# Sync & Secrets CLI

Five tools that all move or protect files, and five very different mental models. `rclone` talks to 70+ cloud backends and treats them like a filesystem. `rsync` is the decades-old local/SSH workhorse everything else gets compared to. Syncthing is continuous peer-to-peer sync with no central server. Proton Drive CLI and Proton Pass CLI are both new (2025–2026) single-shot scripting tools built on Proton's end-to-end encryption — not background sync engines. Knowing which one to reach for saves more time than knowing any one of them deeply.

## On This Page



## Which tool for the job

| Tool | Best for |
|---|---|
| **rclone** | One-off or scheduled transfers to/from cloud storage (S3, Drive, Dropbox, 70+ backends). Also mounts cloud storage as a local filesystem. |
| **rsync** | Local-to-local or local-to-SSH-remote file sync. Fastest, most battle-tested delta-transfer tool for machines you control. |
| **Syncthing** | Continuous two-way sync between your own devices, no cloud middleman, no central server. Set-and-forget, not scripted per-run. |
| **Proton Drive CLI** | Scripted, one-shot actions against Proton Drive — upload after a build, snapshot before an audit, revoke access. Not a background sync engine (yet). |
| **Proton Pass CLI** | Pull secrets/credentials into scripts, CI/CD, and SSH workflows without ever putting them in plaintext files or shell history. |

## rclone — Remotes & Config

Rclone talks to a remote through a named config entry — set it up once, reference it by name everywhere else.

```
rclone config                    # interactive config wizard
rclone config create name type   # create a remote non-interactively
rclone config show                # show all configured remotes
rclone config file                # show the config file path
rclone listremotes                # list all configured remote names
rclone obscure 'password'         # obscure a password for the config file (not real encryption)
```

## rclone — Copy, Sync & Move

`copy` only adds/updates — it never deletes. `sync` makes the destination an exact mirror, deleting anything not in the source. That distinction is the whole ballgame.

```
rclone copy /local remote:backup       # local → cloud, additive only, never deletes at destination
rclone copy remote:backup /local       # cloud → local restore, additive only
rclone sync /local remote:backup       # local → cloud MIRROR — deletes at destination what's gone locally
rclone move /local remote:backup       # copy then delete from source once verified
rclone check src dst                   # verify source and destination match, no transfer
rclone --dry-run sync src dst          # preview a sync with zero changes made — ALWAYS run this first
```

> `sync` **deletes at the destination.** Always run with `--dry-run` first, or add `--max-delete N` so it aborts instead of silently wiping a destination that changed more than expected.

```
# Safer sync with a backup of anything it would delete/overwrite
rclone sync src dst --backup-dir remote:old-versions/$(date +%Y%m%d) --max-delete 100
```

```
# Common flags worth knowing
--dry-run              # preview, no changes — use first
-P / --progress        # real-time transfer progress
--bwlimit 10M           # cap bandwidth
--transfers 16 --checkers 32 --fast-list   # high-performance transfer
--include "*.jpg" --exclude "*"            # filter what moves
```

## rclone — Bisync (Two-Way)

Bidirectional sync between two paths, tracking changes on both sides between runs. Rclone's own docs call this an advanced command — read the limitations section before trusting it with anything you can't lose.

```
# First run establishes the baseline (required)
rclone bisync /local remote:path --resync

# Every run after that
rclone bisync /local remote:path
```

> `--resync` only on the very first run (or deliberately, to re-baseline). Running it again treats one side as authoritative and can overwrite changes on the other. `--max-delete N` and `--check-access` are worth adding for anything important.

## rclone — Mount & Filters

```
rclone mount remote:path /mnt/point --daemon                  # mount as a local FUSE filesystem, backgrounded
rclone mount remote:path /mnt/point --vfs-cache-mode full     # cache files locally for performance / offline rea
fusermount -u /mnt/point         # unmount (Linux)
umount /mnt/point                # unmount (macOS)
rclone copy src dst --min-size 1M --max-size 1G   # only transfer files in a size range
rclone copy src dst --max-age 7d                  # only files modified in the last 7 days
```

## rsync — Basics & the Trailing Slash

Same source-then-destination order as `cp`. The footgun is the trailing slash on the *source* — it changes whether the source directory itself gets copied, or just its contents.

```
rsync -av /src/folder  /dst/    # copies folder itself → /dst/folder/
rsync -av /src/folder/ /dst/    # copies folder's CONTENTS → /dst/*  (no wrapping dir)
```

> Trailing slash on the source means "contents of"; no trailing slash means "the directory itself." This is the single most common rsync mistake, and it's silent — no error, just the wrong layout at the destination.

## rsync — Flags That Matter

| Flag | Meaning |
|---|---|
| `-a, --archive` | archive mode — recursive, preserves perms/times/symlinks/owner (the daily driver) |
| `-v, --verbose` | show what's happening |
| `-z, --compress` | compress data during transfer (helps over slow links) |
| `-P` | shorthand for `--progress --partial` (resumable, shows progress) |
| `-n, --dry-run` | preview only — combine with `-av` to see what would happen |
| `--delete` | delete files at destination that don't exist in source — makes it a real mirror |
| `-L, --copy-links` | follow symlinks and copy the target's data instead of the link |
| `--exclude PATTERN` | skip files/dirs matching a pattern (repeatable) |

> `--delete` turns rsync into a mirror, same risk profile as `rclone sync`. Always pair a first run with `-n` (dry-run) before adding `--delete` for real.

```
# The command you actually run most often
rsync -avP --delete src/ dst/
```

## rsync — Remote Sync over SSH

```
rsync -avz -e ssh src/ user@host:/dst/                          # sync to a remote host over SSH
rsync -avz -e 'ssh -p 2222' src/ user@host:/dst/                 # custom SSH port
rsync -avz -e 'ssh -i ~/.ssh/id_ed25519' src/ user@host:/dst/    # specific SSH key
rsync --link-dest=../previous /src/ /dst/current/                # hard-link unchanged files against a previous b
```

## Syncthing — CLI & Headless Setup

Syncthing is a background daemon with a web GUI by default. On a headless box, everything under `syncthing cli` talks to the REST API of an already-running instance — start the daemon first.

```
syncthing                                          # start the daemon (implicit 'serve')
syncthing serve --no-browser --gui-address=0.0.0.0:8384   # headless-friendly start
syncthing generate --gui-user=admin --gui-password=-       # generate initial config + device ID, no REST API nee
syncthing device-id                                # print this device's ID
syncthing paths                                    # show config/data directory locations
syncthing --version                                # print version info
```

```
# syncthing cli — talks to a running instance's REST API
syncthing cli config folders list
syncthing cli config folders <id> devices add --device-id <ID>
syncthing cli config devices add --device-id <ID> --name <name>
syncthing cli show system
syncthing cli operations restart
syncthing cli errors show
```

Every `cli` subcommand has `--help`, and the tree is deep — `config`, `show`, `operations`, `errors`, and `debug` are the five top-level groups. Discover interactively rather than memorizing the whole hierarchy.

> `syncthing debug reset-database` forces a full rescan/resync and must only be run while Syncthing is *not* running. Data isn't deleted, but every file gets rehashed — expensive on large folders.

## Proton Drive CLI — Auth & Filesystem

A single binary (`proton-drive`), built on the same SDK as the official apps, for scripting Drive from cron, CI, and deploy pipelines. It's not a sync daemon — it runs one operation and exits. Sign-in happens through your browser; sessions are stored in your OS keychain, never a password on the command line.

```
proton-drive auth login                                                  # sign in via browser
proton-drive filesystem list /my-files                                   # list a folder
proton-drive filesystem upload ./reports/* /my-files/Reports --conflict-strategy skip
proton-drive filesystem download /my-files/Reports ./backups             # download to local
proton-drive version                                                     # confirm installed build
proton-drive help                                                        # full command set
```

```
# Typical flow: upload a build artifact after CI finishes
proton-drive auth login
proton-drive filesystem upload ./dist/* /my-files/Releases/$(date +%Y%m%d) --conflict-strategy skip
```

Add `--json` (or `-j`) to any command for machine-readable output when piping into other tools.

> Only the official app has a background sync engine. The CLI runs an operation and exits — good for "do this after the build finishes," wrong tool for "keep this folder continuously mirrored." Use Syncthing or rclone bisync for that.

## Proton Drive CLI — Sharing

```
proton-drive sharing status /my-files/Reports
proton-drive sharing invite --user person@pm.me --role editor /my-files/Reports
proton-drive sharing invite --user person@pm.me --role editor --message "text" /my-files/Reports
```

Fair use follows the same policy as the GUI apps: only upload/download what actually changed. Re-uploading unchanged files or rewriting whole folders on every run can get an account temporarily throttled.

## Proton Pass CLI — Install, Login & Vaults

Binary is `pass-cli`. Requires Pass Plus/Family/Professional or a bundle plan. Everything routes through end-to-end encryption — this isn't a plaintext secrets store with a CLI bolted on.

```
curl -fsSL https://proton.me/download/pass-cli/install.sh | bash   # install (macOS/Linux)
brew install protonpass/tap/pass-cli                               # install via Homebrew
pass-cli login                                                     # web-based login
pass-cli login --interactive user@proton.me                        # login entirely in the terminal
pass-cli info                                                      # verify current session
pass-cli logout                                                    # end the session
```

```
# Vaults
pass-cli vault list
pass-cli vault create --name "Team Project"
pass-cli vault share --vault-name "Team Project" person@company.com --role editor
pass-cli vault delete --vault-name "Old Vault"    # permanent — takes every item with it
```

> `vault delete` is permanent and takes every item in the vault with it. No trash, no undo.

```
# Personal access tokens — scoped, revocable, the right way to log in from a pipeline
PROTON_PASS_PERSONAL_ACCESS_TOKEN="pst_xxxx...xxxx::TOKENKEY" pass-cli login
```

Create the token first with `pass-cli pat create`, then grant it access to only the vaults/items it needs. Scoped tokens beat logging in with a full account session inside a pipeline.

## Proton Pass CLI — Items & Secrets

```
pass-cli item list --vault-name "Personal"
pass-cli item create login --title "GitHub" --username u --generate-password --url https://github.com
pass-cli item view --vault-name "Personal" --item-title "GitHub"     # aliases: get, show
pass-cli item view "pass://Work/GitHub/password"                     # view a field via URI
pass-cli item update --item-id ID --field "password=newvalue"
pass-cli item delete --share-id ID --item-id ID                      # permanent
```

> `item delete` is permanent, same as vault delete. No recovery.

**Secret references — the `pass://` URI.** Every field in Pass can be addressed directly without ever printing the value to your terminal history:

```
pass://<vault-identifier>/<item-identifier>/<field-name>
```

```
pass://Work/GitHub/password
pass://Personal/Email Login/username
pass://Work/GitHub/totp            # current TOTP code
pass://Work/GitHub/totp?totp=uri   # raw otpauth:// URI instead
```

## Proton Pass CLI — Run, Inject & SSH Agent

This is the part that actually replaces "secrets sitting in a `.env` file" — reference them with `pass://`, let the CLI resolve at run time, and the plaintext never touches disk.

```
# Resolve pass:// references from your environment and inject them into a command
export DB_PASSWORD='pass://Production/Database/password'
pass-cli run -- ./my-app
# app sees the real password; your shell history and .env file never do
```

```
# Same, but sourced from .env files (later files win)
pass-cli run --env-file base.env --env-file secrets.env -- node server.js
```

By default `run` masks secret values in stdout/stderr as `<concealed by Proton Pass>`. Use `--no-masking` only when you deliberately need to see them.

```
# SSH agent integration
pass-cli ssh-agent load                                    # load Pass-stored keys into your existing ssh-ager
pass-cli ssh-agent start                                   # run Pass CLI itself as an SSH agent
pass-cli ssh-agent daemon start                             # run the agent detached, with a PID file
pass-cli ssh-agent daemon status                             # check daemon status
pass-cli item create ssh-key generate --title "Deploy Key"    # generate a key stored directly in Pass

export SSH_AUTH_SOCK=$HOME/.ssh/proton-pass-agent.sock         # point SSH at the Pass agent
```

## Things that bite you

- **rclone `sync` and rsync `--delete` both mirror the destination.** Anything at the destination that isn't in the source gets deleted. Dry-run first, every time, no exceptions.
- **rsync's trailing slash on the source silently changes the result.** No error, just the wrong directory layout at the destination. Get in the habit of always deciding the slash on purpose.
- **rclone bisync isn't "fire and forget" like Syncthing.** It's explicitly labeled advanced by its own docs; misusing `--resync` after the first run can overwrite the side you didn't mean to.
- **Proton Drive CLI has no background sync engine.** It's a one-shot tool — great for a cron job or a deploy step, wrong tool if you wanted continuous mirroring. That's what the desktop app (or Syncthing/rclone) is for.
- **Proton Pass personal access tokens beat full account logins in CI.** Scope a token to only the vaults a pipeline needs, not the whole account — if the pipeline leaks, the blast radius stays small.
- **Never put a raw secret value in an environment variable you export normally.** Use `pass://` references with `pass-cli run` so the actual value only ever exists in the child process's memory, never in your shell history.
- **Syncthing's headless setup needs the daemon running before `cli` commands work.** `syncthing cli ...` talks to a REST API — if nothing's serving that API yet, every cli subcommand just fails to connect.