# Preface

## Welcome

Welcome to The Direct Path to Linux Ubuntu. This book grew out of MyWebUniversity.com, a free educational site I founded on August 5, 2000, to make comprehensive UNIX and Linux training available to anyone, at any pace, at no cost. What you are holding now is the print companion to that site — the same 43-chapter path, expanded with the depth and structure a printed book allows: worked examples, complete code listings, and reference tables you can flip back to without a browser.

I have spent more than twenty-five years working as a systems analyst and instructor — currently as an Enterprise System Analyst IV at NASA's Jet Propulsion Laboratory, and previously teaching Computer Science and Information Science courses at National University, California State University, Fullerton, and Pasadena City College. Everything in this book reflects what I have found actually helps students go from typing their first command to running production systems with confidence.

## How to Use This Book

This book is organized as a direct path, not a reference you must read cover to cover:

- Chapters 1–5 build your foundation: history, basic commands, intermediate commands, advanced commands, and shell scripting.
- Chapters 6–21 cover web servers, manual page systems, and reference documentation — skim these once, then return to them as you need specific lookups.
- Chapters 22–30 are a working tour of nine programming languages: enough syntax, semantics, and compile/run instructions to write and execute real programs in each.

- Chapters 31–36 cover the data and interchange formats every developer meets daily: HTML/CSS, JSON, XML, YAML, SQL, and LDAP.
- Chapters 37–43 move into the operational side of modern IT: cloud computing, automation, containers, web servers in production, security, and networking.

If you are new to Linux, start with Chapter 2 and practice every command yourself rather than only reading it. Familiarity comes from repetition, not memorization. If you already have a UNIX or Linux background, feel free to jump directly to the chapters most relevant to your goals.

## Companion Online Resources

Every chapter in this book has a living companion page at MyWebUniversity.com, including a browser-based Linux practice terminal (Chapter 15), over 9,000 searchable Ubuntu 26.04 LTS manual pages in PDF and HTML (Chapter 21), and video walkthroughs in both English and Dari (Chapters 17–20). The book stands on its own, but the website is there whenever you want to see a command run live or watch a topic explained on video.

## Dedication

*This book is dedicated to the memory of my beloved brother, Dr. Amanullah Lutfy (R.I.P.), whose support and belief in education shaped the mission behind MyWebUniversity.com. [Author: feel free to expand this dedication in your own words — this is a starting placeholder based on the tribute already published on your site.]*

## Before You Begin

Keep a terminal open as you read. Every command and code example in this book is meant to be typed, run, and experimented with — not just read. Good luck, and welcome to the direct path.

# Chapter 1

# About UNIX and Linux Operating Systems

*History and foundations*

## 1.1 The Birth of UNIX at Bell Labs (1969)

UNIX was created in 1969 at AT&T's Bell Laboratories by Ken Thompson and Dennis Ritchie, growing out of an earlier, more ambitious project called Multics (Multiplexed Information and Computing Service) that Bell Labs had abandoned as too complex. Thompson wrote the first version of UNIX in assembly language on a spare PDP-7 minicomputer; by 1973, Ritchie and Thompson had rewritten the kernel in C, a language Ritchie designed specifically to make an operating system portable across different hardware — a radical idea at the time, since operating systems were traditionally written in assembly and tied to one machine.

Bell Labs, then part of AT&T, was barred by a 1956 antitrust consent decree from selling computer software commercially, so it licensed UNIX source code to universities for a nominal fee. That decision seeded a generation of computer scientists who studied, modified, and extended UNIX — most notably at the University of California, Berkeley, whose students and staff produced the Berkeley Software Distribution (BSD), adding virtual memory, the TCP/IP networking stack, and the vi editor. AT&T's own commercial lineage became System V. The two branches diverged enough in behavior that the 1980s and early 1990s saw genuine incompatibilities — the so-called "UNIX wars" — which the POSIX standard (IEEE Std 1003) was created to resolve by defining a common set of system calls, utilities, and shell behavior that any compliant UNIX-like system should support.

AT&T's Bell Laboratories later became part of Lucent Technologies, which in 2006 merged with the French company Alcatel to form Alcatel-Lucent (today part of Nokia). The AT&T Archives film "UNIX: Making Computers Easier to Use" (1982) is a good primary-

source watch if you want to see Thompson and Ritchie's original motivations in their own words.

## 1.2 The GNU Project and the Birth of Linux

In 1983, Richard Stallman launched the GNU Project ("GNU's Not Unix") with the goal of building a completely free, UNIX-compatible operating system — free as in freedom, not necessarily free of charge. Over the 1980s, the GNU Project produced high-quality replacements for nearly every UNIX userland tool: the GNU Compiler Collection (GCC), the Bash shell, GNU Emacs, and the GNU Core Utilities (cat, ls, grep, and the rest of the commands you met in Chapters 2–4). What GNU lacked was a kernel.

That gap was filled in 1991, when Linus Torvalds, a 21-year-old student at the University of Helsinki, posted to the Usenet newsgroup comp.os.minix that he was writing "a (free) operating system (just a hobby, won't be big and professional like GNU)" for 386(486) AT clones. That kernel became Linux. Combined with the existing GNU userland tools, the result is properly called GNU/Linux, though common usage shortened it to simply "Linux." Because Torvalds released the kernel under the GNU General Public License (GPL), anyone could study, modify, and redistribute it — a decision that is arguably the single biggest reason Linux spread as fast as it did.

## 1.3 From Kernel to Distribution: How Ubuntu Fits In

A Linux kernel by itself does not make a usable operating system — it needs a bootloader, core utilities, package manager, and (usually) a desktop environment bundled around it. A "distribution" (distro) is exactly that bundle. Early distributions like Slackware (1993) and Debian (1993) established the model; Debian in particular introduced the APT package management system that Ubuntu still uses today.

Ubuntu itself was founded in 2004 by Mark Shuttleworth through his company Canonical Ltd., built directly on top of Debian, with a stated mission of making Linux accessible to ordinary desktop users, not only engineers. Ubuntu ships new releases every six months,

with a Long-Term Support (LTS) release every two years — identified by a year.month version number, so Ubuntu 26.04 LTS was released in April 2026 and is supported with security updates for years afterward. This book is written against Ubuntu 26.04 LTS throughout.

Other major UNIX and Linux-family operating systems you will encounter in the industry include Red Hat Enterprise Linux (RHEL) and its community rebuild Rocky Linux (both covered briefly in Chapters 10–11), SUSE, Sun Microsystems' Solaris (now Oracle Solaris), IBM's AIX, Hewlett-Packard's HP-UX, SGI's IRIX, and Apple's macOS, which is built on a BSD-derived UNIX core (Darwin) and is UNIX-certified.

## 1.4 UNIX vs. Linux: What's Actually Different

"UNIX" today refers both to the original AT&T lineage and to the trademark held by The Open Group, which certifies operating systems (including macOS) as officially UNIX-compliant. Linux was never certified UNIX — it is UNIX-like (or POSIX-compliant), built from scratch rather than descended from the original source tree, which is why you will sometimes see it written as *NIX or Un*x to mean "UNIX and its lookalikes" collectively.

In practice, day-to-day command usage is nearly identical between them: the same shells, the same core utilities, the same directory hierarchy philosophy. The differences that matter most in production are kernel internals, licensing, and vendor support — which this book addresses concretely rather than abstractly, since Chapters 10 and 11 walk through Rocky Linux and Red Hat Enterprise Linux specifically.

## 1.5 Multitasking and Multithreading

UNIX was one of the first operating systems designed from the ground up for preemptive multitasking — the kernel's scheduler can interrupt any running process at any time to give another process CPU time, so users perceive many programs running "at once" even on a single processor core. A process is an independently scheduled

unit with its own memory space; a thread is a lighter-weight unit of execution that shares memory with other threads inside the same process, letting one program perform several tasks concurrently without the overhead of separate memory spaces. Linux implements both through the same underlying primitive, the task_struct, which is one of the reasons Linux threading (via the pthreads library) is efficient enough to underlie almost every modern server workload, from web servers to databases.

## 1.6 System Administration Fundamentals

System administration on UNIX and Linux is built around a small number of consistent ideas that recur throughout this book: everything is a file (devices, sockets, and even some kernel interfaces appear in the filesystem under /dev, /proc, and /sys); permissions are enforced per user and group (owner/group/other, read/write/execute — see chmod and chown in Chapter 3); processes are owned by users and can be inspected, signaled, and killed (Chapters 2 and 4); and the root user (UID 0) can bypass permission checks entirely, which is why sudo, rather than routinely logging in as root, is the modern best practice.

## 1.7 Shells: sh, csh, ksh, bash, tcsh, zsh

The shell is the command interpreter that sits between you and the kernel. UNIX has produced several shell lineages, and Linux distributions typically ship more than one:

- sh — the original Bourne shell (Stephen Bourne, 1977), the ancestor of most modern shells and still the POSIX baseline for portable scripts.
- csh / tcsh — the C shell (Bill Joy, BSD), with C-like syntax; tcsh adds command-line editing and completion.
- ksh — the Korn shell, which merged Bourne-shell compatibility with csh-like interactive features.
- bash — the GNU Bourne Again Shell, the default login shell on virtually every Linux distribution including Ubuntu, and the shell used throughout Chapter 5's scripting examples.

- zsh — a more feature-rich shell popular in developer environments (and the default on modern macOS), largely bash-compatible with extra scripting conveniences.

You can check your current shell at any time with echo $SHELL, and switch shells for a session with the shell's own command (for example, typing zsh to start a Z shell session).

## 1.8 Networking, Remote Access, and Web Servers — A Preview

Classic UNIX remote-access tools — rsh, rcp, rexec, telnet, and ftp — transmit credentials and data in plain text and are considered obsolete for anything but isolated lab use. Modern systems use SSH (Secure Shell, via the sshd daemon) for encrypted remote login, remote command execution, and secure file copy, and this book uses ssh and its companions (scp, rsync) throughout later chapters. Web serving on Linux is covered in depth in Chapter 6 (Apache and historic alternatives) and Chapter 41 (Apache and NGINX in production), including the classic "LAMP stack" (Linux, Apache, MySQL, PHP/Python/Perl) that still underlies a large share of the web today.

## 1.9 Compilers, Linkers, and Debuggers — A Preview

UNIX popularized a toolchain model that Linux inherited wholesale: a compiler translates source code to object files (gcc for C, discussed fully in Chapter 22), the assembler (as) turns assembly into machine code, the linker (ld) combines object files and libraries into a final executable, and a debugger (gdb, or the historic adb and mdb on other UNIX variants) lets you step through a running program's execution. You saw several of these tools already in Chapter 4, and Chapters 22–30 put them to use across nine different programming languages.

## 1.10 Your Homework

Before moving on to Chapter 2, spend some time researching the following topics on your own — using Google, the official Ubuntu

documentation, or MyWebUniversity.com's manual page archive. You do not need to master them yet; the goal is simply to recognize the vocabulary before you meet it again in later chapters:

- UNIX vs. LINUX, and the UNIX and LINUX kernel
- UNIX and LINUX history, multitasking, and multithreading
- UNIX and LINUX system administration fundamentals
- Sun Solaris (Sun Microsystems / Oracle)
- Remote access tools: rsh, rcp, rexec, ftp, telnet, sshd
- Shells: sh, ksh, bash, csh, tcsh, zsh
- Toolchain commands: ld, as, link, make, gcc, mdb, adb, jdb
- Manual page tools: man, whatis, whereis, locate
- Web servers: Apache, Netscape, Tomcat, iPlanet
- Early UNIX-adjacent languages: C, C++, Lisp, Fortran, Pascal, Java, Perl, and databases such as MySQL and Oracle
- Other UNIX and Linux variants: AT&T UNIX, FreeBSD, IRIX, Fedora, openSUSE, CentOS, AIX, HP-UX, Clix, DEC OSF/1

With this foundation in place, Chapter 2 moves from history into practice, with your first hands-on Linux commands.

# Chapters 2–5 — Draft Content

*Command lists and shell-script names sourced from MyWebUniversity.com. Descriptions, examples, and sample output written for this manuscript — review and adjust output samples to match your live server before publishing.*

# Chapter 2

## Basic UNIX and Linux Commands

## Chapter 2 Command Deep-Dives

**banner • cal • cat • nslookup • mkdir**

*Transcribed from the live, working examples at MyWebUniversity.com/cgi-bin/CH2.cgi, matching Chapter 2: Basic UNIX and Linux Commands. Every example below reproduces the actual output generated by the site's online Ubuntu 26.04 LTS terminal at the time of writing.*

# 1. banner — print large banner text

```
$ whatis banner
banner (1) - make posters
```

banner renders text as large, poster-sized block letters made of ASCII characters — originally meant for dot-matrix line printers, and still a fun way to make a heading stand out on a terminal or in a printed log. Below are all eight of the worked examples from the companion site, each showing the exact command and its generated output.

**$ banner A**

```
   #
  # #
 #   #
#     #
#######
#     #
#     #
```

**$ banner 2026 Happy New Year**

```
 #####    ###    #####   #####
#     #  #   #  #     # #     #
      # # #   #       # #
 #####  #  #  #  #####  ######
#       #   # # #       #     #
#        #   #  #       #     #
#######   ###   #######  #####

#     #
#     #    ##    #####   #####    #   #
#     #   #  #   #    #  #    #    # #
#######  #    #  #    #  #    #     #
#     #  ######  #####   #####      #
#     #  #    #  #       #          #
#     #  #    #  #       #          #

#     #
##    #  ######  #    #
# #   #  #       #    #
#  #  #  #####   #    #
#   # #  #       # ## #
#    ##  #       ##  ##
#     #  ######  #    #
```

```
#     #
 #   #   ######    ##    #####
  # #    #        #  #   #    #
   #     #####   #    #  #    #
   #     #       ######  #####
   #     #       #    #  #   #
   #     ######  #    #  #    #
```

**$ banner A B C**

```
   #
  # #
 #   #
#     #
#######
#     #
#     #

######
#     #
#     #
######
#     #
#     #
######

 #####
#     #
#
#
#
#     #
 #####
```

**$ banner Peace For All**

```
######
#     #  ######    ##     ####   ######
#     #  #        #  #   #    #  #
######   #####   #    #  #       #####
#        #       ######  #       #
#        #       #    #  #    #  #
#        ######  #    #   ####   ######

#######
#         ####   #####
#        #    #  #    #
#####    #    #  #    #
#        #    #  #####
#        #    #  #   #
#         ####   #    #
```

```
   #
  # #    #       #
 #   #   #       #
#     #  #       #
#######  #       #
#     #  #       #
#     #  ######  ######
```

**$ banner Thanks For Watching!**

```
#######
   #     #    #    ##    #    #  #    #   ####
   #     #    #   #  #   ##   #  #   #   #
   #     ######  #    #  # #  #  ####     ####
   #     #    #  ######  #  # #  #  #         #
   #     #    #  #    #  #   ##  #   #   #    #
   #     #    #  #    #  #    #  #    #   ####


#######
#         ####   #####
#        #    #  #    #
#####    #    #  #    #
#        #    #  #####
#        #    #  #   #
#         ####   #    #


#     #                                                    ###
#  #  #    ##     #####   ####   #    #     #    #    #   ####    ###
#  #  #   #  #      #    #    #  #    #     #    ##   #  #    #   ###
#  #  #  #    #     #    #       ######     #    # #  #  #         #
#  #  #  ######     #    #       #    #     #    #  # #  #  ###
#  #  #  #    #     #    #    #  #    #     #    #   ##  #    #   ###
 ## ##   #    #     #     ####   #    #     #    #    #   ####    ###
```

**$ banner Please Join Our Channel**

```
######
#     #  #       ######    ##     ####   ######
#     #  #       #        #  #   #       #
######   #       #####   #    #   ####   #####
#        #       #       ######       #  #
#        #       #       #    #  #    #  #
#        ######  ######  #    #   ####   ######


      #
      #   ####      #    #    #
      #  #    #     #    ##   #
      #  #    #     #    # #  #
#     #  #    #     #    #  # #
#     #  #    #     #    #   ##
```

```
 #####    ####      #    #    #


#######
#     #  #    #  #####
#     #  #    #  #    #
#     #  #    #  #    #
#     #  #    #  #####
#     #  #    #  #   #
#######   ####   #    #


 #####
#     #  #    #    ##    #    #  #    #  ######  #
#        #    #   #  #   ##   #  ##   #  #       #
#        ######  #    #  # #  #  # #  #  #####   #
#        #    #  ######  #  # #  #  # #  #       #
#     #  #    #  #    #  #   ##  #   ##  #       #
 #####   #    #  #    #  #    #  #    #  ######  ######
```

**$ banner The Direct Way To Linux Ubuntu**

```
#######
   #     #    #  ######
   #     #    #  #
   #     ######  #####
   #     #    #  #
   #     #    #  #
   #     #    #  ######


######
#     #     #    #####   ######   ####    #####
#     #     #    #    #  #       #    #     #
#     #     #    #    #  #####   #          #
#     #     #    #####   #       #          #
#     #     #    #   #   #       #    #     #
######      #    #    #  ######   ####      #


#     #
#  #  #    ##     #   #
#  #  #   #  #     # #
#  #  #  #    #     #
#  #  #  ######     #
#  #  #  #    #     #
 ## ##   #    #     #


#######
   #      ####
   #     #    #
   #     #    #
   #     #    #
   #     #    #
```

```
   #      ####


#
#           #    #    #  #    #  #    #
#           #    ##   #  #    #   #  #
#           #    # #  #  #    #    ##
#           #    #  # #  #    #    ##
#           #    #   ##  #    #   #  #
#######     #    #    #   ####   #    #


#     #
#     #  #####   #    #  #    #   #####  #    #
#     #  #    #  #    #  ##   #     #    #    #
#     #  #####   #    #  # #  #     #    #    #
#     #  #    #  #    #  #  # #     #    #    #
#     #  #    #  #    #  #   ##     #    #    #
 #####   #####    ####   #    #     #     ####
```

**$ banner Please Subscribe Our Channel**

```
######
#     #  #       ######    ##     ####   ######
#     #  #       #        #  #   #       #
######   #       #####   #    #   ####   #####
#        #       #       ######       #  #
#        #       #       #    #  #    #  #
#        ######  ######  #    #   ####   ######


 #####
#     #  #    #  #####    ####    ####   #####      #    #####   ######
#        #    #  #    #  #       #    #  #    #     #    #    #  #
 #####   #    #  #####    ####   #       #    #     #    #####   #####
      #  #    #  #    #       #  #       #####      #    #    #  #
#     #  #    #  #    #  #    #  #    #  #   #      #    #    #  #
 #####    ####   #####    ####    ####   #    #     #    #####   ######


#######
#     #  #    #  #####
#     #  #    #  #    #
#     #  #    #  #    #
#     #  #    #  #####
#     #  #    #  #   #
#######   ####   #    #


 #####
#     #  #    #    ##    #    #  #    #  ######  #
#        #    #   #  #   ##   #  ##   #  #       #
#        ######  #    #  # #  #  # #  #  #####   #
#        #    #  ######  #  # #  #  # #  #       #
#     #  #    #  #    #  #   ##  #   ##  #       #
```

```
#####   #    #  #    #  #    #  #    #  ######  ######
```

## 2. cal — display a calendar

```
$ whatis cal
cal (1) - display a calendar
```

cal prints a calendar for a single month, a full year, or a range of months around the current date, depending on its arguments. All ten worked examples from the companion site are reproduced here — note that the full-year and multi-month examples are set in a smaller monospace size so the columns stay aligned within the page width.

**$ cal**

```
     July 2026
Su Mo Tu We Th Fr Sa
           1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

**$ cal -j**

```
          July 2026
 Su  Mo  Tu  We  Th  Fr  Sa
            182 183 184 185
186 187 188 189 190 191 192
193 194 195 196 197 198 199
200 201 202 203 204 205 206
207 208 209 210 211 212
```

**$ cal 1931**

```
                               1931

      January               February                March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7   1  2  3  4  5  6  7
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   8  9 10 11 12 13 14
11 12 13 14 15 16 17  15 16 17 18 19 20 21  15 16 17 18 19 20 21
18 19 20 21 22 23 24  22 23 24 25 26 27 28  22 23 24 25 26 27 28
25 26 27 28 29 30 31                        29 30 31


      April                 May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                  1  2      1  2  3  4  5  6
 5  6  7  8  9 10 11   3  4  5  6  7  8  9   7  8  9 10 11 12 13
12 13 14 15 16 17 18  10 11 12 13 14 15 16  14 15 16 17 18 19 20
19 20 21 22 23 24 25  17 18 19 20 21 22 23  21 22 23 24 25 26 27
26 27 28 29 30        24 25 26 27 28 29 30  28 29 30
```

```
                    31


        July                August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1           1  2  3  4  5
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   6  7  8  9 10 11 12
12 13 14 15 16 17 18   9 10 11 12 13 14 15  13 14 15 16 17 18 19
19 20 21 22 23 24 25  16 17 18 19 20 21 22  20 21 22 23 24 25 26
26 27 28 29 30 31     23 24 25 26 27 28 29  27 28 29 30
                      30 31


      October             November            December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7           1  2  3  4  5
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   6  7  8  9 10 11 12
11 12 13 14 15 16 17  15 16 17 18 19 20 21  13 14 15 16 17 18 19
18 19 20 21 22 23 24  22 23 24 25 26 27 28  20 21 22 23 24 25 26
25 26 27 28 29 30 31  29 30                 27 28 29 30 31
```

## $ cal 1970

```
                              1970
      January                 February                March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7   1  2  3  4  5  6  7
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   8  9 10 11 12 13 14
11 12 13 14 15 16 17  15 16 17 18 19 20 21  15 16 17 18 19 20 21
18 19 20 21 22 23 24  22 23 24 25 26 27 28  22 23 24 25 26 27 28
25 26 27 28 29 30 31                       29 30 31


      April                   May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                  1  2      1  2  3  4  5  6
 5  6  7  8  9 10 11   3  4  5  6  7  8  9   7  8  9 10 11 12 13
12 13 14 15 16 17 18  10 11 12 13 14 15 16  14 15 16 17 18 19 20
19 20 21 22 23 24 25  17 18 19 20 21 22 23  21 22 23 24 25 26 27
26 27 28 29 30        24 25 26 27 28 29 30  28 29 30
                      31


       July                  August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1           1  2  3  4  5
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   6  7  8  9 10 11 12
12 13 14 15 16 17 18   9 10 11 12 13 14 15  13 14 15 16 17 18 19
19 20 21 22 23 24 25  16 17 18 19 20 21 22  20 21 22 23 24 25 26
26 27 28 29 30 31     23 24 25 26 27 28 29  27 28 29 30
                      30 31


      October             November            December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7           1  2  3  4  5
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   6  7  8  9 10 11 12
11 12 13 14 15 16 17  15 16 17 18 19 20 21  13 14 15 16 17 18 19
18 19 20 21 22 23 24  22 23 24 25 26 27 28  20 21 22 23 24 25 26
25 26 27 28 29 30 31  29 30                 27 28 29 30 31
```

## $ cal 2 1998

```
   February 1998
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

**$ cal 6 2001**

```
     June 2001
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

**$ cal -A 2**

```
                              2026
        July                   August                 September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1           1  2  3  4  5
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   6  7  8  9 10 11 12
12 13 14 15 16 17 18   9 10 11 12 13 14 15  13 14 15 16 17 18 19
19 20 21 22 23 24 25  16 17 18 19 20 21 22  20 21 22 23 24 25 26
26 27 28 29 30 31     23 24 25 26 27 28 29  27 28 29 30
                      30 31
```

**$ cal -A 5**

```
                              2026
        July                   August                 September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1           1  2  3  4  5
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   6  7  8  9 10 11 12
12 13 14 15 16 17 18   9 10 11 12 13 14 15  13 14 15 16 17 18 19
19 20 21 22 23 24 25  16 17 18 19 20 21 22  20 21 22 23 24 25 26
26 27 28 29 30 31     23 24 25 26 27 28 29  27 28 29 30
                      30 31


      October                November                December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7           1  2  3  4  5
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   6  7  8  9 10 11 12
11 12 13 14 15 16 17  15 16 17 18 19 20 21  13 14 15 16 17 18 19
18 19 20 21 22 23 24  22 23 24 25 26 27 28  20 21 22 23 24 25 26
25 26 27 28 29 30 31  29 30                 27 28 29 30 31
```

**$ cal -B 2**

```
      May 2026              June 2026             July 2026
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                1  2      1  2  3  4  5  6           1  2  3  4
```

```
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   5  6  7  8  9 10 11
10 11 12 13 14 15 16  14 15 16 17 18 19 20  12 13 14 15 16 17 18
17 18 19 20 21 22 23  21 22 23 24 25 26 27  19 20 21 22 23 24 25
24 25 26 27 28 29 30  28 29 30              26 27 28 29 30 31
31
```

## $ cal -B 5

```
   February 2026           March 2026            April 2026
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7   1  2  3  4  5  6  7            1  2  3  4
 8  9 10 11 12 13 14   8  9 10 11 12 13 14   5  6  7  8  9 10 11
15 16 17 18 19 20 21  15 16 17 18 19 20 21  12 13 14 15 16 17 18
22 23 24 25 26 27 28  22 23 24 25 26 27 28  19 20 21 22 23 24 25
                      29 30 31              26 27 28 29 30


      May 2026              June 2026             July 2026
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                1  2      1  2  3  4  5  6            1  2  3  4
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   5  6  7  8  9 10 11
10 11 12 13 14 15 16  14 15 16 17 18 19 20  12 13 14 15 16 17 18
17 18 19 20 21 22 23  21 22 23 24 25 26 27  19 20 21 22 23 24 25
24 25 26 27 28 29 30  28 29 30              26 27 28 29 30 31
31
```

## 3. cat — concatenate and print files

```
$ whatis cat
cat (1) - concatenate files and print on the standard
output
cat (1) - concatenate and display files
```

cat prints the contents of one or more files to standard output, and is equally often used to combine several files into one continuous stream. All six worked examples read from real Ubuntu 26.04 LTS system files — note the codename "Resolute Raccoon" confirmed in the live output below.

**$ cat -n /etc/os-release**

```
 1      PRETTY_NAME="Ubuntu 26.04 LTS"
 2      NAME="Ubuntu"
 3      VERSION_ID="26.04"
 4      VERSION="26.04 LTS (Resolute Raccoon)"
 5      VERSION_CODENAME=resolute
 6      ID=ubuntu
 7      ID_LIKE=debian
 8      HOME_URL="https://www.ubuntu.com/"
 9      SUPPORT_URL="https://help.ubuntu.com/"
10      BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
11      PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-
and-policies/privacy-policy"
12      UBUNTU_CODENAME=resolute
13      LOGO=ubuntu-logo
```

**$ cat -EA /etc/os-release**

```
PRETTY_NAME="Ubuntu 26.04 LTS"$
NAME="Ubuntu"$
VERSION_ID="26.04"$
VERSION="26.04 LTS (Resolute Raccoon)"$
VERSION_CODENAME=resolute$
ID=ubuntu$
ID_LIKE=debian$
HOME_URL="https://www.ubuntu.com/"$
SUPPORT_URL="https://help.ubuntu.com/"$
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"$
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-
policies/privacy-policy"$
UBUNTU_CODENAME=resolute$
LOGO=ubuntu-logo$
```

**$ cat -n /etc/networks**

```
1       # symbolic names for networks, see networks(5) for more
information
2       link-local 169.254.0.0
```

**$ cat /etc/networks /etc/os-release**

```
# symbolic names for networks, see networks(5) for more information
link-local 169.254.0.0
PRETTY_NAME="Ubuntu 26.04 LTS"
NAME="Ubuntu"
VERSION_ID="26.04"
VERSION="26.04 LTS (Resolute Raccoon)"
VERSION_CODENAME=resolute
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-
policies/privacy-policy"
UBUNTU_CODENAME=resolute
LOGO=ubuntu-logo
```

**$ cat /etc/os-release**

```
PRETTY_NAME="Ubuntu 26.04 LTS"
NAME="Ubuntu"
VERSION_ID="26.04"
VERSION="26.04 LTS (Resolute Raccoon)"
VERSION_CODENAME=resolute
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-
policies/privacy-policy"
UBUNTU_CODENAME=resolute
LOGO=ubuntu-logo
```

**$ cat -n /etc/os-release /etc/networks**

```
1       PRETTY_NAME="Ubuntu 26.04 LTS"
2       NAME="Ubuntu"
3       VERSION_ID="26.04"
4       VERSION="26.04 LTS (Resolute Raccoon)"
5       VERSION_CODENAME=resolute
6       ID=ubuntu
7       ID_LIKE=debian
```

```
 8      HOME_URL="https://www.ubuntu.com/"
 9      SUPPORT_URL="https://help.ubuntu.com/"
10      BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
11      PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-
and-policies/privacy-policy"
12      UBUNTU_CODENAME=resolute
13      LOGO=ubuntu-logo
14      # symbolic names for networks, see networks(5) for more
information
15      link-local 169.254.0.0
```

## 4. nslookup — query Internet name servers interactively

```
$ whatis nslookup
nslookup - query Internet name servers interactively
```

nslookup queries DNS to resolve a hostname to its IP address (or vice versa). All three worked examples query real domains against the local systemd-resolved stub resolver at 127.0.0.53, which is the default resolver address on Ubuntu 26.04 LTS.

```
$ nslookup www.mywebuniversity.com
Server:             127.0.0.53
Address:      127.0.0.53#53

Name:  www.mywebuniversity.com
Address: 192.168.254.29
```

```
$ nslookup ourunix.com
Server:             127.0.0.53
Address:      127.0.0.53#53

Non-authoritative answer:
Name:  ourunix.com
Address: 47.180.19.94
```

```
$ nslookup -q=A www.mywebuniversity.com
Server:             127.0.0.53
Address:      127.0.0.53#53

Name:  www.mywebuniversity.com
Address: 192.168.254.29
```

The 127.0.0.53#53 address is systemd-resolved's local stub listener, not an external DNS server — it forwards queries on to your configured upstream resolvers (see /etc/resolv.conf and Chapter 43 for more on DNS resolution).

# 5. mkdir — make directories

```
$ whatis mkdir
mkdir (1) - make directories
```

mkdir creates one or more new, empty directories. Unlike the previous four commands, mkdir produces no output when it succeeds — silence is confirmation, and you verify the result with ls afterward. Both worked examples from the companion site are shown below, with that verification step added.

```
$ mkdir new
```

```
$ mkdir new
$ ls -ld new
drwxr-xr-x 2 wahid wahid 4096 Jul 25 09:00 new
```

Creates a single new directory named new in the current working directory.

```
$ mkdir -p dir1/subdir-A/subdir-B
```

```
$ mkdir -p dir1/subdir-A/subdir-B
$ find dir1
dir1
dir1/subdir-A
dir1/subdir-A/subdir-B
```

The -p (parents) flag creates dir1, then subdir-A inside it, then subdir-B inside that, all in a single command — and, importantly, does not raise an error if some of those parent directories already exist, which makes it the safer default for scripts that need to guarantee a directory path exists.

*Core commands every Linux Ubuntu user should know, with one worked example each.*

## banner

**whatis:** *print large banner text on the printer/terminal*

```
$ banner Linux
```

**Result:** Prints the word "Linux" in large block letters made of asterisks across the terminal.

## cal

**whatis:** *displays a calendar*

```
$ cal 2026
```

**Result:** Displays a full 12-month calendar for the year 2026, one month per block.

## cat

**whatis:** *concatenate and print files*

```
$ cat notes.txt
```

**Result:** Prints the entire contents of notes.txt to the terminal.

## clear

**whatis:** *clear the terminal screen*

```
$ clear
```

**Result:** Clears all previous output from the terminal window, leaving a fresh prompt at the top.

## date

**whatis:** *print or set the system date and time*

```
$ date
```

**Result:** Wed Jul 23 09:41:07 PDT 2026

## echo

**whatis:** *display a line of text*

```
$ echo "Hello, Ubuntu!"
```

**Result:** Hello, Ubuntu!

## head

**whatis:** *output the first part of a file*

```
$ head -n 5 access.log
```

**Result:** Prints the first 5 lines of access.log.

## hostname

**whatis:** *show or set the system's host name*

```
$ hostname
```

**Result:** ubuntu-server-01

## host

**whatis:** *DNS lookup utility*

```
$ host mywebuniversity.com
```

**Result:** Resolves mywebuniversity.com to its IP address(es) and shows the A/AAAA records.

## id

**whatis:** *print real and effective user and group IDs*

```
$ id
```

**Result:** uid=1000(wahid) gid=1000(wahid) groups=1000(wahid),27(sudo)

## info

**whatis:** *read Info documents*

```
$ info coreutils
```

**Result:** Opens the GNU Info hypertext browser for the coreutils documentation.

## paste

**whatis:** *merge lines of files*

```
$ paste file1.txt file2.txt
```

**Result:** Joins corresponding lines of file1.txt and file2.txt side by side, separated by a tab.

## ps

**whatis:** *report a snapshot of current processes*

```
$ ps aux
```

**Result:** Lists every running process with user, PID, CPU/memory usage, and command.

## pwd

**whatis:** *print name of current/working directory*

```
$ pwd
```

**Result:** /home/wahid/projects

## uname

**whatis:** *print system information*

```
$ uname -a
```

**Result:** Linux ubuntu-server-01 6.8.0-generic #1 SMP x86_64 GNU/Linux

## less

**whatis:** *view file contents one screen at a time*

```
$ less /var/log/syslog
```

**Result:** Opens syslog in a scrollable pager; press q to quit, / to search.

## ls

**whatis:** *list directory contents*

```
$ ls -ltr
```

**Result:** Lists files in the current directory in long format, sorted by modification time (oldest first).

## man

**whatis:** *display the manual page for a command*

```
$ man ls
```

**Result:** Opens the full manual page for the ls command, including its options and usage.

## mkdir

**whatis:** *make directories*

```
$ mkdir project_backup
```

**Result:** Creates a new directory named project_backup in the current location.

## more

**whatis:** *view file contents page by page*

```
$ more report.txt
```

**Result:** Displays report.txt one screen at a time; press Space to advance.

## pg

**whatis:** *browse pagewise through a text file*

```
$ pg install.log
```

**Result:** Displays install.log a page at a time with prompts to continue.

## ping

**whatis:** *send ICMP ECHO_REQUEST to network hosts*

```
$ ping -c 4 8.8.8.8
```

**Result:** Sends 4 ICMP packets to 8.8.8.8 and reports round-trip time and packet loss.

## dig

**whatis:** *DNS lookup utility*

```
$ dig mywebuniversity.com
```

**Result:** Returns detailed DNS resolution data including the ANSWER section with resolved IP addresses.

## nslookup

**whatis:** *query Internet name servers interactively*

```
$ nslookup mywebuniversity.com
```

**Result:** Server: 127.0.0.53 Address: mywebuniversity.com <resolved-IP>

## atrm

**whatis:** *delete jobs queued by at*

```
$ atrm 3
```

**Result:** Removes queued job number 3 from the at job scheduler.

## atq

**whatis:** *list pending jobs queued by at*

```
$ atq
```

**Result:** Lists all jobs currently scheduled with at, showing job number, date, and queue.

## sort

**whatis:** *sort lines of text files*

```
$ sort names.txt
```

**Result:** Prints names.txt with all lines sorted alphabetically.

## touch

**whatis:** *change file timestamps / create empty files*

```
$ touch newfile.txt
```

**Result:** Creates newfile.txt if it doesn't exist, or updates its modification time if it does.

## w

**whatis:** *show who is logged on and what they are doing*

```
$ w
```

**Result:** Displays the current uptime, load average, and a table of logged-in users with their active processes.

## who

**whatis:** *show who is logged on*

```
$ who
```

**Result:** wahid pts/0 2026-07-23 09:15 (192.168.1.20)

# Chapter 3

# Intermediate UNIX and Linux Commands

# Chapter 3 Command Deep-Dives

**chmod • chgrp • chown • passwd • alias**

*Transcribed and formatted from MyWebUniversity.com/cgi-bin/CH3.cgi, matching Chapter 3: Intermediate UNIX and Linux Commands.*

## A Note on UNIX and Linux Examples

Although this book is titled The Direct Path to Linux Ubuntu and is written primarily against Ubuntu 26.04 LTS, you will notice that the five worked sessions in this chapter were captured on OpenSolaris rather than Ubuntu — visible in details like uid=101(wahid), group staff, and the use of zoneadm and su -. This is intentional, not an oversight. As Chapter 1 explains, Linux is a UNIX-like operating system built independently but sharing the same POSIX foundation, and commands such as chmod, chgrp, chown, passwd, and alias behave almost identically across UNIX and Linux precisely because that shared heritage runs deep. Including examples from both traditions is a deliberate choice: it shows you the small, real differences you will actually encounter as a working systems administrator moving between platforms — Solaris, Ubuntu, Red Hat, or otherwise — rather than pretending every UNIX-family system behaves in lockstep. Where a detail is specific to Solaris (such as the zoneadm command or the su - environment behavior) rather than universal, this book calls it out explicitly, so you always know which parts are portable knowledge and which are platform-specific.

# 1. chmod — change file mode bits

```
$ whatis chmod
chmod (1) - change file mode bits
chmod (1) - change the permissions mode of a file
chmod (2) - change access permission mode of file
```

chmod changes the access permissions for one or more files and directories. To understand it fully, you first need to understand umask, which determines the default permissions new files and directories receive. In this example session, user wahid has a umask of 0022, meaning new directories default to rwxr-xr-x (755) and new files default to rw-r--r-- (644) — the umask value is subtracted from the maximum possible permission (777 for directories, 666 for files).

## Permission bits, symbolic and octal

| Symbolic | Octal | Meaning |
|---|---|---|
| rwx | 7 | All permissions: read, write, and execute |
| rw- | 6 | Read and write, no execute |
| r-x | 5 | Read and execute, no write |
| r-- | 4 | Read only |
| -wx | 3 | Write and execute, no read |
| -w- | 2 | Write only |
| --x | 1 | Execute only |
| --- | 0 | No access at all |

Each digit is the sum of read (4), write (2), and execute (1) for owner, group, and others respectively. For example, rwxr-xr-x = 7,5,5 = 755: full access for the owner, read+execute for group and others — a typical permission set for an executable script.

## Checking identity and umask

```
$ whoami
wahid
$ id
uid=101(wahid) gid=10(staff) groups=10(staff)
$ umask
0022
```

### Creating a file and directory, and reading their default permissions

```
$ mkdir newdir
$ touch newfile
$ ls -ld newdir newfile
drwxr-xr-x 2 wahid staff 2 2011-01-15 21:09 newdir
-rw-r--r-- 1 wahid staff 0 2011-01-15 21:10 newfile
```

777 − 022 = 755 for the new directory; 666 − 022 = 644 for the new file — exactly what umask predicts.

### Changing permissions with chmod

```
$ chmod 777 newdir
$ ls -ld newdir
drwxrwxrwx 2 wahid staff 2 2011-01-15 21:09 newdir

$ chmod 755 newdir
$ ls -ld newdir
drwxr-xr-x 2 wahid staff 2 2011-01-15 21:09 newdir

$ chmod 664 newfile pkg-publisher this
$ ls -l
total 16
drwxr-xr-x 2 wahid staff 2 2011-01-15 21:09 newdir
-rw-rw-r-- 1 wahid staff 0 2011-01-15 21:10 newfile
-rw-rw-r-- 1 wahid staff 11258 2011-01-03 22:09 pkg-
publisher
drwxr-xr-x 2 wahid staff 8 2011-01-15 20:02 text
-rw-rw-r-- 1 wahid staff 0 2011-01-15 21:29 this
```

### Making a script executable

A common real-world case: a shell script created with echo has no execute bit, so running it directly fails with "Permission denied" until chmod adds the execute bit.

```
$ echo "df -hF zfs" > df-zfs.sh
$ ls -l df-zfs.sh
-rw-r--r-- 1 wahid staff 11 2011-01-17 13:01 df-zfs.sh
```

```
$ ./df-zfs.sh
bash: ./df-zfs.sh: Permission denied

$ sh -x df-zfs.sh          # run without changing
permissions, using sh's -x trace mode
$ df -hF zfs
Filesystem             Size  Used  Avail  Use%  Mounted
on
rpool/ROOT/opensolaris-1   20G  7.2G   12G   38%  /

$ chmod 755 df-zfs.sh
$ ls -l df-zfs.sh
-rwxr-xr-x 1 wahid staff 11 2011-01-17 13:01 df-zfs.sh

$ ./df-zfs.sh          # now runs directly, since the
execute bit is set
Filesystem             Size  Used  Avail  Use%  Mounted
on
rpool/ROOT/opensolaris-1   20G  7.2G   12G   38%  /
```

## 2. chgrp — change group ownership

```
$ whatis chgrp
chgrp (1) - change group ownership
chgrp (1) - change file group ownership
```

chgrp changes which group owns a file or directory. Like chmod and chown, you need sufficient privilege — either you already belong to the target group, or you have root access — to make the change.

### Checking current group ownership

```
$ pwd
/export/home/wahid/training/OpenSolaris/pkgdir
$ ls -l
total 13
-rw-r--r-- 1 wahid staff 11258 2011-01-03 22:09 pkg-
publisher
drwxr-xr-x 2 wahid staff 8 2011-01-15 17:36 text
```

### An unprivileged attempt fails as expected

```
$ id
uid=101(wahid) gid=10(staff) groups=10(staff)
$ chgrp root *
chgrp: changing group of `pkg-publisher': Not owner
chgrp: changing group of `text': Not owner
```

wahid only belongs to group staff, so attempting to reassign ownership to group root is rejected — you cannot hand a file to a group you don't belong to without elevated privilege.

### Switching to root and changing group ownership

```
$ su -
Password:
# cd /export/home/wahid/training/OpenSolaris/pkgdir
# chgrp root *
# ls -l
total 13
-rw-r--r-- 1 wahid root 11258 2011-01-03 22:09 pkg-
publisher
```

```
drwxr-xr-x 2 wahid root 8 2011-01-15 17:36 text
```

Note that chgrp without -R is not recursive: the text directory itself changed to group root, but the files inside it did not.

### Recursive group change with -R

```
# chgrp -R root *
# ls -lR text
text:
-rw-r--r-- 1 wahid root  2856 2011-01-03 22:04 pkg-
info.txt
-rw-r--r-- 1 wahid root 475569 2011-01-03 23:38 pkg-
install-amp.txt
-rw-r--r-- 1 wahid root  2358 2011-01-03 22:06 pkg-
list.txt

# chgrp -R staff *          # revert back to the
original group
# ls -l
-rw-r--r-- 1 wahid staff 11258 2011-01-03 22:09 pkg-
publisher
drwxr-xr-x 2 wahid staff 8 2011-01-15 17:36 text
```

# 3. chown — change file owner and group

```
$ whatis chown
chown (1) - change file owner and group
chown (1) - change file ownership
chown (2) - change owner and group of a file
```

chown changes the owning user of a file or directory, and optionally the group at the same time using the owner:group syntax. Like chgrp, changing ownership normally requires root privilege.

## An unprivileged attempt fails

```
$ pwd
/export/home/wahid/training/OpenSolaris/pkgdir
$ id
uid=101(wahid) gid=10(staff) groups=10(staff)
$ ls -l
-rw-r--r-- 1 wahid staff 11258 2011-01-03 22:09 pkg-
publisher
drwxr-xr-x 2 wahid root  8 2011-01-15 20:02 text

$ chown root text
chown: changing ownership of `text': Not owner
```

## As root: changing owner only

```
$ su
Password:
# pwd
/export/home/wahid/training/OpenSolaris/pkgdir
# chown root text
# ls -l
-rw-r--r-- 1 wahid staff 11258 Jan  3 22:09 pkg-
publisher
drwxr-xr-x 2 root  root  8 Jan 15 20:02 text
```

## Changing owner and group together

```
# chown wahid:staff text
# ls -l text
-rw-r--r-- 1 wahid staff  2856 Jan  3 22:04 pkg-info.txt
```

```
-rw-r--r-- 1 wahid staff 475569 Jan  3 23:38 pkg-
install-amp.txt
```

## Recursive ownership change with -R

```
# chown -R root:sys text
# ls -ld text
drwxr-xr-x 2 root sys 8 Jan 15 20:02 text
# ls -l text
-rw-r--r-- 1 root sys  2856 Jan  3 22:04 pkg-info.txt
-rw-r--r-- 1 root sys 475569 Jan  3 23:38 pkg-install-
amp.txt

# chown -R wahid text          # restore ownership
# exit
$ id
uid=101(wahid) gid=10(staff) groups=10(staff)
```

As with chgrp, always double-check with ls -l (or ls -ld for a directory itself, without descending into it) before and after any ownership change — an accidental chown -R on the wrong directory can lock you out of your own files.

# 4. passwd — change login password and password attributes

```
$ whatis passwd
passwd - change login password and password attributes
```

passwd changes a user's login password. Any user can change their own password by simply running passwd with no arguments; changing another user's password, or root's, requires root privilege.

## Changing another user's password as root

```
# id
uid=0(root) gid=0(root)
groups=0(root),1(other),2(bin),3(sys)...

# passwd -r files devuser
New Password:
Re-enter new Password:
passwd: password successfully changed for devuser
```

The -r files option specifies that the change should be written to the local /etc/shadow file rather than a network directory service — on systems using centralized authentication, -r ldap or -r nis would target LDAP or NIS instead, an important distinction to understand once you reach Chapter 36's coverage of LDAP and directory services.

## Preventing password expiration

```
# passwd -x -1 devuser
passwd: password information changed for devuser
```

The -x -1 option disables the maximum password age, so the account is never forced to change its password automatically — useful for service accounts, though generally discouraged for interactive human users under most security policies.

## Verifying the change

```
# grep -i devuser /etc/passwd
```

```
devuser:x:101:10:WahidLutfy:/home/devuser:/bin/bash

# grep -i devuser /etc/shadow
devuser:$6$illustrativeHashOnly$doNotUseInProduction:150
61::::::
```

*Note: the /etc/shadow line above is shown with an illustrative placeholder hash rather than a real one — never publish an actual password hash, even from a training environment, since offline cracking tools can sometimes recover the original password from it.*

# 5. alias — create a shortcut name for a command

```
$ whatis alias
A shorter name for assigning UNIX commands, scripts, and
other programs.
```

alias lets you define a short, memorable name for a longer command or command pipeline. Its exact syntax depends on which shell you're using: Bourne-family shells (sh, ksh, bash) use name='command', while C-family shells (csh, tcsh) use name command with no equals sign and no quotes required around simple commands.

## Bourne/Bash syntax

```
$ ps
PID TTY  TIME CMD
707 pts/3 0:03 bash

$ alias myf='find . -name "*" -print -exec ls -l {} \;'
$ alias myls='ls -ltr'

$ alias
alias myf='find . -name "*" -print -exec ls -l {} \;'
alias myls='ls -ltr'
```

Aliases can wrap any command with any options baked in — including an admin-oriented example specific to the system being managed:

```
$ alias myzl='zoneadm list'
$ myzl
global

$ alias myzl='zoneadm list -p'          # redefine with
an added option
$ myzl
0:global:running:/::native:shared
```

Running the alias myf (equivalent to a recursive ls -lR) confirms the alias behaves exactly like typing out its full definition:

```
$ myf
.
total 1
-rw-r--r-- 1 wahid staff 0 2011-01-09 17:56 newfile
./newfile
-rw-r--r-- 1 wahid staff 0 2011-01-09 17:56 ./newfile
```

## C shell (csh/tcsh) syntax

Switching shells changes both the prompt (to %) and the alias syntax — no equals sign, and the command is typically left unquoted for simple cases:

```
$ /bin/csh
% ps
PID TTY  TIME CMD
1585 pts/3 0:00 csh

% alias myz zoneadm list -cv
% alias mydf df -hF ufs
% alias
myz (zoneadm list -cv)
mydf (df -hF ufs)
```

Aliases are session-local by default in both shell families — they disappear when the shell exits. To make an alias permanent, add the same alias definition to your shell's startup file: ~/.bashrc for bash, or ~/.cshrc for csh/tcsh, and open a new shell (or source the file) for it to take effect.

*Practical daily-workflow commands for file management, permissions, and text processing.*

## alias

**whatis:** *create a shortcut name for a command*

```
$ alias ll='ls -la'
```

**Result:** Defines ll as shorthand; typing ll afterward runs ls -la.

## bc

**whatis:** *an arbitrary precision calculator language*

```
$ echo "5*12" | bc
```

**Result:** 60

## chgrp

**whatis:** *change group ownership of a file*

```
$ chgrp developers report.txt
```

**Result:** Changes the group owner of report.txt to developers.

## chmod

**whatis:** *change file access permissions*

```
$ chmod 755 deploy.sh
```

**Result:** Sets deploy.sh so the owner can read/write/execute and everyone else can read/execute.

## chown

**whatis:** *change file owner and group*

```
$ chown wahid:staff report.txt
```

**Result:** Changes the owner of report.txt to wahid and the group to staff.

## passwd

**whatis:** *change a user's password*

```
$ passwd
```

**Result:** Prompts for the current password, then a new password, and confirms the change.

## cmp

**whatis:** *compare two files byte by byte*

```
$ cmp file1.txt file2.txt
```

**Result:** Reports the first byte and line where the two files differ, or nothing if identical.

## uncompress

**whatis:** *expand .Z compressed files*

```
$ uncompress archive.Z
```

**Result:** Decompresses archive.Z back into archive.

## xterm

**whatis:** *terminal emulator for the X Window System*

```
$ xterm &
```

**Result:** Opens a new terminal window running in the background.

## cp

**whatis:** *copy files and directories*

```
$ cp -r project/ project_backup/
```

**Result:** Recursively copies the project directory and its contents to project_backup.

## cut

**whatis:** *remove sections from each line of a file*

```
$ cut -d',' -f1 data.csv
```

**Result:** Prints only the first comma-separated field from every line of data.csv.

## df

**whatis:** *report file system disk space usage*

```
$ df -h
```

**Result:** Shows disk usage for all mounted filesystems in human-readable form (GB/MB).

## diff

**whatis:** *compare files line by line*

```
$ diff old.txt new.txt
```

**Result:** Shows the lines that differ between old.txt and new.txt, marked with < and >.

## dirname

**whatis:** *strip the last component from a file path*

```
$ dirname /home/wahid/notes.txt
```

**Result:** /home/wahid

## basename

**whatis:** *strip directory and suffix from a file path*

```
$ basename /home/wahid/notes.txt
```

**Result:** notes.txt

## du

**whatis:** *estimate file space usage*

```
$ du -sh /var/log
```

**Result:** Reports the total size of /var/log in human-readable form, e.g. 128M.

## eject

**whatis:** *eject removable media*

```
$ eject /dev/sr0
```

**Result:** Ejects the optical or removable media mounted at /dev/sr0.

## env

**whatis:** *display or run a program in a modified environment*

```
$ env
```

**Result:** Lists every environment variable currently set in the shell session.

## file

**whatis:** *determine file type*

```
$ file report.pdf
```

**Result:** report.pdf: PDF document, version 1.7

## find

**whatis:** *search for files in a directory hierarchy*

```
$ find /home/wahid -name "*.sh"
```

**Result:** Recursively lists every .sh file under /home/wahid.

## finger

**whatis:** *display information about a user*

```
$ finger wahid
```

**Result:** Shows login name, full name, home directory, shell, and last login time for wahid.

## grep

**whatis:** *search text using patterns*

```
$ grep -i "error" syslog
```

**Result:** Prints every line in syslog containing "error", case-insensitively.

## whatis

**whatis:** *display a one-line manual page description*

```
$ whatis ls
```

**Result:** ls (1) - list directory contents

## groups

**whatis:** *print the groups a user belongs to*

```
$ groups wahid
```

**Result:** wahid : wahid sudo developers

## gzip

**whatis:** *compress files*

```
$ gzip access.log
```

**Result:** Compresses access.log into access.log.gz and removes the original.

## egrep

**whatis:** *search text using extended regular expressions*

```
$ egrep "error|fail" syslog
```

**Result:** Prints every line containing either "error" or "fail".

## kill

**whatis:** *terminate a process by PID*

```
$ kill -9 4521
```

**Result:** Forcefully terminates the process with PID 4521.

## last

**whatis:** *show a listing of last logged-in users*

```
$ last -5
```

**Result:** Lists the five most recent login sessions with usernames, terminals, and times.

## link

**whatis:** *create a hard link to a file*

```
$ link report.txt report_copy.txt
```

**Result:** Creates report_copy.txt as a hard link pointing to the same data as report.txt.

## ln

**whatis:** *make links between files*

```
$ ln -s /opt/app/current /opt/app/latest
```

**Result:** Creates a symbolic link named latest pointing to the current directory.

## pgrep

**whatis:** *look up processes by name*

```
$ pgrep -l nginx
```

**Result:** Lists the PIDs and names of all running processes matching "nginx".

## whereis

**whatis:** *locate binary, source, and manual page files*

```
$ whereis python3
```

**Result:** python3: /usr/bin/python3 /usr/share/man/man1/python3.1.gz

## which

**whatis:** *locate a command's executable in PATH*

```
$ which python3
```

**Result:** /usr/bin/python3

# Chapter 4

## Advanced UNIX and Linux Commands

## Chapter 4 Command Deep-Dives

**user/group mgmt • systemctl • pwck/grpck • tar/gzip • systemd • grep family • figlet • dpkg • apt • lsof • expr • journalctl**

*Transcribed from MyWebUniversity.com/cgi-bin/CH4_Col.cgi, matching Chapter 4: Advanced UNIX and Linux Commands. Where the companion site's sandboxed terminal returns real live output (marked clearly below), that output is reproduced exactly. Several administration commands in this chapter (user/group management, systemctl, systemd, dpkg, apt) require root privilege or are blocked in the site's public sandbox for safety — for those, representative Ubuntu 26.04 LTS output is provided instead and clearly labeled as such.*

# 1. useradd, userdel, usermod, groupadd, groupdel, groupmod

```
useradd - create a new user or update default new user
information
userdel - delete a user account and related files
usermod - modify a user account
groupadd - create a new group
groupdel - delete a group
groupmod - modify a group definition on the system
```

> *The companion site's worked example for these six commands was captured on an older Solaris system, and the page's formatting doesn't preserve the actual typed commands cleanly enough to transcribe verbatim. Rather than guess at the original sequence, the walkthrough below demonstrates the same six commands accurately on Ubuntu 26.04 LTS.*

These six commands manage local user and group accounts, and are almost always run with sudo since they modify /etc/passwd, /etc/shadow, /etc/group, and /etc/gshadow.

## Creating a user and a group

```
$ sudo groupadd developers
$ sudo useradd -m -s /bin/bash -G developers newuser
$ id newuser
uid=1002(newuser) gid=1002(newuser)
groups=1002(newuser),1001(developers)
```

-m creates the user's home directory (/home/newuser); -s sets the login shell; -G adds the new account to the developers secondary group at creation time.

## Setting the account's password

```
$ sudo passwd newuser
New password:
Retype new password:
passwd: password updated successfully
```

## Modifying an existing account with usermod

```
$ sudo usermod -aG sudo newuser
```

```
$ groups newuser
newuser : newuser developers sudo
```

-aG (append, supplementary Groups) adds newuser to the sudo group without removing its existing group memberships — always combine -a with -G; using -G alone replaces the account's entire group list.

## Renaming and modifying a group with groupmod

```
$ sudo groupmod -n devs developers
$ getent group devs
devs:x:1001:newuser
```

## Removing a group and a user

```
$ sudo groupdel devs
groupdel: cannot remove the primary group of user
'newuser'
```

groupdel refuses to remove a group that is still another account's primary group — a built-in safety check. Cleaning up the user account first resolves it:

```
$ sudo userdel -r newuser
$ id newuser
id: 'newuser': no such user
```

-r removes the account's home directory and mail spool along with the account itself; without -r, the home directory is left behind.

# 2. systemctl — control the systemd system and service manager

```
$ whatis systemctl
systemctl - Control the systemd system and service
manager
```

> *The site's sandboxed terminal returns "The command is not authorized to run" for systemctl, since it can start, stop, and reconfigure system services — not something a public sandbox can safely allow. The output below is representative of a real Ubuntu 26.04 LTS system.*

## Checking a service's status

```
$ systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded
(/lib/systemd/system/apache2.service; enabled)
     Active: active (running) since Sat 2026-07-25
09:12:03 PDT; 2h 14min ago
   Main PID: 812 (apache2)
      Tasks: 6 (limit: 4665)
     Memory: 22.4M
```

## Starting, stopping, and enabling a service

```
$ sudo systemctl start apache2
$ sudo systemctl stop apache2
$ sudo systemctl enable apache2
Created symlink /etc/systemd/system/multi-
user.target.wants/apache2.service →
/lib/systemd/system/apache2.service.
```

enable configures a service to start automatically at boot; it does not start the service immediately — pair it with start (or use enable --now to do both in one command).

# 3. pwck & grpck — password and group file checkers

```
$ whatis pwck
pwck, grpck - password/group file checkers
```

pwck and grpck verify the internal consistency of the local account database — /etc/passwd and /etc/shadow for pwck, /etc/group and /etc/gshadow for grpck — catching problems like a user whose home directory no longer exists, a missing shell, or duplicate UIDs/GIDs. Both require root access to read the shadow files.

## Running pwck

```
$ sudo pwck
user 'newuser': directory '/home/newuser' does not exist
user 'newuser': no group '1002' found
pwck: no changes
```

Output only appears for accounts with a detected inconsistency — a clean run against a normal Ubuntu system typically prints nothing beyond pwck: no changes.

## Running grpck

```
$ sudo grpck
grpck: no changes
```

Both commands default to checking the system's standard files when run with no arguments, but can also be pointed at alternate files — useful when auditing a copy of the account database rather than the live one, e.g. sudo pwck /tmp/passwd.bak /tmp/shadow.bak.

# 4. tar, gzip, gunzip, zcat — archiving and compression

```
tar (1) - an archiving utility
gzip (1) - compress or expand files
gunzip (1) - compress or expand files
zcat (1) - compress or expand files
```

*The companion site's worked example for these four commands extracted as a very long, merged block of output with the individual typed commands stripped out by the page's formatting. The walkthrough below demonstrates the same operations — archiving, compressing, inspecting, and restoring files — cleanly on Ubuntu 26.04 LTS.*

## Archiving a directory with tar

```
$ tar -czvf project-backup.tar.gz project/
project/
project/main.py
project/README.md
project/data/
project/data/input.csv
```

-c creates a new archive, -z compresses it with gzip, -v lists each file as it's added (verbose), and -f names the output archive file.

## Inspecting an archive without extracting it

```
$ tar -tzvf project-backup.tar.gz
drwxr-xr-x wahid/wahid       0 2026-07-25 10:02 project/
-rw-r--r-- wahid/wahid    1840 2026-07-25 10:01
project/main.py
-rw-r--r-- wahid/wahid     512 2026-07-25 10:01
project/README.md
drwxr-xr-x wahid/wahid       0 2026-07-25 10:02
project/data/
-rw-r--r-- wahid/wahid   38210 2026-07-25 10:02
project/data/input.csv
```

-t lists the archive's table of contents; combined with -z and -v it works on a compressed .tar.gz archive exactly the same way.

### Extracting an archive

```
$ tar -xzvf project-backup.tar.gz -C /tmp/restore/
project/
project/main.py
project/README.md
project/data/
project/data/input.csv
```

### gzip, gunzip, and zcat on a single file

```
$ gzip access.log
$ ls -l access.log.gz
-rw-r--r-- 1 wahid wahid 8213 Jul 25 10:05 access.log.gz

$ zcat access.log.gz | head -3
192.168.1.20 - - [25/Jul/2026:09:00:01] "GET / HTTP/1.1"
200 512
192.168.1.21 - - [25/Jul/2026:09:00:03] "GET /style.css
HTTP/1.1" 200 2048
192.168.1.22 - - [25/Jul/2026:09:00:07] "GET
/favicon.ico HTTP/1.1" 404 0

$ gunzip access.log.gz
$ ls -l access.log
-rw-r--r-- 1 wahid wahid 41230 Jul 25 10:05 access.log
```

zcat lets you read the contents of a .gz file without permanently decompressing it — useful for quickly checking a compressed log with head, tail, or grep.

# 5. systemd — the system and service manager

```
$ whatis systemd
systemd (1) - systemd system and service manager
```

> *systemd itself is the background init system (PID 1), not an interactive command — you manage it through systemctl (Section 2 above) and journalctl (Section 14 below), which is why the site's sandbox blocks direct systemctl status calls under the systemd example page too.*

systemd replaced the older SysV init system on Ubuntu starting with 15.04, and is responsible for booting the system, starting and supervising services (units), managing mount points, sockets, and timers, and handling dependency ordering between them. Every service you start with systemctl is a systemd unit file, typically found under /lib/systemd/system/ or /etc/systemd/system/.

## Viewing what systemd is currently managing

```
$ systemctl list-units --type=service --state=running
UNIT                  LOAD   ACTIVE SUB     DESCRIPTION
apache2.service       loaded active running The Apache
HTTP Server
ssh.service           loaded active running OpenBSD
Secure Shell server
systemd-journald.service loaded active running Journal
Service
```

# 6–8. grep, egrep, fgrep — pattern matching

```
grep (1) - print lines matching a pattern
egrep (1) - print lines matching a pattern (extended
regex)
fgrep (1) - print lines matching a pattern (fixed
strings)
```

*The companion site's pages for grep, egrep, and fgrep link only to their manual pages and don't include worked examples. The three examples below fill that gap, demonstrating the practical difference between the three variants on the same input.*

## grep: basic pattern matching

```
$ grep "error" /var/log/syslog
Jul 25 09:14:02 web01 kernel: error: usb 1-2: device
descriptor read failed
Jul 25 09:41:10 web01 apache2: error: [client
192.168.1.30] File does not exist
```

## egrep: extended regular expressions (alternation, quantifiers)

```
$ egrep "error|fail(ed)?" /var/log/syslog
Jul 25 09:14:02 web01 kernel: error: usb 1-2: device
descriptor read failed
Jul 25 09:41:10 web01 apache2: error: [client
192.168.1.30] File does not exist
Jul 25 10:02:55 web01 sshd: Failed password for invalid
user test from 203.0.113.5
```

egrep is equivalent to grep -E — the | (alternation) and (...)? (optional group) syntax used here only works in extended mode; plain grep would need every special character escaped with a backslash.

## fgrep: fixed-string matching (no regex interpretation)

```
$ fgrep "192.168.1.30" /var/log/syslog
Jul 25 09:41:10 web01 apache2: error: [client
192.168.1.30] File does not exist
```

fgrep is equivalent to grep -F — useful when your search text itself contains regex metacharacters (like the dots in an IP address) that you want matched literally rather than interpreted.

# 9. figlet — render text as large ASCII-art letters

```
$ whatis figlet
No manual entry for figlet (not installed by default)
```

figlet isn't installed on Ubuntu by default, which makes it a good real-world example of the full install-and-verify workflow — exactly what the companion site's live session walks through.

## Discovering the command isn't installed

```
$ figlet Ubuntu
Command 'figlet' not found, but can be installed with:
sudo apt install figlet
```

## Trying to install without sudo fails

```
$ apt install figlet
E: Could not open lock file /var/lib/dpkg/lock-frontend
- open (13: Permission denied)
E: Unable to acquire the dpkg frontend lock
(/var/lib/dpkg/lock-frontend), are you root?
```

## Installing with sudo

```
$ sudo apt install figlet
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed: figlet
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
Need to get 133 kB of archives.
...
Setting up figlet (2.2.5-3) ...
```

## Verifying and using it

```
$ which figlet
/usr/bin/figlet

$ figlet Ubuntu
```

```
 _   _ _                    _
| | | | |__   _   _ _ __ | |_ _   _
| | | | '_ \| | | | '_ \ __| | | |
| |_| | |_) | |_| | | | | |_| |_| |
 \___/|_.__/ \__,_|_| |_|\__|\__,_|
```

figlist (installed alongside figlet) shows every available font, which you can select with figlet -f fontname "your text".

# 10. dpkg — package manager for Debian-based systems

```
$ whatis dpkg
dpkg (1) - package manager for Debian
```

> *The site's sandbox returns "not authorized to run" for dpkg, since it directly queries and can modify the installed-package database. The output below is representative of a real Ubuntu 26.04 LTS system.*

## Checking whether a package is installed

```
$ dpkg -l openssh-client
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-
inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err:
uppercase=bad)
||/ Name           Version         Architecture
Description
+++-==============-===============-============-
=======================
ii  openssh-client 1:9.9p1-3ubuntu1 amd64        secure
shell (SSH) client
```

## Listing the files a package installed

```
$ dpkg -L openssh-client
/.
/usr
/usr/bin
/usr/bin/ssh
/usr/bin/scp
/usr/bin/sftp
/usr/share/man/man1/ssh.1.gz
```

# 11. apt — high-level package management front end

```
apt (8) - command-line interface
apt-get (8) - APT package handling utility
sources.list (5) - list of configured APT data sources
apt.conf (5) - configuration file for APT
apt-config (8) - APT configuration query program
```

> *The site's sandbox blocks all apt subcommands, including read-only ones like search, since apt shares the same system package lock as install/remove. The output below is representative of a real Ubuntu 26.04 LTS system.*

## Searching for a package

```
$ apt search openssh
openssh-client/noble-updates 1:9.9p1-3ubuntu1 amd64
  secure shell (SSH) client, for secure access to remote
machines

openssh-server/noble-updates 1:9.9p1-3ubuntu1 amd64
  secure shell (SSH) server, for secure access from
remote machines
```

## Showing details for one package

```
$ apt show openssh-client
Package: openssh-client
Version: 1:9.9p1-3ubuntu1
Priority: standard
Section: net
Installed-Size: 4,241 kB
Depends: libc6, libssl3, libselinux1
Description: secure shell (SSH) client, for secure
access to remote machines
```

# 12. lsof — list open files

```
$ whatis lsof
lsof (8) - list open files
```

On UNIX and Linux, nearly everything — including open network sockets — is represented as a file, which makes lsof one of the most useful diagnostic tools for finding what's using a port, which process has a file locked, or what a specific user or process currently has open.

## Finding what's listening on a port

```
$ sudo lsof -i :443
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nginx    812  root    6u  IPv4  28190      0t0  TCP
*:https (LISTEN)
nginx    813  www-data 6u IPv4  28190      0t0  TCP
*:https (LISTEN)
```

## Listing every file a specific process has open

```
$ lsof -p 812
COMMAND PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
nginx   812 root  cwd    DIR    8,1     4096 262145 /
nginx   812 root  txt    REG    8,1  1180024 393228
/usr/sbin/nginx
nginx   812 root    6u  IPv4  28190      0t0    TCP
*:https (LISTEN)
```

## 13. expr — evaluate expressions

```
$ whatis expr
expr - evaluate expressions
```

expr evaluates a simple arithmetic or string expression and prints the result — still commonly used inside older Bourne-shell scripts for basic math before $(( )) arithmetic expansion became standard. All four worked examples below are the live, real output from the companion site's online terminal.

```
$ expr 5 + 5
10

$ expr 5 - 2
3

$ expr 5 \* 5
25

$ expr 10 / 2
5
```

Note the backslash before the multiplication asterisk (5 * 5) — without it, the shell would try to expand * as a filename glob before expr ever sees it.

# 14. journalctl — query the systemd journal

```
$ whatis journalctl
journalctl (1) - Query the systemd journal
```

journalctl reads the centralized, binary systemd journal that collects log output from every systemd-managed service, the kernel, and boot messages, replacing the older practice of grep-ing through scattered plain-text log files in /var/log. Both examples below are the live, real output from the companion site's online terminal, confirming this book's Ubuntu 26.04 LTS environment is running systemd 259.

**$ journalctl --version**

```
systemd 259 (259.5-0ubuntu3)
+PAM +AUDIT +SELINUX +APPARMOR +IMA +IPE +SMACK +SECCOMP +GCRYPT -
GNUTLS +OPENSSL
+ACL +BLKID +CURL +ELFUTILS +FIDO2 +IDN2 -IDN +KMOD +LIBCRYPTSETUP
+LIBCRYPTSETUP_PLUGINS +LIBFDISK +PCRE2 +PWQUALITY +P11KIT
+QRENCODE +TPM2
+BZIP2 +LZ4 +XZ +ZLIB +ZSTD +BPF_FRAMEWORK +BTF -XKBCOMMON -UTMP
+SYSVINIT +LIBARCHIVE
```

**$ journalctl --help  (abridged)**

```
journalctl [OPTIONS...] [MATCHES...]

Query the journal.

Source Options:
     --system                Show the system journal
     --user                  Show the user journal for the current
user
  -m --merge                 Show entries from all available
journals

Filtering Options:
  -S --since=DATE            Show entries not older than the
specified date
  -U --until=DATE            Show entries not newer than the
specified date
  -b --boot[=ID]             Show current boot or the specified
boot
  -u --unit=UNIT             Show logs from the specified unit
  -p --priority=RANGE        Show entries within the specified
priority range
```

```
  -k --dmesg                Show kernel message log from the
current boot

Output Control Options:
  -o --output=STRING        Change journal output mode (short,
json, cat, ...)
  -n --lines[=[+]INTEGER]   Number of journal entries to show
  -r --reverse              Show the newest entries first
  -f --follow               Follow the journal

See the journalctl(1) man page for details.
```

In everyday use, journalctl -u apache2 --since today or journalctl -f (to follow new entries live, like tail -f) are the two invocations you'll reach for most often.

*Power tools for system administration, package management, performance monitoring, and development.*

## useradd

**whatis:** *create a new user account*

```
$ sudo useradd -m -s /bin/bash newdev
```

**Result:** Creates user newdev with a home directory and bash as the default shell.

## userdel

**whatis:** *delete a user account*

```
$ sudo userdel -r newdev
```

**Result:** Removes user newdev and deletes their home directory.

## usermod

**whatis:** *modify an existing user account*

```
$ sudo usermod -aG sudo newdev
```

**Result:** Adds newdev to the sudo group without removing existing group memberships.

## groupadd

**whatis:** *create a new group*

```
$ sudo groupadd developers
```

**Result:** Creates a new group named developers.

## groupdel

**whatis:** *delete a group*

```
$ sudo groupdel developers
```

**Result:** Removes the developers group from the system.

## groupmod

**whatis:** *modify a group definition*

```
$ sudo groupmod -n devs developers
```

**Result:** Renames the developers group to devs.

## users

**whatis:** *print the usernames of currently logged-in users*

```
$ users
```

**Result:** wahid wahid root

## systemctl

**whatis:** *control the systemd system and service manager*

```
$ sudo systemctl restart nginx
```

**Result:** Restarts the nginx service and returns to the prompt when complete.

## catman

**whatis:** *create/update the pre-formatted manual page cache*

```
$ sudo catman
```

**Result:** Rebuilds the cached, pre-formatted versions of all manual pages for faster access.

## pwck

**whatis:** *verify the integrity of password files*

```
$ sudo pwck
```

**Result:** Checks /etc/passwd and /etc/shadow for consistency and reports any anomalies.

## grpck

**whatis:** *verify the integrity of group files*

```
$ sudo grpck
```

**Result:** Checks /etc/group and /etc/gshadow for consistency and reports any anomalies.

## type

**whatis:** *describe how a command name would be interpreted*

```
$ type ls
```

**Result:** ls is aliased to `ls --color=auto'

## ext4

**whatis:** *the fourth extended filesystem*

```
$ mkfs.ext4 /dev/sdb1
```

**Result:** Formats partition /dev/sdb1 with the ext4 filesystem.

## mkfs

**whatis:** *build a Linux filesystem on a device*

```
$ sudo mkfs -t ext4 /dev/sdb1
```

**Result:** Creates a new ext4 filesystem on /dev/sdb1, erasing existing data.

## wget

**whatis:** *non-interactive network file downloader*

```
$ wget https://mywebuniversity.com/toc1.html
```

**Result:** Downloads toc1.html from the site and saves it to the current directory.

## journalctl

**whatis:** *query the systemd journal*

```
$ journalctl -u nginx --since today
```

**Result:** Shows all systemd journal log entries for the nginx service recorded today.

## Intro

**whatis:** *introduction to user commands*

```
$ man intro
```

**Result:** Opens the introductory manual page explaining Linux user command conventions.

## lsattr

**whatis:** *list file attributes on a Linux filesystem*

```
$ lsattr notes.txt
```

**Result:** ----i--------e-- notes.txt

## fstab

**whatis:** *static information about filesystems*

```
$ cat /etc/fstab
```

**Result:** Lists the filesystems that are automatically mounted at boot, with device, mount point, and options.

## mount

**whatis:** *mount a filesystem*

```
$ sudo mount /dev/sdb1 /mnt/data
```

**Result:** Mounts the /dev/sdb1 partition at /mnt/data.

## pgrep

**whatis:** *look up processes by name*

```
$ pgrep -l ssh
```

**Result:** Lists PIDs and names of all processes matching "ssh".

## umount

**whatis:** *unmount a filesystem*

```
$ sudo umount /mnt/data
```

**Result:** Unmounts the filesystem currently mounted at /mnt/data.

## pkill

**whatis:** *signal processes by name*

```
$ pkill -9 firefox
```

**Result:** Forcefully terminates all processes named firefox.

## pmap

**whatis:** *report the memory map of a process*

```
$ pmap 4521
```

**Result:** Lists the memory segments used by process 4521, with sizes and permissions.

## tar / gzip / gunzip

**whatis:** *archive and compress/decompress files*

```
$ tar -czvf backup.tar.gz project/
```

**Result:** Creates a compressed archive backup.tar.gz containing the project directory.

## pstree

**whatis:** *display running processes as a tree*

```
$ pstree -p
```

**Result:** Shows all processes in a hierarchical parent-child tree, with PIDs.

## grep

**whatis:** *search text using patterns*

```
$ grep "root" /etc/passwd
```

**Result:** Prints the line in /etc/passwd describing the root account.

## egrep

**whatis:** *search using extended regular expressions*

```
$ egrep "^root|^wahid" /etc/passwd
```

**Result:** Prints lines starting with root or wahid.

## fgrep

**whatis:** *search for fixed strings (no regex)*

```
$ fgrep "TODO" *.sh
```

**Result:** Prints every line containing the literal string TODO across all .sh files.

## set

**whatis:** *set or display shell options and variables*

```
$ set -x
```

**Result:** Enables shell debug mode, echoing every command before it runs.

## setenv

**whatis:** *set an environment variable (csh/tcsh)*

```
$ setenv EDITOR vim
```

**Result:** Sets the EDITOR environment variable to vim in a csh/tcsh shell.

## services

**whatis:** *list network services (/etc/services)*

```
$ cat /etc/services | grep http
```

**Result:** Shows the port/protocol entries defined for HTTP-related services.

## env

**whatis:** *display or run with a modified environment*

```
$ env PATH=/usr/bin ls
```

**Result:** Runs ls using only /usr/bin in its PATH.

## curl

**whatis:** *transfer data from or to a server*

```
$ curl -I https://mywebuniversity.com
```

**Result:** Displays the HTTP response headers returned by the server.

## figlet

**whatis:** *render text as large ASCII-art letters*

```
$ figlet Ubuntu
```

**Result:** Prints "Ubuntu" as large block letters built from ASCII characters.

## docker

**whatis:** *build, run, and manage containers*

```
$ docker run -d -p 8080:80 nginx
```

**Result:** Starts an nginx container in the background, mapping port 8080 to container port 80.

## systemd

**whatis:** *the system and service manager*

```
$ systemctl list-units --type=service
```

**Result:** Lists all currently loaded systemd services and their states.

## ufw

**whatis:** *uncomplicated firewall management*

```
$ sudo ufw allow 22/tcp
```

**Result:** Adds a firewall rule permitting inbound TCP traffic on port 22 (SSH).

## apt

**whatis:** *high-level package management front end*

```
$ sudo apt install htop
```

**Result:** Downloads and installs the htop package along with its dependencies.

## apt-get

**whatis:** *APT package handling utility*

```
$ sudo apt-get update
```

**Result:** Refreshes the local package index from all configured repositories.

## apt-cache

**whatis:** *query the APT package cache*

```
$ apt-cache search linux-tools
```

**Result:** Lists all packages whose name or description matches “linux-tools”.

## sources.list

**whatis:** *APT repository configuration file*

```
$ cat /etc/apt/sources.list
```

**Result:** Displays the list of package repository URLs APT uses for updates and installs.

## apt.conf

**whatis:** *APT configuration file*

```
$ cat /etc/apt/apt.conf
```

**Result:** Displays custom APT behavior settings, such as proxy or retry configuration.

## apt-config

**whatis:** *query APT's internal configuration*

```
$ apt-config dump | head
```

**Result:** Prints the resolved APT configuration variables currently in effect.

## stat

**whatis:** *display detailed file or filesystem status*

```
$ stat report.txt
```

**Result:** Shows size, permissions, owner, and access/modify/change timestamps for report.txt.

## pldd

**whatis:** *list dynamic shared objects loaded by a process*

```
$ pldd 4521
```

**Result:** Lists every shared library currently loaded by process 4521.

## pwdx

**whatis:** *print the current working directory of a process*

```
$ pwdx 4521
```

**Result:** 4521: /home/wahid/projects

## hostnamectl

**whatis:** *control the system hostname*

```
$ hostnamectl set-hostname web01
```

**Result:** Sets the system's hostname to web01, persisting across reboots.

## tar

**whatis:** *archive files*

```
$ tar -xzvf backup.tar.gz
```

**Result:** Extracts the contents of backup.tar.gz into the current directory.

## ldd

**whatis:** *print shared library dependencies*

```
$ ldd /usr/bin/nginx
```

**Result:** Lists every shared library that the nginx binary depends on, with resolved paths.

## strace

**whatis:** *trace system calls made by a program*

```
$ strace -c ls
```

**Result:** Runs ls while tracing and summarizing every system call it makes.

## core

**whatis:** *the memory image of a crashed process*

```
$ ls core.*
```

**Result:** Lists core dump files produced when a program crashed.

## adb

**whatis:** *Android Debug Bridge (device communication)*

```
$ adb devices
```

**Result:** Lists connected Android devices and their serial numbers.

## gcc

**whatis:** *GNU C compiler*

```
$ gcc -o hello hello.c
```

**Result:** Compiles hello.c into an executable named hello.

## swapon

**whatis:** *enable devices/files for paging and swapping*

```
$ sudo swapon /swapfile
```

**Result:** Activates /swapfile as additional virtual memory (swap space).

## gdb

**whatis:** *the GNU debugger*

```
$ gdb ./hello
```

**Result:** Launches the interactive debugger against the hello executable.

## dd

**whatis:** *convert and copy files at the block level*

```
$ dd if=/dev/zero of=testfile bs=1M count=100
```

**Result:** Creates a 100MB file named testfile filled with zero bytes.

## dpkg

**whatis:** *package manager for Debian-based systems*

```
$ sudo dpkg -i package.deb
```

**Result:** Installs the local package.deb file directly.

## gunzip

**whatis:** *decompress .gz files*

```
$ gunzip access.log.gz
```

**Result:** Decompresses access.log.gz back into access.log.

## syslog

**whatis:** *the system logging facility/file*

```
$ tail -f /var/log/syslog
```

**Result:** Continuously displays new lines appended to the system log in real time.

## htop

**whatis:** *interactive process viewer*

```
$ htop
```

**Result:** Opens a color, interactive, real-time view of running processes and resource usage.

## modinfo

**whatis:** *show information about a kernel module*

```
$ modinfo e1000e
```

**Result:** Displays the description, author, license, and parameters of the e1000e network driver module.

## snap

**whatis:** *install and manage Snap packages*

```
$ sudo snap install code --classic
```

**Result:** Installs the Visual Studio Code snap package with classic confinement.

## nm

**whatis:** *list symbols from object files*

```
$ nm hello.o
```

**Result:** Lists all defined and undefined symbols in the hello.o object file.

## lsmod

**whatis:** *show loaded kernel modules*

```
$ lsmod
```

**Result:** Lists every kernel module currently loaded, with size and dependency count.

## lshw

**whatis:** *list detailed hardware configuration*

```
$ sudo lshw -short
```

**Result:** Prints a summarized inventory of CPU, memory, storage, and network hardware.

## netstat

**whatis:** *print network connections and statistics*

```
$ netstat -tulnp
```

**Result:** Lists all listening TCP/UDP ports along with the process using each one.

## iostat

**whatis:** *report CPU and I/O device statistics*

```
$ iostat -x 2
```

**Result:** Shows extended disk I/O statistics, refreshing every 2 seconds.

## vmstat

**whatis:** *report virtual memory statistics*

```
$ vmstat 2 5
```

**Result:** Reports memory, swap, and CPU statistics five times, two seconds apart.

## sar

**whatis:** *collect and report system activity*

```
$ sar -u 2 5
```

**Result:** Reports CPU utilization five times, two seconds apart.

## cron

**whatis:** *the time-based job scheduler daemon*

```
$ crontab -e
```

**Result:** Opens the current user's crontab file for editing scheduled jobs.

## free

**whatis:** *display memory usage*

```
$ free -h
```

**Result:** Shows total, used, and available RAM and swap in human-readable units.

## mpstat

**whatis:** *report per-processor statistics*

```
$ mpstat -P ALL 2
```

**Result:** Reports utilization for every CPU core, refreshing every 2 seconds.

## time

**whatis:** *measure how long a command takes to run*

```
$ time ls -R /
```

**Result:** Runs the recursive listing and reports real, user, and system execution time.

## nohup

**whatis:** *run a command immune to hangups*

```
$ nohup ./backup.sh &
```

**Result:** Runs backup.sh in the background so it keeps running after the terminal closes.

## exec

**whatis:** *replace the shell with a given command*

```
$ exec bash
```

**Result:** Replaces the current shell process with a fresh bash session.

## ncal

**whatis:** *displays a calendar (alternate layout)*

```
$ ncal 2026
```

**Result:** Displays a compact calendar for 2026 with weeks as columns.

## gcore

**whatis:** *generate a core dump of a running process*

```
$ gcore 4521
```

**Result:** Creates a core dump file capturing the memory state of process 4521.

## rpm

**whatis:** *package manager for RPM-based systems*

```
$ rpm -qa | grep kernel
```

**Result:** Lists all installed RPM packages whose name contains "kernel".

## lsof

**whatis:** *list open files and the processes using them*

```
$ lsof -i :8080
```

**Result:** Lists every process currently holding TCP/UDP port 8080 open.

## expr

**whatis:** *evaluate expressions*

```
$ expr 5 + 7
```

**Result:** 12

## screen

**whatis:** *terminal multiplexer for persistent sessions*

```
$ screen -S build
```

**Result:** Starts a new named terminal session called build that persists after disconnect.

## elf

**whatis:** *executable and linkable format (binaries)*

```
$ readelf -h /bin/ls
```

**Result:** Displays the ELF header information for the ls binary, including architecture and entry point.

## tcpdump

**whatis:** *capture and display network packets*

```
$ sudo tcpdump -i eth0 port 443
```

**Result:** Captures and displays live HTTPS traffic on interface eth0.

## ld

**whatis:** *the GNU linker*

```
$ ld -o hello hello.o
```

**Result:** Links the hello.o object file into an executable named hello.

## as

**whatis:** *the GNU assembler*

```
$ as -o hello.o hello.s
```

**Result:** Assembles hello.s into the object file hello.o.

## make

**whatis:** *build automation tool*

```
$ make install
```

**Result:** Runs the install target defined in the local Makefile.

# Chapter 5

# UNIX and Linux Shell Scripting

*Worked shell-script examples in Bourne shell, BASH, and C-shell, with their expected output.*

## foreach.csh

*A C-shell (csh) script demonstrating the foreach loop, which iterates over a list of values.*

```
Script:
#!/bin/csh
foreach color (red green blue)
    echo "Color: $color"
end
```

```
Output:
Color: red
Color: green
Color: blue
```

## if-args.sh

*A Bourne-shell script that compares two numeric command-line arguments and reports which is larger.*

```
Script:
#!/bin/sh
if [ $# -ne 2 ]; then
    echo "Usage: $0 num1 num2"
    exit 1
fi
if [ "$1" -gt "$2" ]; then
    echo "$1 is greater than $2"
elif [ "$1" -lt "$2" ]; then
    echo "$1 is less than $2"
else
    echo "$1 is equal to $2"
fi
```

```
Output:
$ ./if-args.sh 100 200
100 is less than 200
```

## if-smaller-number.sh

*A script that reads two numbers and prints the smaller one, illustrating a simple comparison pattern.*

```
Script:
#!/bin/sh
echo "Enter first number:"
read a
echo "Enter second number:"
read b
if [ "$a" -lt "$b" ]; then
    echo "Smaller number is $a"
else
    echo "Smaller number is $b"
fi
```

```
Output:
Enter first number:
15
Enter second number:
42
Smaller number is 15
```

## if.csh

*A C-shell script showing basic if/else conditional syntax used in csh scripting.*

```
Script:
#!/bin/csh
set count = 5
if ($count > 0) then
    echo "Count is positive"
else
    echo "Count is zero or negative"
endif
```

```
Output:
Count is positive
```

## if-statement.sh

*A Bourne-shell script demonstrating a single if statement testing whether a file exists.*

```
Script:
#!/bin/sh
FILE="/etc/hostname"
if [ -f "$FILE" ]; then
    echo "$FILE exists."
else
    echo "$FILE not found."
fi
```

```
Output:
/etc/hostname exists.
```

## update-str1-str2.sh

*A BASH script that performs string substitution, replacing one substring with another using parameter expansion.*

```
Script:
#!/bin/bash
str1="Learning Linux is fun"
str2=${str1/Linux/Ubuntu}
echo "Original: $str1"
echo "Updated:  $str2"
```

```
Output:
Original: Learning Linux is fun
Updated:  Learning Ubuntu is fun
```

# Chapters 6–43 — Draft Content

*Written for this manuscript using standard, accurate technical references, matching the chapter list and scope from MyWebUniversity.com/toc1.html. Review commands and version numbers against your live environment before publishing.*

# Chapter 6

## UNIX and Linux Web Servers

*Web fundamentals and server concepts*

The most common production stack on Linux is Apache, MySQL/MariaDB or PostgreSQL, and PHP, Python, or Perl — commonly abbreviated LAMP (Linux, Apache, MySQL, PHP). NGINX has since overtaken Apache as the most widely deployed web server, especially for high-concurrency workloads and as a reverse proxy in front of application servers. Both are covered in production depth in Chapter 41; this chapter introduces the concepts you need before getting there: what a web server actually does (accepts HTTP requests, maps them to files or application processes, and returns HTTP responses), the request/response cycle, virtual hosts (serving multiple domains from one server), and the historical alternatives you may still encounter in legacy environments, such as Netscape Enterprise Server (later Sun ONE / iPlanet) and Apache Tomcat for Java servlet applications.

### 6.1 Installing and Starting Apache on Ubuntu

```
sudo apt update
sudo apt install apache2
sudo systemctl enable --now apache2
sudo systemctl status apache2
```

Once running, the default Apache welcome page is served at http://localhost/ (or your server's IP address). Site configuration files live under /etc/apache2/sites-available/, and the a2ensite / a2dissite commands enable or disable individual virtual host configurations without editing the main config file.

### 6.2 A Minimal Virtual Host

```
<VirtualHost *:80>
    ServerName example.com
    DocumentRoot /var/www/example
    ErrorLog ${APACHE_LOG_DIR}/example-error.log
```

```
    CustomLog ${APACHE_LOG_DIR}/example-access.log
combined
</VirtualHost>
```

See Chapter 41 for reverse proxying, SSL/TLS with Let's Encrypt, and NGINX-specific configuration.

# Chapter 7

## Search for Linux Manual Pages Dynamically

*Interactive man-page search*

Every command in this book has a manual page, and Linux gives you three complementary tools to search them without already knowing the exact command name.

### 7.1 man, whatis, and apropos

- man <command> — opens the full manual page for a command you already know by name.
- whatis <command> — prints just the one-line description from the man page's NAME section (this is exactly the format used throughout Chapters 2–4's command tables).
- apropos <keyword> — searches the short descriptions of every installed man page for a keyword, which is how you find a command when you only know what you want to do, not its name.

```
$ apropos "list directory"
ls (1)          - list directory contents
dir (1)         - list directory contents
vdir (1)        - list directory contents in long format
```

Both whatis and apropos depend on a search index (the "whatis database") that is normally rebuilt automatically by a nightly cron job. If a newly installed package's man pages aren't showing up in apropos, rebuild the index manually:

```
sudo mandb
```

MyWebUniversity.com's dynamic PDF generator (Chapter 12) builds on this same idea, letting you type any command name into a web form and receive an instantly generated, searchable PDF of its manual page.

# Chapter 8

## Linux Ubuntu Manual Pages in PDF Format

*Browse PDFs by section*

Manual pages are grouped into numbered sections by convention — Section 1 for user commands, Section 2 for system calls, Section 3 for library functions, Section 5 for file formats, Section 8 for administration commands, and so on. MyWebUniversity.com pre-generates PDF versions of thousands of Ubuntu 26.04 LTS manual pages, organized by these sections, so you can browse or download them for offline reference without a terminal at all.

### 8.1 Two Representative Examples

Rather than reproduce thousands of individual PDF links here, two representative examples illustrate the pattern every page in the archive follows:

- bash(1) — the Bash shell's own manual page, over 5,000 lines in its complete form, covering every builtin, expansion rule, and invocation option; an essential reference once you're past Chapter 5's scripting introduction.
- crontab(5) — the file-format manual page describing the five-field time specification used by cron (Chapter 4), a good example of a Section 5 (file formats) page as opposed to a Section 1 (commands) page.

The full, continuously updated index is browsable at MyWebUniversity.com/Man_PDF/index.html.

# Chapter 9

## The C and C++ Include Header Files

*/usr/include references*

Every C or C++ program begins with #include directives that pull in declarations for standard library functions. Understanding what lives in each header saves you from guessing. The most commonly used C standard headers, all found under /usr/include on Ubuntu, are:

- stdio.h — standard input/output: printf, scanf, fopen, fclose, fprintf.
- stdlib.h — general utilities: malloc, free, exit, atoi, rand, qsort.
- string.h — string and memory manipulation: strlen, strcpy, strcmp, memcpy, memset.
- math.h — floating-point math: sqrt, pow, sin, cos, floor, ceil (link with -lm).
- ctype.h — character classification and conversion: isalpha, isdigit, toupper, tolower.
- stdint.h — fixed-width integer types: int32_t, uint64_t, and friends, for portable code.
- unistd.h — POSIX operating system API: fork, exec, read, write, close, sleep.
- time.h — date and time functions: time, difftime, strftime, localtime.

C++ wraps most of these in namespaced equivalents prefixed with c (e.g. <cstdio>, <cstring>) and adds its own extensive standard library headers — <iostream>, <vector>, <string>, <map>, <algorithm>, and <memory> among the most frequently used, all covered with working examples in Chapter 23.

### 9.1 Finding What's in a Header

```
man 3 printf          # Section 3: library functions
grep -r 'define' /usr/include/limits.h
```

man 3 <function> is the fastest way to check an exact function signature and required header without leaving the terminal.

# Chapter 10

## The Linux Rocky Release 9.2 Manual Pages

*Rocky Linux documentation in PDF*

Rocky Linux is a community-driven, downstream rebuild of Red Hat Enterprise Linux, created in 2021 after CentOS shifted from being a stable RHEL rebuild to a rolling-release upstream (CentOS Stream). Rocky Linux 9.2 is RPM-based rather than Debian/APT-based like Ubuntu, which means package management commands differ (dnf/yum instead of apt), but the vast majority of Section 1 user commands you learned in Chapters 2–4 behave identically, since they come from the same upstream GNU/Linux utilities.

### 10.1 Key Differences from Ubuntu

- Package manager: dnf install httpd (Rocky) vs. apt install apache2 (Ubuntu).
- Default firewall tool: firewalld (Rocky) vs. ufw (Ubuntu).
- Filesystem layout mostly follows the same Filesystem Hierarchy Standard (FHS) on both.

MyWebUniversity.com's Rocky 9.2 manual page PDF archive lets you cross-reference any command's Rocky-specific man page directly against its Ubuntu equivalent.

# Chapter 11

# The Red Hat Enterprise Linux Release 9.2 Manual Pages

*RHEL documentation in PDF*

Red Hat Enterprise Linux (RHEL) is the commercially supported distribution that Rocky Linux (Chapter 10) rebuilds from source. RHEL is the most widely deployed enterprise Linux distribution, and its subscription includes vendor support, certified hardware compatibility, and long support lifecycles (typically 10 years per major version) — a common requirement in regulated industries such as finance, healthcare, and government.

## 11.1 What to Expect in the RHEL Archive

The RHEL 9.2 manual page PDFs mirror the Rocky 9.2 archive almost page-for-page, since both trace back to the same upstream source RPMs. The practical value of having both side by side is verifying vendor-specific behavior — for example, RHEL's subscription-manager and registration tooling has no Rocky equivalent, since Rocky has no subscription model.

# Chapter 12

# Dynamic Generation of Linux Ubuntu Manual Pages in PDF Format

*Generate PDFs on demand*

Rather than only browsing a static archive, MyWebUniversity.com provides a dynamic tool: type any installed command name into a web form, and the server runs man on that command live and converts the formatted output into a downloadable PDF on the spot. This is useful for commands installed by packages that aren't part of the pre-built archive — anything you've apt install'd yourself, for instance.

## 12.1 The Same Idea, Locally

You can reproduce this workflow yourself on any Ubuntu system without a browser at all, which is worth knowing for offline or scripted documentation generation:

```
man -t bash | ps2pdf - bash-manual.pdf
```

The -t flag tells man to format the page for troff/PostScript output instead of the terminal, and ps2pdf converts that PostScript stream into a PDF you can archive or print.

# Chapter 13

## The Linux Ubuntu Manual Pages in PDF Format

*Browse generated PDFs*

This chapter's companion page is the browsable output of Chapter 12's generator — a continuously growing library of Ubuntu-specific manual pages already converted to PDF, so returning visitors don't need to regenerate pages that have already been requested once. Two pages worth knowing well as you progress through this book:

- systemctl(1) — covers every subcommand for managing systemd services, referenced throughout Chapters 4, 6, and 41.
- sshd_config(5) — the SSH daemon's configuration file format, essential once you start managing remote servers (Chapter 43).

# Chapter 14

## Linux Ubuntu BASH and Microsoft PowerShell Integration (pwsh)

*Cross-shell integration guide*

PowerShell (pwsh) is Microsoft's cross-platform shell and scripting language, and Microsoft has published an official .deb package for Ubuntu since PowerShell 6. Having it available alongside Bash is genuinely useful in mixed Windows/Linux environments, since it can query Windows Active Directory, Azure, and Microsoft 365 from a Linux box, and its object-based pipeline (as opposed to Bash's text-based pipeline) is worth understanding even if Bash remains your daily driver.

### 14.1 Installing PowerShell on Ubuntu 26.04 LTS

```
sudo apt update
sudo apt install -y wget apt-transport-https software-
properties-common
wget -q
https://packages.microsoft.com/config/ubuntu/26.04/packa
ges-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt update && sudo apt install -y powershell
pwsh
```

### 14.2 Bash and PowerShell Command Equivalents

- ls -la → Get-ChildItem -Force (alias: gci, or ls, dir also work in pwsh)
- cat file.txt → Get-Content file.txt (alias: gc, cat)
- grep "error" file.log → Select-String "error" file.log
- ps aux → Get-Process (alias: ps, gps)
- export VAR=value → $env:VAR = "value"

A companion PDF walking through a longer side-by-side command reference is available on MyWebUniversity.com.

# Chapter 15

## Learn Linux Ubuntu Online

*Practice in the web interface*

What makes MyWebUniversity.com's approach to teaching Linux distinctive is a browser-based Linux terminal: a live, sandboxed shell you can practice in directly from the book's companion site, with no local installation, virtual machine, or dual-boot required. It runs the exact commands covered in Chapters 2–5, so you can follow along with each example in this book and immediately try variations of your own.

### 15.1 Getting the Most Out of the Online Terminal

- Type the book's examples first, verbatim, to confirm you get the same output shown in each chapter.
- Then deliberately break things — typo a flag, point a command at a file that doesn't exist — and read the resulting error message carefully. Understanding error messages is as valuable a skill as memorizing correct syntax.
- Use the online terminal's session to practice piping several of Chapter 3 and 4's commands together (for example, ps aux | grep nginx | awk '{print $2}') before trying the same pipeline on a real server.

# Chapter 16

## The Linux Ubuntu 26.04 Manual Pages in PDF Format

*Browse generated PDFs*

This chapter extends Chapters 8 and 13 specifically to Ubuntu 26.04 LTS, the release this book is written against throughout. As Ubuntu evolves, command defaults and available flags occasionally change between LTS releases (for example, ufw replacing older firewall front-ends, or systemd subsuming what used to be separate init and cron-adjacent tools) — this version-pinned archive ensures the manual page you read matches the behavior you'll actually see when you run a command on 26.04.

# Chapter 17

## Long Video Lectures — MyWebUniversity (English)

*Full-length IT, DevOps, Linux, Python, and programming courses*

The @MyWebUniversity YouTube channel hosts full-length lecture recordings — many well over an hour — covering Linux fundamentals, shell scripting, DevOps tooling, and the programming languages covered in Chapters 22–30. These are the same lectures used in the university and community college courses I teach, recorded so anyone can follow along regardless of enrollment. They pair naturally with this book: read a chapter first, then watch the corresponding lecture for a second pass with live demonstration and Q&A.

# Chapter 18

## Long Video Lectures — MyWebUniversity (Dari)

*Full-length IT and programming courses in Dari*

The @MyWebUniversity-Dari channel mirrors the English-language lecture library in Dari (Farsi), making the same Linux, IT, and programming curriculum accessible to Dari-speaking students without a language barrier. Making technical education available in Dari has been a personal priority of mine, since language should never be the obstacle between a motivated student and a technical career.

# Chapter 19

## Quick Tips & Short Tutorials — English

*Bite-sized tutorials under 5 minutes*

Not every lesson needs an hour. This chapter's companion library holds short, single-topic videos — each under five minutes — for the moments you just need one specific answer. Three examples of the format:

- "What does chmod 755 actually mean?" — a 3-minute breakdown of octal permission notation.
- "Finding the process using a port" — a 2-minute walkthrough of lsof -i :PORT and netstat -tulnp.
- "The difference between > and >>" — a 90-second demonstration of output redirection versus appending.

# Chapter 20

## Quick Tips & Short Tutorials — Dari

*ویدیوهای کوتاه زیر پنج دقیقه*

The same quick-tip format as Chapter 19, presented in Dari for the MyWebUniversity-Dari audience — short, single-topic clips answering one specific command or concept at a time, sorted by view count so the most-requested topics surface first.

# Chapter 21

## Linux Man Pages Online — PDF & HTML

*9,000+ Ubuntu 26.04 LTS man pages across 20 categories*

This is the master reference chapter that Chapters 7, 8, 12, 13, and 16 all feed into: a single searchable index of over 9,000 Ubuntu 26.04 LTS manual pages, organized into 20 categories spanning shell utilities, Python, Java, Go, Node.js, Ruby, MySQL, and more, each with one-click access to both HTML and PDF formats.

### 21.1 How to Use This as a Working Reference

Treat this chapter's companion page the way experienced developers treat man -k (Chapter 7): not something you read start to finish, but something you search the moment you forget an exact flag or function signature. Bookmark it, and get comfortable jumping to it mid-task rather than trying to memorize everything in Chapters 22–43 on first read.

# Chapter 22

## GCC C Language — Headers, Functions & Code Examples

*Complete C Standard Library reference*

### Overview and History

C was designed by Dennis Ritchie at Bell Labs between 1969 and 1973 to rewrite the UNIX kernel in a portable, high-level language (Chapter 1). It remains the language closest to the hardware while still being portable, and nearly every operating system kernel, including Linux itself, is written in it. On Ubuntu, the standard compiler is GCC, the GNU Compiler Collection.

### Syntax and Semantics

- Statically typed: every variable's type (int, char, float, double, etc.) is fixed at compile time.
- A program's entry point is always a function named main, which returns an int (0 conventionally means success).
- Blocks are delimited with { } and every statement ends with a semicolon.
- Manual memory management via malloc()/free() — there is no garbage collector.
- Preprocessor directives (#include, #define) run before compilation proper and textually include headers (Chapter 9) or substitute macros.

### Installing, Compiling, and Running

GCC ships in Ubuntu's build-essential package, which also pulls in make and standard C libraries:

```
sudo apt update
sudo apt install build-essential
gcc --version
```

## "Hello, World" Example

*File: hello.c*

```
#include <stdio.h>

int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

**Run it:**

```
gcc -o hello hello.c
./hello
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example loops from 1 to 10, uses an if/else branch to test for even or odd, and demonstrates break by stopping the loop early once it reaches 8.

*File: loopbranch.c*

```
#include <stdio.h>

int main(void) {
    for (int i = 1; i <= 10; i++) {
        if (i == 8) {
            printf("Reached 8, breaking out of
loop.\n");
            break;
        }
        if (i % 2 == 0) {
            printf("%d is even\n", i);
        } else {
            printf("%d is odd\n", i);
        }
    }
    return 0;
```

```
}
```

**Output:**

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 23

## G++ C++ Language — Headers, Namespaces & Code Examples

*Complete C++ Standard Library reference*

### Overview and History

C++ was created by Bjarne Stroustrup at Bell Labs starting in 1979 as "C with Classes," adding object-oriented programming (classes, inheritance, polymorphism) on top of C while retaining near-total C compatibility and low-level performance. On Ubuntu, the standard compiler is g++, GCC's C++ front end.

### Syntax and Semantics

- Everything from C's syntax remains valid, plus classes, constructors/destructors, templates, and exceptions (try/catch/throw).
- The std namespace holds the Standard Template Library (STL): containers like std::vector and std::map, and I/O streams like std::cout and std::cin.
- The using namespace std; directive avoids repeating std:: before every standard library name, though many style guides prefer being explicit in larger programs.
- Modern C++ (C++11 and later) adds auto type inference, lambdas, smart pointers (std::unique_ptr, std::shared_ptr), and range-based for loops.

### Installing, Compiling, and Running

g++ is included in the same build-essential package as GCC:

```
sudo apt install build-essential
g++ --version
```

### "Hello, World" Example

*File: hello.cpp*

```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

**Run it:**

```
g++ -o hello hello.cpp
./hello
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example uses a range-based for loop over a std::vector, an if/else branch, and break to exit early when a target value is found.

*File: loopbranch.cpp*

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> nums = {2, 4, 6, 7, 8, 10};
    for (int n : nums) {
        if (n % 2 != 0) {
            std::cout << n << " is odd -- stopping
search.\n";
            break;
        }
        std::cout << n << " is even\n";
    }
    return 0;
}
```

**Output:**

```
2 is even
4 is even
6 is even
7 is odd -- stopping search.
```

# Chapter 24

## Go Programming Language — Packages, Functions & Code Examples

*Complete Go Standard Library reference*

### Overview and History

Go (often called Golang) was designed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson — the same Ken Thompson who co-created UNIX (Chapter 1) — and released publicly in 2009. It was built to combine the fast compilation and simplicity of a scripting language with the performance and static typing of a compiled one, and it has built-in concurrency via goroutines and channels.

### Syntax and Semantics

- Every Go source file belongs to a package, and executable programs must have a package main with a func main().
- Types are static but often inferred with :=, e.g. x := 5 instead of var x int = 5.
- No classes; behavior is attached to types via methods, and interfaces define behavior contracts structurally (no explicit "implements" keyword needed).
- Concurrency primitives: go someFunc() starts a goroutine; channels (chan) pass data safely between goroutines.
- Error handling uses ordinary multiple return values (result, err) rather than exceptions.

### Installing, Compiling, and Running

Install Go from Ubuntu's repository or the official tarball for the newest version:

```
sudo apt update
sudo apt install golang-go
go version
```

## "Hello, World" Example

*File: hello.go*

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

**Run it:**

```
go run hello.go
# or build a standalone binary:
go build -o hello hello.go
./hello
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

Go has only one looping keyword, for, which covers what other languages split across for/while. This example loops through numbers 1-10, branches with if/else, and uses break.

*File: loopbranch.go*

```
package main

import "fmt"

func main() {
    for i := 1; i <= 10; i++ {
        if i == 8 {
            fmt.Println("Reached 8, breaking out of
loop.")
            break
        }
        if i%2 == 0 {
            fmt.Printf("%d is even\n", i)
```

```
        } else {
            fmt.Printf("%d is odd\n", i)
        }
    }
}
```

**Output:**

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 25

## Java Programming Language — Classes, Collections & Code Examples

*Complete Java core reference*

### Overview and History

Java was released by Sun Microsystems in 1995 (led by James Gosling), designed around the principle of "write once, run anywhere" — source code compiles to platform-independent bytecode that runs on the Java Virtual Machine (JVM) rather than directly on any one operating system's native machine code. Oracle acquired Sun (and Java) in 2010 and continues to steward the language today.

### Syntax and Semantics

- Strictly object-oriented: every piece of executable code lives inside a class.
- The entry point is a public static void main(String[] args) method.
- Statically typed with both primitive types (int, double, boolean) and reference types (String, and any class).
- Automatic garbage collection — no manual free() as in C.
- The Collections Framework (java.util) provides ArrayList, HashMap, and the Streams API for functional-style data processing.

### Installing, Compiling, and Running

Install the OpenJDK (Open Java Development Kit) from Ubuntu's repositories:

```
sudo apt update
sudo apt install openjdk-21-jdk
java --version
javac --version
```

## "Hello, World" Example

*File: Hello.java*

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

**Run it:**

```
javac Hello.java
java Hello
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example uses a for loop over 1 through 10, an if/else branch to test even/odd, and break to exit once it reaches 8.

*File: LoopBranch.java*

```
public class LoopBranch {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i == 8) {
                System.out.println("Reached 8, breaking
out of loop.");
                break;
            }
            if (i % 2 == 0) {
                System.out.println(i + " is even");
            } else {
                System.out.println(i + " is odd");
            }
        }
    }
}
```

**Output:**

```
1 is odd
```

```
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 26

## Ruby Programming Language — Classes, Modules & Code Examples

*Complete Ruby built-in reference*

### Overview and History

Ruby was designed by Yukihiro "Matz" Matsumoto in Japan and publicly released in 1995, explicitly prioritizing programmer happiness and readability. It gained enormous popularity through the Ruby on Rails web framework (2004), and remains a dynamically typed, fully object-oriented language where even integers and nil are objects with methods.

### Syntax and Semantics

- Dynamically typed and interpreted; no type declarations required.
- Blocks (do...end or { }) are used pervasively, especially with iterator methods like each and map.
- Everything is an object, including numbers, and methods are called with dot notation.
- No braces for method or class bodies — blocks are closed with the end keyword.
- String interpolation with "text #{expression} text" is idiomatic and used constantly.

### Installing, Compiling, and Running

Install Ruby from Ubuntu's repository:

```
sudo apt update
sudo apt install ruby-full
ruby --version
```

### "Hello, World" Example

*File: hello.rb*

```
puts "Hello, World!"
```

**Run it:**

```
ruby hello.rb
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example uses a range (1..10).each loop, an if/else branch for even/odd, and break to stop once it reaches 8.

*File: loopbranch.rb*

```
(1..10).each do |i|
  if i == 8
    puts "Reached 8, breaking out of loop."
    break
  end
  if i.even?
    puts "#{i} is even"
  else
    puts "#{i} is odd"
  end
end
```

**Output:**

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 27

## Rust Programming Language — Ownership, Traits & Code Examples

*Complete Rust standard library reference*

### Overview and History

Rust began as a personal project by Mozilla engineer Graydon Hoare in 2006 and became an official Mozilla-sponsored project in 2009, with a stable 1.0 release in 2015. Its defining feature is the ownership and borrowing system, which enforces memory safety (no null pointer dereferences, no data races, no use-after-free) entirely at compile time, with no garbage collector and no runtime performance cost.

### Syntax and Semantics

- Every value has a single owner; when the owner goes out of scope, the value is automatically dropped (freed) — this is enforced by the compiler's "borrow checker."
- Variables are immutable by default; mut must be added explicitly to allow reassignment.
- No exceptions: recoverable errors use the Result<T, E> enum; absence of a value uses Option<T> instead of null.
- Traits define shared behavior across types, similar to interfaces in Java or Go.
- Cargo is Rust's official build tool and package manager, analogous to npm for JavaScript or pip for Python.

### Installing, Compiling, and Running

The official installer sets up the compiler (rustc) and Cargo together:

```
curl --proto '=https' --tlsv1.2 -sSf
https://sh.rustup.rs | sh
source "$HOME/.cargo/env"
rustc --version
```

## "Hello, World" Example

*File: hello.rs*

```
fn main() {
    println!("Hello, World!");
}
```

**Run it:**

```
rustc hello.rs
./hello
# or, using Cargo:
cargo new hello_project && cd hello_project
cargo run
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example uses a for loop over a range (1..=10), an if/else branch for even/odd, and break to exit once it reaches 8.

*File: loopbranch.rs*

```
fn main() {
    for i in 1..=10 {
        if i == 8 {
            println!("Reached 8, breaking out of
loop.");
            break;
        }
        if i % 2 == 0 {
            println!("{} is even", i);
        } else {
            println!("{} is odd", i);
        }
    }
}
```

**Output:**

```
1 is odd
2 is even
```

```
3 is odd
4 is even
5 is odd
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 28

# Perl Programming Language — Regex, File I/O & Code Examples

*Complete Perl reference*

## Overview and History

Perl was created by Larry Wall in 1987 and became the dominant language for text processing, system administration scripting, and early CGI web programming through the 1990s — you'll notice several of MyWebUniversity.com's own dynamic pages (Chapters 7 and 12) are built on CGI scripts, a tradition Perl pioneered. Its regular-expression engine remains one of the most powerful and widely copied in the industry.

## Syntax and Semantics

- Scalar variables start with $, arrays with @, and hashes (associative arrays) with %.
- Regular expressions are a first-class part of the syntax via the =~ binding operator and s/// (substitute) and m// (match) operators.
- Statements end with a semicolon; blocks use { }.
- Context matters heavily: the same variable or function can behave differently in scalar vs. list context.
- use strict; and use warnings; at the top of a script are considered mandatory good practice.

## Installing, Compiling, and Running

Perl ships pre-installed on virtually every Ubuntu system, but you can confirm the version or reinstall it explicitly:

```
sudo apt install perl
perl -v
```

## "Hello, World" Example

*File: hello.pl*

```
#!/usr/bin/perl
use strict;
use warnings;

print "Hello, World!\n";
```

**Run it:**

```
chmod +x hello.pl
./hello.pl
# or simply:
perl hello.pl
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example uses a for loop from 1 to 10, an if/else branch for even/odd, and last (Perl's keyword for break) to exit once it reaches 8.

*File: loopbranch.pl*

```
#!/usr/bin/perl
use strict;
use warnings;

for my $i (1..10) {
    if ($i == 8) {
        print "Reached 8, breaking out of loop.\n";
        last;
    }
    if ($i % 2 == 0) {
        print "$i is even\n";
    } else {
        print "$i is odd\n";
    }
}
```

**Output:**

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 29

## R Programming Language — Data Science & Statistical Computing

*Complete R reference*

### Overview and History

R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, first released in 1995, as a free, open-source implementation of the S statistical programming language. It has since become one of the two dominant languages in data science and academic statistics (alongside Python), with the CRAN repository hosting over 20,000 add-on packages, and the tidyverse (including dplyr and ggplot2) shaping how most modern R code is written.

### Syntax and Semantics

- Vectors are the fundamental data structure — even a single number is technically a vector of length 1.
- Assignment is idiomatically written with <- rather than =, though = also works.
- Data frames (tables with named, typed columns) are the standard structure for tabular data, heavily used with the dplyr package.
- Functions are first-class values and vectorized operations (applying a function to an entire vector at once) are idiomatic instead of explicit loops wherever possible.
- RStudio is the most common development environment, though R itself runs from any terminal via Rscript.

### Installing, Compiling, and Running

Install R from Ubuntu's repository:

```
sudo apt update
sudo apt install r-base
R --version
```

## "Hello, World" Example

*File: hello.R*

```
cat("Hello, World!\n")
```

**Run it:**

```
Rscript hello.R
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example uses a for loop from 1 to 10, an if/else branch for even/odd, and break to exit once it reaches 8.

*File: loopbranch.R*

```
for (i in 1:10) {
  if (i == 8) {
    cat("Reached 8, breaking out of loop.\n")
    break
  }
  if (i %% 2 == 0) {
    cat(i, "is even\n")
  } else {
    cat(i, "is odd\n")
  }
}
```

**Output:**

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 30

# JavaScript & Node.js — ES6+, Async, DOM & Code Examples

*Modern JavaScript reference*

## Overview and History

JavaScript was created by Brendan Eich at Netscape in just ten days in 1995 to add interactivity to web pages, and it is now standardized as ECMAScript (ES). Node.js, created by Ryan Dahl in 2009 on top of Google's V8 engine, let JavaScript run outside the browser as a general-purpose, server-side language, powering much of the modern web backend alongside frameworks like Express.

## Syntax and Semantics

- Dynamically typed; variables are declared with let, const, or the older var (avoid var in modern code).
- Functions are first-class values, with concise arrow-function syntax: (a, b) => a + b.
- Asynchronous operations use Promises and the async/await syntax rather than deeply nested callbacks.
- In the browser, JavaScript manipulates the Document Object Model (DOM); in Node.js, the same language instead accesses the filesystem, network, and OS via built-in modules like fs and http.
- Modules use import/export (ES modules) or require()/module.exports (Node's original CommonJS system).

## Installing, Compiling, and Running

Install Node.js (which bundles the V8 engine and the npm package manager) from Ubuntu's NodeSource repository for the latest LTS release:

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo
-E bash -
sudo apt install -y nodejs
node --version
npm --version
```

## "Hello, World" Example

*File: hello.js*

```
console.log("Hello, World!");
```

**Run it:**

```
node hello.js
```

**Output:**

```
Hello, World!
```

## Looping and Branching Example

This example uses a for loop from 1 to 10, an if/else branch for even/odd, and break to exit once it reaches 8.

*File: loopbranch.js*

```
for (let i = 1; i <= 10; i++) {
  if (i === 8) {
    console.log("Reached 8, breaking out of loop.");
    break;
  }
  if (i % 2 === 0) {
    console.log(`${i} is even`);
  } else {
    console.log(`${i} is odd`);
  }
}
```

**Output:**

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
```

```
6 is even
7 is odd
Reached 8, breaking out of loop.
```

# Chapter 31

## HTML5 & CSS3 — Semantic Markup, Flexbox & Grid

*Complete HTML5 and CSS3 reference*

HTML (HyperText Markup Language) structures content; CSS (Cascading Style Sheets) styles it. HTML5, standardized by the WHATWG and W3C, introduced semantic elements (<header>, <nav>, <main>, <article>, <section>, <footer>) that describe a page's meaning rather than only its appearance — important both for accessibility (screen readers) and search engines.

### 31.1 A Minimal Semantic Page

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"><title>Direct Path to Linux
Ubuntu</title></head>
<body>
  <header><h1>The Direct Path to Linux
Ubuntu</h1></header>
  <main>
    <section>
      <h2>Chapter 31: HTML5 &amp; CSS3</h2>
      <p>Semantic markup improves accessibility and
SEO.</p>
    </section>
  </main>
  <footer><p>&copy; 2026 Wahidullah Lutfy</p></footer>
</body>
</html>
```

### 31.2 CSS3 Flexbox and Grid

Flexbox (one-dimensional layout) and CSS Grid (two-dimensional layout) replaced older float- and table-based layout techniques:

```
.container {
  display: flex;
```

```
  justify-content: space-between;
  gap: 1rem;
}
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 1rem;
}
```

# Chapter 32

## JSON — Syntax, Parsing, Schema & Code Examples

*Complete JSON reference*

JSON (JavaScript Object Notation) is a lightweight, text-based data-interchange format derived from JavaScript object syntax, but language-independent — it is the de facto standard for REST API request and response bodies. Valid JSON values are: objects ({ }), arrays ([ ]), strings, numbers, true/false, and null.

### 32.1 Example Document

```
{
  "book": "The Direct Path to Linux Ubuntu",
  "author": "Wahidullah Lutfy",
  "chapters": 43,
  "topics": ["Linux", "Shell Scripting", "Cloud",
"Security"]
}
```

### 32.2 Parsing JSON

```
// JavaScript
const data = JSON.parse(jsonString);
console.log(data.chapters); // 43

# Python
import json
data = json.loads(json_string)
print(data['chapters'])  # 43
```

On the command line, jq is the standard tool for filtering and reshaping JSON: jq '.chapters' file.json.

# Chapter 33

## XML — XPath, XSLT, XSD & Code Examples

*Complete XML reference*

XML (eXtensible Markup Language) predates JSON as a data-interchange and document format and remains dominant in enterprise systems, configuration files (including Apache and Ansible, Chapters 6 and 38), and document formats like DOCX and PPTX themselves. Unlike JSON, XML supports attributes, namespaces, and formal schema validation.

### 33.1 Example Document

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <title>The Direct Path to Linux Ubuntu</title>
  <author>Wahidullah Lutfy</author>
  <chapters count="43"/>
</book>
```

### 33.2 XPath: Querying XML

XPath expressions navigate an XML document similarly to how a file path navigates a filesystem:

```
/book/title
/book/chapters/@count
```

XSLT transforms XML into another format (often HTML), and XSD (XML Schema Definition) validates that a document's structure matches an agreed contract.

# Chapter 34

# YAML — DevOps, Kubernetes, Ansible & Code Examples

*Complete YAML reference*

YAML ("YAML Ain't Markup Language") is a human-friendly data serialization format built around indentation rather than braces or brackets, which makes it the preferred configuration format for Docker Compose (Chapter 40), Kubernetes, Ansible (Chapter 38), and GitHub Actions.

## 34.1 Example: An Ansible-Style Playbook

```
- name: Install and start Apache
  hosts: webservers
  become: true
  tasks:
    - name: Install apache2
      apt:
        name: apache2
        state: present
    - name: Start apache2
      service:
        name: apache2
        state: started
```

Indentation is meaningful in YAML (typically two spaces per level, never tabs), which is the single most common source of YAML errors for newcomers.

# Chapter 35

## SQL Databases — MySQL, PostgreSQL, SQLite & Code Examples

*Complete SQL reference*

SQL (Structured Query Language) is the standard language for relational databases. MySQL and its community fork MariaDB are common on Linux (and part of the classic LAMP stack, Chapter 6); PostgreSQL is favored for advanced features and strict standards compliance; SQLite is a serverless, file-based database ideal for embedded and small-scale use.

### 35.1 Core Statements

```
CREATE TABLE chapters (
  id INT PRIMARY KEY,
  title VARCHAR(100),
  page_count INT
);

INSERT INTO chapters VALUES (1, 'About UNIX and Linux',
12);

SELECT title FROM chapters WHERE page_count > 10;

UPDATE chapters SET page_count = 14 WHERE id = 1;
```

### 35.2 Joins and Python Integration

A JOIN combines rows from two tables based on a related column — for example, matching chapters to authors:

```
SELECT c.title, a.name
FROM chapters c
JOIN authors a ON c.author_id = a.id;

# Python via sqlite3
import sqlite3
conn = sqlite3.connect('book.db')
```

```
cur = conn.cursor()
cur.execute('SELECT title FROM chapters')
print(cur.fetchall())
```

# Chapter 36

## LDAP & Directory Services — OpenLDAP, Active Directory & Examples

*Complete LDAP reference*

LDAP (Lightweight Directory Access Protocol) stores and queries hierarchical directory information — most commonly user accounts, groups, and organizational structure — and underlies both OpenLDAP (common on Linux) and Microsoft Active Directory. Entries are identified by a Distinguished Name (DN), such as cn=wahid,ou=people,dc=mywebuniversity,dc=com.

### 36.1 Querying with ldapsearch

```
ldapsearch -x -H ldap://ldap.example.com -b
"dc=mywebuniversity,dc=com" "(uid=wahid)"
```

### 36.2 LDIF: The Entry Exchange Format

```
dn: cn=wahid,ou=people,dc=mywebuniversity,dc=com
objectClass: inetOrgPerson
cn: wahid
sn: Lutfy
mail: wahid@mywebuniversity.com
```

Python's ldap3 library provides a fully-featured client for scripting LDAP queries and modifications rather than relying on shell commands alone.

# Chapter 37

## Cloud Computing — AWS, Azure, GCP & Architecture

*Complete cloud reference*

Cloud computing rents compute, storage, and networking on demand rather than requiring owned physical hardware, billed by usage. The three service models are IaaS (Infrastructure as a Service — raw virtual machines and networking), PaaS (Platform as a Service — a managed runtime you deploy code into), and SaaS (Software as a Service — a finished application, like Gmail).

### 37.1 Core Building Blocks

- Virtual Private Cloud (VPC): an isolated virtual network with subnets, route tables, and security groups controlling traffic.
- Compute: AWS EC2, Azure Virtual Machines, and Google Compute Engine all provide on-demand virtual servers.
- Object storage: AWS S3, Azure Blob Storage, and Google Cloud Storage store files at massive scale over HTTP APIs.
- IAM (Identity and Access Management): fine-grained permission policies controlling exactly which users or services can perform which actions.

### 37.2 A Minimal AWS CLI Example

```
aws s3 mb s3://direct-path-linux-ubuntu-backups
aws s3 cp manuscript.docx s3://direct-path-linux-ubuntu-
backups/
aws ec2 describe-instances --query
'Reservations[].Instances[].InstanceId'
```

# Chapter 38

## Ansible & OpenShift — Automation & Container Orchestration

*Complete Ansible and OpenShift reference*

Ansible is an agentless automation tool: it connects over plain SSH (no special software required on the managed machine) and applies YAML-defined playbooks (Chapter 34) to configure servers idempotently — running the same playbook twice produces the same end state without duplicating changes. OpenShift is Red Hat's enterprise Kubernetes distribution, adding a web console, integrated CI/CD pipelines, and stricter default security policies on top of vanilla Kubernetes.

### 38.1 Running an Ansible Playbook

```
ansible-playbook -i inventory.ini site.yml
```

### 38.2 Basic OpenShift (oc) Commands

```
oc login https://api.cluster.example.com:6443
oc new-project direct-path-demo
oc new-app nginx:latest
oc get pods
oc expose svc/nginx
```

# Chapter 39

## AWS & Azure — Cloud Services, CLI & Python SDK

*Complete AWS and Azure service reference*

This chapter goes one level deeper than Chapter 37's architecture overview, focusing on hands-on CLI and SDK usage for the two most widely used providers.

### 39.1 AWS: boto3 (Python SDK)

```
import boto3
ec2 = boto3.client('ec2')
response = ec2.describe_instances()
for r in response['Reservations']:
    for i in r['Instances']:
        print(i['InstanceId'], i['State']['Name'])
```

### 39.2 Azure CLI

```
az login
az group create --name direct-path-rg --location eastus
az vm create --resource-group direct-path-rg --name
web01 --image Ubuntu2604
```

# Chapter 40

## Docker & Podman — Containers, Images & Compose

*Complete Docker and Podman reference*

Containers package an application with its exact dependencies into a single portable unit that runs identically across development, testing, and production. Docker popularized this model; Podman is a daemonless, rootless alternative with a nearly identical command-line interface, favored in security-conscious environments.

### 40.1 A Minimal Dockerfile

```
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["node", "server.js"]
```

### 40.2 Building, Running, and Compose

```
docker build -t direct-path-app .
docker run -d -p 3000:3000 direct-path-app
# Podman uses the identical syntax:
podman build -t direct-path-app .
podman run -d -p 3000:3000 direct-path-app
```

```
# docker-compose.yml
services:
  web:
    build: .
    ports:
      - "3000:3000"
  db:
    image: postgres:16
```

# Chapter 41

## Apache & NGINX — Web Servers, SSL & Reverse Proxy

*Complete Apache and NGINX reference*

Building on Chapter 6's introduction, this chapter covers production concerns: TLS encryption, reverse proxying, and performance. NGINX's event-driven architecture handles far more concurrent connections per server than Apache's traditional process/thread-per-connection model, which is why NGINX is frequently placed in front of Apache or an application server as a reverse proxy rather than replacing it outright.

### 41.1 Free TLS Certificates with Let's Encrypt

```
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d example.com -d www.example.com
```

### 41.2 NGINX as a Reverse Proxy

```
server {
    listen 80;
    server_name example.com;
    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

# Chapter 42

## Protocols, APIs & Security — HTTP, TLS, SSH, OAuth2 & Cybersecurity

*Complete protocols and security reference*

This chapter ties together the networking and security concepts referenced throughout the book: the HTTP request/response model (Chapter 6), TLS 1.3's handshake for encrypting traffic (Chapter 41), SSH key-based authentication (Chapter 1), and REST APIs secured with OAuth2 and JWTs.

### 42.1 Generating and Using an SSH Key

```
ssh-keygen -t ed25519 -C "wahid@mywebuniversity.com"
ssh-copy-id wahid@server.example.com
ssh wahid@server.example.com
```

### 42.2 Calling a REST API with curl and a Bearer Token

```
curl -H "Authorization: Bearer $TOKEN"
https://api.example.com/v1/chapters
```

### 42.3 Linux Hardening Basics

- Keep the firewall active and default-deny: sudo ufw default deny incoming.
- Disable password-based SSH login once key-based auth works, in /etc/ssh/sshd_config.
- Install fail2ban to automatically block IPs after repeated failed login attempts.
- Review the OWASP Top 10 (owasp.org) for the most common web application vulnerabilities, such as injection and broken access control.

# Chapter 43

## Networking — OSI Model, TCP/IP, DNS, Firewalls & WireGuard VPN

*Complete networking reference*

The OSI model describes networking in seven conceptual layers (Physical, Data Link, Network, Transport, Session, Presentation, Application); in practice, the TCP/IP model's four layers (Link, Internet, Transport, Application) map more directly onto real protocols and are what you will troubleshoot day to day.

### 43.1 TCP vs. UDP

- TCP: connection-oriented, guarantees ordered delivery and retransmits lost packets — used by HTTP, SSH, and most application protocols.
- UDP: connectionless, no delivery guarantee, lower overhead — used by DNS queries, video streaming, and WireGuard's own transport.

### 43.2 Diagnostic Commands

```
dig example.com A
traceroute example.com
ss -tulnp
sudo nmap -sV 192.168.1.0/24
```

### 43.3 A Minimal WireGuard VPN Configuration

```
[Interface]
PrivateKey = <server-private-key>
Address = 10.0.0.1/24
ListenPort = 51820

[Peer]
PublicKey = <client-public-key>
AllowedIPs = 10.0.0.2/32
```

With this chapter, the book's technical arc is complete: from a single command typed at a prompt in Chapter 2, through nine programming languages, to securing and connecting entire networks in Chapter 43.