Small snippets worth sharing: A collection of tricks to solve common problems

These are tools and tricks I use regularly which are too small to give them full articles but too useful not to describe them.

Inhaltsverzeichnis

Calculate the CSP script-src hash for an inline script-tag

```
echo -n 'var inline = 1;' > /tmp/foobar
sha256sum /tmp/foobar | cut -d " " -f 1 | xxd -r -p | base64
```

For background, see mdn: Content-Security-Policy/script-src.

Build Freenet with Java 8 on Guix

```
guix environment --ad-hoc icedtea:jdk -- \
  bash -c 'ssh kav freenet/run.sh stop; \
    ./gradlew --no-daemon clean build -x test && \
    for i in freenet.jar freenet.jar.new; do \
      scp build/libs/freenet.jar kav:freenet/$i; \
    done; \
    ssh kav freenet/run.sh start'
```

jump between versions within a filesystem tree

```
# start ~/path/to/source/branch-7.9/src/path/to/folder
cd $(echo $PWD | sed s/7.9/master/)
# now at ~/path/to/source/branch-master/src/path/to/folder
```

I use this regularly at work to avoid deep navigation. Typically via `C-r 7.9/master` —
and for the reverse `C-r master/7.9`.

Optimize bash defaults: increase history size, update history instantly, share history

This is essential to re-use commands without typing them.

Add the following to `~/.bashrc`:

```
# better bash history handling
# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
export HISTSIZE=100000
export HISTFILESIZE=1000000
# append to the history file, don't overwrite it
shopt -s histappend
# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
export HISTCONTROL=ignoredups:erasedups
# update the history with every command, not only at exit
export PROMPT_COMMAND="history -a;$PROMPT_COMMAND"

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "**" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
shopt -s globstar
```

Activate readline and colors in the Guile REPL

To be enjoyable to use interactively, Guile requires readline and colors.

Just put the following in `~/.guile`:

```
(cond ((false-if-exception (resolve-interface '(ice-9 readline)))
       =>
       (lambda (module)
         ;; Enable completion and input history at the REPL.
         ((module-ref module 'activate-readline))))
      (else
       (display "Consider installing the 'guile-readline' package for
```

```
convenient interactive line editing and input history.\n\n")))

        (unless (getenv "INSIDE_EMACS")
          (cond ((false-if-exception (resolve-interface '(ice-9 colorized)))
                 =>
                 (lambda (module)
                   ;; Enable completion and input history at the REPL.
                   ((module-ref module 'activate-colorized))))
                (else
                 (display "Consider installing the 'guile-colorized' package
for a colorful Guile experience.\n\n")))))
```

optimize scanned image for homework in Gnome

With this trick you get a right-click menu in Nautilus (Gnome file manager) that optimizes a scanned file for sending as homework assignment result.

Save the following as `~/.local/share/nautilus/scripts/optimize-scan-for-homework` and run `chmod +` `~/.local/share/nautilus/scripts/optimize-scan-for-homework`.

```
#!/run/current-system/profile/bin/bash

# This script makes a scanned image suitable (=small enough) for
# sending as homework assignment by replacing almost-white pixes from
# scans by white pixels and then running pngquant

# thanks for the urldecoder goes to to https://stackoverflow.com/a/37840948
# license: cc by-sa (as this is stackoverflow)
urldecode() { : "${*//+/ }"; echo -e "${_//%/\\x}"; }
base="$(echo $(urldecode "$NAUTILUS_SCRIPT_CURRENT_URI") | cut -d / -f3-)"
while [ ! -z "$1" -a ! -e "$base/$1" ]; do shift; done
filename="$base/$1"

if [ -f "$filename" ] && [[ x"$(file -b --mime-type "$filename" | sed s,/.*,,)" = x'i
    COLORFILE="${filename%.*}-colors.${filename#*.}"
    if [ -f "${COLORFILE}" ]; then
        zenity --error --width 400 \
               --text "Temporary file ${COLORFILE} already exists, not overwriting" \
               --title "temp file exists: ${COLORFILE}";
    else
        convert "${filename}" -contrast-stretch 1%x80% "${COLORFILE}"
        pngquant --skip-if-larger --strip --speed 1 "${COLORFILE}" || zenity --error
                 --width 400 \
```

```
                    --text "running pngquant on ${COLORFILE} failed";
    fi
else
    zenity --error --width 400 \
                    --text "image optimization needs an image file,
but \n${filename}\n is not an image file.\n
Its mime type is $(file -b --mime-type "$filename")" \
                    --title "not an image: ${filename}"
fi
```

Evaluate website logs with goaccess

uses goaccess.

```
cp logs/access_log_2021*.gz /tmp/
cd /tmp/
gunzip access_log_2021-0*
cat access_log* > aggregated_log.log
goaccess --all-static-files --ignore-crawlers -f aggregated_log.log
```

Now hit the number of the part you're interested in, jump to next with `tab`.

Sort with `s`, expand with `enter` or `space` or `o`, scroll down with `page down` or `CTRL-f` and up with `page up` or `CTRL-b`.

Hit `?` for more info.

Example, unique visitors per day, ordered by number of visitors:

```
Dashboard - Overall Analyzed Requests (01/Jan/2021 - 10/Mar/2021) [Active Panel: Visitors]

 Total Requests    3807168 Unique Visitors 135940 Unique Files 382099 Referrers  0
 Valid Requests    3807155 Processed Time  118     Static Files 6319   Log Size   865.34 MiB
 Failed Requests   13       Excl. IP Hits   0       Unique 404   24334  Bandwidth  449.37 GiB
 Log File          /tmp/aggregated_log.log

> 1 - Unique visitors per day - Including spiders                        Total: 69/69

Hits  Vis.     %   Bandwidth Data
----- ---- ----- ----------- ----
45687 2933 1.20%    6.42 GiB 17/Feb/2021 ||||||||||||||||||||||||||||||||||
64797 2851 1.70%    7.36 GiB 01/Mar/2021 ||||||||||||||||||||||||||||||||||||||||||
65280 2700 1.71%    6.00 GiB 02/Mar/2021 |||||||||||||||||||||||||||||||||||||||
55005 2560 1.44%    7.47 GiB 19/Feb/2021 |||||||||||||||||||||||||||||||||||||
65623 2416 1.72%    6.03 GiB 06/Jan/2021 ||||||||||||||||||||||||||||||||||||
71201 2339 1.87%    8.87 GiB 28/Feb/2021 |||||||||||||||||||||||||||||||||||||||||||
70002 2308 1.84%    5.40 GiB 03/Mar/2021 |||||||||||||||||||||||||||||||||||||||||||

  2 - Requested Files (URLs)                                            Total: 366/382099

Hits  Vis.    %   Bandwidth Mtd Proto    Data
----- ----- ----- ----------- --- -------- ----
58161 6429 1.53%    5.75 GiB GET HTTP/1.1 /rss.xml
863    544 0.02%    3.87 GiB GET HTTP/2.0 /files/2017-10-11-new-horizons-for-science-im-IM
627    384 0.02%    2.99 GiB GET HTTP/1.1 /files/2017-10-11-new-horizons-for-science-im-IM
```

To create a csv file:

```
goaccess --date-format='%d/%b/%Y' \
        --time-format='%H:%M:%S' \
        --log-format='%h %^[%d:%t %^] "%r" %s %b "%R" "%u"' \
        --max-items=99999999 --all-static-files --ignore-crawlers \
        -f /tmp/aggregated_log.log \
        -o /tmp/agg.csv
```

url-encode / url-decode

To encode unicode chars for a URI, just just save this as ~/.local/bin/url-encode and
make it executable

```
#!/usr/bin/env bash
exec -a "$0" emacs --batch --eval "(progn (require 'package) (package-initialize)
  (add-to-list 'package-archives '(\"melpa\" . \"https://melpa.org/packages/\"))
  (package-refresh-contents)(package-install 'urlenc) (require 'urlenc))" \
  --eval "(princ (urlenc:encode-string \""$@"\" urlenc:default-coding-system))" \
  --eval '(princ "\n")'
```

and this as ~/.local/bin/url-decode and make it executable

```
#!/usr/bin/env bash
exec -a "$0" emacs --batch --eval "(progn (require 'package) (package-initialize)
  (add-to-list 'package-archives '(\"melpa\" . \"https://melpa.org/packages/\"))
  (package-refresh-contents)(package-install 'urlenc) (require 'urlenc))" \
  --eval "(princ (urlenc:encode-string \""$@"\" urlenc:default-coding-system))" \
  --eval '(princ "\n")'
```

Usage:

```
url-decode $(url-encode '1,2+3!4')
# the single quotes prevent the '!' from mangling your command
```

Prepare this in one command:

```
echo '#!/usr/bin/env bash'"
exec -a \"\$0\" emacs --batch --eval \"(progn (require 'package) (package-initialize)
  (add-to-list 'package-archives '(\\\"melpa\\\" . \\\"https://melpa.org/packages/\\\
  (package-refresh-contents)(package-install 'urlenc) (require 'urlenc))\" \
  --eval \"(princ (urlenc:encode-string \\\"\"\$@\"\\\" urlenc:default-coding-system)
  --eval '(princ \"\\n\")'
" > ~/.local/bin/url-encode
echo '#!/usr/bin/env bash'"
exec -a \"\$0\" emacs --batch --eval \"(progn (require 'package) (package-initialize)
  (add-to-list 'package-archives '(\\\"melpa\\\" . \\\"https://melpa.org/packages/\\\
  (package-refresh-contents)(package-install 'urlenc) (require 'urlenc))\" \
```

```
  --eval \"(princ (urlenc:decode-string \\\"\"\"\$@\"\\\\\" urlenc:default-coding-system)
  --eval '(princ \"\\n\")'
" > ~/.local/bin/url-decode
chmod +x ~/.local/bin/url-{de,en}code
```

This is not the fastest command, because it spins up a full Emacs, but definitely faster
than opening a website — and privacy preserving.

dump all quassel irc channel logs

This uses the quassel dumplog script to extract all channel logs from a quassel db into
plaintext logs.

```
DB="path/to/quassel-storage.sqlite"
OUT="path/to/target-directory"
USER="user"
for i in $(python2 dumplog.py -u "${USER}" -d "${DB}"); do
    for j in $(python2 dumplog.py -u "${USER}" -d "${DB}" -n "$i" | head -n -2 | tail
        python2 dumplog.py -u "${USER}" -d "${DB}" -n "$i" -c "$j" -o "${OUT}"--"$i"-
    done
done
```

To work with logs from FLIP, I had to patch in some rudimentary error recovery:

```
diff -u dumplog-0.0.1/quasseltool.py dumplog-0.0.1/quasseltool.py
--- dumplog-0.0.1/quasseltool.py
+++ dumplog-0.0.1/quasseltool.py
@@ -252,7 +252,10 @@
             self.mynick = sender[0]
             return "\nSession Start: %s\nSession Ident: %s\n"%(self._now2(row[2]), s
         else:
-            return "%s *** %s (%s) has joined %s\n"%(self._now(row[2]), sender[0], s
+            try:
+                return "%s *** %s (%s) has joined %s\n"%(self._now(row[2]), sender[0
+            except IndexError:
+                return (str(sender) + str(row)).replace("\n", "----")

    def part(self, row):
        sender = row[3].split("!")
@@ -260,7 +263,10 @@
             self.mynick = ""
             return "Session Close: %s\n"%self._now2(row[2])
        else:
-            return "%s *** %s (%s) has left %s\n"%(self._now(row[2]), sender[0], sen
```

```
+            try:
+                return "%s *** %s (%s) has left %s\n"%(self._now(row[2]), sender[0],
+            except IndexError:
+                return (str(sender) + str(row)).replace("\n", "----")

     def quit(self, row):
         sender = row[3].split("!")
```

Diff finished.   Thu Aug 12 21:51:53 2021

Also see dropping old logs from quassel, but with adjustment: You get the time with

```
sqlite3 quassel-storage.sqlite
```

```
select strftime('%s','now','-90 day');
```

And get something like:

```
1621148264
```

Now you add three zeros and check what you'd get:

```
select * from backlog where time < 1621148264000;
```

As a sanity test, check whether there are messages from the future. Since you **stopped the quasselcore** before these tests (you did, right?), there should be none:

```
select strftime('%s','now','-1 second'); -- right now it is 1628924157
select * from backlog where time > 1628924157000;
```

Now you can create a backup and then drop everything older than 90 days:

```
cp quassel-storage.sqlite quassel-storage.sqlite.bak

-- First doublecheck again
select COUNT(*) from backlog where time > 1621148264000 ; -- newer messages: 748458
select COUNT(*) from backlog where time < 1621148264000 ; -- old messages: 25471749
-- now drop the old messages
delete from backlog where time < 1621148264000 ; -- careful: there is no going back.
-- check whether it worked
select COUNT(*) from backlog; -- 748458
-- actually free the diskspace, this saved 2.7GiB of diskspace for me
VACUUM;
```

this trick wasn't really a shell-trick, but rather a commandline trick. I'd rather have a not so narrow view here on the tricks here where it makes them more useful :-) .

loop over files with spaces by time, oldest first

```
SAVEIFS=$IFS
IFS=$(echo -en "\n\b")
for i in $(ls --sort=time -r)
do
  echo "$i"
done
IFS=$SAVEIFS
```

This adjusts IFS to avoid splitting by spaces. For more IFS tricks, see BASH Shell: For Loop File Names With Spaces.

[2021-09-17 Fr]

Update value in sqlite

This is here because I had to look it up to update my config for FLIP and FMS.

```
# accesss the database
sqlite3 flip.db3

-- get help
.help
-- see the tables in the database
.tables
-- activate table headers to see the names of the options
.headers on
-- see the values in tblOption (the columns are given in the headers)
select * from tblOption;
-- set the OptionValue for Option FCPPort
update tblOption set OptionValue = 9481 where Option = "FCPPort";
-- exit
.quit
```

[2021-10-13 Mi]

use xargs with ls: linebreak as separator

I often want to call some command for every file in a folder. Normal xargs fails for files with spaces in them.

`xargs -d "\n"` to the rescue: It uses the newline as argument separator instead of the space.

So executing a command on every file in a directory therefore only requires:

```
ls | xargs -d "\n" command
```

Typical usage: Shuffle-play the 30 newest videos in a folder with mpv:

```
ls --sort=time -c1 ~/path/to/videos/* | head -n 30 | xargs -d "\n" mpv --shuffle
```

You can extend this to arbitrarily complex command with `-I`:

```
cat ~/media-files-i-like.log \
    | xargs -d '\n' -I {} fd "{}" \
    | xargs -d '\n' -I {} cp "{}" media-i-like/
```

[2021-11-11 Do]

arrange multiple images on a page, keeping space around the images

```
pdfjam --trim="-1cm -1cm -1cm -1cm" true --nup 2x3 ...images
```

(optionally use `--landscape` for landscape view)

[2021-11-24 Mi]

multiprocessing made easy with xargs

A very common usecase: I want to run a command with many different arguments, but I want to run at most 32 processes at a time to avoid insta-OOMing my machine:

```
for i in {1..64}; do echo $i; done | xargs -P 32 -I % echo %
```

`-P 32` allows for up to 32 simultaneous processes.

`-I %` sets the input placeholder to %, so I can use `echo %` similar to `echo $i` in a direct loop.

[2021-12-03 Fr]

Stop and continue a process by PID (i.e. in another terminal)

So you started Emacs from the terminal, then activated exwm as window manager, switched back to the terminal and stopped Emacs with CTRL-z so your X-session is stopped and you cannot enter `fg` or `bg`? How to get it active again?

You can continue an arbitrary PID with kill — or by name with pkill:

```
pkill -CONT emacs
```

as mirror operation you can use the signal stop to pause any process by PID:

```
# hard stop
kill -STOP PID
# polite stop (keyboard stop, may be ignored)
kill -TSTP PID
```

sidenote: it did not actually restart the exwm for me, but stop and cont by PID should still come in handy.

[2022-01-05 Mi]

Transparently run script as root via sudo — if needed

If you have a script that needs to run as root and you want to ensure that users know that sudo will be run, you can add a header to conditionally run it with sudo:

```
if [ "$EUID" -ne 0 ]; then
  echo This script needs root priviledges. 1>&2
  echo Executing sudo --login $(realpath "$0") in 3 seconds 1>&2
  for i in {1..3}; do
      echo -n .
      sleep 1
  done
  echo " " now executing sudo and sudo --login $(realpath "$0")
  sudo echo || exit # ensure nicer exit if the user aborts
  exec sudo --login $(realpath "$0")
fi
# run your privileged code here.
```

[2022-03-27 So]

select specific characters from a string (splicing in bash)

You can use parameter expansion in bash to select a string by index and length (zero-indexed):

```
V=abcxyz
echo ${V:3:2}
```

```
xy
```

Use `${#var}-N` to index from the end.

```
V=abcxyz
echo ${V:${#V}-5:3}
```

Parameter expansion can do a LOT more. Do read up on it in the manual.

[2022-04-23 Sa]

Where does this command come from? (in Guix)

When I want to know which package brought a given command, I use `ls -l $(which prog)`, because Guix just symlinks the binaries from packages in `/gnu/store/<unreadable-hash>-pack` into ~/guix-profile/bin/

That has proven to be pretty useful more often than I expected.

[2022-06-10 Fr]

cron-job at randomly selected time: guard with random

If you want to run repeated jobs in a way that does not expose when your computer is active, you need some waiting. The usual tool for repeated jobs is cron, but waiting (`sleep seconds`) isn't a good idea there, because it can block other jobs.

A simpler method is to select multiple possible times of the day and then guard your cron-job with a test for random. Example:

```
# run at a random afternoon hour every day
0 14,15,16,17,18 * * * test 0 -eq $(($RANDOM % 5)) && date >> /tmp/randomly-selected-
```

This runs *on average* once per day, but it is not guaranteed to run exactly once per day to prevent providing information about the days your computer isn't running. It could skip one day and run three times during the next day; it could even run 5 times on one day, but on average it should run once per day.

If instead you want to run program exactly once per day but at a random time, you can use a cron-job that delegates the exact timing to at.

[2022-06-19 So]

sanitize filenames

Filenames with spaces and non-alphanumeric letters create problems in many contexts. This replaces all the dangerous letters by underscores but avoids overwriting existing files:

```
for i in *" "*; do
  if ! test -e "$(echo "$i" | sed 's/[^\.A-Za-z0-9/-]/_/g')"; then
    mv "$i" "$(echo "$i" | sed 's/[^\.A-Za-z0-9/-]/_/g')"
  fi
done
```

[2022-07-01 Fr]

unpack a *.deb (Debian package)

```
ar x PACKAGE.deb
tar xf data.tar.xz
tar xf control.tar.xz
```

[2022-10-05 Mi]

Gnuplot "fourliner"for mean and stddev of the difference between two datafiles

```
f(x) = mean_y
```

```
fit f(x) "< grep LINE results.Guile-guile | sed s/.*,// > /tmp/guile; grep LINE resul
```

```
stddev_y = sqrt(FIT_WSSR / (FIT_NDF + 1 ))
```

```
plot mean_y-stddev_y with filledcurves y1=mean_y lt 1 lc rgb "#bbbbdd", mean_y+stddev
```

From phyast.pitt.edu/~zov1/gnuplot/html/statistics.html.

Also see orgmode.org/worg/org-contrib/babel/examples/org-babel-gnuplot.html.

[2022-10-08 Sa]

Exit with explicit die

The function:

```
function die() {
    echo "${1}" 1>&2
    exit 1
}
```

Usage:

```
if [[ x"${ESSENTIAL_VARIABLE}" == x"" ]]; then
    die "Essential variable ESSENTIAL_VARIABLE is not set. Exiting."
fi
```

```
cat does-not-exist || die "Cannot cat file does-not-exist: does not exist. Exiting."
```

This enables clean error handling in bash scripts.

Using the function die is copied from the very clean scripts of Gentoo.

[2022-10-10 Mo]

Set an emergency static IP address from the commandline with ifconfig and route

If `dhcpcd` doesn't work and you just need connectivity to update your GNU/Linux, this is invaluable:

```
sudo ifconfig eth0 192.168.2.123 netmask 255.255.255.0;
sudo route add default gw 192.168.2.1 eth0;
```

Plug in an ethernet cable, replace `192.168.2.123` by the IP you want, `192.168.2.1` by the router IP and `eth0` by your device.

You can find your device by calling `ip link list`.

[2022-11-18 Fr]

relink hardlinks to eliminate overhead from duplicate files

If you have multiple copies of media files on your disk, you can save space by relinking hardlinks in all subfolders:

```
sudo hardlink -v -v -c -s 1m .
```

```
-v -v - very verbose (show all checked files)
-c    - only check the content, ignore access rights
-s 1m - ignore files smaller than 1 MiB
```

[2022-12-19 Mo]

Mediathek-Podcasts mit castget: keine Folge mehr verpassen

Um die Mediatheken der Öffentlich Rechtlichen angenehm nutzen zu können, sind die RSS-feeds von mediathekviewweb klasse: Sie ermöglichen es, alle Filme der Mediathek als podcast zu nutzen. Zusammen mit castget habe ich endlich einen guten Weg, die Öffentlich Rechtlichen unabhängig vom Browser zu nutzen. Beispiel: Ich will keine "Folge"von "Der Schwarm"im ZDF (`!zdf`) verpassen. Deswegen suche ich nach: `Der Schwarm Folge !zdf`

Datei `~/.castgetrc`:

```
[derschwarm]
url=https://mediathekviewweb.de/feed?query=Der%20Schwarm%20Folge%20!zdf
spool=/mnt/schatten/sonstiges/mediathek-downloads
```

Befehl (mit ad-hoc installation in Guix):

```
for i in $(grep -F '[' ~/.castgetrc | sed 's/\[//g;s/\]//g'); do
  guix shell castget -- castget -vrp $i;
done
```

Das ist, wie Fernsehen sein sollte. Danke ÖRR und danke MVW!

[2023-02-27 Mo]

Extract a highly compressed meme from video with ffmpeg

```
ffmpeg -i $INPUT_FILE -ss $START_SECONDS -to $STOP_SECONDS \
   -c:v libaom-av1  -crf 60 -c:a libopus -b:a 48k -g 999 \
   -lag-in-frames 25 -strict -2 -aq-mode 2 -tile-columns 3 \
   -tile-rows 3 -auto-alt-ref 1 \
   -threads 16 -cpu-used 6 \
   $OUTPUT_FILE.mp4
```

(requirements: ffmpeg with libaom and libopus)

This gets animated 1280x720 video with lots of movement down to 430kbits/s, so 100 seconds only require 5MiB of storage.

You get some visual artifacts in fast changing elements, but that's pretty awesome nonetheless.

And if you're lazy like me, you just define the variables as variables and copy the command. For example:

```
export INPUT_FILE=sintel.mkv START_SECONDS=20 STOP_SECONDS=120 OUTPUT_FILE=sintel-mem
COPY_OF_THE_COMMAND_ABOVE
```

To get even smaller, you can reduce quality, scale it down, or reduce audio-quality:

- `--crf 63` adds more encoding artifacts. Adding `-cpu-used 3` increases encoding time by about factor 3 compared to `-cpu-used 7`, but reduces artifacts again.

- `-filter:v scale=720:-1` reduces the width to 720 pixels, but you'll have to reduce `-tile-rows` to 2.

- `-b:a 48k` is usually safe, `-b:a 36k` sometimes produces artifacts.

Even with `--crf 63` you still get crisp text. *AV1 is awesome. And yes, I enjoy the speed of my new Ryzen CPU . . .*

Those together get you down to `3 MiB` for 100s (less than 240kbits/s).

Maximum video compression with ffmpeg

To encodue multiple videos in parallel in a for-loop, you need what you learned in the previous trick, and you also need nohup and subfolders. Example (replace `My_Video_Name` and `"mp4"`):

```
for Q in 62; do EXT="mp4" && time for i in My_Video_Name*.${EXT}; do
   (mkdir -p "$(basename "$i" .${EXT})";
    cd "$(basename "$i" .${EXT})";
    nice nohup ffmpeg -y -i "../${i}" \
         -c:v libaom-av1 -b:v 0 -crf $Q -aq-mode 2 -an  \
         -tile-columns 1 -tile-rows 1 -row-mt 1 -threads 12 \
         -cpu-used 8 -auto-alt-ref 1 -lag-in-frames 25 -g 999 \
         -filter:v scale=720:-1 \
         -pass 1 -f null /dev/null;
    nice nohup ffmpeg -y -i "../${i}" \
         -c:v libaom-av1 -b:v 0 -crf $Q -aq-mode 2 \
         -tile-columns 1 -tile-rows 1 -row-mt 1 -threads 12 \
         -cpu-used 3 -auto-alt-ref 1 -lag-in-frames 25 -g 999 \
         -c:a libopus -b:a 36k \
         -filter:v scale=720:-1 \
         -pass 2 \
         "$(basename "$i" .${EXT})"-av1-q${Q}.webm) &
done ; done
tail -F My_Video_Name*/nohup.out # to watch the progress
```

[2023-04-06 Do]

play a musical accord from the shell with midi and lilypond

```
echo "\\score {{<c, c c' e' g''>} \\midi{}}" \
  | lilypond -o /tmp/music --format=midi - \
  && timeout 2 timidity /tmp/music.midi
```

[2023-04-12 Mi]

Confining a program to specific CPU cores

To force a program to only run on some defined cores you can use taskset. For example:

```
taskset -c 2,3,5 <program> <program arguments>
```

If you want to use a random core to reduce power-consumption without stressing one core much more than the others on the long run, you can select that core at random.

For example for Yacy on a 32 core-machine:

```
YACYCORE=$(($RANDOM % 31)) # 32 - 1: zero-indexed
(cd /path/to/yacy_search_server/ \
    && (curl http://localhost:8090 2>/dev/null \
          | grep -q YaCy >/dev/null \
          || guix shell openjdk@17:jdk -- \
                nice -n 4 taskset -c $YACYCORE \
                ./startYACY.sh >/dev/null 2>&1))
```

[2023-05-15 Mo]

run command on all cores with parallel

With the example of transcoding every file to an mp3 into the subfolder `mp3/`.

```
mkdir -p mp3
for i in *.*; do
    sem --jobs 32 --id ffmpeg \
        "nohup ffmpeg -y -i \"$i\" \"mp3s/${i%%.*}.mp3\""
done
sem --id ffmpeg --wait
```

You cannot leave anything out.

Uses GNU Parallel.

[2023-05-18 Do]

Convert color values rgb hex

If you want to convert color values from the shell, you can install the convert-color-cli npm packages that brings with it 5 MiB of dependencies in 99 packages.

Or you can grab a beautiful shell-script from Arch and turn it into a function:

```
function hexconvert () {
    if [ $# -eq 0 ]; then
        echo missing color value
        echo "Usage: $0 [HEX] or [RGB] color value"
        echo ""
        echo "Example HEX to RGB: $0 0000ff"
        echo "Example RGB to HEX: $0 0,0,255"
        echo ""
        exit 1
    fi
    if [[ $1 =~ ([[:xdigit:]]{2})([[:xdigit:]]{2})([[:xdigit:]]{2}) ]]; then
        printf "(%d, %d, %d)\n" \
               0x"${BASH_REMATCH[1]}" 0x"${BASH_REMATCH[2]}" 0x"${BASH_REMATCH[3]}"
    elif [[ $1 =~ ([[:digit:]]{1,3}),([[:digit:]]{1,3}),([[:digit:]]{1,3}) ]]; then
        printf "#%02x%02x%02x\n" \
               "${BASH_REMATCH[1]}" "${BASH_REMATCH[2]}" "${BASH_REMATCH[3]}"
    fi
}
hexconvert c0ffee
hexconvert 250,202,222
```

(choose your own hexword)

[2023-07-03 Mo]

Get a vanilla Debian docker shell

For packages that don't work on your distro of choice:

```
docker pull debian && docker run -it debian
```

[2023-08-19 Sa]

Delete old docker data (prune)

Docker keeps volumes indefinitely, so if you regularly pull up databases, they can take up significant space on root. For me it took 275 GiB. To get rid of them:

```
docker volume prune
```

(can delete data)

There are also prune commands for `image` and `container`.

For a dangerous full prune that throws away all docker images, use

```
docker images -q | xargs docker image rm --force && \
docker images prune && \
docker container prune && \
docker volume prune
```

[2023-08-28 Mo]

allow modern tar format in automake / autoconf

GNU autoconf / automake tipp:

```
# tar-ustar: use ustar format of tar (POSIX 1003.1-1988) to lift the
# 99 character limit on filenames (it is now 155 for the directory and
# 256-dir for the file). Needs Automake 1.9 or newer
AM_INIT_AUTOMAKE([1.9 tar-ustar])
```

This is planned to become a default in make dist. The limit is there to support some really old tar implementations. Currently the default is still tar-v7 with its filename limit of 99 chars.

See https://www.gnu.org/software/automake/manual/1.10/html_node/Options.html#index-Option_002c-tar_002dv7
and https://www.gnu.org/software/tar/manual/html_node/Formats.html#Formats

[2023-09-20 Mi]

Install or Update Baldur's Gate 3 with lgogdownloader

Because I'll need that command regularly. This requires having bought the game on GOG.com (where there's no DRM).

```
cd /path/to/games && \
  guix shell lgogdownloader -- \
    lgogdownloader --language=de --galaxy-language=de \
```

```
    --galaxy-install baldurs_gate_iii
```

To run Baldur's Gate installed this way I have the custom launcher `~/.local/bin/baldurs-gate-3`:

```
#!/usr/bin/env bash
cd "/path/to/games/Baldurs Gate 3/bin" && wine64 bg3.exe
```

> Since patch 4 you may need to use `bg3_dx11.exe`.
> Since patch 6 you may need to use `bg3.exe` again.

[2023-10-17 Di]

Automatically color-correct and shrink many scanned images with imagemagick -level — i.e. for inclusion in a PDF

Scans usually have bright gray background and dark gray lines, but you'll often want white background and black lines. This is how to fix that:

```
for i in *.png; do
  convert -level  5%x85% $i ${i%%.png}-contrast.png;
done
```

You'll need to adjust `5%x85%` to the right limits. Increasing the first gives you more black, increasing the second gives you more white. See imagemagick.org/Usage/color_mods/#level

you might need to replace `convert` by `magick`, depending on the version of the package.

If the images are PNGs, you may want to slim them down with pngquant:

```
pngquant --speed=1 *-contrast.png # creates images named *-fs8.png
```

If you want to subsequently turn them into a PDF (one reason for processing many scans), you can then use pdfjam:

```
pdfjam *-contrast-fs8.png
```

[2023-11-06 Mo]

Convert all pages from a PDF to images

Use `pdftocairo` from Poppler:

```
pdftocairo -png eop-cards.pdf
```

Optionally optimize the images:

```
pngquant --speed=1 *png
for i in *-fs8.png; do mv "$i" $(echo "$i" | sed s/-fs8//); done
```

[2024-03-02 Sa]

Colorize many images with imagemagick

If your printer is out of black ink but you need to print anyway, you may need this:

```
for i in *png; do convert "$i" -colorspace gray -fill blue -tint 100 "${i%.png}-tint
```

Now the dark parts are blue instead of black, but white stays white.

For explanation of the `${i%.png}` magic, see Bash Pattern Matching in the Advanced Bash-Scripting Guide of The Linux Documentation Project.

To memorize: `%%` cuts from the end, because `%` cuts the end of whitespace in LaTeX.

[2024-03-02 Sa]

recover photos from USB stick

```
dd if=/dev/sdX of=X.dd bs=8192 status=progress
guix shell testdisk -- photorec *.dd
```

If the stick is damaged, use ddrescue instead of dd. Can take a while.

[2024-03-22 Fr]

keep ssh connections alive via config

Add to `~/.ssh/config` of the **client**:

```
# prevent ssh connections from breaking due to inactivity
Host *
    ServerAliveInterval 240
```

[2024-06-20 Do]

Benchmarking different Guile versions

Requires Guix, Guile, Git, Mercurial, grep, sed, and bash.

Set `ITERATIONS=10` to get data for more meaningful statistics. But be prepared to wait quite long.

```
export PROGRAMS=/tmp # adjust this if you already have them
export VERSIONS="v3.0.10 v3.0.9 v3.0.8";
export ITERATIONS=1 # integer >= 1, how often to run the tests
cd $PROGRAMS || exit 1
git clone https://github.com/ecraven/r7rs-benchmarks
git clone https://git.savannah.gnu.org/git/guile.git
hg clone https://hg.sr.ht/~arnebab/wisp
cd $PROGRAMS/r7rs-benchmarks;
for i in $VERSIONS; do
    (cd $PROGRAMS/guile;
     git fetch --all
     git checkout $i;
     guix shell -D guile gperf sed guile -- \
         bash -x -c 'make clean; find . -iname '*.go' | xargs rm;
                     autoreconf -i;
                     ./configure CFLAGS="$CFLAGS -march=native";
                     make -j6');
    for j in $(seq 1 $ITERATIONS); do
        GUILE=$PROGRAMS/guile/meta/guile ./bench guile all;
        cat results.Guile >> results.Guile--$i && rm results.Guile;
    done
done
rm all.csv;
for i in $VERSIONS; do
    grep -a -h '+!CSVLINE' results.Guile--$i \
      | sed s/guile/guile--$i/g \
      | sed 's/+!CSVLINE!+//' >> all.csv;
done
for i in $VERSIONS; do
    $PROGRAMS/wisp/examples/evaluate-r7rs-benchmark.w \
    $PROGRAMS/r7rs-benchmarks/all.csv guile--$i 2>/dev/null;
done | grep -A2 "Geometric Mean slowdown"
```

To only re-evaluate in detail already gathered statistics:

```
export PROGRAMS=/tmp
export VERSIONS="v3.0.10 v3.0.9 v3.0.8";
cd $PROGRAMS || exit 1
for i in $VERSIONS; do
    $PROGRAMS/wisp/examples/evaluate-r7rs-benchmark.w \
    $PROGRAMS/r7rs-benchmarks/all.csv guile--$i 2>/dev/null;
done
```

[2024-06-30 So]

Inline gpg verification instructions with bash heredoc

Did you know that you can show people how to verify something you wrote on the shell without any quoting?

This command reliably confirms that I wrote this Hello World:

```
gpg --recv-key F34D6A1235D04903CD22D5C013EF8D452403C3EB; gpg --verify <<- "EOL"
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

Hello World
-----BEGIN PGP SIGNATURE-----

iQIzBAEBCAAdFiEE801qEjXQSQPNItXAE++NRSQDw+sFAmaRq08ACgkQE++NRSQD
w+vT7xAAk9CQ4DMMVHHyPE3lpKqc0pOhKapMemsk7RGxTlgSJr4m63JjiHaGyxFC
9fhsGpuyRE7cFMRfKNFT6T6XD0KkTcTlElPNkTrBXwdfMDuP24dd5scKIMLY0pr1
1qxBpc4p4JLBx44HQTrkAQ6OZ/gJR2faTF9RFLYTijV/d4RjwKEZm9x46y1wTonh
miQh/iP5TeF+ozEZOkFYGvTbZUBcBacE5sHETUdUnrHkT9sdkeNdv0olj9b2lwgb
BjdVQJaAvk9z9iut7+77vxhxfNiRojnUj/FUrgfyE6iwLfhl7WTL2PLh943rR7o4
THsDQNZy7I7rzk3ZyQhkZ2GQMMUbmG3YBvO9xqf8cTMSq9B5muIeH2h6grRlP9JV
Hgt2JrTKfbpg5+smpcoIY7x1d35pe9ufx3X/GN7qP2VnJVVhWna9wIBhe6si0OPf
YHJOlQC3REhlZvpnWp/5rlZ3cafOdl6sV2Lh+BWxd/C5uZCwFbFbQtHJ/rBzSrGz
5bxMFwIihF+BHfZ+6sCG5LKNUQISIg525cSxJotLXGjh+QZwzM1A+u59OILbUSPv
VAEQv4ELCZjaWNiuCv4UXMk4aINIqxkrMOoBEiTQH1X7erUGdkResHZRrWiLFoMY
RwQ2IR5Na2cfg94HGz8b/2tMynXUGwzN9Xybx4esMM9h9UfUGYM=
=MEV4
-----END PGP SIGNATURE-----
EOL
```

Note the `<<- "EOL"` construct. That's a bash heredoc in the safe (not evaluating) variant.

Sure, people have to verify that there's no `rm -rf /not/my/home/.gpg` between the two commands, but as far as teaching newcomers how to verify what I wrote, this is the simplest I saw.

And if you use a bash-scripts, you should know heredoc. It may seem obscure, but nothing else comes even close to its versatility.

I'll leave you with a fun one:

```
for i in {1..5}; do
  guile -c '(set! (@@ (system repl common) repl-welcome) (λ _ #f))
           (eval (read) (current-module))' <<-  EOL
```

```
      (display "Hello ${i}\n")
EOL
done
```

[2024-07-12 Fr]

Plot `top` **output directly with gnuplot**

To plot CPU and memory data simply collected via top.

First collect the data, let's assume your processes are called dryads-sun (for reasons).

Use `top -n1` to get just one piece output, order by something which puts your processes at the top. `sed` here adds the current date and time as first element to each line. `grep -v grep` removes grep from the output :)

```
for i in {1..100000}; do
  top -n1 -co %MEM | grep dryads-sun \
    | sed "s/^/$(date --iso=seconds) /g" \
    | grep -v grep >> dryads-sun-load-data.csv; sleep 1;
done
```

Then plot the result. Use ARGB colors and `png truecolor` to get transparency.

```
set title "Dryads Sun Load Test Results Over Time\ndryads-wake.1w6.org/sun\nThank you
set ytics nomirror
set y2tics
set ylabel "CPU load / %"
set y2label "memory consumption / %"
set y2range [-15:15]
set y2tics 0,5,15
set yrange [0:200]
set ytics 0,25,100
set xdata time
set xtics rotate
set xlabel "Time / day hour:minute"
# time format to parse
set timefmt "%Y-%m-%dT%H:%M:%S+00:00"
# time format to output
set format x "%d %H:%M"
plot "dryads-sun-load-data-limited.csv" using 1:11 axis x1y2 title "memory" lc rgb "#
      "dryads-sun-load-data-limited.csv" using 1:10 title "CPU" lc rgb "#dd9400d3"
set output "dryads-sun-first-mini-load-eval.png"
# use truecolor for transparency
set term png truecolor
```

```
replot
```


Dryads Sun Load Test Results Over Time
dryads-wake.1w6.org/sun
Thank you for testing!
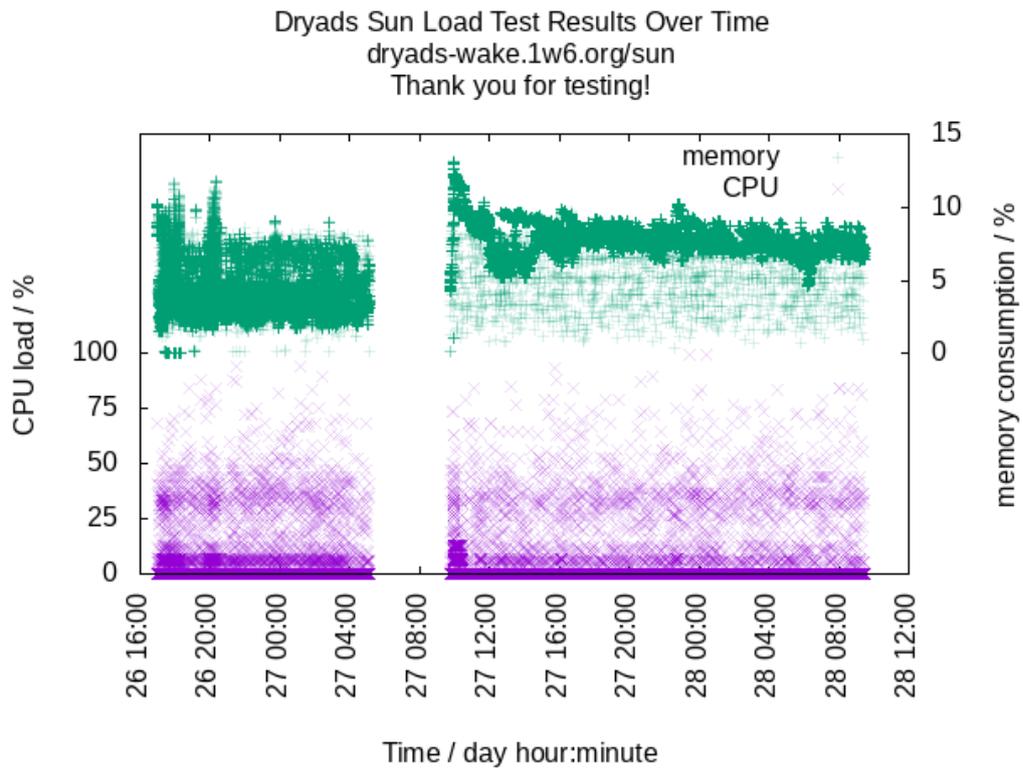
[2024-07-28 So]

Bash Strict Mode

To make your bash scripts stop on error and with undefined variables, use:

```
set -euo pipefail
```

More details: Use Bash Strict Mode (Unless You Love Debugging) (by Aaron Maxwell).

Also as beautiful comic: Julia Evans b0rk: errors

[2024-09-03 Di]

force fsck after boot with just touch

```
sudo touch /forcefsck
```

Then reboot.

[2024-09-04 Mi]

Makefile with recursive wildcard to call any build tool with make

```
# recursive wildcard, which wildcard should be
rwildcard=$(wildcard $(addsuffix $2, $1)) $(foreach d,$(wildcard $(addsuffix *, $1)),

SOURCES := $(call rwildcard,src/,*.java) $(call rwildcard,src/,*.kt)
.compiled: $(SOURCES)
    THE_BUILD_TOOL && touch "$@"
```

This now only runs `THE_BUILD_TOOL` when a source file changed.

[2024-09-19 Do]

Activate signal-desktop from signal-cli

```
D=$(mktemp -d)
import $D/qr.png
URL=$(zbarimg $D/qr.png | cut 8-)
cd path/to/signal-cli && \
  ./gradlew -q run --args="-a YOUR_NUMBER addDevice --uri $URL"
```

- mktemp is from the GNU coreutils
- import is from imagemagick
- zbarimg is from zbar
- signal-cli is signal-cli :)

[2024-10-12 Sa]

kick/ban someone on IRC with silent ops

This is what works on Libera.Chat if you are channel owner:

```
/msg chanserv op #CHANNELNAME
/ban NICK reason to ban
/kick NICK reason to kick
/msg chanserv deop #CHANNELNAME
```

Not really a shell trick, except if you run IRC from the shell, but this looks like a good place to keep this.

In IRC it's common not to wear the `op` hat if not needed. My reason is that by dropping the hat when not needed I avoid biasing communication.

Also see the Quick Ops Guide on Libera.Chat.

[2024-12-12 Do]

Shuffle files in filesystem order for a primitive mp3 player

Some primitive mp3 players just play files on USB sticks in their filesystem order (on FAT32-systems, the only ones they support).

To shuffle a playlist, you can't just rename the files. You have to add them in the order you want them played.

Here's a tiny shellscript to shuffle the files by moving them into a subfolder and then moving them back in shuffled order.

All files passed as arguments are moved first, so they are played first.

```bash
#!/usr/bin/env bash
if [[ x"$1" == x"--help" ]]; then
    echo Shuffle the filesystem order of all mp3 files in the folder by moving them o
    echo Pass explicit files to have them added first.
    echo
    echo bash $0 [--help] [FILE ...]
    exit 0
fi
cd "$(dirname "$(realpath "$0")")" &&
    mkdir temp_files_for_shuffle_first/ &&
    mv "$@" temp_files_for_shuffle_first/ &&
    mkdir temp_files_for_shuffle/ &&
    mv *.mp3 temp_files_for_shuffle/ &&
    ls temp_files_for_shuffle_first/* | xargs -d "\n" -I {} mv "{}" ./ &&
    ls temp_files_for_shuffle/*.mp3 | shuf | xargs -d "\n" -I {} mv "{}" ./ &&
    rmdir temp_files_for_shuffle_first/ &&
    rmdir temp_files_for_shuffle/
```

[2025-03-22 Sa]