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

Dr. Arne Babenhauserheide

<2020-05-18 Mo>

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.

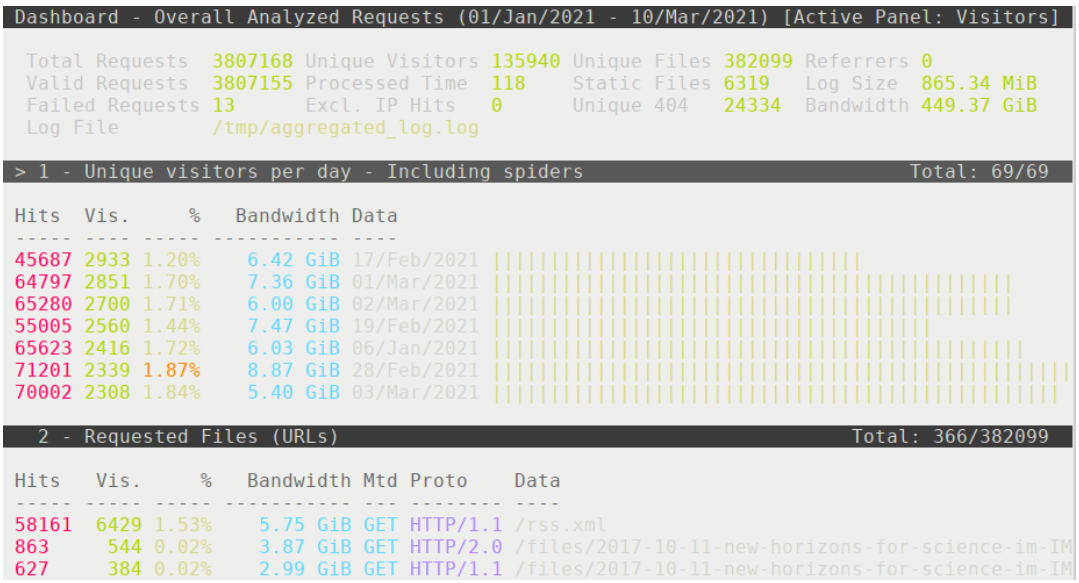
```
cp logs/access_log_2021*.gz /tmp/
cd /tmp/
gunzip access_log_2021-0*
cat access_log* > aggregated_log.log
goaccess --all-static-files --ignore-crawlers -f aggregated_log.log
```

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:



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], send
+            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]