Table of Contents

Part 2 of 2 - Explore all 232+ Linux commands at dargslan.com/learn/linux-commands
Each command includes syntax, options, practical examples with output, and pro tips.

$ pkill

Intermediate

Send signals to processes based on name and attributes

pkill sends signals to processes based on name patterns and other criteria. It is more flexible than killall because it matches against the full command line, not just the process name, and supports regex patterns.

pkill and pgrep are companion commands - pgrep finds processes, pkill kills them,...

Options & Flags

`-f`	Match against full command line, not just process name
`-u`	Match processes owned by user
`-9`	Send SIGKILL instead of default SIGTERM
`-x`	Require exact match of process name
`-t`	Match processes on terminal
`-P`	Match by parent PID
`-n`	Kill only the newest matching process
`-o`	Kill only the oldest matching process

Practical Examples

Example: Kill by process name

```
$ pkill nginx
```

Sends SIGTERM to all processes whose name matches nginx.

Example: Kill by full command line

```
$ pkill -f 'node app.js'
```

Matches against the entire command line, useful when process name alone is not unique.

Example: Kill user sessions

```
$ pkill -u john -t pts/3
```

Kills all of john's processes on terminal pts/3.

Example: Force kill by pattern

```
$ pkill -9 -f 'php artisan queue:work'
```

Force kills all PHP queue workers.

Example: Kill child processes

```
$ pkill -P 1234
```

Kills all direct children of process 1234.

Tips & Best Practices

Pro Tip: Test with pgrep first: Before running pkill, test your pattern with pgrep -a pattern to see which processes would match. This prevents accidentally killing wrong processes.

Warning: -f matches broadly: pkill -f matches the FULL command line. pkill -f 'python' would match any process with 'python' anywhere in its arguments. Be specific.

Note: pkill vs killall: pkill matches by pattern (partial match by default). killall matches by exact name. pkill supports -f for full command line matching.

$ ps

Beginner

Report a snapshot of current running processes

The ps command displays information about active processes. It provides a snapshot of current processes at the moment you run it, showing process IDs, CPU/memory usage, command names, states, and more.

ps supports two main syntax styles: BSD (ps aux) and UNIX (ps -ef). Both show similar informat...

Options & Flags

`aux`	Show all processes with user info (BSD style)
`-ef`	Show all processes with full format (UNIX style)
`-u`	Show processes for a specific user
`-p`	Show info for specific PID
`--sort`	Sort output by field
`-o`	Custom output format
`--forest`	Show process tree
`-C`	Select by command name
`-L`	Show threads

Practical Examples

Example: Show all processes

```
$ ps aux
USER  PID %CPU %MEM COMMAND
root    1  0.0  0.1 /sbin/init
www   234  2.3  1.5 nginx: worker
```

Lists all running processes with user, PID, CPU%, MEM%, and command.

Example: Find a specific process

```
$ ps aux | grep nginx
```

Searches for nginx processes.

Example: Top memory consumers

```
$ ps aux --sort=-%mem | head -10
```

Shows the 10 processes using the most memory.

Example: Top CPU consumers

```
$ ps aux --sort=-%cpu | head -10
```

Shows the 10 processes using the most CPU.

Example: Process tree

```
$ ps -ef --forest
```

Shows parent-child relationships between processes.

Tips & Best Practices

Pro Tip: ps aux vs ps -ef: Both show all processes. ps aux (BSD) shows %CPU, %MEM columns. ps -ef (UNIX) shows PPID (parent PID). Most admins prefer ps aux for resource info.

Note: Snapshot vs real-time: ps shows a single snapshot. For real-time monitoring, use top or htop instead.

Warning: grep includes itself: ps aux | grep nginx also matches the grep process. Fix: ps aux | grep [n]ginx or use pgrep nginx.

$ pstree

Beginner

Display processes as a tree showing parent-child relationships

pstree displays running processes as a tree, showing parent-child relationships. It provides a visual hierarchy of how processes are spawned and organized on the system.

pstree merges identical branches by default, showing the count in brackets. This makes it easy to see how many worker processe...

Options & Flags

`-p`	Show PIDs alongside process names
`-u`	Show user transitions (uid changes)
`-a`	Show command line arguments
`-h`	Highlight current process and ancestors
`-n`	Sort by PID instead of name
`-l`	Long format (do not truncate)
`-s`	Show parents of a specific process

Practical Examples

Example: View process tree

```
$ pstree
systemd???sshd???sshd???bash
        ??nginx???4*[nginx]
        ??php-fpm???8*[php-fpm]
```

Shows all processes in a tree hierarchy.

Example: Tree with PIDs

```
$ pstree -p
```

Shows the tree with process IDs - useful for targeting specific processes.

Example: Show specific user tree

```
$ pstree www-data
```

Shows the process tree for the www-data user.

Example: Show parents of a process

```
$ pstree -s 1234
systemd???sshd???sshd???bash???python3
```

Shows the full ancestry chain from PID 1 to process 1234.

Example: Tree with arguments

```
$ pstree -a
```

Shows command line arguments for each process.

Tips & Best Practices

Pro Tip: Quick service audit: pstree shows how many workers each service has: pstree | grep nginx reveals nginx???4*[nginx] meaning 4 workers.

Note: Merged branches: pstree merges identical children. 4*[nginx] means 4 identical nginx worker processes. Use -c to show them individually.

Warning: Long lines truncated: Use -l to prevent truncation of long command lines. Without it, wide trees may be cut off.

$ renice

Intermediate

Alter priority of running processes

renice changes the scheduling priority (niceness) of running processes. While nice sets priority at launch time, renice modifies priority of processes that are already running.

renice can target processes by PID, by user (all processes of a user), or by process group. It is commonly used to lowe...

Options & Flags

`-n`	Specify niceness value
`-p`	Target process by PID
`-u`	Target all processes of a user
`-g`	Target process group

Practical Examples

Example: Lower process priority

```
$ renice -n 15 -p 1234
1234 (process ID) old priority 0, new priority 15
```

Sets the process to low priority so it does not hog CPU.

Example: Increase priority (root)

```
$ sudo renice -n -5 -p 1234
```

Gives a critical process higher priority.

Example: Renice all user processes

```
$ sudo renice -n 19 -u backupuser
```

Sets all of backupuser's processes to lowest priority.

Example: Fix runaway process

```
$ renice -n 19 -p $(pgrep -f "heavy_process")
```

Drops the priority of a CPU-hogging process without killing it.

Example: Renice process group

```
$ renice -n 10 -g 5678
```

Changes priority for all processes in the process group.

Tips & Best Practices

Pro Tip: Quick priority fix: When a process is hogging CPU: renice -n 19 -p PID. This instantly makes it yield to other processes without killing it.

Warning: Cannot reverse as non-root: Regular users can increase niceness but cannot decrease it. Once you nice a process to 19, you need root to bring it back.

Note: Use htop instead: htop lets you renice processes interactively: select process, press F7/F8 to change niceness. More convenient than command line.

$ screen

Intermediate

Terminal multiplexer for persistent sessions

screen is a terminal multiplexer that creates persistent terminal sessions you can detach from and reattach to later. It allows running multiple shell sessions within a single terminal and keeps them alive after SSH disconnection.

screen is essential for remote server work where SSH connections ...

Options & Flags

`-S`	Name the session
`-r`	Reattach to a detached session
`-ls`	List all sessions
`-d`	Detach a session from another terminal
`-d -r`	Detach and reattach (force)
`-dm`	Start a detached session running a command
`-x`	Attach to a session (multi-display mode)

Practical Examples

Example: Start named session

```
$ screen -S deploy
```

Creates a new screen session named "deploy".

Example: Detach from session

```
$ Ctrl+A then D
[detached from 12345.deploy]
```

Detaches from the current session, leaving it running in the background.

Example: List sessions

```
$ screen -ls
12345.deploy  (Detached)\n67890.backup  (Attached)
```

Shows all running screen sessions.

Example: Reattach to session

```
$ screen -r deploy
```

Reconnects to the deploy session after SSH reconnection.

Example: Start detached with command

```
$ screen -dmS backup rsync -av /data/ /backup/
```

Starts a background session running rsync without attaching to it.

Tips & Best Practices

Pro Tip: Essential key bindings: Ctrl+A is the command prefix. Ctrl+A D=detach, Ctrl+A C=new window, Ctrl+A N=next window, Ctrl+A P=previous, Ctrl+A K=kill window.

Note: screen vs tmux: tmux has better defaults, scripting, and split-pane support. screen is more widely pre-installed. Both solve the same problem.

Warning: Dead sessions: If screen -ls shows 'Dead' sessions, clean them up with screen -wipe.

$ tmux

Intermediate

Terminal multiplexer with session management

tmux (terminal multiplexer) creates persistent terminal sessions with multiple panes and windows. It is the modern replacement for screen, offering better split-pane support, configuration, and scripting.

tmux allows you to create sessions with multiple windows, split windows into panes, detach ...

Options & Flags

`new -s`	Create a named session
`attach -t`	Attach to existing session
`ls`	List all sessions
`kill-session`	Kill a session
`split-window`	Split current pane
`send-keys`	Send keys to a session (scripting)

Practical Examples

Example: Start named session

```
$ tmux new -s project
```

Creates a new tmux session named "project".

Example: Detach

```
$ Ctrl+B then D
[detached (from session project)]
```

Detaches from the session, leaving it running.

Example: List sessions

```
$ tmux ls
project: 3 windows (created Mon Jan 1 12:00:00 2024)
```

Shows all active tmux sessions.

Example: Reattach

```
$ tmux attach -t project
```

Reconnects to the project session.

Example: Split panes

```
$ Ctrl+B then %  (vertical)\nCtrl+B then "  (horizontal)
```

Splits the current window into panes for side-by-side work.

Tips & Best Practices

Pro Tip: Essential shortcuts: Ctrl+B is the prefix. D=detach, C=new window, N=next window, P=previous, %=split vertical, "=split horizontal, arrow=switch pane.

Note: Customize in .tmux.conf: Create ~/.tmux.conf for custom key bindings, mouse support, and appearance. Many people remap the prefix from Ctrl+B to Ctrl+A.

Warning: Prefix key collision: Default prefix Ctrl+B conflicts with some terminal apps. Set a custom prefix in .tmux.conf: set-option -g prefix C-a

$ top

Beginner

Display real-time system resource usage and processes

top provides a real-time, dynamic view of running processes sorted by CPU usage. It is the standard tool for monitoring system resource consumption including CPU, memory, swap, and load averages.

top updates its display at regular intervals (default 3 seconds), showing process ID, user, CPU perc...

Options & Flags

`-d`	Set update interval in seconds
`-p`	Monitor specific PIDs only
`-u`	Show only processes for a user
`-b`	Batch mode (for scripting/logging)
`-n`	Number of iterations before exit
`-H`	Show individual threads
`-i`	Hide idle processes
`-o`	Sort by field
`-c`	Show full command line

Practical Examples

Example: Standard monitoring

```
$ top
```

Opens real-time process monitor. Press q to quit.

Example: Fast refresh

```
$ top -d 1
```

Updates every 1 second for real-time monitoring during incidents.

Example: Monitor specific user

```
$ top -u www-data
```

Shows only processes owned by the www-data user.

Example: Single snapshot for logging

```
$ top -bn 1 | head -20
```

Batch mode captures one snapshot - useful for logging and scripts.

Example: Monitor specific PIDs

```
$ top -p 1234,5678
```

Monitors only the specified process IDs.

Tips & Best Practices

Pro Tip: Interactive shortcuts: While in top: M=sort by memory, P=sort by CPU, k=kill process, r=renice, c=toggle full command, 1=show per-CPU, q=quit.

Note: Load average explained: Load average shows 1/5/15 minute averages of processes wanting CPU. On a 4-core system, load 4.0 means 100% utilization. Above core count indicates overload.

Warning: %CPU can exceed 100%: On multi-core systems, a process can show >100% CPU - it means it is using multiple cores. 200% = 2 full cores.