORWARD`: Evaluates packets passing *through* the server (routed from one interface to another).
        - `PREROUTING`: (NAT table only) Alters packets the split-second they arrive, *before* a routing decision is made.
        - `POSTROUTING`: (NAT table only) Alters packets right *before* they leave the server.
    - When a packet matches a rule, `iptables` hands it off to a Target (the action). You must recognize these four targets:
        - `ACCEPT`: The packet is allowed to pass through the firewall.
        - `DROP`: The packet is silently ignored. The firewall drops it completely and sends no response back to the sender (makes the port look closed/hidden).
        - `REJECT`: The packet is blocked, but the firewall sends an error packet back to the sender (explicitly stating the connection was refused).

        - `LOG`: The packet is noted in the system logs (`/var/log/messages`), and then the firewall continues reading the next rule down the line.
    - Flags:
        - `-A`: Appends a rule to the *end* of a chain. (Adding a rule)
        - `-I`: Inserts a rule at a specific line number (e.g., `-I INPUT 1` puts it at the absolute top of the list).
        - `-D`: Deletes a specific rule from a chain.
        - `-L`: Lists all active rules currently running.
        - `-F`: Flushes (wipes clean) all rules from the selected table. (DANGEROUS)
        - `-p`: Filters by network Protocol (`tcp`, `udp`, `icmp`).
        - `-s`: Filters by Source IP address or subnet (e.g., `-s 192.168.1.50`).
        - `-d`: Filters by Destination IP address.
        - `--dport`: Filters by Destination Port number (e.g., `--dport 22`). *Note: You must use `-p tcp` or `-p udp` before you can use this flag.*
        - `-j`: Sets the Jump target (the action to execute, like `-j ACCEPT`).
        - `-m set --match-set`: This is the trigger flag. It tells `iptables` to look at your `ipset` named **"blocklist"** and see if the incoming Source (`src`) IP matches anything inside it.
- **ipset** = utility to interact with massive lists of IP addresses in iptables. **`ipset`** is a companion utility to legacy `iptables` (and supported by `firewalld`). It is designed to solve one massive problem: firewall performance scale. If you try to block 10,000 malicious IP addresses manually using individual `iptables` or `firewall-cmd` rules, your server's network performance will crawl to a halt. `ipset` fixes this by allowing you to store thousands of IP addresses, subnets, MAC addresses, or ports inside a highly efficient hash table structure in the Linux kernel memory.
    - `ipset create <set_name> hash:ip`: Creates an empty storage container. (Use `hash:net` if you are storing entire subnets instead of single IPs). *Example:* `ipset create blocklist hash:ip`
    - `ipset add <set_name> <IP>`: Drops a new IP into your storage container. *Example:* `ipset add blocklist 203.0.113.42`

    - `ipset list`: (Most Heavily Tested) Displays all active ipsets in memory and every IP address currently hiding inside them.
    - `ipset save` / `ipset restore`: Just like `iptables`, `ipset` lives in volatile memory. You must use `ipset save > /path/to/file` to back it up to the disk so it survives a reboot.

OS & Account Hardening / Secure Remote Access

- **sealert** = a human-readable SELinux troubleshooting tool (GUI or `sealert -a /var/log/audit/audit.log` on the CLI) that reads raw AVC denial messages and explains, in plain English, what was blocked and suggests the exact `audit2allow` or `semanage` fix to resolve it.
- **sshd_config Directives** = key security-relevant settings inside `/etc/ssh/sshd_config` that control how the SSH daemon accepts connections.
    - **`PermitRootLogin`** = controls whether the `root` account can log in directly over SSH. Best practice is `no` — administrators should log in as a normal user and `sudo` instead.
    - **`AllowUsers`** = an explicit whitelist of usernames permitted to log in via SSH; anyone not listed is denied, regardless of their local password.
    - **`AllowGroups`** = same concept as `AllowUsers`, but grants SSH access to every member of the specified group(s).
- **Secure Boot / UEFI** = a UEFI firmware feature that only allows the system to boot bootloaders and kernels signed by a trusted key, preventing unsigned or maliciously modified boot code (like a bootkit) from loading before the OS even starts.
- **chmod** = change file mode (read, write, or execute permissions for users, groups, or others). Syntax: chmod [options] [mode] [file_or_directory]
    - -R = recursively changes permission on a directory and everything inside it (DANGEROUS!)
    - Octal Method (The Sums): * `r=4`, `w=2`, `x=1`
        - Format: `[Special Bits][User][Group][Other]`
        -
        - Example `754`: User(4+2+1=7), Group(4+0+1=5), Other(4+0+0=4)
    - Symbolic Method (The Sentence): `[Who][Operator][Permission]`
        - Who: `u`, `g`, `o`, `a`
        - Operator: `+`, `-`, `=`
        - Example: `chmod u+x` (Adds execute to owner)

    - Special Bits (The 4th Digit / Advanced): Prerequisite, in order for special bits to work, you need to use chown to change the owner of the file. Then, the special bits can be used to run the file as that new file owner.
        - SUID (`u+s` or `4`): Runs with file owner's permissions. (ex. chmod 4777 /home/shared_project or chmod u+s /home/shared_project). When securing a system, you will often remove the s bit (u-s) because it is the #1 attack vector for privilege escalation attacks.
        - SGID (`g+s` or `2`): Inherits group from parent directory.(ex. chmod 2777 /home/shared_project or chmod g+s /home/shared_project). When securing a system, you will often remove the s bit (g-s) because it is the #1 attack vector for privilege escalation attacks.
        - SGID (`g+s` or `2`): Inherits group from pare
        - Sticky Bit (`+t` or `1`): Prevents deletion by non-owners in shared directories (e.g., `/tmp`). (ex. chmod 1777 /home/shared_project or chmod +t /home/shared_project)
    - Equivalency: `chmod a=rwx script.sh` is identical to `chmod 777 script.sh`.
- **chown** = changes who owns a file or directory. Syntax: chown [options] [owner][:group] [file_or_directory]
    - `-R` (Recursive): Applies the change to the directory and all of its contents (files and subdirectories). (VERY DANGEROUS!)
    - `-v` (Verbose): Prints a status message for every file processed. This is very helpful when running recursive changes to ensure you know what is being modified.
    - `-c` (Changes): Like verbose, but only reports when a change is *actually made*. It skips files that already had the correct ownership.
    - `-h` (No-dereference): If the file is a symbolic link, this flag changes the ownership of the link itself, rather than the file the link points to. This is a common "trick" question on certification exams.
    - chown user file.txt = change the owner of a file
    - chown :group file.txt = changes the group assigned to a file
    - chown user:group file.txt = changes both the owner and group at once
- **chgrp** = change group ownership of a file or directory. While `chown` (Change Owner) can handle both the user and the group simultaneously, `chgrp` is a specialized tool that focuses exclusively on managing group-based permissions. You can use chown to do everything chgrp does and most people do. Basically the only reason this command exists is to make sure you don’t accidentally

change the file's owner. It is "idiot-proof" Syntax: chgrp [options] [group_name] [file_or_directory]

  - `-R` (Recursive): This is essential. It applies the group change to the specified directory and every file/subdirectory inside it. *Example:* `chgrp -R developers /var/www/html` (Changes the group of the entire web folder to `developers`).
  - `-v` (Verbose): This tells the system to print a message for every file it processes. This is helpful when you are running a recursive change and want to see what is happening in real-time.

- **setfacl** = set file access control list. Discretionary Access Control (DAC). While chmod gives you three options, setfact lets you create custom, granular rules for specific users or groups without changing the file's primary ownership. Syntax: setfacl [options] [rule] [file_or_directory]
  - Rule Format: `[u|g|o|m]:[name]:[permissions]`
    - `u` (user), `g` (group), `o` (others), `m` (mask)
    - `name` is the username or group name.
    - `permissions` are standard `rwx` letters.
    - Example: setfacl -m u:alice:rw file.txt (grants r/w access to alice)
  - `-m` (Modify): Modify the ACL of a file or directory. (Used 90% of the time).
  - `-x` (Remove): Remove specific entries from the ACL.
  - `-b` (Remove All): Remove all extended ACL entries, resetting the file to base permissions.
  - `-R` (Recursive): Apply the change to all files/subdirectories inside a folder.
  - `-d` (Default): Set Default ACLs. This is a "must-know"—it tells a directory that any *new* files created inside will automatically inherit these specific permissions.
- **getfacl** = (Get File Access Control List) is the companion command to `setfacl`. While `setfacl` is your "pen" to write permissions, `getfacl` is your "magnifying glass" to inspect exactly what those permissions are. Syntax: getfacl [options] [file_or_directory]
  - `-R` (Recursive): Displays the ACLs for a directory and every single file/subdirectory inside it. This is vital when auditing a directory tree where permissions might be inconsistent.
  - `-p` (Physical): By default, `getfacl` follows symbolic links to the target file. The `-p` flag disables this, showing you the ACL of the link itself rather than the file it points to.
  -

- **umask** = (User File Creation Mask) is the system's automatic gatekeeper for defense in depth. It defines the default permissions a file or directory receives the moment it is created. When you create a file, the system doesn't give it "open" permissions by default. The `umask` acts like a subtraction mask—it tells the system which permissions to *strip away* from the default maximums. umask never gives permissions, it removes them. Syntax: umask [options]
    - The `umask` is a three-digit octal number that you subtract from the default maximum.
        - Example: If your `umask` is `022`
            - For a Directory: `777 - 022 = 755`
                - Result: `rwxr-xr-x` (Owner can do anything; others can only read and enter).
        - For a File: `666 - 022 = 644`
            - Result: `rw-r--r--` (Owner can read/write; others can only read).
    - -S = check it with a symbolic link (the easy way)
- **sudo** = executes a command as another user (usually the root/superuser). The list of users and groups approved to run sudo are found in `/etc/sudoers` and `/etc/sudoers.d` Syntax: `sudo [options] [command]`
    - -i = simulates initial login for root user by loading root's environmental variables. This is the modern alternative to sudo su
    - -s = opens shell as root but keeps current user's environment
    - -u <user> = runs command a specific user rather than root
    - -l = Lists the current user's explicitly allowed privileges. If you want to know what commands you are allowed to run on a machine, type this first.
    - -k = "Kills" or clears your cached password timestamp. By default, once you type your password for `sudo`, it remembers you for about 15 minutes so you don't have to keep re-typing it. `-k` resets that timer instantly.
    - `-v` (Validate): Updates the `sudo` "timestamp" (the timer that keeps you authenticated for 15 minutes) without actually running a command. It's useful to "refresh" your access before starting a long task.
- **visudo** = the native safety-net utility used to edit the `/etc/sudoers` and `/etc/sudoers.d` files and its drop-in companions. It does three things to protect the system: File Locking: It places a lock on the `sudoers` file. If another administrator tries to run `visudo` at the exact same time, they are blocked. Temporary Buffering: It doesn't actually open `/etc/sudoers`. Instead, it creates a hidden, temporary copy of the file (usually inside `/etc/` or `/var/tmp/`) and opens *that* copy in your text editor. Syntax Compilation: When you save and exit

the editor, `visudo` parses the temporary file line-by-line. If the syntax is perfect, it cleanly overwrites the real `/etc/sudoers` file.
    - `-c` (Check-only Mode): Runs a clean syntax, ownership, and permission audit on your active files. It skips loading a text editor completely, outputting an `OK` or a specific line error.
    - `-f [path]` (File Target): Redirects `visudo` to apply its protective buffer and compiler to an alternative file target (like your drop-in chunks) instead of the default `/etc/sudoers` file.
    - `-s` (Strict Checking): Heightens the validation ruleset. It treats unassigned structural elements (like defining an Alias group macro but forgetting to map any rules to it) as critical errors.
    - `-q` (Quiet Mode): Suppresses standard logging output. This is typically paired alongside `-c` in administrative automation scripts (`visudo -cq`) to test system health silently via exit codes.
    - `-h` (Help): Prints a concise text manifest of flag triggers and usage syntax.
    - `-V` (Version): Outputs the current core version tracking data for the utility.
- **su** = substitute user, switches the current user account (often used to drop into root shell, but largely replaced by sudo -i). By default, su switches your identity but keeps your original file path and current working directory. Syntax: su [options] [username]
    - - or -l = perform a full login with the user account and also change the current working directory to the new user. Runs only one command and switches back. Without this flag,
    - -l = switches to the user, runs only one command and switches back
    - -c = run a single command as the target user without actually switching your shell permanently (ex. su -c "dnf update" root)
    - -s = allows you to specify a different shell (ex. su -s /bin/bash jake)
- **chattr** = change attribute. Key command for file resilience in Linux. Has to be run with sudo. While standard file permissions (`chmod`) control who can read, write, or execute a file, `chattr` controls what can be done to the file itself, regardless of who owns it—even if the user is `root`. The chattr command was initially designed to keep people, even root from accidentally deleting important files. Operates at the kernel level. Syntax: chattr [operator] [attribute] [filename} (ex. chattr +i /etc/important.conf)
    - Operators:
        - `+`: Adds the attribute.
        - `-`: Removes the attribute.
        - `=`: Sets the attribute exclusively (removing all others).

    - Attributes:
        - i = Immutable. The most important. The file cannot be deleted, renamed, or written to.
        - a = Append-only. The file can only be opened for adding data. You cannot overwrite existing data.
        - A = No Atime. Prevents the system from updating the "Access Time" on the file (saves disk I/O).
        - s = Secure Delete. When deleted, the blocks are zeroed out (wiped) from the disk.
- **lsattr** = Lists the attributes of the file. Standard Linux commands like `ls`, `ll`, or `stat` only show you standard permissions (read/write/execute) and ownership (user/group). They are **blind** to the attributes set by `chattr`.
    - Interpreting Output: (----i---------e--- /etc/shadow)
        - The `-` (dash): Represents an attribute that is not set.
        - The `i`: Indicates the Immutable attribute is active.
        - The `a`: Indicates the Append-only attribute is active.
        - The `e`: Indicates the file is using Extents. (You will see this on almost every file; it is standard for modern ext4/xfs filesystems and can be ignored for security purposes).
    - `-a` (All): Lists all files in directories, including hidden files (those starting with a dot).
    - `-R` (Recursive): Lists attributes for files inside a directory and all subdirectories. This is vital when you are trying to find where a "locked" file is hiding inside a large configuration folder.
- **restorecon** = restore context. A critical command used in Linux systems that utilize SELinux (Security-Enhanced Linux). If you move a file from a home directory to the web root, it often retains its original "home" label. SELinux will then block the web server from reading it because the labels don't match. restorecon compares the current label of the file against the system's default policy (the "golden" configuration) and fixes it. This command requires sudo. Syntax: restorecon [options] [file_or_directory] (ex. sudo restorecon -Rv /var/www/html/my_site)
    - `-v` (Verbose): Show changes as they happen. This is highly recommended so you know exactly what files were "fixed."
    - `-R` (Recursive): Apply the fix to the directory and everything inside it. This is almost always used when you move a folder to a new location.
    - `-F` (Force): Force the reset of the context, even if the policy doesn't explicitly tell it to. (Use with caution).

    - `-n` (No-op/Dry-run): This is a "what if" flag. It shows you what *would* be changed without actually applying the changes. Perfect for testing before making system-wide modifications.
- **semanage** = used to make permanent changes to the system-wide SELinux policy. (DANGEROUS!) Use semanage if there is no setseboolean value available for what you want to do. Syntax: semanage [object] [action] [options] (ex. sudo semanage fcontext -a -t httpd_sys_content_t "/my_web_data(/.*)?"). This command requires sudo
    - Essential "Objects" and Actions: You will most commonly use `semanage` with these objects:
        - `fcontext`: Manages file context mapping (defining what label a directory/file path should *always* have).
            - -l = lists the file content permissions
        - `port`: Manages network port labels.
        - `user` / `login`: Manages SELinux users and their mapping to Linux accounts.
    - `-a` (Add): Add a new record.
    - `-d` (Delete): Remove an existing record.
    - `-m` (Modify): Modify an existing record.
    - `-t` (Type): sets the content type
    - `-l` (List): List all records for that object. (Extremely useful for checking current policies).
- **chcon** = change context. This is the manual way to change the SELinux security fontext or a file or directory. While `semanage` changes the *permanent policy* in the database, `chcon` changes the label on the *file itself* immediately. This command requires sudo. FYI: Because `chcon` does not update the system's policy database, your changes will be **wiped out** if you run `restorecon` or if the system performs a filesystem relabeling. Syntax: chcon [options] [context] [file_or_directory]
    - `-t` (Type): Specify only the SELinux Type (the most common use).
    - `-R` (Recursive): Apply the change to the directory and all contents.
    - `-v` (Verbose): Display the changes made (highly recommended).
    - `--reference=[FILE]`: This is the most "pro" way to use it. Instead of typing out a complex label, you can tell `chcon` to "copy the label from this other file."
- **getenforce** = used to check the current operating mode of SELinux on your system.

- **setenforce** = Change the operating mode of SELinux. Syntax: setenforce [Mode]. This command requires sudo. Any change you make with setenforce is lost upon reboot. If you change it to `Permissive` to troubleshoot a problem and then restart your server, it will revert to the state defined in `/etc/selinux/config`. This file is also the only way to completely disable SELinux.
    - `1` (or `Enforcing`): Tells SELinux to actively enforce security policies. It will block unauthorized actions and log them.
    - `0` (or `Permissive`): Tells SELinux to "look the other way." It will log all policy violations as if it were enforcing, but it will not block any actions. This doesn't disable SELinux, but puts it into "log-only" mode.
- **getsebool** = used to inspect the status of SELinux booleans. Think of Booleans as "on/off switches" for specific SELinux features. Instead of writing complex policies to allow or deny an action, SELinux developers created these switches to let administrators easily toggle common functionalities (like "Allow web servers to access the home directory"). Syntax: getsebool [option] [boolean_name|all]
    - `-a` (All): Lists all available SELinux booleans on the system. Because there are hundreds of them, you will almost always use this with `grep` to find the one you want.
- **setsebool** = tool used to toggle SELinux Booleans. While getsebool shows you the booleans, setsebool is actually how you turn them on or off. You can use `1` or `on` to enable the boolean, and `0` or `off` to disable it. Syntax: sudo setsebool [flags] [boolean_name] [value] (ex. sudo setsebool -P ftp_home_dir off)
    - `-P` (Persistent): This is the most important part of the command. * Without `-P`, your change is only active until the next reboot.
- **audit2allow** = the "automatic problem solver" for SELinux (HIGHLY USEFUL TOOL BUT CAN BE DANGEROUS). It is a utility that reads the audit logs (where the system records every single action that SELinux blocked) and automatically generates the specific policy rules needed to allow those blocked actions. You don't usually run `audit2allow` by itself. It works best as part of a pipeline using grep. (ex. grep "search_term" /var/log/audit/audit.log | audit2allow -m my_module_name)
    - `-a` (or `--all`): Reads all logs in the audit log (useful if you don't know exactly what to grep for).
    - `-w` (or `--why`): Explains *why* the action was blocked in plain English. This is incredibly helpful for learning.
    - `-m [name]` (or `--module`): Specifies the name of the SELinux policy module you are creating.

    - `-l` (or `--last`): Only reads the audit logs since the last time the policy was reloaded (helpful to avoid re-generating old rules).
    - `-M` (combined with the name): This automatically generates a "Type Enforcement" (`.te`) file and a compiled policy package (`.pp`) ready for the system to use. (ex. grep "denied" /var/log/audit/audit.log | audit2allow -M myfix)
- **fail2ban-client** = command line tool for interacting with fail2ban daemon. fail2ban is the daemon that autobans IP addresses when it detects brute force attempts. Any changes made with this command are ephemeral and get wiped out upon next boot. That is why people usually use the /etc/fail2ban/jail.local file when making permanent changes to fail2ban. The command is mainly used for active attach mitigation, quick status checks or temporary mitigations, but doesn't allow you to edit the default ban rules. That is what the jail.local file is for. (Exam Tip: Always modify `jail.local` or files in `jail.d/`. Never edit `jail.conf` directly, as package updates will overwrite your changes.)
    - **status** *Purpose:* Displays global operational status of the fail2ban server, showing the total number of currently active jails.
    - **status <jail>** *Purpose:* Displays verbose statistics for a designated jail (e.g., fail2ban-client status sshd). This is the most crucial diagnostic subcommand on the exam; it reveals the path to the parsed log file, currently banned IP addresses, and historical ban counts.
    - **set <jail> unbanip <IP>** *Purpose:* Manually removes a ban on a specific IP address within a particular jail (e.g., fail2ban-client set sshd unbanip 192.168.1.45). This is an absolute favorite scenario for CompTIA regarding troubleshooting false positives.
    - **set <jail> banip <IP>** *Purpose:* Forces an immediate, manual ban on a specific IP address within a selected jail, overriding normal log-parsing threshold behaviors.
    - **reload** *Purpose:* Reloads the entire fail2ban configuration runtime environment (re-reading /etc/fail2ban/jail.local) without restarting the actual underlying daemon process or dropping historical runtime tracking.
    - **ping** *Purpose:* Executes a basic daemon heartbeat check. If the fail2ban process is up and running properly, it will reply directly with "pong".
- **ssh** = remote into another computer's CLI. Syntax: ssh [flags] [username]@[hostname_or_IP]. Port 22 must be open on the remote server for SSH to work by default.
    - `-p [port]` — Specifies a non-standard port to connect to (default is TCP port 22). Note that the companion command `scp` uses a capital `-P` for ports.

    - `-i [path_to_private_key]` — Forces SSH to authenticate using a specific private key file instead of a text password.
    - `-v` / `-vv` / `-vvv` — Enables verbose diagnostic logging. This is your primary tool for troubleshooting connection lags or public key rejection.
    - `-X` / `-Y` — Enables X11 Forwarding. This allows you to launch a graphical application on the remote server and render its visual window locally on your desktop.
    - `-L [local_port]:[target_host]:[remote_port] user@server` — Configures Local Port Forwarding. Opens a port on your local machine. It secures and maps a port on your local client machine through the SSH tunnel to a destination service. Opens a port on your machine. (You push traffic OUT).
        - SSH Tunneling outbound to another machine
        - target_host = where the traffic goes after it exits the tunnel. It is always resolved from the perspective of the machine standing at the EXIT of the tunnel. In the case of -L, this is the remote server
        - Think of -L like a routing parameter for ssh that says “Listen locally for entrypoint data, and route traffic out to the remote server at the EXIT."
    - `-R [remote_port]:[target_host]:[local_port] user@server` — Configures Remote Port Forwarding. Opens a port on a remote server. The inverse of local forwarding; it opens an entry port on the remote machine and routes traffic backward to your machine. Opens a port on the server. (The outside world sends traffic IN).
        - SSH Tunneling inbound to your machine
        - target_host = where the traffic goes after it exits the tunnel. It is always resolved from the perspective of the machine standing at the EXIT of the tunnel. In the case of -R, this is the user’s machine.
        - Think of -R like a routing parameter for ssh that says “Listen remotely for the entrypoint data, and route traffic back to my machine at the EXIT."
    - `-o [option]` — Allows you to inject inline configuration variables on the fly, bypassing your system's global config settings (e.g., `-o StrictHostKeyChecking=no` or `-o PubkeyAuthentication=no`).
- **ssh-agent** = An SSH Agent (`ssh-agent`) is a local background program (a daemon) that sits in your machine's temporary memory and acts as a secure digital wallet for your SSH private keys.Its main job is to remember your keys and their passphrases so that you don't have to type your password every single time you connect to a server, push code to GitHub, or run automation scripts.

ssh-agent command starts the SSH agent process itself. On most modern operating systems, this starts automatically when you log in.

- **`ssh-add`**: Tells the SSH agent to look at a private key file, decrypt it (asking you for your passphrase one final time), and store it in memory.
    - `ssh-add ~/.ssh/id_ed25519` (Adds a specific key)
    - `ssh-add -l` (Lists all keys currently stored in your agent's memory)
    - `ssh-add -D` (Deletes all keys from the agent's memory, locking the wallet)
    - `ssh-add -d [key]`: Deletes a specific key from the agent's memory.
- **sshd** = ssh daemon. While ssh allows you to connect on the client, sshd is the service daemon that you run on the server. Usually, you never use the bare sshd command, but instead edit the SSH config file in /etc/ssh/sshd_config. You will also start or stop the service using systemctl. When you start the `sshd` service (usually via `systemctl`), it sits in the background (as a daemon) listening on a network port (default: 22). When a client connects: It handles the encryption key exchange, It validates the user credentials (passwords or SSH keys), It creates a secure, encrypted "tunnel" for your shell session to operate through.
    - `-t` (Test mode): This is your best friend. It checks the configuration file (usually `/etc/ssh/sshd_config`) for syntax errors without actually starting the server. (Use this most of the time)
    - `-T`: Dumps the active server configuration for security auditing.
    - `-d` (Debug mode): Runs `sshd` in the foreground and outputs verbose debug information to the terminal. It stops after the first connection, which is perfect for troubleshooting authentication issues.
    - `-f [config_file]`: Specifies an alternative configuration file to use instead of the default `/etc/ssh/sshd_config`.
- **`ssh-keygen`** = generate a private/public keypair to authenticate to another system. Only the server with the private key will be able to sign into the server with the public key
    - `-t [algorithm]` (Type) Specifies the mathematical algorithm used to generate the key pair. *Example:* `ssh-keygen -t ed25519`
    - `-b [bits]` (Bits) Dictates the structural bit-length size of the key. This is critical for legacy RSA keys, which should always be set to at least 3072 or 4096 bits. (Modern `ed25519` keys use a fixed length, so they ignore this flag completely). *Example:* `ssh-keygen -t rsa -b 4096`
    - `-C "[text]"` (Comment) Appends a custom text label to the very end of the public key string. This acts as an identification tag so you know exactly

which device or user the key belongs to. *Example:* `ssh-keygen -t ed25519 -C "backup-server-key"`
    - `-f [path/filename]` (Filename) Forces the utility to save the resulting key pair to a specific custom path or filename, rather than defaulting to the standard user path (`~/.ssh/id_rsa` or `~/.ssh/id_ed25519`). *Example:* `ssh-keygen -t ed25519 -f ~/.ssh/production_key`
    - `-N "[passphrase]"` (New Passphrase) Allows you to inject a secure password passphrase to encrypt the private key inline via the command line. For automated scripts or server-to-server backups where human intervention is impossible, you pass a blank string to make the key passphrase-less. *Example:* `ssh-keygen -t ed25519 -N ""`
    - `-R [hostname_or_IP]` (Remove Host Key) Removes all historical cryptographic key strings belonging to a specific hostname or IP address from your local ~/.ssh/known_hosts file. *The Exam Scenario:* A remote server's motherboard was replaced, or its OS was reinstalled, changing its identity footprint. When you try to connect via SSH, your console triggers a giant warning: "WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!" and blocks the connection. Running ssh-keygen -R 10.0.0.5 clears the old signature, allowing you to establish a fresh trust profile.
    - -l (List/View Fingerprint) Scans an existing public key file and prints its unique SHA-256 fingerprint hash string. This allows you to verify that a key file matches a signature found inside a server's security audit logs without revealing the full key text. *`Example:`* `ssh-keygen -l -f ~/.ssh/id_ed25519.pub`
- **ssh-copy-id** = sends the public key over to another system so the system with the private key can authenticate automatically to it
    - `-i [path_to_public_key]` (Identity File) Explicitly tells the script which public key file to read and copy. If you omit the `.pub` extension (e.g., pointing to `id_ed25519`), the utility automatically appends `.pub` to ensure your private key file never leaves your machine. *Example:* `ssh-copy-id -i ~/.ssh/id_ed25519.pub user@10.0.0.5`
    - `-p [port]` (Port) Specifies the remote destination port if the target server is listening for SSH traffic on a non-standard port instead of standard TCP port 22. *Example:* `ssh-copy-id -p 2222 -i ~/.ssh/id_ed25519.pub user@10.0.0.5`
    - `-n` (Dry Run) Executes a safety drill. It simulates the network handshake and outputs the exact key data that *would* have been copied to the remote server, without altering any physical configuration files on either system.

    - `-f` (Force Mode) By default, `ssh-copy-id` checks to see if your key already exists on the destination server to prevent duplicate strings. The `-f` flag bypasses this safety verification pass and forces the string to append regardless.
- **sftp** = ftp over ssh. Allows you to send files securely. This is NOT just FTP with TLS enabled, it is built completely on SSH for security. It operates on port 22 just like SSH and also encrypts both credentials and data.
    - You want to make sure SFTP settings inside of /etc/ssh/sshd_config are:
        - Match Group sftp-users
            - ChrootDirectory /home/%u
            - ForceCommand internal-sftp
            - X11Forwarding no
            - AllowTcpForwarding no
    - -P <port> (Specify Port) = specifies the port to connect on the remote host (default it 22)
        - ex. sftp -P 2222 user@remote-server
    - `-b <batchfile>` (Batch Mode) = Tells SFTP to run automatically without human interaction by reading a list of file-transfer commands from a local text file. This is heavily tested in the context of automation and cron jobs.
    - `i <identity_file>` (Select Key) = Selects the specific file from which the identity (private key) for public key authentication is read. This is used if you aren't using the default `~/.ssh/id_rsa` or `id_ed25519` keys. Example: `sftp -i ~/.ssh/custom_key user@remote-server`
    - `-C` (Enable Compression) = Enables data compression. This uses the SSH compression algorithm to speed up transfers over slow networks. Example: `sftp -C user@remote-server`
    - **Interactive Subcommands:** Once you successfully log in using `sftp user@server`, your terminal prompt changes to `sftp>`. You are now inside an interactive file-transfer environment.The Linux+ exam expects you to know how to navigate *both* your local laptop and the remote server simultaneously. To do this, SFTP uses a clever `l` prefix system:
        - Commands without an `l` happen on the remote server.
        - Commands with an `l` happen on your local machine.
        - ex. lpwd prints the working directory for the server you remoted into
    - **Core Transfer Commands:** Used to transfer files once inside of sftp

        - `put <filename>`: Uploads a file from your local machine to the remote server.
        - `get <filename>`: Downloads a file from the remote server to your local machine.
        - `exit` or `quit`: Closes the secure SFTP connection.
- **chroot** = change root, changes the root directory (/) for a specific process. 99% of the time, you will never use this command, but will instead write the ChrootDirectory parameter into the /etc/ssh/sshd_config file. Generally, you will set the parameter to ChrootDirectory /home/%u. chroot creates a "jail cell" to contain a user from snooping around the actual root filesystem. Always used when deploying an SFTP server. When you apply a `chroot` jail to an SFTP user, you trap them inside a specific directory (like `/home/sftpuser`). To that user's session, `/home/sftpuser` **is** the root directory (`/`). They can type `cd ..` all day long, but they cannot escape. This implements the principle of least privilege. Syntax: chroot [OPTION] NEWROOT [COMMAND [ARG]...]
    - --userspec=USER:GROUP = Specifies the exact user and group (either by name or numeric ID) to run the command as *inside* the jail.
    - --groups=G_LIST = Specifies a comma-separated list of supplementary groups for the user inside the jail.
- **MFA / TOTP** = (Multifactor Authentication / Time-based One-Time Password) requires a second proof of identity beyond a password — typically a 6-digit code that changes every 30 seconds.
    - **`google-authenticator`** = a PAM module and CLI setup tool that generates a TOTP secret (and QR code) for a user, then enforces a matching one-time code on login via PAM integration.
    - **`oathtool`** = a command-line utility that generates or validates OATH one-time passwords (HOTP/TOTP), often used to script or test MFA codes without a phone app. (ex. `oathtool --totp -b <secret>`)
- **Restricted Shells** = shells assigned to a user account that intentionally limit what that account can do when it logs in.
    - **`/sbin/nologin`** = set as a user's login shell to block interactive login entirely while still allowing the account to exist (used heavily for system/service accounts). Attempting to log in prints a message and disconnects.
    - **`/bin/rbash`** = (restricted bash) a limited version of Bash that prevents the user from changing directories with `cd`, setting `PATH`/`SHELL`, using absolute command paths, or redirecting output — used to sandbox a user to a narrow set of tasks.

- **pam_tally2** = Displays a comprehensive dump of all monitored system accounts that have accumulated at least one login failure. (Deprecated and replaced by faillock)
    - **`-u <username>`** (or `--user`) *Purpose:* Targets a specific user account to see their exact current bad-attempt counter. *Example:* `sudo pam_tally2 -u jsmith`
    - **`-r`** (or `--reset`) *Purpose:* **Resets the failure counter back to 0.** This instantly unlocks a user account when an administrator needs to clear a false-positive lockout lockout state manually. It is almost always used in combination with the `-u` flag. *Example:* `sudo pam_tally2 -u jsmith -r` (Clears `jsmith`'s failures, unlocking the account).
- **faillock** = Displays a comprehensive system-wide dump of all recorded authentication failures for all local accounts across the operating system. Replaced pam_tally2
    - **`--user <username>`** *Purpose:* Targets a specific user account to see their current authentication failure records, timestamps, and originating terminal or IP address. This is your primary diagnostic flag when a user submits a ticket saying they cannot log in. *Example:* `sudo faillock --user jsmith`
    - **`--reset`** *Purpose:* **Resets the failure counter back to 0.** This instantly unlocks a user account when an administrator needs to clear a false-positive lockout state manually. It is almost always combined with the `--user` flag. *Example:* `sudo faillock --user jsmith --reset` (Clears `jsmith`'s failures, unlocking the account).

Cryptography

- **Argon2** = the modern, memory-hard key derivation function (KDF) used by LUKS2 to turn a passphrase into an encryption key. Being memory-hard makes brute-force attacks with GPUs/ASICs far more expensive than with older KDFs like PBKDF2.
- **HMAC** = (Hashed Message Authentication Code) combines a cryptographic hash function (like SHA-256) with a secret key to verify both the integrity and the authenticity of a message — proving the data wasn't altered and that it came from someone who holds the key.
- **sha256sum / sha512sum** = generate a hash for data integrity or provide proof of data integrity. The only difference between the two is that sha512sum is meant for larger, more secure hashes.

    - `-c` (check) flag: automatically compares the actual hash to the vendor-provided hash (ex. sha256sum -c SHA256SUMS.txt)
- **gpg** = asymmetric (public-key) cryptography for individual files. **Your Private Key:** This is the physical key that opens the padlock. You must guard this with your life and **never** share it with anyone. **Your Public Key:** Think of this as an open padlock. You can hand it out to anyone in the world, post it on the internet, or send it to your coworkers. **To Encrypt:** If Alice wants to send a secret file to Bob, she uses **Bob's Public Key** to lock the file. Once locked, only one thing in the universe can unlock it: **Bob's Private Key**. Even Alice can't unlock it anymore once it's encrypted! **To Sign (Integrity):** If Bob wants to prove a file definitely came from him and wasn't altered by a hacker, he encrypts it with his **Private Key**. Anyone can use Bob's world-readable **Public Key** to decrypt it. If it decrypts successfully, it proves Bob's signature is authentic.
    - --gen-key = generates a new keypair
    - --list-keys = lists the keys on your systems
    - `-r`: Specify the recipient (whose public key to use).
    - `-c` = (conventional) flag, used to lock a file with an old standard password
    - --clearsign = tells the system to read your private key, generate a unique signature block at the bottom of the script, and prove to your servers that the script hasn't been altered by an attacker.
    - `--gen-key` or `--generate-key`: Creates a new public/private key pair.
    - `--list-keys`: Displays keys in your public keyring.
    - `--import <file>`: Imports a public or private key.
    - `--export -a "<Name>"`: Exports a public key in an ASCII-armored format.
    - `-e` / `--encrypt`: Encrypts data (requires specifying a recipient with `-r`).
    - `-d` / `--decrypt`: Decrypts data.`-s` / `--sign`: Signs a file to guarantee authenticity.
    - --export-secret-keys --armor johndoe@email.com > my-private-key.asc = export your private key so you can back it up (DANGEROUS. DO NOT LET THIS FILE GET INTO THE WRONG HANDS)
- **cryptsetup** = manage full-disk encryption in Linux using the **LUKS2 (Linux Unified Key Setup 2)** standard. Syntax: cryptsetup [OPTIONS] <ACTION> <DEVICE> [MAPPED_NAME]
    - luksFormat = Destroys data on a partition and initializes it as an encrypted LUKS volume. (DANGEROUS, deletes all data on the partition)
    - luksOpen = Validates the password and maps the scrambled partition to a usable virtual device in `/dev/mapper/`

    - luksClose = Unmaps the virtual device, locks the drive, and purges the decryption key from RAM.
    - luksDump = Reads and displays the header metadata of the drive without unlocking it.
    - luksErase <device> = This is the "nuclear option" for total cryptographic destruction. It completely zeroes out **all** keyslots across the entire LUKS header instantly. (DANGEROUS)
    - luksKillSlot <device> <slot_number> = This targets and securely scrubs a *specific* keyslot from the drive. It is often used for targeted cryptographic destruction (e.g., if a specific administrator leaves the company and you want to permanently destroy their access token/key material). (DANGEROUS)
    - --type <luks1|luks2> = explicitly defines the metadata format
    - `--key-file` (or `-d`) = Instead of forcing a human to type a password at a prompt, this tells `cryptsetup` to read a binary file or a text file to unlock the drive.
    - `--readonly` (or `-r`) = Forces the device to map as read-only during a `luksOpen` command.
- **openssl / libressl** = software toolkit for encrypting data in transit, specifically SSL/TLS certificates (the tech that turns `http://` into secure `https://`).It can use any hashing or encryption algorithm you want to use. LibreSSL is the more secure version of openssl, but it uses the same syntax and subcommands.
    - The Three Key files:
        - `.key` (The Private Key): This is the cryptographic foundation. Just like your GPG private key, this file must stay hidden on your server with strict permissions (`600`). If a hacker steals your private key, they can impersonate your website and steal user data.
        - `.csr` (Certificate Signing Request): This is an application form. It contains your public key and your identity details (your company name, domain name, country, etc.). You bundle this up and send it to a trusted third party (like DigiCert or Let's Encrypt) to prove who you are.
        - `.crt` or `.pem` (The Certificate): This is the final digital ID card sent back to you by the certificate authority. You install this alongside your private key into your web server (like Apache or Nginx) so users see the secure green padlock in their browsers.
    - `genrsa`: The sub-command telling OpenSSL to use the RSA algorithm. (ex. openssl genrsa -out server.key 2048)

        - `2048`: The bit-length (key strength). 2048 is the minimum industry standard for RSA.
    - genpkey = generate private key. Usually used to specify an algorithm other than RSA, but can be used for any algorithm. (ex. openssl genpkey -algorithm Ed25519 -out server.key)
        - `-algorithm`: Swaps the toolkit engine from RSA to whatever algorithm you specify.
    - `req` = certificate signing request
        - `-new`: Tells OpenSSL you are requesting a *new* certificate configuration. (ex. openssl req -new -key server.key -out server.csr)
            - `-key`: Points to the private key you made in the previous step to sign the request.
        - `-x509`: This is the ultimate exam keyword. The moment you see `-x509`, you are creating a final, self-signed certificate rather than a temporary request form. (ex. openssl req -x509 -newkey rsa:2048 -keyout server.key -out server.crt -days 365)
            - `-days 365`: Sets the expiration date to exactly one year.
    - `x509`: The utility used to view or modify signed certificates. (ex. openssl x509 -in server.crt -text -noout)
        - `-text -noout`: Tells OpenSSL to display the human-readable text details (like expiration dates and domain names) and suppress the raw scrambled binary code.
- **wg / wg-quick** = wireguard commands used to interact with wireguard VPN connections. Wireguard is usually used with server-to-server connections, most often in the cloud.
    - wg = dashboard of wireguard showing interface keys and connected peers.
    - wg-quick up wg0 = bring a tunnel up.
    - wg-quick down wg0 = bring a tunnel down.
- **openssl** = implement TLS/SSL protocols. and general-purpose cryptographic library.
    - s_client – Activates the secure network client framework. This subcommand implements a generic SSL/TLS client that establishes a secure tunnel to a remote server, making it invaluable for testing network daemons like HTTPS, IMAPS, or SMTPS.
        - -connect [host]:[port] – Specifies the remote server address and encrypted port to test (e.g., -connect www.example.com:443).
        - -showcerts – Dumps all X.509 certificates in the server's certificate chain to stdout, allowing you to manually verify the authority chain.

        - -brief – Strips away excessive cryptographic logs, providing a concise summary of the TLS connection parameters and error messages.
    - genrsa – Generates an unencrypted RSA private key payload.
        - -out [filename] – Directs the command to write the output data to a designated file instead of stdout (e.g., openssl genrsa -out server.key 2048).
    - req – Initializes the PKI Certificate Signing Request (CSR) creation framework.
        - -new – Prompts the administrator for identity fields (Common Name, Organization, etc.) to generate a brand-new CSR.
        - -key [key_file] – Paths the existing private key file required to sign the creation request or certificate.
        - -x509 – Bypasses a public CA submission and instructs OpenSSL to output a self-signed certificate instead of a request.
        - -days [number] – Sets the absolute expiration timeline for the certificate (e.g., -days 365).
    - x509 – Invokes the certificate viewing and signing engine. This subcommand acts as a display and alteration utility for existing X.509 format security certificates.
        - -in [cert_file] – Specifies the target certificate file to read into the utility engine.
        - -text – Translates the raw Base64 encoded certificate structure into a completely legible plain-text layout.
        - -noout – Suppresses the default behavior of printing the encoded certificate block to the terminal screen, usually paired with -text to isolate human-readable information.

Compliance & Audit

- **Security Banners** = text files displayed to a user at various points in the login process, often used to display legal warnings before authentication.
    - **`/etc/issue`** = displayed before the local login prompt (console/TTY), before the user has authenticated.
    - **`/etc/issue.net`** = the equivalent banner shown to remote users connecting over the network (e.g., via Telnet or a configured SSH banner), before login.
    - **`/etc/motd`** = (message of the day) displayed immediately after a successful login — used for announcements rather than legal warnings.

- **OpenSCAP** = (Open Security Content Automation Protocol) a framework and toolset (`oscap`) for automatically scanning a system against standardized security policies/benchmarks (like a CIS profile) and generating a compliance report showing exactly which checks passed or failed.
- **`ClamAV`** = ClamAV itself is not a command but a suite of commands and the open-source antivirus standard on Linux distros. Below are the specific commands you type in the command line.
    - `freshclam`: Updates the virus signature database (essential to run before scanning).
    - `clamscan`: The command used to scan files and directories (e.g., `clamscan -r /home` to scan recursively).
    - `clamd`: The background daemon that runs for real-time or scheduled scanning.
- **aide** = advanced intrusion detection system.
    - `aide --init`: This creates the initial file integrity database baseline (`aide.db.new.gz`). It hashes all specified system files based on permissions, size, modification time, and cryptographic checksums (SHA-256/512).
    - `aide --check`: The standard command run by administrators (often automated via cron). It compares the live system files against the baseline database to output any additions, deletions, or modifications.
    - `aide --update`: Run after an administrator performs legitimate system updates (like updating packages using `apt` or `dnf`). This updates the database so those changes aren't flagged as false positives.
- **rkhunter** = looks specifically for rootkits, backdoors, and local system exploits.
    - `rkhunter --update`: Downloads the latest signatures, properties, and known malicious indicators. (Must be run before scanning).
    - `rkhunter --propupd`: (Property Update). This is critical. Like AIDE, it takes a snapshot of your clean local system file properties. If you update your kernel or system software, you *must* run this command, or `rkhunter` will throw false-positive alerts thinking your updated binaries are rootkits.
    - `rkhunter --check` (or `-c`): Executes the actual automated diagnostic checks. It will pause between sections (like system command checks, network checks, local host checks) requiring you to hit Enter unless you pass the `--sk` (skip keypress) flag.
- **shred** = unlike the **rm** command, **shred** provides secure data destruction by repeatedly overwriting the file's data block sectors with a mix of random patterns,

zeroes, or pseudo-random bits. This disrupts the underlying magnetic or physical signature of the original data, making forensic recovery nearly impossible.
    - `-n <number>` (or `--iterations=<number>`): Specifies the number of times `shred` will overwrite the data. By default, `shred` overwrites the file 3 times with randomized data blocks. Passing `-n 5` changes that to 5 iterations. Higher iterations provide higher compliance security but take significantly longer.
    - `-u` (or `--remove`): This is a heavily tested flag. By default, `shred` overwrites the data inside the file, but it leaves the file itself existing on the system (now filled with garbage data). Adding `-u` tells the utility to overwrite the data, truncate it, and then deallocate/delete the file from the filesystem structure.
    - `-z` (or `--zero`): Adds a final, hidden overwrite iteration consisting completely of null zeroes (`0x00`). This is used to hide the fact that a secure shredding operation occurred, making the file look like a clean, blank slate rather than suspicious random garbage data.
    - `-v` (or `--verbose`): Displays the live progress of the overwrite passes on the screen so you can track how many passes are complete and the file offsets being written.
    - `-f` (or `--force`): Automatically changes the file permissions if necessary to allow writing, ensuring the shredding operation succeeds even if the file is currently marked read-only.
- **badblocks** = scans the storage sectors to look for damaged or unreadable areas. By default, running `badblocks /dev/sdb` performs a non-destructive, read-only scan of the storage sectors to look for damaged or unreadable areas.
    - `-w` (Destructive Write-Test): The `-w` flag triggers an intensive, destructive write-test. Instead of just reading, it actively overwrites every single block on the specified storage drive or partition. What **shred** does to files, **badblock -w** does to drives and partitions. (DANGEROUS)

Troubleshooting & Monitoring

- **SNMP** = (Simple Network Management Protocol) the standard protocol for monitoring and managing network devices (switches, routers, servers) from a central monitoring station.
    - **`snmpwalk`** = queries a device and walks an entire branch of its MIB tree, returning every value under that branch. (ex. `snmpwalk -v2c -c public 10.0.0.1 system`)

    - **`snmpget`** = retrieves the value of one specific OID (object identifier) from a device, rather than an entire branch.
    - **Traps** = unsolicited, event-driven alerts a device pushes to a monitoring server on its own (e.g., 'interface went down') rather than waiting to be polled.
    - **MIBs** = (Management Information Bases) the schema/dictionary files that map human-readable names (like `ifDescr`) to the numeric OIDs a device actually reports data under.
- **GRUB** = (Grand Unified Bootloader) the bootloader most Linux distros use to load the kernel. A misconfigured GRUB (bad `/boot/grub2/grub.cfg`, wrong root partition UUID, missing kernel entry) is a classic 'server won't boot' scenario.
    - **`grub2-mkconfig`** = regenerates the `grub.cfg` file from the templates in `/etc/grub.d/` and the settings in `/etc/default/grub`. (ex. `grub2-mkconfig -o /boot/grub2/grub.cfg`)
    - **`update-grub`** = a Debian/Ubuntu convenience wrapper that runs the equivalent of `grub-mkconfig -o /boot/grub/grub.cfg` for you.

Automation & Orchestration Concepts

- **Kickstart** = a Red Hat/Fedora-family tool for fully unattended, hands-off OS installation. A single Kickstart file (`.ks`) declares partitioning, package selection, network config, users, and post-install scripts up front, so the same install can be repeated identically across many servers with zero interactive prompts. (Compare to `cloud-init`, which configures a VM/cloud instance *after* it boots from a pre-built image, rather than driving the installer itself.)

Python

- **Setting Up a Virtual Environment** = a self-contained Python installation directory that isolates a project's dependencies from the system-wide Python and from other projects, preventing version conflicts.
    - **`python3 -m venv <env_name>`** = creates a new virtual environment directory containing its own interpreter and `pip`.
    - **`source <env_name>/bin/activate`** = activates the environment in the current shell, so `python`/`pip` point at the isolated copies instead of the system ones.
    - **`deactivate`** = exits the virtual environment and returns to the system Python.
- **Built-in Modules** = functionality included with every standard Python install, requiring no extra installation (e.g., `os` and `sys` for OS interaction, `json` for

parsing, `subprocess` for running shell commands, `argparse` for CLI arguments).

- **Installing Dependencies** = external, third-party packages are installed with `pip`, Python's package manager.
  - **`pip install <package>`** = downloads and installs a package from PyPI into the active environment.
  - **`pip install -r requirements.txt`** = installs every package pinned in a requirements file, ensuring a reproducible environment across machines.
  - **`pip freeze`** = prints every installed package and its exact version, typically redirected into a `requirements.txt` file.
- **Python Fundamentals** = the core language mechanics an administrator needs to read and lightly modify automation scripts.
  - **Indentation** = unlike Bash's `{ }` or `done`, Python uses whitespace indentation itself to define code blocks (if/for/function bodies). Inconsistent indentation (mixing tabs and spaces) is a syntax error, not a style choice.
  - **Current Versions** = Python 3.x is the actively maintained line; Python 2 reached end-of-life in 2020 and should not appear in new scripts or system tooling.
- **Data Types and Structures** = Python's built-in types for holding values.
  - **`Boolean`** = `True` or `False`, used for conditional logic.
  - **`Dictionary`** = an unordered collection of `key: value` pairs, similar to a JSON object (ex. `{"host": "web01", "port": 443}`).
  - **`Floating Point`** = a decimal number (e.g., `3.14`), as opposed to a whole integer.
  - **`Integer`** = a whole number with no decimal component (e.g., `42`).
  - **`List`** = an ordered, mutable collection of items, written in square brackets (ex. `["eth0", "eth1"]`).
  - **`String`** = a sequence of text characters, enclosed in quotes (ex. `"hello"`).
  - **Extensible Using Modules and Packages** = Python's functionality can be extended arbitrarily far by importing additional modules (single files) or packages (directories of modules), whether built-in, pip-installed, or your own.
- **PEP 8** = (Python Enhancement Proposal 8) the official style guide for Python code — covering naming conventions, indentation width (4 spaces), line length, and whitespace — followed so code stays consistent and readable across teams.

Git

- **.gitignore** = a file listing patterns for files and directories that Git should never track or stage (e.g., build artifacts, credentials, `__pycache__/`).

- **git init** = initializes a brand-new, empty Git repository in the current directory by creating a hidden `.git/` metadata folder.
- **git clone <url>** = downloads a complete copy of an existing remote repository (full history included) into a new local directory.
- **git config** = reads or sets Git configuration values, such as identity or default behavior. (ex. `git config --global user.name "Alice"`)
- **git add <file>** = stages changes (new, modified, or deleted files) into the index, marking them to be included in the next commit. (ex. `git add .` stages everything in the current directory)
- **git commit** = permanently records the currently staged changes as a new snapshot in the repository's history, along with a descriptive message. (ex. `git commit -m "Fix nginx config"`)
- **git status** = shows which files are staged, unstaged, or untracked relative to the last commit.
- **git diff** = shows the exact line-by-line changes between the working directory, the staging area, and/or a previous commit.
- **git log** = displays the commit history for the current branch, newest first, including author, date, and message.
- **git branch** = lists, creates, or deletes branches — independent lines of development. (ex. `git branch feature-x` creates a new branch without switching to it)
- **git checkout** = switches the working directory to a different branch or commit. (ex. `git checkout feature-x`) Increasingly replaced by the more explicit `git switch`/`git restore`, but still ubiquitous on the exam and in the field.
- **git merge** = integrates the changes from one branch into the current branch, creating a merge commit if the histories have diverged.
  - **`squash`** = `git merge --squash` combines every commit from the source branch into a single new commit on the target branch, keeping history clean at the cost of losing the individual commit granularity.
- **git rebase** = replays the current branch's commits on top of another branch's tip, producing a linear history instead of a merge commit. Rewrites commit hashes, so it should be avoided on commits already pushed and shared with others.
- **git fetch** = downloads new commits and branches from a remote repository into your local copy, without merging them into your current branch.
- **git pull** = a combination of `git fetch` followed immediately by `git merge` (or `rebase`, with `--rebase`), bringing your current branch up to date with its remote counterpart.
- **git push** = uploads your local commits to a remote repository, updating the remote branch to match yours.

- **git reset** = moves the current branch pointer to a different commit, optionally unstaging (`--mixed`, the default) or discarding (`--hard`) changes made since.
- **git stash** = temporarily shelves uncommitted changes so you can switch context (e.g., to fix an urgent bug on another branch) without committing half-finished work. `git stash pop` reapplies them later.
- **git tag** = creates a fixed, named pointer to a specific commit, typically used to mark release versions (ex. `git tag v1.2.0`).

Ansible

- **ansible-playbook** = this command is used to execute structured, multi-task configuration automation scripts written in YAML (Playbooks).
    - **`-i <path>`** / **`--inventory <path>`**: Specifies the path to a custom inventory file listing your target hosts. If omitted, Ansible defaults to `/etc/ansible/hosts`.
    - **`-C`** / **`--check`**: Runs the playbook in **Check Mode** (dry run). It simulates the execution, showing what *would* change on the targets without actually making any modifications.
    - **`--syntax-check`**: Validates the playbook structure and indentation for YAML syntax errors without running any tasks against remote hosts.
    - **`-l <group/host>`** / **`--limit <group/host>`**: Restricts the playbook execution to a specific subset of hosts or inventory groups (e.g., limiting a web server update to only `[webservers]`).
    - **`-e <vars>`** / **`--extra-vars <vars>`**: Injects or overrides configuration variables at runtime via the command line (e.g., `-e "version=2.4.1"`). This has the highest precedence in Ansible.
    - **`-K`** / **`--ask-become-pass`**: Prompts the administrator for the privilege escalation (`sudo`) password when `become: true` is set.
    - **`-v`** / **`-vv`** / **`-vvv`** / **`-vvvv`**: Controls log verbosity. Each additional `v` provides deeper tracing, with `-vvvv` providing full SSH connection and execution debugging output.
- **ansible** = This utility executes **ad-hoc commands** used to agentless server automation. It is used to run a single, quick command or module across one or many managed systems immediately without needing to write a formal playbook.
    - **-m <module>**: Defines which Ansible module to execute (e.g., -m ping, -m dnf, -m user). If this flag is omitted, it defaults to the command module.
    - **-a "<arguments>"**: Passes arguments or options directly to the chosen module (e.g., -m command -a "systemctl restart sshd").

    - **-i <path> / --inventory <path>**: Specifies the host directory or file mapping target nodes.
    - **-b / --become**: Forces the ad-hoc task to run with elevated root privileges (sudo or privilege escalation).
    - **-K / --ask-become-pass**: Prompts the user for the privilege escalation password.

Puppet

- **puppet** = command used to interact with the puppet application for agent-based server automation.
    - **puppet agent** = This command interacts directly with the Puppet client daemon running locally on a managed target node to fetch and enforce configuration changes.
        - **-t or --test**: Runs the agent sequentially in the foreground exactly once. It initializes a single check-in, runs facter, outputs verbose logging directly to the terminal, applies the compiled catalog from the Primary server, and exits. This is the most crucial command for real-time testing and troubleshooting.
        - **--enable**: Re-enables the background Puppet agent daemon service if it was previously halted or blocked, allowing it to resume its standard automatic check-in schedule (typically every 30 minutes).
        - **--disable "[message]"**: Disables the local background agent daemon from checking in with the primary server. Providing an optional descriptive message locks the execution queue and tells other administrators why automatic state convergence has been paused.
    - **puppet module** = Executed primarily on the Puppet Primary server, this subcommand interfaces with local directories or remote repositories like the Puppet Forge to manage the macro packaging of configuration components.
        - **install <author>-<module_name>**: Downloads an explicit infrastructure package from the Puppet Forge and resolves any nested module dependencies automatically within the environment paths.
        - **list**: Scans the local module path directories and prints a complete tree of all installed modules along with their specific version metrics.
        - **uninstall <author>-<module_name>**: Removes the designated module directory completely from the local filesystem layout

    - **`puppet parser`** = This subcommand allows an administrator to audit written code on the terminal before deploying changes or restarting daemons.
        - **`validate <filename.pp>`**: Evaluates a specific Puppet Program (`.pp`) manifest file against the official Puppet DSL grammar rules. If it finds syntax typos, missing braces, or declaration errors, it prints them; if the manifest file is structurally flawless, it returns an exit code of `0` with zero terminal output.
    - **puppet resource** = A powerful administrative probe tool used to query or manipulate localized OS components through the lens of declarative Puppet primitives.
        - **puppet resource <type> [<name>]**: Inspects a specific system state and instantly translates it into standard, syntactically correct Puppet code format.
            - *Example:* Running puppet resource user root will inspect the local /etc/passwd entry for the root user and print a clean user { 'root': ensure => present, uid => '0', ... } block directly to the screen.
- **facter**: is a standalone command-line inventory utility that discovers and reports localized system-level data points (referred to as **facts**) about the host hardware, operating system, and network infrastructure.
    - facter <fact_name>: Filters the output down to single metrics (e.g., facter os.name or facter networking.ip)
    - -j or --json: Formats the entire system profile output or targeted facts into structural JSON strings. This is ideal when piping facter output into other automation scripts, API components, or parsing utilities like jq.
    - -y or --yaml: Formats the output array into structured YAML format, matching data serialization workflows common to other automation environments like Ansible.
    - -p or --custom-facts: Instructs the tool to look for, evaluate, and include localized custom facts or external variables defined in the system plugin directories alongside standard core facts.

OpenTofu

- **opentofu** = While Ansible and Puppet are used for configuration management on servers that you already connect to, OpenTofu is used to actually automatically provision servers or virtual networks with standard baseline configurations. The tofu CLI utility evaluates declarative configuration files (written in HashiCorp Configuration Language, typically ending in .tf) inside a working directory. It

targets specific subcommands to interface with the code, the state file, and target infrastructure APIs. Subcommands You Need to Know

- Example Resource Block of a .tf file:
    - Terraform
        - resource "libvirt_domain" "web_server" {
            - name   = "production-web"
            - memory = "2048"
            - vcpu   = 2
        - }
- init: Prepares a working directory containing OpenTofu configuration files. This is the first command that must be run after writing a new configuration. It initializes the backend configuration and downloads the required providers (plugins) specified in the code.
- plan: Performs a dry run of the execution pipeline. It reads the existing state file, compares it against the active configuration files, queries the provider APIs to verify what is running in the real world, and outputs an execution plan showing exactly which resources will be created, modified, or destroyed without making actual changes.
- apply: Executes the actions proposed in the plan. It establishes standard API calls to the providers to provision, update, or reconfigure the declared resources. Upon successful execution, it writes the metadata of the newly deployed infrastructure into the state file.
- destroy: Serves as a convenient mechanism to tear down all infrastructure managed by the specific configuration. It reads the local or remote state file to discover all active tracking IDs and sends destruction API requests to the target platforms, removing all associated resources and clearing the state file.
- validate: Validates the configuration files in a directory to ensure they are syntactically valid and internally consistent. It only checks the code mechanics and does not access any remote APIs or state files.
- fmt: Automatically scans configuration files in the directory and applies standard formatting and canonical layout rules to improve code readability.

Kubernetes

- **Kubernetes** = a container orchestration platform that automates deploying, scaling, networking, and healing containerized applications across a cluster of machines (nodes), rather than managing containers one host at a time like Docker/Podman do.
- **kubectl** = the primary command-line tool for interacting with a Kubernetes cluster's API server.

    - **`kubectl get <resource>`** = lists resources of a given type (e.g., `kubectl get pods`, `kubectl get nodes`).
    - **`kubectl describe <resource> <name>`** = prints detailed configuration and recent events for a single resource — the first stop when troubleshooting.
    - **`kubectl apply -f <file.yaml>`** = declaratively creates or updates resources to match the state defined in a YAML manifest.
    - **`kubectl logs <pod>`** = prints the stdout/stderr of a container running inside a pod.
    - **`kubectl exec -it <pod> -- <cmd>`** = runs a command (often `/bin/bash`) inside a running pod's container interactively.
    - **`kubectl delete <resource> <name>`** = removes a resource from the cluster.
- **Pods** = the smallest deployable unit in Kubernetes; a pod wraps one or more tightly-coupled containers that share the same network namespace and storage.
- **Deployments** = a higher-level controller that manages a set of identical pod replicas, handling rolling updates, rollbacks, and self-healing (automatically replacing pods that crash or are evicted).
- **Services** = a stable network endpoint (a fixed IP/DNS name) that load-balances traffic to a changing set of pods, so other workloads don't need to track individual pod IPs as pods are replaced.
- **ConfigMaps** = a Kubernetes object for injecting non-sensitive configuration data (environment variables, config files) into pods, decoupled from the container image itself.
- **Secrets** = functionally like a ConfigMap, but intended for sensitive data (passwords, tokens, keys). Values are base64-encoded (not encrypted by default) and access can be restricted separately from ConfigMaps.
- **Volumes** = storage attached to a pod that can outlive individual container restarts within that pod (or, with the right backing storage, outlive the pod itself), used the same way a Docker volume persists data beyond a container's lifecycle.
- **Variables** = environment variables can be injected into a pod's containers directly in the manifest, or pulled dynamically from a ConfigMap or Secret so the same image can be reused across environments with different configuration.

Docker Swarm

- **Docker Swarm** = Docker's own built-in, lighter-weight alternative to Kubernetes for orchestrating containers across multiple hosts. It trades some of Kubernetes' flexibility for a much simpler setup, using the same `docker` CLI you already know.

- **docker swarm init** = turns the current Docker host into a Swarm manager node, generating a join token for other hosts to become workers or additional managers.
- **Nodes** = a physical or virtual machine participating in the swarm; nodes are either managers (schedule work, maintain cluster state) or workers (just run containers).
- **Service** = the Swarm equivalent of a Deployment — a declarative definition of which image to run, how many replicas, and what ports/networks to use, which Swarm keeps running across the cluster.
  - **`docker service create`** = defines and launches a new service across the swarm. (ex. `docker service create --replicas 3 --name web nginx`)
  - **`docker service update`** = changes a running service's configuration (image version, replica count, env vars) with a rolling update.
- **Tasks** = a single running container instance that belongs to a service; a service with 3 replicas is made up of 3 tasks distributed across the available nodes.
- **Networks** = Swarm automatically provisions an overlay network spanning every node in the cluster, so containers in the same service can reach each other by name regardless of which physical host they land on.
- **Scale** = `docker service scale <service>=<n>` changes the number of running task replicas for a service up or down on demand.