Your FreeBSD VPS

A GSP Administration Handbook

Covering FreeBSD 15, 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

 2026 EDITION

Table of Contents

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 are logged in directly as root or because you have switched to root with `su` or run the command with `doas`.

For example:

```
$ ls -l
# pkg install apache24
```

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

File and directory names are shown in monospace, such as `/usr/local/etc/apache24/httpd.conf`. Where a path includes a placeholder you must replace, it appears in angle brackets, such as `/usr/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 FreeBSD 15 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 is FreeBSD's traditional way of marking accounts that are allowed to become root via `su`, and — once you install it — `doas`. 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. FreeBSD 15 includes everything needed for this in the base system — there is nothing to install.

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 — by default, FreeBSD 15 sets new accounts to use `sh`, though `csh`, `tcsh`, `bash`, and `zsh` are all available via `pkg install` if you prefer a different shell or its interactive features (history, tab completion, etc.).

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 FreeBSD Basics

FreeBSD is a Unix operating system, and if you've never used one 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 `ee` (FreeBSD's built-in beginner-friendly editor), `nano`, or `vi`

If you've administered Linux before, most of this will feel familiar — FreeBSD's commands and file layout differ from Linux in some details (for example, configuration for software you install yourself lives under `/usr/local/etc`, not `/etc`), but the underlying concepts are the same.

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 (`adduser`, 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 `/usr/local/www/apache24/data` 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. FreeBSD 15 doesn't include a statistics tool out of the box, but two are a single `pkg install` away:

- **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, and the security habits that go with them
- **Chapter 9** — logs, backups, and keeping an eye on your VPS's health
- **Chapter 10** — keeping FreeBSD and everything on it patched and locked down
- **Appendix A** — installing additional software with `pkg` and the Ports Collection
- **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 FreeBSD, and welcome to your new VPS.

Chapter 1: Introducing Your FreeBSD VPS

How the System Works

Your GSP VPS is a complete, private FreeBSD 15 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 minimal, current FreeBSD 15 installation. FreeBSD's philosophy is to keep the base system small and let you add exactly the software you need with the package system, `pkg`. A fresh VPS already includes:

- The FreeBSD base system: kernel, shell, core utilities, OpenSSH (for remote login and file transfer), and the `pkg` package manager
- A firewall (`pf`) installed but not yet configured — see Chapter 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 `pkg install` command followed by some configuration. Think of the rest of this handbook as a guided tour of turning a blank FreeBSD 15 system into the web and mail server your site needs.

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 FreeBSD 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 FreeBSD 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.

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 (Chapter 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-click-style installs (`pkg`, 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 FreeBSD'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 FreeBSD, membership in `wheel` is what `su` checks before allowing a user to become root (assuming they also know the root password). Once you install `doas` (Chapter 2), the same account can run individual commands as root without a full session switch.

In practice, this gives you a hierarchy:

```
root
  └── members of "wheel" (your administrative user, and any
       colleagues you choose to add)
        └── 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 `su` or `doas` only for the specific commands that genuinely need root — installing packages, editing files under `/etc` or `/usr/local/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 `su`. This is a security best practice (it ensures every root session 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 part of the FreeBSD base system (as `sshd`, the server, and `ssh`, the client) 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
FreeBSD 15.0-RELEASE (GENERIC) releng/15.0
$
```

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 FreeBSD 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 needed even in single-user mode
`/etc`	Configuration for the FreeBSD base system (e.g., `/etc/rc.conf`, `/etc/ssh/sshd_config`)
`/usr`	The bulk of the base system: user commands (`/usr/bin`), manual pages (`/usr/share/man`), and…
`/usr/local`	…everything you install yourself with `pkg`, including its own `etc`, `bin`, and `www` subdirectories
`/usr/local/etc`	Configuration for installed packages (Apache, Sendmail, Dovecot, Webmin, etc.)
`/usr/local/www`	Default document roots for installed web servers
`/var`	Frequently-changing data: logs (`/var/log`), mail queues (`/var/spool`), and more
`/home` (a link to `/usr/home`)	Personal directories for ordinary user accounts
`/root`	The root account's home directory
`/tmp`	Temporary files, cleared periodically

The single most important habit to build is this: **software FreeBSD ships with configures itself under `/etc`; software you add with `pkg` configures itself under `/usr/local/etc`.** When this handbook says "edit Apache's configuration," it means a file under `/usr/local/etc/apache24/`, not `/etc`.

Navigating the File System

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

```
$ pwd                  # print working (current) directory
$ cd /usr/local/etc    # 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 FreeBSD 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    512 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.

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

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.

Editing Files

Sooner or later, every chapter in this handbook involves editing a configuration file. FreeBSD 15 includes a simple screen editor, `ee`, in the base system — no installation required — and `nano` and `vi`/`vim` are both one `pkg install` away if you prefer them.

ee (Easy Editor)

ee is designed to be usable without any prior knowledge: when you open a file, the bottom of the screen lists the available commands, all triggered with the Control key.

```
$ ee /usr/local/etc/apache24/httpd.conf
```

The most useful shortcuts are shown on screen, but the essentials are:

- **Ctrl+K**: cut the current line (also used to cut a selected block)
- **Ctrl+U**: paste (un-cut)
- **Ctrl+W**: search
- **Ctrl+X**: exit, with a prompt to save changes

nano

If you've used nano on Linux before, FreeBSD's is the same program:

```
# pkg install nano
$ nano /usr/local/etc/apache24/httpd.conf
```

Like ee, 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 few options make nano much more pleasant for editing configuration files. Turn them on for a single session from the command line:

```
$ nano -wl /etc/rc.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 (or, for every user, the system-wide /usr/local/etc/nanorc). The single most useful thing to put there is the set of **syntax-highlighting definitions** that ship with the package — they colorize Apache, Sendmail, shell, Perl, and dozens of other file formats so mistakes stand out. A good starting ~/.nanorc:

```
freebsd@gsp:~ % cat .nanorc
## ~/.nanorc — nano configuration file

# Include all syntax highlighting definitions
include "/usr/local/share/nano/*.nanorc"
include "/usr/local/share/nano/extra/*.nanorc"

# A few quality-of-life 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
```

The two include lines are the important part: they pull in every *.nanorc highlighting definition the package installed under /usr/local/share/nano/ (and the extra set), so opening, say, httpd.conf or a Perl script immediately shows keywords, strings, and comments in distinct colors. Confirm the definitions are present with ls /usr/local/share/nano/; if that directory is empty, install the package's highlighting files with pkg install nano (they are included with current versions).

vi / vim

vi is the traditional Unix editor — always present on every Unix system, including a freshly installed FreeBSD base with no packages added — 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 /usr/local/etc/apache24/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.

`pkg install vim` provides a more featureful version (`vim`) with 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**. FreeBSD ships several, and you can install more; they 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`: it is the one shell guaranteed to be present.

Shell	Family	Where it comes from	Best for
`sh`	Bourne	Base system	Scripts; the portable default. FreeBSD 15 gives new accounts `sh`.
`csh`	C shell	Base system	Historical FreeBSD interactive default; root's traditional shell
`tcsh`	C shell	Base system	`csh` with command history, completion, and editing

`bash`	Bourne	`pkg install bash`	The Linux default; familiar, huge ecosystem of scripts
`ksh`	Bourne	`pkg install ksh93`	The Korn shell; powerful scripting, POSIX-compatible
`zsh`	Bourne	`pkg install zsh`	Interactive power-user shell (themes, smart completion)
`tclsh`	Tcl	`pkg install tcl`	Not a login shell — the Tcl scripting-language interpreter
`scotty`	Tcl	`pkg install tkined/scotty`	Not a login shell — Tcl plus network-management commands (SNMP, DNS, ping)

> **Note:** `tclsh` and `scotty` are listed because people reach for them as "shells," but they are really interactive interpreters for the **Tcl** language rather than command shells you would log in with. `tclsh` runs Tcl scripts (`tclsh script.tcl`) and gives a `%` prompt for typing Tcl interactively; `scotty` is `tclsh` extended with the Tnm network-management commands, useful for scripted SNMP queries, DNS lookups, and reachability checks. You would not set either 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/sh
$ cat /etc/shells          # the list of shells allowed as login shells
```

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

```
$ chsh -s /usr/local/bin/bash      # change your own shell (logout/in to
take effect)
# pw usermod jsmith -s /usr/local/bin/zsh   # 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 (in every shell here except plain `sh` and plain `csh`, which lack interactive

completion — another reason to install `bash`, `zsh`, or use `tcsh`). 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)	`~/.profile`, `~/.bash_profile`, `~/.zprofile`	`~/.login`
Startup file (each shell)	`~/.bashrc`, `~/.zshrc`, `$ENV` for `ksh`	`~/.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. 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 FreeBSD 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: su and doas

FreeBSD's base system provides `su` ("substitute user," almost always used to become root):

```
$ su
Password:
#
```

You'll be prompted for the **root password** (not your own), and — if your account is in the `wheel` group — dropped into a root shell. Type `exit` or press Ctrl+D to return to your own account.

`su` is simple but coarse: it gives you a full root shell until you remember to exit, and it requires sharing the root password with everyone who needs occasional root access. **doas**, a small privilege-escalation tool originally from OpenBSD, addresses both issues and is the approach GSP recommends. It isn't part of the base system, so install it first:

```
# pkg install doas
```

Then create `/usr/local/etc/doas.conf` (root-owned, mode 600):

```
permit persist youruser as root
```

`persist` means doas will remember your authorization for a few minutes, similar to `sudo`'s default behavior, so you aren't prompted on every single command. With this in place:

```
$ doas pkg install nano
doas (youruser@example) password:
```

You authenticate with **your own password**, not root's, and only the single command runs with elevated privileges. For a colleague who should be able to manage packages and services but never read another user's mail, you can write a much narrower rule, for example:

```
permit nopass colleague cmd /usr/sbin/service
permit nopass colleague cmd /usr/sbin/pkg
```

> **Note:** If you've used Linux, `doas` plays the same role as `sudo`. `sudo` itself is also available via `pkg install sudo` if you're more comfortable with its syntax and ecosystem (e.g., existing scripts that call `sudo`) — the two are not mutually exclusive, but most new FreeBSD installations standardize on one or the other to keep `/usr/local/etc/doas.conf` or `/usr/local/etc/sudoers` as the single source of truth for who can do what.

Creating New User Accounts

adduser

`adduser` is FreeBSD's interactive account-creation tool and the easiest way to add a normal login:

```
# adduser
Username: jsmith
Full name: Jane Smith
Uid (Leave empty for default):
Login group [jsmith]:
Login group is jsmith. Invite jsmith into other groups? []: wheel
Login class [default]:
Shell (sh csh tcsh bash zsh nologin) [sh]: bash
Home directory [/home/jsmith]:
Home directory permissions (Leave empty for default):
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username   : jsmith
Password   : *****
Full Name  : Jane Smith
Uid         : 1002
Class       :
```

```
Groups      : jsmith wheel
Home        : /home/jsmith
Shell       : /usr/local/bin/bash
OK? (yes/no): yes
adduser: INFO: Successfully added (jsmith) to the user database.
Add another user? (yes/no): no
Goodbye!
```

Answering "wheel" at the "Invite into other groups" prompt is what would let this account use `su`/`doas` to become root — leave it blank for an account that should never have administrative access.

> **Note:** `adduser` proposes a home directory of `/home/jsmith`, while later chapters refer to paths such as `/usr/home/youruser`. These point to the same place: on FreeBSD `/home` is a symbolic link to `/usr/home`, so either form works wherever you see it.

pw useradd

`pw` is the lower-level tool `adduser` itself calls, and is better suited to scripting or to creating many similar accounts. To create a non-administrative account with a home directory and a Bash shell in one line:

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

Common `pw useradd` options:

Option	Meaning
`-n name`	Username
`-c comment`	"Full name" / GECOS comment
`-d dir`	Home directory
`-m`	Create the home directory if it doesn't exist
`-s shell`	Login shell
`-G group,...`	Additional (secondary) group memberships
`-g group`	Primary login group

Editing Existing Accounts

chpass

`chpass` opens an editable summary of an account in your default editor:

```
# chpass jsmith
#Changing user database information for jsmith.
Login: jsmith
Password: $6$abc123...
Uid [#]: 1002
Gid [# or name]: 1002
Change [month day year]:
Expire [month day year]:
Class:
Home directory: /home/jsmith
Shell: /usr/local/bin/bash
Full Name: Jane Smith
Office Location:
Office Phone:
Home Phone:
Other information:
```

Save and exit your editor to apply the changes. As an ordinary user, running `chpass` (without #) on your own account lets you edit the small set of fields you're allowed to change — your shell and contact-information fields — without root.

passwd

To change a password — your own, or (as root) anyone's:

```
$ passwd                # change your own password
# passwd jsmith         # change jsmith's password (root only)
```

pw usermod

For one-off changes without opening an editor:

```
# pw usermod jsmith -s /usr/local/bin/zsh        # change shell
# pw usermod jsmith -G wheel                     # add to wheel (grants
su/doas access)
```

Disabling and Removing Accounts

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

```
# pw lock jsmith
```

and to restore access later:

```
# pw unlock jsmith
```

A locked account can't log in (including via SSH key, SFTP, or Roundcube/IMAP, since all of these check the same account status), but its files, mail, and crontab are untouched.

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

```
# rmuser jsmith
```

`rmuser` is interactive and asks for confirmation before each destructive step, which makes it harder to remove the wrong account by mistake than the lower-level `pw userdel`.

> **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 crontab entries referencing that user's scripts. Check for these before (or instead of) running `rmuser` 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),0(wheel),68(www)
```

To create a new group and add members:

```
# pw groupadd webteam
# pw groupmod webteam -m jsmith,asmith
```

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).

Login Classes and Resource Limits

FreeBSD groups default resource limits — maximum processes, open files, CPU time, and so on — into named **login classes**, defined in `/etc/login.conf`. Most accounts use the `default` class, which is generous enough for normal use. If you run a VPS that hosts several independent clients, you can create a dedicated login class with tighter limits (for example, capping how many processes or how much memory a single account's runaway script can consume) and assign it with:

```
# pw usermod jsmith -L clientclass
```

After editing `/etc/login.conf`, rebuild its database with:

```
# cap_mkdb /etc/login.conf
```

> **Note:** FreeBSD's traditional disk **quota** system (`quota`, `edquota`) works on UFS file systems. If your VPS uses ZFS (the default for new GSP VPS images), disk usage limits are instead set per-dataset with `zfs set quota=10G zroot/usr/home/jsmith` — see Chapter 9 for more on ZFS basics.

Important Commands and Files

Command/File	Purpose
`adduser`	Interactive account creation
`rmuser`	Interactive, safe account removal
`pw`	Low-level user/group management (`useradd`, `usermod`, `userdel`, `groupadd`, `groupmod`, `lock`, `unlock`)
`chpass`	Edit account details in an editor
`passwd`	Change a password
`id`	Show a user's UID, GID, and group memberships
`su`	Become another user (typically root)
`doas` / `sudo`	Run a single command as another user (typically root)
`/etc/passwd`	Account database (read with `cat`, never edit directly — use the tools above)
`/etc/group`	Group membership database
`/etc/login.conf`	Login class definitions and resource limits
`/usr/local/etc/doas.conf`	`doas` authorization rules

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 is available as a FreeBSD package:

```
# pkg install webmin
```

The package installs Webmin under `/usr/local/etc/webmin` (configuration) and `/usr/local/lib/webmin` (program files), along with an `rc.d` startup script. Enable and start it:

```
# sysrc webmin_enable=YES
# service webmin start
```

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/
```

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 for you 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 with your administrative user's username and password — the FreeBSD package configures Webmin to authenticate against the system's normal user database, so there's no separate Webmin-only password to keep track of, at least initially.

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 it to your administrative account(s). By default, any account in the `wheel` group can authenticate to Webmin and will have full root-equivalent access through it. 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.

Consider firewalling port 10000. If you (and any colleagues) always connect from a small set of known IP addresses, restricting port 10000 to those addresses with `pf` (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 in `pf` 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. `pkg 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 as tabs across the top of the screen, with the modules in the selected category listed down the left side. 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 `adduser`, `pw`, and `chpass`. 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 `chpass` does, 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 companion **Groups** tab lists and edits `/etc/group` — useful for the shared-web-team group described in Chapter 2 without needing to remember `pw groupmod` 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 `pkg`. It lists installed packages, flags ones with available updates (cross-referencing the same vulnerability database `pkg audit` uses — see Chapter 10), and lets you install new packages by searching the FreeBSD package repository. Running updates through this module is equivalent to `pkg upgrade` at the command line, with a confirmation screen showing exactly what will change first.

System → Bootup and Shutdown

This module lists every service FreeBSD knows how to start (everything with an `rc.d` script, including ones you've installed via `pkg`), shows whether each is enabled in `/etc/rc.conf`, and lets you start, stop, restart, enable, or disable a service with a click. It's the graphical equivalent of:

```
# sysrc apache24_enable=YES
# service apache24 start
```

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 (UFS or ZFS), and available space — a quick way to check whether your VPS is running low on disk space without remembering `df -h`. For ZFS systems, this module can also create and manage datasets, which is how GSP recommends setting per-account storage limits (Chapter 2).

Networking → Network Configuration / Firewall

Depending on the modules enabled on your VPS, this category may include a `pf` firewall editor. We recommend understanding `pf`'s rule syntax 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 `/etc/pf.conf` directly over 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 in the top-right corner of the screen, 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" FreeBSD 15 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:

1. 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.
2. A **mailbox store and retrieval server** holds delivered mail and lets users fetch it over IMAP or POP3. This is **Dovecot**.
3. **Mail clients** — Roundcube (webmail) and/or desktop and mobile apps — connect to the mailbox store to read mail, and to the MTA (or directly to Dovecot's submission service) to send it.

Why FreeBSD 15's Default Isn't Enough

FreeBSD 15 includes **dma** (the DragonFly Mail Agent) as its default mail handler, a deliberately minimal program that can hand outgoing mail from local programs (cron job output, system alerts, `pkg` notifications) to an upstream relay. It does not listen for incoming SMTP connections, doesn't support multiple hosted domains or virtual mailboxes, and has no concept of IMAP or POP3. This is sufficient 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 mail server. (Older FreeBSD versions, including the system the original VPS v2 handbook was written for, shipped with **Sendmail** filling this role; FreeBSD 15 removed it from the base system specifically because most installations don't need a full MTA, but it remains available — and well-supported — as a package.)

GSP's standard configuration for a VPS that hosts email is therefore:

```
# pkg install sendmail dovecot
```

The rest of this chapter assumes both are installed.

Setting Up Sendmail

Switching the System Mailer

FreeBSD's `mailwrapper(8)` mechanism means commands like `sendmail`, `mailq`, and `newaliases` are actually symlinks to `mailwrapper`, which consults `/etc/mail/mailer.conf` to decide which real program to run. After installing the `sendmail` package, edit `/etc/mail/mailer.conf` so these commands point at the new Sendmail binaries instead of `dma`:

```
sendmail        /usr/local/sbin/sendmail
send-mail       /usr/local/sbin/sendmail
mailq           /usr/local/sbin/sendmail
newaliases      /usr/local/sbin/sendmail
hoststat        /usr/local/sbin/sendmail
purgestat       /usr/local/sbin/sendmail
```

> **Note:** Confirm the exact installation path with `pkg info -l sendmail` — it has been `/usr/local/sbin/sendmail` across recent package revisions, but it's worth a quick check after any major upgrade.

Enable and start the service:

```
# sysrc sendmail_enable=YES
# sysrc sendmail_submit_enable=YES
# service sendmail start
```

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. The package installs a sample configuration directory — typically `/usr/local/etc/mail/` — containing a starting `.mc` file and a `Makefile` that wraps the `m4` invocation, so that after editing the `.mc` file you simply run:

```
# cd /usr/local/etc/mail
# make install restart
```

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 `/usr/local/etc/mail/local-host-names`, one per line:

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

After changing this file, `make install restart` again to reload Sendmail.

Aliases

`/etc/mail/aliases` (or `/usr/local/etc/mail/aliases`, depending on your `.mc`'s `ALIAS_FILE` setting) 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:/usr/local/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:

```
# /usr/local/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:

```
# cd /usr/local/etc/mail && make install restart
```

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/mail/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/mail/aliases`	`/usr/local/etc/mail/virtusertable` *(see your `.mc`)*
Rebuild after editing	`newaliases`	`make install restart` (in `/usr/local/etc/mail`)

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:

```
# /usr/local/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/mail/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 `aliases`, and `make install restart` after touching `virtusertable`.

Access Control and Basic Spam Defenses

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

```
# /usr/local/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:

```
# cd /usr/local/etc/mail && make install restart
```

For more substantial spam and virus filtering, `pkg install rspamd` (a modern content-filtering milter) 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, available via `pkg install vacation` (or already present depending on your Sendmail package's options), works through a user's `~/.forward` file:

```
\jsmith, "|/usr/local/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.

Setting Up Dovecot

Dovecot provides the IMAP and POP3 service that Roundcube and desktop/mobile clients connect to.

```
# pkg install dovecot
```

Dovecot's configuration lives under `/usr/local/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 (`conf.d/10-master.conf` and `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 FreeBSD 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 acme.sh):

```
ssl = required
ssl_cert = </usr/local/etc/apache24/certs/example.com/fullchain.pem
ssl_key = </usr/local/etc/apache24/certs/example.com/key.pem
```

Enable and start:

```
# sysrc dovecot_enable=YES
# service dovecot start
```

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.

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

```
# conf.d/10-master.conf — expose Dovecot's auth socket to Sendmail (for SMTP
AUTH)
service auth {
  unix_listener /var/run/dovecot/auth-client {
    mode = 0660
  }
  # Point Sendmail's SASL config (/usr/local/lib/sasl2/Sendmail.conf) at
```

```
this socket.
}

# conf.d/20-imap.conf – cap simultaneous connections from one IP (stops a
# misconfigured client from opening hundreds of sessions)
protocol imap {
  mail_max_userip_connections = 20
}

# conf.d/15-lda.conf and 20-lmtp.conf – enable the Sieve filter plugin so
# Roundcube's filters and vacation messages 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. The standard approach is **Cyrus SASL** configured to check passwords via Dovecot's authentication socket, so that the same username and password works for both sending (Sendmail) and receiving (Dovecot) mail. This involves:

1. `pkg install cyrus-sasl cyrus-sasl-saslauthd` (or configuring SASL to talk directly to Dovecot's `auth-client` socket, depending on your Sendmail package's SASL options)
2. Enabling `AUTH_PLAIN` and `AUTH_LOGIN` in Sendmail's `.mc` via `TRUST_AUTH_MECH` and `confAUTH_MECHANISMS`
3. Pointing `/usr/local/lib/sasl2/Sendmail.conf` at Dovecot's auth socket

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 FreeBSD

Handbook's mail chapter (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 FreeBSD at the operating-system level, and tuning SpamAssassin's own configuration.

At the FreeBSD (OS) level

Install SpamAssassin and the small "milter" that lets Sendmail hand each message to it:

```
# pkg install spamassassin spamass-milter
```

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

```
# sa-update                          # download the latest rules
# sysrc spamd_enable=YES
# sysrc spamd_flags="-c -m 5 --max-conn-per-child=16"   # -c = use per-user
prefs
# service sa-spamd start
```

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

```
# sysrc spamass_milter_enable=YES
# sysrc spamass_milter_socket="/var/run/spamass-milter.sock"
# sysrc spamass_milter_socket_owner="mailnull"
# sysrc spamass_milter_socket_group="mailnull"
# sysrc spamass_milter_socket_mode="660"
# service spamass-milter start
```

and add the milter to Sendmail's `.mc`, then rebuild (Sendmail setup is earlier in this chapter):

```
INPUT_MAIL_FILTER(`spamassassin', `S=local:/var/run/spamass-milter.sock, F=,
T=C:15m;S:4m;R:4m;E:10m')dnl
# cd /usr/local/etc/mail && make install restart
```

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/local/bin/sa-update && /usr/sbin/service sa-spamd reload
```

> **Note:** SpamAssassin scoring is CPU- and memory-hungry on a small VPS. Running it as `spamd` (above) and capping the child processes with `-m 5` keeps it from overwhelming a 1–2 GB VPS. 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 `/usr/local/etc/mail/spamassassin/local.cf`. The most commonly adjusted ones:

```
# /usr/local/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)
# service sa-spamd reload
```

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

```
# pkg install roundcube
```

This pulls in PHP and a PHP-FPM dependency 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:

```
# pkg install php-sqlite3
```

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

Configuring Roundcube

The package installs Roundcube's files under `/usr/local/www/roundcube`, including a sample configuration file. Copy it and edit the copy:

```
# cd /usr/local/www/roundcube/config
# cp config.inc.php.sample config.inc.php
```

The settings you need to change:

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

// IMAP server (Dovecot)
$config['default_host'] = 'tls://localhost';
$config['default_port'] = 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
$config['des_key'] = 'replace-this-with-24-random-characters';
```

Initialize the database schema:

```
# mkdir -p /var/db/roundcube && chown www:www /var/db/roundcube   # keep the
DB outside the web-served tree
# cd /usr/local/www/roundcube
# bin/initdb.sh --dir SQL
```

Publishing Roundcube on the Web

Add an alias to your Apache configuration (Chapter 6 covers Apache's configuration layout in detail) so that `https://example.com/webmail/` serves Roundcube:

```
Alias /webmail /usr/local/www/roundcube/public_html
<Directory /usr/local/www/roundcube/public_html>
    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.

> **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 Roundcube 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 `syslog`). 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. Chapter 9 covers log rotation so this file doesn't grow without bound.

Important Commands, Directories, and Files

Item	Purpose
`/etc/mail/mailer.conf`	Maps `sendmail`/`mailq`/etc. to the active MTA binary
`/usr/local/etc/mail/*.mc`, `Makefile`	Sendmail's m4 source configuration

`/usr/local/etc/mail/local-host-names`	Domains this server accepts mail for
`/usr/local/etc/mail/aliases`	Address-to-address forwarding
`/usr/local/etc/mail/virtusertable`	Per-domain / per-address mailbox mapping
`/usr/local/etc/mail/access`	Relay and reject rules
`newaliases`	Rebuild the aliases database
`cd /usr/local/etc/mail && make install restart`	Rebuild Sendmail's tables/config and restart
`/usr/local/etc/dovecot/`	Dovecot configuration
`~/Maildir`	A user's mailbox storage
`/var/log/maillog`	Mail server log
`/usr/local/www/roundcube/config/config.inc.php`	Roundcube configuration

For More Information

- The FreeBSD Handbook's chapter on Electronic Mail (`docs.freebsd.org`) covers Sendmail, Dovecot, and alternative MTAs (Postfix, OpenSMTPD) in more depth than this handbook attempts.
- The Dovecot wiki (`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

The original VPS handbook devoted a substantial chapter to FTP — its server software, client programs, anonymous access, and logon banners. FreeBSD 15 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 FreeBSD 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 part of the FreeBSD base system, so this works without installing anything on the VPS; your own computer needs `rsync` too, which is preinstalled on macOS and Linux and available via `pkg install rsync` if you're managing files from another FreeBSD machine.

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.

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

- `# pw useradd clientuser -m -s /usr/sbin/nologin # passwd clientuser`

2. In `/etc/ssh/sshd_config`, add a `Match` block for this user (or a group containing several such users):

- ```
  Match User clientuser ChrootDirectory
  /usr/local/www/sites/clientsite ForceCommand internal-sftp
  AllowTcpForwarding no X11Forwarding no
  ```

3. For `ChrootDirectory` to work, the directory itself (`/usr/local/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. `/usr/local/www/sites/clientsite/htdocs`, which you then point Apache's document root at for that virtual host (Chapter 6).

4. Restart `sshd` to apply:

- `# service sshd restart`

`clientuser` can now connect with any SFTP client and will see only `/` (which is actually `/usr/local/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/auth.log` on the server 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** and **proftpd** are both available via `pkg install`, and either can be configured for password-protected accounts, anonymous upload/download areas, and custom login banners — much as the original VPS v2's FTP chapter described. Because of the cleartext-credential concern above, GSP recommends:

- Limiting FTP accounts to a chrooted directory (both `vsftpd` and `proftpd` support this natively, similar to the SFTP chroot above)
  ```

- 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

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 FreeBSD 15 VPS instances, covers publishing your first site, hosting multiple domains on one VPS, and securing it with HTTPS.

Installing Apache

```
# pkg install apache24
```

Enable and start it:

```
# sysrc apache24_enable=YES
# service apache24 start
```

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

Directory Layout

Apache's package installs everything under `/usr/local`, keeping it cleanly separated from the FreeBSD base system:

Path	Purpose
`/usr/local/etc/apache24/httpd.conf`	Main configuration file
`/usr/local/etc/apache24/Includes/`	Drop-in `.conf` files, automatically loaded — the recommended place for your own configuration
`/usr/local/etc/apache24/extra/`	Optional configuration snippets shipped with Apache (virtual hosts, SSL, language settings), referenced from `httpd.conf` via commented-out `Include` lines
`/usr/local/www/apache24/data/`	Default document root — files here are served at `https://example.com/`
`/var/log/httpd-*.log`	Access and error logs

> **Note:** Don't edit `httpd.conf` itself for site-specific configuration if you can avoid it — future `pkg upgrade apache24` runs may offer to replace it (saving your old version as `httpd.conf.pkgsave`), and merging your changes back in is extra work. Anything you place in `/usr/local/etc/apache24/Includes/*.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 `www` user) can read them:

```
# chown -R youruser:www /usr/local/www/apache24/data
# find /usr/local/www/apache24/data -type d -exec chmod 755 {} \;
# find /usr/local/www/apache24/data -type f -exec chmod 644 {} \;
```

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

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 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 `/usr/local/etc/apache24/Includes/`, for example `/usr/local/etc/apache24/Includes/example.com.conf`:

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

    <Directory /usr/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 `Includes/` automatically — no central list to maintain.

Reload Apache after adding or changing a virtual host:

```
# apachectl configtest && service apache24 reload
```

`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.

> **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 `httpd.conf` (or, better, a file in `Includes/`) 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 server-wide, set it in the top-level `<Directory "/usr/local/www/apache24/data">` block in `httpd.conf`.

Finally, make sure Apache itself stays current:

```
# pkg upgrade apache24
```

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

HTTPS with Let's Encrypt and acme.sh

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 **acme.sh** — a dependency-light shell script — is the tool GSP recommends for obtaining and renewing them on FreeBSD.

```
# pkg install acme.sh
```

The package creates a dedicated `acme` user and a home directory at `/var/db/acme`. The simplest way to issue a certificate for an Apache site is the **webroot** method, which works by placing a temporary file in your site's document root that Let's Encrypt's servers fetch to confirm you control the domain:

Recent versions of acme.sh default to the ZeroSSL certificate authority, which requires registering an email address first. To use Let's Encrypt as this section describes, set it as the default certificate authority once (you only need to do this a single time, before your first `--issue`):

```
# acme.sh --set-default-ca --server letsencrypt
# acme.sh --issue -d example.com -d www.example.com \
    -w /usr/home/youruser/public_html
```

On success, acme.sh stores the certificate and key under `/var/db/acme/example.com/`. Install (copy) them to a stable location Apache will read from, and set up automatic renewal copies with `--install-cert`:

```
# acme.sh --install-cert -d example.com \
    --cert-file /usr/local/etc/apache24/certs/example.com/cert.pem \
    --key-file /usr/local/etc/apache24/certs/example.com/key.pem \
    --fullchain-file /usr/local/etc/apache24/certs/example.com/fullchain.pem
\
    --reloadcmd "service apache24 reload"
```

Then add an HTTPS virtual host alongside the HTTP one (Apache's `mod_ssl` module, enabled by default in the `apache24` package, provides `<VirtualHost *:443>` and the `SSL*` directives):

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

    SSLEngine on
    SSLCertificateFile
/usr/local/etc/apache24/certs/example.com/cert.pem
    SSLCertificateKeyFile
/usr/local/etc/apache24/certs/example.com/key.pem
    SSLCertificateChainFile
/usr/local/etc/apache24/certs/example.com/fullchain.pem
</VirtualHost>
```

acme.sh installs its own cron entry to renew certificates automatically (Let's Encrypt certificates are valid for 90 days, and acme.sh renews them well before expiry); the `--reloadcmd` above ensures Apache picks up a renewed certificate without manual intervention.

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

Important Commands, Directories, and Files

Item	Purpose
`pkg install apache24`	Install Apache
`/usr/local/etc/apache24/httpd.conf`	Main configuration
`/usr/local/etc/apache24/Includes/*.conf`	Your site/virtual host configuration (recommended location)
`/usr/local/www/apache24/data/`	Default document root
`apachectl configtest`	Check configuration syntax before reloading
`service apache24 reload`	Apply configuration changes without dropping connections
`service apache24 restart`	Fully restart Apache
`/var/log/httpd-*.log`	Access and error logs
`acme.sh --issue` / `--install-cert`	Obtain and install 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 `Includes/*.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 (Chapter 8)

Redirects and Rewrites

`mod_rewrite`, enabled by default in the `apache24` 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 FreeBSD `apache24` package builds almost all of them as **DSOs** — Dynamic Shared Objects, the `mod_*.so` files under `/usr/local/libexec/apache24/`. "Dynamically-loaded" means they are *not* compiled into the Apache binary; instead, each is loaded at startup only if `httpd.conf` contains a matching `LoadModule` line. This is what lets you add or drop a capability by editing one line and reloading, with no recompiling — and it is why turning a feature on so often comes down to "uncomment a `LoadModule` line." The loader itself, `mod_so`, is the one module compiled in statically, since it is what makes loading the others possible.

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; the files Apache ships are already listed in `httpd.conf`, most of the commonly-disabled ones commented out with a leading `#`:

```
LoadModule rewrite_module libexec/apache24/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_so`	Always (static)	Loads all the other dynamic modules; the foundation

mod_mpm_event	Yes	The processing engine (multi-processing module) that handles connections
mod_ssl	Yes	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	Often off	Set Cache-Control/Expires so browsers cache static assets
mod_proxy + mod_proxy_fcgi	Often off	Forward requests to another process — how PHP-FPM is served (Chapter 8)
mod_proxy_http	Often off	Reverse-proxy to an app server (Node, Python, etc.) behind Apache
mod_auth_basic + mod_authn_file	Yes	HTTP Basic password protection (below)
mod_alias	Yes	Redirect, Alias, and ScriptAlias directives
mod_status	Often off	A /server-status page of live worker/traffic stats (restrict it!)
mod_info	Off	A /server-info dump of the running configuration (restrict it!)
mod_security (3rd-party)	pkg install	A web-application firewall, for sites that need request filtering

To **enable** a module that is installed but not loaded, find (or add) its `LoadModule` line in `httpd.conf`, remove the leading `#`, and reload:

```
# nano /usr/local/etc/apache24/httpd.conf      # uncomment the LoadModule
line
# apachectl configtest && service apache24 reload
```

`apachectl configtest` (run it *every* time before reloading) catches the most common module mistake: enabling a module whose dependency is still commented out — 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 `httpd.conf` 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 `/usr/local/etc/apache24/extra/httpd-mime.types` (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 `Includes/*.conf` file. Most modern file types (`.json`, `.svg`, `.woff2`, etc.) are already present in current Apache 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.

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

```
# htpasswd -c /usr/local/etc/apache24/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 /usr/home/youruser/public_html/staging>
    AuthType Basic
```

```
    AuthName "Staging Area"
    AuthUserFile /usr/local/etc/apache24/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 /usr/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 `apache24` package's default `mod_ssl` configuration is reasonable for most sites, but two settings are worth checking in `/usr/local/etc/apache24/extra/httpd-ssl.conf`:

```
SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite HIGH:!aNULL:!MD5:!3DES
SSLHonorCipherOrder on
```

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

> **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 `httpd-ssl.conf`.

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 `httpd-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, 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, and the security habits that matter once your VPS is running someone else's code as well as your own.

Installing PHP

FreeBSD's PHP packages are versioned (for example, `php84` for PHP 8.4); check `pkg search php8` for the versions currently available, and pick the one your application requires (most current applications support the latest two or three PHP releases). Install PHP itself, the FastCGI process manager, and the extensions most applications need:

```
# pkg install php84 php84-fpm php84-mysqli php84-pdo php84-pdo_sqlite php84-
pdo_pgsql php84-gd php84-mbstring php84-curl php84-zip php84-opcache
```

> **Note:** Recent PHP releases no longer provide an Apache module (`mod_php`) in the FreeBSD ports tree — PHP 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.

Enable and start php-fpm:

```
# sysrc php_fpm_enable=YES
# service php-fpm start
```

By default, php-fpm listens on a TCP socket at `127.0.0.1:9000`. The Apache configuration in the next section forwards PHP requests to a Unix socket instead, so edit `/usr/local/etc/php-fpm.d/www.conf` to change the `listen` address and give the socket to the `www` user Apache runs as, then restart php-fpm:

```
listen = /var/run/php-fpm.sock
listen.owner = www
listen.group = www
listen.mode = 0660

# service php-fpm restart
```

Connecting Apache to PHP-FPM

Enable `mod_proxy_fcgi` (and `mod_proxy`, which it depends on) in `httpd.conf` if not already active, then add to your virtual host or a global `Includes/*.conf` file:

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

```
</FilesMatch>

DirectoryIndex index.php index.html
```

Reload Apache, 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. FreeBSD packages three options:

- **SQLite** — a serverless, single-file database, included via the `php84-pdo_sqlite` 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: `pkg install mariadb123-server` (check `pkg search mariadb` for the current default version).
- **PostgreSQL** — a feature-rich database server favored by many newer applications and frameworks: `pkg install postgresql16-server` (again, check `pkg search postgresql` for the current version).

For MariaDB:

```
# pkg install mariadb123-server mariadb123-client
# sysrc mysql_enable=YES
# service mysql-server start
# mysql_secure_installation
```

`mysql_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.

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:

1. 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.
2. Create a configuration file — usually by copying a `config-sample.php`, `.env.example`, or similar — with your database credentials and any site-specific settings.
3. Set file ownership and permissions (below).
4. Run any installer the application provides (often by visiting a URL in a browser, or running a command-line setup script).

File Ownership and the "Execute Bit," Revisited

Chapter 1 introduced file permissions; for applications, two 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 `www` 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 /usr/home/youruser/public_html/uploads>
    <FilesMatch "\.php$">
        Require all denied
    </FilesMatch>
</Directory>
```

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

```
$ chmod 640 config.php
$ chown youruser:www 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 (a form field, a URL parameter, an uploaded filename) 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 queries by concatenating strings**, and to be cautious about plugins or themes from sources that don't have a track record of prompt security fixes.

Limit what a compromised application can do. If one application on your VPS is compromised — through a vulnerability in itself, a plugin, or a weak admin password — the damage should ideally be contained to that application. Running each site's files under a distinct user/group (Chapter 2) and using separate database users with access only to their own database (above) means a compromised application can't read another site's database or overwrite another site's files, even though both run as the same `www` user at the web server level.

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 you're running has such a feature and you don't need it, disabling it removes an entire category of risk.

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 php-fpm's log (`/var/log/php-fpm/`, or as configured in `/usr/local/etc/php-fpm.d/www.conf`)

“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 (location varies), plus confirm the database service is running: `service mysql-server status`
“403 Forbidden”	`/var/log/httpd-error.log` will name the specific permission Apache rejected

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 `/usr/local/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.

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 single most useful server-administration command is the one that shows every open port and the program behind it:

```
# sockstat -4 -6 -l        # list listening (-l) IPv4/IPv6 sockets and their
programs
# sockstat -4 -l | grep :25
```

`sockstat` is FreeBSD's native tool; the output names the user, the command, the PID, and the local address/port. The classic `netstat -an | grep LISTEN` shows the same ports without the program names. If a port you don't recognize is open, find out which package owns the program with `pkg which $(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 (`ntpd`/`openntpd`)
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	MySQL/MariaDB	Database — bind to localhost only, never expose
5432	TCP	PostgreSQL	Database — bind to localhost only, never expose

6379	TCP	Redis	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 `sockstat -l` that anything sensitive shows `127.0.0.1:PORT` and not `*:PORT`.

Starting, stopping, and enabling services. FreeBSD manages long-running services (daemons) with the `service` command and the `/etc/rc.conf` enable flags (set most easily with `sysrc`):

```
# service apache24 status        # is it running?
# service apache24 start         # start it now
# service apache24 restart       # stop then start
# service apache24 reload        # re-read config without dropping
connections
# sysrc apache24_enable=YES      # start it automatically at every boot
# service -e                     # list every service enabled to start at
boot
```

The distinction trips people up: `service ... start` runs a service *right now* but does not survive a reboot; the matching `sysrc name_enable=YES` is what makes it come back after the VPS restarts. You almost always want both.

Logs

Nearly every service this handbook has covered writes a log file under `/var/log/`:

Log file	Written by
`/var/log/maillog`	Sendmail and Dovecot (Chapter 4)
`/var/log/httpd-access.log`, `/var/log/httpd-error.log` (or per-site equivalents)	Apache (Chapters 6–8)

/var/log/auth.log	SSH and su/doas authentication attempts
/var/log/messages	General system messages

newsyslog: Automatic Log Rotation

Left alone, these files would grow forever. FreeBSD's newsyslog runs periodically (driven by an entry in /etc/crontab) and, based on rules in /etc/newsyslog.conf, renames the current log to .0, compresses older rotations, deletes ones past a configured limit, and tells the relevant service to start writing to a fresh file.

A newsyslog.conf entry looks like:

```
# logfile                          owner:group  mode  count  size  when
flags
/var/log/httpd-example.com-error.log  www:www      644    7     *    @T00
JC
```

This rotates the file daily at midnight (@T00), keeps 7 rotations, compresses old ones (J for bzip2/xz-style compression — check newsyslog.conf(5) for the exact flag your version uses), and creates a fresh empty log file after rotation if one doesn't already exist (C). Telling the service itself to reopen its log after rotation is done with the optional pid-file and signal columns (not shown here) — one reason the package-provided entries are easier than writing your own by hand. The package for any service you install (Apache, Sendmail, Dovecot) typically adds its own appropriate entries automatically; cat /etc/newsyslog.conf to see what's already configured, and add entries for any custom log paths (such as a virtual host's per-site logs from Chapter 6) that aren't covered yet.

Scheduling Tasks: cron and periodic

cron runs commands on a schedule. System-wide jobs live in /etc/crontab; an individual user's own jobs are managed with crontab -e, which opens their personal crontab in your default editor (Chapter 1 covers ee, nano, and vi).

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 * * *   /usr/home/youruser/bin/backup.sh

# Check a site is responding every 5 minutes
*/5 * * * *  /usr/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 — which is 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.

periodic is FreeBSD's framework for routine system housekeeping — tasks like rotating logs (which calls `newsyslog`), checking for security updates, and cleaning temporary files. It runs daily, weekly, and monthly via entries in `/etc/crontab`, and its output is emailed to **root** by default. Two settings make this useful in practice:

1. Make sure root's mail reaches you — add to `/usr/local/etc/mail/aliases` (Chapter 4):

- `root:  youruser`

  then `newaliases`.

2. Review (or trim) what periodic actually checks via `/etc/periodic.conf` — for example, `daily_status_security_enable="YES"` (the default) includes a daily summary of login attempts and changes to setuid programs, which is worth reading even if briefly.

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  up 12 days,  3:14, 1 user, load averages: 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. When you first run it, the screen begins with a five-line summary followed by the process table header:

```
$ top

last pid: 79562;  load averages:  0.36,  0.29,  0.31    up 21+14:08:41
13:34:35
55 processes:  1 running, 54 sleeping
CPU:  0.1% user,  0.0% nice,  0.1% system,  0.0% interrupt, 99.8% idle
Mem: 19M Active, 15G Inact, 3660K Laundry, 2073M Wired, 707M Buf, 6524M Free
Swap: 1024M Total, 25M Used, 999M Free, 2% Inuse

  PID USERNAME    THR PRI NICE   SIZE    RES STATE    C   TIME    WCPU
COMMAND
```

Every term in that header means something specific:

Line 1 — overall activity. `last pid` is the process ID assigned to the most recently created process (a rough sense of how much process churn there is). `load averages` are the same 1/5/15-minute figures as `uptime`. `up 21+14:08:41` is the **uptime** — 21 days, 14 hours, 8 minutes since the last boot — and the final `13:34:35` is the current clock time.

Line 2 — process counts. The total number of processes, broken down by **state**. The states you'll see are `running` (actually 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`, `zombie` (finished but not yet reaped by its parent), or `waiting`.

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

Field	Meaning
`user`	Running ordinary user programs (your web server, PHP, scripts)
`nice`	Running user programs whose priority was lowered with `nice`
`system`	Running kernel code on programs' behalf (disk, network, system calls)
`interrupt`	Servicing hardware interrupts (network cards, timers)
`idle`	Doing nothing — high idle is *good*; it means spare capacity

Line 4 — physical memory (RAM). FreeBSD reports memory by role, and the categories surprise people coming from other systems:

Field	Meaning
`Active`	Recently-used pages belonging to running programs
`Inact`	Inactive pages — not used lately but still holding data, reclaimable on demand
`Laundry`	Dirty inactive pages waiting to be written to disk before they can be reused
`Wired`	Pinned in RAM and never paged out (the kernel itself, ZFS's ARC cache)
`Buf`	The filesystem buffer cache

`Free`	Completely unused memory

A common worry is "almost nothing is Free!" — but on a healthy FreeBSD system that is *normal and good*: the OS keeps recently-used data in `Inact`/`Wired` rather than wasting RAM by leaving it `Free`, and hands it back the instant a program needs it. The number to watch is not `Free` but **swap activity** (next line).

Line 5 — swap (disk used as overflow memory). `Total`, `Used`, and `Free` swap space, plus `Inuse` as a percentage. A little swap in use is fine. *Steadily growing* swap, or constant swap activity, is the real sign of memory pressure — that 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`
`USERNAME`	The account the process runs as (`www` for Apache workers, `mysql`, etc.)
`THR`	Number of threads in the process
`PRI`	Kernel scheduling priority (lower runs sooner)
`NICE`	The "niceness" you can set: −20 (greedy) to +20 (yields to others)
`SIZE`	Total virtual memory the process has mapped
`RES`	**Resident** memory — the part actually in RAM right now (usually the number you care about)
`STATE`	What it's doing: `RUN`, `sleep`, `wait`, a lock name, or a CPU number
`C`	Which CPU core it last ran on
`TIME`	Total CPU time the process has accumulated
`WCPU`	Weighted recent CPU usage — the percentage that sorts the list
`COMMAND`	The program's name

A few interactive keystrokes make `top` far more useful while it's running: press `o` then `res` to sort by resident memory instead of CPU (to find the memory hog), `P` 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 www                     # every process running as the Apache (www)
user
$ ps -p 79562 -o pid,rss,command   # 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 /usr/home/youruser          # total size of one directory
$ du -sh /var/log/*                  # size of each item under /var/log
$ du -h -d 1 /usr/local | 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 instead of every subdirectory), -h prints human-readable units (K/M/G), 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 `sr` "scan rate" column, or in `swap-in`/`swap-out`) 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 `doas kill` for a process owned by another account (such as `www`).

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 /usr/home/youruser/backups/site-$(date +%Y%m%d).tar.gz \
    -C /usr/home/youruser public_html

# Copy it somewhere off the VPS
$ rsync -avz /usr/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 > /usr/home/youruser/backups/myapp-$(date
+%Y%m%d).sql

# PostgreSQL
$ pg_dump myapp > /usr/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.

ZFS Snapshots

If your VPS uses ZFS (the default for new GSP VPS images), ZFS **snapshots** provide an extremely cheap, near-instantaneous point-in-time copy of an entire dataset — useful as a safety net immediately before a risky change (a major upgrade, a database migration):

```
# zfs snapshot zroot/usr/home@before-upgrade
```

A snapshot costs almost nothing to create and initially takes no extra space — it only consumes space as the live data diverges from the snapshot. List snapshots with `zfs list -t snapshot`, and roll back to one (discarding everything since) with `zfs rollback`. Because a rollback discards changes, and because snapshots live on the same physical disks as the data they're snapshots of, **ZFS snapshots are not a substitute for off-VPS backups** — they protect against "I broke something five minutes ago," not against disk failure or accidental `zfs destroy`.

Restoring Files

Restoring from a `tar` archive:

```
$ tar xzf site-20260601.tar.gz -C /usr/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. Two options:

```
# pkg install goaccess
$ goaccess /var/log/httpd-example.com-access.log -o
/usr/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 processes interactively), or be scheduled via cron to regenerate the HTML report nightly.

```
# pkg install awstats
```

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

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

```
# pkg install analog
```

Analog is configured through a single file, `/usr/local/etc/analog.cfg`, where you set the log to read, the report's output file, and which breakdowns (pages, referrers, browsers, search terms, status codes) to include:

```
# /usr/local/etc/analog.cfg
LOGFILE     /var/log/httpd-example.com-access.log
OUTFILE     /usr/home/youruser/public_html/stats/analog.html
HOSTNAME    "example.com"
LOGFORMAT   COMBINED
```

Then generate the HTML report — typically from cron (Chapter 9) so it refreshes on a schedule — by running `analog` with no arguments (it reads the config file automatically):

```
$ analog                        # writes the OUTFILE named in analog.cfg
```

Analog produces a single self-contained HTML page of tables and small 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) — almost every error below writes an explanatory line somewhere under `/var/log`.

"A service won't start." Run it in check mode and read the exact complaint before anything else:

```
# service apache24 restart        # the failure message often names the
line/file
# apachectl configtest            # Apache: pinpoints the bad directive
# sendmail -bv root               # Sendmail: trace how an address resolves
# tail -n 40 /var/log/messages    # the catch-all system log
```

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 `www` user, so every directory in the path *and* the file itself must be readable by `www`, and directories must be executable (searchable). Check from the top down and fix ownership/mode rather than resorting to `chmod 777` (which is a security problem, not a fix):

```
$ ls -ld /usr/home/youruser /usr/home/youruser/public_html   # each must be
at least 755
$ namei -l /usr/home/youruser/public_html/index.html         # show perms at
every level
# chown -R youruser:www /usr/home/youruser/public_html
```

"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, which names the exact reason.

"500 Internal Server Error" is almost always the *application's* fault, not Apache's — a PHP fatal error, a bad `.htaccess` line, or a script without its execute bit. The real message is in the error log:

```
$ tail -f /var/log/httpd-example.com-error.log     # then reload the page to
see the error appear
```

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

```
# sockstat -4 -l | 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 `service ... status` and `sockstat -l`. *Timed out* means traffic isn't getting through at all — almost always the **firewall** (Chapter 10): confirm the port is allowed in `/etc/pf.conf`, and check you haven't been caught by `blacklistd`/`fail2ban` (below) 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`. On a ZFS VPS, also check whether old snapshots are holding space: `zfs list -t snapshot`.

"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:

1. **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 being down; check GSP's status page or contact support.
2. **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.
3. **Is the relevant service running?** `service apache24 status`, `service sendmail status`, `service dovecot status`, etc. — or check Webmin's Bootup and Shutdown module (Chapter 3).
4. **What does the log say?** Almost every chapter in this handbook names the relevant log file — check it for errors with a timestamp matching when the problem started.
5. **Is the disk full?** `df -h` — a full disk causes failures in services that seem unrelated to storage.
6. **Did something change recently?** A configuration edit, a package upgrade, an application update — `pkg upgrade -n` shows what would be upgraded (and, by checking `/var/log/messages` or shell history, what already was).

Important Commands and Files

Item	Purpose
`/etc/newsyslog.conf`	Log rotation rules
`crontab -e`	Edit your personal scheduled jobs
`/etc/crontab`	System-wide scheduled jobs (including periodic)
`/etc/periodic.conf`	Configure daily/weekly/monthly housekeeping
`top`, `ps aux`, `uptime`, `vmstat`	Inspect current resource usage
`df -h`	Disk space per filesystem

`tar`, `rsync`, `mysqldump/pg_dump`	File and database backups
`zfs snapshot`, `zfs rollback`, `zfs list -t snapshot`	ZFS point-in-time copies (ZFS systems only)

Where to Go Next

Chapter 10 covers the security side of ongoing maintenance — keeping FreeBSD and everything you've installed patched, and configuring the firewall.

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 `pf` firewall, SSH hardening, and a checklist for the services covered in earlier chapters.

Staying Patched

Two separate update mechanisms cover a FreeBSD VPS: one for the base operating system, and one for everything you've installed with `pkg`.

freebsd-update

`freebsd-update` applies security and errata patches to the FreeBSD base system — the kernel and the userland tools that came with your installation — without requiring a full reinstall or rebuild from source.

```
# freebsd-update fetch
# freebsd-update install
```

`fetch` downloads any available updates and shows what would change; `install` applies them. Some updates (kernel patches) require a reboot to take effect, which `freebsd-update` will tell you. To automate checking (leaving the decision to install to you, or automating that too once you're comfortable), add a weekly cron entry:

```
# Check for base system updates every Sunday at 3 AM
0 3 * * 0   /usr/sbin/freebsd-update cron
```

`freebsd-update cron` only fetches and notifies (by mail, to root — see Chapter 9 for making sure that reaches you); it doesn't install anything automatically, which is the right default for a server you don't want rebooting unattended.

pkg upgrade and pkg audit

Everything installed via `pkg` — Apache, Sendmail, Dovecot, Webmin, Roundcube, PHP, and so on — is updated independently of the base system:

```
# pkg update          # refresh the list of available packages
# pkg upgrade          # install available updates
```

`pkg audit` cross-references your installed packages against the FreeBSD VuXML vulnerability database and flags anything with a known security issue:

```
# pkg audit -F
```

Run this regularly (a weekly cron entry alongside `freebsd-update cron` is reasonable), 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.

> **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 pf Firewall

`pf` ("packet filter") is FreeBSD's firewall, included in the base system but not configured or enabled by default. 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 yet.

A Starting Ruleset

Create `/etc/pf.conf`:

```
# Interface carrying your public IP — check with `ifconfig`
ext_if = "vtnet0"

# Default: block everything...
set block-policy drop
scrub in on $ext_if all

block in on $ext_if all
pass out on $ext_if all keep state

# ...then allow specific incoming traffic
pass in on $ext_if proto tcp to ($ext_if) port 22  keep state   # SSH/SFTP
pass in on $ext_if proto tcp to ($ext_if) port { 80, 443 } keep state  #
HTTP/HTTPS
pass in on $ext_if proto tcp to ($ext_if) port { 25, 587, 993, 995 } keep
state  # mail
pass in on $ext_if proto icmp all keep state                    # ping
```

> **Note:** `vtnet0` is the network interface name on many virtualized FreeBSD instances (the VirtIO network driver), but confirm yours with `ifconfig` — look for the interface with your public IP address assigned. Using the wrong interface name means the ruleset won't apply to the traffic you intended.

This ruleset blocks everything inbound by default, then explicitly allows SSH/SFTP (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:

```
pass in on $ext_if proto tcp to ($ext_if) port 10000 keep state
```

or, better, restrict it to specific source addresses:

```
pass in on $ext_if proto tcp from { 198.51.100.7 } to ($ext_if) port 10000
keep state
```

Enabling pf

```
# sysrc pf_enable=YES
# service pf start
```

> **Warning:** Test firewall changes carefully — a typo that blocks port 22 locks you out of SSH, and from there, out of fixing the firewall remotely. Before making changes, check whether GSP's control panel for your VPS provides a "console" or "VNC" access 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 and have no console access, GSP support can help recover access — see Appendix C.

To check the active ruleset and see traffic being blocked or passed:

```
# pfctl -sr               # show the active rules
# tcpdump -n -i $ext_if   # watch raw traffic, if you suspect something's
being
                            # blocked that shouldn't be
```

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 — a passphrase for the key itself, which is checked locally on your computer and never sent to the server):

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

Disable Password Authentication

Once key-based login is confirmed working — and **only then**, since this step removes the fallback — edit `/etc/ssh/sshd_config`:

```
PasswordAuthentication no
# service sshd restart
```

> **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 `sshd_config` and `service sshd restart` again without being locked out entirely.

blacklistd: Blocking Repeat Offenders

FreeBSD's base system includes `blacklistd`, which watches for repeated failed authentication attempts (SSH, and other services that support it) and tells `pf` to temporarily block the offending address — similar in effect to the popular third-party tools `fail2ban` and `sshguard`, but built in.

```
# sysrc blacklistd_enable=YES
# service blacklistd start
```

Add to `/etc/pf.conf` (before the `pass` rules):

```
table <blacklistd> persist
block quick from <blacklistd>
```

and to `sshd_config`:

```
UseBlacklist yes
```

After a configurable number of failed login attempts from one address (default thresholds are reasonable for most VPS instances), that address is added to the `<blacklistd>` table and silently dropped by `pf` for a period of time. View current entries with `blacklistctl dump -a`.

fail2ban: A Configurable Alternative

`blacklistd` is built in and excellent for SSH, but it only protects services that have been written to talk to it. **fail2ban** takes a different, more general approach: it *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 many administrators run it instead of, or alongside, `blacklistd`.

Install it from packages:

```
# pkg install py311-fail2ban
# sysrc fail2ban_enable=YES
```

fail2ban is configured under `/usr/local/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 `/usr/local/etc/fail2ban/jail.local`:

```
# /usr/local/etc/fail2ban/jail.local

[DEFAULT]
# How fail2ban enforces blocks on FreeBSD — use the pf action
banaction = pf
# 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
```

The `pf` ban action expects a table to put offenders in. Add one to `/etc/pf.conf` (near the `blacklistd` table from the previous section), and block from it early in the ruleset:

```
table <fail2ban> persist
block quick from <fail2ban>
```

Reload pf (`service pf reload`) and start fail2ban:

```
# service fail2ban start
```

Manage and inspect it with the `fail2ban-client` tool:

```
# 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:** Run *either* `blacklistd` *or* fail2ban for a given service, not both pointed at the same thing — two systems banning the same address is harmless but confusing to debug. A common split is to let `blacklistd` handle SSH (it is built in and efficient) and add fail2ban only for the application-level services it can't see, such as Roundcube logins or Sendmail AUTH failures. Whichever you choose, **always** put your own IP in the ignore/whitelist so a forgotten password can't lock you out of your own VPS.

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, `blacklistd`	This chapter, Chapter 1
Webmin	Restrict port 10000 via `pf` or SSH tunnel; keep updated	Chapter 3
Sendmail	`access` table reviewed for unintended `RELAY` entries	Chapter 4
Roundcube/Dovecot	Always accessed over HTTPS/TLS	Chapter 4
Apache	`ServerTokens Prod`, `ServerSignature Off`, directory listing disabled, HTTPS + HSTS	Chapters 6–7
Web applications	Updated promptly; uploads directory can't execute scripts; secrets not world-readable	Chapter 8
Backups	Off-VPS copies exist and have been test-restored	Chapter 9

Staying Informed

- **FreeBSD Security Advisories** are published at `freebsd.org/security/` and announced on the `freebsd-security-notifications` mailing list — subscribe to be notified of issues affecting the base system.
- `pkg audit -F`, run regularly (above), covers everything installed via packages.
- **The FreeBSD Handbook's Security chapter** (`docs.freebsd.org`) goes into more depth on topics like jails and `securelevel` for VPS owners who want to isolate individual services from each other even further than the user/group separation in Chapter 8.

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 — pkg and the Ports Collection

Every chapter in this handbook has used `pkg install` to add software. This appendix covers pkg more completely, and introduces the **Ports Collection** for the rare cases where a package isn't enough.

pkg Basics

Command	Effect
`pkg install <name>`	Install a package and its dependencies
`pkg search <text>`	Search the package repository by name or description
`pkg info`	List installed packages
`pkg info <name>`	Show details (version, dependencies, description) for one package
`pkg info -l <name>`	List every file a package installed
`pkg which <path>`	Find which installed package owns a given file
`pkg upgrade`	Upgrade all installed packages
`pkg delete <name>`	Remove a package
`pkg autoremove`	Remove packages that were pulled in only as dependencies and are no longer needed
`pkg version -v`	Compare installed versions against the repository
`pkg clean`	Remove cached package files no longer needed
`pkg audit -F`	Check installed packages against known vulnerabilities (Chapter 10)

`pkg search` is often the fastest way to find the exact package name for something — for example, `pkg search roundcube` confirms the package is called `roundcube` (rather than, say, `mail/roundcube1.6`) before you `pkg install` it.

A typical first encounter with a new piece of software follows the same short arc — search for it, look before you leap, install, confirm:

```
$ pkg search nginx                  # find the exact package name
$ pkg info -d nginx                 # what would it pull in as dependencies?
# pkg install nginx                 # install it (and its dependencies)
```

```
$ pkg info nginx                    # confirm the installed version
$ pkg info -l 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):

```
# pkg update                        # refresh the catalog of what's available
# pkg upgrade                       # install newer versions of what you have
# pkg autoremove                    # drop orphaned dependencies nothing
needs anymore
# pkg clean -a                      # reclaim disk space from the download
cache
```

> **Tip:** `pkg` keeps a log of everything it does in `/var/log/pkg.log`, and `pkg query '%n-%v %t' | sort` can show what you installed and when — useful for working out "what changed?" after an upgrade. If an upgrade ever leaves a package's config behind as `*.pkgsave` (because you'd edited the original), `pkg` tells you so; merge your changes into the new file at your convenience.

The Ports Collection

The **Ports Collection** is the source from which most `pkg` packages are themselves built — a tree of small "Makefiles" (one per piece of software) that know how to download, configure, compile, and install it, along with any dependencies. Almost everyone should use `pkg` rather than building from ports directly: it's faster (no compilation) and the packages are built and tested centrally. Two situations where ports are still useful:

- A package is built with default options, but you need a non-default build-time option (a particular module or feature compiled in or out)
- You want to track a newer version than the package repository currently provides (the package repository typically lags the absolute latest port slightly)

Getting the Ports Tree

As of FreeBSD 15, the Ports Collection is distributed only via **Git** — the older `portsnap` tool has been retired. To fetch it:

```
# pkg install git
# git clone https://git.FreeBSD.org/ports.git /usr/ports
```

For a tree matching a specific quarterly branch (which receives security patches but not new versions, useful for stability) rather than the actively-developed `main` branch:

```
# git clone --branch 2026Q2 https://git.FreeBSD.org/ports.git /usr/ports
```

Update an existing tree with `git -C /usr/ports pull`.

Building a Port

```
# cd /usr/ports/mail/roundcube
# make config       # choose build-time options, if any (skip for defaults)
# make install clean
```

`make config` opens a checklist of optional features for that port (if it has any); `make install clean` downloads source code, compiles it, installs it, and removes the temporary build files afterward. Building from source can take anywhere from under a minute to well over an hour depending on the software and your VPS's resources — large applications (web browsers, for instance, though unlikely on a server) can be impractical to build on a small VPS.

Finding and Managing Ports

With the tree in place under `/usr/ports`, a few `make` targets cover most of what you'll do. Run them from inside a port's directory (e.g. `/usr/ports/www/nginx`):

Command	Effect
`make search name=nginx`	Search the tree for a port by name (run from `/usr/ports`)
`make config`	Open the build-options checklist for this port
`make showconfig`	Show the options currently selected
`make install clean`	Download, compile, install, then delete the build files
`make deinstall`	Uninstall the port
`make rmconfig`	Forget saved options and start over
`make missing`	List dependencies not yet installed

Saved build options live under `/var/db/ports/`, so they persist across rebuilds — set them once with `make config` and later upgrades remember your choices.

Because building from source by hand gets unwieldy once several ports depend on each other, two helper tools are worth knowing. **`portmaster`** (`pkg install portmaster`) rebuilds a port and everything that depends on it in the right order with one command. **`poudriere`** (`pkg install poudriere`) builds your ports in clean, isolated jails and produces your *own* package repository — overkill for a single VPS, but the right tool if you maintain several servers that should all install the same custom-built packages.

> **Note:** Mixing source-built ports and binary `pkg` packages on the same system can lead to version mismatches, since the two are updated on different schedules. The simplest, most reliable policy for a typical VPS is to use `pkg` for everything and reach for ports only for the specific package that needs a non-default option —

> then let `pkg upgrade` manage the rest. If you build a lot from ports, consider `poudriere` so that even your custom builds are installed *as* packages and stay consistent.

A Note on pkgbase

FreeBSD 15 includes, as a technology preview, **pkgbase** — packaging the base system itself (the kernel, `/bin`, `/sbin`, and so on) using `pkg`, the same tool used for everything in this appendix. The goal is for a future FreeBSD release to manage the entire system — base and packages — through one tool and one update mechanism. For FreeBSD 15, the traditional combination of `freebsd-update` for the base system and `pkg` for packages (Chapter 10) remains the primary, supported approach for most installations, including GSP VPS images using the standard "Distribution Sets" installation method. If you're curious, FreeBSD's release notes describe how to opt into pkgbase — 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 (typical for a 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

Users

Command	Effect

`adduser`	Interactive new account creation
`rmuser <user>`	Interactive, safe account removal
`pw useradd/usermod/userdel`	Scriptable account management
`pw lock/unlock <user>`	Disable/re-enable an account without deleting it
`passwd [user]`	Change a password
`chpass [user]`	Edit account details
`su`	Become root (full session)
`doas <command>`	Run one command as root

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
`ifconfig`	Show network interfaces and addresses
`dig <domain>`	Query DNS records
`ping <host>`	Test basic connectivity

Packages and Services

Command	Effect

`pkg install/search/info/upgrade/delete <name>`	Package management (Appendix A)
`pkg audit -F`	Check for known vulnerabilities
`service <name> start/stop/restart/status`	Control a service
`sysrc <name>_enable=YES`	Enable a service at boot
`apachectl configtest`	Check Apache configuration before reloading

Mail

Command	Effect
`newaliases`	Rebuild the mail aliases database after editing `/etc/mail/aliases`
`cd /usr/local/etc/mail && make install restart`	Rebuild Sendmail tables and restart
`tail -f /var/log/maillog`	Watch mail activity

Appendix C: Getting Help

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

FreeBSD Documentation

- **The FreeBSD Handbook** (`docs.freebsd.org/en/books/handbook/`) is the authoritative, comprehensive guide to FreeBSD itself — installation, the base system, networking, and more, kept up to date with each release.
- **Manual pages**, accessed with `man <command>` (e.g., `man pkg`, `man pf.conf`, `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.
- **FreeBSD manual pages online.** The very 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. The online set covers the FreeBSD base system and many add-on packages. On the VPS itself, `man <command>` remains the authoritative copy, since it always matches the exact version you have installed.
- **The FreeBSD Forums** (`forums.freebsd.org`) are an active community where questions about FreeBSD itself — as opposed to the specific applications covered in Chapters 4–8 — are welcomed and generally answered quickly.

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**: the FreeBSD Handbook's Electronic Mail chapter, and `sendmail.org`
- **Dovecot**: `doc.dovecot.org`
- **Roundcube**: `roundcube.net/about/docs`
- **PHP**: `php.net/docs.php`

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. FreeBSD does not install Perl in the base system, so add it from packages:

```
# pkg install perl5
$ 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/local/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 FreeBSD VPS, and choosing the right one avoids a lot of grief.

1. Prefer the FreeBSD package when one exists. Most popular CPAN modules are packaged with a `p5-` prefix, and installing them this way means `pkg upgrade` keeps them current and there is nothing to compile:

```
$ pkg search p5-JSON
# pkg install p5-JSON p5-LWP-Protocol-https p5-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:

```
# pkg install p5-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) 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]> upgrade                         # update all installed CPAN
modules
cpan[4]> quit
```

Useful housekeeping commands, whichever client you use: `perldoc Some::Module` reads a module's documentation, `perl -MSome::Module -e 'print $Some::Module::VERSION'` checks whether (and which version) it is installed, and `cpan-outdated` (from `pkg install p5-App-cpanoutdated`) lists modules with newer versions available so you can pipe them to `cpanm` for a bulk update.

> **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 files. 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 `pkg`, after which the module builds cleanly.

Compilers: clang, gcc, and Building Software

FreeBSD ships with a complete C/C++ toolchain in the **base system** — there is nothing to install to compile most software. The default compiler is **Clang/LLVM**, reached through the traditional names `cc` and `c++` (which *are* Clang on FreeBSD):

```
$ cc --version                  # this is Clang on FreeBSD
$ cat > hello.c <<'EOF'
#include <stdio.h>
int main(void){ printf("Hello from FreeBSD!\n"); return 0; }
EOF
$ cc -O2 -Wall -o hello hello.c
$ ./hello
Hello from FreeBSD!
```

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

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

> **Note:** FreeBSD's `make` is **BSD make**, which differs from the GNU `make` many Linux projects expect. If a project's `Makefile` fails with unfamiliar syntax errors, it probably wants GNU make: install it with `pkg install gmake` and run `gmake` instead of `make`.

Installing GCC. Some software specifically requires the GNU Compiler Collection (or you may simply prefer it). Install a versioned package and call it by its versioned name, so it coexists cleanly with the base Clang:

```
# pkg install gcc                # installs the current default GCC (e.g.
gcc14)
$ gcc14 --version
$ gcc14 -O2 -Wall -o hello hello.c
$ g++14 -std=c++20 -o app app.cpp
```

To make a build use GCC instead of the default Clang, point the standard compiler variables at it:

```
$ make CC=gcc14 CXX=g++14
```

Other toolchains are a single `pkg install` away when you need them — `pkg install llvm` for additional LLVM tools and newer Clang versions, `gcc`'s package also provides `gfortran` for Fortran, and whole modern-language toolchains are packaged too (`pkg install rust` for Rust's `cargo`, `pkg install go` for Go, `pkg install node` for JavaScript). As always, prefer the package over building a compiler from source — bootstrapping a compiler on a small VPS can take hours.