Managing Software and Processes

Objectives:

- ✓ 102.3 Manage shared libraries
- ✓ 102.4 Use Debian package management
- ✓ 102.5 Use RPM and YUM package management
- ✓ 103.5 Create, monitor, and kill processes
- ✓ 103.6 Modify process execution priorities

LOOKING AT PACKAGE CONCEPTS

Most Linux users want to download an application and use it. Thus, Linux distributions have created a system for bundling already compiled applications for distribution. This bundle is called a package, and it consists of most of the files required to run a single application. You can then install, remove, and manage the entire application as a single package rather than as a group of disjointed files.

Package Management

- Tracking software packages on a Linux system is called **package management**.
- Linux implements package management by using a database to track the installed packages on the system.
- The package management database keeps track of not only what packages are installed but also the exact files and file locations required for each application.
- Different Linux distributions have created different package management systems.
- Two of these systems have risen to the top and become standards:
  - ✓ Red Hat package management (RPM)
  - ✓ Debian package management (Apt)

Package Management

❖ Each package management system uses a different method of tracking application packages and files, but they both track similar information:

✓ Application files

✓ Library dependencies

✓ Application version

USING RPM

Developed at Red Hat, the RPM Package Manager (RPM) utility lets you install, modify, and remove software packages. It also eases the process of updating software.

RPM Distributions and Conventions

- Used in the Red Hat Linux distribution, Fedora and CentOS, etc.
- There are other distributions that are not Red Hat based, such as openSUSE and OpenMandriva Lx, that employ RPM as well.
- RPM package files have an `.rpm` file extension and follow this naming format:
  - ✓ `PACKAGE-NAME-VERSION-RELEASE.ARCHITECTURE.rpm`

RPM Package File Conventions

PACKAGE-NAME

- ✓ The PACKAGE-NAME is the name of the software package.
- ✓ Different distributions may have different PACKAGE-NAMEs for the same program and that software package names may differ from program names.

VERSION

- ✓ The VERSION is the program’s version number and represents software modifications that are more recent than older version numbers.
- ✓ Traditionally a package’s version number is formatted as two to three numbers and/or letters separated by dots (.).
  - 1.13.1 and 7.4p1.

RPM Package File Conventions

❖ RELEASE

- ✓ The RELEASE is also called the build number.
- ✓ It represents a smaller program modification than does the version number.
- ✓ In addition, due to the rise of continuous software delivery models, you often find version control system (VCS) numbers listed in the release number after a dot.
  - 22 and 94.gitb2f74b2.
  - Some distros include the distribution version in the build number.
    - el7 (Red Hat Enterprise Linux v7) or fc29 (Fedora, formerly called Fedora Core, v29) after a dot.

RPM Package File Conventions

❖ ARCHITECTURE

- ✓ This is a designation of the CPU architecture for which the software package was optimized.
- ✓ Typically you'll see **x86_64** listed for 64-bit processors.
- ✓ Sometimes **noarch** is used, which indicates the package is architecturally neutral.
- ✓ Older CPU architecture designations include **i386** (x86), **ppc** (PowerPC), and **i586** and **i686** (Pentium).

RPM Distributions and Conventions

```
docker-1.13.1-94.gitb2f74b2.el7.centos.x86_64.rpm

emacs-24.3-22.el7.x86_64.rpm

openssh-7.4p1-16.el7.x86_64.rpm

zsh-5.0.2-31.el7.x86_64.rpm
```

The rpm Command Set

❖ **rpm** - RPM Package Manager

```
rpm ACTION [OPTION] PACKAGE-FILE
```

✓ `-e, --erase`	Removes the specified package
✓ `-F, --freshen`	Upgrades a package only if an earlier version already exists
✓ `-i,--install`	Installs the specified package
✓ `-q, --query`	Queries whether the specified package is installed
✓ `-U, --upgrade`	Installs or upgrades the specified package
✓ `-V, --verify`	Verifies whether the package files are present and the package's integrity
✓ `-vh`	Shows the progress of an update and what it's doing.

Querying RPM Packages

- Use the -q action to perform a simple query on the package management database for installed packages.
- You can add several options to the query action to obtain more detailed information.
  - ✓ `-c, --configfiles`
    - Lists the names and absolute directory references of package configuration files
  - ✓ `-i, --info`
    - Provides detailed information, including version, installation date, and signatures
  - ✓ `--provides`
    - Shows what facilities the package provides
  - ✓ `-R, --requires`
    - Displays various package requirements (dependencies)

Querying RPM Packages

- ✓ `-s, --state`
  - Provides states of the different files in a package, such as normal (installed), not installed, or replaced
- ✓ `--whatprovides`
  - Shows to what package a file belongs
- ✓ `-a, --all`
  - Query all installed packages.
- ✓ `-p, --package PACKAGE_FILE`
  - Query an (uninstalled) package PACKAGE_FILE.

The rpm Command Set

```
# yumdownloader zsh

# rpm -Uvh zsh-5.0.2-34.el7_7.2.x86_64.rpm

# rpm -q zsh

# rpm -q docker

# rpm -qi zsh

# rpm -qR zsh

# rpm -qc zsh

# rpm –qa

# rpm -qRp zsh-5.0.2-34.el7_7.2.x86_64.rpm

# rpm -q --whatprovides /usr/bin/zs
```

Verifying RPM Packages

- Keeping a watchful eye on your system’s packages is an important security measure.
- If you receive nothing or a single dot (.) from the rpm -V command, that’s a good thing.

Verifying RPM Packages

- Potential integrity response codes and what they mean:

✓ **?**	Unable to perform verification tests
✓ **5**	Digest number has changed
✓ **c**	File is a configuration file for the package
✓ **D**	Device number (major or minor) has changed
✓ **G**	Group ownership has changed
✓ **L**	Link path has changed
✓ **missing**	Missing file
✓ **M**	Mode (permission or file type) has changed
✓ **P**	Capabilities have changed
✓ **S**	Size of file has changed
✓ **T**	Time stamp (modification) has changed
✓ **U**	User ownership has changed

The rpm Command Set

```
# rpm -V zsh

# rpm -e zsh

# rpm -q zsh

package zsh is not installed
```

Extracting Data from RPMs

- **rpm2cpio** - Extract cpio archive from RPM Package Manager (RPM) package.

```
rpm2cpio [filename]
```

Extracting Data from RPMs

- **cpio** - copy files to and from archives

```
cpio [OPTION...] [< archive]
```

- ✓ **-d, --make-directories**
  - Create leading directories where needed.
- ✓ **-i, --extract'**
  - Run in copy-in mode. see “Copy-in mode”.
- ✓ **-v, --verbose'**
  - List the files processed

Extracting Data from RPMs

1. Extract files from an RPM package file

```
$ rpm2cpio zsh-5.0.2-34.el7_7.2.x86_64.rpm > zsh.cpio
```

2. Move the files from the cpio archive into directories

```
$ cpio –idv < zsh.cpio
```

What is Repository

- The `rpm` commands are useful tools, but they have limitations.
  - ✓ If you're looking for new software packages to install, it's up to you to find them.
  - ✓ Also, if a package depends on other packages to be installed, it's up to you to install those packages first, and in the correct order.
- To solve that problem, each Linux distribution has its own central clearinghouse of packages, called a repository.
- The repository contains software packages that have been tested and known to install and work correctly in the distribution environment.

Using YUM

- Most Linux distributions create and maintain their own repositories of packages.
- There are also additional tools for working with package repositories.
  - ✓ These tools can interface directly with the package repository to find new software and even automatically find and install any dependent packages the application requires to operate.
- The core tool used for working with Red Hat repositories is the YUM utility
  - ✓ short for **YellowDog Update Manager**, originally developed for the YellowDog Linux distribution.
- Its yum command allows you to query, install, and remove software packages on your system directly from an official Red Hat repository.

The yum Command Set

❖ **yum** - Yellowdog Updater Modified

```
yum [options] [command] [package ...]
```

✓ The yum command uses the /etc/yum.repos.d/ directory to hold files that list the different repositories it checks for packages.

✓ `check-update`	Checks the repository for updates to installed packages
✓ `clean`	Removes temporary files downloaded during installs
✓ `deplist`	Displays dependencies for the specified package
✓ `groupinstall`	Installs the specified package group
✓ `info`	Displays information about the specified package
✓ `install`	Installs the specified package
✓ `list`	Displays information about installed packages

The yum Command Set

✓ `localinstall`	Installs a package from a specified RPM file
✓ `localupdate`	Updates the system from specified RPM files
✓ `provides`	Shows to what package a file belongs
✓ `reinstall`	Reinstalls the specified package
✓ `remove`	Removes a package from the system
✓ `resolvedep`	Displays packages matching the specified dependency
✓ `search`	Searches repository package names and descriptions for specified keyword
✓ `shell`	Enters yum command-line mode
✓ `update`	Updates the specified package(s) to the latest version in the repository
✓ `upgrade`	Updates specified package(s) but removes obsolete packages

The yum Command Set

```
# yum install emacs

# yum grouplist

# yum reinstall emacs

# yum remove emacs
```

Using ZYpp

- **zypper** - Command-line interface to ZYpp system management library (libzypp)

```
zypper [--global-opts] command [--command-opts] [command-arguments]
```

- ✓ The zypper utility allows you to shorten some of its commands.
  - You can shorten **install** to **in**, **remove** to **re**, and **search** to **se**
- ✓ `help`: Displays overall general help information or help on a specified command
- ✓ `install`: Installs the specified package
- ✓ `info`: Displays information about the specified package
- ✓ `list-updates`: Displays all available package updates for installed packages from the repository

Using ZYpp

- ✓ `lr`: Displays repository information
- ✓ `packages`: Lists all available packages or lists available packages from a specified repository
- ✓ `what-provides`: Shows to what package a file belongs
- ✓ `refresh`: Refreshes a repository's information
- ✓ `remove`: Removes a package from the system
- ✓ `search`: Searches for the specified package(s)
- ✓ `update`: Updates the specified package(s) or if no package is specified, updates all currently installed packages to the latest version(s) in the repository
- ✓ `verify`: Verifies that installed packages have their needed dependencies satisfied

```
$ sudo zypper install emacs

$ zypper info emacs

$ zypper what-provides /usr/bin/emacs

$ sudo zypper remove emacs


$ zypper se nmap
```

USING DEBIAN PACKAGES

The Debian package management system is mostly used on Debian-based Linux distros, such as Ubuntu. With this system you can install, modify, upgrade, and remove software packages.

Debian Package File Conventions

- Debian bundles application files into a single **.deb** package file for distribution that uses the following filename format:
  - ✓ `PACKAGE-NAME-VERSION-RELEASE_ARCHITECTURE.deb`
- In the ARCHITECTURE, you typically find **amd64**, denoting it was optimized for the AMD64/Intel64 CPU architecture.
- Sometimes **all** is used, indicating the package is architecturally neutral.

Debian Package File Conventions

```
docker_1.5-1build1_amd64.deb

emacs_47.0_all.deb

openssh-client_1%3a7.6p1-4ubuntu0.3_amd64.deb

vim_2%3a8.0.1453-1ubuntu1_amd64.deb

zsh_5.4.2-3ubuntu3.1_amd64.deb
```

```
$ sudo apt-get download vim
```

The dpkg Command Set

- **dpkg** - package manager for Debian

```
dpkg [option...] action PACKAGE-FILE
```

- ✓ `-c, --contents`
  - Displays the contents of a package file
- ✓ `--configure`
  - Reconfigures an installed package
- ✓ `--get-selections`
  - Displays currently installed packages
- ✓ `-i, --install`
  - Installs the package; if package is already installed, upgrad
- ✓ `-I, --info`
  - Displays information about an uninstalled package file
- ✓ `-l, --list`
  - Lists all installed packages matching a specified pattern
- ✓ `-L, --listfiles`
  - Lists the installed files associated with a package
- ✓ `-p, --print-avail`
  - Displays information about an installed package
- ✓ `-r, --remove`
  - Removes an installed package but leaves the configuration files
- ✓ `-P, --purge`
  - Removes an installed package, including configuration files
- ✓ `-s, --status`
  - Displays the status of the specified package
- ✓ `-S, --search`
  - Locates the package that owns the specified files

The dpkg Command Set

```
$ sudo apt-get download zsh

$ dpkg -l

$ dpkg -I zsh_5.4.2-3ubuntu3.1_amd64.deb

$ dpkg --content zsh_5.4.2-3ubuntu3.1_amd64.deb | less

$ sudo dpkg -i zsh_5.4.2-3ubuntu3.1_amd64.deb

$ dpkg -s zsh

$ sudo dpkg -P zsh ≠ $ sudo dpkg -p zsh
```

Looking at the APT Suite

- The Advanced Package Tool (APT) suite is used for working with Debian repositories.
- This includes
  - ✓The **apt-cache** program that provides information about the package database
  - ✓The **apt-get** program that does the work of installing, updating, and removing packages.
- The APT suite of tools relies on the **/etc/apt/sources.list** file to identify the locations of where to look for repositories.

Using apt-cache

- **apt-cache** - query the APT cache

  ```
  apt-cahce [OPTION]
  ```

  - ✓ **depends**
    - Displays the dependencies required for the package
  - ✓ **pkgnames**
    - Prints the name of each package APT knows
  - ✓ **search**
    - Displays the name of packages matching the specified item
  - ✓ **showpkg**
    - Lists information about the specified package
  - ✓ **stats**
    - Displays package statistics for the system
  - ✓ **unmet**
    - Shows any unmet dependencies for all installed packages or the specified installed package

Using apt-cache

```
$ apt-cache pkgnames | grep zsh

$ apt-cache search zsh

$ apt-cache showpkg zsh
```

Using apt-get

- **apt-get** - APT package handling utility -- command-line interface

```
apt-get [OPTION]
```

- ✓ **autoclean**
  - Removes information about packages that are no longer in the repository
- ✓ **check**
  - Checks the package management database for inconsistencies
- ✓ **clean**
  - Cleans up the database and any temporary download files
- ✓ **dist-upgrade**
  - Upgrades all packages, but monitors for package dependencies

Using apt-get

- ✓ **dselect-upgrade**
  - Completes any package changes left undone
- ✓ **install**
  - Installs or updates a package and updates the package management database
- ✓ **remove**
  - Removes a package from the package management database
- ✓ **source**
  - Retrieves the source code package for the specified package
- ✓ **update**
  - Retrieves updated information about packages in the repository
- ✓ **upgrade**
  - Upgrades all installed packages to newest versions

Using apt-get

- Installing a package

```
$ sudo apt-get install zsh
```

- Upgrading a package

```
$ sudo apt-get install emacs
```

Reconfiguring Packages

- **dpkg-reconfigure** - reconfigure an already installed package

```
dpkg-reconfigure [options] packages
```

✓ This command will throw you into a text-based menu screen that will lead you through a series of simple configuration questions.

✓ Using following command to see what package is supported:

```
$ ls /var/lib/dpkg/info/*.config
```

Reconfiguring Packages

- **debconf-show** - query the debconf database
  - ✓`debconf-show packagename`
  - ✓This tool allows you to view the package’s configuration.

```
$ sudo dpkg-reconfigure cups

$ sudo debconf-show cups
```

MANAGING SHARED LIBRARIES

In managing your system’s applications, you need to understand libraries and, more specifically, shared libraries.

Library Principles

- A system library is a collection of items, such as program functions.
  - ✓ Functions are self-contained code modules that perform a specific task within an application, such as opening and reading a data file.
  - ✓ The benefit of splitting functions into separate library files is that multiple applications that use the same functions can share the same library files.
- These files full of functions make it easier to distribute applications, but also make it more complicated to keep track of what library files are installed with which applications.

Library Principles

- Linux supports two different flavors of libraries.
  - ✓ **Static libraries** (also called **statically linked libraries**) that are copied into an application when it is compiled.
  - ✓ **Shared libraries** (also called **dynamic libraries**) where the library functions are copied into memory and bound to the application when the program is launched.
    - This is called loading a library.
  - ✓ A shared library file employs the following filename format:

```
libLIBRARYNAME.so.VERSION
```

Locating Library Files

❖ When a program is using a shared function, the system will search for the function's library file in a specific order; looking in directories stored within the

1. `LD_LIBRARY_PATH` environment variable
2. Program's `PATH` environment variable
3. `/etc/ld.so.conf.d/` folder
4. `/etc/ld.so.conf` file
5. `/lib*/` and `/usr/lib*/` folders

Loading Dynamically

- When a program is started, the dynamic linker (also called the dynamic linker/loader) is responsible for finding the program's needed library functions.
- After they are located, the dynamic linker will copy them into memory and bind them to the program.
- Historically, the dynamic linker executable has a name like `ld.so` and `ld-linux.so*`.
- You can employ the `locate` utility to find its actual name and location

```
$ locate ld-linux
$ /usr/lib64/ld-linux-x86-64.so.2 /usr/bin/echo "Hello World"
```

Library Management Commands - Managing the Library Cache

- The **library cache** is a catalog of library directories and all the various libraries contained within them.
- The system reads this cache file to quickly find needed libraries when it is loading programs.
- When new libraries or library directories are added to the system, this library cache file must be updated.

Managing the Library Cache

- **ldconfig** - configure dynamic linker run-time bindings
  - ✓ **-v, --verbose**
    - Verbose mode. Print current version number, the name of each directory as it is scanned, and any links that are created. Overrides quiet mode.

```
$ ldconfig -v 2> /dev/null | grep libmysqlclient
libmysqlclient.so.18 -> libmysqlclient.so.18.0.0
```

Troubleshooting Shared Library Dependencies

- **ldd** - print shared object dependencies

```
ldd [option]... file...
```

✓It displays a list of the library files required for the specified application.

```
$ ldd /usr/bin/echo
linux-vdso.so.1 => (0x00007ffd3bd64000)
libc.so.6 => /lib64/libc.so.6 (0x00007f7c39eff000)
/lib64/ld-linux-x86-64.so.2 (0x00007f7c3a2cc000)
```

MANAGING PROCESSES

Linux must keep track of lots of different programs, all running at the same time. This section covers how Linux keeps track of all the active applications, how you can peek at that information, as well as how to use command-line tools to manage the running programs.

Examining Process Lists

- At any given time lots of active programs are running on the Linux system.
- Linux calls each running program a process.
- The Linux system assigns each process a process ID (PID) and manages how the process uses memory and CPU time based on that PID.
- When a Linux system first boots, it starts a special process called the `init` process.
- The `init` process is the core of the Linux system;
  - ✓ it runs scripts that start all of the other processes running on the system,

Viewing Processes with ps

- **ps** - report a snapshot of the current processes.

```
ps [options]
```

- ✓ By default, the ps program shows only the processes that are running in the current user shell
- ✓ **a** Display every process on the system associated with a tty terminal
- ✓ **-A, -e** Display every process on the system
- ✓ **-C CommandList** Only display processes running a command in the CommandList
- ✓ **-g GIDList, -group GIDList** Only display processes whose current effective group is in GIDList
- ✓ **-G GIDList, -Group GIDList** Only display processes whose current real group is in GIDList
- ✓ **-N** Display every process except selected processes

Viewing Processes with ps

- ✓ **p PIDList, -p PIDList, --pid PIDList** Only display PIDList processes
- ✓ **-r** Only display selected processes that are in a state of running
- ✓ **-t ttyList, --tty ttyList** List every process associated with the ttyList terminals
- ✓ **-T** List every process associated with the current tty terminal
- ✓ **-u UserList, --user UserList** Only display processes whose effective user (username or UID) is in UserList
- ✓ **-U UserList, --User UserList** Only display processes whose real user (username or UID) is in UserList
- ✓ **x** Remove restriction of "associated with a tty terminal"; typically used with the a option

Understanding Process States

- **Sleeping: Processes that are swapped into virtual memory.**
  - ✓ Often the Linux kernel places a process into sleep mode while the process is waiting for an event.
  - ✓ When the event triggers, the kernel sends the process a signal.
  - ✓ If the process is in **interruptible sleep mode**, it will receive the signal immediately and wake up.
  - ✓ If the process is in **uninterruptible sleep mode**, it only wakes up based on an external event, such as hardware becoming available.
    - It will save any other signals sent while it was sleeping and act on them once it wakes up.
- **Zombie: If a process has ended but its parent process hasn't acknowledged the termination signal because it's sleeping.**

Selecting Processes with ps

- To see every process on the system using standard syntax:

```
$ ps -e
$ ps -ef
```

- To see every process on the system using BSD syntax:

```
$ ps ax
$ ps aux
```

- To print a process tree:

```
$ ps -ejH
$ ps axjf
```

- To see every process running as root (real & effective ID) in user format:

```
$ ps -U root -u root u
```

Viewing Processes with top

- **top** - display Linux processes
  - ✓ `1` Toggles the single CPU and Symmetric Multiprocessor (SMP) state
  - ✓ `t` Toggles display of the CPU information line
  - ✓ `m` Toggles display of the MEM and SWAP information lines
  - ✓ `f` Adds or removes different information columns
  - ✓ `F or O` Selects a field on which to sort the processes (%CPU by default)
  - ✓ `h` Toggles showing of threads
  - ✓ `z` Toggles color and mono mode
  - ✓ `k` Kills a specific process (only if process owner or if root user)
  - ✓ `d` or s Changes the update interval (default three seconds)
  - ✓ `q` Exits the top command

Managing process

```
$ uptime
11:19:43 up 1:01, 3 users, load average: 1.03, 0.69, 0.37
$ free -h
total used free shared buff/cache available
Mem: 3.9G 1.0G 2.2G 30M 710M 2.6G
Swap: 472M 0B 472M
$ watch uptime
```

Employing Multiple Screens

- if you are limited to using terminals in a nongraphical environment, you can still open window sessions side-by-side to perform multiple operations and monitor their displays.
- This is accomplished through a terminal multiplexer.
  - ✓ `screen`
  - ✓ `tmux`

Multiplexing with screen

- **screen** - screen manager with VT100/ANSI terminal emulation

✓**-ls**	Print screen window information
✓**-r screen-id**	To reattach to the screen
✓**Ctrl+A**	Issue a command within the window

- **Focuses:** Split a window screen up into multiple windows

Multiplexing with screen – Ctrl+A+

\	Kill all of a processes' windows and terminate screen
Shift+|	Split current screen window vertically into two focuses
Tab	Jump to next window focus
D	Detach from current screen window
K	Kill current window
N	Move to next screen window
P	Move to previous screen window
C	Create a window
Shift+S	Split current screen window horizontally into two focuses

Screen lab

1. Type screen to create the first window screen, then Press the Enter key to exit the Welcome screen, if one is shown.
2. Issue your desired monitoring command, such as top.
3. Press the Ctrl+A prefix and then the Shift+S key combinations to split the window into two regions (focuses).
4. Press the Ctrl+A prefix and then the Tab key to jump to the bottom focus.
5. Press the Ctrl+A prefix and then the C key to create a window within the bottom focus.
6. Press the Ctrl+A prefix and then the | key to split the current window vertically. Now the focus is in the lower-left window.
7. Press the Ctrl+A prefix and then the Tab key to jump to the lower-right focus.
8. Press the Ctrl+A prefix and then the C key to create a window within the lower-right focus.
9. Press the Ctrl+A key combination and then the \ key. Type Y and press Enter to enact the command

Multiplexing with tmux

- **tmux** – terminal multiplexer

```
tmux new
```

- ✓ **ls**
  - Displaying a detached window
- ✓ **attach-session -t session-ID**
  - Reattach to a particular detached window session
- ✓ **Ctrl+B**
  - Issue a command within the window

tmux lab

&	Kill the current window
%	Split current screen window vertically into two panes
"	Split current screen window horizontally into two panes
D	Detach from current window
L	Move to previous window
N	Move to next window
O	Move to next pane
Ctrl+O	Rotate panes forward in current window

Understanding Foreground and Background Processes

- Some programs can take a long time to run, and you may not want to tie up the commandline interface.
- Fortunately, there’s a simple solution to that problem:
  - ✓run the program in background mode.

Sending a Job to the Background

- Place an ampersand symbol (&) after the command.
- A great program to use for background mode demonstration purposes is the `sleep` command.
- This utility is useful for adding pauses in shell scripts.
- You simply add an argument indicating the number of seconds you wish the script to freeze.
- Thus, sleep 3 would pause for three seconds.

Display Background process with jobs

❖ **jobs** - Display status of jobs in background

```
jobs [OPTION…]
```

✓ `-l` lists process IDs in addition to the normal information

Sending a Job to the Background

```
$ sleep 3000 &
[1] 1539
$ jobs
[1]+ Running sleep 3000 &
$ jobs -l
[1]+ 1539 Running sleep 3000 &
```

Sending a Running Program to the Background

- **bg** - Move jobs to the background.

```
bg [job_spec ...]
```

✓ First use **Ctrl+Z** to pause the process.

```
$ bash CriticalBackups.sh
^Z
[2]+ Stopped bash CriticalBackups.sh
$ bg %2
[2]+ bash CriticalBackups.sh &
$ jobs -l
[1]- 1539 Running sleep 3000 &
[2]+ 1540 Running bash CriticalBackups.sh &
```

Bringing Jobs to the Foreground

- **fg** - Move job to the foreground

```
fg [job_spec]
```

✓Using % before job_number

```
$ jobs -l
[1]- 1539 Running sleep 3000 &
[2]+ 1540 Running bash CriticalBackups.sh &
$ fg %2
bash CriticalBackups.sh
```

Stopping a Job

- **kill** - send a signal to a process

```
kill [options] <pid> [...]


$ jobs -l
[1]- 1539 Running sleep 3000 &
[2]+ 1540 Running bash CriticalBackups.sh &
$ kill %1
[1]- Terminated sleep 3000
$ jobs -l
[2]+ 1540 Running bash CriticalBackups.sh &
```

Keeping a Job Running after Logout

- **nohup - Run a command immune to hangups, with output to a non-tty**

```
nohup COMMAND [ARG]...
```

- ✓ nohup command will force the application to ignore any input from STDIN
- ✓ By default STDOUT and STDERR are redirected to the `$HOME/nohup.out` file.

```
$ nohup bash CriticalBackups.sh &
[1] 2090
$ nohup: ignoring input and appending output to 'nohup.out'
```

Managing Process Priorities

❖ **nice** - run a program with modified scheduling priority

```
nice [OPTION] [COMMAND [ARG]...]
```

✓ The `nice` and `renice` commands allow you to set and change a program's **niceness level**, which in turn modifies the priority level assigned by the system to an application.

✓ -n VALUE

- The VALUE parameter is a numeric value from –20 to 19.
- The lower the number, the higher priority the process receives.
- The default niceness level is zero (without using nice).

Managing Process Priorities

❖ **renice** - alter priority of running processes

```
renice [-n] priority [-g|-p|-u] identifier...
```

✓ The **renice** command allows you to change the priority of multiple processes based on a list of PID values, all of the processes started by one or more users, or all of the processes started by one or more groups.

✓ Only if you have super user privileges can you set a nice value less than 0 (increase the priority) of a running process

Managing Process Priorities

```
$ nice -n 10 bash CriticalBackups.sh

$ ps -l 1949

$ renice 15 -p 1949

1949 (process ID) old priority 10, new priority 15

$ sudo renice -n -5 -p 1949

1949 (process ID) old priority 15, new priority -5

$ sudo renice -10 -p 1949

1949 (process ID) old priority -5, new priority -10
```

Sending Signals to Processes

- Sometimes a process gets hung up and just needs a gentle nudge to either get going again or stop.
- Other times, a process runs away with the CPU and refuses to give it up.
- In both cases, you need a command that will allow you to control a process.
- To do that, Linux follows the Unix method of interprocess communication.
- In Linux, processes communicate with each other using process signals.
- A process signal is a predefined message that processes recognize and may choose to ignore or act on.

Linux process signals

Number	Name	Description
1	HUP	Hangs up
2	INT	Interrupts
3	QUIT	Stops running
9	KILL	Unconditionally terminates
11	SEGV	Segments violation
15	TERM	Terminates if possible
17	STOP	Stops unconditionally, but doesn’t terminate
18	TSTP	Stops or pauses, but continues to run in background
19	CONT	Resumes execution after STOP or TSTP

Sending Signals with the kill Command

- **kill** - send a signal to a process

```
kill [options] <pid> [...]
```

- ✓By default, the kill command sends a TERM signal to all the PIDs listed on the command line.
- ✓To send a process signal, you must either be the owner of the process or have super user privileges.
- ✓**-s <signal>, -<signal>, --signal <signal>**
  - Specify the signal to be sent. The signal can be specified by using name or number.

Sending Signals with the kill Command

- The generally accepted procedure is to first try the TERM signal.
- If the process ignores that, try the INT or HUP signal.
  - ✓ If the program recognizes these signals, it will try to gracefully stop doing what it was doing before shutting down.
- The most forceful signal is the KILL signal.
  - ✓ When a process receives this signal, it immediately stops running.
  - ✓ Use this as a last resort, as it can lead to corrupted files.

Sending Signals with the kill Command

```
$ ps 2285
PID TTY STAT TIME COMMAND
2285 pts/0 S 0:00 bash SecurityAudit.sh
$ kill 2285
[1]+ Terminated bash SecurityAudit.sh
$ ps 2285
PID TTY STAT TIME COMMAND
```

```
$ ps 2305
PID TTY STAT TIME COMMAND
2305 pts/0 T 0:00 vi
$ kill 2305
$ ps 2305
PID TTY STAT TIME COMMAND
2305 pts/0 T 0:00 vi
$ kill -s HUP 2305
$ ps 2305
PID TTY STAT TIME COMMAND
2305 pts/0 T 0:00 vi
$ kill -9 2305
[1]+ Killed vi
$ ps 2305
PID TTY STAT TIME COMMAND
```

Sending Signals with the killall Command

- **killall** - kill processes by name

```
killall [options] name [...]
```

- ✓ **-I, --ignore-case**
  - Do case insensitive process name match.
- ✓ **-r, --regexp**
  - Interpret process name pattern as a POSIX extended regular expression.
- ✓ **-s, --signal, -SIGNAL**
  - Send this signal instead of SIGTERM.
- ✓ **-u, --user**
  - Kill only processes the specified user owns. Command names are optional.

Sending Signals with the killall Command

```
$ ps
PID TTY TIME CMD
1441 pts/0 00:00:00 bash
1504 pts/0 00:00:00 stressor.sh
1505 pts/0 00:00:00 stress-ng
1506 pts/0 00:00:05 stress-ng-matri
1507 pts/0 00:00:00 stressor.sh
1508 pts/0 00:00:00 stress-ng
1509 pts/0 00:00:02 stress-ng-matri
1510 pts/0 00:00:00 stressor.sh
[…]
1517 pts/0 00:00:00 ps
```

```
$ killall stress-ng
[…]
$ ps
PID TTY TIME CMD
1441 pts/0 00:00:00 bash
1519 pts/0 00:00:00 ps
```

Sending Signals with the pkill Command

- **pgrep, pkill** - look up or signal processes based on name and other attributes

```
pgrep [options] pattern

pkill [options] pattern
```

- ✓ **-signal, --signal signal**
  - Defines the signal to send to each matched process
- ✓ **-g, --pgroup pgrp,...**
  - Only match processes in the process group IDs listed.
- ✓ **-t, --terminal term,...**
  - Only match processes whose controlling terminal is listed.
- ✓ **-P, --parent ppid,...**
  - Only match processes whose parent process ID is listed.

Sending Signals with the pkill Command

```
$ pgrep -t tty3
1716
1804
1828
1829
1831
1832
1836
1837
1838
1839
1840
$ ps 1840
PID TTY STAT TIME COMMAND
1840 tty3 R+ 0:39 stress-ng --class cpu -a 10 -b 5 -t 5m --matrix 0
```

```
$ sudo pkill -t tty3
$ pgrep -t tty3
1846
$ ps 1846
PID TTY STAT TIME COMMAND
1846 tty3 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear tty3 linux
```