

Contents

File transfer	1
Install	1
LocalSend — phone ↔ computer	1
rsync over SSH — computer ↔ computer	2
Authorize a sender's key (one-time)	2
Receive a transfer	2
How the temporary opening works	2
Security model	3
Google Drive	3
Create the Google OAuth client (once per host)	3
Subcommands	4
How rotate-password avoids a lockout	4
Where it lives	5

File transfer

How files move on and off these machines without the cloud: **LocalSend** for phone ↔ computer, and **rsync over SSH** for computer ↔ computer. Everything is local-network only, opened *temporarily* on untrusted networks (and auto-closed), and installs from a single recipe.

The one cloud exception is a Google Drive mount at ~/GoogleDrive, maintained with rclone-gdrive.

Install

```
just file-transfer          # tools + firewall + ~/Downloads/Transfers + docs
just file-transfer-harden  # receiving machines only: inbound SSH key-only +
LLMNR off
```

file-transfer is idempotent and machine-agnostic — it opens 53317 on whatever firewall zone holds the active interfaces, and renders a per-host ~/Downloads/Transfers/README.md. The helper scripts and the niri menu entries arrive with just stow.

Warning: Why file-transfer-harden is a separate step

It disables SSH **password** login (key-only). Run it only on machines that will *receive* rsync, and make sure you have a key installed — or are sitting at the machine — first. sshd is left disabled; ssh-here starts it on demand.

LocalSend — phone ↔ computer

LocalSend discovers devices over UDP 53317 and transfers over TCP 53317.

- **Home network:** permanently allowed on the home zone — your phone just finds the machine.
- **Untrusted network:** open it for the session from the niri power menu (**Mod+Escape** twice → **Sharing**):

Action	Menu entry	Command
Open + launch	 LocalSend share (temp)	localsend-here on
Close now	 LocalSend close	localsend-here off

The opening is runtime-only with a **30-minute timeout**, so it reverts on its own (and a reboot wipes it).

rsync over SSH — computer ↔ computer

Direction decides what you need:

- **This machine** → **remote** (you push out): nothing to set up — `rsync -av ~/stuff/ user@remote:/dest/`.
- **Remote** → **this machine** (something connects in): inbound SSH is key-only and every transfer key is locked to `~/Downloads/Transfers`.

Authorize a sender's key (one-time)

`add-rsync-key 'ssh-ed25519 AAAA... you@laptop'`

Option	Effect
(default)	upload-only (-wo -no-del) into ~/Downloads/Transfers
--ro	download-only
--dir NAME	lock to ~/NAME instead

The key is installed with a **forced command**:

```
restrict,command="/usr/bin/rrsync -wo -no-del Downloads/Transfers" ssh-ed25519
...
```

`restrict` strips PTY and forwarding; `rrsync` (see `man rrsync`) refuses anything but `rsync` and jails it to that folder. So the key can do nothing but drop files there — no shell, no other paths — even if it leaks.

Receive a transfer

1. Open: menu  **SSH/rsync in (temp)** or `ssh-here on` — starts `sshd`, opens 22 for 30 minutes.
2. Sender: `rsync -av ./files/ <host>:` (the path is relative to the inbox folder).
3. Close: menu  **SSH/rsync close** or `ssh-here off`.

How the temporary opening works

`localsend-here` and `ssh-here` share one pattern:

1. Find the zone of the default-route interface — the network you're *actually* on.
2. Add the port to that zone **at runtime only**, with `--timeout=30m`.
3. Never `--permanent`, so it's gone on reboot/reload and can't silently persist.

That's why a conference-Wi-Fi opening is safe: scoped to the live network, and self-closing.

Security model

- Inbound SSH is **public-key only**, no root — `/etc/ssh/sshd_config.d/10-hardening.conf`.
- `sshd` is **disabled**; started on demand by `ssh-here`, stopped on `off`.
- Transfer keys are `rsync-only`, `upload-only`, `no-delete`, `single-folder`.
- LLMNR is disabled (`file-transfer-harden`) — SMB discovery (`mDNS` + `NetBIOS`) is unaffected.
- Every temporary firewall hole is runtime-only with a 30-minute expiry.

Google Drive

`rclone-gdrive.service` (`systemd` user unit) runs `rclone mount gdrive: ~/GoogleDrive` with a full VFS cache. Everything that makes it work is **per host** and never in this repo:

- the `rclone` config `~/.config/rclone/rclone.conf` (Drive token + OAuth client), **encrypted**;
- its password, in the Secret Service (oo7): `secret-tool lookup service rclone key config_password`;
- a Google OAuth client of its own (Desktop app), **one per host**.

`rclone-gdrive` (also just `gdrive <subcommand>`) sets that up and maintains it.

Create the Google OAuth client (once per host)

In the Google Cloud Console, in any project of yours:

1. **APIs & Services** → **Library** → enable the **Google Drive API**.
2. **OAuth consent screen** (Google Auth Platform): user type **External**, then under **Audience** click **Publish app** so the status is **In production**.
3. **Credentials** (Clients) → **Create credentials** → **OAuth client ID** → application type **Desktop app**, named after the host (`xps`, `daisy`). Keep the client ID and secret for `rclone-gdrive` setup.

Warning: Publish the app — don't leave it in Testing

While the consent screen is in **Testing**, Google expires the refresh token after **7 days** and the mount stops working every week. Published but *unverified* is fine for personal use: at consent Google shows “Google hasn’t verified this app” — click **Advanced** → **Go to ... (unsafe)**.

One client per host means a host can be revoked or rotated without touching the other.

Subcommands

Command	When
<code>rclone-gdrive setup</code>	First time on a host. Checks the tools, generates the keyring password if there is none (never shown), encrypts an existing plaintext config, asks for the OAuth client and opens the browser for consent, then enables and starts the unit and checks the mount. Idempotent: on a finished host it just reports everything is fine.
<code>rclone-gdrive status</code>	Anytime. Read-only: tools, keyring password (yes/no), config encrypted and decryptable, gdrive: present, token works (one Drive API call, on a throwaway copy of the config), unit enabled + active, mount up. Exit 1 if anything is wrong; never prints a secret.
<code>rclone-gdrive reauth</code>	The token expired or was revoked (status says so, the mount errors). Re-consent in the browser.
<code>rclone-gdrive rotate-client</code>	New OAuth client (the old one leaked, or you're cleaning up). Asks for the new ID + secret and re-consents (the token is bound to the client), then reminds you to delete the old client in the console.
<code>rclone-gdrive rotate-password</code>	New keyring password for the config encryption. Also resolves a rotation that was interrupted.

At consent rclone asks two questions: answer **y** to “Use web browser” and **n** to “Shared Drive”. Every command that rewrites the config stops the unit first (the running mount may rewrite the token) and starts it again afterwards. If reauth or rotate-client fails, or you press Ctrl-C at the browser step, the previous config (old client + token) is put back.

`--config PATH` (or `$RCLONE_CONFIG`) points it at another config, e.g. a throwaway one for testing. The unit always mounts the default config.

How rotate-password avoids a lockout

At every step, one of the passwords in the keyring decrypts one config on disk:

1. Back up the config (still on the old password) to `rclone.conf.rotate-backup`.
2. Generate the new password into a **staging** keyring entry (key `config_password_next`), next to the current one.

3. Re-encrypt straight from the old password to the new one (`rclone config encryption set` with a password command that returns each in turn), so no plaintext config is ever written. Check that the new one decrypts it and `gdrive:` is still there.
4. Promote: copy the staged password over `config_password`, check again, then delete the staging entry and the backup.

If a step fails, or the machine dies halfway, the next `rotate-password` (or `setup`) sees the leftovers and works out what to do from which password decrypts what: finish the promotion, restore the backup, or just clean up. If nothing decrypts, it stops and deletes nothing. `rclone`'s exit code can't be trusted here (`encryption set` exits 0 even when it fails), so every step is checked with `rclone config encryption check`.

Where it lives

Thing	Path
Helper scripts	<code>~/.local/bin/{localsend-here, ssh-here, add-rsync-key}</code>
System setup	<code>~/.local/bin/{setup-file-transfer, harden-file-transfer-ssh}</code>
Menu entries	niri power menu (rodii-power-menu, Tools → Sharing)
Doc templates	<code>~/.local/share/file-transfer/*.md</code> → rendered into <code>~/Downloads/Transfers/</code>
Packages	file-transfer category in <code>setup/packages.yaml</code>
Recipes	<code>just file-transfer</code> , <code>just file-transfer-harden</code> , <code>just gdrive <subcommand></code>
Google Drive	<code>~/.local/bin/rclone-gdrive</code> , <code>~/.config/systemd/user/rclone-gdrive.service</code>