Linux Academy

Linux Professional Institute Certification Level 1 Exam 2

Study Guide

Kenny Armstrong kenny@linuxacademy.com

December 14, 2018

Contents

Prerequisites

Linux - CentOS 6, CentOS7 (or other SysVinit or Systemd Distribution)

Topic 105: Shells, Scripting and Data Management

105.1 Customize and Use the Shell Environment (Shells, Variables, Setting, and Sourcing)

1. Special shells:
    - /bin/false - Returns a non-zero code to the request which will block any user request to login
    - /sbin/nologin - Also blocks login requests, but returns a text message result
2. Variables (environment):
    - "Shortcuts" to a variety of values and types
        - For example: `BUCKET=123456`
            - This will set an environment variable called `BUCKET` and populate it with the initial numeric value of `123456`
    - Can be displayed on the command line (or used as part of a command or substitution) by preceding the variable with the `$` character
        - For example: `echo $BUCKET`
            - Will display the current value of the variable `BUCKET` (in our previous example, this would be `123456`)
    - Variables can be numeric or textual and text variables can contain spaces if escaped or enclosed in quotes
        - For example: `MYVAR="Space Value"` or `MYVAR=Space\ Value`
            - The first method (quotes) is the most common method of assigning text to a variable
    - Export
        - Shows exported variables (when run alone)
        - Keyword that, when preceding a variable, will allow the value to be passed on to other shells or children of the current shell (normal behavior is that the variable value is only visible in the current shell, referred to as **variable scope**)
        - For example: `BUCKET=123`
            - `echo $[BUCKET*2]`
                - This will produce the value `246`
            - Exiting the current `bash` session and creating another will not carry the value to the new child
3. Source (sourcing values from another file)

- Equivalent of *including* the values specified in a file into another command, shell, or other command
- For example: `sourcefile.sh`, contains the variable `BUCKET=123`
  - Capture the variable in `sourcefile.sh` in the current shell: `source ./sourcefile.sh`
    - `echo $BUCKET` from the bash prompt after would then display the `123` value in the variable
- Shortcut for sourcing
  - For example: `source ./sourcefile.sh` can be replaced with `../sourcefile.sh`

4. `unset`

- Variables can be unset with the `unset` command
  - For example: `unset BUCKET && echo $BUCKET`
    - In our example, would produce blank output as it unsets the variable

5. `set`

- Shows all variables and functions in the current environment
- Also allows the enabling/disabling of various shell features
  - For example: `set -x`
- Will print each command to the terminal as it is executed
  - For example: `set +x`
- Will disable the printing of each command to the terminal

105.1 Customize and Use the Shell Environment (Environment and Aliases)

1. `env`

- When run alone, displays current environment variables and their values
- Can be used to modify the current environment right before you run a command (or a script when used at the top)
  - For example: `#!/bin/env java`
    - This will allow the shell to search the environment `PATH` variable for the `java` application before running the command
- Often used to *erase* the current environment before doing something
  - `-i`: Erase (temporarily, just for the command or script) the current environment before running anything after
  - `-c '[command]'`: Will run the command indicated, either in a pristine environment (when used with `-i`) or the current environment (when not used with `-i`)

2. `alias`

- Allows you to substitute one command for another or to set a *shortcut* equal to a more complicated command with switches

    - For example: `alias ll='ls -al --color=yes | more'`
        - Common alias so that the `ll` shortcut becomes an alias in the environment for a full color listing of the current directory, paginated with `more`, as needed
- Can be used to override an existing command
    - For example: `alias rm="echo 'You do not really want to remove that do you?'"`
- Aliases can be overridden by:
    - Escaping the command (i.e., `\rm`)
    - Using full path to the command (i.e., `/bin/rm`)

3. `unalias`

- Removes (for the current environment) the indicated alias
    - For example: `unalias rm`
        - Unsets the override on the `rm` command set in our example above
- `-a`: Remove ALL aliases set in the current environment for the duration of the session
- **Note**: The only way to permanently remove an alias is to remove it from the script (startup or environment) that sets it to begin with

4. `function`

- Allows you to create a "mini application" that can accomplish multiple things
- The `function` keyword must appear right before the beginning of the definition. A function can run over multiple lines and is structured like the following:

```
function myfunc() {
 cp /path/to/dir /path/to/destination
 tar cvf destination.tar /path/to/destination
}
```

    - This function, when called (which, in this case is called like any command, by referring to it by name, `myfunc`), would copy files from one location to another and then create a tar file from that created location called `destination.tar`
- You can use special variables in addition to anything normally available in the environment
    - `$1`: Represents the first option that is passed to the function at the command line
    - `$*`: Method of referring to ALL options at once

5. `profile`

- Located in `/etc`
- Global configuration file that applies its settings to all user environments (as long as they use bash)
    - "Sourced" on each login (is a script)
- Used to set environment variables (like `PATH` or `PS1` for the bash prompt) and some other conditional environment settings based on various conditions (regular users vs. administration/root users for example)

6. `.bash_profile`
    - Located in a user's home directory (i.e., `/home/user`)
    - "Sourced" or executed second (if it exists) as part of the login process (after the global profile)
    - Affects the current user's environment (and ONLY that user's environment)
    - **Note**: Can be called `.profile` - but if both exist, only `.bash_profile` will apply
7. `bashrc`
    - Located in `/etc`
    - Global configuration file that applies its settings to all user environments (as long as they use bash)
    - "Sourced" or executed after the profile from the user's `.bash_profile` (above) if it exists
    - Commonly used to define aliases and functions for the environment for all users
8. `.bashrc`
    - Located in user's home directory (i.e., `/home/user`)
    - "Sourced" or executed (if it exists) from the `.bash_profile` of the user
    - Affects the current user's environment (and ONLY that user's environment)
    - Another file that commonly is used to define aliases, bash prompt appearance, functions for the environment, etc.
9. `.bash_login`
    - Similar to the various "profile" scripts, but located in the user's home directory, will allow you to set environment variables, functions, aliases, etc. on login
        - **Note**: If `.profile` or `.bash_profile` exists, this file will NOT be read/sourced and run
10. `.bash_logout`
    - This is the final script that is sourced/executed during a login session for a user
    - Commonly used to kill user processes, clear the screen from any logged in console, sign out of any applications using credentials, unmount user specific shares, etc.
11. Sample login session
    1. Connect and login user/password
    2. `/home/user/.bash_login` executes (assuming it exists AND `.bash_profile` or `.profile` do not exist)
    3. `/etc/profile` executes (assuming it exists)
    4. `/home/user/.bash_profile` executes (assuming it exists)
        - If not, then `.profile` will execute if it exists
    5. `/home/user/.bashrc` executes (sourced from the user's `.bash_profile` or `.profile`)
    6. Session ends with `logout` or `exit` (or CTL+D), `.bash_logout` is executed

105.2 Customize or Write Simple Scripts (Reading and Writing Files and Executing Commands)

- Shell scripts
  - Simply, a collection of commands that do something, often with or to optional parameters provided when the script is launched
  - Scripts start by identifying the shell type they are to execute their commands in
    - For example: `#!/bin/bash`
      - Indicates that this script will execute using the `/bin/bash` command as the shell it will run within
  - Comments are delineated with a `#` character
    - For example: `# this is a comment`
      - Nothing that is after that character will be interpreted by the shell or script
      - Comments are a single line only; multi-line comments must be preceded with the `#` on each line
  - Commands can be executed within a shell script just as they would on the command line (the script will also assume the permissions of the person executing the command)
  - Parameters
    - These are strings, numbers, etc. that are passed to the environment, along with the command, that the script can use
      - For example: `./myscript.sh /home/user/testfile.txt`
        - This would call the script called `myscript.sh` from the current directory and pass a single parameter into it with the value `/home/user/testfile.txt`
    - Are referred to by the number of the parameter
      - `$[#]`: The number of the parameter that you are using in the command within your script
        - In our example, that would run the designated command on the `/home/user/testfile.tx` file passed in when we started the script
    - Parameter numbering starts with 1, parameters that do not exist will not error but will simply be interpreted as "empty"
  - Example script: `myscript.txt`

```
#!/bin/bash
# create an empty file and add some content to it
touch $1
echo "This is a test file" > $1
# clear the screen and then display the file created
clear
cat $1
```

  - Location
    - `/usr/bin`: System available scripts
    - `/usr/local/bin`: Scripts installed/created only for use on the local system
    - `/home/user/bin`: Scripts created by the `user` user and available only to that user
  - Executable permissions
    - Make sure you set the script executable
      - For example: `chmod 755 myscript.sh`
        - Will make sure that everyone has execute permissions on the script

105.2 Customize or Write Simple Scripts (Testing String, Numbers and Files, Looping and Conditionals)

1. Conditionals
    - Allows you the ability to control the conditions that the commands in your script will run
    - `if/then/else/fi`
        - Structure of an `if` conditional statement, in plain English: "if this then that else then that fi (end)"
        - "Fall down" of `if/then/else` `if/else` `if/else/fi`
            - Structure of a multiple statement and multiple test
        - Used to test variable values and then act on them
            - For example: `if [[ "$?" == "0" ]]`
                - Will determine if the preceding command succeeded or failed
                - **Note**: The variable `$?` is a special variable that can be tested immediately after any command. If the value is `0`, then the command succeeded (did not error). Any other value, the command failed.
        - Can be used to test commands and values directly
            - For example: `if pgrep sshd`
                - Will be true if the `sshd` process returns a PID (i.e., it is running)
    - `!`: Often used to test whether something is NOT something
        - For example: `if !pgrep sshd`
            - Will be true if the `sshd` process is NOT running
2. Test
    - Command to "test" various conditions with a parameter and value
        - For example: `test -f /home/user/testfile.txt`
            - Will test for the existence of a file called `/home/user/testfile.txt`
    - Combined with a conditional like `if`, can be an effective method to test for various conditions on strings, numbers and files
        - `-d [file]`: `file` exists and is a directory
        - `-f [file]`: `file` exists and is a regular file
        - `-h [file]`: `file` exists and is a symbolic link
        - `-r/w/x [file]` - `file` exists and the user has read, write or execute permissions (as indicated by letter)
        - `-s [file]` - `file` exists and is larger than 0 bytes in size
        - For example: `if test -d /home/user`
            - Will test to determine if the `/home/user` file is a directory or not
    - The `test` keyword can be omitted completely using the "square brackets" method (i.e., `[` and `]`)
        - For example: `if [ -f /home/user/testfile.txt ]`
            - Will test for the existence of a file called `/home/user/testfile.txt`

        - **Note**: A space must be after the opening bracket and before the closing bracket or you will receive "command not found" errors
    - Strings
        - Testing string lengths
        - `-n [stringname]`: The string is non-zero length
        - `-z [stringname]`: The string is zero length
        - Equality (`=`) tested with a single equal sign
            - For example: `if [[ "$MYSTRING" = "Some Value" ]]`
                - Tests to see if the `$MYSTRING` variable was "Some Value"
        - Non-equality (`!=`)
            - For example: `if [[ "$MYSTRING" != "Some Value" ]]`
                - Tests to see if the `$MYSTRING` variable was NOT "Some Value"
    - Integers
        - Comparative operators
        - `-eq`: Test for equality
        - `-ne`: Test for non-equality
        - `-gt`: Greater than
        - `-ge`: Greater than or equal to
        - `-lt`: Less than
        - `-le`: Less than or equal to
            - For example: `if [[ $MYAGE -gt 50 ]]`
                - Will test to see if the variable `$MYAGE` is greater than 50
    - Multiple tests
        - Can be combined via the `&&` (and) operator or the `||` (or) operator
            - For example: `if [[ $MYAGE -gt 50 && $MYAGE -lt 100 ]]`
                - Will test to see if the variable `$MYAGE` is greater than 50 but less than 100 (which means I am old, but maybe not dead)

3. Looping

    - Allows you to determine how many times a particular part of your script will run
    - `for/do/done`
        - Loops over a fixed number of items
            - For example: `for city in Cincinnati Cleveland Columbus; do`
                - Will execute the subsequent commands over the three cities in the list
    - `while/done`
        - Loops until the indicated condition is false, loops while true
            - For example: `while [[ $MYLOOP -lt 10 ]]`
                - Will loop as long as the `$MYLOOP` variable is less than 10
    - `until`
        - Loops like `while` was written with a `!` at the end (continues while false, stops when true)

- `seq`
  - Helpful when you need to loop a known number of times
  - `-w`: Pad the output with leading zeros as the length of the number changes
  - `[#] [#] [#]` - beginning number, counting number, ending number (**Note**: When only two parameters are provided, they are beginning/ending)

4. `read`
   - Useful for reading input from a console (and/or file); similar to the `cat` command
   - Can be used to take input from the user
     - For example: `read FIRSTNAME`
       - Will prompt user for a `FIRSTNAME` and store it in that variable to be used later

- `exit`
  - Allows you to return a non-zero error code to the environment which could then be read/logged/reacted to by other applications
    - For example: `exit 66`
      - Will return the error code "66" to the environment and a subsequent `echo $?` would return that value

Topic 106 - User Interfaces and Desktops

106.1 Install and Configure X11 (Installing the X Window System, Desktops, and the Xorg Configuration File)

1. X11 / Xorg
   - X11 was the original graphical desktop for Unix and Linux
   - Xorg, although completely compatible, was designed to replace it and offers additional abstraction for security and performance
2. Window managers
   - Adds the components for the "windows" that are drawn to the screen by the X server (scrollbars, max/min buttons, etc)
3. Desktops
   - The interactive portion of the GUI; sets styles, icons, virtual desktops, etc.
   - Examples:
     - KDE
     - Gnome
     - Unity
     - XFCE

- Common components in desktops:
  - Media players
  - The window manager
  - Control panel (settings)
  - Themes/styles
  - File manager

4. Installation
   - Will be different for almost every distribution
   - For example: `yum groupinstall "X Window System" "Desktop" "Desktop Platform"`
   - For example: `sudo apt-get install xserver-xorg-core unity`
     - Would install all the necessary components for the X Window System in general and then add the default desktop environment for a CentOS 6 distribution that did not have one already (i.e., server install or minimal install)
5. `/etc/X11/xorg.conf`
   - The primary configuration file for the X Window System
   - Major sections (exam)
     - Files: Files that are used by your X server (i.e., fonts)
     - Module: Devices
     - InputDevice: Keyboards and mice (and perhaps special keypads, if detected)
     - Device: Video card(s) and any driver references
     - Monitor: Monitor(s) detected
     - Screen: Description of resolutions and color depths that are supported for the detected monitor and X server
   - Can be created (basic detected values)
     - `X -configure`
       - Will usually place a file called `xorg.conf.new` in the home directory of thee user running the command
     - `Xorg -configure`
       - If the above command does not work, the newer versions of X will use this one
6. XFS
   - Font server for X
   - Provides access to fonts for the X server
   - Test:
     - `service xfs status`
       - Will determine if it is running on Red Hat/CentOS systems
7. `FontPath`
   - If the font server is not in use, this may exist in the `xorg.conf` file
     - For example: `FontPath "/usr/X11R6/lib/X11/fonts/100dpi"`
       - Sets the path to find 100dpi fonts for the X server to use

106.1 Install and Configure X11 (X Utilities for Screen and Window Information)

1. `xwinfino`
    - Displays a plethora of window information on the chosen window on the desktop
    - Running the command will present you with a special pointer to choose the window to query for info
2. `xdpyinfo`
    - Provides information on the current X session
    - **Exam Topic**: Common question around utilities that can be used to display the screen resolution and color depth
3. `xhost`
    - Controls access to the X server itself
    - Run alone, shows you the current access status
    - `+ [IP]`: Disable access control, opening up the server from any host (or from just the indicated IP)
    - `-`: Enable access control, only authorized clients can connect
4. `DISPLAY`
    - Legacy variable that would allow X redirection

106.2 Setup a Display Manager

1. `xdm`
    - Display manager that is part of the Xorg software package
    - `/usr/bin`
        - Directory where it will exist, if installed
    - `xorg-x11-xdm`
        - Package for the display manager
    - Not generally installed/used, unless no full desktop environment is being used
    - `/etc/X11/xdm`
        - Configuration directory
2. `kdm`
    - KDE display manager (legacy)
    - Replaced by `kwin` and later `lightdm` (see below)
    - `/etc/kde/kdm`

        - Configuration directory
    - `/usr/bin`
        - Executable location

3. `gdm`
    - Gnome display manager
    - `/etc/gdb`
        - Configuration directory
    - `/usr/bin`
        - Executable directory

4. `lightdm`
    - Works as a service
        - `systemctl status lightdm` (systemd systems)
    - `/etc/lightdm/lightdm.conf`
        - If it exists, will contain the configuration for the display manager
    - Designed to be a lighter weight display manager (and replaced `kdm`)
    - `/usr/share/doc/lightdm`
        - Directory containing sample configuration file

5. Common function for all display managers
    - Handle login to the desktop

6. Changing preferred desktop manager
    - Red Hat/CentOS (Legacy v6 and before)
        - `/etc/X11/prefdm`: Script that tests for the preferred display manager
            - Set `$DISPLAYMANAGER` value to "lightdm" (unavailable, except newest versions of CentOS)
    - Debian/Ubuntu
        - `dpkg-reconfigure [current desktop manager]`: Allows reconfiguration of display manager, screen should prompt for any installed DM

106.3 Accessibility

1. Sticky/repeat keys
2. Slow/bounce/toggle keys
3. Mouse keys
4. High contrast/large print

5. Screen reader
    - Settings on some desktops
    - Orca
    - `emacspeak` (for emacs editor)
6. Braille display
7. Screen magnifier
8. On-screen keyboard
    - GOK (Gnome On-screen Keyboard)

Topic 107 - Administrative Tasks

107.1 Manage User and Group Accounts and Related System Files (UIDs and GIDs, Password Files)

1. UID
    - User ID
    - A numeric ID from 0 to over 4 billion
    - Typical user accounts will range from 500 to 65,000
2. GID
    - A group ID
    - A numeric ID from 0 to over 4 billion
3. Special UIDs
    - UID 0: Root/admin user on any system
    - UID 1: `bin` user (system binaries and non-login accounts)
    - UID 48: Apache user (if installed)
    - UID 99: `nobody` account (used for a variety of things, FTP anonymous access for example, may also map to a root account for certain NFS configurations (i.e., the `root_squash` option))
4. Special GIDs
    - GID 0: Root/admin group (members of this group have access to restricted resources)
    - GID 1: `bin` group (system binaries and non-login accounts)
    - GID 100: `users` group (put users in this group to give access to resources by assigning the group ownership to this group)
5. `/etc/passwd`

- Defines user specific information like home directory, mapping of the username to UID, the default login shell, etc.
- Default permissions:
  - Red Hat `-rw-r--r--` (644)
  - Debian `-rw-r--r--` (644)
- IDs defined here range from 0 to 499 (predefined system accounts; user accounts are above that range)
- For example: `user:x:1001:1001::/home/user:/bin/bash`
  - `[username]:[password is in shadow file]:[UID]:[GID(primary)]:[Description]:[home directory]:[default login shell]`

6. Special shells
   - `/bin/false`: User cannot login (used for system accounts)
   - `/etc/nologin`: Displays a message that the account is not available (if `/etc/nologin.txt` exists, displays that message instead)
7. `/etc/shadow`
   - Mapping of usernames and (if the account has one assigned) their hashed password value
     - Default permissions:
       - Red Hat `-r--------` (400)
       - Debian `rw-r-----` (640)
   - **Note**: Removing the password (hashed value) from here can allow you to access that user's account if you have sudo rights
     - For example: `user:$6$ejcjmixY$AvmZqcTyRcdmbkdK0BOOYZzwM7Q0g/XnWsvK2Ky1Hxd.EkiL6NA4daiH`
     - `[username]:[encrypted password]:[last change of pwd]:[max days before required change]:[warning days before expiration]:[grace period after expiration]:[expiration date]:[reserved]`
     - All of this is controlled on legacy systems by the Shadow Suite, newer systems use the more powerful PAM modules
8. `/etc/group`
   - Defines group specific information like mapping of the group name to the group ID, other members of the group, etc.
   - Groups can contain multiple users
     - For example: `user:x:1001:test,temp`
       - `[group name]:[group password in shadow file]:[GID]:[secondary group members - if any]`
9. **Note**: When logging into the system, a user MUST use their username. Logging in via UID is not permitted, even if the UID exists.
10. `getent`

- Utility to allow you to search both local (`/etc/passwd` and `/etc/shadow`) for account information, as well as network sources (i.e., LDAP)
    - For example: `getent passwd user`

11. `/etc/nsswitch.conf`

    - determines the order that user account information is searched for on a system during login
        - For example:
          ```
          password: files nis
          shadow: files nis
          ```
            - Will search for LOCAL files for account information and then search the remote user database configured if not found

12. Special log in files

    - `/etc/motd`: If exists, displays the contents of this file
    - `.hushlogin`: If it exists in the home directory of the user, will prevent the login from checking mail or displaying the previous login information
    - `/etc/login.defs`: Defaults for a user when created with the `useradd` command
    - `/etc/securetty`: Defines where the root user is allowed to log in (if it does not exist, root can login from anywhere)
    - `/etc/usertty`: Defines parameters for user logins (locations, days or times, etc)
        - **Note**: Only used if the system does not have PAM modules (pluggable authentication modules)

107.1 Manage User and Group Accounts and Related System Files (Managing User Accounts)

1. NOTE: Although it is possible to create a user by directly editing the `/etc/passwd` and `/etc/group` files, this poses security risks AND can contain errors that will prevent user login or cause unintended consequences

2. `useradd`

    - Standard utility to add new users to a system
    - Can behave differently on Red Hat based vs. Debian based systems, best bet is to run the command with explicity parameters
    - `-m`: Create home directory
    - `-d [directory]`: Specify the home directory to create
    - `-k [skeleton directory]`: Copies the contents of the specified directory (usually `/etc/skel`) to new user's home directory
    - `-g [primary GID]`: Specify the user's primary GID
    - `-u [UID]`: Assign the indicated UID to the user (**Note**: Will error if exists)
    - `-e [date]`: The date this account is disabled after

- `-G [secondary GID]`: Allows you to set a secondary group for the user
- `-f [# days]`: sets the number of days after a password reaches max age that the account will still allow login
- `-o [non-unique UID]`: Allows the creation of a user with a UID that is NOT unique (duplicate)
- `-s [path and file of login shell]`: A full path and name of the default login shell for the user

3. `/etc/skel`
    - The contents of this directory can be copied to a new user's home directory depending on how the user is added
4. `/etc/default/useradd`
    - Contains the default values for the `useradd` command when those parameters are not used
5. `groupadd`
    - Much like `useradd`, will add a new group as indicated
    - **Note**: Does not add users to the group
    - `-g [GID]`: Create the group with the indicated GID
6. `usermod`
    - Modify the characteristics and/or membership of existing users
    - `-c [description]`: Modifies the user description in the `/etc/passwd` file
    - `-d [new home directory]`: Changes the user's home directory
    - `-e [date]`: Change the date of account expiration
    - `-f [# days]`: Change the number of days after a password reaches max age that the account will still allow login
    - `-g [GID]`: Change the user's primary GID
    - `-G [GID]`: Change the user's secondary GID(s) (can be multiple groups in a comma delimited list)
    - `-s [path and file of login shell]`: Changes the full path and name of the default login shell for the user
    - `-u [UID]`: Changes the UID (**Note**: Will change home directory to match, but not any other user owned files)
    - `-L`: Locks the user's account
    - `-U`: Unlocks the user's account
7. `groupmod`
    - Modify the characteristics of the indicated group
    - `-g [GID]`: Alters the GID of the indicated group

8. `userdel`
    - Removes the indicated user's account
    - `-r`: Also removes all user mail, owned print jobs, cron jobs and the home directory with all contents (**Note**: All other files owned by the user will still exist, ownership reverting to the UID of the removed user, thus becoming orphaned)
        - **Note**: You cannot delete a user that is in use or has a process associated with the account
9. `groupdel`
    - Removes the indicated group
    - Files/directories that are owned by the group will then revert to the GID; you can change ownership at that point
10. `passwd`
    - Set/change the password on the indicated user
    - **Note**: Users can change their own password, but not other user's password (without sudo), root can change any user's password
11. `chage`
    - Change the aging parameters of the indicated user's account and password
    - Changes values in the `/etc/shadow` file
    - `-m [# days]`: How long user must wait (in days) between password changes
    - `-M [# days]`: How long before a user must change their password
    - `-d [date]`: Sets the last changed value for the password
    - `-E [date]`: Changes the expiration value
    - `-I [# days]`: Number of days inactive after expiration or max limit before account is locked
    - `-W [# days]`: Warning for the number of days before a user must change their password
    - `-l`: Display all values for the indicated user

107.2 Automate System Administration Tasks by Scheduling Jobs (cron, crond and crontab jobs)

1. `cron`
    - The primary job scheduling system in Linux
    - Jobs are configured to run on a fixed schedule (either once or multiple times as needed)
    - `crond`
        - The service that is responsible for making sure cron jobs are run
    - `/etc/cron.*`
        - `cron.d`: Custom job schedule configuration directory (system cron jobs)

- `cron.hourly`: Jobs that run hourly
- `cron.daily`: Jobs that run daily
- `cron.weekly`: Weekly jobs
- `cron.monthly`: Monthly jobs
- **Note**: In all but `cron.d`, these directories are just scripts with no other scheduling information included in them and they will not always run at the same time, but will run within the specified time "frame"

2. `crontab`

- Utility to allow the creation of jobs (specific to user running the command)
- `-l`: List all cron jobs for the logged in user
- `-e`: Edit the cron jobs for the logged in user
- `-u [username]`: Apply the option to the user indicated

3. Format of a cron job entry

- `[Minute 0-59][Hour 0-23][Day of Month 1-31][Month 1-12][Day of Week 0-7][CMD]`
  - **Note**: If a system cron job located in `/etc/cron.d`, there will be a sixth field that will dictate which `USER` will run said cron (see `/etc/cron.d/raid-check` for example)
  - **Note**: In Day of Week, both number 0 and 7 indicate Sunday
  - **Note**: In time, 0 is midnight
- Each column must have a value, even if the value is `*` which means it applies to all potential values in the field
  - For example: `30 23 * 2 7`
    - This job will be scheduled for 30 minutes past the 23rd hour of the day (11:30 PM), on any day of the month of February that is also a Sunday
    - **Note**: Using a three letter abbreviation, you can substitute the name of the month for the fourth field as well as the name of the day for the fifth field
    - For example: `30 23 * feb sun`
      - Equivalent to the previous example
- Ranges
  - A range can be handled with a dash between two values
    - For example: `30 23 * 2-4 7`
      - This job will be scheduled for 30 minutes past the 23rd hour of the day (11:30 PM), on any day of the months of February through April, that is also a Sunday
  - Can also be designated by a comma separated list
    - For example: `30 23 * 2,3,12 1,3`
      - This job will be scheduled for 30 minutes past the 23rd hour of the day (11:30 PM), on any day of the months of February, March and December that is also a Monday or Wednesday
  - Values can also be designated in "step" format
    - For example: `0 */4 * 2,3,12 1,3`

        - This job will be scheduled at the top of the hour, every fours hours, on any day of the months of February, March and December that is also a Monday or Wednesday
    - Job / command to run
        - Any valid shell command, application or script
        - **Note**: The cron job for a user will have a very minimal environment (`.bash_profile`, `.bashrc`, `.bash_login`, etc.) does not run, so providing a full path to commands and or variables need to be included in scripts or you may get errors or unexpected results (`PATH`, for example, will have only `/usr/bin` and `/bin`)
        - **Note**: You can set some special environment variables above your job
        - `MAILTO`: Output from the job will be emailed to the indicated address
        - `SHELL`: Run the job with the indicated shell
        - `CRON_TZ`: Use the indicated timezone for the crontab
    - Job redirection
        - Other than the aforementioned `MAILTO` variable, you can direct output to `/dev/null` or another file
            - For example: `0 */4 * 2,3,12 1,3 /path/application/script.sh > /dev/null`
                - Will effectively "throw away" all output (`/dev/null`)
            - For example: `0 */4 * 2,3,12 1,3 /path/application/script.sh 2>&1 > /root/some.log`
                - Will take all standard and error output and redirect it to the `/root/some.log` file

4. `/var/spool/cron`
    - All user crons created/edited with crontab are located here
5. `/etc/cron.allow`
    - Whitelist of users who can run cron jobs
    - If this file exists and is empty, only root can access crontabs
6. `/etc/cron.deny`
    - Blacklist of users who cannot run cron jobs
    - If this file exists and is empty, all users can access their crontabs and run jobs
    - **Note**: Order of precedence will apply `cron.allow` and ignore `cron.deny` if it exists

107.2 Automate System Administration Tasks by Scheduling Jobs (Anacron and Other Scheduled Commands)

1. `anacron`
    - "Simplified" cron, used to augment `crond`
    - Runs jobs that can be run with less time precision, particularly catching up on running jobs that were scheduled while the system was shut off

2. `/etc/anacrontab`
    - This is where all jobs are defined
    - All jobs will run as root user
    - No job can be run more than once per day
    - All jobs are run consecutively
    - Service is available, but generally run from `cron.daily`
3. Format of job
    - Environment variables at the top
    - `[period in days][delay in minutes][job-identifier][command to run]`
        - For example: `1 5 cron.daily nice run-parts /etc/cron.daily`
            - Runs `cron.daily` scripts once per day, delayed by five minutes from system start/boot, running all of those jobs using the `nice` utility to `run-parts` (which is a utility to run each file in the given directory, in this case `/etc/cron.daily`)
4. Batch scheduled jobs
    - Sometimes referred to as "ad hoc"; you just want to run something in the future, but just once
5. `at`
    - May not be installed, install package `at`
    - Start `atd` service
    - A method for running ad hoc jobs
    - Uses a special prompt for entry of the command to run at the scheduled time
        - For example:
          ```
          at 11pm today
          at> wall "Log off everyone" <-- end with CTL+D
          at> <EOT>
          ```
            - Will schedule the `wall` command to display the message indicated at 11 PM today
        - For example: `echo "wall \"Log off everyone\" " | at 23:00 today`
            - Alternative method for the same job
    - `-l`: List all `at` jobs currently scheduled
6. `atq`
    - Shows a summary of all jobs scheduled with `at`
    - Will not show the details, but a time and job ID
7. `atrm`
    - Allows you to remove the job ID indicated

8. `/var/spool/at`
    - Directory containing the jobs to run
9. `/etc/at.allow`
    - Whitelist of users who can run `at` jobs
    - If this file exists and is empty, only root can access `at`
10. `/etc/at.deny`
    - Blacklist of users who cannot run `at` jobs
    - If this file exists and is empty, all users can access `at` and run jobs
    - **Note**: Order of precedence will apply `at.allow` and ignore `at.deny` if it exists

107.3 Localization and Internationalization (Dates and Timezones)

1. Determining time on a Unix/Linux system
    - Time is tracked from the "epoch", which is a special date known as the "birthday" of Unix systems, or January 1st, 1970
    - It exists as a timestamp, in seconds, since that time
2. `date`
    - Displays the current date and time along with the configured timezone
    - You can control the format of the output with the `+` character followed by special formatting instructions
    - `%d`: Two digit date
    - `%H`: Two digit hour, 24 hour format
    - `%m`: Two digit month
    - `%M`: Two digit minute
    - `%Y`: Four digit year
    - `%z`: Time zone offset
    - For example: `date +"%H:%M-%m%d%Y-%z"`
        - Would display the time as "23:11-04212017-0500"
3. Time zone
    - A name representing an "offset" from the UTC (Universal Coordinated Time), also known as GMT (Greenwhich Mean Time)
4. `/usr/share/zoneinfo`
    - the top level directory containing all time zone definitions
    - **Note**: These are binary files and cannot simply by viewed on the console

5. `/etc/localtime`
    - The system time zone (can be a full time zone copy OR a link to the configured time zone)
6. `TZ` (variable)
    - Allows you to override the system wide time zone setting in the above directory
    - Often set in a user's home directory as part of the `.bashrc` file
7. `tzselect`
    - Allows you to find the name of the time zone you want to use
    - **Note**: This command does not change the time zone
8. `timedatectl` (Red Hat based systems) - `dpkg-reconfigure tzdata` (Debian based systems)
    - `list-timezones`: List all the timezones to choose from
    - `set-timezone [country/zone]`: Set to the indicated timezone
    - Make the actual changes, setting the `/etc/localtime` system setting to the time zone chosen
9. `/etc/timezone`
    - Local file that Debian systems use to store the name of the time zone configured

107.3 Localization and Internationalization (Locales and Character Encoding)

1. `locale`
    - A way of representing your language, country and encoding type
    - `-a`: Show you the locales that are installed on your system
2. Linux and locales
    - GNU `gettext`
        - Library that handles internationalization
        - Relies on environment variables to determine which locale is necessary
            - `LANGUAGE`: Used when displaying messages, but does not affect formatting
            - `LC_ALL`: Force the indicated locale even if other variables are set differently
            - `LC_XXX`: Administrator variable to override a locale for any element in the locale
            - `LANG`: Encoding type
3. ASCII
    - American Standard Code for Information Exchange
    - Encoded into 7 bits, giving 128 total possible characters

    - English encoding
    - Once those characters ran out, storing went to 8 bits, giving another 128 possible characters

4. Code pages

    - Character set definitions (where unused characters are assigned their own character)
    - Vendor specific, changing between the characters requires changing the code page
    - ISO-8859 standard
        - Definition of standard code pages
            - ISO-8859-1: Latin alphabet with english characters
            - ISO-8859-3: Turkish characters with other language characters
    - Was determined to be too chaotic/sprawling

5. Unicode

    - Defines every character as a number (code point)
    - Originally encoded in 2 bytes, giving you 16k possible characters (called UCS-2)
    - Once again, the number of characters in this spec was exceeded
    - UTF-16 was introduced to allow any character over 16k to be represented with a second pair of bytes
    - UTF-8 allows 1 to 6 bytes to be used to encode a character, with the character length encoded into the high order bits of the character number and maintains full compatibility with the original spec 128 character ASCII code
    - UTF is the dominant encoding type

6. `iconv`

    - A utility used to convert between character encodings
    - `-c`: Clears unknown characters
    - `-f [type]`: From indicated type
    - `-t [type]`: To the indicated type
    - `-l`: Lists all available encoding types
        - For example: `iconv -c -f ASCII -t MACCYRILLIC VNCHOWTO > VNCHOWTO.new.cyrillic`
            - Would clear any unknown characters in the file stream from `VNCHOWTO`, and convert from ASCII to MACCYRILLIC encoding, writing the new file the `VNCHOWTO.new.cyrillic`
    - **Note**: This is not a language translator, simply a character encoding translator

Topic 108 - Essential System Services

108.1 Maintain System Time (Hardware Clock, System Clock and NTP)

1. System clock

- A kernel module that responds with time values when writing files or being queried on date/time

2. `date`

   - Displays the current date and time along with the configured timezone
   - You can control the format of the output with the + character followed by special formatting instructions
   - `%d`: Two digit date
   - `%H`: Two digit hour, 24 hour format
   - `%m`: Two digit month
   - `%M`: Two digit minute
   - `%s`: Number of seconds since the epoch
   - `%Y`: Four digit year
   - `%z`: Time zone offset
   - For example: `date +"%H:%M-%m%d%Y-%z"`
     - Would display the time as "23:11-04212017-0500"
   - `-u`: Displays the system UTC time
   - Can also be used to change the date/time by omitting the + before the value
     - Format is `MMDDhhmm[[CC]YY][.ss]]`
       - `MM`: Two digit month
       - `DD`: Two digit day
       - `hh`: Two digit hour, 24 hour format
       - `mm`: Two digit minute
       - `CC`: Two digit century (optional)
       - `YY`: Two digit year (optional)
       - `.ss`: Two digit seconds (optional)

3. Hardware clock

   - The clock that is on the system motherboard (or on the virtual host's motherboard), usually set and then maintained by battery during power off

4. Drift

   - The amount of time that the hardware and system clocks differ

5. `hwclock`

   - Allows you to work with the hardware clock directly
   - Outputs the hardware clock date/time
   - **Note**: The hardware clock is unaware of time zones
   - For example: `hwclock` - returns something like "Mon 19 Jun 2017 09:13:54 PM UTC -0.965537 seconds"

        - The offset at the end if the amount of time from when the command was executed and the clock was read until display of the value
    - You can synchronize the hardware and system clock two ways:
        - For example: `hwclock -w` (or `--systohc`)
            - Will synchronize the system clock to the hardware clock
        - For example: `hwclock -s` (or `--hctosys`)
            - Will synchronize the hardware clock to the system clock

6. `/etc/adjtime`
    - Contains values that track calibration to the clock and a final value to display time in LOCAL or UTC time
    - If the file does not exist, no calibration has ever been done and time will default to UTC
7. `ntp` / `ntpd`
    - May need to install `ntp` package for the utilities here
    - Network Time Protocol
    - Network time protocol daemon (service)
    - Allows you to define a pool of network servers that are synchronized to a globally distributed network of time servers
    - Those that get a time update from a "reference" clock (like the Naval observatory in the USA) are called "stratum 1 servers"
8. `ntpdate`
    - Allows you to set the clock against the indicated NTP server
9. `pool.ntp.org`
    - Site displaying names of public NTP servers
    - Aliases commonly configured: 0.pool.ntp.org through 3.pool.ntp.org
10. `/etc/ntp.conf`
    - Pool of NTP servers
    - Defines drift file to track clock drift
11. `/var/lib/ntp/drift`
    - File that track system time drift
12. `ntpq`
    - Query an NTP server for stats and connect to local system by default
    - Special prompt
        - Peers: Time hosts already associated with
        - Associations: More details on each server

108.2 System Logging

1. `syslog` (sometimes `rsyslog`, `klogd`)
    - Simple protocol to log system messages
    - Applications can access the facility through a syslog library call or via command line `logger` utility
    - Daemon (`syslogd`) will process the message, levels of logging can be:
        - `0`: Emergency
        - `1`: Alert
        - `2`: Critical
        - `3`: Error
        - `4`: Warning
        - `5`: Notice
        - `6`: Info
        - `7`: Debug
    - Logging at a particular level restricts messages logged to that level or above (log level 3 means 0,1,2,3 events are logged while 4 and below are dropped/ignored)
2. `rsyslog`
    - Alternative "fast" version of `syslog`
3. `syslog-ng`
    - Alternative "next generation" `syslog`
4. Logging "facilities"
    - Specific logs that are tied to various items (describing what is logged)
        - `kern`: Kernel messages
        - `user`: User level messages
        - `mail`: Mail server
        - `daemon`: Services/daemons
        - `auth`: Security logs (public)
        - `syslong`: Internal syslog messages
        - `lpr`: Printing
        - `cron`: Job scheduling
        - `authpriv`: Security logs (private)
        - `local0-7`: User definable (8 different ones are available)
5. `/var/log/message`
    - Where all log messages go (except `mail`)
6. `/var/log/maillog`

- Mail messages are written here

7. `logger`

   - Allows you or a command to log a message to `/var/log/messages`
   - CTL+D to end and write the message
   - `-i`: Passed additional information to syslog

8. `/etc/syslog.conf`

   - Syslog configuration file
   - Defines where certain facilities will write logs

9. `systemd`

   - Uses its own logging system called `journal` (with `journald` being the daemon for it)
   - Adopted on most modern `systemd` based distributions
   - Primarily, the difference lies in that the logging is done to a binary rather than plain text file, allowing you the ability to query metadata, command line details, PIDs, binaries and security privileges (some of which just are not available with a plain text file)
   - Because it is part of the service management system, all daemon messaging is automatically logged rather than the `sysvinit` variant of `syslog` wherein each service is responsible for how and what messages are logged

10. `/var/log/journal`

    - Binary file wherein the `systemd` log is stored

11. `journalctl`

    - The command used to view the aforementioned journal file on the console
    - `-f`: Allows you to follow as new log messages are written
    - `SYSLOG_IDENTIFIER=[value]`: Allows you to add a filter to just view matching log entries
    - `-u [service]`: Only display messages logged from the indicated service
    - `-o verbose`: Display a LOT more information about the requested service or filter
    - `-e`: Jump to the end of the log
    - `-x`: Adds information to explain the context under which a logged event or error occurs (like in service start failures)

12. `/etc/systemd/journald.conf`

    - Configuration file for `journald`
    - Common settings for size of log and whether logging is forwarded to `syslog` as well (or installed equivalent)

13. Log rotation

- Periodically backup, compress and/or remove log files meeting certain criteria in order to manage disk space and I/O performance
- `/etc/logrotate.conf` (and anything in `/etc/logrotate.d`)
  - Primary configuration for log rotation (defaults and system files to rotate)
  - Each file in `logrotate.d` adds or overrides settings to the defaults in the configuration file
    - **Note**: Files in `logrotate.d` are usually added/maintained by the package manager as part of the install/update/removal of packages
- Open files in log rotation are dealt with in one of three ways:
  - Move the log, signal file close and reopen logs; not all applications will support this method
  - Restart the application/service after moving the log file, should begin logging in a new "empty" file; can be problematic if restarting the service affects users or connections
  - Copy the log and then truncate the existing log in place; expensive in terms of disk performance and log entries can be lost during the transaction

108.3 Mail Transfer Agent (MTA) Basics (Email Overview)

1. Mail servers
   - Sendmail
     - Legacy email server, probably the "standard" that distributions would install. Prone to security issues and was notoriously difficult to configure, using the "M4" language to do so.
   - Postfix
     - Developed originally by IBM, largely replaced Sendmail as the default mail server on modern distributions. Incorporates policy support to reduce spam and separates processes for security purposes.
   - Qmail
     - Secure replacement for Sendmail. Separate processes allow for better security and isolation and a much simpler configuration make this a good choice. However, development has stagnated recently.
   - Exim
     - Like Sendmail in that it is considered more "monolithic" in type and configuration, although it is known to have a better security history. The configuration is simpler and is sometimes seen as the default mail server in older versions of Debian distributions.
2. SMTP
   - Simple Mail Transfer Protocol: It is a protocol that really defines how e-mail is transferred and saved and is part of the TCP/IP application layer as well as settings rules that email applications follow.

3. MUA
    - Mail User Agent: This is whatever application you use to create and send email (Thunderbolt, Evolution, SquirrelMail, etc).
4. MSA
    - Mail Submission Agent: Acts as an intermediary or gateway between the MUA and an MTA to start the transfer of e-mail.
5. MTA
    - Mail Transfer Agent: Accepts email from the MUA and sends it (if needed) to the receiving mail server (another MTA if this is not the destination). There are a number of MTA server in Linux (Postfix, which we will use, Sendmail, and more).
6. MDA
    - Mail Delivery Agent: Receives email from the MTA and then delivers it to the local mail spool for retrieval by any of dozens of client email applications. Sometimes an MTA can also function as an MDA, but often (`procmail` for example), they are independent applications that can also filter mail (like spam).
7. POP
    - Post Office Protocol: Used by MUAs to get email (covered in more detail later).
8. IMAP
    - Internet Message Access Protocol: Used by MUAs to get email (covered in more detail later).
9. MX Record
    - These are the mail DNS records we created earlier in this course. These records are used by MTAs to determine the authoritative mail server for any particular email message.

108.3 Mail Transfer Agent (MTA) Basics (Creating Aliases)

1. Although configuration and installation of mail servers are NOT part of the exam, try to follow along on your servers; simply install `sendmail` and start the service
2. `/etc/aliases`
    - Ability to refer to a user by another name/account
        - For example (entry in file): `sysadmin: root`
            - Would indicate that an email arriving on the system for the `sysadmin` account would actually be delivered to `root`
    - Format of an alias: `alias: account[,another,another]`

        - The brackets are optional, defining multiple accounts that an alias refers to is in a comma delimited list
    - **Note**: An alias can be any name whether an account or not, can also be used to send an account's email to another account
        - For example: `root: user,user1`
            - Send `root`'s mail to `user` and `user1`
    - Can also accept email address as an account to send local mail to (if mail server is configured)
        - For example: `sysadmin: user@linuxacademy.com`
            - Would send `sysadmin` emails to `user@linuxacademy.com` address
    - Email can also be sent to a file from an alias
        - For example: `sysadmin: /root/sysadmin.email`
            - Would send all email to the file `sysadmin.email` in the root directory
    - Can be sent to a script/application
        - For example: `support: | /opt/HelpDesk/create_ticket.sh`
            - Would send email and cause a script to execute (notice the script must be preceded with a pipe `|` character)

3. `/etc/aliases.db`
    - Database file that your local MDA will read to determine where to send email; it has to be updated when any changes are made to the aliases on the system

4. `newaliases`
    - The command that updates the `aliases.db` with the changes in the aliases file above
    - Errors will display if found, success will return no results
    - If you make a change to aliases and do not run this command, it will be logged in `/var/log/messages` that the DB is older than the aliases file

5. `.forward`
    - Home directory file for users to define their own forwarding rules
    - No alias needs to be indicated, since it is for the account it exists within; only need to indicate the account to forward it to
    - **Note**: If forwarding email to an external address AND you want the email to also be delivered locally, you can "escape" the alias
        - For example:
          `\user`
          `someone@linuxacademy.com`
        - Both the local `user` and external email address would receive that account's email

6. `mailq`
    - Command that shows the mail queue

7. `/var/spool/mqueue`
    - Email that is waiting to be delivered
    - Will ONLY exist if mail is in the queue; on a correctly configured server, it should be empty
8. `/var/log/maillog`
    - Tracks all of the email messages as they pass in and out of the system

108.4 Manage Printers and Printing (CUPS and `lpr`)

1. `lpd`
    - The traditional "line printing daemon", before CUPS, was the default method to output to a printer/printing device
    - **Note**: For purposes of practice, you will install `lpd` which will install the `cups-lpd` package so that they work with all legacy commands and `cups-pdf` to install a printer to work with
2. `ipp`
    - Internet printing protocol (web based)
3. CUPS
    - Common Unix Printing System
    - Combination of converters, filters and "printer drivers" that a document can be sent to when output to printing devices, scripts to create SVG, PDF or other formatted documents
4. `cupsd`
    - Daemon for CUPS
    - CUPS operates on port 631 and can be accessed through your local browser (*http://localhost:631*)
5. `/etc/cups`
    - Configuration directory for CUPS
    - Configuration file list:
        - `classes.conf`: Configures class definitions
        - `cupsd.conf`: Primary configuration file for the daemon
        - `cupsd.conf.default`: Sample default configuration file to revert to as backup
        - `printers.conf`: Configuration of each printer on the system
        - `ppd`: Directory of PPD (printer driver files) on each printer on the system
6. `cupsreject`
    - By indicating a printer name, it will set that printer to reject all submitted jobs

7. `cupsaccept`
    - By indicating a printer name, it will set that printer to accept all submitted jobs
8. `cupdisable`
    - Disables the indicated printer (but will still accept jobs, will just hold in queue)
9. `cupsenable`
    - Enables the indicated printer
10. `cupsctl`
    - Used to control CUPS configuration, run with no options, displays the current configuration
11. `lpadmin`
    - `lp` compatibility utility to manage the `lp` queue (remember, compatibility plugin with CUPS, so will apply to the whole printing configuration locally)
    - Can be configured to add a test printer with the following command:
        - `lpadmin -p MyTestPrinter -E -v /dev/null -m raw`
12. `lpstat`
    - Show current `lp` configuration (printers and defaults)
    - `-p`: Show printers
    - `-d`: Show defaults
    - `-a`: Display all print queue and status
    - `-r`: Indicates status of service
    - `-s`: Summary of system configuration
    - `-t`: Verbose summary of system configuration
13. `lpoptions`
    - Allows the setting of printer options at the command line
    - `-d [printer]`: Sets the default printer to the indicated value
    - `-p [printer]`: Deals with the indicated printer
    - `-l`: List options for the indicated printer/queue
        - For example: `lpoptions -p CUPS-PDF -l`
            - Would display all options for the CUPS-PDF printer
14. `lp`
    - Command line (legacy) utilities for printing
    - For example: `echo "my test print job" | lp`

        - Would print to the default printer the results of the echo command
    - `-d [printer]`: The destination printer (if not default)
    - `-n [#]`: Print indicated number of copies

15. `lpr`

    - Command line (legacy) utilities for printing
        - For example: `echo "my test print job" | lpr`
        - Would print to the default printer the results of the echo command
    - `-P [printer]`: The destination printer (if not default)
    - `-#[#]`: Print indicated number of copies

16. `lpq`

    - Displays the print queue and jobs
    - `-a`: Display all print jobs for all printers
    - `-P [printer]`: Display print jobs for the indicated printer

17. `lprm`

    - Allows you to remove print jobs
    - **Note**: Running the command without a job ID will remove the first job on the queue
    - `-P [printer]`: Work with the indicated printer
    - `-a`: Remove all print jobs in the queue

Topic 109 - Networking Fundamentals

109.1 Fundamentals of Internet Protocols

1. IP (Internet Protocol)

    - A method of uniquely identifying an address (destination) for a specific system. Two primary versions:
        - IPv4: Standard address structure of four "octets" containing numbers between 0-255 for each (example: `192.168.72.211`)
        - IPv6: Intended as a replacement for IPv4, consists of a 128 bit hexadecimal number for addressing (example: `2DAF:FF40:0928:CD01:4433:00DD:0988:FFFF`)

2. TCP (Transmission Control Protocol)

    - The method by which all transactions between IP addresses (of any version) are communicated. Defines a system of transmissions and acknowledgement to verify traffic arrives and can be assembled in the correct order.

3. UDP (User Datagram Protocol)

- Often considered "complementary" to IP, but is a "stateless" connection. No error checking or retransmission of packets takes place, even if the transmission of the packet failed.

4. ICMP (Internet Control Message Protocol)

    - Designed for networking devices (routers, intelligent switches, firewalls, etc.) to send error messages. In addition, it can perform queries around network service availability (as in the case where the `ping` command is used to test whether and address responds to a request).

5. IP Class Ranges

    - Five ranges are specified in the RFC 1918. The values in each determine the total number of hosts that are available in each class (addresses):
        - Class A: 1 through 126 – this leaves 16,777,214 host addresses available
        - Class B: 128 through 191 – this leaves 65,534 host addresses available
        - Class C: 192 through 223 – this leaves 254 host addresses available
        - Class D: 224 through 239 – multicast and not used for host addresses
        - Class E: 240 through 254 – reserved for future use

6. Network Mask

    - Defines a logical network (called a subnet) that indicates the start and ending of a range of addresses.

7. Ranges

    - Each address class range has an associated network/subnet mask. The bit number (designated by the `/`) is called the Classless Inter-Domain Routing notation (CIDR).
        - Class A: 255.0.0.0 (or /8)
        - Class B: 255.255.0.0 (or /16)
        - Class C: 255.255.255.0 (or /24)

8. Gateway

    - The destination where network traffic goes that has no other matching route or is not intended for the local system network itself.

9. Broadcast Address

    - Every network has one, operating on the address number 255.
    - For example: 192.168.1.0/24 network
        - 192.168.1.255 is the broadcast address
    - This is how an IP network sends traffic that can be seen by all hosts on a network.

10. Common Ports/Services (for the Exam)

    - Expect any of these to show up on the exam:
        - 20/21: FTP

- 22: SSH
- 23: Telnet
- 25: SMTP
- 53: DNS
- 80: HTTP
- 110: POP3
- 123: NTP
- 139: NETBIOS
- 143: IMAP
- 161/162: SNMP
- 389: LDAP
- 443: HTTPS
- 465: SMTPS
- 514: SYSLOG
- 636: LDAPS
- 993: IMAPS
- 995: POP3S

109.2 Basic Network Configuration (Working with Interfaces and Routes)

1. `nmcli`
    - The NetworkManager command-line interface utility
    - `dev`: Shows device information (relating to the networking hardware)
    - `con`: Shows the connection configuration information
2. `ifconfig`
    - Utility to view all active interfaces (including the loopback adapter)
    - `-a`: Force the display of ALL interfaces (active or not) in the report
    - Key fields:
        - `ether`: Hardware (MAC) address, a 48 bit interface adapter's physical address
        - `inet`: The network address assigned to that interface
        - `broadcast`: Broadcast address for the system's network
        - `netmask`: Network mask or logical network segment information
    - **Note**: This is a legacy network command that is being phased out and replaced by the next one
3. `ifup`
    - Brings up the indicated network interface
4. `ifdown`
    - Brings down the indicated network interface

5. `ip`
    - Unified network and routing management command designed to replace the functionality of most of the other commands you need to know for the exam
    - For example: `ip addr show`
        - Will provide you the same information as the generic `ifconfig` command above
6. Interface configuration files
    - Red Hat/CentOS (v6)
        - `/etc/sysconfig/network-scripts`
            - A directory containing a host of scripts responsible for the configuration of all interfaces on the system
                - For example: `ifcfg-eth1`
                    - Responsible for the configuration (static or DHCP) of the address information for the ETH1 interface on your system
        - Changes to the network interface configuration are applied by restarting the network service
            - For example: `service network restart`
    - Debian/Ubuntu (not tested in the LPIC exam)
        - `/etc/network/interfaces`
        - Configured with `system-config-networking` or `netcardconfig`
        - Changes to the network interface configuration are applied by restarting the networking service
        - For example: `/etc/init.d/networking restart`
7. Default gateway
    - The destination of ALL traffic whose destination is not on the local system network OR does not have another matching static route configured
    - Can be configured within either `/etc/sysconfig/network` or `/etc/sysconfig/network-scripts/ifcfg-` (where # is the interface number)
        - For example: `GATEWAY=192.168.1.1`
            - Would configure the system's default gateway for the IP 192.168.1.1
8. `route`
    - Displays the current routing table
    - Adds/removes routes as indicated
        - For example: `route add default gw 192.168.1.1`
            - Would manually add a default gateway to the system going to 192.168.1.1
        - For example: `route add 192.168.10.211 lo`
            - Would route all returned traffic to the loopback adapter (quick and dirty solution for DoS attack from a single IP)
    - `add`: Adds a gateway/route/destination

- `default`: Key word for adding/removing a default gateway
- `gw`: Short for gateway, all traffic not matching other rules is routed here
- `[IP Address]`: The IP of the route/gateway/network

9. IP forwarding
    - The ability for your host to forward packets to another location and respond
    - Allows your system to function as a router
    - Two methods to enable:
        1. `echo 1 > /proc/sys/net/ipv4/ip_forward`
        2. edit `/etc/sysctl.conf` and add `net.ipv4.ip_forward=1`
        - **Note**: Method one is not permanent but will take immediate effect, method two requires a reboot (or combined with method one)
10. Renewing a DHCP address lease
    - `dhcpd -k`: Kills and restarts the daemon to get another lease
    - `dhclient`: After restarting network services, run to get another lease
    - `pump`: Legacy utility to obtain a new DHCP address
11. `hostnamectl`
    - Command that is used to configure a local system's hostname persistently

109.3 Basic Network Troubleshooting

1. `ping`
    - Utility to test the response from a particular network address
    - `-n [#]`: Ping indicated number of times
2. `netstat`
    - Can print network connections, routing tables, interface stats, etc.
    - `-a`: Show all sockets on active interfaces
    - `-c`: Refresh stats every 1 second
    - `-p`: Shows name and PID for each socket
    - `-t`: Show TCP stats
    - `-r`: Show the routing table
    - `-n`: Do not attempt name resolution (IP only)
3. `traceroute` (root only)
    - Utility to determine the distance (in hops) between your system and a desired endpoint, as well as the response time of each hop along the way

- `-n`: Do not attempt name resolution for each hop, IP only

4. `traceroute6`
    - IPv6 equivalent to `traceroute`
5. `tracepath` (all users)
    - Utility to determine the distance (in hops) between your system and a desired endpoint, as well as the response time of each hop along the way
    - `-n`: Do not attempt name resolution for each hop, IP only
6. `tracepath6`
    - IPv6 equivalent to `tracepath`
7. `ping6`
    - IPv6 ping equivalent to `ping`

109.4 Configure Client Side DNS

1. `/etc/hosts`
    - Local file container name(s) associated with an IP
2. `/etc/resolv.conf`
    - Defines the system DNS servers and domains for name resolution
3. `/etc/nsswitch.conf`
    - Determines the order in which local name resolution vs. remote name resolution queries take place
        - For example: `hosts: files dns`
            - Would cause the system to check `/etc/hosts` for name resolution matches BEFORE checking nameservers defined in `/etc/resolv.conf`
4. **Note**: The following commands require the `bind-utils` package
5. `host`
    - Obtain DNS information about the indicated host (performs DNS queries)
6. `getent`
    - `hosts [name]`: Get information about the indicated host name (will read `/etc/nsswitch.conf` to determine the order to look)
7. `dig`
    - `[server] [domain] [record type]`: All optional except domain
        - `server`: Specify a DNS server to use
        - `domain`: The specific domain to query for
        - `record type`: Different record types (i.e., NAME, CNAME, MX, etc.)

Topic 110 - Security

110.1 Perform Security Administration Tasks

1. `root`
    - The system administration account
    - Security best practice not to log into a system directly as root, but to "become" root
    - Two methods:
        1. `su`: You will provide the root password and will then become root (su = superuser)
        2. `sudo su`: You will provide your password and, as long as you have sudo privileges (sudo = superuser do), you will then become root
2. `su`
    - Super user
        - For example: `su root`
            - Become the root user, but login shell scripts (`.bash_login`, `.bashrc`, etc.) will not execute so you will not inherit values in them
        - For example: `su - root`
            - Become root and all login shell scripts will execute and the full environment and settings will be available
            - Can also be accomplished with `su -l root`
    - Applies to regular users as well (as long as you know the associated password)
        - **Note**: Root can `su` to any user without knowing the user's password
3. `visudo`
    - Special editing mode for VI that will allow editing and syntax/error checking of the `/etc/sudoers` file
        - **Note**: The editor used can be changed by setting the `EDITOR` environment variable to any other available text editor
4. `/etc/sudoers`
    - File that lists users that can execute commands with elevated root level privileges
    - Format requires the user be listed (or a group if an entire group should have root privileges) along with what they can do
        - For example: `user ALL=(ALL) ALL`
            - Would provide the `user` account sudo rights for any command using elevated privileges
        - For example: `user ALL=(ALL) /bin/systemctl`
            - Would limit the `user` to only be able to run `systemctl` with sudo privileges (so they can restart services)
5. `find`

- In addition to general commands to find files, directories, etc., can be used to find files with certain kinds of permissions or owned by certain users
- Used to recover from inadvertent (or malicious) changing of permissions
- `-user [user]`: Find files owned by the indicated user
- `-perm [permissions]`: Find files that have the specified permissions
  - Can include "special bit" permissions
  - `+`: Find anything matching the first number
  - `-`: Find the exact match for the first number
- `-type [type]`: Find files, links, devices, directories
- `-exec [commands] {} \;`: Execute the indicated command(s) on the files found
  - For example: `find / -user root -perm 0777 -type f -exec ls -al {} \; | mail -s "Files owned by root with world rwx permissions" root`
    - This would find all files owned by root with world read/write/execute permissions and then email that list with all characteristics to the root user with the indicated subject

6. Other items that can be used to investigate/secure the system (covered in LPIC-1 Exam 1 or elsewhere in this course)

   - `passwd`, `fuser`, `lsof`, `chage`, `usermod`, `ulimit`, `w`, `/etc/inittab`

110.2 Setup Host Security

1. Service management

   - Determining how/when a service is running (always on, running on demand, or disabled)

2. **Note**: Neither of these are generally installed by default on modern distributions

3. `inetd`

   - Legacy daemon/service for providing system services "on demand" or as needed
   - `/etc/inetd.conf`: Primary configuration file containing a single line for each control
     - For example: `telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd`
       - Sample configuration for telnet, the following order of fields:
         - Service name: The name of the service to control (`telnet`)
         - Socket type: Values can be `stream`, `dgram`, `rdm`, `seqpacket` (`stream`)
         - Protocol: `/etc/protocols` file defines the values, generally either TCP or UDP (`tcp`)
         - Wait/nowait: Wait for single threaded services, nowait for multi-threaded services (`nowait`)
         - User/group: The group or user account the service will run as (`root`)
         - Path to command: Full path to the application/service to run (`/usr/sbin/in.telnetd`)
         - Arguments: Anything that may be required to complete the service configuration (`in.telnetd`)

4. `xinetd`
    - Replacement for `inetd` allowing more granular control of services
        - `/etc/xinetd.conf`: Primary configuration file, including files in `/etc/xinetd.d` with files (one each) per service that is controlled
            - The request will come to the daemon, which will check for the service type and port and then scan for the appropriate service configuration file in `/etc/xinit.d`
            - Key fields:
                - `cps = [#] [#]`: First number limits the connections per second to the service, second number is the delay before more connections are answered
                - `instances = [#]`: The total number of daemons allowed at any time
5. Typical services
    - `finger`, `imap`, `rsh`/`rsync`, `telnet`
    - Sample `rsync` file from `/etc/xinet.d`

```
service telnet
{
  flags     = REUSE
  socket_type   = stream
  wait      = no
  user      = root
  server        = /usr/sbin/in.telnetd
  log_on_failure    += USERID
  disable       = no
}
```

    - **Note**: Common exam topic is how to run a service from `xinetd` as another user other than root (**hint**: see the `user` field in each service configuration file)
        - `service`: Name of the service
        - `disable`: `yes` (disable and do not accept connections) and `no` (do not disable and allow connections)
        - `flags`: Varies by service
        - `socket type`: Values can be `stream`, `dgram`, `rdm`, `seqpacket`
        - `wait`/`nowait`: `wait` for single threaded services, `nowait` for multi-threaded services
        - `user`/`group`: The group or user account the service will run as
        - `server_args`: Arguments to send to the service
        - `log_on_failure`: Log failures including the `USERID`
    - Enabling, set `disable = no` and restart `xinetd`
6. TCP wrappers
    - For `inetd`, using only `/etc/hosts.allow` and `/etc/hosts.deny` as parameters of `tcpd`
    - `xinetd`, the library `libwrap.a` allows those services to use `/etc/hosts.allow` and `/etc/hosts.deny`
    - For example (format of each):
      `[service/daemon]: [host(s)]: [option]: [option]`
      `in.telnetd: 10.1.10.0/24`

        - Would allow (or deny) users of the 10.1.10.0/24 network access to the `telnet` service (depending on the file it exists in)
    - Methods to use in defining policies:
        - Deny by default: Denies ALL hosts access to ALL services (`ALL:ALL`)
            - Used as a fallback in `hosts.deny` since matches in `hosts.allow` will override
        - Allow by default: Implicitly trust everyone and provide access to everything
            - Security risk for obvious reasons
        - Mix: Allow and deny selectively

7. Order of precedence for `hosts.allow` and `hosts.deny`

    - `hosts.allow`: Read first, matches are allow and `hosts.deny` skipped (entirely)
        - Changes to either `hosts.allow/deny` take immediate effect on save
    - Sequential read: Multiple entries for the same service will cause only the first match to apply
    - If files do not exist, no rules apply
    - Different options can be added rather than just denying service
        - For example: `in.telnetd: 10.1.10.0/24 : twist /bin/echo "Service 404 - Service Not Found"`

110.3 Securing Data with Encryption (Using GnuPG to Encrypt and Decrypt Files)

1. `gnupg`

    - GnuPG Privacy Guard
    - Public and private key creation for encrypting data

2. `gpg`

    - Utility working with keys
    - `--gen-key`
        - Will prompt you for key type (RSA would be the general default)
        - Follow prompts to provide the data in creation of your keypairs
    - Entropy: the "seed" that is used to generate encryption keys, it has to do with the randomness of certain events on your system (generating it correctly can take a long time on large keys)
        - Generate entropy (for testing only)
            - `yum install rngd-tools`
                - `rngd -r /dev/urandom`
    - `-a`: Generate a text public key to provide
    - `-o [filename]`: The output file

        - **Note**: When generating keys, be sure you have logged into the system directly with the username and password or the environment or GPG will not be setup without manual configuration
        - **Note**: Once complete, the key name will be listed on the output, **NOTE** that value
    - `--import [key]`: Import the indicated key (public key from user who will be sending you a file)
    - `--list-keys`: List all keys you have imported
3. Encrypt a file to send to someone
    - Import the public key file from the intended person
    - Decrypt the file using the imported key and the required passphrase
        - For example: `gpg file.gpg`
            - Will decrypt the file provided the right key has been imported AND you have the passphrase it was created with

110.3 Securing Data with Encryption (SSH Keys - Public and Private - for Secure Connections)

1. `ssh`
    - Secure shell
        - Related commands (also secure): `scp`, `ssh-agent`, `ssh-add`
    - `-l [user] [host]`: Logs in as the specified user to the host
    - `[user]@[host]`: Logs in as the specified user to the host
    - `-X`: Enable SSH X Window System forwarding
    - `-x`: Disable SSH X Window System forwarding
2. `/etc/ssh/sshd_config`: Main configuration for `sshd`
3. `/etc/ssh/ssh_host_[rsa/dsa]_key`: Specific encryption private keys with permissions of 600 to restrict access to root user
4. `/etc/ssh/ssh_host_[rsa/dsa]_key.pub`: Specific encryption public keys with permissions of 644 to restrict access to root user and read for other users
5. `/etc/ssh/known_hosts`
    - File used to check public keys of known/trusted hosts (does not exist by default)
6. `~/.ssh/known_hosts`
    - File containing the public key of known/trusted hosts that have connected to/from existing host by the user whose directory it exists in
7. `~/.ssh/authorized_keys`
    - Stores public keys for logging in as the user that owns the directory

8. `ssh-keygen`
    - Creates a public/private key pair for use with SSH
    - `-b [#]`: Encryption key size (i.e., 1024, 2048, etc.)
    - `-t [type]`: Encryption key type (DSA or RSA - RSA is more secure and is currently the default)
        - Will prompt for a password - blank will allow you to use the key to login completely without password whereas entering a passphrase effectively creates two factor authentication (key + passphrase)
        - File permissions on keys should be either 644 (older) or 600 (newer)
9. `ssh-copy-id`
    - Copies your public key to the user and host as indicated
        - For example: `ssh-copy-id user@user.mylabserver.com`
            - After the password is entered for the indicated user, will copy the public key from this host and user to the remote host and user's `authorized_keys` file and you can thereafter login on that system and account with this key
    - Manual method: Copy/paste the contents of your public key into the remote user's `authorized_keys` file and set the permissions at 600
    - Next connection will either work (no passphrase setup) or prompt just for passphrase
10. `ssh-agent`
    - Wrapper for SSH that allows you to pass items (keys) into the SSH shell for connectivity
    - Executing `ssh-agent` starts the agent on the indicated shell
        - For example: `ssh-agent bash`
            - Starts a bash prompt with the agent wrapping it
11. `ssh-add`
    - Will prompt you for your passphrase on your public key (if set)
    - Once entered, subsequent uses will not require entry until exited