100+ Stealth & Safety Tips for Linux Users

A Cybersecurity Guide to The Universe

Practical, no-nonsense privacy and security habits for everyday Linux users — encryption, hardening, safe defaults, and a healthy skepticism toward security theater. Serious where it counts, dryly amused where it helps.

102 tips · 11 categories · Compiled August 2026 · SplitZrate.com

---

Contents

1. Network & Traffic Privacy 2
2. Disk & Data Encryption 3
3. Secure Communication & Remote Access 4
4. System & Kernel Hardening 5
5. Browser & Application Privacy 7
6. Authentication & Credentials 8
7. Secure Deletion & Digital Hygiene 9
8. Logging, History & Footprint Control 10
9. Physical Security & Opsec Habits 10
10. Anonymity Tooling & Specialized Distros 11
11. Myths, Security Theater & Overkill 12

Appendix: Command Cheat Sheet 13

1. Network & Traffic Privacy

Get the order of operations right here in particular — several of these change what your machine is reachable on, and getting the sequence wrong is the single most common way people accidentally lock themselves out of their own remote server.

1.1 Route traffic through a trustworthy VPN, not just "a VPN" A VPN doesn't remove trust from the equation, it relocates it — from your ISP to the VPN provider. Picking one because of a catchy ad instead of a track record just means trusting a company you know less about than the one you were trying to get away from. Look for an independently audited no-logs policy (not just a claim on the homepage), a jurisdiction with reasonable legal protections, and ideally a provider that's already been tested for real — a server seizure or subpoena that turned up nothing useful, documented publicly.

> **GOOD TO KNOW:** "No logs" is a claim until it's been independently audited or tested in court. Read the audit report, not the marketing page.

1.2 Encrypt your DNS lookups Example: `DNSOverTLS=yes` in `/etc/systemd/resolved.conf`, then `systemctl restart systemd-resolved`. Without this, every domain name you look up travels the network in plain text — trivially readable by your ISP, a coffee shop's router, or anyone else on the same network segment, even when the website itself loads over HTTPS afterward. Confirm it actually took effect with `resolvectl status` and check for "DNSOverTLS: yes" next to your active interface — a saved config file that didn't get picked up is a very easy thing to miss.

1.3 Randomize your Wi-Fi MAC address Example: `nmcli connection modify Wi-Fi wifi.cloned-mac-address random`. Stops passive tracking of your device across networks and venues by its hardware address alone — retail Wi-Fi analytics and some ad networks build surprisingly detailed movement profiles from nothing but MAC addresses seen at different access points over time.

> **GOOD TO KNOW:** Randomization only helps until you log in somewhere identifiable on that connection — at that point the network operator can link the session to you anyway. It closes the passive-tracking gap, not the active one.

1.4 Use WireGuard for tunnels you control A lightweight, modern VPN protocol built into the kernel since 5.6, with a dramatically smaller codebase than older stacks like OpenVPN — WireGuard is roughly 4,000 lines of code versus OpenVPN's much larger surface, which matters a great deal for how realistically either one can be audited. Worth using for your own point-to-point tunnels (home to laptop, server to server), not just as the protocol behind a commercial VPN app.

1.5 Check what's actually listening on your machine Example: `ss -tulpn` lists every open TCP/UDP port and the exact process behind it. Audit it occasionally — a package you installed months ago for one project can leave a service listening on `0.0.0.0` long after you've forgotten it exists, with nothing around to flag it for you. Anything you don't recognize is worth tracing back to a package (`dpkg -S` the binary path) before deciding whether it's supposed to be there.

1.6 Set up a kill switch, not just a VPN client Firewall rules that block all non-VPN traffic the moment the tunnel drops mean a dropped connection fails closed instead of silently falling back to your real IP mid-download. Most dedicated VPN apps offer this as a toggle; if you're running WireGuard by hand, the equivalent is a firewall rule that only permits outbound traffic through the `wg0` interface (plus the VPN handshake itself) and drops everything else.

1.7 Watch for WebRTC leaks in the browser WebRTC can expose your real IP address even behind an active VPN, because it negotiates connections directly between peers using STUN/ICE — a path that doesn't necessarily route through your VPN tunnel at all. Disable it outright, or restrict ICE candidates (Firefox: `media.peerconnection.enabled`), if it isn't something you actively rely on for video calls.

> **GOOD TO KNOW:** This mainly matters for browser traffic. A system-wide VPN with a proper kill switch already closes most of this gap at the OS level.

1.8 Firewall by default-deny, not default-allow Example: `ufw default deny incoming`, then explicitly allow only what you need. A firewall that permits everything except a blocklist isn't really a firewall — it's a list of things you remembered to worry about.

> **CAUTION:** Doing this over a remote SSH session in the wrong order locks you out of the machine instantly, with no way back in except console/KVM access. Always allow your SSH port first, confirm it, and only then set the default-deny policy:

```
sudo ufw allow OpenSSH
sudo ufw default deny incoming
sudo ufw enable
```

1.9 Use Tor for what actually needs Tor Tor is built for anonymity — routing traffic through three independently-operated relays so no single point knows both who you are and what you're accessing. For a lot of everyday privacy needs, though, a well-configured VPN plus a hardened browser gets you further with far less latency; save Tor for the situations that specifically call for anonymity, not general-purpose browsing.

1.10 Choose your DNS resolver deliberately Run your own resolver (`unbound`, with DNSSEC validation enabled) or pick a provider you actually trust, instead of silently using whatever your router or ISP hands out via DHCP by default. Your DNS resolver sees every domain you look up, in order, all day — it's one of the most complete activity logs of your online behavior that exists, and most people never choose it deliberately at all.

1.11 Verify your VPN is doing what you think it's doing A quick `curl ifconfig.me` before and after connecting confirms your IP actually changed — don't just trust the app's green checkmark. Then go further: check for DNS leaks (a site like dnsleaktest.com, or manually confirming your resolver is the VPN's, not your ISP's) and specifically check for an IPv6 leak.

> **CAUTION:** Plenty of VPN clients tunnel IPv4 correctly but leave IPv6 traffic routing outside the tunnel entirely, unencrypted and directly attributable to you — a well-known, still-surprisingly-common gap. If your VPN doesn't explicitly support IPv6, disable IPv6 system-wide instead of trusting it to "just not matter."

---

2. Disk & Data Encryption

2.1 Encrypt the whole disk, not just the home folder Full-disk encryption with LUKS at install time protects swap, temp files, and application logs too — not just the documents folder most people think of first. A "just encrypt home" setup still leaves plenty of sensitive fragments (crash dumps, cache files, swapped-out memory) sitting in the clear on the rest of the disk.

2.2 Know what luksOpen actually protects `cryptsetup luksOpen` protects data at rest — while the machine is powered off or the volume is locked. Once the disk is unlocked and the system is running, the data is exactly as exposed as it would be on an unencrypted disk; full-disk encryption defends against a stolen or seized machine, not against a compromise of the running system.

2.3 Encrypt swap separately if you skipped full-disk encryption Swap can contain sensitive memory contents — passwords typed into forms, decrypted document fragments, private key material — even from a session that otherwise felt perfectly secure. Set it up with `cryptsetup` and a random key generated at each boot (`/dev/urandom`-backed), since swap doesn't need to survive a reboot the way your root filesystem does.

2.4 Understand the tradeoff before enabling TPM auto-unlock Convenient — the disk unlocks automatically at boot without typing a passphrase — but it narrows your threat model down to "an attacker never gets physical access to the running or sleeping machine." Know that's the deal before you make it, particularly on a laptop that travels.

> **CAUTION:** A firmware/BIOS update, a motherboard swap, or certain Secure Boot changes can invalidate the TPM's stored state and silently disable auto-unlock —

always keep the LUKS passphrase memorized or safely stored as a fallback, never rely on TPM auto-unlock as the only way in.

2.5 Use a strong LUKS passphrase, not a short PIN LUKS key slots are separate from your login password, and people routinely treat them far too casually. Brute-forcing a weak disk-encryption passphrase offline, with no lockout or rate-limiting to slow an attacker down, is far easier than most people assume — this is the one password on your system that arguably deserves the most entropy, not the least.

2.6 Keep an offline recovery key somewhere physical Losing a LUKS passphrase means losing the data, permanently — there is no reset-my-password flow for disk encryption. Generate and store a proper recovery key:

```
sudo cryptsetup luksHeaderBackup /dev/sdXn --header-backup-file luks-header.img
```

Store the resulting header backup and your passphrase somewhere physically separate from the machine — a safe, a safety deposit box, anywhere that isn't "also in this laptop bag."

2.7 Encrypt backups, not just the live disk An unencrypted backup drive undoes all the effort spent encrypting the laptop it came from, and rarely announces the fact — backup drives are disproportionately likely to end up forgotten in a drawer, lent to someone, or lost while traveling, precisely because they don't feel like "the important one."

2.8 GPG-encrypt individual files for the gaps full-disk encryption doesn't cover Example: `gpg --encrypt --recipient you@example.com file.txt`. Useful for anything that needs to survive being copied somewhere less trusted than your own disk — an email attachment, a USB drive handed to someone else, a file uploaded to shared cloud storage.

> **CAUTION:** If you lose the private key (or its passphrase), an encrypted file is gone for good — there's no backdoor, by design. Back up your GPG private key and revocation certificate as carefully as any other irreplaceable credential.

2.9 Verify encryption actually engaged after a reboot A misconfigured bootloader can silently fall back to an unencrypted kernel or initramfs path without any obvious symptom. Check `lsblk -f` shows `crypto_LUKS` on the root device after every significant system update — don't just assume a working setup stays working forever.

---

3. Secure Communication & Remote Access

Everything in this category changes how you're allowed to reach a machine remotely — test each change from a session or console access you can fall back on, not the only connection you have to the box.

3.1 Disable SSH password authentication entirely Example: `PasswordAuthentication no` in `sshd_config`, then `systemctl restart sshd`. Key-based auth alone removes an entire class of brute-force login attempts — there's simply no password to guess.

> **CAUTION:** Confirm key-based login actually works in a second, separate terminal before you close the session you're currently using to make this change. If the key isn't set up correctly, this locks you out with no password fallback left.

```
# In sshd_config:
PubkeyAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
```

3.2 Disable root login over SSH Example: `PermitRootLogin no` — log in as a normal user and use `sudo` instead, so every remote action is attributable to a specific person, not to an anonymous "root." Combine this with the password-auth change above and every login is both traceable and key-verified.

> **CAUTION:** Make sure your non-root user is already in the sudoers group and can actually authenticate with a key before you disable root login — otherwise you can end up with a server nobody can administer remotely at all.

3.3 Use ed25519 keys instead of legacy RSA Example: `ssh-keygen -t ed25519 -C "you@example.com"`. Smaller, faster to verify, and it sidesteps 1024-bit RSA keys — genuinely weak, and still floating around in old scripts, tutorials, and forgotten `authorized_keys` files from years ago. (2048-bit RSA isn't in the same boat: NIST still rates it acceptable, with no sunset date currently in view — but ed25519 remains smaller and faster to verify regardless, so there's little reason to reach for RSA at all for a new key.)

3.4 Rate-limit SSH login attempts with fail2ban Automatically bans IPs after a configurable number of failed logins, cutting off brute-force attempts without you having to watch the logs yourself. A default install watching `/var/log/auth.log` catches the vast majority of automated scanning traffic within minutes of it starting.

3.5 Restrict SSH access by source, not just by password strength Example: `AllowUsers you@203.0.113.0/24` in `sshd_config`, or better, a VPN-only bastion host that's the sole path in. This limits who can even attempt to authenticate in the first place — a much stronger position than relying on authentication strength alone against unlimited attempts from anywhere.

3.6 Use end-to-end encrypted messaging for anything sensitive SMS and most default messaging apps are not end-to-end encrypted out of the box — carriers, platform operators, and in some cases advertisers can technically access message content. Don't assume "message" automatically means "private"; Signal and similar tools that are end-to-end encrypted by default and by design are a meaningfully different guarantee.

3.7 Sign your Git commits Example: `git config commit.gpgsign true`, with a GPG or SSH signing key configured. Proves a commit actually came from you, which matters more than people think the moment a repo has more than one contributor — an unsigned commit history is trivially easy to spoof by anyone with push access.

3.8 Don't paste secrets into chat apps "just this once" Most chat clients keep history forever, sync to the cloud, and stay fully searchable by anyone with account access — including, eventually, people who join the channel years later. Treat anything typed there as permanent and discoverable; use a secrets manager or a one-time-view sharing tool instead, even for "just this once."

3.9 Use an SSH config file instead of memorizing flags A `~/.ssh/config` with per-host `IdentityFile` entries keeps you from ever accidentally offering the wrong key to the wrong host — which matters a lot if you use different keys for work, personal projects, and anything sensitive, since a key offered (even unsuccessfully) can still be logged by the receiving server.

3.10 Rotate SSH keys periodically, and after any device changes hands A key that's been sitting untouched on a laptop for six years is a key you've probably stopped thinking about entirely — which is exactly the problem. Set a calendar reminder, and treat "the laptop went in for repair" or "I lent this to someone for a week" as an automatic trigger to rotate, not just a maybe.

---

4. System & Kernel Hardening

A few of these change what's allowed to run or connect on the machine — test on a non-critical box or a snapshot/VM first, since an overly strict policy can break a working service just as effectively as an attacker could.

4.1 Turn on AppArmor and actually check its status Example: `aa-status` shows which profiles are enforcing versus merely logging. A profile stuck in "complain" mode isn't protecting anything yet — it's just taking notes; move confirmed-safe profiles to `aa-enforce` deliberately, one at a time.

4.2 Run systemd-analyze security on the services that matter Example: `systemd-analyze security sshd.service` scores a unit's sandboxing on a 0–10 exposure scale and lists exactly which hardening directives are still missing, with suggestions — a genuinely useful starting checklist rather than a vague "harden your services" instruction.

4.3 Sandbox systemd services with capability limits Directives like `ProtectSystem=strict`, `NoNewPrivileges=yes`, and `PrivateTmp=yes` shrink what a compromised service can actually reach, even if an attacker gets code execution inside it.

> **CAUTION:** Test overrides on a copy of the service first — an overly strict sandbox can silently break a service that needs to write to a path you didn't account for, and the failure often just looks like the service refusing to start with a vague error.

```
# systemctl edit myservice.service
[Service]
ProtectSystem=strict
ProtectHome=yes
NoNewPrivileges=yes
PrivateTmp=yes
```

4.4 Keep unattended security updates on for headless servers `unattended-upgrades` closes the gap between a patch shipping and you remembering to apply it — often the entire ballgame for a box you don't log into daily. Configure it to email you a summary rather than run silently forever, so a failed update doesn't go unnoticed for months.

4.5 Disable services you don't actually use Every listening service is attack surface, whether or not you ever intended to expose it. `systemctl list-unit-files --state=enabled` is a solid starting point for a cleanup pass — it's common to find print spoolers, Avahi/mDNS, or an old database server still enabled on a machine that hasn't needed them in years.

4.6 Run a file-integrity monitor AIDE builds a cryptographic baseline of file hashes across the system and flags unexpected changes on a scheduled scan — often the first real sign something's wrong on a server nobody's watching closely day to day. Store the baseline database somewhere an attacker who compromises the box can't also modify, or the whole point is defeated.

4.7 Whitelist trusted USB devices with USBGuard Blocks unrecognized USB devices from being used at all — mice, keyboards, storage, everything — which matters more than people assume given how easy "BadUSB"-style attacks (a device that pretends to be a keyboard and injects commands) have become to pull off with cheap hardware.

> **CAUTION:** Generate and review the policy from your own currently-connected devices before enabling enforcement, especially on a machine you manage remotely with no physical KVM access — getting this wrong can lock out your own keyboard and mouse with no local way to fix it.

```
sudo usbguard generate-policy > /etc/usbguard/rules.conf
# review the file, THEN:
sudo systemctl enable --now usbguard
```

4.8 Audit sudo usage, don't just grant it `auditd` watching `/etc/sudoers`, plus a periodic read of `/var/log/auth.log`, tells you who actually used elevated privileges and when — not just who's technically allowed to. On a shared or multi-admin system, this distinction matters a lot more than the access list alone.

4.9 Keep the kernel itself updated, not just userspace packages Kernel-level vulnerabilities bypass almost everything else on this list, since the kernel underlies every sandbox, container boundary, and permission check on the system. Check `uname -r` against your distro's currently supported kernel on a regular cadence, and reboot after kernel updates — a patched-but-unbooted kernel provides zero protection.

4.10 Run rootkit checks periodically `rkhunter --check` or `chkrootkit` won't catch a sophisticated, custom-built attacker, but they catch a surprising number of unsophisticated, off-the-shelf rootkits — which is most of what actually circulates. Run them from a live/rescue environment occasionally for a second opinion, since a compromised system can theoretically lie to tools running on it.

4.11 Use Lynis for a full hardening audit Example: `lynis audit system` — a scored report of concrete hardening suggestions specific to your actual configuration, not a generic checklist copied from a blog post. It's a good tool to re-run every few months, since "hardened" isn't a permanent state — new packages and config drift erode it gradually, with no alarm going off when they do.

4.12 Don't run daily workloads as root A compromised root shell is a compromised machine, full stop. A compromised regular user account is a much smaller, much more containable problem — one that still requires a privilege-escalation step before it becomes as bad as starting from root already.

---

5. Browser & Application Privacy

5.1 Turn on resist-fingerprinting mode in Firefox `privacy.resistFingerprinting` in `about:config` normalizes your browser's reported fingerprint — screen size, timezone, fonts, canvas rendering — so you blend in with thousands of other users instead of standing out as a unique, trackable configuration.

> **GOOD TO KNOW:** This setting can break some sites in small ways (odd timezone display, blocked canvas-based features) — it's a real tradeoff between fingerprinting resistance and everyday convenience, not a free lunch.

5.2 Use per-site container tabs Firefox's Multi-Account Containers extension isolates cookies and site data per container, so your shopping tab genuinely can't see what your banking tab is doing — and neither can the ad networks embedded on either site, since they can't correlate identifiers across the container boundary.

5.3 Sandbox desktop apps with Flatpak permissions Example: `flatpak override --nofilesystem=home org.example.App`. Most Flatpak apps request far more filesystem access than they actually need by default — `flatpak info --show-permissions org.example.App` is worth running on anything you install, since "sandboxed" doesn't automatically mean "tightly sandboxed."

5.4 Use Firejail for anything you don't fully trust yet Example: `firejail firefox` wraps a process in its own sandbox with restricted access to the rest of the system — handy for one-off downloads, unfamiliar binaries, or testing something before deciding whether it belongs on your main system at all.

5.5 Review browser extension permissions on a schedule An extension that asked for "read and change all your data on all websites" two years ago, back when it was a small useful tool, is worth a second look today — extensions get sold, get their maintainers change, or end up bundled with tracking after the fact with no changelog calling it out, and the original permission grant just carries forward regardless.

5.6 Block third-party trackers at the network level A DNS-based blocklist (Pi-hole, or even a well-maintained hosts file) stops a huge share of tracking before it ever reaches the browser at all, protecting every app on the network — not just whichever browser has an extension installed.

5.7 Don't reuse your primary browser profile for sensitive research A separate profile (`firefox -P`) with its own cookies, history, and logged-in sessions keeps unrelated activity from cross-contaminating — genuinely useful for anything from job hunting while employed to medical research you'd rather not have algorithmically linked to your main identity.

5.8 Check what your apps are actually phoning home to `ss -tulpn` paired with a look at your firewall's outbound connection log will surprise you about how "chatty" some desktop apps are by default — telemetry, update checks, and analytics calls that happen whether or not you ever opted in explicitly.

5.9 Keep JavaScript engines updated, not just the browser chrome Most real-world browser exploits target the JS engine specifically, not the visible UI. An old Chromium or Firefox core hiding under a current-looking rebranded skin (a common pattern in some "privacy browsers") is still an old, vulnerable engine underneath — check the actual engine version, not just the app's marketing version number.

---

6. Authentication & Credentials

6.1 Use a password manager, not a memorized pattern KeePassXC (local, encrypted database, no cloud dependency) or Bitwarden (synced across devices) both beat "the same password with a number appended" by an enormous margin — a password manager lets every single account have a genuinely unique, high-entropy password without you needing to remember any of them.

6.2 Add a hardware key as a second factor A FIDO2 key — `pamu2fcfg` for local login, WebAuthn for supported sites — resists phishing in a way SMS and app-based one-time codes fundamentally can't, because the cryptographic challenge is bound to the actual site's domain and can't be relayed to a fake one.

6.3 Never share a login between more than one purpose A leaked password from an unrelated breach is only dangerous to accounts that reuse it — which, for most people, is most of them, because breaches happen to services you don't control and can't predict. A password manager removes the friction that makes reuse tempting in the first place.

6.4 Set a real passphrase for sudo, disable blanket NOPASSWD `NOPASSWD` entries in sudoers are convenient right up until a script, a hijacked session, or an unattended terminal turns that convenience into a wide-open door with nothing standing in the way at all.

> **GOOD TO KNOW:** If automation genuinely needs passwordless sudo, scope it to the exact command it needs — never a blanket `ALL=(ALL) NOPASSWD: ALL` — so the convenience doesn't outlive its original narrow purpose.

6.5 Use SSH keys with a passphrase, not a bare private key An unencrypted private key sitting on disk is exactly one stolen laptop away from becoming a stolen identity — anyone who copies the file can use it immediately, with nothing else required. A passphrase (combined with `ssh-agent` so you're not retyping it constantly) adds a real barrier.

6.6 Rotate credentials after anyone else has had physical access A repair shop, a lent laptop, a shared conference machine, a device that briefly left your sight at a border crossing — treat all of these as a reason to rotate relevant credentials, not just a reason to assume it's probably fine. Rotation is cheap; the alternative isn't.

6.7 Use ssh-agent instead of typing your passphrase into scripts `ssh-add` loads your key into memory for the session, so automation never needs the passphrase embedded anywhere in a file, an environment variable, or — worse — hardcoded directly into a script that might end up committed to a repo.

6.8 Check breach-notification data periodically Knowing a credential is already circulating is the difference between rotating it calmly on your own schedule and finding out the hard way, after it's already been used. Make it a habit, not a one-time check after a scare.

6.9 Lock down account recovery paths as carefully as the password itself A strong password behind a weak "recovery email" (itself protected by an even weaker password) or an SMS-

based reset is only ever as strong as its weakest recovery path — attackers who can't guess your password will happily go around it through account recovery instead.

---

7. Secure Deletion & Digital Hygiene

Two of these tips involve commands that permanently destroy data with no undo — double-check the target device name before you run anything here, every single time.

7.1 Know that shred doesn't reliably work on SSDs Wear leveling means the physical flash cells behind a file's logical block address move around constantly, transparently, as part of normal SSD operation. Overwriting a file "in place" with `shred` often never touches the actual physical data on flash storage at all — the drive just writes the new pattern to a different physical cell and marks the old one as stale.

7.2 Use full-disk encryption as the real answer for SSD deletion Deleting the LUKS key (`cryptsetup luksErase` or simply destroying all key slots) is instant, verifiable, and doesn't depend on the physical quirks of the storage medium the way file-shredding tools do — cryptographic erasure sidesteps the wear-leveling problem entirely instead of fighting it.

7.3 Strip metadata before sharing photos and documents Example: `exiftool -all= photo.jpg` — removes GPS coordinates, device model, and timestamps most people never realize are embedded in the file. For a whole folder at once: `exiftool -all= -r /path/to/folder`.

7.4 Securely wipe a drive before disposal or resale `blkdiscard /dev/sdX` (SSD — uses the drive's own secure-erase command, fast and thorough) or several passes of `shred -vz /dev/sdX` (spinning disks) — pick the tool that actually matches the storage medium, since the SSD advice above applies here too.

> **CAUTION:** Getting the device name wrong here wipes the wrong drive, permanently and with no recovery. Always verify the target first — never trust muscle memory or a command from your shell history:
>
> ```
> lsblk
> # confirm the model, size, and mount points match the drive you mean
> # to wipe, THEN and only then run the actual wipe command
> ```

7.5 Clean up temp and swap files on a schedule `systemd-tmpfiles --clean` paired with encrypted swap closes a surprisingly common gap in an otherwise "clean" system — temp directories build up fragments of documents, thumbnails, and cache data that outlive the application that created them.

7.6 Check your clipboard manager for lingering secrets Clipboard history tools happily store every password you've ever copied unless told not to — including ones copied straight out of a password manager for a single, supposedly momentary paste. Exclude password-manager copies explicitly, or clear clipboard history on a short timer.

7.7 Empty the trash like you actually mean it Most file managers' "delete" is a move, not an erase — the file stays fully recoverable in the trash folder until it's emptied, and often recoverable with forensic tools for a while after that too, since the underlying blocks usually aren't overwritten immediately.

7.8 Verify what "clean install" actually leaves behind A factory reset or reinstall doesn't guarantee data removal across every filesystem and storage combination — journaling filesystems, SSD wear leveling, and leftover partitions can all leave recoverable fragments behind. Verify rather than assume, especially before handing hardware to someone else or sending it for recycling.

---

8. Logging, History & Footprint Control

8.1 Control what your shell remembers `HISTCONTROL=ignorespace` in your shell config lets you prefix any sensitive command with a leading space to keep that one line out of history entirely — handy for a one-off command containing a password or token you don't want sitting in `~/.bash_history` indefinitely.

8.2 Know what journald is actually retaining `SystemMaxUse=200M` in `/etc/systemd/journald.conf` caps retention by size. Check `journalctl --disk-usage` to see what's already piled up by default — on a long-running server it's often far more than people expect, and it's rarely been reviewed for what's actually in it.

8.3 Review auth logs for what they're actually recording `/var/log/auth.log` keeps a running record of every sudo invocation, every SSH login attempt (successful or not), and every `su` — useful for spotting intrusions, and just as useful to anyone else who gets read access to that file, which is worth remembering when setting its permissions.

8.4 Clear bash history deliberately, not by hand-editing the file `history -c && history -w` clears both the in-memory session history and the on-disk file cleanly, instead of leaving a half-edited `~/.bash_history` with inconsistent line numbers or a truncated last entry from a manual edit gone wrong.

8.5 Think about what your login banner is broadcasting A MOTD announcing the exact distro version and kernel release to anyone who connects — even before authenticating — is free reconnaissance handed straight to an attacker scanning for a specific known vulnerability. A generic banner costs nothing and gives away nothing.

8.6 Check what timestamps your files are carrying `stat file.txt` exposes creation, modification, and access times that can reconstruct a fairly precise activity timeline you never meant to publish — worth checking before sharing a file bundle where the timing of your work isn't something you want visible.

8.7 Be deliberate about what gets logged in the first place The least risky log entry is the one that was never verbose enough to be sensitive to begin with. Review what a service actually logs (request bodies? full headers? user input?) rather than assuming default logging levels are automatically safe — cleanup after the fact is a poor substitute for restraint at the source.

8.8 Remember what "private browsing" actually covers It means private on this device only — it clears local history and cookies when the window closes. It does absolutely nothing to hide traffic from your network operator, your VPN provider, or the sites you visit; it is not a substitute for encrypted DNS or a VPN, and treating it as one is a common, costly misunderstanding.

---

9. Physical Security & Opsec Habits

This is the category where the cheapest tips (a screen lock shortcut, a cable lock) prevent the same real-world outcomes as some of the more elaborate technical ones — don't skip the physical basics because they feel less "technical."

9.1 Lock your screen every single time you step away `loginctl lock-session` bound to a keyboard shortcut removes the excuse of "it's just for a second" — the average opportunistic look-over-the-shoulder or quick USB-drop attack takes far less time than that "second" actually lasts.

9.2 Set a BIOS/UEFI password separate from your disk encryption passphrase Without one, boot order and boot media can potentially be changed by anyone with a few unsupervised minutes and a USB stick, opening the door to boot-time attacks that happen before your disk encryption ever gets a chance to matter.

> **CAUTION:** Write this password down and store it somewhere separate from the machine. Forgetting a BIOS/UEFI password can mean a trip to the manufacturer or, on some hardware, replacing the motherboard entirely — there's often no user-accessible reset.

9.3 Use a cable lock in shared or public spaces Opportunistic laptop theft doesn't care how good your encryption is if the machine simply walks out the door while you're in line for coffee — a cable lock is cheap, unglamorous, and stops the single most common physical-security failure mode outright.

9.4 Cover the webcam physically, not just in software A hardware slide or a piece of opaque tape doesn't depend on trusting that no application, driver, or piece of malware has re-enabled the camera in software without any visible sign — a physical cover is a guarantee, a software toggle is a claim.

9.5 Prefer a passcode over biometric unlock when crossing borders In a number of jurisdictions, compelling a passcode carries meaningfully stronger legal protections than compelling a fingerprint or face unlock, because a passcode is treated as something you know (protected) while a biometric is treated as something you are (often not). Know the specific rules for where you're traveling before you travel, not after — and disable biometric unlock temporarily if in doubt.

9.6 Carry a dedicated travel device for sensitive trips A minimal, disposable setup with only what's needed for the specific trip limits what's actually exposed if the device is seized, searched, or lost — no years of accumulated correspondence, no full password manager vault, nothing beyond what that trip genuinely requires.

9.7 Use tamper-evident seals on hardware you can't watch constantly A broken seal on a server chassis or a laptop's bottom panel is far cheaper to notice than discovering unauthorized access after the fact through some other, indirect symptom — it turns "I have no idea if anyone opened this" into a quick visual check.

9.8 Store a spare recovery key somewhere that isn't "on the device it recovers" The single most common way to lose access to your own encrypted data is storing the only recovery key inside the thing it's supposed to unlock — a recovery key on the same encrypted disk it recovers protects against exactly nothing.

9.9 Treat "just this once, on someone else's computer" as a real risk A one-time login on an untrusted machine can mean a one-time keylogger — permanently, and often silently, with the credentials harvested long before you'd ever notice anything was wrong. If it's genuinely unavoidable, change that password from a trusted device immediately afterward.

---

10. Anonymity Tooling & Specialized Distros

10.1 Reach for Tails when you need amnesia, not just privacy Tails boots from removable media and forgets everything on shutdown by design, routing all traffic through Tor automatically — a fundamentally different guarantee than "a privacy-focused desktop" that just happens to be configured carefully, because nothing persists between sessions unless you explicitly set up encrypted persistent storage.

10.2 Use Qubes OS when compartmentalization matters more than convenience Separate qubes (individually isolated VMs) for banking, browsing, and work mean a compromise in one doesn't automatically become a compromise of everything — the security boundary is enforced by the hypervisor itself, not just by discipline or careful browsing habits.

10.3 Understand what Whonix actually isolates Routing all traffic through Tor at the gateway VM level protects against a compromised workstation VM leaking your real IP even if malware gets full control of it — a meaningfully different, stronger threat model than a browser-only Tor setup, where a single browser exploit can potentially deanonymize you directly.

10.4 Don't reach for an anonymity distro without an anonymity-shaped problem Tails and Qubes solve real problems very well, but running them for everyday tasks that don't need that level of isolation mostly adds friction — slower hardware compatibility, a steeper learning curve, more manual steps — without adding meaningful safety for a threat model that never needed it.

10.5 Match the tool to the actual adversary, not the scariest one you can imagine The mitigations for "an ex-partner checking your laptop" and "a nation-state" are not the same list, and treating them as the same list usually means doing neither one well — an honest, specific threat model is worth more than maximal paranoia applied indiscriminately.

10.6 See the full distro rundown for the deep cuts *100 Linux Distros You Never Knew Existed* covers a whole chapter of security- and anonymity-focused distros — Kali, Qubes, Whonix, Tails, and several far more obscure entries — in more detail than fits in a single category here.

---

11. Myths, Security Theater & Overkill

11.1 "chmod 777 fixes permission errors" It does, the same way removing a smoke detector fixes it beeping. It works, right up until it very much doesn't — and by then the file or directory has usually been writable by literally anyone on the system for however long it took to notice.

11.2 Changing your SSH port isn't security, it's noise reduction Moving from 22 to 2222 cuts down on automated scanner spam cluttering your logs. It does nothing against anyone actually targeting you specifically — a targeted attacker runs a full port scan in seconds and finds it immediately. Worth doing for your sanity and your log's signal-to-noise ratio — not for your safety.

11.3 A password with a "!" at the end isn't a strong password Attackers know that trick too — it's one of the first substitutions any modern cracking wordlist tries. Length and genuine randomness matter far more than the handful of predictable pattern substitutions everyone, including the attacker, already expects.

11.4 Antivirus software isn't doing much on a well-maintained Linux desktop The bigger wins are patching promptly, minimizing what's installed and running, and never executing random binaries as root — not a signature scanner idling in the background, consuming resources while catching almost nothing that a well-maintained system wasn't already resistant to.

11.5 Hiding your hostname doesn't hide your machine `hostname totally-not-a-server` is not a security control. Anyone doing real reconnaissance is reading open ports and service banners, not admiring your naming sense of humor — though it is, admittedly, still funny.

11.6 A VPN doesn't make you anonymous, it makes you someone else's problem instead of your ISP's It shifts who can see your traffic from your ISP to your VPN provider. It does not erase the fact that someone still can — anonymity and privacy are related but genuinely different guarantees, and marketing copy loves to blur that line.

11.7 "I'll just remember not to run that as root" is not a security policy It's a New Year's resolution, and it has roughly the same success rate by February — actual technical controls (sudo, restricted accounts, policy) beat willpower every single time willpower is tested at 2am during an incident.

11.8 Incognito mode is not a VPN, a firewall, or a disguise It forgets your local history when the window closes. That is the complete list of things it actually does — your network operator, your VPN (if any), and every site you visit can still see exactly what they could see before.

11.9 A sticker over the webcam is good opsec; a sticker over the microphone in your head is not Cameras get covered, microphones get forgotten — and a lot of modern attacks care far more about audio than most people assume, because a covered camera still feels safe while a live, unmuted microphone keeps working regardless.

11.10 Security through obscurity isn't worthless, it's just not sufficient Hiding a service — an unlisted port, an unpublished endpoint, a "nobody knows this exists" admin panel — helps right up until it's found, and then it offers exactly zero additional protection. Treat it as a bonus layer on top of real controls, never as the layer itself.

11.11 The most overkill security setup in the world doesn't help if the password is still "changeme123" Sophistication at the top of the stack doesn't compensate for weakness at the bottom of it — hardware keys, air-gapped backups, and a fully audited firewall are all wasted effort if the actual account password was never changed from the default. Check the boring stuff first. Always.

Appendix: Command Cheat Sheet

Every command or named tool mentioned in this guide, grouped the same way the guide itself is. Tip # points back to the full explanation; See also points to another SplitZrate title when it genuinely covers the same ground in more depth — left blank everywhere it doesn't, rather than forced. D = Ubuntu Server, Distilled · LC = 100+ Linux Commands · BT = Bash 100+ Tricks. Category 11 (Myths, Security Theater & Overkill) is commentary, not reference material, so it isn't included here.

Network & Traffic Privacy

Command / Tool	What it does	Tip #	See also
`sudo ufw allow OpenSSH && sudo ufw default deny incoming`	Default-deny firewall, SSH allowed first (see Tip 1.8 for the safe order)	1.8	D §4.2
`ss -tulpn`	List every open port and the process behind it	1.5	D §4.7 · LC: ss
`nmcli connection modify Wi-Fi wifi.cloned-mac-address random`	Randomize the Wi-Fi MAC address	1.3	
`DNSOverTLS=yes` (`/etc/systemd/resolved.conf`)	Encrypt DNS lookups	1.2	
`curl ifconfig.me`	Confirm your public IP actually changed after connecting a VPN	1.11	
WireGuard	Lightweight, kernel-native VPN protocol for your own tunnels	1.4 / 1.6	
`unbound`	Run your own validating DNS resolver	1.10	
Tor	Anonymity network routing traffic through independent relays	1.9	

Disk & Data Encryption

Command / Tool	What it does	Tip #	See also
`cryptsetup luksOpen`	Unlock a LUKS-encrypted volume	2.2	
`sudo cryptsetup luksHeaderBackup /dev/sdXn --header-backup-file luks-header.img`	Back up a LUKS header before you need it	2.6	
`gpg --encrypt --recipient you@example.com file.txt`	Encrypt a single file with GPG	2.8	
`lsblk -f`	Confirm `crypto_LUKS` shows on the root device	2.9	LC: lsblk

Secure Communication & Remote Access

Command / Tool	What it does	Tip #	See also
`PasswordAuthentication no` / `PermitRootLogin no` (sshd_config)	Disable SSH password login and root login (see Tips 3.1–3.2 for the safe sequence)	3.1 / 3.2	D §4.3
`ssh-keygen -t ed25519 -C "you@example.com"`	Generate a modern SSH key pair	3.3	D §2.5
`sudo apt install fail2ban`	Ban IPs automatically after repeated failed logins	3.4	D §4.4
`AllowUsers you@203.0.113.0/24` (sshd_config)	Restrict SSH access to specific source IPs	3.5	
`~/.ssh/config` (IdentityFile)	Bind the right key to the right host automatically	3.9	
`git config commit.gpgsign true`	Sign Git commits with GPG	3.7	
Signal	End-to-end encrypted messaging by default	3.6	

System & Kernel Hardening

Command / Tool	What it does	Tip #	See also
`aa-status`	Check which AppArmor profiles are enforcing vs. just logging	4.1	D §4.6
`systemd-analyze security sshd.service`	Score a systemd unit's sandboxing	4.2	
`ProtectSystem=strict` / `NoNewPrivileges=yes` / `PrivateTmp=yes`	Sandbox a systemd service (see Tip 4.3 for a full override)	4.3	
`unattended-upgrades`	Apply security updates automatically	4.4	D §4.5
`systemctl list-unit-files --state=enabled`	List every enabled service, for a cleanup pass	4.5	LC: systemctl
AIDE	File-integrity monitoring via a hash baseline	4.6	
`sudo usbguard generate-policy`	Whitelist currently-connected USB devices before enforcing	4.7	
`auditd`	Audit sudo usage and file access at the kernel level	4.8	D §2.4
`uname -r`	Check the currently running kernel version	4.9	
`rkhunter --check` / `chkrootkit`	Scan for known rootkits	4.10	
`lynis audit system`	Full, scored system hardening audit	4.11	

Browser & Application Privacy

Command / Tool	What it does	Tip #	See also
`privacy.resistFingerprinting` (about:config)	Normalize Firefox's browser fingerprint	5.1	
Multi-Account Containers	Isolate cookies and site data per container tab	5.2	
`flatpak override --nofilesystem=home org.example.App`	Restrict a Flatpak app's filesystem access	5.3	
`firejail firefox`	Sandbox any application in its own restricted environment	5.4	
Pi-hole	Block trackers network-wide at the DNS level	5.6	

Command / Tool	What it does	Tip #	See also
`firefox -P`	Launch a separate, isolated browser profile	5.7	

Authentication & Credentials

Command / Tool	What it does	Tip #	See also
KeePassXC / Bitwarden	Local or synced password manager	6.1	
`pamu2fcfg`	Register a FIDO2 hardware key for local login	6.2	
`NOPASSWD` (sudoers, scoped to one command)	Passwordless sudo — only for the exact command that needs it	6.4	D §2.4
`ssh-agent` / `ssh-add`	Hold a decrypted SSH key in memory for the session	6.5 / 6.7	

Secure Deletion & Digital Hygiene

Command / Tool	What it does	Tip #	See also
`lsblk`	Confirm the target device before any destructive wipe — every time	7.4	LC: lsblk
`blkdiscard /dev/sdX`	Securely wipe an SSD via its own secure-erase command	7.4	
`shred -vz /dev/sdX`	Securely wipe a spinning disk with multiple passes	7.1 / 7.4	BT 11.4
`cryptsetup luksErase`	Instantly, cryptographically erase a LUKS volume	7.2	
`exiftool -all= photo.jpg`	Strip GPS, device, and timestamp metadata from a file	7.3	
`systemd-tmpfiles --clean`	Clean up temp files on a schedule	7.5	BT 11.5

Logging, History & Footprint Control

Command / Tool	What it does	Tip #	See also
`HISTCONTROL=ignorespace`	Keep space-prefixed commands out of bash history	8.1	LC: history
`SystemMaxUse=200M` (journald.conf) / `journalctl --disk-usage`	Cap and check journald's disk usage	8.2	D §8.3 + §11.3 · LC: journalctl
`/var/log/auth.log`	Review sudo, SSH, and su activity	8.3	D §2.4
`history -c && history -w`	Clear bash history cleanly, in-memory and on disk	8.4	LC: history
`stat file.txt`	Inspect a file's creation, modification, and access timestamps	8.6	LC: stat

Physical Security & Opsec Habits

Command / Tool	What it does	Tip #	See also
`loginctl lock-session`	Lock the screen on demand, bindable to a shortcut	9.1	

Anonymity Tooling & Specialized Distros

Command / Tool	What it does	Tip #	See also
Tails	Amnesic live OS — forgets everything on shutdown, routes all traffic through Tor	10.1	
Qubes OS	Compartmentalized OS using isolated VMs ("qubes")	10.2	
Whonix	Tor-gateway VM isolation for a compromised workstation VM	10.3	

© SplitZrate 2026 · 100+ Stealth & Safety Tips for Linux Users