Your Rocky Linux VPS

A GSP Administration Handbook

Covering Rocky Linux 10, Webmin, Sendmail, Dovecot, Apache, and Roundcube

GSP Services

Mail:	1900 Cavalier Cir
	Crofton, MD 21114-1705
	USA
Web:	https://www.gsp.com
Phone:	1.866.477.4400
E-Mail:	service@gsp.com

Table of Contents

Document Conventions 1
Getting Started in 8 Steps 2
Two User Identities 2
Step 1: Point Your Domain Name at the VPS 2
Step 2: Connect to Your VPS 3
Step 3: Learn the Linux Basics 3
Step 4: Add Users and Virtual Hosts 4
Step 5: Upload Your Web Files 4
Step 6: Configure Email 4
Step 7: Watch Your Web Statistics 5
Step 8: Go Beyond the Basics 5
Chapter 1: Introducing Your Rocky Linux VPS 6
How the System Works 6
Why a VPS? VPS vs. Virtual Hosting vs. Do-It-Yourself 6
Root User and Administrative User 8
Administering the Server: SSH and SFTP 9
Connecting from Common Clients 9
The Linux File System 9
A Tour of the Top-Level Directories 9
Navigating the File System 10
File Ownership and Permissions 10
Changing Permissions and Ownership 11
Editing Files 11
nano 11
vi / vim 12
Choosing and Using a Shell 14
Where to Go Next 15
Chapter 2: Users and Accounts 16
Categories of Account 16
Switching to Root: sudo and su 16
Creating New User Accounts 17
Editing Existing Accounts 18

Disabling and Removing Accounts 18
Groups 19
Resource Limits 19
Important Commands and Files 19
Where to Go Next 20
Chapter 3: Webmin — Your Web-Based Control Panel 21
Installing Webmin 21
Securing Access to Webmin 22
A Tour of Webmin 22
System → Users and Groups 22
Others → File Manager (Filemin) 23
System → Software Package Updates 23
System → Bootup and Shutdown 23
Servers → Apache Webserver 23
Servers → Sendmail Mail Server 23
System → Disk and Network Filesystems 24
Networking → Linux Firewall 24
Webmin → Webmin Configuration 24
Logging Out 24
Where to Go Next 24
Chapter 4: The Mail Server and Roundcube Webmail 25
Email Architecture: MTA, Mailbox Store, and Mail Clients 25
Why Rocky Linux's Default Isn't Enough 25
Setting Up Sendmail 25
Switching the System Mailer from Postfix to Sendmail 25
The .mc File and local-host-names 26
Aliases 26
Virtual Domains and Virtual Users 27
Aliases vs. Virtual Aliases (valiases) 27
Access Control and Basic Spam Defenses 28
Autoresponders ("Vacation" Messages) 29
Setting Up Dovecot 29
SMTP Authentication 30
Filtering Spam with SpamAssassin 31

At the system (OS) level 31
Within SpamAssassin 32
Configuring Mail Clients 33
Roundcube Webmail 33
Installing Roundcube 33
Configuring Roundcube 34
Publishing Roundcube on the Web 34
Using Roundcube 35
Maintaining Mail Logs 35
Important Commands, Directories, and Files 35
For More Information 36
Where to Go Next 36
Chapter 5: Transferring Files with SFTP 37
Why SFTP Instead of FTP 37
Command-Line SFTP 37
scp 37
rsync over SSH 38
Graphical SFTP Clients 38
Restricting an Account to SFTP Only 38
If You Need Classic FTP 39
Where to Go Next 40
Chapter 6: The Web Server 41
Installing Apache 41
Directory Layout 41
Publishing Web Content 42
Virtual Hosting: Multiple Sites on One VPS 42
A Note on AllowOverride and .htaccess 43
Securing Apache 43
HTTPS with Let's Encrypt and Certbot 44
Important Commands, Directories, and Files 45
Where to Go Next 45
Chapter 7: Advanced Web Server Configuration 46
A Working Reference of Apache Directives 46
Where Files Live 46

Redirects and Rewrites 46
Custom Error Pages 47
Working with Dynamically-Loaded Modules 47
MIME Types 48
Password-Protected Directories 48
Server-Side Includes (SSI) 49
HTTPS and TLS, in More Depth 49
Redirecting HTTP to HTTPS 50
HSTS (HTTP Strict Transport Security) 50
Cipher and Protocol Configuration 50
For More Information 51
Where to Go Next 51
Chapter 8: Web Applications and Security 52
Installing PHP 52
Connecting Apache to PHP-FPM 52
Adding a Database 53
Deploying an Application 53
File Ownership, the "Execute Bit," and SELinux Contexts 54
A Practical Security Checklist 55
Troubleshooting Application Errors 56
Where to Go Next 56
Chapter 9: Maintaining Your VPS 57
Server Administration: Services and Common Ports 57
Logs 58
logrotate: Automatic Log Rotation 59
Scheduling Tasks: cron and systemd Timers 59
Monitoring Load and Processes 60
Reading top 60
ps: a snapshot of processes 62
du and df: where the disk went 62
Killing a Runaway Process 62
Backups 63
File-Based Backups 63
Database Backups 63

LVM Snapshots 63
Restoring Files 64
Watching Web Statistics 64
Troubleshooting Common Errors 65
A Troubleshooting Checklist 66
Important Commands and Files 67
Where to Go Next 67
Chapter 10: Securing Your VPS 68
Staying Patched 68
dnf upgrade and Security Errata 68
Automating Updates with dnf-automatic 68
The firewalld Firewall 69
A Starting Ruleset 69
SSH Hardening 70
Switch to Key-Based Authentication 70
Disable Password Authentication 70
fail2ban: Blocking Repeat Offenders 71
SELinux: Mandatory Access Control 71
Hardening Checklist for Earlier Chapters 73
Staying Informed 73
Where to Go Next 73
Appendix A: Package Management — dnf, Repositories, and Building from Source 74
dnf Basics 74
Repositories 75
Building from Source 75
A Note on Image Mode (bootc) 76
Appendix B: Quick Command Reference 77
Files and Navigation 77
Permissions and Ownership 77
Users 77
Processes and Resource Usage 77
Networking 78
Packages and Services 78
Firewall and SELinux 78

Mail 78
Appendix C: Getting Help 80
Rocky Linux and RHEL Documentation 80
Application Documentation 80
GSP Support 81
Appendix D: Scripting and Development Tools 82
Scripting with Perl 82
Managing CPAN Modules 83
Compilers: gcc, clang, and Building Software 84

Document Conventions

This handbook uses a small set of conventions to keep instructions unambiguous, especially when a single task can be completed either from a terminal or from the Webmin control panel.

Command prompts. Commands you type at a shell prompt are shown in a monospace font. The character at the start of the line tells you which account should run the command:

- A dollar sign (`$`) means the command should be run as your ordinary administrative user.
- A number sign (`#`) means the command requires root privileges, either because you ran it with `sudo`, or because you have switched to a root shell with `sudo -i` or `su -`.

For example:

```
$ ls -l
# dnf install httpd
```

The prompt character itself is never part of what you type.

File and directory names are shown in monospace, such as `/etc/httpd/conf/httpd.conf`. Where a path includes a placeholder you must replace, it appears in angle brackets, such as `/home/<username>/public_html`.

Webmin navigation is shown as a path through the menu, separated by arrows, such as **System → Users and Groups**. This refers to the category shown in the left-hand menu of the Webmin interface, followed by the module you should click.

Placeholders. Throughout this handbook, `example.com` stands in for your own domain name, and `203.0.113.10` stands in for the public IPv4 address GSP has assigned to your VPS. Replace these with your real domain and IP address wherever they appear. (`203.0.113.0/24` is a block reserved by RFC 5737 specifically for use in documentation, so it will never collide with a real address.)

Notes, tips, and warnings are set off from the main text:

> **Note:** Background information or an explanation of why something works the way it does.

> **Tip:** A shortcut, an alternative approach, or a way to save yourself trouble later.

> **Warning:** An action that can lock you out of your VPS, expose it to attackers, or destroy data if performed carelessly.

Getting Started in 8 Steps

This chapter is a map of the whole handbook. If you only read one chapter before diving in, read this one — it walks through the first things every GSP customer needs to do with a new Rocky Linux 10 VPS, and points to the chapter that covers each topic in depth.

Two User Identities

Every GSP VPS gives you access to two kinds of accounts, and understanding the difference between them now will save you a lot of confusion later.

root is the superuser. It can read, write, or delete any file; install or remove any package; and reconfigure any service, including ones that could make the server unreachable if misconfigured. GSP does not log you in as root by default, and we recommend you avoid working as root for day-to-day tasks. Instead, become root only for the specific command that needs it, then return to your normal account.

Your administrative user is the personal account GSP created for you when your VPS was provisioned (its name was included in your welcome email). This account is a member of the `wheel` group, which on Rocky Linux is the group whose members are allowed to run commands as root with `sudo`. You should do almost everything from this account: editing your web files, reading your mail, running Webmin, and so on.

Chapter 2, "Users and Accounts," explains how to create additional accounts of both kinds — for example, a separate login for a colleague, or a restricted account for a client who only needs to upload files to one web site.

Step 1: Point Your Domain Name at the VPS

Before anything your VPS serves is reachable by its name, your domain's DNS records need to point at the IP address GSP assigned to it.

If your domain is registered with GSP or with a registrar that lets you manage DNS yourself, create or update these records:

Record	Type	Value
example.com	A	203.0.113.10
www.example.com	A	203.0.113.10 (or a CNAME to example.com)
mail.example.com	A	203.0.113.10 (the host name mail clients connect to — see Chapter 4)
example.com	MX	10 example.com (so mail for @example.com is delivered to this VPS)

DNS changes are not instant — depending on your registrar and the previous record's TTL (time to live), it can take anywhere from a few minutes to 48 hours for the new records to be visible everywhere on the Internet. You can check propagation with `dig example.com` from another machine, or with any of the public "DNS checker" web tools.

> **Tip:** While you wait for DNS to propagate, you can still reach your VPS directly by its IP address — for example, `ssh youruser@203.0.113.10` — and you can test a web site before its domain resolves by adding a temporary line to your own computer's hosts file.

Step 2: Connect to Your VPS

Almost everything in this handbook assumes you can get a command-line session on your VPS. Rocky Linux 10 includes everything needed for this in the base install — the OpenSSH server is installed and running out of the box.

From macOS or Linux, open a terminal and run:

```
$ ssh youruser@example.com
```

From Windows 11, the built-in Terminal app includes an OpenSSH client; the same command works unchanged. (Older Windows versions may need to enable the optional "OpenSSH Client" feature first, or use a third-party client such as PuTTY.)

The first time you connect, SSH will show you the server's host key fingerprint and ask you to confirm it. GSP includes the current fingerprints for your VPS in your welcome email — compare them before typing "yes," since this confirmation is what protects you from connecting to an impostor server on a compromised network.

> **Tip:** Password authentication works out of the box, but we strongly recommend switching to SSH key pairs as soon as you're comfortable — see "SSH Hardening" in Chapter 10. Keys are both more convenient (no typing a password every time) and dramatically more resistant to automated attacks.

Once connected, you have a shell — on Rocky Linux, new accounts use `bash` by default, the same shell most Linux distributions and their documentation assume. Other shells (`tcsh`, `zsh`, `ksh`) are a single `dnf install` away if you prefer one of them.

To move files to and from the VPS, use **SFTP**, which runs over the same SSH connection and credentials — see Chapter 5.

Step 3: Learn the Linux Basics

Rocky Linux is an enterprise Linux distribution — binary-compatible with Red Hat Enterprise Linux — and if you've never used a Unix-like system before, a little orientation goes a long way. Chapter 1 covers:

- How the file system is laid out, and where your web files, mail, and configuration files live
- How to move around with `cd`, `ls`, and `pwd`, and how to read and write files with `cat`, `less`, and a text editor
- How file permissions work, and how to read the output of `ls -l`
- How to edit a configuration file using `nano` (the friendly default) or `vi`/`vim`

If you've administered another Linux distribution before, almost all of this will be familiar; if you're coming from a BSD or a different Unix, the concepts are identical and only a few details differ (for example, software you install with `dnf` keeps its configuration under `/etc`, alongside the rest of the system, rather than in a separate `/usr/local/etc` tree).

Step 4: Add Users and Virtual Hosts

If more than one person needs access to your VPS, or if you plan to host more than one web site on it, you'll want additional accounts and virtual hosts.

You can do both from the command line (`useradd`, and editing Apache's configuration directly — see Chapters 2 and 6), or from **Webmin**, the web-based control panel installed on your VPS. Webmin's **System → Users and Groups** module provides point-and-click account management, and its **Servers → Apache Webserver** module includes a guided "Create Virtual Host" form. Chapter 3 introduces Webmin in full; Chapter 6 covers virtual hosting in depth.

Step 5: Upload Your Web Files

With your account created, you're ready to put content on the site. The most common ways to get files onto your VPS are:

- **SFTP**, using a graphical client such as FileZilla or Cyberduck, or the `sftp` command
- **rsync over SSH**, ideal for keeping a local copy of a site in sync with the server
- **Webmin's File Manager** module, for occasional edits or uploads through your browser
- **Git**, if your site is built from a repository — clone it directly onto the VPS and use a deploy script or `git pull` to update it

By default, Apache serves files from `/var/www/html` for the main site, and from each virtual host's own document root for additional sites. Chapter 5 covers file transfer in detail, and Chapter 6 covers where to put files for each site you host.

Step 6: Configure Email

Every account on your VPS can send and receive email at `username@example.com` (and at any other domain you've configured — see Chapter 4). There are two ways to read that mail:

- **Roundcube webmail**, installed at `https://example.com/roundcube/` (or a subdomain such as `https://webmail.example.com/`, if you've set one up). Roundcube provides a full webmail interface — compose, folders, address book, filters, and an out-of-office autoresponder — from any browser, with nothing to configure on your own computer.
- **A desktop or mobile mail app** (Apple Mail, Outlook, Thunderbird, the built-in iOS/Android mail apps), configured to use IMAP and SMTP with the settings in Chapter 4.

Both methods talk to the same mailbox, so you can use Roundcube from a borrowed computer and your phone's mail app at the same time without anything getting out of sync.

Step 7: Watch Your Web Statistics

Once your site is live, you'll want to know who's visiting. A minimal Rocky Linux install doesn't include a statistics tool, but two are a single `dnf install` away (from the EPEL repository — see Appendix A):

- **GoAccess** is a fast, modern, terminal-and-browser log analyzer that can produce a live-updating HTML report from Apache's access log.
- **AWStats** is an older but still well-maintained tool that produces the kind of monthly-summary report many people are used to.

Chapter 9 shows how to install either tool, point it at your log files, and (optionally) publish its report as a password-protected page on your site.

Step 8: Go Beyond the Basics

Once the essentials are working, the rest of this handbook covers everything else you'll need as your site grows:

- **Chapter 7** — squeezing more out of Apache: rewrite rules, custom error pages, HTTPS best practices
- **Chapter 8** — running PHP applications and databases, the SELinux contexts they need, and the security habits that go with them
- **Chapter 9** — logs, backups, and keeping an eye on your VPS's health
- **Chapter 10** — keeping Rocky Linux and everything on it patched and locked down
- **Appendix A** — installing additional software with `dnf`, EPEL, and (when you need it) source RPMs
- **Appendix B** — a one-page command reference you can keep open in another tab
- **Appendix C** — where to go for help when this handbook doesn't cover it

Welcome to Rocky Linux, and welcome to your new VPS.

Chapter 1: Introducing Your Rocky Linux VPS

How the System Works

Your GSP VPS is a complete, private Rocky Linux 10 system running inside a virtualized server. Unlike shared hosting, where you share a single web server configuration with hundreds of other customers, your VPS gives you root-level control over the entire operating system: you decide what software runs, how it's configured, and who can access it. The trade-off is that more responsibility falls on you — but this handbook, Webmin, and GSP's support team are here to make that responsibility manageable.

"Out of the box," your VPS boots into a current, minimal Rocky Linux 10 installation. Rocky's philosophy — inherited from Red Hat Enterprise Linux, which it is built to be binary-compatible with — is a small, stable base that you extend with exactly the packages you need using the `dnf` package manager. A fresh VPS already includes:

- The base system: the Linux kernel, the `bash` shell, the core GNU utilities, OpenSSH (for remote login and file transfer), and the `dnf` package manager
- `firewalld`, the firewall, installed **and enabled** — with a default zone that already blocks most inbound traffic except SSH (see Chapter 10 to open the ports your services need)
- SELinux, running in **enforcing** mode by default — an extra layer of access control that confines services to what they're supposed to do (Chapters 8 and 10)
- Your administrative user account, created during provisioning and described in your welcome email

It does **not** yet include a web server, mail server, database, or control panel — these are exactly the pieces this handbook walks you through installing and configuring, generally with a single `dnf install` command followed by some configuration. Think of the rest of this handbook as a guided tour of turning a blank Rocky Linux 10 system into the web and mail server your site needs.

> **Note:** Two of those defaults — `firewalld` enabled and SELinux enforcing — are switched on from the very first boot. They occasionally surprise people coming from systems where the firewall and mandatory access control are opt-in: a service can be running perfectly yet still be unreachable (firewall) or unable to read its own files (SELinux). Both are worth keeping on; Chapters 8 and 10 explain how to work *with* them rather than turning them off.

Why a VPS? VPS vs. Virtual Hosting vs. Do-It-Yourself

Before diving into administration, it helps to understand what kind of hosting you have and why it is structured the way it is. There are three common ways to put a web site or mail server online, and a GSP VPS sits deliberately in the middle of them.

Virtual (shared) hosting is the simplest and cheapest option. Your site lives alongside dozens or hundreds of other customers on a single server, all sharing one copy of the

operating system, one web server, and one mail server. A control panel lets you upload files, add an email address, and little else. You never see a command line, never patch the OS, and never configure Apache — because you can't: those pieces are shared and locked down. This is wonderful until you need something the provider hasn't pre-approved: a specific PHP extension, a non-standard Apache module, a background process, a second daemon, or simply more isolation from a noisy neighbor whose runaway script is slowing down the whole machine.

The do-it-yourself approach is the opposite extreme: you rent or buy a physical server (or run one in your office), install the operating system, and own every layer top to bottom. You get total control — but also total responsibility. You are now in charge of hardware failures, power and cooling, network connectivity, off-site backups, physical security, and the OS. For most people the hardware and connectivity burden alone outweighs the control benefit, and a single failed disk at 3 a.m. becomes your problem.

A Virtual Private Server (VPS), which is what GSP provides, gives you the control of the do-it-yourself approach without the hardware burden of it. Your VPS is a complete, private Rocky Linux system — its own kernel, its own file system, its own root account — running as an isolated guest on GSP's cloud network. You get full root access and can install and configure anything Rocky Linux supports, exactly as if it were your own machine. What you don't have to deal with is the physical layer: GSP's hosting regions provide redundant power, cooling, and network paths, current-generation virtualized compute on SSD-backed storage, and automated, cross-region snapshots — so a hardware fault is GSP's problem, not yours. Those platform snapshots protect against hardware failure, but they don't replace keeping your own backups of your site and data; Chapter 9 explains how to set those up.

The trade-offs line up like this:

Concern	Virtual (shared) hosting	VPS (your GSP VPS)	Do-it-yourself server
Root / full control	No	**Yes**	Yes
Install any software	No (provider's menu only)	Yes	Yes
OS configuration & patching	Provider's job	**Your job** (this handbook)	Your job
Isolation from other customers	Low (shared OS)	**High** (private OS)	Total
Hardware, power, cooling, network	Provider	**Provider (GSP)**	**You**
Off-site backups	Usually provider	GSP snapshots + your own (Ch. 9)	You
Up-front cost & effort	Lowest	Moderate	Highest
Scales by	Moving to a bigger plan	Resizing the VPS	Buying hardware

When a VPS is the right choice. Choose a VPS when you have outgrown shared hosting — you need a particular software stack, want to run more than a simple web site (a mail server, a database, a custom application, a scheduled job), or want to be insulated from

other tenants — but you do not want to own and babysit physical hardware. That covers the large majority of small-business and developer workloads, which is exactly why this handbook exists: the rest of it is a guided tour of using that root access responsibly.

> **Note:** Everything in shared hosting is also possible on your VPS — a control panel (Webmin, Chapter 3), one-command installs (`dnf`, used throughout), and email addresses (Chapter 4). The difference is that on a VPS these are choices you make and can change, not limits someone else imposed.

Root User and Administrative User

Chapter "Getting Started" introduced the two account types your VPS uses; this section explains the relationship between them in more depth, since it underlies almost every other chapter.

root (UID 0) is Linux's superuser account. Every permission check the kernel performs is bypassed for root: it can read or modify any file regardless of ownership, bind to any network port, load kernel modules, and shut the machine down. Because of this, root is also the account most attackers target, and the account where a single typo can do the most damage — `rm -rf` typed in the wrong directory as root can destroy the entire system, where the same command as a normal user is limited to files that user owns.

Your administrative user is an ordinary account with one important property: it belongs to the `wheel` group. On Rocky Linux, the default `/etc/sudoers` policy grants members of `wheel` the ability to run commands as root with `sudo`, after confirming their own password. There is no need to know or share the **root password** at all.

In practice, this gives you a hierarchy:

```
root
  +-- members of "wheel" (your administrative user, and any
  |    colleagues you choose to add) — run commands as root via sudo
  |
  +-- ordinary users (mailbox-only accounts, SFTP-only
       accounts for clients, etc. — Chapter 2)
```

Day to day, you'll log in as your administrative user, do most of your work there, and reach for `sudo` only for the specific commands that genuinely need root — installing packages, editing files under `/etc`, and managing services.

> **Warning:** GSP sets `PermitRootLogin no` in the SSH server configuration on new VPS instances, meaning you cannot log in directly as root over SSH — you must log in as your administrative user and then use `sudo`. This is a security best practice (it ensures every root action is traceable to a named account) and we recommend leaving it in place. Chapter 10 explains the setting if you ever need to review it.

Administering the Server: SSH and SFTP

SSH (Secure Shell) is the encrypted protocol you use for every interactive command-line session with your VPS. It's provided by the `openssh-server` package (running as `sshd`, with the `ssh` client also installed) and is enabled by default.

A typical session looks like this:

```
$ ssh youruser@example.com
youruser@example.com's password:
Last login: Wed Jun 10 08:14:22 2026 from 198.51.100.7
[youruser@vps ~]$
```

Useful variations:

```
$ ssh -p 2222 youruser@example.com      # connect to a non-default port
$ ssh youruser@example.com 'uptime'     # run one command and exit
```

SFTP (SSH File Transfer Protocol) reuses the same connection, port, and credentials as SSH, but instead of giving you a shell, it gives you a file-transfer session — list directories, upload, download, rename, and delete files. Because it rides on top of SSH, there is nothing extra to install or open in the firewall. Chapter 5 covers SFTP clients and usage in detail.

> **Note:** Plain FTP and Telnet — both covered at length in older hosting handbooks — are not installed on your VPS and are not recommended. Both send credentials and data in clear text, meaning anyone on the same network segment (a coffee-shop Wi-Fi network, for instance) can read them. SSH and SFTP provide everything those protocols did, encrypted, using software that's already installed.

Connecting from Common Clients

Your computer	Recommended client
macOS	Built-in Terminal (`ssh`/`sftp` commands), or Cyberduck/FileZilla for a graphical SFTP view
Windows 11	Built-in Terminal (OpenSSH client), or WinSCP/FileZilla for graphical SFTP
Linux	Built-in terminal (`ssh`/`sftp`), or any graphical file manager with an "sftp://" location
iPad/iPhone/Android	Terminal apps such as Termius (SSH), or Working Copy / FE File Explorer (SFTP)

The Linux File System

Everything on your VPS — programs, configuration, your web site, your mail — is a file somewhere in a single tree rooted at `/`. There is no concept of separate drive letters as in Windows; external storage, if any, is mounted as a directory within this same tree.

A Tour of the Top-Level Directories

Path	Contains
/bin, /sbin	Essential commands. On Rocky Linux these are symbolic links into /usr/bin and /usr/sbin (the "usr merge")
/etc	Configuration for the whole system — both the base OS and software you install

	with dnf (e.g., /etc/ssh/sshd_config, /etc/httpd/)
/usr	The bulk of the installed system: programs (/usr/bin, /usr/sbin), libraries (/usr/lib64), and shared data (/usr/share/man)
/usr/local	Software you install by hand, outside dnf — empty on a fresh system; dnf packages never use it
/var	Frequently-changing data: logs (/var/log), the mail spool (/var/spool/mail), and web content (/var/www)
/var/www/html	Apache's default document root — files here are served at https://example.com/
/home	Personal directories for ordinary user accounts (often its own LVM volume)
/root	The root account's home directory
/tmp	Temporary files, cleared on a schedule and on reboot
/opt, /srv	Optional add-on software, and site-specific service data, respectively

The single most important habit to build is this: on Rocky Linux, configuration for the base system and for anything you add with `dnf` alike lives under `/etc`. When this handbook says "edit Apache's configuration," it means a file under `/etc/httpd/` — there is no separate `/usr/local/etc` tree to remember, as there is on some other Unix systems.

Navigating the File System

A handful of commands cover most day-to-day navigation:

```
$ pwd                  # print working (current) directory
$ cd /etc/httpd        # change directory
$ cd ~                 # go to your home directory
$ cd ..                # go up one level
$ ls                   # list files in the current directory
$ ls -la               # list all files, including hidden ones, with details
```

Filenames on Linux are case-sensitive (`Index.html` and `index.html` are different files), can contain almost any character, and traditionally avoid spaces — if you must use one, either quote the whole name (`"my file.txt"`) or escape the space (`my\ file.txt`).

A leading dot marks a hidden file or directory — for example, `.ssh` in your home directory holds your SSH keys and is hidden from a plain `ls` to reduce clutter, not for security (use `ls -a` to see it).

File Ownership and Permissions

Run `ls -l` in any directory and you'll see output like this:

```
$ ls -l
-rw-r--r--. 1 youruser youruser 1024 Jun  9 14:02 index.html
drwxr-xr-x. 2 youruser youruser   29 Jun  9 14:01 images
-rwxr-xr-x. 1 youruser youruser  256 Jun  8 09:11 backup.sh
```

The first column encodes the file's type and permissions as ten characters:

- The first character is `-` for a regular file, `d` for a directory, or `l` for a symbolic link.

- The next nine characters are three groups of three: permissions for the **owner**, the **group**, and **everyone else**. Each group is `r` (read), `w` (write), and `x` (execute), or `-` if that permission isn't granted.

So `-rw-r--r--` means: a regular file, the owner can read and write it, and the group and everyone else can only read it. `drwxr-xr-x` means: a directory, the owner can read/write/enter it, and everyone else can read and enter it but not modify it.

> **Note:** On a Rocky Linux system you'll notice a trailing `.` after the nine permission characters (as above). That dot signals that the file carries an SELinux security context (Chapters 8 and 10); a `+` in its place would mean a POSIX access-control list (ACL) is set. Ordinary permissions still work exactly as described — the dot is just telling you SELinux is also in the picture.

The third and fourth columns are the file's owner and group. The web server, for instance, runs as the `apache` user, which is why files it needs to read (or, for upload directories, write) often need to be owned by or readable by the `apache` group.

Changing Permissions and Ownership

```
$ chmod 644 index.html        # owner: read/write; group & others: read
$ chmod 755 backup.sh         # owner: read/write/execute; group & others: read/execute
$ chmod -R 755 images/        # apply recursively to a directory and its contents
# chown youruser:youruser file.txt    # change owner and group (root only)
```

The numeric form of `chmod` adds up read (4), write (2), and execute (1) for each of owner/group/other. `644` is therefore "owner can read and write (4+2=6), everyone else can only read (4)" — the standard permission for a web page. `755` is the equivalent for something that needs to be executable (a script or a directory you need to be able to enter).

> **Tip:** A directory needs its execute bit set for anyone to be able to `cd` into it or access files inside it by name, even if those files themselves are readable. If a web page exists but visitors get "Forbidden," check the permissions on every directory in its path, not just the file itself — and, on Rocky Linux, also check its SELinux context (Chapter 8), which is a second, independent reason Apache can be denied access to a file whose ordinary permissions look correct.

Editing Files

Sooner or later, every chapter in this handbook involves editing a configuration file. Rocky Linux ships the traditional `vi` editor in the base install, and the friendlier `nano` — usable with no prior knowledge — is a single `dnf install` away (and present already on many installs). Either one will do; the rest of this section covers both.

nano

If you've used `nano` before, Rocky's is the same program. Install it if it isn't already present:

```
# dnf install nano
$ nano /etc/httpd/conf/httpd.conf
```

nano lists its key commands along the bottom of the screen, where ^ means the Control key (so ^O is Ctrl+O). The ones you'll use constantly:

Keys	Action
Ctrl+O	Write out (save) the file — press Enter to confirm the name
Ctrl+X	Exit (prompts to save if there are unsaved changes)
Ctrl+W	Where is — search; Ctrl+W again repeats the search
Ctrl+\	Search **and** replace
Ctrl+K / Ctrl+U	Cut the current line / paste ("uncut") it back
Ctrl+A / Ctrl+E	Jump to start / end of the line
Ctrl+Y / Ctrl+V	Page up / page down
Ctrl+_	Go to a specific line (and column) number
Ctrl+G	Open the built-in help, which lists every command
Alt+U / Alt+E	Undo / redo the last change
Ctrl+C	Show the current cursor position (line, column)

A couple of options make nano more pleasant for editing configuration files. Turn them on for a single session from the command line:

```
$ nano -wl /etc/httpd/conf/httpd.conf   # -w = don't wrap long lines, -l = show line numbers
```

or, far better, make them permanent in a ~/.nanorc file in your home directory. Rocky's nano package already ships the **syntax-highlighting definitions** and the stock /etc/nanorc includes them, so opening, say, httpd.conf or a shell script colorizes keywords, strings, and comments automatically. A good starting ~/.nanorc:

```
## ~/.nanorc — per-user nano settings

set linenumbers      # show line numbers in the margin
set constantshow     # always show the cursor's line/column
set tabstospaces     # insert spaces instead of tab characters
set tabsize 4        # ...four of them per tab
set autoindent       # keep the previous line's indentation on Enter
set softwrap         # wrap long lines on screen without inserting newlines

# If syntax files aren't already included system-wide, add:
# include "/usr/share/nano/*.nanorc"
```

Confirm the highlighting definitions are present with ls /usr/share/nano/; if that directory is empty, the nano package installs them.

vi / vim

vi is the traditional Unix editor — always present on every Rocky Linux system, even a minimal one (where it is provided by the vim-minimal package) — and is extremely powerful once learned, though its modal design (separate "insert" and "command" modes) trips up newcomers. The single idea to internalize: you are normally in **command mode**,

where letters are editing commands, not text; you switch to **insert mode** to type, and press Esc to switch back.

```
$ vi /etc/httpd/conf/httpd.conf
```

Getting in and out of insert mode (from command mode):

Key	Effect
i / a	Insert before / after the cursor
I / A	Insert at the start / end of the current line
o / O	Open a new line below / above and insert
Esc	Leave insert mode, back to command mode

Saving and quitting (type these in command mode, beginning with a colon):

Command	Effect
:w	Write (save) the file
:wq or ZZ	Write and quit
:q	Quit (refuses if there are unsaved changes)
:q!	Quit and discard unsaved changes
:w newname	Save a copy under a different name

Moving around and editing (command mode):

Command	Effect
h j k l	Move left, down, up, right (arrow keys also work)
0 / $	Jump to start / end of line
gg / G	Jump to the first / last line of the file
:42	Jump to line 42
w / b	Forward / back one word
x / dd	Delete one character / the whole line
dw / D	Delete to end of word / end of line
yy / p	Yank (copy) a line / paste it after the cursor
u / Ctrl+R	Undo / redo
/text then n	Search forward for text, then jump to the next match
:%s/old/new/g	Replace every old with new in the whole file
:set number	Show line numbers (:set nonumber to hide)

> **Tip:** If you ever find yourself stuck in `vi` with no idea what mode you're in, press **Esc** a couple of times (returning you to command mode) and then type `:q!` and Enter to bail out without saving — then start over.

`dnf install vim-enhanced` provides the fuller `vim` (syntax highlighting, undo history, split windows, and more) while remaining compatible with every command above; its built-in tutorial, started by running `vimtutor`, is the fastest way to get comfortable.

> **Tip:** Whichever editor you choose, get in the habit of making a copy of an important configuration file before editing it — `cp httpd.conf httpd.conf.bak` takes a second and can save an evening if a change goes wrong and you need to revert quickly.

Choosing and Using a Shell

When you log in over SSH, the program that prints the prompt, reads what you type, and runs your commands is your shell. Rocky Linux gives new accounts `bash`, and you can install others; shells fall into two broad families that behave differently enough that mixing them up is a common source of "but that worked yesterday" confusion.

The **Bourne family** — `sh`, `bash`, `ksh`, `zsh` — descends from the original Unix shell and shares one scripting syntax (`if ... then ... fi`, `VAR=value`, `export VAR`). The **C-shell family** — `csh` and `tcsh` — uses a different, C-like syntax (`if (...) then ... endif`, `set var = value`, `setenv VAR value`) and is **not** script-compatible with the Bourne family. A script written for `sh` will not run under `csh`, and vice versa. This is why almost all system scripts begin with `#!/bin/sh` or `#!/bin/bash`: the Bourne family is the portable default — and on Rocky Linux it is also your login shell.

Shell	Family	Where it comes from	Best for
bash	Bourne	Base system (default)	The default login shell; scripts and everyday interactive use
sh	Bourne	Base (a symlink to bash)	POSIX mode — what `#!/bin/sh` scripts run under
tcsh	C shell	dnf install tcsh	C-shell interactive use; history, completion, editing
zsh	Bourne	dnf install zsh	Interactive power-user shell (themes, smart completion)
ksh	Bourne	dnf install ksh	The Korn shell; powerful, POSIX-compatible scripting
dash	Bourne	dnf install dash	A tiny, fast POSIX shell, popular for scripts
tclsh	Tcl	dnf install tcl	Not a login shell — the Tcl scripting-language interpreter

> **Note:** `tclsh` is listed because people sometimes reach for it as a "shell," but it is really an interactive interpreter for the Tcl language rather than a command shell you would log in with: it runs Tcl scripts (`tclsh script.tcl`) and gives a `%` prompt for typing Tcl interactively. You would not set it as a user's login shell.

Seeing and changing your shell. Your current shell is named in the `$SHELL` variable and recorded in your account record:

```
$ echo $SHELL
/bin/bash
$ cat /etc/shells          # the list of shells allowed as login shells
```

A shell only appears in `/etc/shells` once its package is installed (the Rocky packages add themselves automatically). To switch your own login shell to one that is listed:

```
$ chsh -s /bin/zsh                # change your own shell (logout/in to take effect)
# usermod -s /bin/zsh jsmith      # or, as root, change another account's
```

Navigating, whichever shell you choose. The everyday navigation commands are the same across all of them, because they are separate programs, not shell features: `cd` to change directory, `pwd` to print the working directory, `ls` to list files, and Tab to auto-complete a name (`bash`, `zsh`, and `tcsh` all support interactive completion). What differs between the families is mostly the "house style" commands:

Task	Bourne family (sh/bash/ksh/zsh)	C-shell family (csh/tcsh)
Set a variable	VAR=value	set var = value
Export to programs	export VAR=value	setenv VAR value
List environment	env or export	setenv or env
Define an alias	alias ll='ls -l'	alias ll 'ls -l'
Run last command again	!!	!!
Startup file (login)	~/.bash_profile, ~/.zprofile	~/.login
Startup file (each shell)	~/.bashrc, ~/.zshrc	~/.cshrc or ~/.tcshrc
Command history	history	history

> **Tip:** Put your aliases and prompt customizations in the per-shell startup file (`~/.bashrc`, `~/.zshrc`, or `~/.cshrc`) so they apply to every new shell, and put things that should run only once at login (like setting `PATH` or a welcome banner) in the login file. On Rocky Linux, `~/.bash_profile` typically sources `~/.bashrc` for you, and system-wide defaults live in `/etc/profile` and `/etc/bashrc`. If you set a variable and a later command "can't see" it, you probably set it but forgot to `export` (Bourne) or `setenv` (C shell) it so child programs inherit it.

Where to Go Next

With a shell session open and the basics of files and permissions under your belt, Chapter 2 covers how to create and manage user accounts — both for yourself and for anyone else who needs access to this VPS.

Chapter 2: Users and Accounts

Most VPS owners eventually need more than one account: a colleague who needs their own login, a client who should only be able to upload files to their own web site, or a dedicated account that owns a particular application. This chapter covers the categories of account your VPS uses and the commands for creating, changing, and removing them — both at the command line and, where it's easier, through Webmin.

Categories of Account

root and **your administrative user** were introduced in Chapter 1. Two further categories will come up as your VPS grows:

Mail-only / virtual users are accounts that exist primarily so a person can send and receive email at `name@example.com`, without necessarily needing shell access to the VPS at all. Chapter 4 covers how Sendmail and Dovecot use these accounts.

Restricted or SFTP-only users are accounts created for people — often clients — who should be able to upload files to one specific directory (typically a virtual host's document root) and nothing else. Chapter 5 shows how to confine such an account with `ChrootDirectory` in `sshd_config`.

All of these are, underneath, ordinary Linux user accounts; what differs is which groups they belong to, what shell they're given, and how the services on your VPS are configured to treat them.

Switching to Root: sudo and su

The standard way to run a command as root on Rocky Linux is `sudo`. Because your administrative user is in the `wheel` group, and the default `/etc/sudoers` policy grants `wheel` full access, it works out of the box — you authenticate with **your own** password, not root's:

```
$ sudo dnf install nano
[sudo] password for youruser:
```

Only the single command runs with elevated privileges, and `sudo` remembers your authorization for a few minutes afterward so you aren't prompted on every command. For a handful of commands in a row, start an interactive root shell instead, and `exit` (or Ctrl+D) when done:

```
$ sudo -i            # a full root login shell
# ...
# exit
```

The older `su` is also available: `su -` prompts for the **root** password and gives you a root shell. On a VPS where you administer with `sudo`, you may never need it — but it's there if you prefer it or are following older documentation.

To grant a colleague narrower access — say, the ability to manage packages and services but nothing else — don't edit `/etc/sudoers` directly. Run `visudo`, which checks your syntax before saving (a mistake in `sudoers` can lock everyone out of root), and prefer a drop-in file under `/etc/sudoers.d/`:

```
# visudo -f /etc/sudoers.d/colleague

# /etc/sudoers.d/colleague
colleague ALL=(root) /usr/bin/dnf, /usr/bin/systemctl
```

> **Note:** If you've used `doas` on BSD systems, `sudo` plays the same role and is the Rocky Linux default. `doas` itself is available from the EPEL repository (`dnf install opendoas`) if you prefer its smaller configuration file, but most Rocky installations standardize on `sudo` so that `/etc/sudoers` (and `/etc/sudoers.d/`) is the single source of truth for who can do what.

Creating New User Accounts

On Linux the workhorse is `useradd`, which creates an account non-interactively from the options you give it. To create a normal login with a home directory and a Bash shell, then set its password:

```
# useradd -m -s /bin/bash -c "Jane Smith" jsmith
# passwd jsmith
```

`useradd` creates the home directory (`-m`, the default on Rocky), assigns the shell (`-s`), and stores the person's full name in the GECOS comment field (`-c`). The account is created locked until you set a password with `passwd`. To make the account an administrator — able to use `sudo` — add it to the `wheel` group at creation time, or afterward:

```
# useradd -m -s /bin/bash -G wheel -c "Jane Smith" jsmith   # admin from the start
# usermod -aG wheel jsmith                                  # ...or grant it later
```

> **Note:** There is no interactive `adduser` walk-through on Rocky Linux as there is on some systems — `adduser` is simply a symbolic link to `useradd`. If you'd like a guided, fill-in-the-blanks form for new accounts, Webmin's **System → Users and Groups** module (Chapter 3) provides exactly that.

Common `useradd` options:

Option	Meaning
-c comment	"Full name" / GECOS comment
-d dir	Home directory (defaults to /home/<name>)
-m	Create the home directory (the default on Rocky)
-M	Do *not* create a home directory (e.g., a mail-only account)
-s shell	Login shell (use /sbin/nologin for no shell access)
-G group,...	Additional (secondary) group memberships
-g group	Primary login group

-u uid	Specific numeric user ID

Editing Existing Accounts

Linux spreads the job that one tool might do elsewhere across a few focused commands:

Command	Changes
usermod -s /bin/zsh jsmith	The login shell (or `chsh -s` for your own account)
usermod -aG apache jsmith	Add the account to a secondary group (the `-a` is essential — without it, `-G` *replaces* all secondary groups)
usermod -L / -U jsmith	Lock / unlock the account's password
chfn jsmith	The full name and contact (GECOS) fields
chage -l jsmith	View password-aging settings; `chage jsmith` to edit them interactively
passwd jsmith	Set or change the account's password (root only; `passwd` alone changes your own)

As an ordinary user, `chsh` (change shell) and `chfn` (change finger/contact info) let you edit the small set of fields you're allowed to change on your own account without root.

Disabling and Removing Accounts

To temporarily lock an account — for example, while a client's invoice is overdue — without deleting their files or mailbox:

```
# usermod -L jsmith          # lock (equivalently: passwd -l jsmith)
```

and to restore access later:

```
# usermod -U jsmith          # unlock (equivalently: passwd -u jsmith)
```

A locked account can't log in with a password (and, if you also want to block key-based and mail access, set the shell to `/sbin/nologin` and/or expire the account with `chage -E 1 jsmith`); its files, mail, and cron jobs are untouched.

To remove an account entirely, including its home directory and mail spool:

```
# userdel -r jsmith
```

Without `-r`, `userdel` removes the account but leaves its home directory and files in place.

> **Warning:** Removing an account does not automatically remove files that account owns outside its home directory — for example, web files in a virtual host's document root, or cron jobs referencing that user's scripts. Check for these (`find / -user jsmith` will list them) before, or instead of, running `userdel` if the account "owns" a web site you want to keep online.

Groups

A **group** is a named set of users that can be granted permissions collectively — for instance, every account that should be able to edit a particular web site's files. View a user's groups with:

```
$ id jsmith
uid=1002(jsmith) gid=1002(jsmith) groups=1002(jsmith),10(wheel),48(apache)
```

To create a new group and add members:

```
# groupadd webteam
# gpasswd -a jsmith webteam        # add jsmith to webteam
# usermod -aG webteam asmith      # another way to add a member
```

A common pattern for a shared web site: create a group, add everyone who edits that site to it, set the document root's group ownership to that group, and make the directory and files group-writable (`chmod 664` for files, `chmod 2775` for directories — the leading `2` sets the "setgid" bit so new files inherit the directory's group automatically).

Resource Limits

Linux controls per-user resource limits — maximum processes, open files, locked memory, and so on — through PAM's "limits" module, configured in `/etc/security/limits.conf` and drop-in files under `/etc/security/limits.d/`. Most accounts need no changes; the defaults are generous enough for normal use. If you host several independent clients and want to cap how much one account's runaway script can consume, add lines like these:

```
# /etc/security/limits.d/90-clients.conf
# <domain>   <type>  <item>   <value>
@clients     hard    nproc    200       # max processes for members of the clients group
@clients     hard    nofile   4096      # max open files
jsmith       hard    cpu      30        # max CPU minutes per process
```

A user sees their current limits with `ulimit -a`. The limits take effect at the user's next login. For limiting a *service* rather than a person, systemd offers per-unit controls (`MemoryMax=`, `TasksMax=`, `CPUQuota=`) in the unit's configuration, which is often the better lever for a daemon.

> **Note:** For disk-usage limits, Rocky's default file system is XFS on LVM. XFS supports **project quotas** (`xfs_quota`), which cap the size of a directory tree — handy for limiting one client's site. Traditional per-user quotas are also available via the `quota` package (`dnf install quota`, then `edquota` / `setquota`). Either requires enabling the appropriate mount option; see Chapter 9 for more on disks and LVM.

Important Commands and Files

Command / File	Purpose
useradd / usermod / userdel	Create / modify / remove accounts

passwd	Change a password
chage	View and set password-aging and account expiry
chsh / chfn	Change login shell / contact (GECOS) fields
groupadd / gpasswd / groupdel	Manage groups and their membership
id	Show a user's UID, GID, and group memberships
sudo / su	Run a command as root / switch to a root shell
visudo	Safely edit the sudo policy (validates before saving)
/etc/passwd	Account database (read with cat; never edit directly)
/etc/shadow	Encrypted passwords and aging info (root-only)
/etc/group	Group membership database
/etc/sudoers, /etc/sudoers.d/	Who may run what as root via sudo
/etc/security/limits.conf	Per-user resource limits

Where to Go Next

With accounts in place, Chapter 3 introduces **Webmin**, the browser-based control panel installed on every GSP VPS, which provides a graphical front end for many of the tasks in this chapter (and most of the rest of this handbook).

Chapter 3: Webmin — Your Web-Based Control Panel

GSP installs **Webmin**, a mature, actively-developed, open-source control panel that covers user management, file management, mail configuration, and web server configuration — and a great deal more, since it's developed for general Unix system administration rather than for one hosting product.

This chapter covers installing and securing Webmin, then takes a tour of the modules you'll use most. Webmin's interface changes gradually over time as new versions are released, so treat the screenshots-in-words below as a guide to *where things are* rather than an exact pixel-for-pixel description.

Installing Webmin

Webmin isn't in the standard Rocky repositories, but the project publishes its own repository and a setup script that adds it. Run the setup script once, then install with `dnf` as usual:

```
# curl -o setup-repos.sh https://raw.githubusercontent.com/webmin/webmin/master/setup-repos.sh
# sh setup-repos.sh
# dnf install webmin
```

The package installs Webmin under `/etc/webmin` (configuration) and `/usr/libexec/webmin` (program files), and registers a `systemd` service. Enable it to start at boot and start it now, in one command:

```
# systemctl enable --now webmin
```

By default, Webmin runs its own lightweight web server (`miniserv`) on **port 10000**, using HTTPS with a self-signed certificate it generates on first start. Connect to it at:

```
https://example.com:10000/
```

> **Note:** Because `firewalld` is enabled by default on Rocky Linux, port 10000 is closed to the outside world until you open it. The most secure option is to leave it closed and reach Webmin over an SSH tunnel (see "Securing Access," below); if you'd rather open it, the firewall commands are in Chapter 10.

Your browser will warn you that the certificate isn't signed by a trusted authority — this is expected for a self-signed certificate. You can either accept the warning (the connection is still encrypted; it's the identity verification that's missing) or replace the certificate with a Let's Encrypt certificate for your domain, which removes the warning. Webmin's own **Webmin Configuration → SSL Encryption** module can do this once you have a certificate from Chapter 6's TLS instructions, or you can point it at the same certificate files Apache uses.

Log in as `root` with the root password, or — better on a VPS where root SSH is disabled — add a Webmin login mapped to your administrative user under **Webmin → Webmin Users**, so the people who manage the server each have their own named Webmin account.

Securing Access to Webmin

Because Webmin can do almost anything root can do, treat access to it with the same care as SSH access.

Restrict who can log in. Under **Webmin → Webmin Users**, you can create Webmin-specific logins with access limited to particular modules — useful if, say, a colleague should be able to manage email through Webmin but shouldn't be able to touch the firewall or user accounts.

Firewall port 10000. If you (and any colleagues) always connect from a small set of known IP addresses, restricting port 10000 to those addresses with `firewalld` (Chapter 10) means the Webmin login page isn't exposed to the entire Internet at all. If your IP address changes frequently, an alternative is to leave port 10000 closed entirely and reach Webmin over an SSH tunnel:

```
$ ssh -L 10000:localhost:10000 youruser@example.com
```

then browse to `https://localhost:10000/` on your own machine.

Keep it updated. `dnf upgrade webmin` periodically pulls in security fixes, the same as any other package — see Chapter 10 for a routine that covers your whole VPS.

> **Warning:** Treat the Webmin login the same as the root password: anyone who can authenticate to Webmin can, through one module or another, become root. Don't share this login casually, and remove Webmin Users for colleagues who no longer need access.

A Tour of Webmin

Webmin organizes its modules into categories shown in the left-hand menu, with the modules in the selected category listed beneath it. The categories most relevant to this handbook are **System**, **Servers**, **Networking**, and **Others**.

System → Users and Groups

This module is the graphical equivalent of Chapter 2's `useradd`, `usermod`, and friends. The user list shows every account, its UID, primary group, home directory, and shell; clicking a username opens an edit form covering the same fields, plus group memberships as checkboxes. "Create a new user" provides the same form for new accounts, including an option to copy a "skeleton" home directory and to set an initial password — the closest thing Rocky has to an interactive `adduser`.

The companion **Groups** tab lists and edits `/etc/group` — useful for the shared-web-team group described in Chapter 2 without needing to remember `gpasswd` syntax.

Others → File Manager (Filemin)

Filemin provides a two-pane, drag-and-drop file browser for any directory your Webmin login can access — typically your home directory and your sites' document roots. From here you can:

- Upload and download files (handy for a quick fix from a borrowed computer when you don't have an SFTP client installed)
- Create, rename, move, copy, and delete files and directories
- Edit text files in a browser-based code editor with syntax highlighting
- Change permissions and ownership with a dialog instead of remembering `chmod` numbers
- Extract and create `.zip` / `.tar.gz` archives

This is Webmin's browser-based file manager.

System → Software Package Updates

This module is a graphical front end to `dnf`. It lists installed packages, flags ones with available updates (cross-referencing the same security errata that `dnf updateinfo` uses — see Chapter 10), and lets you install new packages by searching the Rocky repositories. Running updates through this module is equivalent to `dnf upgrade` at the command line, with a confirmation screen showing exactly what will change first.

System → Bootup and Shutdown

This module lists every `systemd` service the VPS knows about, shows whether each is enabled to start at boot, and lets you start, stop, restart, enable, or disable a service with a click. It's the graphical equivalent of:

```
# systemctl enable --now httpd
```

When a chapter in this handbook says "enable and start the service," you can do either the command-line version shown or use this module.

Servers → Apache Webserver

Once Apache is installed (Chapter 6), this module provides a structured editor for `httpd.conf` and any included configuration files: a list of configured virtual hosts, each with its own settings for document root, server name, log files, and SSL; a directory-by-directory permissions editor; and a module-management screen for enabling Apache modules like `mod_rewrite` and `mod_ssl`. The "Create Virtual Host" wizard is the fastest way to add a new site once DNS for its domain points at your VPS.

Servers → Sendmail Mail Server

Once Sendmail is installed and configured (Chapter 4), this module edits the `.mc` configuration, the access database (for blocking or allowing senders), virtual user/alias tables, and mail queue contents — all areas that previously required hand-editing

`sendmail.cf` or using `m4` directly. The companion **Servers → Dovecot IMAP/POP3 Server** module (where available) covers mailbox-side configuration.

System → Disk and Network Filesystems

Shows mounted filesystems, their type (XFS or ext4) and available space — a quick way to check whether your VPS is running low on disk without remembering `df -h`. Because Rocky's default layout is LVM, the related **System → Logical Volume Management** module is where you'd extend a volume or create the LVM snapshot used for backups in Chapter 9.

Networking → Linux Firewall

This module edits the firewall. We recommend understanding `firewalld`'s zone-and-service model first (Chapter 10) before relying on a graphical editor for it — firewall mistakes can lock you out of the VPS entirely, and it's important to know how to fix the rules directly (or from a recovery console) if that happens.

Webmin → Webmin Configuration

This is Webmin's own settings: which categories and modules are visible, the port and SSL certificate `miniserv` uses, language and theme, and (as mentioned above) **Webmin Users**, for creating logins with restricted module access.

Logging Out

Webmin sessions persist via a browser cookie. The Logout link, usually near your username in the interface, ends the session immediately — worth doing if you've been using Webmin from a shared or public computer.

Where to Go Next

With Webmin installed, the next several chapters return to specific services — starting with email in Chapter 4, the area where Webmin's modules save the most hand-editing of configuration files.

Chapter 4: The Mail Server and Roundcube Webmail

Email is, for many GSP customers, the single most important service their VPS provides — and it's also the area where "out of the box" Rocky Linux differs most from what a hosting environment needs. This chapter explains that gap, walks through closing it with Sendmail and Dovecot, and then covers Roundcube, the browser-based webmail interface.

Email Architecture: MTA, Mailbox Store, and Mail Clients

Three separate jobs make up a working mail system:

- A **Mail Transfer Agent (MTA)** accepts mail from the outside world for your domains, and accepts mail from local users to send out. This is **Sendmail** in this handbook.
- A **mailbox store and retrieval server** holds delivered mail and lets users fetch it over IMAP or POP3. This is **Dovecot**.
- **Mail clients** — Roundcube (webmail) and/or desktop and mobile apps — connect to the mailbox store to read mail, and to the MTA (or Dovecot's submission service) to send it.

Why Rocky Linux's Default Isn't Enough

A fresh Rocky Linux system ships with **Postfix** installed and running, but configured only to listen on the loopback address (`127.0.0.1`). In that default state it does a useful but minimal job: it hands outgoing mail from local programs (cron job output, system alerts, `dnf` notifications) to an upstream relay, and accepts mail addressed to local accounts. It is not configured to accept mail from the Internet for your domains, knows nothing about multiple hosted domains or virtual mailboxes, and provides no IMAP or POP3 service for clients to read mail. That is enough for a system that only needs to send the occasional notification — but a VPS that hosts `@example.com` mailboxes for you and your visitors needs a real, externally-reachable mail server and a mailbox store.

This handbook builds that mail server on **Sendmail** (the MTA) and **Dovecot** (the IMAP/POP3 mailbox store). Both are packaged for Rocky Linux:

```
# dnf install sendmail sendmail-cf dovecot dovecot-pigeonhole
```

`sendmail-cf` provides the `m4` macro files you need to regenerate Sendmail's configuration, and `dovecot-pigeonhole` provides the Sieve filtering plugin used later for Roundcube's filters and autoresponder. The rest of this chapter assumes all of these are installed.

Setting Up Sendmail

Switching the System Mailer from Postfix to Sendmail

Rocky Linux uses the `alternatives` system to decide which program the generic `/usr/sbin/sendmail`, `mailq`, and `newaliases` commands actually run — the Linux equivalent of a "system mailer" switch. After installing the Sendmail package, turn off Postfix, point the `mta` alternative at Sendmail, and enable Sendmail:

```
# systemctl disable --now postfix
# alternatives --set mta /usr/sbin/sendmail.sendmail
# systemctl enable --now sendmail
```

Run `alternatives --config mta` instead if you'd rather pick from a menu. You can confirm the switch took effect with `alternatives --display mta`, which shows Sendmail as the current selection.

The .mc File and local-host-names

Sendmail's configuration is generated from a compact `.mc` ("m4 configuration") file using the `m4` macro processor; the generated output is the much longer `sendmail.cf` that Sendmail actually reads. On Rocky Linux these live in `/etc/mail/`, with a `Makefile` that wraps the `m4` invocation — so after editing `/etc/mail/sendmail.mc` you simply run `make` in that directory and restart:

```
# cd /etc/mail
# vi sendmail.mc
# make
# systemctl restart sendmail
```

> **Warning:** By default, Rocky's `sendmail.mc` restricts the listening daemon to the loopback address: its `DAEMON_OPTIONS` line includes `Addr=127.0.0.1`. Until you change that to `Addr=0.0.0.0` (or remove the clause), Sendmail will **not** accept mail from the Internet. Edit that line, then `make` and restart. This single setting is the most common reason a freshly-installed Sendmail "won't receive mail."

For a VPS hosting mail for `example.com` and any additional domains, list every domain that should be treated as "local" (i.e., mail to `anyone@thatdomain` is for this server, not to be relayed elsewhere) in `/etc/mail/local-host-names`, one per line:

```
example.com
mail.example.com
anotherdomain.com
```

After changing this file, restart Sendmail (or `make` again if you also touched `sendmail.mc`) to reload it.

Aliases

`/etc/aliases` maps one address to one or more destinations. Three common patterns:

```
# Forward one address to another mailbox
webmaster:      youruser

# Forward to multiple recipients (a tiny mailing list)
sales:          alice,bob

# Forward to an external address
postmaster:     youruser@gmail.com
```

After editing this file, rebuild the database it's compiled into:

```
# newaliases
```

For a larger mailing list, point an alias at a text file containing one address per line using the `:include:` syntax:

```
announce:       :include:/etc/mail/lists/announce.txt
```

Anyone can be added or removed from the list by editing `announce.txt` — no `newaliases` needed for changes to the included file itself, only if the alias line changes.

Virtual Domains and Virtual Users

If your VPS hosts mail for more than one domain, `virtusertable` maps full email addresses (or whole domains) to local mailboxes. Make sure the `virtusertable` feature is enabled in `sendmail.mc` (it is, in Rocky's default), then edit `/etc/mail/virtusertable`:

```
# /etc/mail/virtusertable
sales@anotherdomain.com        alice
@anotherdomain.com             jsmith
```

The first line routes one specific address to alice's mailbox; the second is a **catch-all** that routes everything else `@anotherdomain.com` to jsmith. Catch-alls are convenient but also the single biggest source of spam landing in a mailbox — consider whether you really want one before adding it.

After editing, rebuild the database and restart:

```
# make -C /etc/mail && systemctl restart sendmail
```

Aliases vs. Virtual Aliases (valiases)

Newcomers — especially anyone arriving from a cPanel-style control panel, where the two files are literally named `aliases` and `valiases` — are often unsure which of these two mechanisms to use. They solve related but different problems, and the distinction is worth getting straight:

`aliases` (the system alias file, `/etc/aliases`) rewrites the **local part** of an address only — the bit before the `@`. An entry like `webmaster: youruser` means "mail to webmaster (at any local domain this server accepts) goes to youruser." It has no concept of which domain the mail was addressed to, so it is the wrong tool when the same local part must go to different people depending on the domain. It is the right tool for server-wide roles (postmaster, abuse, root), simple forwards, and small `:include:` mailing lists.

`virtusertable` (the virtual user table — the "valiases" equivalent) rewrites the **whole address**, domain included. An entry maps `sales@anotherdomain.com` to a specific mailbox, independently of what `sales@example.com` does. This is what makes genuine multi-domain hosting possible: each domain gets its own namespace of addresses, and you can point `info@` at different people for different domains, or catch-all an entire domain.

	aliases	virtusertable ("valiases")
Matches on	Local part only (name)	Full address (name@domain) or whole domain (@domain)
Domain-aware?	No	Yes
Typical use	postmaster, root, forwards, small lists	Per-domain mailboxes, multi-domain hosting, catch-alls
File	/etc/aliases	/etc/mail/virtusertable
Rebuild after editing	newaliases	make -C /etc/mail (then restart)

The two work **together**, and Sendmail applies `virtusertable` first: a message to `sales@anotherdomain.com` is matched there and rewritten to, say, the local user alice — and that result can then itself be an `aliases` entry that forwards alice somewhere else. A worked example covering both files at once:

```
# /etc/mail/virtusertable  — domain-aware (the "valiases")
info@example.com           youruser
info@anotherdomain.com     alice
sales@anotherdomain.com    sales-team        # -> an aliases entry, below
@anotherdomain.com         alice             # catch-all for the whole domain

# /etc/aliases — local-part only, expands the group
sales-team:                alice, bob, carol
```

Here `info@` resolves to two different people depending on the domain (only `virtusertable` can do that), while `sales-team` is a reusable group defined once in `aliases`. Remember the two different rebuild steps: `newaliases` after touching `/etc/aliases`, and `make -C /etc/mail` after touching `virtusertable`.

Access Control and Basic Spam Defenses

`/etc/mail/access` controls which hosts and addresses Sendmail will relay for, accept from, or reject:

```
# /etc/mail/access
spammer@example.net            REJECT
203.0.113.99                   REJECT
example.com                    RELAY
```

`RELAY` should only be used for hosts and networks you trust (such as your own office IP, if you send mail through this server from a desktop client without authenticating) — an open relay is quickly discovered and abused by spammers, and most blocklists will list your VPS within hours if that happens.

Like the other tables, rebuild after editing:

```
# make -C /etc/mail && systemctl restart sendmail
```

For more substantial spam and virus filtering, `dnf install rspamd` (a modern content-filtering milter, from EPEL) integrates with Sendmail via its milter interface and is a

common addition once your basic mail flow is working — see "For More Information" at the end of this chapter.

Autoresponders ("Vacation" Messages)

The classic Unix `vacation` program works through a user's `~/.forward` file:

```
\jsmith, "|/usr/bin/vacation jsmith"
```

The leading `\jsmith` ensures the message is still delivered normally in addition to triggering the autoreply. Each recipient is sent at most one autoreply per period (configurable, default about a week), to avoid mail loops between two vacationing addresses.

For mailboxes accessed primarily through Roundcube, the **Sieve-based vacation** approach described later in this chapter is usually more convenient, since users can turn it on and off themselves from the webmail interface without shell access — and it doesn't depend on a separate `vacation` binary being installed.

Setting Up Dovecot

Dovecot provides the IMAP and POP3 service that Roundcube and desktop/mobile clients connect to. Its configuration lives under `/etc/dovecot/`, split across `dovecot.conf` and a `conf.d/` directory of topic-specific files. Key settings for a typical GSP VPS:

Mailbox format and location (`conf.d/10-mail.conf`):

```
mail_location = maildir:~/Maildir
```

Maildir stores each message as an individual file, which is more robust against corruption than the older single-file mbox format and is what Roundcube and most modern clients expect.

Protocols (`dovecot.conf`):

```
protocols = imap pop3 lmtp sieve
```

Including `sieve` enables the Pigeonhole plugin used later for Roundcube's filters and autoresponder.

Authentication (`conf.d/10-auth.conf`): for accounts that are also Linux system users (the simplest setup, and the default this handbook assumes), Dovecot can authenticate against the system's own user database:

```
auth_mechanisms = plain login
!include auth-system.conf.ext
```

TLS (`conf.d/10-ssl.conf`): point Dovecot at the same certificate Apache uses (Chapter 6 covers obtaining one with `certbot`):

```
ssl = required
ssl_cert = </etc/letsencrypt/live/example.com/fullchain.pem
ssl_key = </etc/letsencrypt/live/example.com/privkey.pem
```

Enable and start:

```
# systemctl enable --now dovecot
```

A quick test from the VPS itself:

```
$ openssl s_client -connect localhost:993 -quiet
```

A successful connection prints the certificate chain followed by `* OK` and Dovecot's banner.

> **Note:** If Dovecot logs a permission error reading the certificate, the cause on Rocky is usually SELinux rather than file permissions: certificates under `/etc/letsencrypt/` may not carry the label Dovecot is allowed to read. Either store the cert where Dovecot expects it (`/etc/pki/dovecot/`) or restore the right context with `restorecon -Rv /etc/letsencrypt/live /etc/letsencrypt/archive`. Chapter 10 covers SELinux contexts in general.

A few more settings come up often enough to mention. Each goes in the matching file under `conf.d/`, and a `systemctl reload dovecot` applies it:

```
# conf.d/20-imap.conf — cap simultaneous connections from one IP
protocol imap {
  mail_max_userip_connections = 20
}

# conf.d/20-lmtp.conf — enable Sieve so Roundcube filters/vacation work
protocol lmtp {
  mail_plugins = $mail_plugins sieve
}

# conf.d/10-logging.conf — log every login (handy when chasing a mail problem)
auth_verbose = yes
```

Two commands are invaluable when something doesn't connect:

```
# doveconf -n            # print the active config, omitting unchanged defaults
# doveadm who            # list who is currently logged in over IMAP/POP3
```

`doveconf -n` is the first thing to run (and to paste, if you contact support) when Dovecot behaves unexpectedly — it shows exactly the settings in force, with all the commented-out defaults stripped away, so a stray override is easy to spot.

SMTP Authentication

For users to send mail through your server from outside (their phone on a cellular network, for instance), Sendmail's submission service (port 587) needs to authenticate them — otherwise it would either reject their mail or, worse, relay for anyone. On Rocky Linux the standard approach is **Cyrus SASL** with the `saslauthd` daemon, so that the same

username and password works for both sending (Sendmail) and receiving (Dovecot) mail. This involves:

- `dnf install cyrus-sasl cyrus-sasl-plain`, then enabling `saslauthd` (`systemctl enable --now saslauthd`); by default it authenticates against the system accounts via PAM (`MECH=pam` in `/etc/sysconfig/saslauthd`)
- Enabling `AUTH_PLAIN` and `AUTH_LOGIN` in Sendmail's `.mc` via `TRUST_AUTH_MECH` and `confAUTH_MECHANISMS`, and uncommenting the submission (MSA, port 587) `DAEMON_OPTIONS` line
- Pointing `/usr/lib64/sasl2/Sendmail.conf` at saslauthd with `pwcheck_method: saslauthd`

This is one of the more fiddly parts of a mail server, and Webmin's **Sendmail Mail Server** module includes an "SMTP Authentication" screen that handles the `.mc` changes for you — we recommend using it rather than hand-editing `sendmail.cf`, and consulting the Sendmail documentation (linked at the end of this chapter) if something doesn't connect on the first try.

Filtering Spam with SpamAssassin

Once mail is flowing, the next thing every server owner wants is less spam. **SpamAssassin** is the long-standing, open-source content filter: it scores each message against hundreds of rules (suspicious phrasing, forged headers, known-bad URLs, blocklist hits, and — once trained — a Bayesian statistical filter) and tags anything over a threshold so it can be filed into a Junk folder or rejected outright. Setting it up has two halves: wiring it into the system, and tuning SpamAssassin's own configuration.

At the system (OS) level

Install SpamAssassin and the small "milter" that lets Sendmail hand each message to it (both are in EPEL — see Appendix A):

```
# dnf install spamassassin spamass-milter
```

Pull down the current rule set (SpamAssassin ships with almost none, on purpose, so they stay fresh) and run it as a fast background daemon (`spamd`) rather than starting the Perl interpreter for every message:

```
# sa-update                       # download the latest rules
# systemctl enable --now spamassassin    # the spamd daemon
```

Then enable the milter and tell Sendmail to consult it. Start the milter daemon:

```
# systemctl enable --now spamass-milter
```

and add the milter to Sendmail's `.mc`, then rebuild:

```
INPUT_MAIL_FILTER(`spamassassin', `S=local:/run/spamass-milter/spamass-milter.sock, F=,
T=C:15m;S:4m;R:4m;E:10m')dnl
```

```
# make -C /etc/mail && systemctl restart sendmail
```

Finally, keep the rules current automatically — stale rules are far less effective. Add a nightly `sa-update` to root's crontab (`crontab -e`; cron is covered in Chapter 9):

```
17 4 * * *   /usr/bin/sa-update && /usr/bin/systemctl reload spamassassin
```

> **Note:** SpamAssassin scoring is CPU- and memory-hungry on a small VPS. Running it as `spamd` (above) keeps it from spawning a fresh Perl interpreter per message; you can cap its children with the options in `/etc/sysconfig/spamassassin`. If you also run `clamav` for virus scanning, expect both to want RAM — watch `top` (Chapter 9) after enabling them.

Within SpamAssassin

SpamAssassin's own settings live in `/etc/mail/spamassassin/local.cf`. The most commonly adjusted ones:

```
# /etc/mail/spamassassin/local.cf

required_score          5.0     # score at/above which mail is marked spam (lower = more
aggressive)
rewrite_header Subject  [SPAM]  # prefix the Subject of spam so a client rule can file it
report_safe             1       # attach the original as a safe .eml, rather than altering it
inline

# Bayesian learning — improves a lot once it has seen some mail
use_bayes               1
bayes_auto_learn        1

# Always-allow and always-deny senders (note the underscores)
allowlist_from          *@trusted-partner.com
denylist_from           *@spammy-domain.example

# Trust your own network so internal/relayed mail isn't penalized
trusted_networks        127.0.0.0/8
```

After editing `local.cf`, check it parses and reload the daemon:

```
# spamassassin --lint        # report any configuration errors (silence = good)
# systemctl reload spamassassin
```

Training the Bayesian filter is what turns SpamAssassin from "decent" into "very good." Feed it examples of each kind of mail; the more it sees, the sharper it gets:

```
$ sa-learn --spam   ~/Maildir/.Junk/cur     # these are spam
$ sa-learn --ham    ~/Maildir/cur           # these are legitimate
$ sa-learn --dump magic                     # show how many messages it has learned
```

A practical workflow: tell your users to drag misfiled mail into (or out of) their Junk folder, then run `sa-learn` over those folders nightly from cron, so the filter keeps adapting to the mail your server actually receives.

> **Tip:** Test your setup end-to-end with the **GTUBE** string — a harmless, standardized test pattern that SpamAssassin always scores as spam. Send yourself a message whose body is the GTUBE line (search "SpamAssassin GTUBE" for the exact text) and confirm it gets tagged. That proves the milter, `spamd`, and your scoring are all wired together before you rely on it.

Configuring Mail Clients

With Sendmail and Dovecot running, any standard mail client can be configured with:

Setting	Value
Incoming server (IMAP)	mail.example.com, port 993, SSL/TLS
Incoming server (POP3, if preferred)	mail.example.com, port 995, SSL/TLS
Outgoing server (SMTP)	mail.example.com, port 587, STARTTLS, authentication required
Username	Full email address or account username (try the full address first)
Password	The account's normal password

This works the same in Apple Mail, Outlook, Thunderbird, and the built-in iOS and Android mail apps — the dialog boxes differ, but the values above are what each one is asking for.

> **Note:** Port 25 is for server-to-server delivery, not for mail clients to submit outgoing mail — many residential and mobile ISPs block outbound port 25 entirely to reduce spam from compromised computers. Always configure clients to use port 587 (with authentication) for sending.

Roundcube Webmail

Roundcube provides a full webmail client — usable from any browser, with no client configuration — and is what most GSP customers and their users will use day to day.

Installing Roundcube

Roundcube is packaged in EPEL (Appendix A) as `roundcubemail`:

```
# dnf install roundcubemail
```

This pulls in PHP and its dependencies if not already installed (Chapter 8 covers PHP in more detail for general web applications). Roundcube also needs a database for its own data — message state, address books, and settings (separate from the mail itself, which lives in Dovecot's Maildir storage). For a single-domain VPS, **SQLite** is the simplest choice and needs no separate database server:

```
# dnf install php-pdo               # SQLite support ships with php-pdo
```

For larger installations with many users, MariaDB or PostgreSQL (Chapter 8) scale better.

Configuring Roundcube

The package installs Roundcube's files under `/usr/share/roundcubemail`, with its configuration under `/etc/roundcubemail/`. Edit `config.inc.php` there. The settings you need to change:

```
// Database connection — SQLite example
$config['db_dsnw'] = 'sqlite:////var/lib/roundcubemail/roundcube.db?mode=0640';

// IMAP server (Dovecot)
$config['imap_host'] = 'tls://localhost:143';

// SMTP server (Sendmail's submission service)
$config['smtp_host'] = 'tls://localhost:587';
$config['smtp_user'] = '%u';   // use the logged-in user's own credentials
$config['smtp_pass'] = '%p';

// A long, random string unique to this install (24+ characters)
$config['des_key'] = 'replace-this-with-24-random-characters';
```

Initialize the database schema (for SQLite, Roundcube creates the file on first use; for MariaDB, import the bundled schema):

```
# mysql roundcube < /usr/share/roundcubemail/SQL/mysql.initial.sql   # MariaDB only
```

Publishing Roundcube on the Web

The `roundcubemail` package drops an Apache configuration file at `/etc/httpd/conf.d/roundcubemail.conf` that serves the app at `/roundcubemail` and restricts access to localhost by default. Adjust it to serve `https://example.com/webmail/` and allow your users in — for example, an alias plus a `<Directory>` grant:

```
Alias /webmail /usr/share/roundcubemail
<Directory /usr/share/roundcubemail>
    Require all granted
</Directory>
```

After reloading Apache, `https://example.com/webmail/` presents the Roundcube login screen. Log in with the same username and password used for SSH/SFTP and IMAP — Roundcube authenticates against Dovecot, which (per the configuration above) checks the system account database.

> **Note:** On Rocky Linux, for Roundcube (running under Apache) to reach Dovecot and Sendmail over the network, SELinux must allow it: `setsebool -P httpd_can_network_connect on` (and, if Roundcube writes its SQLite database under `/var/lib/roundcubemail`, confirm that directory carries the `httpd_sys_rw_content_t` label). Chapter 8 covers these booleans and contexts in full.

> **Tip:** Always access Roundcube over HTTPS — it's transmitting mail passwords. If you've followed Chapter 6's TLS setup for your main site, the same certificate covers `/webmail` automatically since it's part of the same virtual host.

Using Roundcube

The Roundcube interface should feel familiar to anyone who's used Gmail, Outlook.com, or similar webmail:

- **Compose** a new message with the pencil/"Compose" button; attachments can be added by dragging files into the compose window.
- **Folders** down the left side include Inbox, Sent, Drafts, Trash, and any custom folders or subscriptions you create — these are the same folders a desktop IMAP client sees, since both talk to the same Dovecot mailbox.
- **Address Book** stores contacts and supports multiple address books and groups; contacts can be exported to or imported from vCard format.
- **Settings → Identities** lets a user send from multiple "from" addresses or display names — useful for someone who manages mail for more than one of your hosted domains.
- **Settings → Filters** (provided by the managesieve plugin, talking to Dovecot's Pigeonhole Sieve service on port 4190) lets users create their own mail-sorting rules — "move messages from this sender to this folder," for example — without any server-side configuration from you.
- **Settings → Vacation** (also part of managesieve, on supported configurations) is the modern equivalent of the `vacation` program described earlier: users can turn an autoresponder on and off, and edit its message, entirely from their browser.

Maintaining Mail Logs

Sendmail and Dovecot both log to `/var/log/maillog` (configured via rsyslog). To watch mail activity in real time — useful when troubleshooting why a message didn't arrive:

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

Look for the message's queue ID (an alphanumeric string Sendmail assigns each message) to follow a single message's journey from acceptance through delivery or bounce. The same events are also captured by the systemd journal, so `journalctl -u sendmail` and `journalctl -u dovecot` are an alternative view. Chapter 9 covers log rotation so `/var/log/maillog` doesn't grow without bound.

Important Commands, Directories, and Files

Item	Purpose
alternatives --config mta	Choose which MTA the system mailer commands use
/etc/mail/sendmail.mc, Makefile	Sendmail's m4 source configuration
/etc/mail/local-host-names	Domains this server accepts mail for
/etc/aliases	Local-part address forwarding (rebuild with newaliases)
/etc/mail/virtusertable	Per-domain / per-address mailbox mapping
/etc/mail/access	Relay and reject rules
make -C /etc/mail	Rebuild Sendmail's tables/config

/etc/dovecot/	Dovecot configuration
~/Maildir	A user's mailbox storage
/var/log/maillog	Mail server log
/etc/roundcubemail/config.inc.php	Roundcube configuration

For More Information

- The Rocky Linux and Red Hat Enterprise Linux documentation covers configuring and securing mail servers; the Sendmail project's own site (sendmail.org) is the authoritative reference for the `.mc` macros and `sendmail.cf`.
- The Dovecot documentation (doc.dovecot.org) is the authoritative reference for Dovecot configuration, including the Pigeonhole Sieve plugin used by Roundcube's filters.
- The Roundcube documentation (roundcube.net/about/docs) covers plugins beyond the ones enabled by default, including two-factor authentication and additional address book backends.

Where to Go Next

Chapter 5 covers getting files onto your VPS — including, if you'd like, a custom logo or theme for your Roundcube installation.

Chapter 5: Transferring Files with SFTP

Every chapter so far has assumed you can get files onto and off of your VPS. This chapter covers the recommended way to do that — SFTP — along with how to set up restricted accounts for clients or collaborators who should only be able to reach a single directory.

Why SFTP Instead of FTP

Older hosting handbooks devoted a substantial chapter to FTP — its server software, client programs, anonymous access, and logon banners. A minimal Rocky Linux install doesn't include an FTP server, and GSP doesn't recommend installing one, for a simple reason: the FTP protocol sends usernames, passwords, and file contents in clear text. Anyone positioned between a client and the server — on public Wi-Fi, a compromised router, or an ISP's network — can capture that traffic.

SFTP (SSH File Transfer Protocol — not to be confused with FTPS, which is FTP wrapped in TLS) solves this by running entirely inside the same encrypted SSH connection used for shell access. Since OpenSSH is part of the base system and already running for Chapter 1's `ssh` access, SFTP requires **no additional installation, no additional open firewall port, and no separate set of credentials** — the same username and password (or SSH key) that logs you in with `ssh` works with `sftp`.

Command-Line SFTP

```
$ sftp youruser@example.com
Connected to example.com.
sftp> ls
public_html  Maildir  logs
sftp> cd public_html
sftp> put index.html
Uploading index.html to /home/youruser/public_html/index.html
sftp> get error_log
sftp> mkdir images
sftp> rm old-page.html
sftp> exit
```

Common `sftp` commands mirror their shell equivalents with a few additions:

Command	Effect
ls, cd, pwd, mkdir, rmdir, rm	Same as the shell, but operating on the **remote** side
lls, lcd, lpwd	The local-side equivalents — list/change/show directory on your computer
put file	Upload file to the current remote directory
get file	Download file to the current local directory
put -r dir / get -r dir	Upload or download an entire directory recursively

scp

For a single quick copy without an interactive session, `scp` ("secure copy") uses the same familiar syntax as the local `cp` command, just with a `host:` prefix for remote paths:

```
$ scp report.pdf youruser@example.com:public_html/files/
$ scp youruser@example.com:logs/access_log ./
$ scp -r ./site youruser@example.com:public_html/
```

rsync over SSH

For keeping a local copy of a site in sync with the server — uploading only files that have changed — `rsync` is far more efficient than re-uploading everything:

```
$ rsync -avz --delete ./site/ youruser@example.com:public_html/
```

- `-a` ("archive") preserves permissions, timestamps, and directory structure
- `-v` is verbose, showing what's being transferred
- `-z` compresses data in transit
- `--delete` removes files on the server that no longer exist locally — **omit this flag** the first time you run a sync against a directory that already has files you haven't checked, to avoid accidentally deleting them

`rsync` is included on Rocky Linux (install it with `dnf install rsync` if a minimal image left it out); your own computer needs `rsync` too, which is preinstalled on macOS and most Linux distributions.

Graphical SFTP Clients

If you prefer a file-manager-style interface:

- **FileZilla** (Windows, macOS, Linux) — free, open source, supports SFTP alongside FTP/FTPS
- **Cyberduck** (Windows, macOS) — free, with a simple drag-and-drop interface and bookmarks for saved connections
- **WinSCP** (Windows) — free, includes both a file-manager view and a built-in text editor for remote files
- **Transmit, ForkLift** (macOS, paid) — polished native Mac file transfer apps with SFTP support

All of these connect using the **SFTP** protocol (sometimes labeled "SSH File Transfer Protocol" in a dropdown, to distinguish it from "FTP" and "FTPS") on **port 22** with your normal SSH username and password or key.

> **Tip:** Webmin's File Manager module (Chapter 3) is also available for quick edits from a browser when you don't have an SFTP client handy — useful when troubleshooting from a phone or a public computer.

Restricting an Account to SFTP Only

If you're giving a client or contractor access only to upload files to their site — not a general shell account — OpenSSH can confine an account to SFTP and chroot ("change root") it to a single directory, so it can't see or affect anything else on the VPS.

Create the account as in Chapter 2, with a non-functional shell:

```
# useradd -M -s /sbin/nologin clientuser
# passwd clientuser
```

Then add a `Match` block for this user (or a group containing several such users). On Rocky Linux, `sshd_config` ends with `Include /etc/ssh/sshd_config.d/*.conf`, so the cleanest place for this is a new drop-in file — and a `Match` block must come after all global settings, which a fresh drop-in guarantees:

```
# /etc/ssh/sshd_config.d/20-clients.conf
Match User clientuser
    ChrootDirectory /var/www/sites/clientsite
    ForceCommand internal-sftp
    AllowTcpForwarding no
    X11Forwarding no
```

For `ChrootDirectory` to work, the directory itself (`/var/www/sites/clientsite` in this example) and every directory above it in the path must be **owned by root and not writable by group or other** (mode 755 or stricter). The client's actual writable content goes in a subdirectory they own, e.g. `/var/www/sites/clientsite/htdocs`, which you then point Apache's document root at for that virtual host (Chapter 6).

Restart sshd to apply:

```
# systemctl restart sshd
```

`clientuser` can now connect with any SFTP client and will see only `/` (which is actually `/var/www/sites/clientsite` from the rest of the system's perspective) and its subdirectories — they cannot `cd ..` past it, and `ssh clientuser@example.com` for a shell will be refused (`internal-sftp` only speaks the SFTP protocol, not a shell).

> **Warning:** A typo in `ChrootDirectory` ownership (for example, leaving the chroot root directory group-writable) causes sshd to refuse the connection entirely with a fairly unhelpful "broken pipe" error on the client side. If a chrooted account suddenly can't connect after you've changed permissions inside its directory tree, check `/var/log/secure` on the server (or `journalctl -u sshd`) for the specific permission sshd objected to.

If You Need Classic FTP

Some legacy devices, embedded systems, or older publishing tools only speak FTP and have no SFTP support. If you have a specific need like this, `vsftpd` (`dnf install vsftpd`) is the standard FTP server on Rocky Linux, and it can be configured for password-protected accounts, anonymous upload/download areas, and custom login banners. Because of the cleartext-credential concern above, GSP recommends:

- Limiting FTP accounts to a chrooted directory (`chroot_local_user=YES` in `/etc/vsftpd/vsftpd.conf`)

- Using FTP only over a connection you trust (e.g., from a known office network), or combining it with a VPN
- Treating any FTP password as more exposed than your SSH/SFTP credentials, and not reusing it elsewhere

> **Note:** On Rocky Linux, running `vsftpd` also means setting the SELinux booleans it needs (for example `setsebool -P ftpd_full_access on` to let it write files) and opening the firewall for FTP — `firewall-cmd --permanent --add-service=ftp`. The `ftp` service definition loads the connection-tracking helper that makes FTP's separate data connection work through the firewall. This extra moving-parts cost is one more reason to prefer SFTP.

For anything reachable from arbitrary networks — including "I just need to grab a file from my phone occasionally" — SFTP and the graphical clients above cover the same ground more safely with software that's already installed.

Where to Go Next

With files moving on and off your VPS, Chapter 6 covers the web server that will actually serve them.

Chapter 6: The Web Server

This chapter installs and configures Apache, the web server GSP recommends for Rocky Linux VPS instances, covers publishing your first site, hosting multiple domains on one VPS, and securing it with HTTPS.

Installing Apache

On Rocky Linux, the Apache HTTP Server package is named `httpd`:

```
# dnf install httpd
```

Enable and start it:

```
# systemctl enable --now httpd
```

Then open the web ports in the firewall (Rocky's default `firewalld` blocks them until you do):

```
# firewall-cmd --permanent --add-service=http --add-service=https
# firewall-cmd --reload
```

Visiting `http://example.com/` (once DNS points at your VPS — Step 1 of "Getting Started") should now show Apache's default Rocky Linux test page.

Directory Layout

Apache's package keeps its configuration under `/etc/httpd` and its content under `/var/www`:

Path	Purpose
/etc/httpd/conf/httpd.conf	Main configuration file
/etc/httpd/conf.d/	Drop-in .conf files, automatically loaded — the recommended place for your own site configuration
/etc/httpd/conf.modules.d/	LoadModule lines, split into one file per module group (see Chapter 7)
/var/www/html/	Default document root — files here are served at https://example.com/
/var/log/httpd/access_log, error_log	Access and error logs (per-site logs go here too)

> **Note:** Avoid editing `httpd.conf` itself for site-specific configuration. A future `dnf upgrade httpd` may ship a new default; rather than overwrite your edited file, RPM saves the incoming version next to it as `httpd.conf.rpmnew` (or, if a package owns the file and you changed it, your version is preserved and the new one is `.rpmnew`). Anything you place in `/etc/httpd/conf.d/*.conf` is loaded automatically and left alone by upgrades.

Publishing Web Content

For a single-site VPS, the simplest approach is to serve files directly from the default document root. Upload files there via SFTP (Chapter 5) and set ownership so Apache (which runs as the `apache` user) can read them:

```
# chown -R youruser:apache /var/www/html
# find /var/www/html -type d -exec chmod 755 {} \;
# find /var/www/html -type f -exec chmod 644 {} \;
```

This gives you (the owner) full control, lets Apache's `apache` group read everything, and keeps the files non-executable, which is correct for static HTML, CSS, JavaScript, and images. Chapter 8 covers the additional setup needed for scripts and applications.

> **Note:** On Rocky Linux there is a second gatekeeper besides ownership and permissions: **SELinux**. Files under `/var/www/html` are automatically labeled `httpd_sys_content_t`, the type Apache is allowed to read, so content placed there "just works." If you instead serve a site from a non-default location — a directory under `/home`, for instance — you must give it that label yourself, or Apache will get "Permission denied" even though `ls -l` looks correct. The Virtual Hosting section below and Chapter 8 show how.

A more common arrangement, especially once you're hosting more than one site, is to keep each site's files in your own home directory (or under `/var/www/sites/`) and point Apache at them — which is exactly what virtual hosting, below, does.

Virtual Hosting: Multiple Sites on One VPS

Name-based virtual hosting lets Apache serve different content for different domain names, even though they share the same IP address — the browser tells Apache which hostname it asked for via the HTTP `Host:` header (and, for HTTPS, via TLS's SNI extension), and Apache picks the matching configuration.

Create one file per site under `/etc/httpd/conf.d/`, for example `/etc/httpd/conf.d/example.com.conf`:

```
<VirtualHost *:80>
    ServerName example.com
    ServerAlias www.example.com
    DocumentRoot /home/youruser/public_html
    ErrorLog /var/log/httpd/example.com-error.log
    CustomLog /var/log/httpd/example.com-access.log combined

    <Directory /home/youruser/public_html>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

For an additional hosted domain, create a second file, `anotherdomain.com.conf`, with its own `ServerName`, `DocumentRoot`, and log paths. Apache reads every `.conf` file in `conf.d/` automatically — no central list to maintain.

Reload Apache after adding or changing a virtual host:

```
# apachectl configtest && systemctl reload httpd
```

`configtest` (or `apachectl -t`) checks the configuration for syntax errors before reloading — running it first means a typo causes a clear error message instead of taking down every site on the VPS.

> **Note:** Serving a site from `/home/youruser/public_html` (rather than under `/var/www`) means SELinux doesn't consider those files web content by default. Give the tree the right label once, and it will persist across reboots and relabels:

```
# semanage fcontext -a -t httpd_sys_content_t "/home/youruser/public_html(/.*)?"
# restorecon -Rv /home/youruser/public_html
```

(`semanage` comes from the `policycoreutils-python-utils` package.) Chapter 8 covers SELinux file contexts — including the read-write label for upload directories — in more depth.

> **Tip:** Webmin's **Servers → Apache Webserver** module (Chapter 3) includes a "Create Virtual Host" wizard that writes a file like the one above for you, including sensible defaults for `Options` and logging — useful once you're managing several sites and don't want to copy-paste by hand each time.

A Note on AllowOverride and .htaccess

`AllowOverride All` (used above) lets per-directory `.htaccess` files override configuration — convenient for clients who manage their own redirects or password-protected directories without your involvement, and required by many web applications' installers. The trade-off is a small performance cost (Apache checks for `.htaccess` files on every request) and that anyone who can write to the directory can change Apache's behavior for it. For a site you control entirely yourself, `AllowOverride None` with the equivalent directives placed directly in the virtual host file (as Chapter 7 discusses) is slightly faster and keeps all configuration in one place.

Securing Apache

A few changes to a file in `conf.d/` reduce the information Apache reveals about itself and your server:

```
ServerTokens Prod
ServerSignature Off
```

`ServerTokens Prod` makes Apache's `Server:` response header say only `Apache`, omitting the version number and loaded modules — information that otherwise helps an attacker know

which vulnerabilities might apply. `ServerSignature Off` removes the similar footer Apache adds to its own error pages.

By default, a directory with no `index.html`/`index.php` shows a generated file listing. For most sites this should be disabled — the `-Indexes` option in the virtual host example above does this for that directory; to apply it more broadly, set it in the `<Directory "/var/www/html">` block in `httpd.conf` or your own drop-in.

Finally, make sure Apache itself stays current:

```
# dnf upgrade httpd
```

Chapter 10 covers a routine for keeping this and everything else on your VPS patched.

HTTPS with Let's Encrypt and Certbot

Modern browsers mark plain HTTP sites as "Not Secure," and HTTPS is required for many features (geolocation, service workers, and others). Let's Encrypt provides free, automatically-renewing TLS certificates, and **Certbot** — with its Apache plugin — is the tool GSP recommends for obtaining and renewing them on Rocky Linux. First install Apache's TLS module and Certbot (the latter from EPEL — see Appendix A):

```
# dnf install mod_ssl certbot python3-certbot-apache
```

Then issue a certificate. The Apache plugin reads your virtual hosts, proves control of each domain to Let's Encrypt, installs the certificate, and adds the `<VirtualHost *:443>` configuration for you:

```
# certbot --apache -d example.com -d www.example.com
```

Certbot stores the certificate and key under `/etc/letsencrypt/live/example.com/` and installs a `systemd` timer (`certbot-renew.timer`) that renews certificates automatically — Let's Encrypt certificates are valid for 90 days, and Certbot renews them well before expiry, reloading Apache so the new certificate takes effect. Confirm the timer is active with `systemctl list-timers certbot-renew.timer`, and test renewal without actually doing it using `certbot renew --dry-run`.

If you prefer to manage the `<VirtualHost *:443>` block yourself, issue the certificate with the webroot method (which only obtains the files, changing no config) and reference them in your own virtual host:

```
# certbot certonly --webroot -w /home/youruser/public_html -d example.com -d www.example.com
```

```
<VirtualHost *:443>
    ServerName example.com
    ServerAlias www.example.com
    DocumentRoot /home/youruser/public_html

    SSLEngine on
    SSLCertificateFile      /etc/letsencrypt/live/example.com/fullchain.pem
    SSLCertificateKeyFile   /etc/letsencrypt/live/example.com/privkey.pem
</VirtualHost>
```

> **Note:** A dependency-light alternative to Certbot is `acme.sh`, a single shell script that some administrators prefer because it has no Python dependencies and supports many DNS providers for DNS-01 validation. Either tool works; this handbook standardizes on Certbot because it is the path the Rocky and RHEL documentation assume.

Chapter 7 covers redirecting HTTP to HTTPS and other TLS hardening once the certificate itself is working.

Important Commands, Directories, and Files

Item	Purpose
dnf install httpd mod_ssl	Install Apache and its TLS module
/etc/httpd/conf/httpd.conf	Main configuration
/etc/httpd/conf.d/*.conf	Your site/virtual host configuration (recommended location)
/var/www/html/	Default document root
apachectl configtest	Check configuration syntax before reloading
systemctl reload httpd	Apply configuration changes without dropping connections
systemctl restart httpd	Fully restart Apache
/var/log/httpd/	Access and error logs
certbot --apache / certbot renew	Obtain and renew a Let's Encrypt certificate

Where to Go Next

Chapter 7 builds on this foundation with rewrite rules, custom error pages, and a closer look at TLS configuration; Chapter 8 covers running PHP and database-backed applications on top of what you've set up here.

Chapter 7: Advanced Web Server Configuration

Chapter 6 got a site online with HTTPS. This chapter covers the directives, modules, and techniques that come up once a site needs more than the basics — redirects, clean URLs, password-protected areas, and a closer look at TLS.

A Working Reference of Apache Directives

Rather than reproduce Apache's entire directive reference (the official documentation at httpd.apache.org does this far better, and is kept current with each Apache release), this section covers the directives most GSP customers actually need, organized by what they're for. All of these go inside a `<VirtualHost>` block or a `<Directory>` block within one, in your `conf.d/*.conf` file from Chapter 6.

Where Files Live

Directive	Purpose
DocumentRoot /path	The directory Apache serves files from for this virtual host
DirectoryIndex index.html index.php	Which filename(s) to serve when a URL ends in /, tried in order
Alias /path /filesystem/path	Serve a URL path from a directory outside DocumentRoot (used for Roundcube in Chapter 4)
ScriptAlias /cgi-bin/ /path/to/cgi-bin/	Like Alias, but additionally marks the target as containing executable CGI scripts

Redirects and Rewrites

`mod_rewrite`, loaded by default in the `httpd` package, provides powerful URL manipulation. The two most common uses:

Redirecting an entire site to HTTPS — place this in the `<VirtualHost *:80>` block:

```
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
```

"Clean URLs" for an application that expects all requests to go through a single `index.php` (common for content management systems and frameworks):

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^ index.php [L]
```

This says: if the requested path isn't an existing file or directory, hand the request to `index.php` instead of returning "Not Found" — letting the application's own routing decide what to do with it.

For simple, permanent redirects of one URL (or one whole site) to another, `Redirect` is simpler than `mod_rewrite`:

```
Redirect 301 /old-page.html /new-page.html
Redirect 301 / https://newdomain.com/
```

Custom Error Pages

```
ErrorDocument 404 /not-found.html
ErrorDocument 500 /server-error.html
```

The referenced files are normal files within `DocumentRoot`, styled however you like — far friendlier than Apache's plain default error pages.

Working with Dynamically-Loaded Modules

Apache's functionality is split into modules, and the Rocky `httpd` package builds almost all of them as DSOs — Dynamic Shared Objects, the `mod_*.so` files under `/usr/lib64/httpd/modules/`. "Dynamically-loaded" means they are not compiled into the Apache binary; instead, each is loaded at startup only if a `LoadModule` line names it. On Rocky Linux those `LoadModule` lines are organized into one file per group under `/etc/httpd/conf.modules.d/` (rather than buried in `httpd.conf`), so turning a feature on or off is a matter of editing the right small file — or, for modules shipped as their own package (like `mod_ssl`), installing that package, which drops in its own `.conf`.

List the modules currently active:

```
$ httpd -M                    # all loaded modules (static and shared)
$ httpd -M | grep rewrite     # check whether a specific one is on
```

A `LoadModule` line names the module and the file it lives in:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

The modules worth knowing on a typical web-and-PHP VPS:

Module	Loaded by default?	What it does / why you'd want it
mod_mpm_event	Selected in 00-mpm.conf	The processing engine; use the event MPM with PHP-FPM (Chapter 8)
mod_ssl	dnf install mod_ssl	HTTPS/TLS support — <VirtualHost *:443> and the *SSL* directives (Chapter 6)
mod_rewrite	Yes	Rule-based URL rewriting and redirects (above)
mod_headers	Yes	Add/modify response headers — used for HSTS and caching (below)
mod_deflate	Yes	gzip-compress responses to cut bandwidth and speed up page loads
mod_expires	Yes	Set Cache-Control/Expires so browsers cache static assets
mod_proxy + mod_proxy_fcgi	Yes	Forward requests to another process — how PHP-FPM is served (Chapter 8)
mod_proxy_http	Yes	Reverse-proxy to an app server (Node, Python, etc.) behind Apache

mod_auth_basic + mod_authn_core	Yes	HTTP Basic password protection (below)
mod_alias	Yes	Redirect, Alias, and ScriptAlias directives
mod_status	Yes (restrict it!)	A /server-status page of live worker/traffic stats
mod_info	Yes (restrict it!)	A /server-info dump of the running configuration
mod_security	dnf install mod_security	A web-application firewall, for sites that need request filtering (EPEL)

To **enable** a module that isn't loaded, add (or uncomment) its `LoadModule` line in the appropriate `/etc/httpd/conf.modules.d/` file — or install the package that provides it — and reload:

```
# nano /etc/httpd/conf.modules.d/00-optional.conf   # add/uncomment the LoadModule line
# apachectl configtest && systemctl reload httpd
```

`apachectl configtest` (run it every time before reloading) catches the most common module mistake: enabling a module whose dependency is still unloaded — Apache will refuse to start and the error message names the missing piece. To **disable** a module, comment its line back out and reload. Webmin's **Servers → Apache Webserver** module (Chapter 3) presents every available module as a checkbox, which is often easier than hunting through the config files for the right line.

> **Warning:** `mod_status` and `mod_info` are extremely useful for debugging but expose internal details of your server. If you enable either, wrap its `<Location>` block in a `Require ip ...` (or the Basic-auth protection below) so only you can reach it — never leave `/server-status` open to the public Internet.

MIME Types

Apache uses the system `/etc/mime.types` file (referenced from `httpd.conf` via `TypesConfig`) to decide what `Content-Type` header to send for a file, based on its extension — which in turn affects whether a browser displays a file inline, downloads it, or runs it as a script. To add a type Apache doesn't already know about:

```
AddType application/manifest+json .webmanifest
```

placed in your virtual host or a global `conf.d/*.conf` file. Most modern file types (`.json`, `.svg`, `.woff2`, etc.) are already present in current releases; this is mainly useful for newer or unusual formats.

Password-Protected Directories

For a staging area, admin tool, or any directory that should require a username and password separate from your VPS accounts, Apache's `mod_auth_basic` (loaded by default) provides HTTP Basic Authentication. The `htpasswd` tool comes from the `httpd-tools` package.

Create a password file (outside the document root, so it can't be downloaded directly):

```
# htpasswd -c /etc/httpd/passwords/staging.htpasswd alice
New password:
Re-type new password:
```

Omit `-c` when adding additional users to an existing file. Then protect a directory:

```
<Directory /home/youruser/public_html/staging>
    AuthType Basic
    AuthName "Staging Area"
    AuthUserFile /etc/httpd/passwords/staging.htpasswd
    Require valid-user
</Directory>
```

Visiting that directory now prompts for one of the usernames in `staging.htpasswd` and its password. This is convenient but limited — credentials are sent with every request (protected by HTTPS, but not hashed beyond what `htpasswd` stores), and there's no "forgot password" flow. For anything beyond a handful of trusted collaborators, an application with its own login system (Chapter 8) is more appropriate.

Server-Side Includes (SSI)

SSI lets an HTML file include small dynamic snippets — most commonly the current date or the contents of another file — without a full programming language. Enable it for a directory:

```
<Directory /home/youruser/public_html>
    Options +Includes
</Directory>
```

and use files with a `.shtml` extension (or configure `.html` files to be processed, at a small performance cost for every request):

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

A simple SSI directive, placed inside an HTML comment so it degrades gracefully if SSI isn't enabled:

```
<!--#include virtual="/includes/footer.html" -->
<!--#echo var="DATE_LOCAL" -->
```

SSI was far more common before server-side templating and client-side JavaScript became universal, but it remains a lightweight option for a mostly-static site that needs a shared header or footer across many pages without a full content management system.

HTTPS and TLS, in More Depth

Chapter 6 covered obtaining a certificate; this section covers configuring Apache to use it well.

Redirecting HTTP to HTTPS

The `mod_rewrite` snippet earlier in this chapter handles this, but an even simpler approach for a virtual host that should only ever be reached over HTTPS is:

```
<VirtualHost *:80>
    ServerName example.com
    Redirect permanent / https://example.com/
</VirtualHost>
```

HSTS (HTTP Strict Transport Security)

Once you're confident HTTPS is working reliably, **HSTS** tells browsers to never even attempt a plain-HTTP connection to your domain again — closing the brief window during a redirect where a request could theoretically be intercepted:

```
<VirtualHost *:443>
    # ... existing SSL configuration ...
    Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
</VirtualHost>
```

(`mod_headers`, used here, is loaded by default.) The `max-age` is in seconds — 31,536,000 is one year. Because browsers will refuse plain HTTP for that long once they've seen this header, only add `includeSubDomains` if every subdomain of `example.com` either has HTTPS working or doesn't exist yet.

Cipher and Protocol Configuration

The `mod_ssl` package's default configuration in `/etc/httpd/conf.d/ssl.conf` is reasonable for most sites, but two settings are worth checking:

```
SSLProtocol -all +TLSv1.2 +TLSv1.3
SSLCipherSuite PROFILE=SYSTEM
SSLHonorCipherOrder on
```

This allows only TLS 1.2 and 1.3 (modern browsers no longer offer the obsolete SSLv3/TLS 1.0/1.1 anyway, but disabling them server-side closes the door on downgrade attempts).

> **Note:** Rocky Linux has a system-wide **crypto policy** that sets sane TLS defaults for every application at once — which is why `SSLCipherSuite PROFILE=SYSTEM` above simply defers to it. View the current policy with `update-crypto-policies --show` (usually `DEFAULT`); switching to `FUTURE` tightens it across the whole system. This is the idiomatic Rocky lever for cipher strength, in preference to hand-writing a cipher list per service.

> **Tip:** Once your site is live, the Qualys SSL Labs test (ssllabs.com/ssltest) gives a free, detailed report on your TLS configuration, including specific recommendations if anything here needs adjustment. Re-run it after any change to `ssl.conf` or the crypto policy.

For More Information

- The Apache HTTP Server documentation (httpd.apache.org/docs/2.4/) is the complete and current reference for every directive and module mentioned in this chapter and the last.
- httpd.apache.org/docs/2.4/mod/mod_rewrite.html covers `mod_rewrite` syntax in full, including the rewrite flags (`[R=301,L]`, etc.) used above.
- The Mozilla SSL Configuration Generator (ssl-config.mozilla.org) produces a complete, current `ssl.conf` snippet for a chosen compatibility level (modern, intermediate, or old).

Where to Go Next

Chapter 8 covers what goes behind `index.php` — running PHP applications, connecting them to a database, the SELinux contexts they need, and the security practices that come with running someone else's (or your own) code on your VPS.

Chapter 8: Web Applications and Security

Static HTML covers a lot of ground, but most sites eventually need something dynamic — a contact form, a blog, a shopping cart, or a full content management system. This chapter covers running PHP applications (the most common case by far) behind Apache, adding a database, the SELinux contexts those applications need on Rocky Linux, and the security habits that matter once your VPS is running someone else's code as well as your own.

Installing PHP

Rocky Linux 10 provides PHP in its AppStream repository. Install PHP itself, the FastCGI process manager, and the extensions most applications need:

```
# dnf install php php-fpm php-mysqlnd php-pdo php-gd php-mbstring php-xml php-curl php-zip
php-opcache php-cli
```

> **Note:** PHP no longer runs as an Apache module (`mod_php`) on Rocky Linux — it runs as its own service, **`php-fpm`** ("FastCGI Process Manager"), and Apache forwards requests for `.php` files to it. This is also how PHP is typically deployed with nginx and other web servers, so the configuration below is portable if you ever switch. If your application needs a different PHP version than the AppStream default, the third-party Remi repository packages several in parallel.

Enable and start php-fpm:

```
# systemctl enable --now php-fpm
```

On Rocky Linux, `php-fpm` listens on a Unix socket at `/run/php-fpm/www.sock` and runs as the `apache` user out of the box, so it already matches the web server — there is usually nothing to change. The pool configuration lives in `/etc/php-fpm.d/www.conf`, and PHP's own settings in `/etc/php.ini`.

Connecting Apache to PHP-FPM

The `php` package drops a ready-made `/etc/httpd/conf.d/php.conf` that already routes `.php` requests to the php-fpm socket — so on a stock Rocky install, PHP works as soon as both services are running. For reference (or to replicate in a virtual host), the essential wiring is:

```
<FilesMatch "\.php$">
    SetHandler "proxy:unix:/run/php-fpm/www.sock|fcgi://localhost/"
</FilesMatch>

DirectoryIndex index.php index.html
```

Reload Apache after any change, then drop a test file in your document root:

```
<?php phpinfo();
```

Visiting that file in a browser should show PHP's configuration page — a long table of every setting and loaded extension. **Delete this file once you've confirmed it works**; `phpinfo()`

reveals detailed information about your server that's useful to an attacker and has no reason to be reachable publicly.

Adding a Database

Most PHP applications (content management systems, forums, e-commerce platforms) need a database. Rocky Linux packages three options:

- **SQLite** — a serverless, single-file database, included via the `php-pdo` extension above with no separate server to install. Good for smaller sites and for Roundcube (Chapter 4); not ideal for applications with many simultaneous writers.
- **MariaDB** — a widely-used, MySQL-compatible database server, the choice most PHP applications (WordPress, for example) document first: `dnf install mariadb-server mariadb`.
- **PostgreSQL** — a feature-rich database server favored by many newer applications and frameworks: `dnf install postgresql-server`.

For MariaDB:

```
# dnf install mariadb-server mariadb
# systemctl enable --now mariadb
# mariadb-secure-installation
```

`mariadb-secure-installation` walks through setting a root password for the database server (a separate concept from your VPS's root account), removing anonymous accounts, and disabling remote root login — all sensible defaults for a single-server setup. (PostgreSQL's equivalent first step is `postgresql-setup --initdb`, then `systemctl enable --now postgresql`.)

Create a database and a dedicated user for your application (never use the database's root account in an application's configuration file):

```
CREATE DATABASE myapp;
CREATE USER 'myapp'@'localhost' IDENTIFIED BY 'choose-a-strong-password';
GRANT ALL PRIVILEGES ON myapp.* TO 'myapp'@'localhost';
FLUSH PRIVILEGES;
```

Deploying an Application

Most applications follow the same general pattern:

- Upload (or `git clone`) the application's files into a directory under your virtual host's document root, or make that directory the document root itself.
- Create a configuration file — usually by copying a `config-sample.php`, `.env.example`, or similar — with your database credentials and any site-specific settings.
- Set file ownership, permissions, and SELinux contexts (below).
- Run any installer the application provides (often by visiting a URL in a browser, or running a command-line setup script).

File Ownership, the "Execute Bit," and SELinux Contexts

Chapter 1 introduced file permissions; for applications, three points matter beyond what was covered there.

PHP files do not need to be executable. Apache hands `.php` files to php-fpm, which reads and interprets them — it doesn't run them as a program the way the shell runs `backup.sh`. Standard `644` permissions (owner read/write, everyone else read-only) are correct for application code.

Uploaded files need write access, but never execute access. If your application lets users upload files (images, documents, avatars), the directory they're saved to needs to be writable by the `apache` user — but should never allow PHP files placed there to run. A directory-level rule prevents this even if an attacker manages to upload a `.php` file disguised as an image:

```
<Directory /home/youruser/public_html/uploads>
    <FilesMatch "\.php$">
        Require all denied
    </FilesMatch>
</Directory>
```

SELinux decides what Apache may read and write, by file label. This is the Rocky-specific layer with no equivalent on some other systems, and the most common reason a correctly-permissioned application still fails. Three labels matter:

SELinux type	Use it for
httpd_sys_content_t	Application code and static files Apache only needs to *read* (the default under /var/www)
httpd_sys_rw_content_t	Directories Apache must *write* — upload folders, caches, generated config
httpd_sys_script_exec_t	CGI scripts Apache executes directly (rare with PHP-FPM)

Label an uploads (or cache) directory read-write, then apply it:

```
# semanage fcontext -a -t httpd_sys_rw_content_t "/home/youruser/public_html/uploads(/.*)?"
# restorecon -Rv /home/youruser/public_html/uploads
```

And turn on the SELinux booleans an application needs — for example, to let Apache/PHP connect out to a database on another host, reach a remote API, or send mail:

```
# setsebool -P httpd_can_network_connect_db on    # connect to a remote database
# setsebool -P httpd_can_network_connect on        # connect to remote APIs/services
# setsebool -P httpd_can_sendmail on               # let the app send mail
```

Finally, configuration files containing passwords (database credentials, API keys) should be readable only by the owner and the `apache` group, never world-readable:

```
$ chmod 640 config.php
$ chown youruser:apache config.php
```

A Practical Security Checklist

Full web application security is a large topic — entire books cover it, and the application framework you choose has likely already addressed much of it for you. The items below are the ones a VPS administrator (rather than the application's developers) is responsible for, and where neglecting them tends to cause the most damage.

Keep everything updated. The single most common way sites get compromised isn't a sophisticated attack — it's a known vulnerability in an outdated version of WordPress, a plugin, or PHP itself, for which a fix has existed for months. Set a recurring reminder (or use Webmin's Software Package Updates module, Chapter 3) to check for updates to PHP, your database, and every application and plugin you've installed.

Don't trust input — but that's mostly the application's job. Cross-site scripting (XSS), SQL injection, and similar attacks happen when an application treats data a user provided as code or as a database query fragment instead of as data. Modern frameworks and well-maintained applications handle this correctly by default (parameterized queries, output escaping); the practical implication for you is to avoid hand-rolled scripts that build SQL by concatenating strings, and to be cautious about plugins or themes from sources without a track record of prompt security fixes.

Limit what a compromised application can do. If one application on your VPS is compromised, the damage should ideally be contained to that application. Running each site's files under a distinct user/group (Chapter 2), using separate database users with access only to their own database (above), and — on Rocky — keeping **SELinux in enforcing mode** all work together here: SELinux confines the `apache/php-fpm` processes so that even a compromised application can only touch files carrying the web labels, not the rest of the system.

Use strong, unique credentials everywhere. This applies to your VPS login, database passwords, and every application's admin account. A password manager makes "unique" practical; for application admin accounts, also consider changing default usernames like `admin` where the application allows it, since automated attacks often try common usernames first.

Back up before you update, and test the backup. Chapter 9 covers backups in detail. An update that breaks a site is an inconvenience if you can restore yesterday's working copy in five minutes, and a crisis if you can't.

Watch for SSRF and "fetch a URL" features. Applications that fetch a remote URL on the server's behalf (image proxies, webhook testers, "import from URL" features) can sometimes be tricked into making requests to internal services — including, on some cloud platforms, metadata endpoints that return credentials. If an application has such a feature and you don't need it, disabling it removes an entire category of risk; leaving the `httpd_can_network_connect` boolean off (above) unless an app genuinely needs it is a complementary defense.

Troubleshooting Application Errors

When something goes wrong, the answer is almost always in a log file:

Symptom	Where to look
Blank white page ("White Screen of Death")	/var/log/httpd/error_log and the php-fpm log (journalctl -u php-fpm, or /var/log/php-fpm/www-error.log)
"500 Internal Server Error"	Same as above — this is Apache's generic response when PHP fatally errors
Database connection errors	The application's own log, plus confirm the service is running: systemctl status mariadb
"403 Forbidden" or unexplained denial	/var/log/httpd/error_log — and, if that looks fine, an SELinux denial: ausearch -m avc -ts recent

For development or staging sites (never on a live, public site, since it can leak sensitive details to visitors), temporarily setting `display_errors = On` in `/etc/php.ini` shows PHP errors directly in the browser instead of only in the log — remember to set it back to `Off` afterward.

```
$ tail -f /var/log/httpd/error_log
```

run while reproducing the problem in a browser, often pinpoints the issue immediately — the error message usually includes the exact file and line number where something went wrong.

> **Tip:** On Rocky Linux, when ordinary permissions look correct but Apache still can't read or write a file, suspect SELinux. `ausearch -m avc -ts recent` shows recent denials, and — if you install `setroubleshoot-server` — `sealert -a /var/log/audit/audit.log` translates each one into plain English with the exact command to fix it. This is the single most useful habit for diagnosing "but the permissions are right!" mysteries on Rocky.

Where to Go Next

Chapter 9 covers the ongoing maintenance that keeps everything in this and the previous chapters running smoothly: log rotation, backups, and keeping an eye on your VPS's resource usage.

Chapter 9: Maintaining Your VPS

A VPS that's been set up correctly mostly takes care of itself — but "mostly" is doing some work in that sentence. This chapter covers the routine tasks that keep it that way: managing logs, scheduling automated jobs, watching resource usage, and backing up (and restoring) your data.

Server Administration: Services and Common Ports

A large part of administering a server is knowing which programs are listening for connections, on which ports, and whether they should be. A "port" is simply a numbered doorway on your VPS: when a browser connects to your web site it knocks on port 80 or 443; when mail arrives it knocks on port 25; when you log in over SSH you knock on port 22. Each listening service claims one or more ports, and your firewall (Chapter 10) decides which doorways the outside world is allowed to knock on at all.

Seeing what is listening. The modern tool is `ss`; the older `netstat` shows the same information:

```
# ss -tulpn               # listening TCP/UDP sockets, with the program and PID
# ss -ltnp | grep :25     # who is listening on port 25?
# netstat -tlnp           # the classic equivalent (from the net-tools package)
```

The output names the program, the PID, and the local address/port. If a port you don't recognize is open, find out which package owns the program with `rpm -qf $(which programname)` before deciding whether to disable it.

The well-known ports. Ports below 1024 are the "well-known" (privileged) ports — only root can open them — and the ones below are the services you are most likely to run, troubleshoot, or open in the firewall on a typical web-and-mail VPS:

Port	Protocol	Service	Used for / notes
20 / 21	TCP	FTP (data / control)	Legacy file transfer — **avoid**; use SFTP (Chapter 5)
22	TCP	SSH / SFTP / SCP	Remote login and file transfer — keep open
25	TCP	SMTP	Server-to-server mail delivery (Sendmail)
53	TCP/UDP	DNS	Name resolution; only open if you run a name server
80	TCP	HTTP	Plain web traffic (usually redirected to 443)
110	TCP	POP3	Legacy mail retrieval (plaintext) — prefer 995
123	UDP	NTP	Time synchronization (chronyd, the Rocky default)
143	TCP	IMAP	Mail retrieval (plaintext/STARTTLS) — prefer 993
443	TCP	HTTPS	Encrypted web traffic (Apache + mod_ssl)
465	TCP	SMTPS	Submission over implicit TLS (alternative to 587)
587	TCP	SMTP submission	Authenticated mail sending by your users (Chapter 4)
993	TCP	IMAPS	IMAP over TLS — what Roundcube/clients use (Dovecot)
995	TCP	POP3S	POP3 over TLS (Dovecot)
3306	TCP	MariaDB/MySQL	Database — bind to localhost only, never expose

5432	TCP	PostgreSQL	Database — bind to localhost only, never expose
6379	TCP	Redis/Valkey	Cache/queue — bind to localhost only
8080 / 8443	TCP	Alt-HTTP / Alt-HTTPS	App servers and proxies behind Apache
10000	TCP	Webmin	Control panel (Chapter 3) — restrict by IP or tunnel

> **Tip:** The guiding rule of server administration is **least exposure**: a service that only your own machine needs (a database, a cache, a local mail-filter milter) should listen on `127.0.0.1` (localhost) only, not on your public IP. Then even a firewall mistake can't expose it. Confirm with `ss -ltnp` that anything sensitive shows `127.0.0.1:PORT` and not `0.0.0.0:PORT`.

Starting, stopping, and enabling services. Rocky Linux manages long-running services (daemons) with `systemd`, through the `systemctl` command:

```
# systemctl status httpd         # is it running? (also shows recent log lines)
# systemctl start httpd          # start it now
# systemctl restart httpd        # stop then start
# systemctl reload httpd         # re-read config without dropping connections
# systemctl enable httpd         # start it automatically at every boot
# systemctl enable --now httpd   # enable and start in one step
# systemctl list-unit-files --state=enabled   # everything set to start at boot
```

The distinction trips people up: `systemctl start` runs a service right now but does not survive a reboot; `systemctl enable` is what makes it come back after the VPS restarts. The combined `enable --now` (used throughout this handbook) does both.

Logs

Nearly every service this handbook has covered writes a log file under `/var/log/` — and, in parallel, to the systemd journal:

Log file	Written by
/var/log/maillog	Sendmail and Dovecot (Chapter 4)
/var/log/httpd/access_log, error_log (or per-site equivalents)	Apache (Chapters 6-8)
/var/log/secure	SSH and sudo/su authentication attempts
/var/log/messages	General system messages
/var/log/audit/audit.log	SELinux denials and other audit events

In addition to these text files, `systemd` keeps a structured journal that you query with `journalctl` — often the fastest way to see exactly what a service logged:

```
# journalctl -u httpd              # everything Apache logged
# journalctl -u sshd --since "1 hour ago"
# journalctl -xe                   # the most recent entries, with explanations
# journalctl -f                    # follow new entries live (like tail -f)
```

logrotate: Automatic Log Rotation

Left alone, these files would grow forever. Rocky Linux runs `logrotate` on a schedule (a `systemd` timer, `logrotate.timer`) which, based on rules in `/etc/logrotate.conf` and per-service files under `/etc/logrotate.d/`, renames the current log, compresses older rotations, deletes ones past a configured limit, and signals the relevant service to start writing to a fresh file.

A `logrotate.d` entry looks like:

```
# /etc/logrotate.d/example.com
/var/log/httpd/example.com-*.log {
    daily
    rotate 14
    missingok
    compress
    delaycompress
    sharedscripts
    postrotate
        /bin/systemctl reload httpd > /dev/null 2>&1 || true
    endscript
}
```

This rotates the files daily, keeps 14 compressed rotations, and reloads Apache after rotation so it reopens its log files. The package for any service you install (Apache, Sendmail, Dovecot) typically drops its own appropriate file into `/etc/logrotate.d/`; `ls /etc/logrotate.d/` shows what's already configured, and you add entries (like the one above) for any custom per-site logs from Chapter 6 that aren't covered yet.

Scheduling Tasks: cron and systemd Timers

`cron` (provided by the `cronie` package) runs commands on a schedule. System-wide jobs live in `/etc/crontab` and drop-in files under `/etc/cron.d/`; an individual user's own jobs are managed with `crontab -e`, which opens their personal crontab in your default editor.

A crontab line has five time fields (minute, hour, day-of-month, month, day-of-week) followed by the command:

```
# Run a backup script every night at 2:15 AM
15 2 * * *   /home/youruser/bin/backup.sh

# Check a site is responding every 5 minutes
*/5 * * * *  /home/youruser/bin/check-site.sh
```

`*` means "every value"; `*/5` means "every 5 units." Output from a cron job (anything it prints) is mailed to the account's local mailbox by default — one more reason to make sure mail delivery (Chapter 4) is working, or to redirect a noisy job's output to a log file with `>> /path/to/log 2>&1`.

For routine system housekeeping, Rocky Linux uses the directories `/etc/cron.daily/`, `/etc/cron.weekly/`, and `/etc/cron.monthly/` — drop an executable script into one and it runs on that cadence (driven by `cronie/anacron`). The modern alternative, increasingly

preferred for new work, is a **systemd timer**: a `.timer` unit paired with a `.service` unit, which you list with `systemctl list-timers`. Either is fine; cron is simplest for a one-line job, timers give you logging in the journal and more flexible schedules.

> **Tip:** Make sure root's scheduled-job output actually reaches you: add `root: youruser` to `/etc/aliases` and run `newaliases` (Chapter 4), so the nightly housekeeping summaries land in a mailbox you read rather than piling up unseen in root's local spool.

Monitoring Load and Processes

A handful of commands answer "what is my VPS doing right now," and three of them — `top`, `ps`, and `du` — are worth knowing well, because between them they account for most day-to-day diagnosis. Start with the quick overview:

```
$ uptime
 14:32:10 up 12 days,  3:14,  1 user,  load average: 0.18, 0.22, 0.19
```

The three load-average numbers are the average number of processes wanting CPU time over the last 1, 5, and 15 minutes. On a single-core VPS, sustained numbers above 1.0 mean something is keeping the CPU fully busy; on a multi-core VPS, compare against the number of cores (a load of 2.0 on a 4-core VPS is only half-busy). Comparing the three numbers tells you the trend: a 1-minute figure well above the 15-minute figure means load is climbing.

Reading top

`top` shows a live, continuously-updating view of the system, sorted by CPU usage by default. Press `q` to quit. The screen begins with a five-line summary followed by the process-table header:

```
$ top

top - 13:34:35 up 21 days, 14:08,  1 user,  load average: 0.36, 0.29, 0.31
Tasks:  98 total,   1 running,  97 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  15909.0 total,   6524.0 free,   2073.0 used,   7312.0 buff/cache
MiB Swap:   1024.0 total,    999.0 free,     25.0 used.  13420.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
```

Every term in that header means something specific:

Line 1 — overall activity. The current time, the uptime (`up 21 days, 14:08`), the number of logged-in users, and the same 1/5/15-minute load averages as `uptime`.

Line 2 — task counts. The total number of processes, broken down by **state**: `running` (on the CPU right now), `sleeping` (waiting for something — input, a timer, the disk — which is where healthy processes spend most of their time), plus occasionally `stopped` or `zombie` (finished but not yet reaped by its parent).

Line 3 — CPU time, as percentages that add up to 100%:

Field	Meaning
us (user)	Running ordinary user programs (your web server, PHP, scripts)
sy (system)	Running kernel code on programs' behalf (disk, network, system calls)
ni (nice)	Running user programs whose priority was lowered with nice
id (idle)	Doing nothing — high idle is good; it means spare capacity
wa (iowait)	Waiting on disk or network I/O — high values mean storage is the bottleneck
hi / si	Servicing hardware / software interrupts
st (steal)	Time the hypervisor gave to other guests — on a VPS, sustained steal means the host is overcommitted

Line 4 — physical memory (RAM), and **Line 5 — swap.** Linux reports `total`, `free`, `used`, and `buff/cache` memory, plus an `avail Mem` figure:

Field	Meaning
total	All physical RAM the VPS has
used	In use by running programs (excluding reclaimable cache)
free	Completely unused — often small, and that is fine (see below)
buff/cache	File-system buffers and cache — reclaimable the instant a program needs the memory
avail Mem	An estimate of how much is *available* to start new programs without swapping — the number to actually watch

A common worry is "almost nothing is free!" — but on a healthy Linux system that is normal and good: the kernel keeps recently-used file data in `buff/cache` rather than wasting RAM by leaving it `free`, and hands it back instantly when a program needs it. Watch `avail Mem`, not `free`. The real sign of memory pressure is **swap**: a little swap in use is fine, but steadily growing swap, or constant **swap activity**, is when to consider a larger VPS plan.

The process table columns. Below the header, each row is one process:

Column	Meaning
PID	Process ID — the number you pass to kill
USER	The account the process runs as (apache for the web server, mysql, etc.)
PR / NI	Kernel scheduling priority and the "niceness" you can set: -20 (greedy) to +20 (yields)
VIRT	Total virtual memory the process has mapped
RES	**Resident** memory — the part actually in RAM right now (usually the number you care about)
SHR	Memory shared with other processes (libraries, etc.)
S	State: R (running), S (sleeping), D (uninterruptible I/O wait), Z (zombie)
%CPU / %MEM	Share of one CPU, and share of physical RAM, this process is using
TIME+	Total CPU time the process has accumulated
COMMAND	The program's name

A few interactive keystrokes make `top` far more useful while it's running: press `M` to sort by memory (find the memory hog) or `P` to sort by CPU (the default), `1` to toggle a per-core CPU view, `u` then a username to show only that account's processes, `?` for the full key list, and `q` to quit.

ps: a snapshot of processes

Where `top` is a live dashboard, `ps` prints a one-time snapshot you can search and pipe into other commands:

```
$ ps aux                       # every process, with %CPU, %MEM, and the full command
$ ps aux | grep php            # just the ones matching "php" — find a runaway script
$ ps aux --sort=-%mem | head   # the ten biggest memory users
$ ps -U apache                 # every process running as the Apache user
$ ps -p 79562 -o pid,rss,comm  # specific PID, choosing the columns to show
```

In `ps aux`, the flags mean: `a` = processes of all users, `u` = the detailed user-oriented format, `x` = include daemons that have no controlling terminal. The `RSS` column is resident memory in kilobytes — the same idea as `top`'s `RES`.

du and df: where the disk went

`df` answers "how full is each filesystem"; `du` answers "what is filling it up." A full or nearly-full disk is one of the most common and most easily-overlooked causes of mysterious failures — it can produce everything from "can't send mail" (the queue can't write) to a database refusing to start.

```
$ df -h                          # free space per filesystem, human-readable
$ du -sh /home/youruser          # total size of one directory
$ du -sh /var/log/*              # size of each item under /var/log
$ du -h -d 1 /var | sort -h      # one level deep, sorted smallest-to-largest
$ du -sh * | sort -h | tail      # the biggest things in the current directory
```

The key `du` options: `-s` summarizes (one total per argument), `-h` prints human-readable units, and `-d N` limits how many directory levels deep it descends. The pattern `du -h -d 1 <dir> | sort -h` is the workhorse for hunting down a space hog: it shows each immediate subdirectory's size, sorted so the offender is at the bottom of the list. Drill in one level at a time until you find it.

```
$ vmstat 1
```

prints a line of memory, swap, and CPU statistics every second (until you press Ctrl+C) — useful for seeing whether a slowdown correlates with memory pressure (non-zero figures in the `si`/`so` swap-in/swap-out columns) versus pure CPU load.

Killing a Runaway Process

If `top` or `ps` shows a process that's stuck or consuming far more resources than it should:

```
$ kill 12345          # ask process 12345 to terminate gracefully
$ kill -9 12345       # force-terminate if it doesn't respond
```

You can only signal processes you own; use `sudo kill` for a process owned by another account (such as `apache`).

Backups

A backup strategy has two halves: making the backup, and — equally important, and far more often neglected — **testing that you can restore from it**.

File-Based Backups

For web files, configuration, and anything else that's just files, `tar` combined with `rsync` (Chapter 5) covers most needs:

```
# Create a compressed snapshot of a site's files
$ tar czf /home/youruser/backups/site-$(date +%Y%m%d).tar.gz \
    -C /home/youruser public_html

# Copy it somewhere off the VPS
$ rsync -avz /home/youruser/backups/ you@otherhost:backups/example.com/
```

`$(date +%Y%m%d)` embeds the current date in the filename, so each day's backup gets its own file and you can keep a rolling window of recent ones.

Database Backups

A file copy of a running database's files isn't reliable — use the database's own export tool, which produces a consistent snapshot:

```
# MariaDB
$ mysqldump -u myapp -p myapp > /home/youruser/backups/myapp-$(date +%Y%m%d).sql

# PostgreSQL
$ pg_dump myapp > /home/youruser/backups/myapp-$(date +%Y%m%d).sql
```

Combine this with the file-based approach above (and the same off-VPS copy step) for a complete backup of an application.

LVM Snapshots

Rocky Linux's default disk layout uses LVM (the Logical Volume Manager), which can take an almost-instant point-in-time snapshot of a volume — useful as a safety net immediately before a risky change (a major upgrade, a database migration). First find your volume group and logical volume names with `lvs`, then snapshot:

```
# lvs                                           # list logical volumes (note the VG name)
# lvcreate -L 5G -s -n root_snap /dev/rl/root   # snapshot the root LV ("rl" = your VG)
```

The `-L 5G` sizes the space set aside to record changes since the snapshot was taken — it fills up as the live data diverges, so size it for how much will change before you remove it. To roll back to the snapshot (discarding everything since), merge it and reboot; to discard the snapshot instead, remove it:

```
# lvconvert --merge /dev/rl/root_snap   # roll back on next reboot
# lvremove /dev/rl/root_snap            # ...or just delete the snapshot
```

> **Note:** Because a rollback discards changes, and because a snapshot lives on the same physical storage as the volume it copies, LVM snapshots are **not** a substitute for off-VPS backups — they protect against "I broke something five minutes ago," not against storage failure or a fat-fingered `lvremove`. GSP's automated cross-region snapshots and your own off-VPS copies cover that second case.

Restoring Files

Restoring from a tar archive:

```
$ tar xzf site-20260601.tar.gz -C /home/youruser/restore-test/
```

Extracting to a separate directory first (rather than directly over the live site) lets you compare before overwriting anything — and is the "test your backup" step mentioned at the start of this section. Periodically — quarterly is a reasonable cadence for a small site — actually do a test restore of your most recent backup. A backup you've never restored from is a backup you don't actually know works.

Watching Web Statistics

Step 7 of "Getting Started" mentioned installing a log analyzer once your site is live. All three below come from EPEL (Appendix A). The modern choice is GoAccess:

```
# dnf install goaccess
$ goaccess /var/log/httpd/example.com-access.log \
    -o /home/youruser/public_html/stats/index.html --log-format=COMBINED
```

GoAccess can also run as a live, terminal-based dashboard (omit `-o` and the file argument), or be scheduled via cron to regenerate the HTML report nightly. AWStats is the older monthly-summary option:

```
# dnf install awstats
```

AWStats is configured per-site via a file under `/etc/awstats/` and is typically run from cron to build a monthly report, then viewed as static HTML or through its CGI interface.

A third, classic option is **Webalizer**, one of the oldest and lightest web-log analyzers — it processes very large logs quickly and with minimal resources, which makes it a good fit for a small VPS:

```
# dnf install webalizer
```

Webalizer is configured through a single file, `/etc/webalizer.conf`, where you set the log to read, the report's output directory, and the hostname; running `webalizer` with no arguments then reads that config and writes a self-contained set of HTML pages and bar

charts. It is less interactive than GoAccess and less detailed than AWStats, but it is extremely lightweight and has been a dependable staple of Unix web hosting for decades.

Whichever you choose, put the generated report behind the password protection from Chapter 7 (`AuthType Basic`) — visitor statistics, including IP addresses and the search terms people used to find your site, aren't something to publish openly.

Troubleshooting Common Errors

Most problems on a VPS produce a recognizable error, and most are diagnosed the same way: read the log, reproduce the error, change one thing, test again. This section collects the errors GSP customers hit most often and the first thing to try for each. The universal first step, whatever the symptom, is to look at the relevant log (above) or the journal — almost every error below writes an explanatory line somewhere.

"A service won't start." Read the exact complaint before anything else:

```
# systemctl status httpd          # shows the failure and the last log lines
# journalctl -xeu httpd            # the full, explained log for that unit
# apachectl configtest            # Apache: pinpoints the bad directive
# sendmail -bv root               # Sendmail: trace how an address resolves
```

A service that ran yesterday and won't start today is most often a typo introduced in its config (run the config-test command for that service) or a full disk (`df -h`).

"Permission denied." The classic web error. Apache runs as the `apache` user, so every directory in the path and the file itself must be readable by `apache`, and directories must be executable (searchable) — **and**, on Rocky, the files must carry a web SELinux label. Check both, from the top down, and fix ownership/mode/label rather than resorting to `chmod 777` (which is a security problem, not a fix):

```
$ namei -l /home/youruser/public_html/index.html   # perms at every level
$ ls -Z /home/youruser/public_html/index.html      # the SELinux context
# restorecon -Rv /home/youruser/public_html         # reset to the labels SELinux expects
```

"403 Forbidden" in a browser usually means either the directory has no `index.html`/`index.php` and directory listing is off (intended — add an index file), or a `Require`/`AllowOverride` rule is blocking access — check the virtual host's `<Directory>` block and the error log.

"500 Internal Server Error" is almost always the application's fault, not Apache's — a PHP fatal error or a bad `.htaccess` line. The real message is in the error log: `tail -f /var/log/httpd/error_log`, then reload the page.

"Address already in use" / "Could not bind." Another program already holds the port. Find it and decide which one should win:

```
# ss -ltnp | grep :80        # what is already listening on port 80?
```

"Connection refused" vs. "Connection timed out." These point in opposite directions. *Refused* means you reached the VPS but nothing is listening on that port (the service is down, or bound to localhost only) — check `systemctl status` and `ss -ltnp`. *Timed out* means traffic isn't getting through at all — almost always the **firewall** (Chapter 10): confirm the port is allowed with `firewall-cmd --list-all`, and check you haven't been caught by `fail2ban` after failed logins.

"No space left on device." Find the full filesystem with `df -h`, then hunt the space hog with `du` (above). Frequent culprits are runaway logs in `/var/log`, an old un-rotated log, or a mail queue backed up in `/var/spool/mqueue`. The systemd journal can also grow — check it with `journalctl --disk-usage` and trim it with `journalctl --vacuum-size=200M`. On an LVM system, you may be able to extend the volume with `lvextend -r`.

"Cannot send mail" / mail not arriving. Inspect the queue and the maillog:

```
$ mailq                              # messages stuck in the queue, with the reason
$ tail -n 50 /var/log/maillog        # delivery attempts and rejections, with text
```

Outbound mail rejected by the receiving server is usually a missing reverse-DNS (PTR) record, SPF, or DKIM — see Chapter 4 and, for the PTR record on your IP, GSP support (Appendix C).

> **Tip:** When you contact GSP support (Appendix C), the single most useful thing you can include is the exact error line from the relevant log — copied and pasted, not paraphrased — together with the command you ran. "It doesn't work" takes a long back-and-forth to diagnose; one line from `/var/log/maillog` or an Apache error log often answers the question outright.

A Troubleshooting Checklist

When something seems wrong and you're not sure where to start, working through these roughly in order catches most issues:

- **Is the VPS itself reachable?** `ping example.com` (or its IP address). If not, the problem may be DNS, your network, or — rarely — the VPS itself; check GSP's status page or contact support.
- **Can you log in via SSH?** If SSH works but a service doesn't, the problem is in that service, not the VPS as a whole.
- **Is the relevant service running?** `systemctl status httpd`, `systemctl status sendmail`, etc. — or Webmin's Bootup and Shutdown module (Chapter 3).
- **What does the log say?** Almost every chapter names the relevant log file (or `journalctl -u <service>`) — check it for errors with a timestamp matching when the problem started.
- **Is the disk full?** `df -h` — a full disk causes failures in services that seem unrelated to storage.
- **Did something change recently?** A config edit, a package upgrade, an application update — `dnf history` shows exactly what was installed or updated, and when.

Important Commands and Files

Item	Purpose
/etc/logrotate.conf, /etc/logrotate.d/	Log rotation rules
journalctl -u <service>	Read a service's logs from the systemd journal
crontab -e	Edit your personal scheduled jobs
/etc/cron.d/, /etc/cron.{daily,weekly,monthly}/	System-wide and periodic scheduled jobs
systemctl list-timers	List systemd timers (the modern cron alternative)
top, ps aux, uptime, vmstat 1	Inspect current resource usage
df -h, du -h -d 1	Disk space per filesystem / per directory
tar, rsync, mysqldump/pg_dump	File and database backups
lvcreate -s, lvconvert --merge, lvs	LVM point-in-time snapshots

Where to Go Next

Chapter 10 covers the security side of ongoing maintenance — keeping Rocky Linux and everything you've installed patched, configuring the firewall, and the SELinux model that underlies so much of this handbook.

Chapter 10: Securing Your VPS

Security isn't a single setting you turn on — it's a handful of habits and a small amount of configuration, most of which only needs to be done once. This chapter covers staying patched, the `firewalld` firewall, SSH hardening, blocking repeat offenders with fail2ban, and the SELinux model that confines your services — plus a checklist for the services covered in earlier chapters.

Staying Patched

One of Linux's conveniences compared with some other systems is that a single tool — `dnf` — updates everything: the kernel, the base operating system, and every package you've installed, all from the same command.

dnf upgrade and Security Errata

```
# dnf check-update         # list packages with available updates
# dnf upgrade              # install all available updates
# dnf upgrade --security   # install only updates that fix security issues
```

Rocky Linux publishes **errata** that tag each update as a bugfix, enhancement, or security fix. The `dnf updateinfo` family of commands cross-references your installed packages against these and flags anything with a known security issue:

```
# dnf updateinfo list --security      # security updates available, by advisory ID
# dnf updateinfo info --security       # the full advisories, with CVE references
```

Run this regularly and treat anything it flags as a priority — these are publicly known vulnerabilities, which means automated scanners are looking for VPS instances that haven't yet applied the fix. Some updates (a new kernel, glibc, systemd) only take full effect after a reboot; `needs-restarting -r` tells you when a reboot is actually required, and `needs-restarting -s` lists services that should be restarted to pick up an updated library.

Automating Updates with dnf-automatic

To check for (and optionally apply) updates on a schedule without remembering to, install `dnf-automatic`:

```
# dnf install dnf-automatic
# vi /etc/dnf/automatic.conf      # set apply_updates = yes (or leave no to only notify)
# systemctl enable --now dnf-automatic.timer
```

With `apply_updates = no` (the cautious default), it emails you when updates are available but installs nothing — the right choice for a server you don't want changing unattended. Setting it to `yes`, or using the `dnf-automatic-install.timer`, applies them automatically; many administrators enable automatic *security* updates while reviewing the rest by hand.

> **Tip:** Webmin's Software Package Updates module (Chapter 3) shows the same information graphically and lets you apply updates with a click, if you'd rather not remember the commands above.

The firewalld Firewall

Rocky Linux's firewall is `firewalld`, and — unlike on some systems — it is **installed and enabled by default**, with a default zone (`public`) that already blocks inbound traffic except SSH. A firewall's job is to allow only the network traffic your VPS actually needs and reject everything else, reducing the "surface area" available to an attacker even for services you haven't gotten around to securing individually.

`firewalld` works in terms of **zones** (your public-facing interface is in the `public` zone) and **services** (named bundles of ports — `http` means 80, `https` means 443, and so on). See the current state with:

```
# firewall-cmd --get-active-zones
# firewall-cmd --list-all
```

A Starting Ruleset

Open exactly the services a web-and-mail VPS needs. The `--permanent` flag writes the rule to disk; a `--reload` then applies it:

```
# firewall-cmd --permanent --add-service=ssh
# firewall-cmd --permanent --add-service=http --add-service=https
# firewall-cmd --permanent --add-service=smtp --add-service=smtp-submission
# firewall-cmd --permanent --add-service=imaps --add-service=pop3s
# firewall-cmd --reload
# firewall-cmd --list-all          # confirm what is now allowed
```

This allows SSH (22), web traffic (80/443), and mail (25 for server-to-server delivery, 587 for authenticated submission, 993/995 for IMAPS/POP3S). Notably **absent** is port 10000 (Webmin) — add it explicitly only if you've decided not to use the SSH-tunnel approach from Chapter 3, and even then, restrict it to the addresses you connect from with a rich rule:

```
# Allow Webmin only from one office address
# firewall-cmd --permanent --add-rich-rule='rule family="ipv4" \
    source address="198.51.100.7" port port="10000" protocol="tcp" accept'
# firewall-cmd --reload
```

> **Warning:** Test firewall changes carefully — a mistake that blocks port 22 locks you out of SSH, and from there, out of fixing the firewall remotely. A useful safety net is `firewall-cmd --add-service=...` *without* `--permanent`: a runtime-only rule is discarded on the next reload or reboot, so if a change locks you out, simply rebooting restores the last saved ruleset. Before making changes, check whether GSP's control panel for your VPS provides a "console" option that works even if the network is unreachable; if it does, have it open in another tab while you test. If you do get locked out with no console, GSP support can help recover access — see Appendix C.

Once you're happy with a runtime ruleset, promote it to permanent in one step with `firewall-cmd --runtime-to-permanent`. Underneath, `firewalld` programs the kernel's

`nftables` engine; you rarely need to touch `nft` directly, but it's there if you want to see the raw rules with `nft list ruleset`.

SSH Hardening

Chapter 1 mentioned that GSP disables direct root login by default. Two further changes meaningfully reduce the most common attack against SSH — automated bots trying common username/password combinations.

Switch to Key-Based Authentication

On your own computer (not the VPS):

```
$ ssh-keygen -t ed25519 -C "your-email@example.com"
```

This creates a private key (`~/.ssh/id_ed25519` — never share this) and a public key (`~/.ssh/id_ed25519.pub` — safe to share). Copy the public key to your VPS:

```
$ ssh-copy-id youruser@example.com
```

(or, if `ssh-copy-id` isn't available, `cat ~/.ssh/id_ed25519.pub | ssh youruser@example.com 'mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys'`.) Confirm you can log in with the key — you should not be prompted for your account password, only (if you set one) the passphrase for the key itself, which is checked locally on your computer and never sent to the server.

Disable Password Authentication

Once key-based login is confirmed working — and **only then**, since this step removes the fallback — create a drop-in file (Rocky's `sshd_config` ends with `Include /etc/ssh/sshd_config.d/*.conf`, so drop-ins are the clean way to override defaults):

```
# /etc/ssh/sshd_config.d/00-hardening.conf
PasswordAuthentication no
PermitRootLogin no
# sshd -t                 # check the config parses
# systemctl restart sshd
```

> **Warning:** Test this in a **second terminal window**, leaving your current SSH session open. If something is wrong with the key setup, the second window will fail to connect — and your still-open first session lets you fix the drop-in and `systemctl restart sshd` again without being locked out entirely.

> **Note:** SSH's available ciphers and key-exchange algorithms are governed by Rocky's system-wide crypto policy (Chapter 7), so you usually don't — and shouldn't — hand-edit cipher lists in `sshd_config`. `update-crypto-policies --show` reports the active policy; switching it to `FUTURE` tightens SSH (and TLS everywhere else) in one move.

fail2ban: Blocking Repeat Offenders

`fail2ban` watches log files for patterns you define ("failed password," "authentication failure," "bad bot"), and when one address trips a rule too many times it adds a temporary firewall block. Because it works from logs, it can protect anything that logs failures — SSH, Sendmail/Dovecot, Webmin, Roundcube, even Apache Basic-auth — which is why it is the standard choice on Rocky Linux. Install it from EPEL (Appendix A), with the firewalld integration:

```
# dnf install fail2ban fail2ban-firewalld
# systemctl enable --now fail2ban
```

`fail2ban` is configured under `/etc/fail2ban/`. The cardinal rule is to **never edit the `.conf` files** — they are replaced on upgrade — but to put your changes in matching `.local` files, which override them. Create `/etc/fail2ban/jail.local`:

```
# /etc/fail2ban/jail.local
[DEFAULT]
# Enforce bans through firewalld
banaction = firewallcmd-rich-rules
banaction_allports = firewallcmd-rich-rules
# Whitelist your own address(es) so you can never lock yourself out
ignoreip  = 127.0.0.1/8 198.51.100.7
bantime   = 1h          # how long a ban lasts
findtime  = 10m         # window in which failures are counted
maxretry  = 5           # failures within findtime that trigger a ban

[sshd]
enabled  = true
```

Reload after editing, then inspect it with the `fail2ban-client` tool:

```
# systemctl reload fail2ban
# fail2ban-client status              # which jails are active
# fail2ban-client status sshd         # currently-banned IPs for the SSH jail
# fail2ban-client set sshd unbanip 203.0.113.99   # release one address
```

> **Note:** Add jails for the application-level services that bots actually hammer — `[dovecot]`, `[sendmail-auth]`, `[roundcube-auth]`, and `[webmin-auth]` are all built in; enable the ones whose services you run. A lighter-weight alternative focused on SSH alone is `sshguard` (also in EPEL); run one or the other for a given service, not both pointed at the same thing. Whichever you choose, always put your own IP in the `ignoreip` whitelist so a forgotten password can't lock you out of your own VPS.

SELinux: Mandatory Access Control

SELinux (Security-Enhanced Linux) is the layer that most distinguishes securing a Rocky Linux VPS from securing a BSD or a minimal Linux system, and it has come up throughout this handbook. It is worth understanding in its own right, because the right response to an SELinux denial is almost never to turn SELinux off.

Ordinary Unix permissions are *discretionary*: the owner of a file decides who may read it. SELinux adds a *mandatory* layer on top, enforced by the kernel regardless of ownership: every process and every file carries a **context** (a type label), and a central policy says which process types may touch which file types. The practical effect is **confinement** — the `apache`/`php-fpm` processes are allowed to read files labeled `httpd_sys_content_t` and little else, so a compromised web application can't read `/etc/shadow` or write outside its web directories, even though it is running with the privileges that would normally allow it.

Rocky Linux ships SELinux in **enforcing** mode by default. Check the current mode, and read the configuration, with:

```
$ getenforce                # Enforcing, Permissive, or Disabled
$ sestatus                  # fuller status, including the policy in use
```

The three modes: **Enforcing** (policy is applied — the default, and where you should stay), **Permissive** (violations are logged but allowed — useful for debugging), and **Disabled** (off entirely — don't). To switch to permissive *temporarily* while diagnosing a problem, `setenforce 0`; `setenforce 1` returns to enforcing. The persistent setting lives in `/etc/selinux/config`.

When a service can't do something and you suspect SELinux, the audit log is the place to look. The most useful tools:

```
# ausearch -m avc -ts recent           # recent SELinux denials (AVCs)
# sealert -a /var/log/audit/audit.log   # plain-English analysis + suggested fix
$ ls -Z /var/www/html                  # show the SELinux context of files
$ ps -eZ | grep httpd                  # show the context a process runs in
```

(`sealert` comes from the `setroubleshoot-server` package.) There are four ways to resolve a denial, in rough order of preference:

- **Fix the file label** when content is in the wrong place or carries the wrong type: `semanage fcontext -a -t httpd_sys_content_t "/path(/.*)?"` then `restorecon -Rv /path` (Chapters 6 and 8).
- **Flip a boolean** when the policy supports what you want but it's switched off: `getsebool -a | grep httpd` lists the toggles, and `setsebool -P httpd_can_network_connect on` makes one permanent (Chapter 8).
- **Label a non-standard port** when a service listens somewhere unusual: `semanage port -a -t http_port_t -p tcp 8080`.
- **Generate a local policy module** only as a last resort, when nothing above fits: `ausearch -m avc -ts recent | audit2allow -M mymodule` then `semodule -i mymodule.pp`. Read what `audit2allow` proposes before installing it — it should be a narrow, sensible permission, not a blanket grant.

> **Warning:** Resist the temptation to "fix" an SELinux problem by setting it to `Disabled`. Doing so throws away a working containment layer that, on a server running other people's code, is exactly the layer you most want. Almost every real denial is solved by one of the four targeted steps above; if you must, run

> *permissive* (which still logs everything) while you work the problem, then return to enforcing.

Hardening Checklist for Earlier Chapters

A quick recap of the security-relevant points from earlier chapters, gathered in one place:

Area	Setting	Covered in
SSH	PermitRootLogin no, key-based auth, fail2ban	This chapter, Ch. 1
Firewall	firewalld: only required services open; Webmin restricted by IP	This chapter, Ch. 3
SELinux	Left in enforcing mode; denials fixed with labels/booleans, not disabled	This chapter, Ch. 8
Sendmail	access table reviewed for unintended RELAY entries	Chapter 4
Roundcube/Dovecot	Always accessed over HTTPS/TLS	Chapter 4
Apache	ServerTokens Prod, ServerSignature Off, listing off, HTTPS + HSTS	Chapters 6-7
Web applications	Updated promptly; uploads can't execute scripts; secrets not world-readable	Chapter 8
Backups	Off-VPS copies exist and have been test-restored	Chapter 9

Staying Informed

- Rocky Linux security advisories are published at rockylinux.org and announced on the `rockylinux-announce` mailing list — subscribe to be notified of issues affecting the distribution.
- Because Rocky tracks Red Hat Enterprise Linux, the Red Hat CVE database and security advisories (access.redhat.com/security) describe the same fixes in detail, including CVE severity and affected versions.
- `dnf updateinfo list --security`, run regularly (above), covers everything installed via packages on the VPS itself.
- For formal compliance checks, `dnf install openscap-scanner scap-security-guide` provides OpenSCAP and the SCAP Security Guide — automated scans against benchmarks like CIS and STIG, for VPS owners who need to demonstrate a hardening baseline.

Where to Go Next

The appendices that follow cover installing additional software (Appendix A), a quick command reference (Appendix B), where to get help (Appendix C), and scripting and development tools (Appendix D).

Appendix A: Package Management — dnf, Repositories, and Building from Source

Every chapter in this handbook has used `dnf install` to add software. This appendix covers `dnf` more completely, explains the repositories your packages come from (including EPEL, which several chapters rely on), and — for the rare case where a package isn't enough — introduces building from source.

dnf Basics

Command	Effect
dnf install <name>	Install a package and its dependencies
dnf search <text>	Search the repositories by name or description
dnf info <name>	Show details (version, repo, description) for one package
dnf list installed	List installed packages
rpm -ql <name>	List every file a package installed
rpm -qf <path> / dnf provides <path>	Find which package owns a given file
dnf upgrade	Upgrade all installed packages
dnf remove <name>	Remove a package
dnf autoremove	Remove dependencies no longer needed by anything
dnf history	Show past transactions (and undo one with dnf history undo)
dnf clean all	Remove cached package metadata and downloads
dnf updateinfo list --security	List available security updates (Chapter 10)

`dnf search` is often the fastest way to find the exact package name for something. A typical first encounter with a new piece of software follows the same short arc — search, look before you leap, install, confirm:

```
$ dnf search nginx                 # find the exact package name
$ dnf info nginx                   # version, repository, and summary
$ dnf repoquery --requires nginx   # what would it pull in as dependencies?
# dnf install nginx                # install it (and its dependencies)
$ rpm -ql nginx | grep /etc        # where did it put its config files?
```

Keeping packages current is a two-step routine you'll run regularly (Chapter 10 folds it into a maintenance habit):

```
# dnf upgrade                      # install newer versions of what you have
# dnf autoremove                   # drop orphaned dependencies nothing needs anymore
# dnf clean all                    # reclaim disk space from the metadata/download cache
```

> **Tip:** Every `dnf` transaction is recorded. `dnf history` lists them; `dnf history info <id>` shows exactly what one transaction changed; and `dnf history undo <id>` rolls a transaction back — invaluable for answering "what changed?" after an upgrade, and occasionally for backing out an update that broke something. If an upgrade ever leaves a config file you had edited untouched and writes the new

> default alongside it as `*.rpmnew`, `dnf` notes it; merge your changes into (or from) the new file at your convenience.

Repositories

A fresh Rocky Linux system installs packages from a few repositories, and you'll add one or two more for software this handbook uses. List the ones currently enabled with `dnf repolist`.

- **BaseOS** — the core operating system: kernel, system libraries, core utilities. Stable and long-lived.
- **AppStream** — applications and runtimes that layer on top of BaseOS: Apache, PHP, MariaDB, language interpreters.
- **CRB (CodeReady Builder)** — additional libraries and development headers; disabled by default, and required by a handful of EPEL packages. Enable it with `dnf config-manager --set-enabled crb` (on the newer dnf5, `dnf config-manager setopt crb.enabled=1`).
- **EPEL (Extra Packages for Enterprise Linux)** — a large, community-maintained repository of software not in the base distribution. Several tools in this handbook live here: GoAccess, AWStats, Webalizer, fail2ban, Certbot, SpamAssassin, and Roundcube among them.

Because so much of this handbook draws on EPEL, enable it (and CRB) early:

```
# dnf install epel-release
# dnf config-manager --set-enabled crb
# dnf upgrade
```

Two more sources are worth knowing for specific needs. The **Remi** repository packages multiple PHP versions in parallel, useful when an application needs a PHP release other than the AppStream default. **COPR** is Fedora's build service for community projects — enable an individual project's repository with `dnf copr enable <user>/<project>`. Add third-party repositories deliberately and from sources you trust: each one is software allowed to install onto your VPS as root.

Building from Source

Almost everyone should use `dnf` rather than building from source: it's faster (no compilation), and the packages are built and tested centrally. Two situations still call for building it yourself: a packaged program built with default options when you need a non-default build-time feature, or a version newer than any repository yet provides. There are two paths.

Rebuilding a source RPM (SRPM) is the closest equivalent to a "ports" build: you take the distribution's own recipe (its `.spec` file and sources), optionally tweak it, and produce your own binary RPM. Install the build tools and a source-download helper first:

```
# dnf install rpm-build rpmdevtools 'dnf-command(download)'
$ dnf download --source httpd                  # fetch the .src.rpm
$ rpmbuild --rebuild httpd-*.src.rpm           # build it as-is...
# ...or: rpm -i httpd-*.src.rpm, edit ~/rpmbuild/SPECS/httpd.spec, then rpmbuild -ba
```

For repeatable, clean builds that don't disturb your live system, `mock` (`dnf install mock`) compiles an SRPM inside a fresh, isolated chroot — the tool to reach for if you maintain custom packages across several servers, much as you would with a dedicated build host.

Compiling an upstream tarball the traditional way works too; install the compiler toolchain (Appendix D) and use the familiar three-step dance:

```
# dnf group install "Development Tools"
$ ./configure --prefix=/usr/local
$ make
# make install
```

> **Note:** Keep hand-built software separate from `dnf`-managed software: install it under `/usr/local` (the default `--prefix` above), which `dnf` never touches, so the two can't collide. Mixing a self-compiled program into `/usr` where a package expects to manage the same files leads to exactly the version mismatches that packaging exists to prevent. For anything you'll install more than once, packaging it as an RPM (above) keeps even your custom builds under `dnf`'s management.

A Note on Image Mode (bootc)

Rocky Linux 10 includes, as a technology preview, **image mode** — an alternative way of managing a system in which the entire operating system (kernel, base, and your added packages) is built and shipped as a single bootable container image with `bootc`, and updates are applied by swapping the whole image atomically rather than package-by-package with `dnf`. The goal is a server whose exact software state is reproducible and rolls back cleanly. For Rocky Linux 10, the traditional **package mode** — `dnf` for everything, as described in this appendix and Chapter 10 — remains the primary, supported approach for most installations, including standard GSP VPS images. If you're curious, Rocky's documentation describes how to try image mode — but there's no need to for anything in this handbook.

Appendix B: Quick Command Reference

A one-page reference for commands covered throughout this handbook, grouped by topic.

Files and Navigation

Command	Effect
pwd	Print current directory
ls, ls -la	List files (all, with details)
cd <dir>, cd .., cd ~	Change directory (up one level, home)
cp, cp -r	Copy a file (recursively, for directories)
mv	Move or rename
rm, rm -r	Remove a file (recursively, for directories — careful!)
mkdir, mkdir -p	Create a directory (and parents as needed)
cat <file>	Print a file's contents
less <file>	View a file one screen at a time (q to quit)
tail -f <file>	Follow a growing file (e.g., a log) in real time
grep <pattern> <file>	Search a file for lines matching a pattern

Permissions and Ownership

Command	Effect
chmod 644 <file>	Owner read/write, others read-only (typical for a web page)
chmod 755 <file>	Owner read/write/execute, others read/execute (script or directory)
chmod -R <mode> <dir>	Apply recursively
chown <user>:<group> <file>	Change owner and group (root only)
id <user>	Show a user's UID, GID, and groups
ls -Z <file>	Show the SELinux context of a file
restorecon -Rv <dir>	Reset files to the SELinux labels policy expects

Users

Command	Effect
useradd -m -s /bin/bash <user>	Create an account with a home directory and shell
userdel -r <user>	Remove an account and its home directory
usermod -aG <group> <user>	Add a user to a secondary group
usermod -L / -U <user>	Lock / unlock an account without deleting it
passwd [user]	Change a password
chage <user>	Edit password aging / account expiry
sudo <command>	Run one command as root
sudo -i	Start a full root shell

Processes and Resource Usage

Command	Effect

top	Live process/resource monitor (q to quit)
ps aux	List all running processes
kill <pid>, kill -9 <pid>	Terminate a process (forcefully)
uptime	Load averages and uptime
df -h	Disk space per filesystem
vmstat 1	Live memory/CPU statistics

Networking

Command	Effect
ssh user@host	Open a shell on a remote host
sftp user@host	Open a file-transfer session
scp <src> <dst>	Copy a file over SSH
rsync -avz <src> <dst>	Efficiently sync files
ip addr	Show network interfaces and addresses (replaces ifconfig)
ss -tulpn	Show listening ports and their programs
dig <domain>	Query DNS records
ping <host>	Test basic connectivity

Packages and Services

Command	Effect
dnf install/search/info/upgrade/remove <name>	Package management (Appendix A)
dnf updateinfo list --security	List available security updates
systemctl start/stop/restart/status <name>	Control a service
systemctl enable --now <name>	Enable a service at boot and start it now
journalctl -u <name>	Read a service's logs
apachectl configtest	Check Apache configuration before reloading

Firewall and SELinux

Command	Effect
firewall-cmd --list-all	Show what the firewall currently allows
firewall-cmd --permanent --add-service=https	Open a service (then --reload)
firewall-cmd --reload	Apply permanent firewall changes
getenforce	Show the SELinux mode (Enforcing/Permissive/Disabled)
setsebool -P <boolean> on	Turn an SELinux boolean on permanently
ausearch -m avc -ts recent	Show recent SELinux denials

Mail

Command	Effect
newaliases	Rebuild the mail aliases database after editing /etc/aliases
make -C /etc/mail	Rebuild Sendmail's tables and configuration

tail -f /var/log/maillog	Watch mail activity
mailq	List messages stuck in the mail queue

Appendix C: Getting Help

This handbook covers the situations most GSP customers run into, but Rocky Linux, the software covered here, and the wider Internet are each much larger than any single document. When you need to go further:

Rocky Linux and RHEL Documentation

- The **Rocky Linux Documentation** (docs.rockylinux.org) is the authoritative guide to Rocky itself — installation, administration, networking, and guides on many of the services in this handbook, kept up to date with each release.
- Because Rocky Linux is binary-compatible with Red Hat Enterprise Linux, the **Red Hat Enterprise Linux documentation** (docs.redhat.com) applies almost verbatim and is exceptionally thorough — particularly its guides to SELinux, networking, and system administration.
- **Manual pages**, accessed with `man <command>` (e.g., `man dnf`, `man firewall-cmd`, `man sshd_config`), document every command and configuration file on your system in detail, and are always for the exact version installed on your VPS.
- **Manual pages online.** The same manual pages are also published on the web at www.gsp.com/support/man/ — handy for reading a man page from your phone or another computer, sending a colleague a link to one, or looking something up before you've even logged in. On the VPS itself, `man <command>` remains the authoritative copy, since it always matches the exact version you have installed.
- The **Rocky Linux community** — the forums at forums.rockylinux.org, the project's chat, and the r/RockyLinux subreddit — is active and welcoming for questions about Rocky itself, as opposed to the specific applications covered in Chapters 4-8.

Application Documentation

For software covered in this handbook, the project's own documentation is the best source for anything beyond what's covered here:

- **Webmin**: webmin.com/docs.html
- **Apache**: httpd.apache.org/docs/2.4/
- Sendmail: sendmail.org (and the Rocky/RHEL mail-server documentation)
- **Dovecot**: doc.dovecot.org
- **Roundcube**: roundcube.net/about/docs
- **PHP**: php.net/docs.php
- MariaDB: mariadb.com/kb/

For applications you've installed on top of all this — content management systems, e-commerce platforms, and so on — that application's own documentation and support community is the right first stop; this handbook covers the VPS underneath it, not every application that might run on it.

GSP Support

If you've worked through the relevant chapter and the application's own documentation and are still stuck — or if something seems wrong with the VPS itself (network connectivity, disk, or anything that prevents you from reaching it at all) — GSP support is available. Having the following ready when you contact us speeds things up considerably:

- Your VPS's hostname or IP address
- What you were trying to do, and the exact command or steps you took
- The exact error message, if any (a screenshot or copy-pasted text, not a paraphrase)
- The relevant section of the log file, if you found one (Chapter 9 covers where to look)

Appendix D: Scripting and Development Tools

A VPS is not only a place to run software — it is a place to write it, whether that means a ten-line Perl script that rotates a report, a C program a vendor asked you to compile, or glue code that ties your services together. This appendix covers the two things customers most often ask about once they're comfortable at the command line: scripting with Perl (and managing its CPAN modules), and the compilers available for building software from source.

Scripting with Perl

Perl has been a Unix system-administration mainstay for decades, and it is still one of the best tools for the everyday job of "read some text, pull out the parts that matter, and do something with them" — exactly what log files, configuration files, and reports require. Rocky Linux often has Perl present already (several system tools use it); install it explicitly if a minimal image left it out:

```
# dnf install perl
$ perl -v                       # confirm the version
```

A Perl script is a plain text file. Make the first line a "shebang" pointing at the interpreter, mark the file executable, and run it:

```
$ cat > hello.pl <<'EOF'
#!/usr/bin/perl
use strict;
use warnings;

my $name = shift // "world";   # first argument, or "world" if none
print "Hello, $name!\n";
EOF
$ chmod +x hello.pl
$ ./hello.pl Dan
Hello, Dan!
```

`use strict;` and `use warnings;` at the top of every script are not optional habits to skip — they catch the typos and undeclared-variable mistakes that cause most Perl bugs, and turn silent misbehavior into a clear message. Where Perl really earns its keep on a server is one-liners and short filters. A few that come up often:

```
# Count how many times each IP appears in an access log
$ perl -lane '$c{$F[0]}++; END{ print "$c{$_}\t$_" for sort {$c{$b}<=>$c{$a}} keys %c }' \
    /var/log/httpd/example.com-access.log | head

# In-place find-and-replace across files (with a .bak backup of each)
$ perl -i.bak -pe 's/old\.example\.com/new.example.com/g' *.conf

# Pull every email address out of a file
$ perl -nE 'say $1 while /([\w.+-]+@[\w.-]+)/g' message.txt
```

The flags are worth learning once: `-e` runs the code on the command line, `-n` wraps it in a loop over input lines, `-p` does the same but prints each line afterward (ideal for edits), `-l`

handles line endings, `-a` auto-splits each line into the `@F` array, and `-i` edits files in place. Together they turn Perl into a precise, scriptable replacement for a pile of `sed`/`awk` incantations.

Managing CPAN Modules

CPAN — the Comprehensive Perl Archive Network — is Perl's enormous library of reusable modules (database drivers, web frameworks, date handling, JSON, almost anything). There are three ways to install a module on a Rocky Linux VPS, and choosing the right one avoids a lot of grief.

1. Prefer the Rocky package when one exists. Most popular CPAN modules are packaged with a `perl-` prefix (in AppStream or EPEL), and installing them this way means `dnf upgrade` keeps them current and there is nothing to compile:

```
$ dnf search perl-JSON
# dnf install perl-JSON perl-LWP-Protocol-https perl-DBD-MySQL
```

2. Use cpanm for modules that aren't packaged. `App::cpanminus` is a modern, zero-configuration CPAN client — far friendlier than the traditional `cpan` shell:

```
# dnf install perl-App-cpanminus
# cpanm Text::CSV               # install system-wide (needs root)
$ cpanm --local-lib=~/perl5 Text::CSV    # ...or into your own home directory
```

Installing into a personal directory with `--local-lib` (or the `local::lib` module, `dnf install perl-local-lib`) is the safest habit: it keeps your modules separate from the system Perl, so a package upgrade can't disturb them and you don't need root. Tell Perl where to find them by adding this to your shell startup file (Chapter 1):

```
eval $(perl -I$HOME/perl5/lib/perl5 -Mlocal::lib=$HOME/perl5)
```

3. The classic cpan shell is always available as a fallback. The first run asks a few setup questions (accept the defaults):

```
# cpan
cpan[1]> install Some::Module
cpan[2]> notest install Some::Module    # skip the test suite if it's flaky
cpan[3]> quit
```

> **Note:** Some CPAN modules are **XS** modules — they include C code and must be compiled, which means they need a C compiler (next section) and sometimes a system library's development headers (the `-devel` packages). If a `cpanm` install fails, scroll up in its output (or read `~/.cpanm/build.log`): the real error is usually a missing library that you can install with `dnf install <name>-devel`, after which the module builds cleanly.

Compilers: gcc, clang, and Building Software

Unlike the base operating system, a compiler toolchain isn't installed by default on a minimal Rocky Linux system — add it in one step. The default compiler on Linux is GCC, reached through the traditional names `cc` and `gcc`:

```
# dnf group install "Development Tools"      # gcc, g++, make, and the usual build tools
# ...or just the essentials: dnf install gcc gcc-c++ make
$ cc --version                               # this is GCC on Linux
$ cat > hello.c <<'EOF'
#include <stdio.h>
int main(void){ printf("Hello from Rocky Linux!\n"); return 0; }
EOF
$ cc -O2 -Wall -o hello hello.c
$ ./hello
Hello from Rocky Linux!
```

Most open-source projects build with the familiar three-step dance; the toolchain above handles it without anything extra:

```
$ ./configure
$ make
# make install
```

> **Note:** On Linux, `make` is **GNU make**, which is what the large majority of projects expect — so, unlike on BSD systems, you rarely need anything else. The exception is the occasional project written for BSD `make`; if a `Makefile` fails with unfamiliar syntax errors and the project documents BSD origins, install `dnf install bmake` and run `bmake` instead.

If you specifically need (or prefer) the Clang/LLVM compiler — some projects build only with it, or you want its diagnostics — install it alongside GCC; the two coexist cleanly:

```
# dnf install clang
$ clang -O2 -Wall -o hello hello.c
$ make CC=clang                # point a build at clang instead of gcc
```

> **Note:** Other languages are equally close at hand: Python 3 is part of the base system (`python3`), and interpreters and toolchains for Node.js, Go, Ruby, Rust, and more are a `dnf install` away (some from AppStream, some from EPEL). The same packaging principles in Appendix A apply — prefer the packaged version, and keep anything you build by hand under `/usr/local`.