The Finger Protocol Renaissance

Exploring a forgotten corner of the network: RFC 1288 and the Finger protocol.

Finger is still useful for local status checks, .plan messages, and retro-style user information displays. It is a simple protocol that provides a small glimpse into classic Unix network culture.

Download the Source

```
wget ftp://ftp.uk.linux.org/pub/linux/Networking/netkit/bsd-finger-0.17.tar.gz
```

Extract

```
tar xzf bsd-finger-0.17.tar.gz
cd bsd-finger-0.17/finger
```

Apply a time.h Patch to Fix Compilation

```
sed -i '1i #include <time.h>' lprint.c
sed -i '1i #include <time.h>' sprint.c
```

Compile

```
make CFLAGS="-std=gnu89"
```

Install

```
sudo mkdir -p /usr/man/man1
sudo make install
```

Example ~/.plan Content

```
echo "Tinkering with TinyCore Linux and breaking things on purpose." \
        > ~/.plan
```

Run Finger Locally

```
finger tc
```

Example Output

```
Login: tc                               Name: Linux User
Directory: /home/tc                     Shell: /bin/sh

On since Tue Mar 17 09:26 (CET) on pts/1 from 192.168.1.129

1 second idle
(messages off)
```

Mail last read Tue Mar 17 06:20 2026 (CET)

Plan:
Tinkering with TinyCore Linux and breaking things on purpose.

Notes

The Finger protocol was defined in RFC 1288 and was widely used on Unix systems for publishing user information.

Modern systems rarely expose Finger services publicly, but the protocol remains useful for local environments, testing, and nostalgic exploration of classic Unix networking.

Ghostscript to the Rescue

Ghostscript is a powerful command-line tool for manipulating PDF documents. Here are a few useful commands that come in handy for everyday work.

Append a Single PDF to an Existing PDF

```
#!/bin/bash

gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite \
        -sOutputFile=final.tmp.pdf \
        original.pdf single.pdf \
        && mv final.tmp.pdf final.pdf
```

Compress a PDF Using the "ebook" Output Profile

```
#!/bin/bash

gs -sDEVICE=pdfwrite -dCompatibilityLevel=1.4 \
        -dPDFSETTINGS=/ebook \
        -dNOPAUSE -dQUIET -dBATCH \
        -sOutputFile=compressed.pdf \
        original.pdf
```

Sort PDF Files Alphabetically and Merge Them

```
#!/bin/bash

gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite \
        -sOutputFile=merge_all.pdf \
        $(printf "%s\n" *.pdf | sort)
```

Notes

The first example appends a PDF to an existing document.

The second example reduces the file size by using Ghostscript’s "ebook" profile, which provides a good compromise between image quality and compression.

The third example sorts all PDF files in the current directory alphabetically before merging them into a single document.

Spam Blocking with the xt_recent Kernel module

The system loads a persistent blacklist during startup to block known spam sources. During boot, previously saved iptables rules are restored and each IP address from a text file is inserted into the kernel's xt_recent list named "spambot".

Boot Configuration

```
[ -f /mnt/sda/tinycore-mydata-backup/ip4tables.rules ] \
        && iptables-restore \
                < /mnt/sda/tinycore-mydata-backup/ip4tables.rules

[ -f /mnt/sda/tinycore-mydata-backup/ip6tables.rules ] \
        && ip6tables-restore \
                < /mnt/sda/tinycore-mydata-backup/ip6tables.rules

while read ip; do
        echo "+$ip" > /proc/net/xt_recent/spambot
done < /mnt/sda/tinycore-mydata-backup/spambot.txt
```

Required Files

```
ip4tables.rules
ip6tables.rules
spambot.txt
```

Example: spambot.txt

```
45.91.64.2
```

Notes

The iptables rules are restored automatically during system startup.

Each line in spambot.txt should contain a single IPv4 or IPv6 address. Every address is added to the kernel's xt_recent list named "spambot", making the blacklist persistent across reboots while keeping the list itself in a simple text file.

Quick Testing SOAP Web Services with Bats

Bats is a lightweight testing framework for Bash and can be used as a simple alternative to GUI-based SOAP testing tools like Postman or SoapUI.

It is useful for automated checks from the command line, CI systems, or cron jobs.

Installation

```
git clone https://github.com/bats-core/bats-core
sudo ./bats-core/install.sh /usr/local
```

Example Test

```
#!/usr/bin/env bats

setup() {
        ENDPOINT="https://mocus.dynv6.net/hello"
}

@test "helloVoid" {
        response=$(curl -s -X POST \
                -H "Content-Type: text/xml" \
                -H "SOAPAction: helloVoid" \
                --data '' \
                "$ENDPOINT")

        result=$(echo "$response" | \
                xmllint --xpath "string(//return)" - \
                        2>/dev/null)

        [[ "$result" = "Hello, world!" ]]
}
```

Run

```
bats test.bats
```

Why Bats?

- Minimal, no GUI
- Works everywhere
- Easy integration with CI or cron
- Plain shell scripting

Easy Monitoring of SYN-RECV Connections

I added a small script that checks how many connections stay in SYN-RECV. If the count goes above a threshold, the system emails me using s-nail. Cron runs it every 15 minutes.

Why This Matters

SYN-RECV happens before any service sees the connection. Too many of them can fill the kernel's backlog and slow down the system. It's usually just Internet noise, but a spike can signal a SYN flood.

Script

```
#!/bin/sh

SYN_COUNT=$(ss -tuna | grep -c SYN-RECV)
THRESHOLD=50

if [ "$SYN_COUNT" -gt "$THRESHOLD" ]; then
        echo "High SYN-RECV count detected: $SYN_COUNT at $(date)" \
                | s-nail -s "Suspicious SYN threshold" tc
fi
```

Using s-nail as a Minimal MTA with a Custom sendmail

TinyCore has no MTA, so I compiled s-nail and let it handle SMTP directly. To satisfy programs that expect /usr/sbin/sendmail, I added a small stub that appends mail to a local mailbox file.

```
#!/bin/sh

MAILBOX="/var/mail/tc"
FROM="tc@tinycore"
DATE=$(date)
CONTENT=$(cat)

{
        echo "From $FROM $DATE"
        echo "$CONTENT"
        echo ""
} >> "$MAILBOX"
```

Cron jobs can report failures through s-nail:

```
... | s-nail -s "Job failed" tc
```

Calling s-nail or mailx

In .profile I display new mail at login:

```
mailx -H | grep '^>N' || echo "No emails in mailbox"
```

Exploring the Charm of X-Face Headers

Inspired by Mick's article "Custom Headers in Claws Mail" (https://baldric.net/2023/03/23/custom-headers-in-claws-mail/), I played with X-Face again just for fun. A few mail clients, like Claws Mail, still display these tiny portraits, although most modern mail applications ignore them. Even if X-Face has mostly disappeared, experimenting with it again is a nostalgic reminder of how inventive early Internet tools were.

matrix.dat

```
3 3
0.0  -1.0   0.0
-1.0  5.0  -1.0
0.0  -1.0   0.0
```

Creating the X-Face Header

```
#!/bin/bash

jpegtopnm images.jpg \
        | pnmscale -xsize 48 -ysize 48 \
        | pnmconvol -matrixfile matrix.dat \
        | ppmtopgm \
        | pgmtopbm -fs \
        | pbmtoxbm \
        | compface
```

Notes

X-Face is a small monochrome image format originally designed for displaying user portraits in email headers. The format was popular in Unix mail environments and Usenet during the early Internet era. Today only a few mail clients still support it, but the simplicity of the format makes it an interesting piece of computing history.