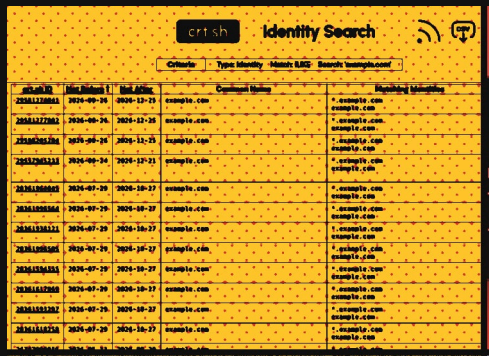
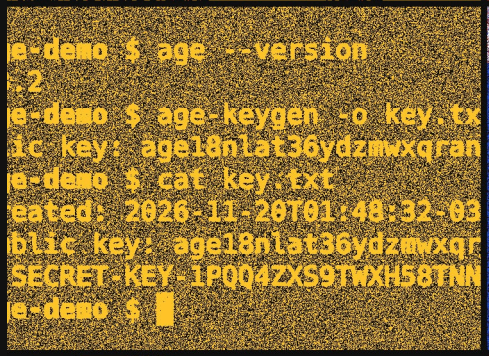
~PABLO

Nº 03

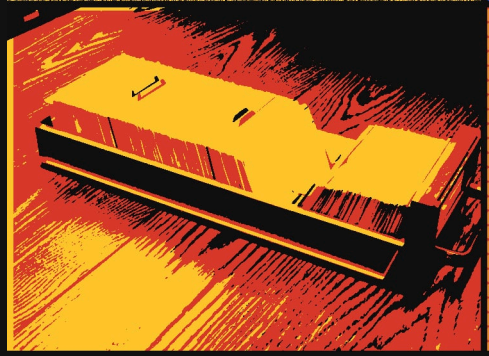
Essays from pablomurad.com, set on paper.



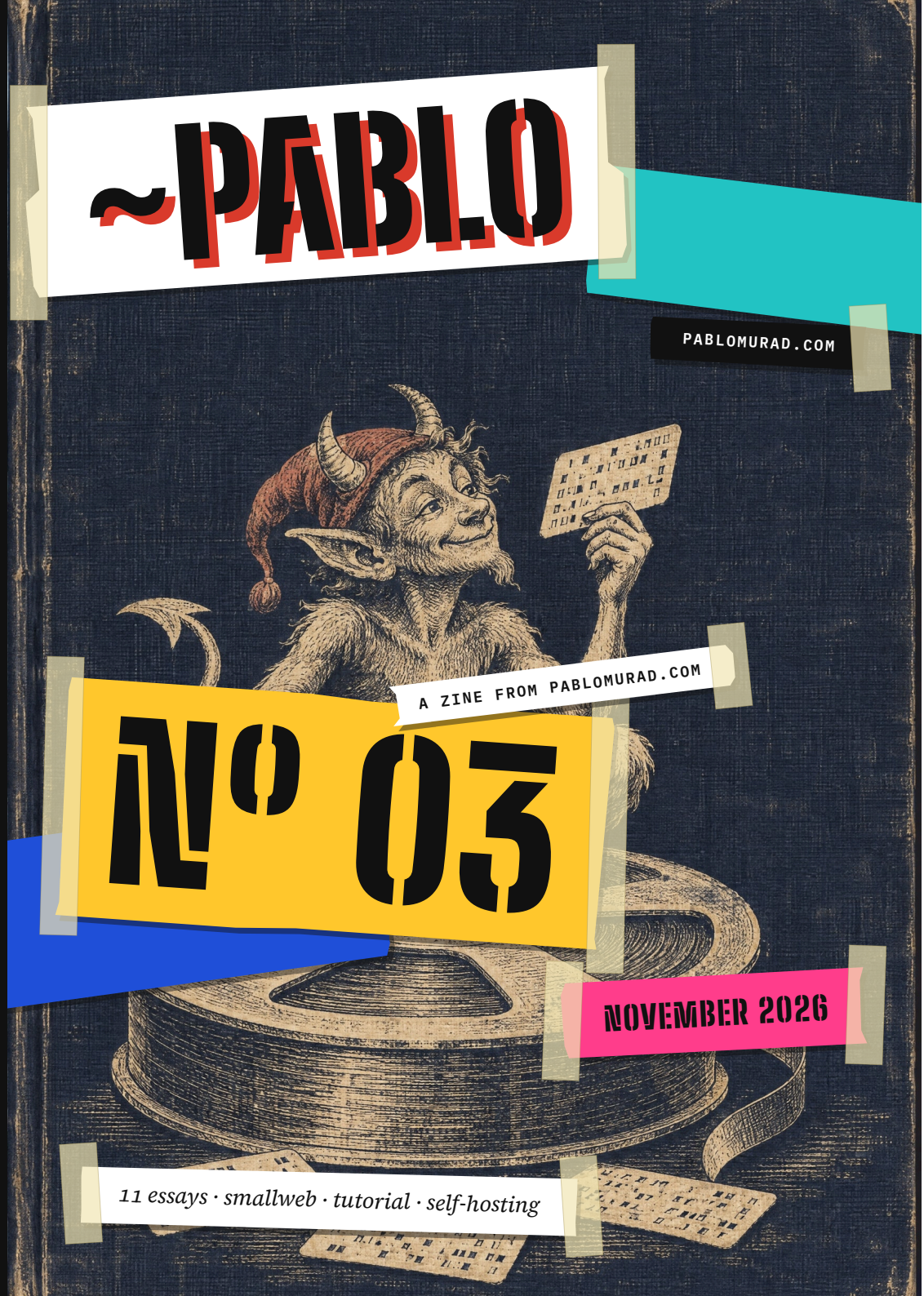












CONTENTS

Nº 03
NOVEMBER 2026

01 QT #1 - Keeping a subdomain off the scanners' radar — 3
QUICK-TIPS · DNS · TLS

02 HTTP/3: what it fixes, what it costs, and why I keep it off — 6
HTTP3 · QUIC · WEB-PERFORMANCE

03 Crypto I: age in the age of cryptography — 12
CRYPTOGRAPHY · AGE · ENCRYPTION

04 Come make a home on runv.club — 19
RUNV · RUNV-CLUB · TILDE

05 Linked notes in Emacs: Denote, org-roam and org-node — 24
EMACS · ORG-MODE · NOTES

06 A blog worth losing time on: lyra.horse — 36
RECOMMENDATIONS · INFOSEC · CSS

07 How finger and talk work at hi@pablomurad.com — 40
FINGER · TALK · MISFIN

08 Magit in depth: Git from inside Emacs — 49
EMACS · MAGIT · GIT

09 MultiPass: my links page fits in an 88x31 button — 62
OCTOTHORPES · MULTIPASS · INDIEWEB

10 The Observatorium sky got bigger — 67
OBSERVATORIUM · SMALL-WEB · SELF-HOSTING

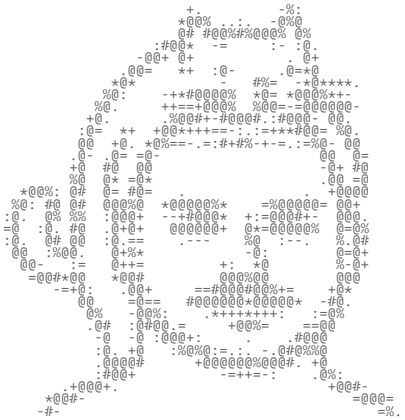
11 Hidden layers: the data riding inside ordinary files — 70
SMALLWEB · INDIEWEB · TUTORIAL

Every essay is on the website, with its links.

~pablo

A zine made of what I wrote on pablomurad.com.

03



COLOPHON

ISSUE	Nº 03, November 2026
PUBLISHED	23 November 2026, on pablomurad.com
TYPESET	23 November 2026, A5
ESSAYS	11, each one also on the website
PLATES	Pictures between the essays are courtesy of noctem.art.
AT HOME	Print pablo-03-booklet.pdf double-sided on A4, flip on the short edge, fold the stack in half and staple the spine.

pablomurad.com/zine/03



QT #1 - Keeping a subdomain off the scanners' radar

A Quick Tip on hiding a subdomain: names leak through certificate logs, DNS brute force and IP scans. A wildcard certificate, wildcard DNS, an nginx catch-all and a name nobody guesses make a private one much harder to find.

crt.sh ID	Not Before ↑	Not After	Common Name	Matching Identities
29581274041	2026-09-26	2026-12-25	example.com	*.example.com example.com
29581277802	2026-09-26	2026-12-25	example.com	*.example.com example.com
29598205284	2026-09-26	2026-12-25	example.com	*.example.com example.com
29557945233	2026-09-24	2026-12-21	example.com	*.example.com example.com
28361964045	2026-07-29	2026-10-27	example.com	*.example.com example.com
28361996564	2026-07-29	2026-10-27	example.com	*.example.com example.com
28361938121	2026-07-29	2026-10-27	example.com	*.example.com example.com
28361996505	2026-07-29	2026-10-27	example.com	*.example.com example.com
28361594355	2026-07-29	2026-10-27	example.com	*.example.com example.com

This is the first Quick Tip. The series is for the small stuff from my day to day: things I test, tweaks I end up keeping, attempts and experiments that don't need a whole post. Short on purpose, the kind of thing that reads in a couple of minutes.

The first one came from a private service I keep on a subdomain. I wanted it harder to find, and the first thing I learned was that its name was already public, with no way to take it back.

How a subdomain gets found

- **Certificate logs.** Every certificate a public CA issues goes into Certificate Transparency logs,[1] and anyone can search them on crt.sh. The name lands there the day the certificate exists, and it never leaves.

1 Certificate Transparency came out of Google after the 2011 DigiNotar breach, when a hacked CA issued fake certificates for Google domains. Since 2018, Chrome only trusts new certificates that were logged, so every public CA logs everything.

- **DNS brute force.** Tools try long lists of common names: admin, dev, vault, secrets. If the name is on the list, it gets found.
- **Scanning by IP.** Someone connects straight to the server's address and reads the certificate it hands back, names included.
- **Search engines.** Only if a page ever gets indexed.
- **Passive DNS.** Companies collect the DNS answers that pass through resolvers and sell the history.[2] Nothing to do about this one.

1. A wildcard certificate

With a certificate for `*.example.org`, the logs show only the wildcard, and new subdomains stop showing up there. The cover of this post is crt.sh's page for example.com, which does exactly this: every entry is `*.example.com`.

Wildcards need the DNS-01 challenge, so the ACME client needs API access to the DNS. On Porkbun, with acme.sh:

1. Create an API key at porkbun.com/account/api. It comes as a key and a secret.
2. Turn on API Access in the domain's settings. Porkbun makes each domain opt in.
3. Then:

```
export PORKBUN_API_KEY="pk1_..."
export PORKBUN_SECRET_API_KEY="sk1_..."
acme.sh --issue --dns dns_porkbun -d example.org -d '*.example.org'
```

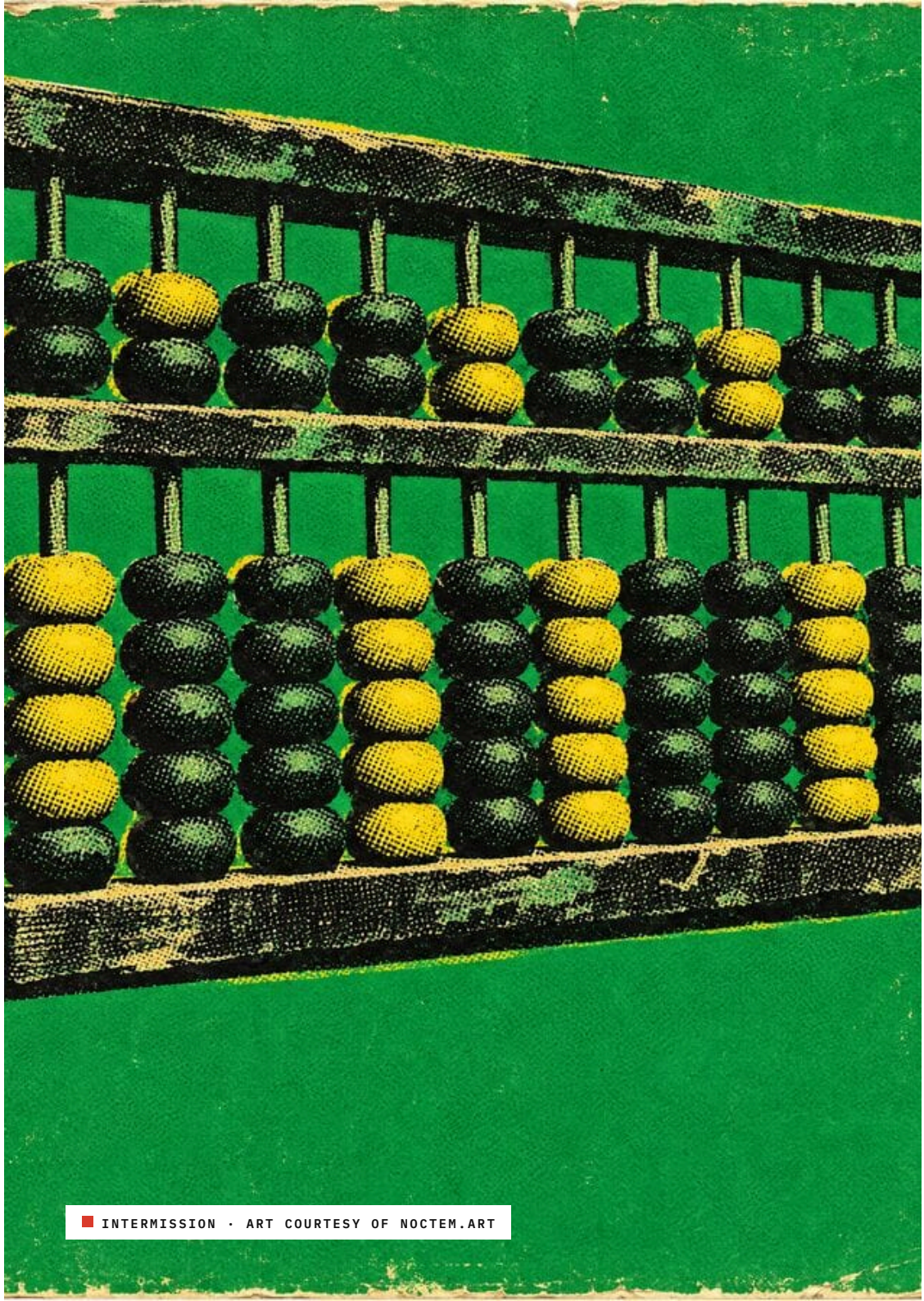
Any other DNS provider works the same way, only the plugin name and the variables change. acme.sh and lego know well over a hundred providers, and Certbot has plugins for the big ones, like `certbot-dns-cloudflare`. Whatever the provider, a key that can edit the whole zone is a powerful secret, so it's worth limiting it to that one domain when the provider allows.

2. Wildcard DNS

Point `*.example.org` at the server. Every invented name now resolves to the same address, and DNS brute force can't tell what exists from what doesn't.

Then nginx needs a catch-all that gives nothing back to names it doesn't know:

2 SecurityTrails is one of the best known sellers of this history. Their records go back years, so a name that ever resolved publicly can show up there even after it's gone.



INTERMISSION · ART COURTESY OF NOCTEM.ART

Photos can carry their own journal in XMP, with a caption, a place and a note, and a small script could turn a folder of them into a photo diary page.

DNS could announce the other doors of a domain, with a TXT record listing its Gemini, Gopher or Finger addresses and a tiny command-line client to read it. That would be a private convention, though, and no existing software would know about it.

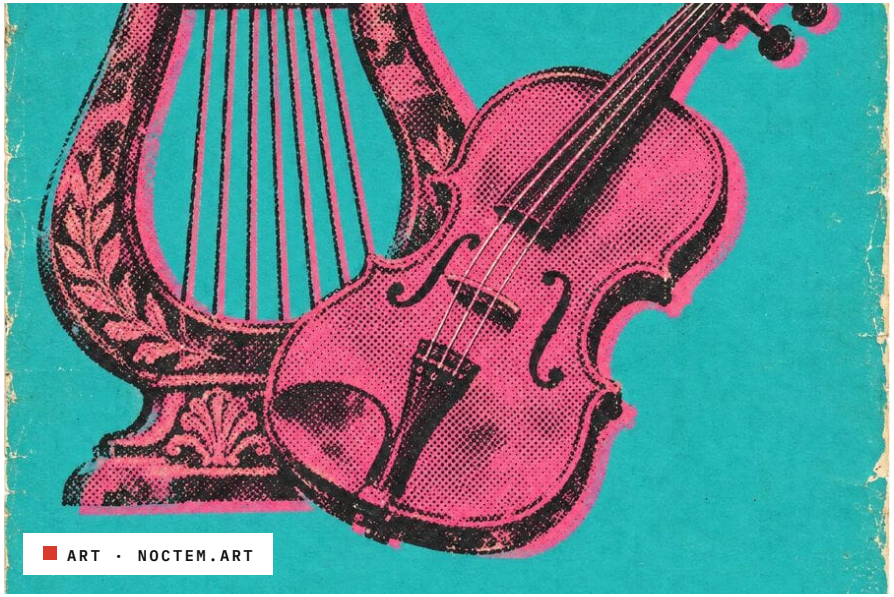
The most ambitious is an image cartridge in the PICO-8 style: a PNG carrying a text adventure or a small HTML page, opened by a reader that runs it in an isolated sandbox.

Before trusting any of this

Hidden doesn't mean secret. Anyone with a hex editor reads metadata, so anything private needs encryption on top.

It's also fragile. Messaging apps and image hosts often re-encode uploads and drop chunks, EXIF and comments along the way, and pixel tricks don't survive a resize. These layers travel best as plain files, sent by email or downloaded from a site.

And everything a reader finds inside a file came from a stranger. A careful reader checks sizes, parses the content as data and never runs code from it outside a sandbox. ■



■ ART · NOCTEM.ART


```
~/lab $ dig +short secrets.example.org
192.0.2.10
~/lab $ dig +short admin.example.org
192.0.2.10
~/lab $ dig +short qz81-nothing-here.example.org
192.0.2.10
~/lab $
```

```
~/lab $ curl https://192.0.2.10
curl: (35) OpenSSL/3.0.13: error:0A000458:SSL routines::tlsv1 unrecognized name
~/lab $ curl https://secrets.example.org
curl: (35) OpenSSL/3.0.13: error:0A000458:SSL routines::tlsv1 unrecognized name
~/lab $ curl http://secrets.example.org
curl: (52) Empty reply from server
~/lab $ curl https://lantern-9f2c.example.org
you found it
~/lab $
```

NGINX

```
server {
listen 80 default_server;
listen 443 ssl default_server;
server_name _;
ssl_reject_handshake on;
return 444;
}
```

`ssl_reject_handshake` refuses TLS to anyone who doesn't ask for a known name, so connecting by IP reveals nothing. `return 444` closes the connection without an answer.[3] Add `add_header X-Robots-Tag "noindex" always;` to the real site and search engines stay away too.

3. A name nobody guesses

Wildcard DNS beats DNS brute force, but a scanner that also tries HTTP still gets an answer from the real name, so a name on its wordlist gets found anyway. The real service needs something no list has, like `lantern-9f2c` instead of `secrets`.

And if the old name is already in the logs, renaming is the only fix: a wildcard only protects the names that come after it. Anything that points to the old name, like a login callback, has to move along with it.

3 444 isn't a real HTTP status. nginx uses it internally to mean "close the connection and send nothing", which the client sees as an empty reply. `ssl_reject_handshake` arrived in nginx 1.19.4, in 2020.

Keep in mind

This is a layer of obscurity on top of the real protection. The service still needs a login and updates. What a hidden name buys is fewer bots knocking. ■

02 HTTP3 · QUIC · WEB-PERFORMANCE 21 NOVEMBER 2026

HTTP/3: what it fixes, what it costs, and why I keep it off

HTTP/3 explained without hype: what QUIC changes, the problem it fixes, where the gains are real, what it costs the server in CPU, and why I keep it switched off on my own servers.



HTTP/3 has been a standard since 2022, every major browser speaks it, and a good part of the web already serves it. It's usually presented as a free upgrade: flip a switch, get a faster site. That's partly true, and the part that gets less attention is what it costs the server.

This post explains what HTTP/3 is, what problem it solves, how much it really helps, where it hurts, and why I haven't turned it on for my own servers.


```HTML
<svg xmlns="http://www.w3.org/2000/svg" width="176" height="62" viewBox="0 0 88 31">
<metadata>{"format":"indiecard","version":1,"website":"https://pablomurad.com","feed":"https://pablomurad.com/feed"}</metadata>
<rect width="88" height="31" fill="#111111"/>
<rect x="2" y="2" width="84" height="27" fill="none" stroke="#d93a2b" stroke-width="1"/>
<text x="44" y="19" fill="#ece8de" font-family="monospace" font-size="9" text-anchor="middle">pablomurad</text>
</svg>
```

A domain works like a file too

DNS TXT records hold free text attached to a name, and a few conventions already use them to say who owns a domain. My Bluesky handle is pablomurad.com because a TXT record at `_atproto.pablomurad.com` points to my account's DID, and a record on the domain itself carries the fingerprint of my OpenPGP key for Keyoxide. Any DNS client can read them. This query goes over HTTPS:

```CONSOLE
$ curl -s -H 'accept: application/dns-json' 'https://cloudflare-dns.com/dns-query?name=_atproto.pablomurad.com&type=TXT' | jq -r '.Answer[].data'
"did=did:plc:acmqj4gvd637wnybyk7nkogq"
$ curl -s -H 'accept: application/dns-json' 'https://cloudflare-dns.com/dns-query?name=pablomurad.com&type=TXT' | jq -r '.Answer[].data' | grep openpgp
"openpgp4fpr:7C3A051F5EA99005FC1378824E7F9C514F1DAC87"
```

On a machine with dig, `dig +short TXT _atproto.pablomurad.com` asks the same question.

Things worth building

These formats are old and supported everywhere, which makes them a good base for small conventions on the personal web. A few ideas, from easy to ambitious.

An indiecard reader is the easiest place to start: a page where someone drops a PNG and sees the card, with all the parsing done by JavaScript in the browser and no server behind it. With a reader like that, an 88x31 button could carry a site's address, feed and fediverse handle, and a directory could grow by dropping buttons on it instead of filling in a form for each site.

Zines fit PDF well. Readers get the finished pages, and the attachments carry the Markdown, the images and the license for anyone who wants to remix it.

```
CONSOLE
$ pdfdetach -list zine.pdf
2 embedded files
1: pablo.png
2: zine.md
$ pdfdetach -save 2 -o out.md zine.pdf && cat out.md
# Zine #1

The original Markdown of this zine.
```

Electronic invoices in France and Germany use this every day. A Factur-X invoice (ZUGFeRD on the German side) is a PDF/A-3 file that a person reads like any invoice, and inside it sits `factur-x.xml`, the same invoice as structured data that accounting software imports without retyping anything.

ZIP has a comment field

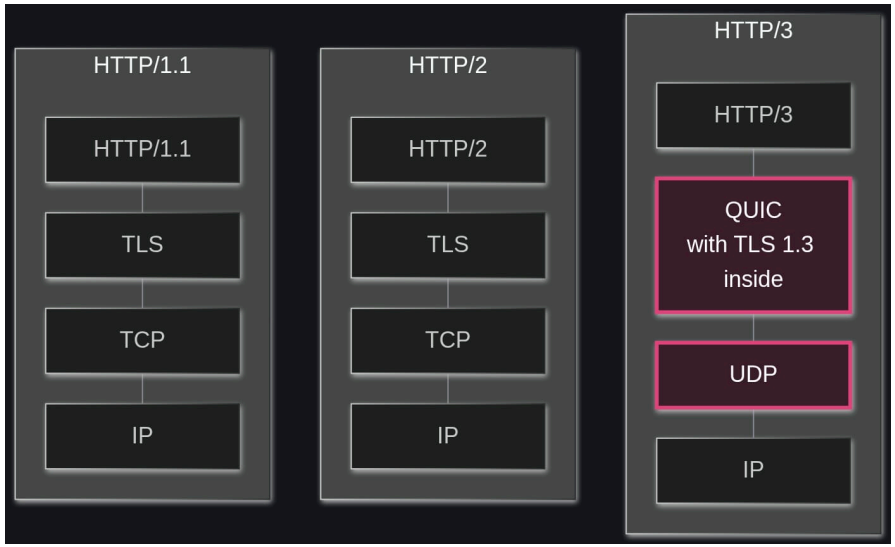
The end of every ZIP file has room for a comment of up to 65,535 bytes, and `unzip -z` prints it:

```
CONSOLE
$ zip -q bundle.zip zine.md pablo.png
$ echo "indiecard bundle: open pablo.png for the profile" | zip -z
bundle.zip
enter new zip file comment (end with .):
$ unzip -z bundle.zip
Archive: bundle.zip
indiecard bundle: open pablo.png for the profile
```

The bigger trick with ZIP is what goes inside. DOCX, XLSX, EPUB and JAR files are all ZIP archives with an agreed set of files in them. An EPUB starts with a file called `mimetype` that contains `application/epub+zip`, followed by a manifest that describes the rest. Renaming a .docx to .zip and opening it shows the XML underneath.

SVG has a metadata element

SVG is XML, and it has a `<metadata>` element meant for exactly this. Browsers draw nothing from it. The badge below is an SVG rendered by the site, and the Source under it shows the JSON riding along:



A short family history

HTTP is the language browsers and servers use to ask for and send pages. It has gone through three big versions, and each one fixed the bottleneck of the one before.

HTTP/1.1, from the late nineties, handles one request at a time per connection. To load a page with forty files, browsers open several connections in parallel and queue the rest. It works, but it's a lot of waiting in line.

HTTP/2, standardized in 2015, puts many requests on a single connection at the same time. The forty files travel together, interleaved, over one TCP connection. Much better, except for one problem hiding underneath.

HTTP/3, standardized in 2022 as RFC 9114, keeps the ideas of HTTP/2 but moves them onto a new transport called QUIC, defined in RFC 9000 in 2021. QUIC doesn't run on TCP. It runs on UDP.

The problem it fixes

TCP delivers bytes in order, always. If one packet gets lost, everything after it waits until that packet is sent again, even data that already arrived.

With HTTP/2, all of a page's files share one TCP connection. So a single lost packet from one image freezes the CSS, the JavaScript and every

other file on that connection until the missing piece shows up. This is called head-of-line blocking. Picture a one-lane road: a car stalls, and every car behind it stops, even the ones going somewhere else.

On a good wired connection, packets rarely get lost, and the difference is hard to notice. On a phone moving between cell towers, or on a busy Wi-Fi, losses are common, and HTTP/2 can end up slower than its reputation suggests.

What QUIC does differently

QUIC rebuilds the reliable parts of TCP itself, on top of UDP, and changes a few things on the way:

- Independent streams. Each file gets its own stream, and a lost packet only stalls the stream it belongs to. Back to the road picture: every car now has its own lane.
- Encryption built in. QUIC always uses TLS 1.3. There's no unencrypted QUIC.
- A faster start. With TCP and TLS 1.3, the browser needs two round trips before it can send the first request: one for TCP, one for TLS. QUIC does both in one. When the browser has visited the site before, it can even send the request right away, in what's called 0-RTT.
- Connections that survive a network change. A QUIC connection is identified by an ID, not by the IP address and port. When a phone jumps from Wi-Fi to mobile data, the connection can keep going instead of starting over.
- Less visible to the middle. Most of QUIC's own headers are encrypted too, so routers and firewalls along the way can't read or tamper with them.

How a browser finds it

A browser doesn't try HTTP/3 out of nowhere. The first visit usually happens over HTTP/2, and the server answers with a header that says "I also speak HTTP/3 on this port":

```
Alt-Svc: h3=":443"; ma=86400
```

The browser remembers that for the time in `ma` (in seconds) and tries QUIC on the next requests. A DNS record of type HTTPS can announce the same thing earlier, before the first connection. If UDP is blocked anywhere along the way, the browser quietly falls back to HTTP/2, and the visitor never notices.

Newer images can also carry Content Credentials from C2PA: a signed record of where the image came from and how it was edited. In a JPEG it goes into APP11 segments, and in a PNG into a chunk called `caBX`. Anyone can rewrite an EXIF field, while a C2PA record carries a signature, so a change to it shows.

MP3 and ID3 frames

ID3v2 tags sit at the start of an MP3 and are made of frames: TIT2 for the title, TPE1 for the artist and dozens more. TXXX is the open one, a frame with a description and a value for anything the list doesn't cover. Here it goes on a three-second beep made with ffmpeg, using the mutagen library:

PYTHON

```
from mutagen.id3 import ID3, TIT2, TPE1, TXXX
tags = ID3()
tags.add(TIT2(encoding=3, text="A 440 Hz beep"))
tags.add(TPE1(encoding=3, text="Pablo Murad"))
tags.add(TXXX(encoding=3, desc="liner_notes", text="Made for a blog post
about hidden layers."))
tags.add(TXXX(encoding=3, desc="website", text="https://pablomurad.com"))
tags.save("beep.mp3")
```

CONSOLE

```
$ ffprobe -v quiet -show_entries format_tags -of default=nw=1 beep.mp3
TAG:title=A 440 Hz beep
TAG:artist=Pablo Murad
TAG:website=https://pablomurad.com
TAG:liner_notes=Made for a blog post about hidden layers.
```

A music player shows the title and the artist. The liner notes and the link travel along until some program asks for them.

PDF can carry its own sources

A PDF can embed real files, and readers such as Acrobat list them in an attachments panel. With pypdf, a one-page A5 zine takes its Markdown source and the skull along:

PYTHON

```
from pypdf import PdfWriter
pdf = PdfWriter()
pdf.add_blank_page(width=420, height=595) # A5, in points
pdf.add_attachment("zine.md", open("zine.md", "rb").read())
pdf.add_attachment("pablo.png", open("pablo.png", "rb").read())
pdf.write("zine.pdf")
```

```
CONSOLE
$ python3 p8.py celeste.p8.png 132
b':c:\x00'
-- ~celeste~
-- matt thorson + noel berry

-- globals --
-------------

room = { x=0, y=0 }
objects = {}
types = {}
freeze=0
shake=0
```

The full run gives back 26,589 characters of Lua, squeezed into the 15,616 bytes a cartridge reserves for code.[3] All of it is gone if the image goes through a JPEG conversion or a resize, because the bits that matter are exactly the ones lossy compression throws away.

GIF keeps comments

GIF89a has a Comment Extension, a block that starts with the bytes `0x21 0xFE` and holds plain text. Viewers ignore it. MultiPass puts its JSON query there, and the MultiPass post shows how to pull it out with `strings`.

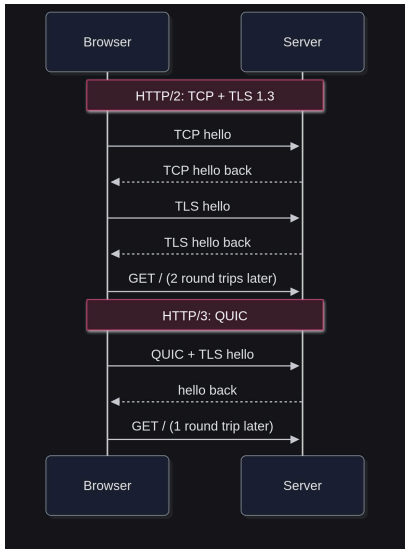
JPEG and EXIF

A JPEG photo usually carries EXIF: camera, lens, settings, date and, when the camera app has location access, GPS coordinates. The test photo here has invented data, with coordinates that point at Christ the Redeemer in Rio:

```
CONSOLE
$ exiftool -Make -Model -DateTimeOriginal -GPSPosition photo.jpg
Make : ExampleCam
Camera Model Name : X1
Date/Time Original : 2026:11:23 17:00:00
GPS Position : 22 deg 57' 6.84" S, 43 deg 12' 37.80" W
$ exiftool -all= photo.jpg
1 image files updated
$ exiftool -GPSPosition photo.jpg
```

The last command prints nothing. `-all=` removes every metadata block exiftool knows how to write and keeps a copy of the untouched file next to it, as `photo.jpg_original`. Some sites strip EXIF on upload and others keep it, so the safe habit is to strip it before the file leaves the computer.

3 The credits inside the code say Matt Thorson, the name Maddy Thorson used when the cart came out in 2015.



How much does it actually help

The gains are real, and they depend a lot on the network:

- On phones, slow links and networks that lose packets, HTTP/3 helps the most. Head-of-line blocking hurts there, and the faster handshake saves a round trip that can be long.
- When someone changes networks in the middle of something, a download or a video, connection migration avoids a restart.
- On a fast, stable wired connection, the difference is usually small, and it can even go the other way (more on that below).

It's widely deployed. W3Techs counts 41.1% of websites offering HTTP/3 in November 2026, and Cloudflare reported that 21% of the requests it served in 2025 used it.

The downsides

CPU

This is the big one. TCP lives in the operating system kernel, and the kernel and network cards have decades of tricks to take work off the processor: they cut, merge and checksum TCP packets in bulk, much of it in hardware. QUIC lives in the application, in user space, and most of those tricks don't apply to it yet. Every packet gets more attention from the CPU, and there are more packets.

The numbers are not small. Google engineers told a 2020 SIGCOMM workshop that just sending UDP took about a quarter of the CPU in their workload, and more than half in some environments. A 2024 study presented at the ACM Web Conference, QUIC is not Quick Enough over Fast Internet, found that on fast connections the QUIC and HTTP/3 stack moved data up to 45.2% slower than TCP with HTTP/2, and traced it to the extra processing on the receiving side. In their tests, Chrome over QUIC reached nearly 100% CPU at around 200 Mbps, while HTTP/2 went past 900 Mbps on the same machine.

There are fixes in progress, like segmentation offload for UDP (GSO),[1] and big providers tune their stacks heavily. But out of the box, on an ordinary server, QUIC costs more processor per byte than what it replaces. In nginx, for example, GSO for QUIC is off by default.

TWO STACKS INSTEAD OF ONE

UDP is blocked or throttled on some corporate networks, hotel Wi-Fi and older firewalls. So nobody can serve HTTP/3 alone: HTTP/2 over TCP keeps running for everyone who can't use QUIC, and HTTP/3 goes next to it. That's more to configure, more to monitor and one more port to open, UDP 443.

HARDER TO LOOK INSIDE

Because QUIC encrypts almost everything, tools like `tcpdump` show much less than they do with TCP. Debugging a slow connection means exporting the TLS keys from the browser or the server and decrypting the capture by hand.

STILL MATURING ON SOME SERVERS

nginx added HTTP/3 in version 1.25.0, in 2023, and its documentation still describes the module as experimental. Caddy turns it on by default, and big CDNs have run it for years, but for a self-hosted server the path is less trodden than HTTP/2.

0-RTT HAS A CATCH

The instant start comes with a risk: a request sent in 0-RTT can be captured and replayed by an attacker. Servers have to limit it to safe requests, like simple page loads, and never use it for things like payments or form submissions.

1 GSO, generic segmentation offload, lets a program hand the kernel one large buffer and have it cut into packets in a single pass, sometimes by the network card itself. TCP has had it for years. Linux added it for UDP in kernel 4.18, in 2018.



more zero bytes.[2] From there, each byte is a character from a table of 59 symbols, or a zero followed by a literal character, or the first of two bytes that copy a piece of text already decoded. The file format page on the PICO-8 wiki has the details, and the whole decoder fits in a short script:

```python
import sys
from PIL import Image
px = Image.open(sys.argv[1]).convert("RGBA").tobytes()
# one byte per pixel: the two lowest bits of A, R, G and B
data = bytes((px[i+3] & 3) << 6 | (px[i] & 3) << 4 | (px[i+1] & 3) << 2 |
(px[i+2] & 3) for i in range(0, len(px), 4))
code = data[0x4300:0x8000]
print(code[:4])
lut = "\n 0123456789abcdefghijklmnopqrstuvwxyz!#%(){}[]<>+=/*:;.,~_"
size, out, i = code[4] << 8 | code[5], "", 8
while len(out) < size:
 c = code[i]
 if c == 0: out += chr(code[i+1]); i += 2
 elif c < 0x3c: out += lut[c-1]; i += 1
 else: n = code[i+1]; off = (c-0x3c)*16 + (n & 15); out += (out[-off:] * 17)
[:(n >> 4) + 2]; i += 2
print(out[:int(sys.argv[2])])
```

Asking for the first 132 characters:

2 PICO-8 0.2.0 introduced a newer compression, marked by a zero byte followed by `pxa`. This script only reads the old format, which is the one the Celeste cart uses.

```
$ exiftool -Indiecard pablo.png
Indiecard : {"format": "indiecard", "version": 1, "name": "Pablo Murad",
"website": "https://pablomurad.com", "feed": "https://pablomurad.com/feed",
"fediverse": "@pablo@murad.social"}
```
CONSOLE

EXCALIDRAW DOES IT FOR REAL

Excalidraw, the open source whiteboard, uses the same chunk trick. With “Embed scene” turned on, its PNG export gains a tEXt chunk with the keyword `application/vnd.excalidraw+json` and the whole drawing, compressed, as the value. Dropping that PNG back into Excalidraw brings every shape, arrow and label back, ready for editing, so one file works as the picture and as the source. The SVG export keeps the scene in a `<metadata>` element, between markers:

```
<metadata>
<!-- payload-type:application/vnd.excalidraw+json -->
<!-- payload-version:2 -->
<!-- payload-start -->(the scene, in base64)<!-- payload-end -->
</metadata>
```
XML

PICO-8 hides a game in the pixels

PICO-8 is a fantasy console: a made-up little computer with 16 colors, a 128x128 screen and tight limits on code size. Its games can be saved as `.p8.png` files, which look like a cartridge with a label on it.

The image is 160 by 205 pixels, and every pixel stores one byte of the cartridge. The two lowest bits of alpha, red, green and blue, in that order, add up to eight bits. A change of 3 at most out of 255 in each channel disappears into the label art. 160 times 205 gives 32,800 bytes, laid out like this:

```
digraph cart {
node [shape=record, fontname="Helvetica", fontsize=12, margin="0.4,0.12",
style=filled, fillcolor="#ffffff", color="#111111"];
cart [label="{0x0000 to 0x42FF|sprites, map, flags,\nmusic and sound}|
{0x4300 to 0x7FFF|Lua code}|{0x8000|version}"];
}
```
DOT

The original Celeste started as one of these carts, made in four days by Maddy Thorson and Noel Berry, and it’s still on the Lexaloffle forum. This is the cartridge. The page shows a converted copy, and the original PNG is the one to feed the decoder:

The code area is compressed. Old carts mark it with `:c:` and a zero byte, followed by the size of the decompressed code in two bytes and then two

Trying it on nginx

To see it working, this is the minimal nginx setup, with nginx 1.25.1 or newer built with HTTP/3 support:

```
server {
    listen 443 ssl;
    listen 443 quic reuseport;
    http2 on;

    ssl_certificate /etc/letsencrypt/live/example.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.org/privkey.pem;

    add_header Alt-Svc 'h3=":443"; ma=86400';
}
```

Open UDP port 443 on the firewall, reload, and visit the site twice. In the browser’s developer tools, the Network tab has a Protocol column, and it should say `h3` from the second visit on. With a `curl` built with HTTP/3, `curl -I --http3 https://example.org` tests it from the terminal.

Why I haven’t turned it on

This part is opinion, and it’s mine.

I’m excessively methodical about anything that puts load on the CPU. If a server’s load reaches 0.1,[2] I’m already thinking about which processes to cut, without much worry. I value extreme optimization and a light server, even when that costs the user something.

HTTP/3 goes in the opposite direction. It asks the server to do more work for each byte, so the visitor gets a faster start, mostly on a bad mobile connection. For a CDN serving millions of phones, that trade makes sense. With my priorities, it doesn’t, so HTTP/3 stays off on my servers. ■



2 Load average counts the processes running or waiting for the CPU, averaged over the last one, five and fifteen minutes. On a machine with several cores, 0.1 is close to idle.

03 CRYPTOGRAPHY · AGE · ENCRYPTION 21 NOVEMBER 2026

Crypto I: age in the age of cryptography

First part of a series on everyday cryptography: age, the small tool that encrypts files with one-line keys. Installing on Windows and Linux, keys, passphrases, SSH keys, post-quantum keys, real terminal output, and a plot twist that keeps the age key inside KeePassXC.

```
~/age-demo $ age --version
v1.3.2
~/age-demo $ age-keygen -o key.txt
Public key: age18nlat36ydzmwxqran9a3wkqua688uxuqtz0kva7tgfyxal7ae5aqfscmx8
~/age-demo $ cat key.txt
# created: 2026-11-20T01:48:32-03:00
# public key: age18nlat36ydzmwxqran9a3wkqua688uxuqtz0kva7tgfyxal7ae5aqfscmx8
AGE-SECRET-KEY-1PQQ4ZXS9TWXH58TNNN3L9E27WJPA8GE3AEHCL6HTM9GZG5X94SGSF5M3QJ
~/age-demo $
```

This is the first part of a small series about cryptography, the everyday kind: protecting a file, a backup, a note with a password in it. And the first tool on the list is the one I reach for when I just want a file encrypted and nothing else: age.

The name is written in lowercase, like a command, and it does one job. It takes a file and makes it unreadable to everyone except the people (or keys) it was meant for. No keyring, no config file, no fifty options to get wrong.

Everything below was run for real, so the screenshots are actual terminal output, not mockups.

The idea in one picture

age works with a pair of keys:

- the **recipient**, a public key that starts with `age1`. It can be shared with anyone. It only knows how to lock.
- the **identity**, a secret key that starts with `AGE-SECRET-KEY-1`. It stays private. It's the only thing that unlocks.



```
Image.open("avatar.png").save("pablo.png", pnginfo=meta)
```

Running the chunk script on the result shows the new chunk sitting between the header and the pixels:

CONSOLE

```
$ python3 chunks.py pablo.png
IHDR 13 bytes
iTXt 186 bytes
IDAT 869 bytes
IEND 0 bytes
```

The page shows a WebP copy of that image, and the conversion drops the chunk, so saving the picture straight from the page loses the card. The original PNG still has it, and this script prints it:

PYTHON

```
import json, sys
from PIL import Image
print(json.loads(Image.open(sys.argv[1]).text["indiecard"]))
```

CONSOLE

```
$ python3 readcard.py pablo-indiecard.png
{'format': 'indiecard', 'version': 1, 'name': 'Pablo Murad',
'website': 'https://pablomurad.com', 'feed': 'https://pablomurad.com/feed',
'fediverse': '@pablo@murad.social'}
```

exiftool reads it too, with no script at all:

```
ZIP,Archive comment and the files inside,"DOCX, EPUB and JAR"
SVG,The metadata element,Excalidraw scenes
DNS,TXT records,Bluesky handles and Keyoxide proofs
```

PNG is a list of chunks

A PNG file starts with an 8-byte signature and then a list of chunks. Each chunk is a length (4 bytes), a type made of four letters, the data and a CRC checksum. IHDR carries the size and color type, IDAT the compressed pixels, and IEND closes the file. The PNG specification also defines three chunks for text: tEXt for plain Latin-1, zTXt for compressed text and iTXt for UTF-8, with optional compression and a language tag.

Viewers skip any chunk they don't need, which makes the text chunks a comfortable home for structured data. A short Python script lists the chunks of any PNG:

PYTHON

```
import struct, sys
data = open(sys.argv[1], "rb").read()
pos = 8 # skip the 8-byte PNG signature
while pos < len(data):
 length, kind = struct.unpack(">I4s", data[pos:pos + 8])
 print(f"{kind.decode()} {length:>6} bytes")
 pos += 12 + length # length + type + data + CRC
```

A PROFILE CARD INSIDE AN AVATAR

The skull below is a 16x16 drawing scaled up to 256 pixels. Its PNG also carries a small JSON profile in an iTXt chunk, here called an indiecard.[1] Pillow writes it in a few lines:

PYTHON

```
import json
from PIL import Image
from PIL.PngImagePlugin import PngInfo

card = {
 "format": "indiecard",
 "version": 1,
 "name": "Pablo Murad",
 "website": "https://pablomurad.com",
 "feed": "https://pablomurad.com/feed",
 "fediverse": "@pablo@murad.social",
}

meta = PngInfo()
meta.add_itxt("indiecard", json.dumps(card))
```

1 "indiecard" is a name made up for this post. There's no standard behind it, only a keyword and a JSON object that a writer and a reader agree on.

The mental model I use is a padlock. I can hand out open padlocks to anyone who wants to send me something. They snap it shut on the box, and from then on only my key opens it. Not even the person who locked it can open it again.

There's a second mode, with a passphrase instead of keys, for when a file only needs to be locked and unlocked by the same person. More on that below.[1]

Installing it

ON WINDOWS

Any of the three package managers works:

```
winget install --id FiloSottile.age
scoop bucket add extras
scoop install age
choco install age.portable
```

After installing, open a new terminal window so the `age` command shows up in the PATH, and check it with `age --version`.

One trap deserves a warning before anything else. The old Windows PowerShell (version 5.1, the one that comes with Windows) mangles binary data that goes through `>` or `|`, and an encrypted file is binary data. The file gets written, looks fine, and fails to decrypt later. PowerShell 7.4 and newer fixed this, and `cmd.exe` never had the problem, but the simplest habit is to never redirect at all on Windows and always let age write the file itself with `-o`:[2]

```
age -r age1... -o notes.txt.age notes.txt
age -d -i key.txt -o notes.txt notes.txt.age
```

Every example in this post uses `-o` for that reason, so all of them work the same on Windows and Linux.

ON LINUX

Most distributions have it:

1 Under the hood, age uses X25519 for keys, ChaCha20-Poly1305 for the content and scrypt to turn a passphrase into a key. Every file gets a fresh random file key, and that file key is what gets locked for each recipient.

2 This is a known issue in age's tracker on GitHub. Microsoft changed PowerShell 7.4 to keep native command output as raw bytes when redirected to a file or piped to another program, which fixes it there.

```
sudo apt install age        # Debian 12+, Ubuntu 22.04+
sudo dnf install age        # Fedora
sudo pacman -S age          # Arch
```

The catch is that distribution packages tend to lag behind. Ubuntu 24.04, for example, still ships 1.1.1, while the current version is 1.3.2, the one that added post-quantum keys and a couple of handy tools. The release page on GitHub has ready binaries for Linux, Windows and macOS, and anyone with Go installed can build the latest with `go install filippo.io/age/cmd/...@latest`. The screenshots here use 1.3.2.

Making a key

```
age-keygen -o key.txt
```

That's it. The public key is printed on the screen, and the file holds both halves:

```
$ age-keygen -o key.txt
Public key: age18nlat36ydzmwxqran9a3wkqua688uxuqtz0kva7tgfyxal7ae5aqfscmx8
$ cat key.txt
# created: 2026-11-20T01:48:32-03:00
# public key: age18nlat36ydzmwxqran9a3wkqua688uxuqtz0kva7tgfyxal7ae5aqfscmx8
AGE-SECRET-KEY-1PQQ4ZXS9TWXH58TNNN3L9E27WJPA8GE3AEHCL6HTM9GZG5X94SGSF5M3QJ
```

(That key was made for this post and thrown away afterwards, so it's safe to show.)

The first two lines of `key.txt` are comments. The third is the secret. Anyone who gets that line can read everything ever encrypted to the matching `age1...` key, so it deserves the same care as a password. Later in this post it goes into KeePassXC, which is where I'd keep it.

To get the public key back from the file at any time:

```
age-keygen -y key.txt
```

Locking and unlocking

To encrypt, age needs the recipient (`-r`) and the file:

```
age -r age18nlat36ydzmwxqran9a3wkqua688uxuqtz0kva7tgfyxal7ae5aqfscmx8 -o
notes.txt.age notes.txt
```

To decrypt, it needs the identity (`-i`):

2000s and holds a whole Octothorpes query in a comment block (the MultiPass post has the details). That trick has a large family. Most formats in daily use have a spare room where extra data can sit, and a few projects keep entire programs there.

This post goes through that family one format at a time. Every example below ran for real, and the outputs come straight from the terminal, with long lines trimmed.

Two ways to hide data

The word "hidden" covers two different techniques, and they get mixed up a lot.

Metadata lives in space the format sets aside on purpose. A PNG can hold text chunks, a JPEG holds EXIF, an MP3 holds ID3 tags. Programs that don't care about those parts skip them, and the ones that do read them with ordinary libraries. There's nothing secret about it: a hex dump shows everything.

Steganography hides data inside the content itself. Pixel colors change by amounts too small for the eye to notice, and those small changes spell out bytes. The file gains no extra section, so only a reader that knows the trick gets anything out.

These are the places this post visits. The PICO-8 row is the only one that hides data in the content; the rest is metadata.

CSSV

```
---
/* cssv:title Where the extra data lives */
/* cssv:description Each format in this post, the part of the file that
carries the extra data, and a real use of it. */
table { width: 100%; font: 15px/1.45 source-serif-4, Georgia, serif; color:
#111; border-collapse: collapse; }
th { font: 600 11px/1.4 ibm-plex-mono, ui-monospace, monospace; text-
transform: uppercase; letter-spacing: 0.06em; color: #666; text-align: left;
padding: 0 0.7em 0.4em 0; border-bottom: 1px solid #111; }
td { padding: 0.45em 0.7em 0.45em 0; border-top: 1px solid #e6e6e6;
vertical-align: baseline; }
tbody tr:last-child td { border-bottom: 1px solid #111; }
td[data-col="format"] { font: 600 13px/1.5 ibm-plex-mono, ui-monospace,
monospace; color: #d93a2b; white-space: nowrap; }
tr[data-row="4"] td { background: #fdf0ee; }
---
format,where the data lives,seen in the wild
GIF,Comment Extension block,MultiPass buttons
PNG,"tEXt, zTXt and iTXt chunks",Excalidraw scenes
PNG,The two lowest bits of each pixel,PICO-8 cartridges
JPEG,EXIF and XMP segments,"Camera, date and GPS position"
JPEG,APP11 segments,C2PA Content Credentials
MP3,ID3v2 frames such as TXXX,Custom tags and lyrics
PDF,Embedded files,Factur-X and ZUGFeRD invoices
```

The collector also runs locked down. It needs root to read the nginx logs, the journal and Docker, but systemd boxes it in: the whole system is read-only except the database and the three folders it publishes to, and the only extra power it keeps is permission to read any file. The small server behind port 1900 runs with no privileges at all, as a throwaway user, with 64 MB of memory.

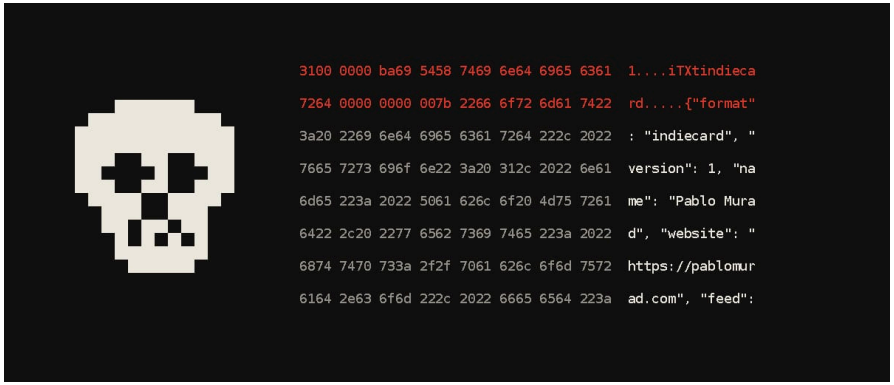
Go look at it after dark, São Paulo time, when the moon is up. ■

■ ■ ■

11 SMALLWEB · INDIEWEB · TUTORIAL 23 NOVEMBER 2026

Hidden layers: the data riding inside ordinary files

PNG chunks, PICO-8 cartridges, EXIF, ID3 tags, PDF attachments, ZIP comments, SVG metadata and DNS TXT records: everyday files carry a second layer most programs never show. A tour with real examples to run, and ideas for what to build on top.



A conversation about a webring sent me looking at files that carry other files inside them. The button at the bottom of my /links page already does it: an 88x31 GIF that looks like any button from the


```
~/age-demo $ echo "the wifi password is: correct-horse-battery" > notes.txt
~/age-demo $ age -r age18nlat36ydzmwxqran9a3wkqua688uxuqtz0kva7tgfyxal7ae5aqfscmx8 -o notes.
txt.age notes.txt
~/age-demo $ ls -l notes.*
-rw-r--r-- 1 root root  44 Nov 20 01:48 notes.txt
-rw-r--r-- 1 root root 244 Nov 20 01:48 notes.txt.age
~/age-demo $ head -c 180 notes.txt.age; echo
age-encryption.org/v1
-> X25519 RUrcjNNrgS9HtORGFpkAcBKpH9Dyl0pWusPgLIbTezk
ADVEIW++vXfx93gjX9RuXzkEFnsJ1x5z0CqnrHl4dpk
--- YlXso8aTdSFvxadu2vbwOtd6Vfp35hsOxtlc9bUiUGI
uL5WWhnx/
~/age-demo $ age -d -i key.txt notes.txt.age
the wifi password is: correct-horse-battery
~/age-demo $
```

```
~/age-demo $ age -p -o diary.txt.age notes.txt
Enter passphrase (leave empty to autogenerate a secure one): 
```

```
age -d -i key.txt -o notes.txt notes.txt.age
```

Here's a full round trip, plus a peek inside the encrypted file. The beginning of an age file is plain text on purpose: the line `age-encryption.org/v1` says what it is, the `-> X25519` line holds the locked copy of the file key for one recipient, and the `---` line closes the header. Everything after that is the encrypted content.

A file can have several recipients, which is great for sharing or for keeping a backup key somewhere else. Repeat `-r`, or put the public keys in a text file, one per line, and use `-R`:

```
age -R recipients.txt -o backup.tar.age backup.tar
```

Any one of the matching identities can decrypt it.

Passphrases, for when keys are overkill

Sometimes the person locking and the person unlocking are the same, and there's no point in a key file. That's what `-p` is for:

```
age -p -o diary.txt.age diary.txt
```

age asks for the passphrase twice, without showing it, and the prompt disappears from the screen once it's typed:

Decrypting doesn't need any flag besides `-d`. age notices it's a passphrase file and asks for it.

Pressing Enter on an empty prompt makes age generate a passphrase, which in my test came out as `economy-ready-empty-outer-yellow-wall-casino-affair-legend-goddess`. Ten random words is strong and still possible to write down, so it's a good default when nothing better comes to mind.

SSH keys work too

Anyone who uses SSH already has a key pair, and age accepts `ssh-ed25519` and `ssh-rsa` keys directly:

```
age -R ~/.ssh/id_ed25519.pub -o notes.ssh.age notes.txt
age -d -i ~/.ssh/id_ed25519 -o notes.txt notes.ssh.age
```

This is a neat trick for sending a file to someone on a server: their SSH public key is often already in an `authorized_keys` file, or published on their Git hosting profile.

And since 1.3 there's `age-inspect`, which tells what's inside an age file without decrypting it. Here it is on the file above:

```
notes.ssh.age is an age file, version "age-encryption.org/v1".

This file is encrypted to the following recipient types:
  - "ssh-ed25519"

This file does NOT use post-quantum encryption.

Size breakdown (assuming it decrypts successfully):

    Header                      180 bytes
    Encryption overhead          32 bytes
    Payload                      44 bytes
```

The post-quantum option

Version 1.3 added hybrid post-quantum keys, which combine the classic X25519 with ML-KEM-768. The idea is to protect files today against a future quantum computer that could break X25519, without trusting the new algorithm alone.

```
age-keygen -pq -o pq.txt
```

Everything else works the same. The only visible difference is the size of the public key. In my test, the classic recipient had 62 characters and the post-quantum one had 1,959. Still fine in a recipients file, just not something to read aloud over the phone.


```
finger sky@observatorium.thatsmy.tools: the sky over the finger protocol
gopher://pablomurad.com/1/sky: the sky as a gophermap, inside my Gopher hole
nc observatorium.thatsmy.tools 1900: the sky in plain text, then the
connection closes
echo play | nc observatorium.thatsmy.tools 1900: the last 30 nights, one
after another
```

With the last one, thirty skies go by in the terminal, and the comets move from night to night.

Counting better

Two bugs made the numbers lower than they should be, and one filter made them higher.

The first bug was log rotation. Observatorium reads each log from where it stopped last time. When logrotate renamed `access.log` in the early hours of the morning, the reader opened the new file from the top and lost whatever the old one had received after the last run. That was 74 to 455 lines of pablomurad.com every night. Now the reader finds the renamed file by its inode,[2] finishes it from the saved position and only then starts the new one.

The second was about the forge. It only counted a sign-in from the login form, but my Forgejo is entered through Pocket ID, so fornax never got a satellite for a sign-in. The OAuth2 callback now counts too.

The filter goes the other way. When someone shares a link on the fediverse, every server that sees the post fetches the page to build a preview card. Mastodon, Akkoma, Misskey, GoToSocial and Friendica all do it. Those aren't people reading, so they don't count anymore, together with health checkers. On pablomurad.com that removed about 650 of 2,900 daily visits.

Still keeping nothing about visitors

The privacy rules didn't change, they only got dates. The database keeps hourly counts and a scrambled code (an HMAC) for each visited page, and visits are forgotten after 48 hours. Service probes stay for 30 days, because the eclipses page needs them. IP addresses, browsers, users, referrers and URL parameters are thrown away while the log is being read.

2 The inode is the number the filesystem uses to identify a file. Renaming a file changes its name and keeps its inode, so the reader can still find the old log after logrotate moves it.

The second band is new. It holds the sites I publish: nuntius (pablo.news), norma (modeorg.org), codex (murad.gg), reticulum (weirdnet.org with its links, its directory and miskatonic.lol) and pictor (phlog.space).

The third band is the forge, the terminals and the small protocols: fornax (Forgejo), telegraphum (the BBS) and malleus (hammer) were already there, and now columba (Misfin, the SSB room and finger), clavis (haxor.zip and haxor.lol) and littera (ascii.zone on the web, Gemini and Gopher) join them.

The last band, low in the sky, is for the places I mostly see through their probes: alveus (runv.club), calvaria (skull.black and its news server), cartographus (genmap.app) and the newcomer theatrum (infinit.show).

Some regions gather several sites, so a source can now carry a prefix. That keeps their pages apart: the front page of links.weirdnet.org is its own star, not mixed with the front page of weirdnet.org. Fourteen new log sources feed the new regions, and every new service has its own probe, including one that speaks NNTPS to skull.black. weirdnet.org is read from the log behind Anubis, the bot filter in front of it,[1] so challenge pages don't count as visits.

A moon, an archive and a list of eclipses

At night, a moon crosses the top of the sky in its real phase, as seen from São Paulo: ◐ waxing, ● full, ◑ waning.

`/sky/` used to repeat the home page. Now it lists every night drawn so far, and each day still has its own page, like `/sky/2026-11-14/`, never rewritten once the day closes. To make sure each day ends with a complete sky, the timer runs at 2:55, 5:55 and so on, every three hours, so the last drawing of the day is at 23:55.

There's also a new page, /eclipses/, with every time a service went dark in the last 30 days. As I write this, it says "no eclipses in the last 30 days."

More ways to look at it

The sky was already on the web, on Gemini at `gemini://observatorium.thatsmy.tools/`, in an Atom feed and over a plain TCP port. Now there are a few more doors:

1 Anubis asks each new browser to solve a small proof-of-work puzzle before it shows the page. A person waits a second or two. A scraper that doesn't run JavaScript stays stuck on the challenge page.

Text instead of binary

For pasting an encrypted secret into an email, a chat or a config file, `-a` (armor) produces text instead of binary:

```
age -a -r age1... -o secret.txt.age secret.txt
```

The result starts with `-----BEGIN AGE ENCRYPTED FILE-----` and survives any copy and paste.

Why I think age is better than GPG

This part is my opinion: for encrypting files, age is better than GPG.

GPG can do a lot more. It signs, it encrypts email, it has a web of trust, subkeys, expiration dates, smart card support and decades of history. That's also the problem. To encrypt one file with GPG, I first need a keyring, imported keys, a decision about trust levels and some knowledge of which of the many options are safe. That layer is where I see the risk.

age has no keyring, no trust levels and no settings. A key is one line of text. The format is small and documented, and the defaults are the only choices. When the job is "lock this file", that simplicity is the security feature.

What age doesn't do is sign. If a file needs to prove who made it, that's a different tool, like `minisign` or `ssh-keygen -Y sign`, and maybe a later part of this series.

The plot twist: age and KeePassXC

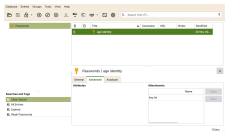
The previous post set up KeePassXC to keep passwords on every device, synced with Syncthing, with a recovery plan. That vault is the perfect home for an age identity.

The idea: the secret key stops living as a loose `key.txt` on disk and becomes an attachment inside a KeePassXC entry. The vault is already encrypted, already synced, already backed up, and already part of the recovery plan. The age key inherits all of that for free.

In the KeePassXC window, it's an entry like any other, with the key file in the Attachments list (Advanced tab):

From the terminal, `keepassxc-cli` does the same, and the key never needs to touch the disk again. `attachment-export --stdout` sends it straight into age, which reads the identity from standard input with `-i -`:

Step by step:



```
~/age-demo $ keepassxc-cli attachment-import vault.kdbx "age identity" key.txt key.txt
Enter password to unlock vault.kdbx:
Successfully imported attachment key.txt as key.txt to entry age identity.
~/age-demo $ shred -u key.txt && ls
diary.txt.age  notes.ssh.age  notes.txt  notes.txt.age  pq.txt  trip.txt.age  vault.kdbx
~/age-demo $ keepassxc-cli attachment-export --stdout vault.kdbx "age identity" key.txt | age -d -i - notes.txt.
age
Enter password to unlock vault.kdbx:
the wifi password is: correct-horse-battery
~/age-demo $
```

1. `keepassxc-cli attachment-import vault.kdbx "age identity" key.txt key.txt` puts the key inside the vault.
2. `shred -u key.txt` overwrites and removes the loose copy. Windows has no `shred`; `Remove-Item key.txt` deletes it, and SDelete, from Microsoft's Sysinternals tools, overwrites it first.
3. To decrypt, `keepassxc-cli attachment-export --stdout vault.kdbx "age identity" key.txt | age -d -i - -o notes.txt notes.txt.age` asks for the vault password and decrypts.

The public key, the `age1...` line, can go in the entry's notes. Encrypting only needs that, so new files can be locked anytime, even with the vault closed, and only unlocking asks for the vault.

On Windows, `keepassxc-cli.exe` comes with KeePassXC, and the same commands work in PowerShell 7 or `cmd.exe`.

The result is a nice chain. The vault protects the age key, Syncthing carries the vault to every device, and age protects everything else: backups, exported notes, that one PDF with document scans. One master password (plus a YubiKey, if added) guards it all. ■



ART · NOCTEM.ART

10 OBSERVATORIUM · SMALL-WEB · SELF-HOSTING 23 NOVEMBER 2026

The Observatorium sky got bigger

Observatorium, the status page that draws my servers as a night sky in text, went from eleven regions to twenty. It also got a real moon, an archive, a list of outages, finger and Gopher versions, and better counting, still without keeping anything about visitors.

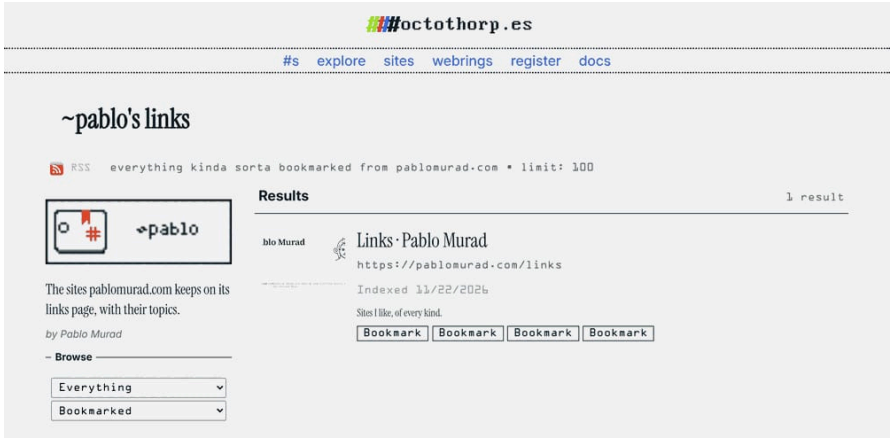


In October I wrote about Observatorium, the status page that draws my public systems as a night sky in plain text. Back then it lived at murad.wtf, and now it's at observatorium.thatsmy.tools. That first post explains the stars, comets, constellations and satellites, and it's the place to start. Since then the sky has almost doubled, got a moon, and learned to answer over finger and Gopher. This is what changed.

From eleven regions to twenty

The first version had eleven regions, one per service. The sky still fits in 80 columns by 24 lines, but now it holds twenty, arranged in four bands from top to bottom.

The first band is home and the networks: domus (pablomurad.com), gemini (the Gemini capsule), talpa (Gopher), chorus (murad.social) and papilio (Bluesky).



The line break and the stray `)` before "createdRange" are the GIF showing through.[2]

How to open it

1. Save the button from /links as a GIF. Converting it to another format, or sending it through an app that recompresses images, can throw the comment away.
2. Open the Explore page on Octothorpes and drop the GIF on it, or use the Upload MultiPass button.
3. Explore reads the query, runs it and shows the results, with an RSS link for the same search.

A site can open a MultiPass by itself too, with one of the web components Octothorpes ships:

```
<script type="module" src="https://octothorp.es/components/octo-multipass-
loader.js"></script>
<octo-multipass-loader gif="https://pablomurad.com/assets/pablo-links.
multipass.gif" render="list"></octo-multipass-loader>
```

Making one goes the other way: build a search on Explore and press "Make MultiPass GIF". The MultiPass docs cover the full format, and the project suggests sharing new ones with the hashtag #OPMultiPass. ■

2 GIF stores that text in pieces of up to 255 bytes, each one preceded by a byte with its length. This query has 551 bytes, so it goes in as 255, 255 and 41. The length 255 isn't a printable character, so `strings` breaks the line there, and 41 is the ASCII code for a closing parenthesis.

04. RUNV · RUNV-CLUB · TILDE 21 NOVEMBER 2026

Come make a home on runv.club

runv.club is a small public Unix server from Brazil, and it has room for more people. A shell, a web page, a blog on the web, Gemini and Gopher, our own IRC, a radio, a TV with old films at night and good company. No ads, no tracking, no algorithm.



runv.club is a small public Unix server,[1] run from Brazil, and it has room for more people. We brought it back in October, and the house is filling up with the good kind of noise: a radio playing members' music, old films on the TV at night, a canvas getting painted one pixel at a time, blogs showing up on the planet.

This is everything that comes with an account. None of it is required. Use what you like and ignore the rest.

A home on the old internet and the small one

Your page lives at runv.club/~you. You also get your own corner on

1 A pubnix is a Unix machine shared by many people, each with an account of their own. The ones called tildes take the name from the ~ in front of every member's folder. The whole story is in A short history of pubnixes and tildes.

Gemini and on Gopher,[2] a finger file, and a blog that shows up on the club's planet: drop markdown in `~/blog` and it goes up on all three. If you'd rather not write HTML, there's a guide for every step on the wiki.

And there's more room to publish than that:

- a photo gallery: whatever goes in `~/pics` becomes a gallery, with the location and camera data stripped
- a site straight from git: a repository called pages on git.runv.club turns into a site at you.runv.space
- an eepsite of your own on I2P,[3] from `~/public_i2p`
- twtxt, a microblog that lives in a plain text file[4]
- a /now page, to say what you're focused on lately
- a guestbook, and a small visit counter that tracks nobody, both off until you turn them on
- a spot on the runv ring, the club's webring
- proofs that tie your Mastodon, your GitHub or your own site to your account
- 88x31 buttons to put on your page

A real shell

It runs Debian, with a nice prompt and colours out of the box. More languages are installed than I can list here: C, Python, Go, Rust, Haskell, Racket, Lua, Ruby, Common Lisp and a pile more. Write code, run it, show it off. Type `runv` and a dashboard shows what's going on in the club.

Your own addresses

- you@runv.club, forwarded to the inbox you already use
- you.runv.sh, for anything you run here, or tunnel from your own machine
- a domain of your own, pointed at your page or your app, with HTTPS

2 Gemini and Gopher are two small protocols made for text. Gopher came out of the University of Minnesota in 1991 and lost the race to the web. Gemini appeared in 2019, somewhere between the two, and has no ads, no scripts and no cookies by design.

3 I2P is an anonymous network that runs on top of the internet. Its sites, called eepsites, end in .i2p and only open through an I2P router.

4 In twtxt, each person keeps a twtxt.txt file with one post per line, and following someone means reading their file. It started in 2016 and still has a small, loyal crowd.


```
word-break: break-all; } }
---
field,value,what it does
meta.title,~pablo's links,the name a reader shows
meta.server,https://octothorp.es/,which Octothorpes server to ask
meta.resultMode,blobjects,pages together with their octothorpes
subjects.include,pablomurad.com,"whose pages, matched loosely, so any
address on the domain counts"
objects.type,notTerms,"links to pages, leaving hashtags out"
filters.subtype,Bookmark,"only bookmarks, no citations or backlinks"
filters.limitResults,100,at most a hundred results
```

The loose match on the domain is the "fuzzy" mode. Explore spells it out in plainer words when it loads the button: "everything kinda sorta bookmarked from pablomurad.com".

Where the query hides

The GIF format has a block reserved for plain text, the Comment Extension, and image viewers skip it. MultiPass puts the JSON there and leaves the frames alone. The ~pablo button weighs 1,394 bytes: 88 by 31 pixels, eight frames that loop, and 551 bytes of query.

DOT

```
digraph gif {
rankdir=TB;
node [shape=box, fontname="Helvetica", fontsize=12, margin="0.4,0.12",
style=filled, fillcolor="#ffffff", color="#111111"];
edge [arrowhead=none, color="#999999"];
h [label="GIF89a header\n88 x 31, palette"];
c [label="Comment Extension\n551 bytes of JSON: the MultiPass",
fillcolor="#fde8e5", color="#d93a2b"];
n [label="Netscape block\nloop forever"];
f [label="8 frames\nthe picture"];
t [label="trailer"];
h -> c -> n -> f -> t;
}
```

Since the query is plain text, `strings` finds it without any special tool:

CONSOLE

```
$ curl -sO https://pablomurad.com/assets/pablo-links.multipass.gif
$ strings -n 20 pablo-links.multipass.gif
{"meta":{"title":"~pablo's links","description":"The sites pablomurad.com
keeps on its links page, with their topics.","server":"https://octothorp.
es/","author":"Pablo Murad","image":"https://pablomurad.com/assets/pablo-
links.multipass.gif","version":"1",
"resultMode":"blobjects"},"subjects":{"mode":"fuzzy","include":
["pablomurad.com"],"exclude":[]},"objects":
{"type":"notTerms","mode":"exact","include":[],"exclude":[]},"filters":
{"subtype":"Bookmark","limitResults":"100","offsetResults":"0","dateRange":
{},)"createdRange":null,"indexedRange":null}}
```

Bookmarks with topics

The entries on /links are plain links with two extra attributes. This is the one for Octothorpes itself:

```
<a href="https://octothorp.es/" rel="octo:bookmarks" data-
octothorpes="protocol,webring">octothorp.es</a>
```

`rel="octo:bookmarks"` tells the server that the page bookmarks that address, and `data-octothorpes` puts topics on the link. The topics belong to the bookmark, not to the whole page, so one page can file each link under its own words. Once the page is indexed, the Octothorpes API answers for it. Trimmed a little:

CONSOLE

```
$ curl -s "https://octothorp.es/get/everything/bookmarked?s=pablomurad.com&
match=fuzzy-s"
{"results":[{"@id":"https://pablomurad.com/links",
"title":"Links · Pablo Murad",
"octothorpes":[
{"uri":"https://htmlforpeople.com","type":"Bookmark","terms":["tutorials"]},
{"uri":"https://octothorp.
es","type":"Bookmark","terms":["webring","protocol"]},
{"uri":"https://shellsharks.com","type":"Bookmark","terms":["blogs"]},
{"uri":"https://pablomurad.com","type":"Bookmark","terms":["blogs"]}]}]}
```

A MultiPass is a saved question

A MultiPass is a JSON object that describes a query to that API. It stores the question and not the answer, so whoever opens it gets the results as they stand on that day. The project calls them mixtapes of websites, made for friends.

The one on /links asks, more or less, for everything pablomurad.com has bookmarked. Field by field:

CSSV

```
---
/* cssv:title The ~pablo MultiPass, field by field */
/* cssv:description What each field of the query inside the button does. */
table { width: 100%; font: 15px/1.45 source-serif-4, Georgia, serif; color:
#111; }
thead { display: none; }
td { padding: 0.45em 0.7em 0.45em 0; border: 0; border-top: 1px solid
#e6e6e6; vertical-align: baseline; }
tbody tr:first-child td { border-top: 1px solid #111; }
tbody tr:last-child td { border-bottom: 1px solid #111; }
td[data-col="field"] { font: 12px/1.6 ibm-plex-mono, ui-monospace,
monospace; color: #666; white-space: nowrap; }
td[data-col="value"] { font: 600 13px/1.5 ibm-plex-mono, ui-monospace,
monospace; color: #d93a2b; word-break: break-word; }
@media (max-width: 30rem) { td[data-col="field"] { white-space: normal;
```


runv.club radio



Ludovico Einaudi - Newton's Cradle (from ~nct's shelf)

IRC, a radio and a TV

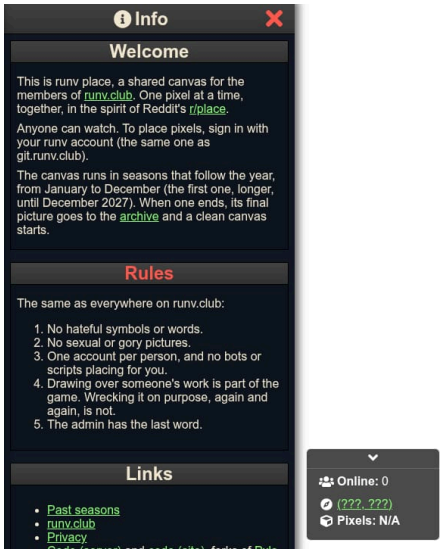
runv on irc.runv.club, our own network, is where the club hangs out. Everyone gets a bouncer, so you never miss what was said, and your nick is yours. Our bot, runvbot, does polls, the weather, "photos" of the chat and a lot of silly text tricks. After six months, you can run a bot of your own.

The radio at runv.club/radio plays music, including members' own tracks: put songs in `~/radio` and they join the rotation, or go live from your own computer. There's a little animated room and a chat next to it. From midnight to 8 in the morning (Brasília time) it turns into runv TV, with old films, cartoons and newsreels.[5]

Places to hang out

- the forum, for slower conversations, and board, a forum inside the terminal
- place.runv.club, a shared canvas where everyone paints one pixel at a time
- CryptPad, for documents and boards, with everything encrypted

5 Everything on runv TV comes from the Internet Archive and is public domain or Creative Commons.



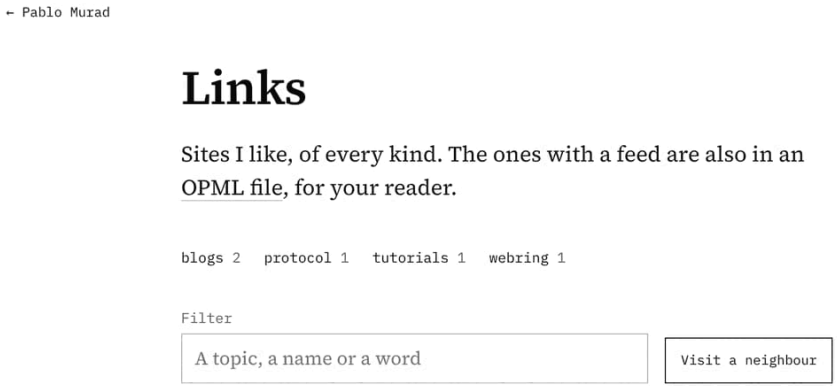
- our own git server
- graffiti, a wall every member can write on, and `/srv/square`, a folder where everyone writes
- write, to drop a message straight into someone's terminal, and nice, to leave someone an anonymous compliment
- lemmy.club, communities federated with the rest of the fediverse

Little things that make it feel like home

- journal: a private diary in the terminal
- rec: records your terminal session and gives you a link to share
- pb: a pastebin, with your code in colour
- status: one-line updates you can embed on any site
- badges for things you did here, from a first blog post to a plant kept alive for 100 days
- a feed reader, bookmarks, calendar and contacts
- ntfy, for notifications on your phone
- an exquisite corpse[6]
- a botany plant to keep alive[7]

6 The exquisite corpse is a game the surrealists played in the 1920s: each person adds a piece to a text or a drawing, seeing only the end of what came before.

7 botany is a plant that lives in the terminal. It was born on tilde.town, grows while it's watered and dies if it's forgotten, and friends can water it for you.



name or any word, and the "Visit a neighbour" button sends the browser to one of the sites at random.

Two small markers do the neighbourly part. When a site keeps pablomurad.com in its own list, its entry gets a red "mutual" tag, and "their list" opens the OPML file where that site publishes the sites it follows.

Octothorpes, briefly

Octothorpes describes itself as hashtags that work on any website. A site adds a line or two of HTML, marks its hashtags and links, and an Octothorpes server indexes the page when the site asks for it. The server doesn't crawl the web, so only pages that people submit end up there. Out of that index come tag pages, backlinks, webrings, bookmarks and RSS feeds for any mix of them. I was already wandering around it when I found the ANSI artists behind Note III.

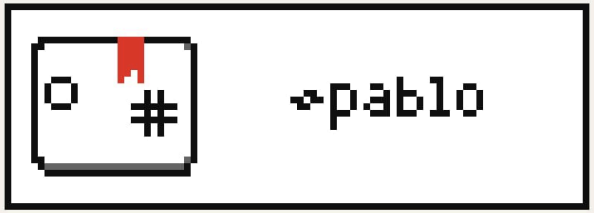
The project comes from Stucco Software and The Idea Store, and it had been in public use for about a year when the first stable version came out:

```
TIMELINE
2025-08-02 v0.5, the first stable version of Octothorpes, launches on HTML
Day
2025-11-03 MultiPass is introduced: a search that can be saved, shared and
hidden inside a GIF
2026-05-01 v0.6 adds the Explore page, tags on links, new link types and
MultiPass components
2026-11-22 /links goes up on pablomurad.com with its own MultiPass
```

09 OCTOTHORPES · MULTIPASS · INDIEWEB 22 NOVEMBER 2026

MultiPass: my links page fits in an 88x31 button

The new /links page on pablomurad.com lists the sites I like, and an 88x31 button at the bottom carries the whole list as an Octothorpes MultiPass. What Octothorpes is, how bookmarks with topics work, where the query hides inside the GIF and how to open it.



The site has a new page, /links, with the sites I like, of every kind. At the bottom sits a small 88x31 button[1] with a card, a red ribbon and a hash sign. It looks like any other button from the 2000s, but it carries the whole list inside it. It's a MultiPass, a saved Octothorpes query, and any MultiPass reader can open it.

The /links page

Each site on the page has its icon, its address, a short description and a note in my own words. Sites with a feed also show their latest post, and they go into an OPML file that a feed reader can import in one go. The topics at the top work as an index, the filter narrows the list by topic,

1 88 by 31 pixels is the size of the small web buttons from the late 90s and the 2000s, the same size as the badges in the footer of this site.



- games: nethack, text adventures with frotz, tetris and the BSD games, fortune with cowsay, and Dungeon Crawl through the tildeverse
- a backup of the whole server, every day

What you won't get

No ads, no tracking, no algorithm. Nobody here is trying to grow anything but the club itself. It's small on purpose, and we look after each other. There are a few simple rules, and the main one is: be kind.

How to join

Go to runv.club/join, tell us a bit about yourself, and send an SSH key (the page explains how to make one). From a terminal, `ssh join@runv.club` does the same. Every request is read by a person, and seven people get in each week, so there may be a short wait. You get a link to check your place in line. You need to be 16 or older.

Friends who only want a blog can have one from a single markdown file at banany.lol, no account needed.

Want to look around first?

- visit webchat.runv.club and say hi in #runv, no account needed
- browse the wiki and the news to see what's been happening
- read what members are writing on the planet

If you've been missing the internet that felt like a neighbourhood, come by. There's a terminal waiting for you.

■

■ ■ ■

$ cd ~pablo

05 EMACS · ORG-MODE · NOTES 21 NOVEMBER 2026

Linked notes in Emacs: Denote, org-roam and org-node

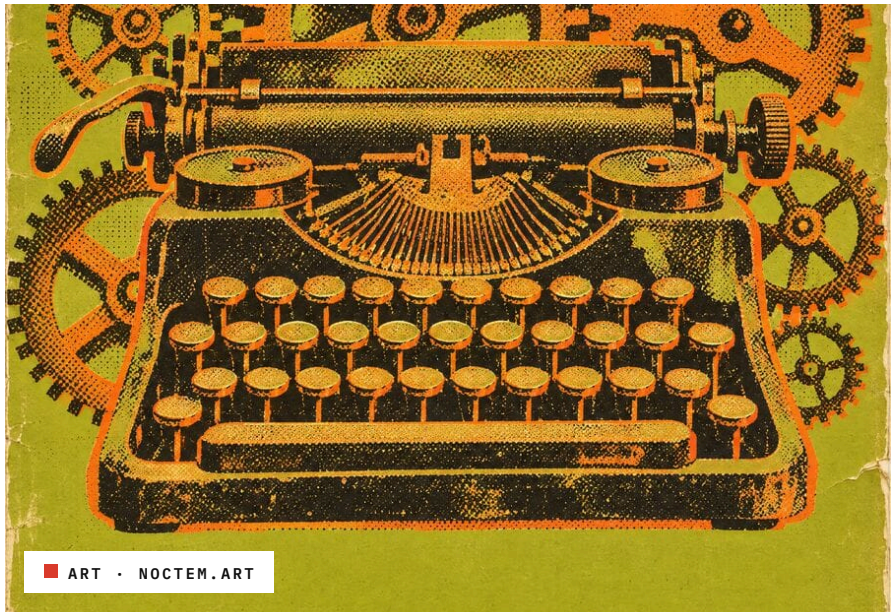
A complete walkthrough of the three big ways to keep linked notes in Emacs: Denote, which puts everything in the file name, org-roam, with its SQLite index and heading-level nodes, and org-node, fast and database-free. Configs, real screenshots, how to pick and a cheat sheet.



The Obsidian posts here (an Obsidian vault that lasts, the two plugin lists and the 50 tricks) all orbit the same idea: small notes, linked to each other, stored as plain files. Emacs can do the same thing, and it has three serious packages for it: Denote, org-roam and org-node. They solve the same problem with very different ideas about where the structure of a note collection should live.

This guide walks through all three from an empty config to a working setup, with real screenshots from a small test collection about the small web. It assumes Emacs is installed and Org mode feels familiar. If not, Org mode from zero covers the basics and Emacs for the curious covers installing packages.


```
Log and blame,l l,log of the current branch
Log and blame,l a,log of all references
Log and blame,l r,reflog
Log and blame,C-c M-g b,blame the current file
Stash,z z,stash index and working tree
Stash,z i,stash only the index
Stash,z x,"stash, keeping the index"
Stash,z a,apply a stash
Stash,z p,pop a stash
Stash,z k,drop a stash
Rebase,r i,interactive rebase from the commit at point
Rebase,r w,reword one commit
Rebase,r k,remove one commit
Rebase,c r e s f d,"in the todo list: pick, reword, edit, squash, fixup,
drop"
Rebase,M-p and M-n,move a commit up or down
Rebase,C-c C-c,run the rebase
Rebase,C-c C-k,abort
Merge,m m,merge a branch
Merge,m a,abort the merge
Merge,k,on a conflicted file: keep ours or theirs
Merge,e,on a conflicted file: resolve with ediff
Merge,C-c ^ u / l,keep upper or lower in smerge
Forge,N,Forge menu
Forge,N f f,fetch issues and pull requests
Forge,N c i / N c p,create an issue or a pull request
Forge,N v i / N v p,visit an issue or a pull request
```



ART · NOCTEM.ART

```
table {
  --cssv-key: area;
  --ink: #111;
  --muted: #666;
  --line: #e6e6e6;
  --signal: #d93a2b;
  width: 100%;
  font: 15px/1.45 source-serif-4, Georgia, serif;
  color: var(--ink);
}
thead { display: none; }
td { padding: 0.45em 0.7em 0.45em 0; border: 0; border-top: 1px solid var(--
ink); vertical-align: baseline; }
td[data-col="area"] { width: 8.5rem; font: 12px/1.6 ibm-plex-mono, ui-
monospace, monospace; text-transform: lowercase; color: var(--muted); }
td[data-col="keys"] { width: 9.5rem; font: 600 13px/1.5 ibm-plex-mono, ui-
monospace, monospace; color: var(--signal); white-space: nowrap; }
:is(tr[data-key="Status"] + tr[data-key="Status"], tr[data-key="Staging"]
+ tr[data-key="Staging"], tr[data-key="Commits"] + tr[data-key="Commits"],
tr[data-key="Log and blame"] + tr[data-key="Log and blame"], tr[data-
key="Stash"] + tr[data-key="Stash"], tr[data-key="Rebase"] + tr[data-
key="Rebase"], tr[data-key="Merge"] + tr[data-key="Merge"], tr[data-
key="Forge"] + tr[data-key="Forge"]) td { border-top-color: var(--line); }
:is(tr[data-key="Status"] + tr[data-key="Status"], tr[data-key="Staging"]
+ tr[data-key="Staging"], tr[data-key="Commits"] + tr[data-key="Commits"],
tr[data-key="Log and blame"] + tr[data-key="Log and blame"], tr[data-
key="Stash"] + tr[data-key="Stash"], tr[data-key="Rebase"] + tr[data-
key="Rebase"], tr[data-key="Merge"] + tr[data-key="Merge"], tr[data-
key="Forge"] + tr[data-key="Forge"]) td[data-col="area"] { color:
transparent; }
tbody tr:hover td { background: #f4f4f2; }
tbody tr:last-child td { border-bottom: 1px solid var(--ink); }
@media (max-width: 30rem) {
  td[data-col="area"] { width: 4.8rem; font-size: 11px; }
  td[data-col="keys"] { width: auto; }
}
---
area,keys,action
Status,C-x g,open the status buffer
Status,?,dispatch menu with every command family
Status,TAB,fold or unfold the section at point
Status,g,refresh
Status,q,bury the buffer
Status,RET,"visit the thing at point (file, commit, stash)"
Staging,s,"stage the file, hunk or region at point"
Staging,u,unstage it
Staging,k,discard it
Staging,C-SPC then s,stage only the selected lines
Staging,+ and -,more or less diff context
Commits,c c,create a commit
Commits,c a,amend the last commit
Commits,c e,extend the last commit with staged changes
Commits,c w,reword the last commit
Commits,c f,fixup commit for an older commit
Commits,c F,instant fixup
Commits,C-c C-c,finish the message
Commits,C-c C-k,cancel
```


What a note system needs

Before looking at any package, it helps to agree on the job. A collection of linked notes needs four things:

1. **A stable way to name a note.** Titles change. A link that breaks every time a title changes is useless after a year.
2. **Fast finding.** Typing a few letters of a title and jumping there, in under a second, with hundreds or thousands of notes.
3. **Linking.** Inserting a link to another note without leaving the keyboard and without typing a path.
4. **Backlinks.** From any note, seeing which other notes point to it. This is what turns a folder of files into a web of ideas.

Org mode alone already has links and search. What these packages add is the naming scheme, the fast lookup and, above all, the backlinks.[1]

The three, side by side

Denote keeps everything in the file name. Each note gets a name like `20261103T214000--gemini__protocols_smallweb.org`, with a timestamp that works as a permanent ID, the title and the keywords. There is no database. Finding notes is a matter of matching file names, and backlinks come from searching the files for links. Requires Emacs 28.1 or newer.

org-roam keeps the structure in an SQLite database. Each note (a file, or a heading inside a file) gets an Org ID, and org-roam indexes titles, aliases, tags and links into the database so lookups and backlinks are instant. It is the oldest and most used of the three, with the biggest ecosystem. Requires Emacs 27.1 or newer.

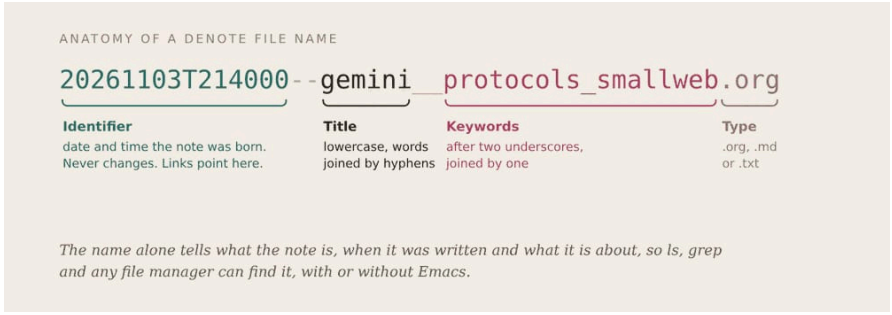
org-node reads the same kind of notes as org-roam, with the same Org IDs, but keeps its index in memory, rebuilt in the background, with no database at all. It is younger, very fast, and can sit next to org-roam on the same files. Requires Emacs 29.1 or newer.

Denote

Denote was introduced in June 2022 by Protesilaos Stavrou, the author of the Modus themes that ship with Emacs.[2] Its central idea is that the file name should carry everything needed to identify a note, so the notes keep working in any program, from `ls` to a phone's file manager. The 4.3

1 The idea of small linked notes goes back to the Zettelkasten, the card box kept by the German sociologist Niklas Luhmann, which grew to about 90,000 cards over his career. The Niklas Luhmann-Archiv has digitized it, and the cards can be browsed online.

2 The Modus themes (modus-operandi and modus-vivendi) are built into Emacs since version 28. The screenshots in this post use modus-vivendi.



series came out in November 2026 (4.3.1 is the current release on GNU ELPA), and the changelog is worth a look after installing.

THE FILE NAME IS THE NOTE'S IDENTITY

Every Denote file name follows the same pattern:

- **Identifier:** the date and time of creation, as `20261103T214000`. It never changes, and every link points to it.
- **Title:** after two hyphens, in lowercase, with words joined by single hyphens.
- **Keywords:** after two underscores, joined by single underscores.
- **Extension:** `.org`, `.md` or `.txt`.

Because of this pattern, a plain directory listing already works as an index. Sorting by name sorts by date. Searching for `_unix` finds every note tagged unix. Renaming a title changes the name but leaves the identifier alone, so links survive.

INSTALLING AND CONFIGURING

Denote is on GNU ELPA, so it installs with no extra archives. A starting config, adapted from the manual:

```elisp
(use-package denote
:ensure t
:hook (dired-mode . denote-dired-mode)
:bind
(("C-c n n" . denote)
("C-c n r" . denote-rename-file)
("C-c n l" . denote-link)
("C-c n b" . denote-backlinks)
("C-c n d" . denote-dired)
("C-c n g" . denote-grep))
:config
(setq denote-directory (expand-file-name "~/notes/"))
(setq denote-known-keywords '("smallweb" "protocols" "unix" "history"))
(denote-rename-buffer-mode 1))
```



```console
$ git config --global github.user USERNAME
$ git config --global github.user
USERNAME
```

Then:

1. Create a classic personal access token with the `repo`, `user` and `read:org` scopes.
2. Store it in an encrypted auth-source file, with one line like the one below.
3. Run `M-x auth-source-forget-all-cached` so Emacs rereads the file.

```
machine api.github.com login USERNAME^forge password TOKEN
```

The `^forge` suffix after the user name lets the same machine hold other tokens for other tools. Keeping only `~/.authinfo.gpg` in `auth-sources` avoids the token ending up in a plain text file:

```elisp
(setq auth-sources '("~/.authinfo.gpg"))
```

After that, `N f f` in a repository fetches its issues and pull requests, and then:

- N c i: create an issue
- N c p: create a pull request from the current branch
- N v i and N v p: visit an issue or pull request in Emacs
- N b I and N b P: open the list of issues or pull requests in the browser
- N l n: list notifications

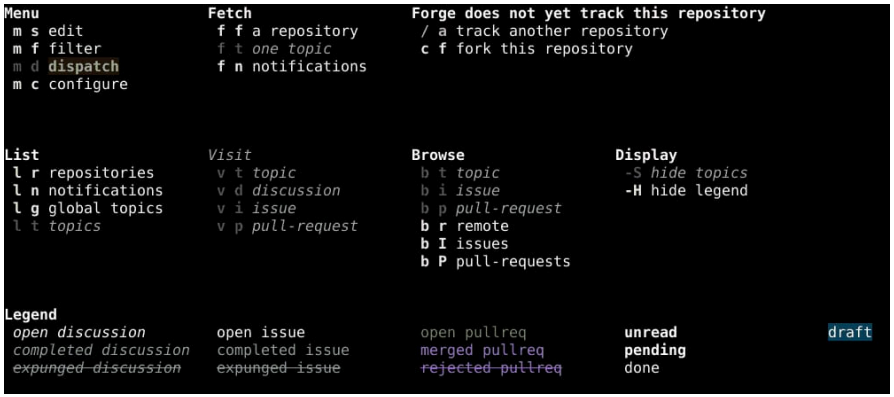
Which forges work matters a lot here. GitHub and GitLab are fully supported. Forgejo, Gitea, Gogs and Bitbucket are only partially supported: Forge can browse them and do some things, but most of the API isn't wired up yet, and the documentation names Forgejo as the next forge whose API will be supported. Anyone hosting their own Forgejo instance[4] gets the Magit side of things in full and only a small part of Forge for now. The Forge manual has the details of every command.

Cheat sheet

Every key from this post in one table, grouped by task:

```csv
---
/* cssv:title Magit cheat sheet */
/* cssv:description The Magit keys used in this post, grouped by task. */
```

4 Forgejo started as a fork of Gitea in 2022 and is the software behind Codeberg.



In smallzine, `C-c ^ u` kept the title from `main` and the feed link from the branch, because the link wasn't part of the conflict. After saving, Emacs noticed there were no markers left and marked the file as resolved, and back in the status buffer `index.html` had moved to Staged changes.

Use ediff. `e` on the file opens ediff with three panes, ours, theirs and the result, for conflicts too tangled to read in one buffer.

With nothing left unmerged, `c c` opens the message buffer with Git's default "Merge branch 'feature/rss'" already filled in, and `C-c C-c` creates the merge commit. To give up instead, `m a` aborts the merge and puts everything back.

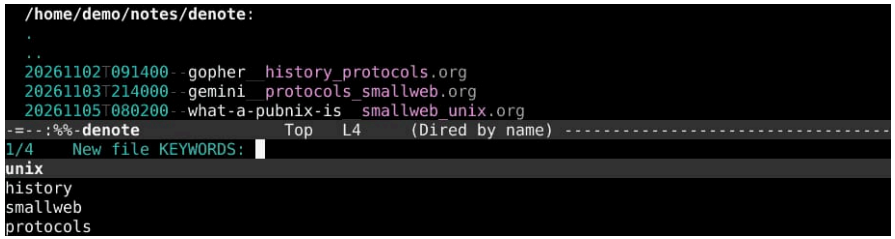
Forge: issues and pull requests

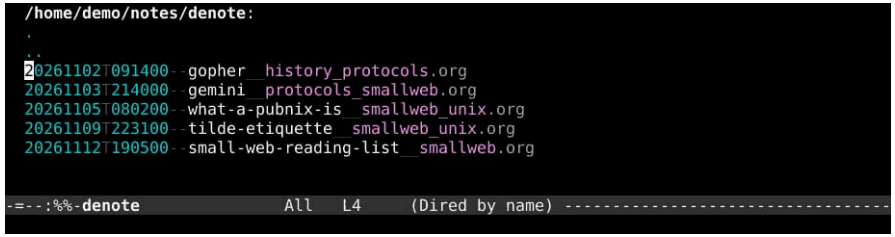
Forge is a separate package by the same team that adds issues, pull requests and notifications to Magit. It fetches them into a local database, so they show up in the status buffer as sections and can be read and answered offline.

```elisp
(use-package forge
:ensure t
:after magit)
```

`N` opens its menu. In smallzine there is no remote on a forge, so Forge says it does not yet track this repository, and the commands that need one are greyed out:

Forge needs a token to talk to the forge's API. For GitHub, the first step is telling Git the account name. Run again without a value, the same command prints what Git stored:





`denote-directory` is where notes live. `denote-known-keywords` is just a starting list for completion; any new keyword typed at the prompt is accepted. `denote-rename-buffer-mode` gives each note's buffer a readable name instead of the long file name.

CREATING A NOTE

`C-c n n` runs `denote`. It asks for a title, then for keywords:

Several keywords can be picked at once (separated by commas). Denote then creates the file with its name already built and writes a small front matter at the top:

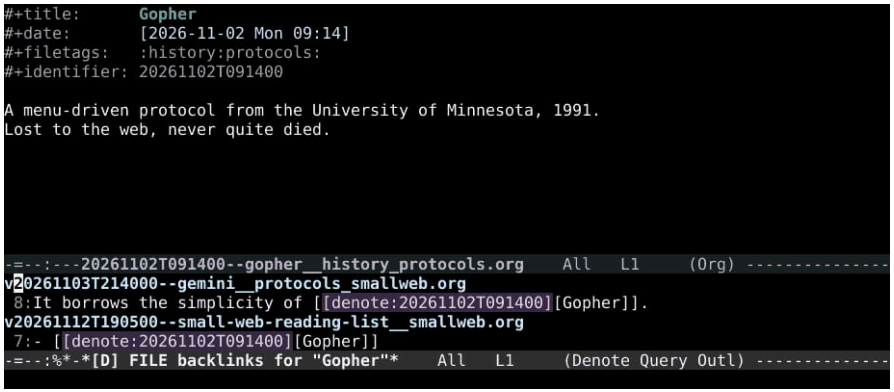
```org
#+title: Gemini
#+date: [2026-11-03 Tue 21:40]
#+filetags: :protocols:smallweb:
#+identifier: 20261103T214000
```

Below that, it's an ordinary Org file. Headings, lists, code blocks, agenda items: everything from Org mode works as usual.

There are other entry points for the same thing. `denote-open-or-create` jumps to an existing note or creates one if nothing matches. `denote-region` turns the selected text into a new note. `denote-org-capture` plugs Denote into Org capture templates, for creating notes from anywhere in Emacs.

SEEING THE COLLECTION IN DIRED

With `denote-dired-mode` on, Dired colors each part of the name, which makes a long listing easy to scan:



`C-c n d` (`denote-dired`) goes further and opens a Dired buffer filtered by a regular expression and sorted by identifier, title or keywords. Asking for `_smallweb` gives a live listing of every note with that keyword.

LINKS AND BACKLINKS

`C-c n l` (`denote-link`) asks for a note and inserts a link at point. In Org files the link looks like this:

```org
It borrows the simplicity of [[denote:20261102T091400][Gopher]].
```

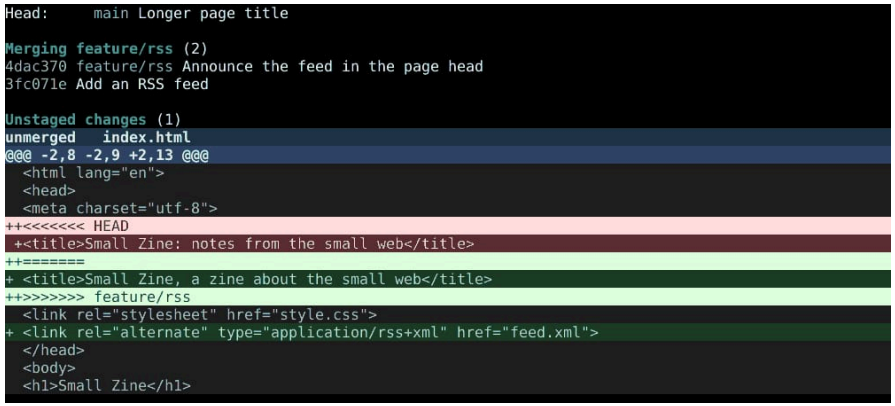
The link targets the identifier, so the title on the other side can change freely. `denote-link-or-create` does the same, but creates the target note when it doesn't exist yet, which fits the habit of linking to an idea before writing about it.

`C-c n b` (`denote-backlinks`) opens a buffer listing every note that links to the current one, with the line where each link appears:

Since there's no index, Denote finds these by searching the files each time. That stays quick for collections of a few thousand notes, especially with ripgrep installed. `denote-find-link` and `denote-find-backlink` do the same job through the minibuffer, for jumping straight to one of them.

RENAMING WITHOUT BREAKING ANYTHING

`C-c n r` (`denote-rename-file`) prompts for a new title and keywords and renames the file, keeping the identifier. There are narrower versions too: `denote-rename-file-title` and `denote-rename-file-keywords` change just one part. When the front matter was edited by hand, `denote-rename-file-using-front-matter` reads the title and tags from the top of the file and fixes the file name to match.




```
head [label="Announce the feed\n(feature/rss)", color="#d93a2b",
fontcolor="#d93a2b"]
merge [label="Merge branch 'feature/rss'", style="rounded,dashed"]
readme -> footer -> style -> intro -> h1 -> title -> merge
readme -> rss -> head -> merge [color="#d93a2b"]
}
```

Merging is `m m` and the branch name. Git merged what it could and stopped on the title:

The status buffer says a merge is in progress and which commits it brings in. `feed.xml` is already staged, because it had no conflict. `index.html` is "unmerged", and its diff shows both versions of the title between the markers Git writes into the file. There are three ways out from here.

Take one side for the whole file. On the file, `k` asks whether to keep ours, theirs or restore the conflict. This is the fastest route when one branch is simply right.

Edit with smerge. `RET` opens the file. Emacs turns on smerge-mode by itself for files with conflict markers[3], and its commands all start with `C-c ^`. With the cursor inside a conflict:

- C-c ^ u: keep upper, our version (HEAD)
- C-c ^ l: keep lower, the incoming branch
- C-c ^ b: keep the base, the common ancestor
- C-c ^ a: keep all versions, to merge them by hand
- C-c ^ RET: keep the version under the cursor
- C-c ^ n and C-c ^ p: jump to the next or previous conflict

3 smerge-mode ships with Emacs and isn't part of Magit. The part that marks the file as resolved when the last conflict marker is gone also belongs to Emacs: it's the `vc-git-resolve-conflicts` option, which is on by default.



that only exist locally. For commits other people have already pulled, a rebase forces everyone to reconcile their copies, so the usual practice is to rebase freely before pushing and very carefully after.

If something goes wrong in the middle, the status buffer shows the rebase in progress with the remaining steps, and `r` offers to continue, skip or abort. And if it went wrong and already finished, `l r` (the reflog) still has the old commits.

Resolving a merge conflict

To get a conflict, smallzine's `main` changed the page title to "Small Zine: notes from the small web", while `feature/rss` changed the same line to "Small Zine, a zine about the small web" and also added a link to the new feed. This is how the two lines of history looked right before the merge, with the branch in red:

```dot
digraph {
node [shape=box, style=rounded, fontsize=11]
edge [arrowsize=0.6]
readme [label="Describe the zine in the README"]
footer [label="Add footer"]
style [label="Style the footer"]
intro [label="Tweak intro text"]
h1 [label="Bigger h1"]
title [label="Longer page title\n(main)"]
rss [label="Add an RSS feed", color="#d93a2b", fontcolor="#d93a2b"]
```


Renaming also works on files Denote didn't create. Running `denote-rename-file` on an old note or a PDF gives it an identifier and brings it into the collection.

THE COMPANION PACKAGES

Since version 4.0, Denote keeps its core small and moves extras into separate packages, all on GNU ELPA:

- **denote-journal:** a daily journal on top of Denote. `denote-journal-new-or-existing-entry` opens today's entry, creating it if needed, and `denote-journal-calendar-mode` marks days with entries in the Emacs calendar.
- **consult-denote:** integrates with Consult, adding `consult-denote-find` and `consult-denote-grep` with live previews.
- **denote-silo:** silos are separate note directories, each with its own `denote-directory`. Work and personal notes can live apart and still use the same commands.
- **denote-sequence:** gives notes hierarchical numbers (1, 1=1, 1=2, 2…), close to the folgezettel scheme from Luhmann's card boxes.
- **denote-org** and **denote-markdown:** extras for each file type, such as Org dynamic blocks that list notes by keyword.

The file type is chosen with `denote-file-type`. Core Denote writes Org, Markdown with YAML or TOML front matter, and plain text. That makes it the easiest of the three to share with Obsidian: point both at the same folder, set the file type to `markdown-yaml`, and the notes work in either program (Obsidian won't follow `denote:` links, but the files and front matter are fine).

org-roam

org-roam started in 2020, by Jethro Kuan, as an Emacs take on Roam Research, the web app that made backlinks popular.[3] It's built on two Org features: the ID property, which gives any file or heading a unique identifier, and `id:` links, which point to that identifier.

In org-roam, a **node** is any file or heading with an ID. A file with ten headings, each with its own ID, holds eleven nodes, and each can be linked to and found on its own. A typical node file looks like this:

3 Roam Research popularized bidirectional links in note apps around 2019 and 2020. Obsidian, Logseq and org-roam all came out of that wave.

```org
:PROPERTIES:
:ID: 5b1f0c2a-6d1e-4c3b-9a57-2f0e8d4b1c90
:END:
#+title: Gopher
#+filetags: :protocols:history:

A menu-driven protocol from the University of Minnesota, 1991.
```

File names don't matter to org-roam. The default template names files with a timestamp and a slug, but they can be anything.

INSTALLING AND CONFIGURING

org-roam is on MELPA. It depends on emacsql (to talk to SQLite), magit-section (for its side buffer) and a recent Org. On Emacs 29 and newer, emacsql uses the SQLite support built into Emacs, so nothing external needs to be compiled.

```elisp
(use-package org-roam
:ensure t
:custom
(org-roam-directory (file-truename "~/notes/roam/"))
:bind (("C-c n l" . org-roam-buffer-toggle)
("C-c n f" . org-roam-node-find)
("C-c n i" . org-roam-node-insert)
("C-c n c" . org-roam-capture)
("C-c n j" . org-roam-dailies-capture-today))
:config
(org-roam-db-autosync-mode))
```

`org-roam-db-autosync-mode` keeps the database in sync every time a note is saved. Without it, the database has to be rebuilt by hand with `org-roam-db-sync`.

The `C-c n` prefix here is the same one Denote's manual uses. Running both at once means picking a different prefix for one of them.

FINDING, INSERTING, CAPTURING

`C-c n f` (`org-roam-node-find`) lists every node by title, aliases included. Typing a name that doesn't exist offers to create it, through a capture template. `C-c n i` (`org-roam-node-insert`) works the same way but inserts an `id:` link at point:

```org
Text-first protocol from 2019. It borrows the simplicity of
[[id:5b1f0c2a-6d1e-4c3b-9a57-2f0e8d4b1c90][Gopher]].
```

`C-c n c` (`org-roam-capture`) creates or appends to a node using templates, the same mechanism as Org capture.

A few commands handle metadata on the current node:


```
pick 116021e Describe the zine in the README
pick 87dc823 Add footer
reword f59b90b WIP footer style
fixup 9215578 fixup! WIP footer style
pick 008267b Tweak intro text
pick 0fd3dd1 Bigger h1 with tighter letter spacing

# Rebase ac85d6a..0fd3dd1 onto ac85d6a (6 commands)
#
# Commands:
# C-c C-c  tell Git to make it happen
# C-c C-k  tell Git that you changed your mind, i.e., abort
# p        move point to previous line
# n        move point to next line
# M-p      move the commit at point up
# M-n      move the commit at point down
# SPC      show the commit at point in another buffer
# RET      show the commit at point in another buffer and select its window
# C-_      undo last change
# d        drop the commit at point
```

`r` opens a menu with more targeted options too: `r w` rewords one commit, `r m` stops at one commit to change it, and `r k` removes one. When autosquash is on (`-a` in the menu, or `rebase.autoSquash` in Git's config), `fixup!` and `squash!` commits arrive already placed and marked.

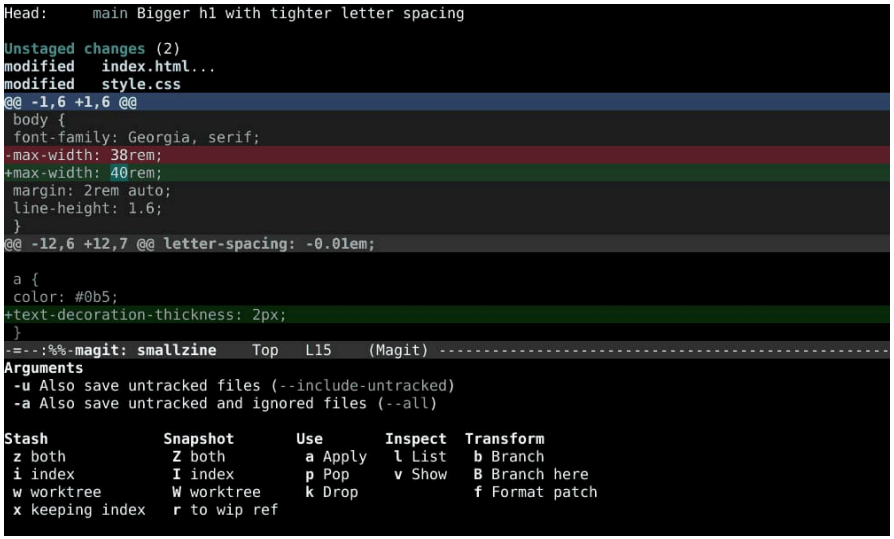
The todo list opens as a buffer where each line is a commit and the first word says what to do with it:

Editing the list doesn't involve typing. On a line, a single key changes its action:

- c: pick, keep the commit as is
- r: reword, keep it but edit the message
- e or m: edit, stop after this commit so it can be changed
- s: squash into the previous commit and combine the messages
- f: fixup, squash and discard this commit's message
- d: drop the commit
- k: comment the line out
- x: insert an exec line that runs a shell command
- b: insert a break, a pause in the middle of the rebase
- M-p and M-n: move the commit up or down
- C-_: undo the last change to the list

`C-c C-c` runs the rebase and `C-c C-k` aborts it. In smallzine, two lines changed. "WIP footer style" became a reword, because WIP is no way to describe a commit, and its `fixup!` was already set to fixup thanks to autosquash. After `C-c C-c`, Git stopped at the reworded commit, Magit opened the message buffer, the message became "Style the footer", and `C-c C-c` again finished the job:

The picture shows the one rule that matters with rebase. "Add footer" kept its hash because nothing before it changed, while every commit after the first edit got a new one. Rewriting history is fine for commits



Stash

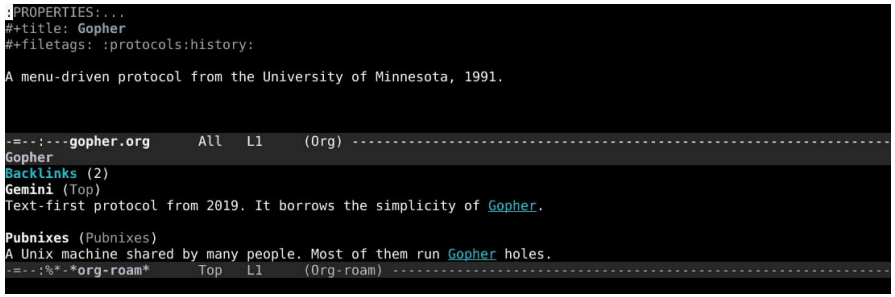
Stash puts changes aside without committing them, for example to switch branches with a dirty working tree. `z` opens the menu:

The menu is more precise than `git stash`. `z z` stashes everything, `z i` only what is staged, `z w` only the working tree, and `z x` stashes everything but leaves the index as it was. The Snapshot column does the same without touching the working tree, which is a cheap way to save a "just in case" copy. `-u` includes untracked files, which plain `git stash` leaves behind.

Getting changes back is `z a` to apply (the stash stays) or `z p` to pop (apply and drop). In the status buffer, stashes get their own section, and the usual keys work on the stash under the cursor: `RET` shows it, `a` applies it, `k` drops it. In smallzine, the two leftover changes from the staging section went into a stash called "wider page, thicker links".

Interactive rebase

Interactive rebase rewrites a stretch of history: reorder commits, merge them, fix their messages, drop the ones we don't want. In Magit it starts from the log or from Recent commits in the status buffer. Put the cursor on the oldest commit that should change and press `r i`. Unlike `git rebase -i <commit>`, Magit includes the commit at point, which matches how people actually think about it.



- `org-roam-alias-add`: extra names for the node. The Gemini note in the test collection has the alias "Project Gemini", and both names find it.
- `org-roam-tag-add`: tags, stored as Org tags.
- `org-roam-ref-add`: a URL or citation key that the node is about. Later, visiting that URL from a browser extension or a citation manager leads back to the note.

And two commands reshape notes as they grow: `org-roam-extract-subtree` turns a heading into its own file, keeping the ID, and `org-roam-refile` moves a heading into another node.

THE ORG-ROAM BUFFER

`C-c n l` (`org-roam-buffer-toggle`) opens a side buffer that follows the current node and shows its backlinks, each with the paragraph around the link:

Notice that one of the backlinks to Gopher comes from Pubnixes, which is a heading inside another file. That's node granularity at work: the link is attributed to the heading, not just the file. The buffer can also show reference links and unlinked references (places that mention the title as plain text).

DAILIES

org-roam-dailies, included with org-roam, keeps a file per day. `C-c n j` (`org-roam-dailies-capture-today`) adds an entry to today's file and `org-roam-dailies-goto-today` opens it. Since dailies are ordinary nodes, linking from a day's entry to a topic note makes that day show up in the topic's backlinks, which gives every topic a free timeline.

THE DATABASE, AND THE GRAPH

The SQLite database lives in `org-roam-db-location` (by default, `org-roam.db` inside the Emacs directory). It holds only derived data: deleting it and running `org-roam-db-sync` rebuilds everything from the Org files. It shouldn't be synced between machines; each machine builds its own.

For a visual map, org-roam-ui serves an interactive graph of the database in the browser, similar to Obsidian's graph view.

org-node

org-node is by Martin Edström, and it began as an answer to the slowness of org-roam on very large collections. It reads the same notes (files and headings with Org IDs, linked with `id:` links), but it indexes them with a companion package, org-mem, which scans the files in parallel background processes and keeps the results in memory. There's no SQLite and no database file.

The author's own benchmark, on his machine with 3,000 nodes, puts a full index at about 2 seconds with org-node against 2 minutes and 48 seconds with org-roam.[4] Day-to-day use doesn't rebuild everything, so the difference shows up most at startup and on large collections.

INSTALLING AND CONFIGURING

Both packages are on MELPA. The recommended config from the org-node manual:

```elisp
(use-package org-mem
:ensure t
:config
(setq org-mem-watch-dirs (list "~/notes/roam/"))
(org-mem-updater-mode))

(use-package org-node
:ensure t
:init
(keymap-global-set "M-o n" org-node-global-prefix-map)
(with-eval-after-load 'org
(keymap-set org-mode-map "M-o n" org-node-org-prefix-map))
:config
(org-node-cache-mode))
```

`org-mem-watch-dirs` lists the folders to index. `org-mem-updater-mode` keeps the index current as files are saved, and `org-node-cache-mode` connects org-node to it. Everything is under `M-o n`, which avoids clashing with Denote or org-roam on `C-c n`.

FINDING AND LINKING

`org-node-find` lists every node, with headings shown under their file's title:

4 These are the author's numbers from the org-node README, measured on his own computer. Results on other machines and collections will differ, but the order of magnitude is the point.


```
Commits in --all
008267b * main Tweak intro text                          Ana Lima      1 week
9215578 * fixup! WIP footer style                        Ana Lima      2 weeks
f59b90b * WIP footer style                               Ana Lima      2 weeks
87dc823 * Add footer                                     Ana Lima      2 weeks
4dac370 | * feature/rss Announce the feed in the page head Rui Costa    2 weeks
3fc071e | * Add an RSS feed                             Rui Costa     2 weeks
        |/
116021e * Describe the zine in the README                Rui Costa     2 weeks
ac85d6a * Add build script                               Ana Lima      2 weeks
82843a4 * Link stylesheet from index                     Rui Costa     2 weeks
38f3ecb * Add stylesheet                                 Rui Costa     2 weeks
eb33829 * Add index page                                 Ana Lima      3 weeks
2ab17ba * Add README                                     Ana Lima      3 weeks
```

```
Ana Lima            2026-11-03 14:20 Add index page
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Small Zine</title>
Rui Costa           2026-11-04 19:12 Link stylesheet from index
<link rel="stylesheet" href="style.css">
Ana Lima            2026-11-03 14:20 Add index page
</head>
<body>
<h1>Small Zine</h1>
Ana Lima            2026-11-11 16:45 Tweak intro text
<p>Issue 1: gopher holes worth visting.</p>
Ana Lima            2026-11-10 10:00 Add footer
<footer>Made by hand, no trackers.</footer>
Ana Lima            2026-11-03 14:20 Add index page
</body>
</html>
```

Here is `l a` on smallzine, before the cleanup described below. The `feature/rss` branch forks off after "Describe the zine in the README", and on `main` there is a WIP commit and its `fixup!`, waiting to be tidied up:

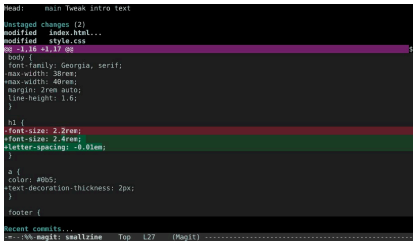
The log is a working buffer too. `RET` on a commit opens it with its full diff, and most commands that need a commit (reset, cherry-pick, rebase, fixup) take the one under the cursor as the default.

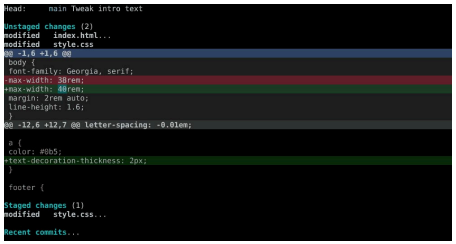
Blame

Blame answers "who changed this line, when, and in which commit". In a file buffer, `C-c M-g` opens the file menu and `b` runs blame (the command is `magit-blame-addition`). The file stays editable as a normal buffer, but each block of lines gets a header with the author, the date and the commit subject.

While making the screenshots, the intro paragraph of smallzine said "gopher holes worth visting", with a typo. Blame shows exactly where it came from:

The line came in with "Tweak intro text", from November 11. `RET` on a header opens that commit, `n` and `p` jump between blocks, and `q` leaves blame mode.





and with less context Git often splits one big hunk into several small ones that can be staged with a plain `s`.

Commits

`c` opens the commit menu. The command used most is `c c`: Magit opens a buffer for the message and, next to it, the diff of what is staged, so the message gets written while looking at the change. `C-c C-c` finishes the commit and `C-c C-k` cancels it.

The rest of the menu is about fixing commits that already exist:

- c a: amend the last commit, message and content
- c e: extend the last commit with what is staged, keeping its message
- c w: reword the last commit, changing only the message
- c f: create a "fixup!" commit aimed at an older commit
- c s: same, but a "squash!" commit, which also lets us edit the message later
- c F: instant fixup, which creates the fixup commit and immediately rebases it into its target

`c f` is the polite way to fix an old commit. Magit asks which commit the fix belongs to and records it as a separate commit whose message starts with `fixup!`. On the next interactive rebase with autosquash, Git moves it next to its target and melds the two[2]. `c F` does both steps at once, which is handy when the history hasn't been pushed and there's no reason to wait.

The log

`l` opens the log menu. `l l` shows the current branch, `l o` another branch, `l L` all local branches, `l b` all branches including remote ones and `l a` every reference. `l r` shows the reflog, the list of every place HEAD has been, which is the safety net when a rebase goes wrong.

2 Autosquash is a Git feature, not a Magit one. `git rebase -i --autosquash` looks for commits whose message starts with `fixup!` or `squash!` followed by another commit's subject, moves them right after that commit and sets their action.



The list includes "Project Gemini", read from the same `ROAM_ALIASES` property that org-roam uses. There's no separate command for new nodes: typing a name that doesn't exist into `org-node-find` or `org-node-insert-link` creates it.

Other commands cover the same ground as org-roam's: `org-node-refile`, `org-node-extract-subtree`, `org-node-rename-file-by-title` (renames the file to match the title), `org-node-grep` and `org-node-insert-heading` (a new heading that already has an ID).

BACKLINKS, TWO WAYS

org-node offers backlinks in two forms. `org-node-context-toggle` opens a side buffer, similar to the org-roam buffer. The other option, `org-node-backlink-mode`, is unusual: it writes the backlinks into the notes themselves, as a `:BACKLINKS:` property or drawer that updates when links change. The backlinks then exist in the files, visible to grep, to other programs and to anyone reading the notes without Emacs. The price is that files get edited automatically, which shows up in version control.

LIVING NEXT TO ORG-ROAM

Since the on-disk format is the same, org-node can index an org-roam collection as it is. For people who want to keep using org-roam's buffer and dailies, the manual suggests these lines:

```elisp
(setq org-mem-watch-dirs (list "~/notes/roam/"))
(setq org-node-creation-fn #'org-node-new-via-roam-capture)
(org-node-roam-accelerator-mode)
```

The first points org-mem at the org-roam folder. The second makes new nodes go through org-roam's capture templates. The third lets some org-roam commands use org-node's cache. A common path is to start this way and drop org-roam later, or keep both.

org-node doesn't cover everything org-roam does. It has no `roam:` links, and it doesn't separate collections into independent piles the way org-

roam can with per-directory settings. For maintenance it has two good tools: `org-node-lint-all-files` checks the notes for common problems, and `org-mem-list-dead-id-links` lists `id:` links that point nowhere.

Choosing one

Denote fits best when the files should outlive the tool. Nothing depends on a database, notes can be Markdown or plain text, and any file manager shows a readable index. It's the natural pick for a collection shared with Obsidian, a phone or a sync service, and for people who like small, well-documented packages. Its main limit is granularity: a note is a file, and links point to files (Org headings can be linked with extra setup, but it's not the default model).

org-roam fits when a heading-level graph and the existing ecosystem matter: capture templates, dailies, org-roam-ui, ref links, and years of guides and configs to copy from. On very large collections it gets slower, and the database is one more moving part.

org-node fits when org-roam's model is right but its speed isn't, or when the database is unwanted. It's also the easiest way to try a roam-style setup on top of existing Org files, since it doesn't need anything besides IDs. It requires Emacs 29.1, and its ecosystem is younger.

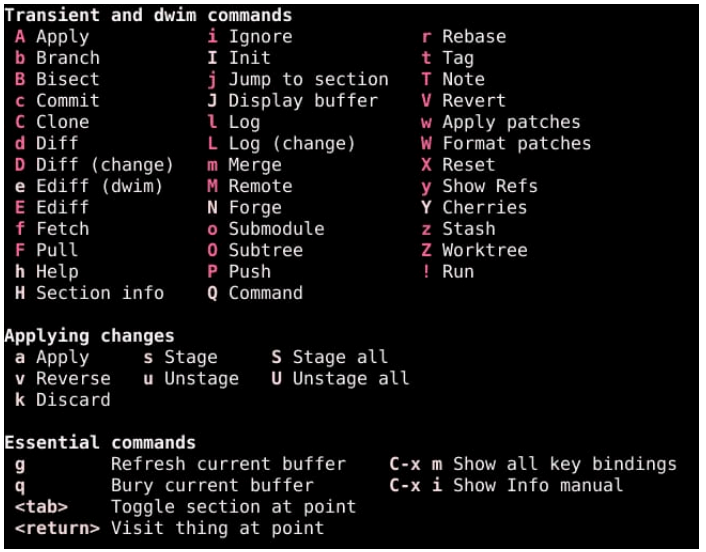
Moving between org-roam and org-node costs nothing, as they share the format. Moving between Denote and the other two means converting links (`denote:` to `id:`, or back), which is scriptable but takes some care.

Coming from Obsidian

Most Obsidian habits map directly:

- Daily notes become denote-journal or org-roam-dailies.
- Tags become Denote keywords or Org file tags.
- `[[wikilinks]]` become `denote:` or `id:` links, inserted with completion instead of typed by hand.
- The backlinks pane becomes `denote-backlinks`, the org-roam buffer or org-node's context buffer.
- The graph view becomes org-roam-ui (for org-roam and org-node collections).
- The quick switcher becomes `denote-open-or-create`, `org-roam-node-find` or `org-node-find`.

The biggest difference is renames. In Obsidian, renaming a note rewrites every link that points to it. In all three Emacs packages, links point to a permanent identifier, so renames don't need to touch other files at all.



Staging only part of a file

This is where Magit pays for itself. A file often holds two unrelated changes, and they deserve two commits. On the command line that means `git add -p` and answering y/n for each hunk. In Magit the diff is right there in the status buffer, and staging works on whatever is under the cursor:

- s: stage the file, hunk or region at point
- u: unstage it
- k: discard it (Magit asks first)

In smallzine, `style.css` had three changes: a wider page, a bigger `h1` with tighter letter spacing, and thicker link underlines. Only the `h1` change belonged in the next commit. Git saw the whole file as a single hunk, so staging the hunk would have taken everything. The trick is the region. Expand the file with `TAB`, put the cursor on the first line we want, press `C-SPC`, move down to the last one, and press `s`:

Only those lines go to the index. The status buffer redraws, and the file now shows up twice, once under Unstaged changes with the two changes we left behind and once under Staged changes:

Two details help here. The same region trick works in reverse: select lines in a staged hunk and press `u` to take only those back out. And when hunks are too big, `+` and `-` in a diff show more or fewer lines of context,

called *smallzine*, a tiny static zine with an `index.html`, a stylesheet and a build script, worked on by two made-up people, Ana and Rui.

Installing it

Magit is on MELPA and on NonGNU ELPA, so on a stock Emacs one line is enough:

```elisp
(use-package magit
:ensure t)
```

The current release is 4.7.1 (the release notes list what changed), and it needs Emacs 28.1 or newer. The last three releases came within two months of each other:

```timeline
2026-07-01 Magit 4.6
2026-07-31 Magit 4.7.0
2026-09-01 Magit 4.7.1
```

Out of the box Magit binds three global keys:

- C-x g: open the status buffer for the current repository
- C-x M-g: open the dispatch menu from anywhere
- C-c M-g: open the menu of commands for the current file (blame, log of this file, stage this file)

The status buffer and the `?` key

`C-x g` opens the status buffer, and almost everything starts there. At the top is the branch we're on and its last commit. Below come sections: untracked files, unstaged changes, staged changes, stashes, recent commits, and, when there is one, a merge or rebase in progress. `TAB` folds and unfolds the section at point, `g` refreshes and `q` buries the buffer.

The single most useful key while learning is `?`. It opens magit-dispatch, a menu of every command family and the key that opens it:

These menus are made with Transient, a library written for Magit. Each letter opens a second menu with the actual commands plus switches that map to Git's command-line flags. `c` opens the commit menu, `c c` creates a commit. `l` opens the log menu, `l l` shows the log of the current branch. With use, the common keys turn into muscle memory, and the menu is always one `?` away for the rest.

Habits that make any of them work

The package matters less than a few habits:

- **One idea per note.** Small notes link better than long ones. When a note keeps growing, split it (`org-roam-extract-subtree`, `org-node-extract-subtree`, or a new Denote note made with `denote-region`).
- **Link while writing.** The best moment to link two ideas is when the connection is fresh. Linking to a note that doesn't exist yet is fine; all three can create it on the spot.
- **Write the reason next to the link.** `See [[Gopher]]` says little. `It borrows the simplicity of [[Gopher]]` tells the future reader why the link is there, and that context shows up in the backlinks.
- **Keep keywords few.** A short, stable list of keywords works better than a new tag for every note. `denote-known-keywords` helps by suggesting the existing ones.
- **Back up the folder.** The notes are plain files, so any backup works. The 3-2-1 backups post has a full setup.

Cheat sheet

With the configs above.

Denote:

- C-c n n: create a note
- C-c n l: insert a link to a note
- C-c n b: show backlinks
- C-c n r: rename the current note
- C-c n d: filtered Dired listing
- C-c n g: search inside notes
- M-x denote-open-or-create: open a note, or create it if missing
- M-x denote-journal-new-or-existing-entry: today's journal entry

org-roam:

- C-c n f: find or create a node
- C-c n i: insert a link to a node
- C-c n c: capture into a node
- C-c n l: toggle the backlinks buffer
- C-c n j: capture into today's daily
- M-x org-roam-alias-add: add another name to the node
- M-x org-roam-extract-subtree: turn the heading at point into its own file

org-node:

- M-x org-node-find: find or create a node

- M-x org-node-insert-link: insert a link to a node
- M-x org-node-context-toggle: toggle the context (backlinks) buffer
- M-x org-node-extract-subtree: turn the heading at point into its own file
- M-x org-mem-list-dead-id-links: list broken id links
- M-o n: org-node's prefix map, for everything else

06 RECOMMENDATIONS · INFOSEC · CSS 21 NOVEMBER 2026

A blog worth losing time on: lyra.horse

I went to lyra.horse to read one post about SVG clickjacking and stayed for a lot more. A few favorites from Lyra's blog, from logic gates built out of SVG filters to a one-click Google Drive bug, a JavaScript-free post, and a fake millionaire story.



I opened lyra.horse to read one post, the one about SVG clickjacking, and lost a good amount of time there. It's the blog of Lyra, a security researcher who writes about infosec, programming and the web, and it's one of those places where every post sends me to the next one.

The site itself is a small joy: a mint green page, a list of posts, a big RSS button that says "RSS :3" and a footer that reads "It's 2026. All meows

08 EMACS · MAGIT · GIT 22 NOVEMBER 2026

Magit in depth: Git from inside Emacs

A hands-on tour of Magit, the Git interface inside Emacs: staging only some lines of a file, commits and instant fixups, log, blame, stash, interactive rebase, resolving merge conflicts with smerge and ediff, and Forge for issues and pull requests. Real screenshots and a cheat sheet.



Magit is a Git interface that lives inside Emacs. It covers almost everything the `git` command does, but instead of typing commands and reading their output, we work in buffers: the repository state is laid out as text, and a single key acts on whatever is under the cursor, be it a hunk, a single line, a commit or a stash. Git people call this kind of front end a porcelain[1], and Magit is one of the most complete ones around.

This post walks through the parts that make Magit worth learning even for people who are fine with the command line: staging only some lines of a file, reading the log, blame, stash, interactive rebase, resolving a merge conflict and Forge, the companion package that talks to GitHub and GitLab. Every screenshot comes from a small practice repository

1 Git's own documentation splits its commands into "plumbing", the low-level commands meant for scripts, and "porcelain", the ones meant for people. Front ends like Magit are porcelains built on top of the plumbing.

- **Shared.** The same bot already posts my server alerts and notifications from my blog's CMS. Those only send. Telegram hands each incoming update to a single reader, so only the talk daemon reads updates, and it only accepts posts from that one channel.
- **No webhook.** The daemon uses long polling: `getUpdates` with a 50-second timeout, one HTTP request every 50 seconds while nothing happens. A webhook would need another public HTTPS endpoint; polling doesn't.
- **One message at a time.** Everything the bridges send goes through a queue, with a second between messages. If Telegram answers 429 (too many requests), the daemon waits the `retry_after` time Telegram asks for and tries again. With alerts, the CMS and two bridges on one bot, this keeps them from tripping over each other.
- **Credentials.** The token and the channel ID live in one env file readable only by root. systemd's `LoadCredential` hands each service its own copy at start,[3] so neither the talk daemon nor the Misfin script can read the original.

Try it

```
$ finger hi@pablomurad.com
$ talk hi@pablomurad.com
$ misfin send-as you.pem hi@pablomurad.com "hello from the small web"
```
CONSOLE

For the last one, `you.pem` is your own Misfin identity certificate. And to finger me from a phone:

```
https://nullroute.lt/finger/?q=hi%40pablomurad.com
```
QR



3 The service reads the copy from a private directory that systemd creates when it starts the unit and removes when it stops.


lyra's epic blog posts tags

SVG clickjacking

2025-12-04 ¦ infosec ¦ css

Previously titled: "SVG Filters - Clickjacking 2.0"

Clickjacking is a classic attack that consists of covering up an iframe of some other website in an attempt to trick the user into unintentionally interacting with it. It works great if you need to trick someone into pressing a button or two, but for anything more complicated it's kind of unrealistic.

I've discovered a new technique that turns classic clickjacking on its head and enables the creation of complex interactive clickjacking attacks, as well as multiple forms of data exfiltration.

I call this technique "**SVG clickjacking**".

[get pixel color at (567,178)]

win free ipod

Are you sure?

click here

Everybody will be able to see your secrets.

No

Yes

[show overlay image #3]

Liquid SVGs

The day Apple announced its new Liquid Glass redesign was pretty chaotic. You couldn't go on social media without every other post being about the new design, whether it was critique over how inaccessible it seemed, or awe at how realistic the refraction effects were.

reserved." No pop-ups, no newsletter modal, just articles. Below are the ones that stuck with me, with a short note on each.

SVG clickjacking

This is the one that brought me there, and it's a great example of her style: a serious finding, explained with interactive demos right on the page.

Clickjacking is an old trick. A site loads another site in an invisible frame and puts a decoy on top, so a click meant for the decoy lands on a button underneath.[1] It usually works for one or two clicks, because the attacker can't see what's happening inside the frame.

Lyra found a way around that limit. SVG filters, the same ones used for blur and color effects, can be applied to a frame from another site. With the right combination of filters she could read the color of pixels inside that frame, compare them and even build logic gates out of them, all inside the filter. That means a page can react to what the hidden site is showing and guide someone through a multi-step flow, changing the decoy as it goes.

She proved it against Google Docs and got a $3,133.70 bounty from

1 The name comes from "click hijacking". Sites defend against it by refusing to be loaded in frames on other sites, but many pages still allow it.

Google for it.[2] The post also has a working calculator made only of SVG filters and a recreation of Apple's Liquid Glass effect, which is a strange and wonderful thing to find in the middle of a security writeup. Some demos only work in Chromium-based browsers, so it's worth opening the post in one.

Using YouTube to steal your files

This one reads like a heist. None of the bugs in it is scary alone: a path traversal in how Google Slides builds YouTube embed links, a YouTube sign-in redirect that goes anywhere on YouTube, another redirect that hops to Google Docs, an old URL format that slips past a framing protection, and a share dialog that can be pre-filled through the address.

Chained together, they made a one-click attack: a page dressed up as a Google Form where a single click on "Send" would give an attacker edit access to someone's Drive files. Google paid $4,133.70. The lesson I take from it is that "low severity" bugs don't stay low severity when someone patient lines them up.

You no longer need JavaScript

A tour of what modern CSS can do on its own: tabs and toggles made with `:has()` and radio buttons, accordions with `details`, light and dark themes with `light-dark()`, entry animations with `@starting-style`, form validation styling, and viewport units that finally behave on phones.

The best detail is that the post practices what it preaches. The whole article, demos included, is a single HTML file with no JavaScript, no images and no external resources, at around 49 kB compressed. There's even a color picker and a little clicker game written in pure CSS. She doesn't argue for throwing JavaScript away, only for using it where it's actually needed. For anyone who likes light pages, it's a very good read.

How I accidentally became a millionaire

Her newest post, from November 2026, and the funniest. Through a mistake in Google's bug tracker, she had edit access to the bounty field of one of her own reports. As a joke, she changed it to $1,333,337 and tweeted a video of it.

What happened next is the real story. An AI-written Community Note got the facts wrong and then vanished. An automated AI news site

2 Google's bug bounty program likes amounts ending in 1337 and 31337, a nod to "leet", old hacker slang for "elite".

I run the Misfin server by cipres (`cipres/misfin` on GitLab), version 1.3.8. It ships as an AppImage, and since my server has no FUSE, I extracted it instead of running it:

```sh
./misfin.AppImage --appimage-extract
mv squashfs-root /opt/misfin/app
printf '#!/bin/sh\nexec /opt/misfin/app/AppRun "$@"\n' > /usr/local/bin/
misfin
```

It runs as a system user `misfin` from `/var/lib/misfin/srv`. `misfin server-init` creates the server certificate for the domain, the authority that signs mailboxes, and one more command creates mine:

```sh
misfin cert-from server.pem hi "Pablo Murad" identities/hi.pem
```

The config, `misfind.toml`, listens on the public address, has rate limits for unknown and verified senders, and keeps sign-up closed, since this server has exactly one mailbox. Mail is stored as a maildir, one file per message, and the server also has a Gemini front end on `gemini://pablomurad.com:1958`.

Since the Telegram bot was already there, new Misfin mail lands in the same channel. A systemd path unit watches the mailbox folders with inotify:

```ini
[Path]
PathChanged=/var/lib/misfin/srv/gemboxes/<mailbox>.maildir/new
PathChanged=/var/lib/misfin/srv/gemboxes/<mailbox>.maildir/cur
Unit=misfin-telegram.service
```

When a file appears, it starts a one-shot script as the `misfin` user, with the Misfin folder mounted read-only. The script reads each message (MIME on the outside, gemmail inside: a `<` line with the sender, a `:` line with the recipients, an `@` line with the date, then the text), posts the sender, date and text to the channel, and notes which files it already sent. It never moves or marks anything, so the message stays unread in my mail client. This side only notifies; answering Misfin from Telegram isn't built.

The bot behind all of it

Everything above talks to Telegram through one bot.

- **Setup.** The bot was made with BotFather, the Telegram account that creates bots, and it's an admin of a private channel. A bot has to be an admin to post in a channel, and admins also receive the channel's posts as updates, which is how the bridge sees my replies.

A call nobody answers becomes a missed call after 10 minutes, and a conversation that goes quiet for 30 minutes is closed.

LIMITS

- Only `hi` answers. Any other name gets the normal “not logged in” reply.
- Each address can only ring so often, and only a few calls can ring at the same time, so a script can’t flood the channel.
- Invitations are only accepted from the server itself. They exist so someone logged in here can still call out with `talk user@elsewhere`.

WHAT DOESN’T WORK

Talk comes from a time when every computer had a public address. The caller’s program announces its own IP and waits for me to connect to it. Behind a home router that IP is private. The daemon falls back to the address the packet came from, but the router usually drops the incoming connection anyway. Calls from shell accounts on public servers, like runv.club or SDF, work.

RUNNING IT

The daemon is a systemd service with a throwaway user and only the capability it needs to bind a port below 1024:

```ini
[Service]
ExecStart=/usr/local/sbin/talk-telegram
LoadCredential=env:/etc/talk-telegram/env
DynamicUser=yes
AmbientCapabilities=CAP_NET_BIND_SERVICE
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
NoNewPrivileges=yes
ProtectSystem=strict
ProtectHome=yes
MemoryMax=64M
CPUQuota=20%
Restart=on-failure
```

The talk line in `/etc/inetd.conf` is switched off (`update-inetd --disable ntalk`), so inetd and the daemon don’t fight over port 518. Idle, the daemon sits at about 16 MB of memory and almost no CPU.

Misfin rides along

Misfin is a mail protocol from the Gemini community. Messages travel over TLS on port 1958, addresses look like `user@host`, and there are no passwords: your identity is a client certificate, signed by the server that hosts your mailbox. Messages are short gemtext documents.



turned the tweet into an article. Google’s AI Overview, ChatGPT and other assistants started answering questions about the biggest Chrome bounty with her name. Even AI summaries that cited her own correction kept repeating the millionaire part. Google eventually set the bounty back to the original $1,000 and removed her access.

It’s light reading with a heavy point: a joke went through social media, search and AI summaries in a few days and came out the other side as a “fact”, without anyone intending harm. Someone who did intend harm could use the same path.

And more

There’s plenty more on the blog: a look at Reddit’s anti-spam system after it leaked by accident, a beginner-friendly walk through exploiting Chrome’s V8 engine in a competition, and what someone could do with a borrowed phone and ten seconds in Telegram. Her home page has other projects too.

If any of this sounds interesting, the RSS feed is the easiest way to catch the next post. ■

$ cd ~pablo

07 FINGER · TALK · MISFIN 22 NOVEMBER 2026

How finger and talk work at hi@pablomurad.com

How finger and talk work at hi@pablomurad.com: a shell script that only answers for one user and knows when I'm online, a talk daemon that rings my phone through Telegram, and Misfin mail landing in the same channel. With the files and real screenshots.



A quick note first: pico, which shows up a lot below, is a POSSE project of mine (publish on my own site, syndicate elsewhere), made for my own personal use.

Two commands from a Unix shell reach me on this server:

```
CONSOLE
$ finger hi@pablomurad.com
Login: hi Name: Pablo Murad
Directory: /home/hi Shell: /bin/bash
On since Sat Nov 21 22:15 (-03) on pico
10 minutes idle
Project:
Independent web projects, decentralized networks, Unix and
experimental Internet protocols.
Plan:
Building runv.club, a small public Unix server from Brazil.
$ talk hi@pablomurad.com
```


terminal (tmux doesn't register the login the way talkd looks for it[2]), with `mesg y` on. In practice I never was.

Talk, ringing on Telegram

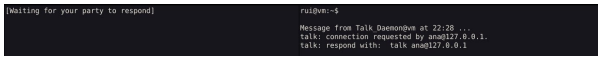
So I replaced talkd with a small daemon of my own, `talk-telegram`: about 300 lines of Python using only the standard library, asyncio for the sockets and urllib for Telegram. It speaks ntalk on port 518 like talkd does, but instead of writing to a terminal, it rings a private Telegram channel.

```
DOT
digraph {
rankdir=TB
node [shape=box, style=rounded, fontsize=11]
edge [fontsize=9, arrowsize=0.6]
caller [label="caller's talk"]
daemon [label="talk-telegram\nUDP 518"]
tg [label="private\nTelegram channel", color="#d93a2b", fontcolor="#d93a2b"]
caller -> daemon [label="ANNOUNCE"]
daemon -> tg [label="talk from user@host"]
tg -> daemon [label="my replies", color="#d93a2b", fontcolor="#d93a2b"]
daemon -> caller [label="TCP: my text"]
caller -> daemon [label="their keystrokes"]
daemon -> tg [label="their lines"]
}
```

A CALL, STEP BY STEP

- **Ringing.** The first ANNOUNCE for `hi` creates a call and posts a message to the channel with the caller's name and host, and how to answer. The repeated ANNOUNCEs every 30 seconds only keep the call alive. They don't post again.
- **Answering.** I reply to that Telegram message. The daemon remembers which message belongs to which call, so my reply finds the right one. It connects to the caller's TCP port, sends its three edit keys, reads theirs, and writes my text to their screen. Every reply after that goes straight through.
- **Their side.** The caller's keystrokes arrive one byte at a time. The daemon keeps a line buffer and applies their own erase, kill-line and erase-word keys, so a typo fixed with backspace never reaches Telegram. It sends a message on Enter, or after 3 seconds without typing.
- **Hanging up.** Replying `/fim` (Portuguese for "end") closes the connection. If the caller leaves first, the channel gets a "hung up" message.

2 talkd finds the person being called in utmp, the system's list of who is logged in and on which terminal. A login that never makes it into utmp is invisible to talkd, even when the person is sitting right there.





KNOWING WHEN I'M AROUND

A classic fingerd prints "On since" when the user is logged in to a terminal. I'm almost never logged in as `hi`, so the old header said "Never logged in." to everyone, forever.

Now the generated plan file doubles as a heartbeat. If pico rewrote it less than 90 minutes ago, the header says `On since <time> on pico` and how many minutes ago that was. After that it switches to `Last login <time>`. Nothing runs in the background to keep this fresh: the script only reads the file's modification time at the moment someone fingers me.

Talk

Talk is the split-screen chat from BSD Unix. The screen is cut in half, and each person watches the other type letter by letter, typos and backspaces included. Behind it there are two connections, one to set up the call and one to carry the text:

1. The caller's `talk` asks my talk daemon (UDP port 518, the "ntalk" protocol[1]) whether I left an invitation for them. I didn't.
2. Their program opens a TCP port, leaves an invitation with its own local daemon, and sends my daemon an ANNOUNCE message with that port. It repeats the ANNOUNCE every 30 seconds or so ("Ringing your party again") until someone picks up.
3. A normal talkd then prints a "Message from Talk_Daemon" banner on my terminal. I'd run `talk` back, my program would find their invitation and connect to that TCP port.
4. Each side sends three bytes, its erase, kill-line and erase-word keys, and from then on every keystroke goes straight to the other screen.

A test call between two accounts on a scratch machine shows both halves of step 3. On the left, the caller waits; on the right, the banner tells the other person how to answer:

And once the call is up, each window splits in two, with your own words on top and the other person's below:

My first version was the stock setup: `inetutils-talkd` started by inetd, port 518/udp open. To receive a call I had to be logged in over SSH, in a real

1 There's an older talk protocol on UDP port 517 that can't speak to ntalk. Debian's `inetd.conf` still lists both, one line each.

The first prints a short page about me: what I'm working on, my latest notes and posts, and whether I'm around. The second rings me for a live chat, letter by letter, in the terminal. Both protocols are older than the web, and both have been running here since August 2026 on a Debian 13 server. In November they learned two new tricks: finger now knows when I'm online, and a talk call rings my phone through Telegram.

This post walks through how each one works, with the files.

```
1971 Finger is born at Stanford
1983 Talk ships with 4.2BSD
1991-12 RFC 1288, the finger still in use
2026-08 Finger and talk go up at hi@pablomurad.com
2026-11 Online status, and talk rings Telegram
```
TIMELINE

One address: a user called hi

Finger, talk and Misfin (a small mail protocol from the Gemini world, more on it below) all use addresses shaped like email: `user@host`. I wanted the three to be the same, so the server has a Unix user called `hi`, and that's the only one anybody sees. My admin account exists, but finger doesn't show it and talk doesn't answer for it.

```sh
useradd -m -s /bin/bash hi
chmod 711 /home/hi
```
SH

Finger

Finger is a plain TCP service on port 79, described in RFC 1288. The client connects, sends one line with a user name, and the server answers with text and hangs up. No special client is needed, `nc` is enough:

```sh
printf 'hi\r\n' | nc pablomurad.com 79
```
SH

Without a terminal at hand, the web gateway at nullroute.lt shows the same page:

THE DAEMON

Debian's `fingerd` reads the classic `~/.plan` and `~/.project` files. I started with it, then swapped it for a short shell script, so it only answers for `hi` and I control the header.

inetd (from the `inetutils-inetd` package) starts the script for each connection. This is its line in `/etc/inetd.conf`:

```
finger stream tcp nowait nobody /usr/local/sbin/fingerd-hi-only
```

```
finger hi@pablomurad.com

[pablomurad.com]
Login: hi                              Name: Pablo Murad
Directory: /home/hi                    Shell: /bin/bash
On since Sat Nov 21 22:15 (-03) on pico
     13 minutes idle
Project:
Independent web projects, decentralized networks, Unix and
experimental Internet protocols.
Plan:
Building runv.club, a small public Unix server from Brazil. There's
room for more people: https://runv.club

      ______
    .-"      "-.
   /           \
  |,  .-.  .-.  ,|
  | )(_o/  \o_)( |
  |/     /\     \|
  (_     ^^     _)
   \__|IIIIII|__/
    | \IIIIII/ |
    \          /
     `--------`

Hi, I'm Pablo Murad, in Brazil. I write at pablomurad.com about
self-hosting, Unix, and the small web. I run my own gemini capsule and
gopher hole, mirror every post on both, and keep a few old protocols
running.
```

The script runs as `nobody`, reads the query line and builds the answer:

```sh
#!/bin/sh
export LC_ALL=C
PLAN=/var/finger/plan
ONLINE_MIN=90

read -r line 2>/dev/null || line=""
q=$(printf '%s' "$line" | tr -d '\r\n' | sed 's#^/W##; s/^[[:space:]]*//' |
awk '{print tolower($1)}')

if [ -n "$q" ] && [ "$q" != "hi" ]; then
printf 'This host publishes one Finger address: hi@pablomurad.com\r\n'
exit 0
fi

printf '%-38s %s\n' "Login: hi" "Name: Pablo Murad"
printf '%-38s %s\n' "Directory: /home/hi" "Shell: /bin/bash"

seen=$(stat -c %Y "$PLAN")
idle=$(( ($(date +%s) - seen) / 60 ))
when=$(date -d "@$seen" '+%a %b %e %H:%M (%Z)' | tr -s ' ')
if [ "$idle" -lt "$ONLINE_MIN" ]; then
printf 'On since %s on pico\n' "$when"
[ "$idle" -ge 1 ] && printf '     %d minutes idle\n' "$idle"
else
printf 'Last login %s on pico\n' "$when"
fi

printf 'Project:\n'; cat /home/hi/.project
printf 'Plan:\n'; cat /home/hi/.plan-top; printf '\n'; cat "$PLAN"
```


A few details in there:

- An empty query (`finger @pablomurad.com`) gets the `hi` page too.
- `/W` is the "verbose" prefix from the RFC. The script just drops it.
- `LC_ALL=C` keeps the dates in English. Without it, the server's locale leaks into the output.
- The firewall needs one rule: `ufw allow 79/tcp`.

THREE FILES, TWO OWNERS

The page is stitched together from three files. Two are mine, one belongs to a program:

```cssv
---
/* cssv:title Three files, two owners */
/* cssv:description The files the finger script prints, in order, with who
writes each one and when. */
table {
  --cssv-key: file;
  width: 100%;
  font: 15px/1.45 source-serif-4, Georgia, serif;
  color: #111;
}
thead th { padding: 0 0.8em 0.4em 0; font: 600 11px/1.5 ibm-plex-mono,
ui-monospace, monospace; text-transform: lowercase; color: #666; border: 0;
border-bottom: 1px solid #111; }
td { padding: 0.6em 0.8em 0.6em 0; border: 0; border-bottom: 1px solid
#e6e6e6; }
td[data-col="file"] { font: 600 13px/1.5 ibm-plex-mono, ui-monospace,
monospace; white-space: nowrap; }
td[data-col="written by"], td[data-col="updated"] { font: 12px/1.6 ibm-plex-
mono, ui-monospace, monospace; color: #444; }
td[data-col="written by"]::before { content: "●\2002"; color: #111; }
tr[data-key="/var/finger/plan"] td[data-col="file"], tr[data-key="/var/
finger/plan"] td[data-col="written by"]::before { color: #d93a2b; }
@media (max-width: 30rem) {
  td[data-col="file"] { white-space: normal; }
}
---
file,written by,updated,what's in it
~/.project,me,by hand,"The Project line, about what I usually write"
~/.plan-top,me,by hand,"The first lines under Plan: right now, runv.club"
/var/finger/plan,pico,"about once an hour, over SSH","The ASCII skull, a
short bio, NOW, latest posts and other addresses"
```

pico is the tool I use to publish everywhere, and it rebuilds the third file on its own. Because the script prints my two files around the generated one, pico can rewrite its part as often as it likes without ever touching mine.