Personal Blog of Rob Muhlestein (rwxrob)

Robert S. Muhlestein (rwxrob)

2025-01-10 15:06:51: Perpetually published

Table of Contents

No TOC Version	https://linktr.ee/rwxrob	Download PDF	Download EPUB

Hello friend. Welcome. Here you'll find my random ranting about nothing and everything. Some ideas start here and end somewhere else. Mostly I write for me but might as well share it.

Bonzai in Go coding book

Thursday, January 9, 2025, 3:02:57PM EST

After concluding that the `clip` project will be migrated to Go I've decided to use that project in *both* the *Terminal Velocity* book and the *Coding from the Get Go* book. Thanks to AsciiDoc I can include the entire chapter in both of them without any problem.

Terminal Velocity

In the *Terminal Velocity* book the `clip` project will be a capstone after learning POSIX and basic Perl shell scripting with a focus on raw development speed. I'll use it to introduce Bonzai specifically with the intent of showing just how fast powerful commands with self tab completion can be realized in Go. *This will assume Go coding ability already.* The entire point of this inclusion is not to each Go but to show just how solid a language it is for these kind of tools—especially when combined with the `embed` package.

Coding from the Get Go

In this book the `clip` project will be one of the final projects after all the basics have been covered. It will serve as an example of the symphony of concepts coming together to create something useful and fun in the world's best language for such things. This will give coders learning Go a solid example of just how much Go dominates all other languages for this kind of work (despite the crap unemployed influencers continue to throw at it).

Perl or Go for `clip`

Thursday, January 9, 2025, 2:30:01PM EST

I'm having a hard time deciding if my `clip` bash script should be rewritten in Perl or Go or both. It's one of my favorite projects that we have created together while live streaming taking just under a day to complete in bash. In fact, I often use it as an example of rapid prototyping because it was just so freaking easy to create.

The use of a plain text delimited file for all the clip data makes it a no-brainer for Perl but Go has the same parsing possibilities.

Dependencies as criterion?

Another factor is the dependency on external tools `mvp` and `yt-dlp`. Having a fancy self-contained, cross-compiled Go binary kind of loses its mystique as soon as you depend on other binary programs being on the system. Go's more of a glue language than other statically compiled options but it brings a lot of development overhead compared to the queen mother of glue languages: Perl. In fact, I'm inclined to use that as a decision control point in my local dot files design. If it depends on *anything else* being installed it automatically gets written in Perl—especially if there is flat-file parsing to be done.

That said, Go can completely embed it's dependencies on other binaries including these. The database of different clips could also be embedded as a static file into the resource fork of the binary making nothing more than the binary required to do everything.

Primary use case

How and where is the `clip` tool going to be used also has a lot to do with this decision. This fun script will only ever be run from my cozy live streaming workstation. This means I can make very solid assumptions about the environment, for example, there will *always* be the latest Perl installed.

It's therefore safe to assume that anyone else ever wanting to use it would be willing to install Perl as well if they don't already have it. In a way, I can use this to demonstrate just how relevant and powerful and fun Perl has always been. And speaking of fun…

The `cowsay` program is also Perl. In fact, most of the fun ASCII-art projects in the world are solidly written in Perl. I do, however, feel like that is a better domain for Go and would like to rewrite `cowsay` and `fishies` (`asciiquarium`) entirely in Go because it is the sort of thing that should never have a dependency on the underlying system and installing specific CPAN modules just to get it to run is hella-annoying.

What about self-tab-completion?

This `clip` tool has several actions that are the first argument. This is the modern model for commands that have multiple functional ways to use them. Perl has zero support for self-completion and although it could easily be written it would bloat the Perl code significantly in ways

that would definitely push it beyond its sweet spot. In fact, anything that requires tab completion has clearly crossed over from being doable in Perl into Go's domain area of expertise—especially with Bonzai and Cobra being available.

Going with Go

It's decided then. I'll rewrite `clip` in Go using Bonzai with completion and live stream doing it with spoken commentary and that will be an excellent project for the *Terminal Velocity* book because it just passes Perl's sweet spot with the completion requirement and is a perfect, useful project for learning Go for creating amazing terminal commands.

Original bash code

Here's the original bash script for posterity:

```
#!/usr/bin/env bash
# shellcheck disable=SC2016
set -e

# Copyright 2021 Rob Muhlestein <rob@rwx.gg>
# Released under Apache-2.0 License
# Please mention https://youtube.com/rwxrob

: "${CLIP_DATA:="$HOME/.config/clip/data"}"
: "${CLIP_DIR:="$HOME/Movies/clips"}"
: "${CLIP_SCREEN:=2}"
: "${CLIP_VOLUME:=-50}"
: "${PAGER:=more}"
: "${EDITOR:=vi}"
: "${HELP_BROWSER:=}"

EXE="${0##*/}"

declare -A help

help[main]='
The `clip` utility (a bash script) is for downloading managing and
playing clips from videos in full screen from the command line.

## Dependencies

Required:

* `bash` (4+)
* `mpv`
* `yt-dlp`

Will use if detected:
```

````
* `keyon` / `keyoff`
* `pandoc`

Environment variables:

```
: ${CLIP_DATA:="$HOME/.config/clip/data"}
: ${CLIP_DIR:="$HOME/Videos/clips"}
: ${CLIP_SCREEN:=1}
: ${CLIP_VOLUME:=-50}
: ${PAGER:=more}
: ${EDITOR:=vi}
: ${HELP_BROWSER:=}
```

## Usage

```
clip dir
clip data [<name>]
clip add <name> <url>
clip edit
clip help [<command>]
clip list
clip (play) [<name>]
clip usage
```
## Commands

* [usage] - print usage summary
* [play]  - play clip, randomize if duplicates, select menu if none
* [list]  - print list of all clips by name
* [data]  - print data for name (random if dupicates)
* [dir]   - print directory path containing clip videos
* [add]   - download and add video from YouTube URL
* [edit]  - open data file for editing with `$EDITOR`
* [help]  - display help for all or specified command

[usage]: help-clip-usage.html
[play]: help-clip-play.html
[list]: help-clip-list.html
[data]: help-clip-data.html
[dir]: help-clip-dir.html
[add]: help-clip-add.html
[edit]: help-clip-edit.html
[help]: help-clip-help.html

## Data File Format

The data file containing clip name, volume, and source information can be edited with
the `edit` command and is updated by the `add` command when adding new clips. The
````

first two numbers after the file name (separated by commas) are for the start second (including decimals) and the length. To add another segment from the same file, add a semicolon and another start and length, and so on. Clips that have the same name will be randomized. The `play` commands accepts a regular expression which can also be used to randomize between several clips.

```
working 100 dXjcvIPSBr4.mkv,19,20
revenge 100 _oyP0QHjty8.webm,17,30
rick 100 dQw4w9WgXcQr.mkv,0,9001
shit 120 seKLsHOkN1U.mkv,2,4.3
shutup 100 k8QE4-BEy4E.mkv,0,10
guy 120 _uMEE7eaaUA.mp4,9.5,9001
verse 240 VX58scb5_B0.mkv,-16.9,14
element 140 gINEJhFyQmk.mp4,5.4,1.9
opinion 140 j95kNwZw8YY.mp4,2,15
code 110 sMMpYiiuEMc.webm,-13,2.2
dance 110 wCDIYvFmgW8.webm,43,24
walken 110 wCDIYvFmgW8.webm,43,24
fever 120 cVsQLlk-T0s.mkv,255.6,1.7
yes 100 lNEX0fbGePg.mkv,120,6.2
lobot 70 taUSotxNaGA.mkv,20.5,1.7
stupid 100 yf8Y85Lfrwk.mkv,35.2,3
fish 100 KezvwARhBIc.mp4,0,9001
no 100 CQ_eDE0OMds.mp4,106.8,6.5
word 110 dTRKCXC0JFg.mp4,-4.2,2.2
phone 110 6xZif3WmG7I.mkv,142,6.5
everyone 100 mX-qK4qG2EY.mkv,127.4,2.1
hack 120 u3CKgkyc7Qo.mkv,21,1.5
planet 120 u3CKgkyc7Qo.mkv,21,1.5
game 100 KXzNo0vR_dU.mkv,60.4,4
ping 180 jr0JaXfKj68.mkv,10.5,2
shark 130 Cs9M1m-dpgM.mp4,6,14
hack 130 u8qgehH3kEQ.mkv,0,5
scan 130 u8qgehH3kEQ.mkv,0,5
stop 120 Iur1d9CP6bA.mkv,206.5,2.25
cube 100 u1Ds9CeG-VY.mkv,0,9001
hack 100 u1Ds9CeG-VY.mkv,7,13
dance 100 u1Ds9CeG-VY.mkv,100.5,2.3
hdance 100 u1Ds9CeG-VY.mkv,100.5,2.3
unix 120 dxIPcbmo1_U.mkv,8.5,2.5
helm 120 9FHw2aItRlw.mkv,0,9001
danger 100 HdKqAVpUOwI.mkv,0,3
irc 180 O2rGTXHvPCQ.mp4,17.5,8
gui 140 hkDD03yeLnU.mkv,0,13
lawn 101 Ho8o92Ro-Ig.mp4,58.8,1.5
excellent 150 lVhATSQHw9k.mkv,3.2,1.4
developers 80 KMU0tzLwhbE.mkv,0,9001
clap 200 DdCYMvaUcrA.mkv,26.8,1.6
```

```
'

help[edit]='
The `edit` command opens the clip data file (`CLIP_DATA`) with the current `$EDITOR`
(default: `vi`).
'

x_edit() { "$EDITOR" "$CLIP_DATA"; }

help[data]='
The `data` command returns a line from the data file (`$CLIP_DATA`) that
matches the argument passed, which can be simply a string or an extended
regular expression. If more than one line matches, then one will randomly be
returned.'

x_data() {
    _filter "$@"
    local -a data
    mapfile -t data <"$CLIP_DATA"
    local -a matches
    IFS=$'\n'
    for c in "${data[@]}"; do
        [[ $c =~ ^$1 ]] && matches+=($c)
    done
    echo "${matches[$((RANDOM % ${#matches[@]}))]}"
}

help[play]='
The `play` command (which is the default when no command is passed)
takes the basename of a video in the `$CLIP_DIR` and plays it with
passed volume (100), starting point (0), and length (10). If the name
passed refers to one or more clips, a random clip from among them will
be select. Also, rather than a specific, any Bash-compatible regular
expression may be passed allowing for interesting combinations. For
example, a `coin` alias could be created to select randomly from between
the *yes* and *no* clips, which could themselves also have multiple in
their groups. (See `man mpv` for more details on how the videos are
played.)'

x_play() {
    if [[ -z "$1" ]]; then
        select c in $(x_list); do
            x_play "$c" || true
        done
        return $?
    fi
    type keyoff &>/dev/null && (keyoff || true)
    local -a data
    data=($(x_data "$@"))
    local -i volume="$CLIP_VOLUME"
    volume+=${data[1]:-100}
```

```
    mpv --fs \
        "--volume=$volume" \
        "edl://$CLIP_DIR/${data[2]//;/;$CLIP_DIR/${data[2]%%,*},}" \
        &>/dev/null

    #   mpv --fs \
    #     "--volume=$volume" \
    #     "--fs-screen=$CLIP_SCREEN" \
    #     "edl://$CLIP_DIR/${data[2]//;/;$CLIP_DIR/${data[2]%%,*},}" \
    #     &>/dev/null

    #type keyon &>/dev/null && (keyon || true)
}

help[add]='
The `add` command will download the provided YouTube URL into the
`$CLIP_DIR` directory and name it according the YouTube identifier
preserving the same file suffix. It then add an entry to the clips data
file (`$CLIP_DATA`) with default settings which can be changed easily
with `edit` later. Be sure to use the shareable URL (rather than the one
in the omnibox) so that the ID is extracted correctly.'

x_add() {
    _filter "$@"
    (($# == 2)) || (echo "usage: $EXE add <name> <video>" && return 1)
    cd "$CLIP_DIR"
    local url="$2"
    local id="${url##*/}"
    yt-dlp "$url" -o "$id.%(ext)s"
    local name base
    path=$(ls "$CLIP_DIR/$id".*)
    file=${path##*/}
    base=${file%.*}
    echo "$1 100 $file,0,9001" >>"$CLIP_DATA"
    cd -
}

help[list]='
The `list` command display a space-delimited list of all possible, unique clip names
sorted alphalexically.
'

x_list() {
    mapfile data <"$CLIP_DATA"
    data=($(printf "%s\n" "${data[@]%% *}" | sort -u))
    echo "${data[@]%% *}"
}

help[dir]='The `dir` command prints the full path to the directory containing the
videos used for all clips (`$CLIP_DIR`).'
```

```
x_dir() { echo "$CLIP_DIR"; }

help[usage]='The `usage` command print a summary of usage for this command.'

x_usage() {
    local cmds="${COMMANDS[*]}"
    printf "usage: %s (%s)\n" "${0##*/}" "${cmds// /|}"
}

help[help]='
The `help` command prints help information. If no argument is passed
displays general help information (main). Otherwise, the documentation
for the specific argument keyword is displayed, which usually
corresponds to a command name (but not necessarily). All documentation
is written in CommonMark (Markdown) and will displayed as Web page if
`pandoc` and `$HELP_BROWSER` are detected, otherwise, just the Markdown is
sent to `$PAGER` (default: more).'

x_help() {
    local name="$1"
    if [[ -z "$name" ]]; then
        for c in "${COMMANDS[@]}"; do
            x_help "$c" buildonly
        done
        x_help main
        return 0
    fi
    local title="$EXE $name"
    [[ $name = main ]] && title="$EXE"
    local file="/tmp/help-$EXE-$name.html"
    if _have pandoc; then
        if _have "$HELP_BROWSER"; then
            pandoc -s --metadata title="$title" \
                -o "$file" <<<"${help[$name]}"
            [[ -z "$2" ]] && cd /tmp && exec "$HELP_BROWSER" "$file"
            return 0
        fi
        pandoc -s --metadata title="$title" \
            -t plain <<<"${help[$name]}" | "$PAGER"
        return 0
    fi
    echo "${help[$name]}" | "$PAGER"
}

# ———— completion and delegation ————--

_have() { type "$1" &>/dev/null; }

_filter() {
    (($# > 0)) && return 0
    while IFS= read -ra args; do
```

```
        "${FUNCNAME[1]}" "${args[@]}"
    done
}

while IFS= read -r line; do
    [[ $line =~ ^declare\ -f\ x_ ]] || continue
    COMMANDS+=("${line##declare -f x_}")
done < <(declare -F)

if [[ -n $COMP_LINE ]]; then
    line=${COMP_LINE#* }
    for c in "${COMMANDS[@]}" $(x_list); do
        [[ ${c:0:${#line}} == "${line,,}" ]] && echo "$c"
    done
    exit
fi

for c in "${COMMANDS[@]}"; do
    if [[ $c == "$EXE" ]]; then
        "x_$EXE" "$@"
        exit $?
    fi
done

declare cmd="$1"
shift || true
for c in "${COMMANDS[@]}"; do
    if [[ $c == "$cmd" ]]; then
        "x_$cmd" "$@"
        exit $?
    fi
done

x_play "$cmd"
```

cht.sh obsoleted by AI

Thursday, January 9, 2025, 1:36:15PM EST

I went to port my `cheat` program and realized I never, ever need this again. No more futzing with the exact perfect query parameters to get what is needed. Hell, I don't even have to spell things correctly with Chloe.

For historical reasons, here is what I used to use for `cheat`:

```
#!/bin/sh
where="$1"
test $# -ge 1 && shift
IFS=+ curl -sS "http://cht.sh/$where/$*"
```

POSIX + Perl backticks

Thursday, January 9, 2025, 1:01:54PM EST

Perl backticks invoke subprocesses to run whatever in the host shell. In the old days I would stay up late ensuring my Perl code didn't have any backticks. Today I realize that is the slippery slope to Perl hell, the one that gave Perl a bad name in 2025. I now use backticks all the time. They are simpler, easier to remember, integrate better with other POSIX shell scripts, and generally just more Unix-y.

Better POSIX integration

Here's a simple example using Perl builtin modules:

```
use File::Basename;
use Cwd;
my $name = basename(cwd());
```

Now the exact same thing using nothing but POSIX that I would regularly be using from the command line itself:

```
my $name = `basename \$(pwd)`;
chomp($name);
```

This invoked two subprocesses but do I really care? Scripting at this level is supposed to be about agile discovery and functionality, not the most efficiency.

Here's another example:

```
use File::Path qw(make_file);
make_file('./path/to/some/where');
```

But this is so much easier to remember and understand:

```
`mkdir -p ./path/to/some/where`;
```

As of 2018 the `-p` became POSIX standard.

Every time I feel regret for taking the subprocess hit, which is absolutely negligible on any modern computer, I just have to remember that I would be invoking many *more* subprocesses if I were to do this in POSIX shell only instead. Besides, using POSIX commands means I can build and test them from the command line itself while I'm building up the Perl code.

The less modules the better

As soon as I find myself looking up the niche docs for `File::Find` (which is absolutely amazing but completely unlike anything from the `find` command) I realized I've probably lost my way. It's hard enough to remember the syntax of the `find` command—the most powerful command in all of Unix and Linux. All that knowledge will not transfer to my command-line skills if I use `File::Find` instead. It only creates highly specialized Perl knowledge.

For the most part I've banned `use` from my Perl scripts in my dot files. Other than `use v5.36` which saves on a lot of typing and enables `say` I really don't need the rest. Everything that I would want a module for generally already has a POSIX shell command that will do the same thing.

Think `awk` on steroids

When I remember to approach Perl like a better `awk` I do not get caught up into these problems. I don't waste time looking up lost, arcane Perl knowledge. I just use what I would every day from the command line.

`mkdir -p` is POSIX (now)

Thursday, January 9, 2025, 11:43:23AM EST

Using `mkdir -p` is one of the most useful ways to create a directory because you never have to worry about the directory existing, which is why some years back I was really troubled to discover that it *was not* POSIX compatible. In fact, people were using it on our IBM project that strictly forbade anything not POSIX.

Well I was happy to be informed by `swills6` on Twitch that the `-p` is now officially POSIX, which explains why `ash` on Alpine had it when I checked.

Reference

The Open Group, "mkdir - make directories," The Open Group Base Specifications Issue 7, 2018 Edition, [Online]. Available: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/mkdir.html. [Accessed: Jan. 8, 2025].

Kris Nova suggested AsciiDoc

Thursday, January 9, 2025, 12:00:59AM EST

I get asked a lot why I use AsciiDoc now, and I have written extensively about justifying the decision from a technical point of view. However, I want to focus on how I learned about it from Kris Nova for a second.

I was obsessed with getting the simplest, most standardize format for knowledge content so that it would not get out of date. Little did I know the AI revolution would entirely blow up the Zettelkasten model, but that's covered elsewhere.

It was during that time that Kris noticed I was focused on Markdown. I think, actually, that I was watching her work on her *Hacking Capitalism* book and I might have started the conversation in the chat. She didn't attack me or my selection of Markdown. She just used AsciiDoc and mentioned that it had been recommended to her by other published authors since O'Reilly and other reputable publishers used it a lot for tech books—especially since it has full support for LaTeX (not just for math). I looked into it and saw the use of in-code references that are bulleted below, which I have seen in dozens of books I have read, and it was immediately obvious that this was the choice *specifically for book publishing*.

The more I looked at the syntax the more annoyed I got. And Ruby? Really?

Here's the thing, though, it is and was the best tool for the publishing job. There is nothing that even holds a candle to it. Everything else tries to be it—even Pandoc—but they all fail.

This was really hard to accept. I knew it was the right tool, but had invested so much time and energy into using Markdown. I even created PEGN to create my own meta-language grammars on top of Markdown that simplified it further with the hope of making it all the more approachable by the masses. I was wrong.

Markdown is absolutely amazing for `README.md` files and the like, or even chatting in Discord, but it does not specify something as basic and rudimentary as including other files—even from remote URI sources. AsciiDoc had all that on the first day because that is the problem it was trying to solve for writers, not simple `README.md` writers and note takers. AsciiDoc is complex but it is also very highly specified. Any Markdown variation that would allow the same requirements is not.

To get a sense of this, look at how MkDocs and Obsidian have bastardized Markdown in ways that no other variation will accept rendering them completely unable to be used by any other parser. AsciiDoc has *never* had this problem.

The conclusion is that because I had made a friend who had written a book and actually had it published I was able to cut through the hype and influencer conjecture on the Internet and learn the *real* tool for the job from someone who *has had that job*. I will be forever grateful to Kris for being so kind and patient about her corrections—if I can even call them that. Hopefully I can become more like her. I miss her so much.

"Language of the year"

Wednesday, January 8, 2025, 8:39:07PM EST

Perl. How about Ada? Okay, then COBOL.

I'm not even kidding. So few people even know these languages but way more code exists in production in them than any trending language—yes including Python.

I'm going to be creating a lot of Perl content, because it is the best tool for the job when doing anything with text processing—especially from the command line. If every single person who uses the terminal can benefit from learning even a little Perl to construct powerful one-liners (which I've covered a lot in other blog posts).

How about "Shovel of the year?"

A language is just a tool. The concept of a "language of the year" is just as stupid as focusing on just one type of hammer or shovel or hair dryer. Languages address different problems in different ways.

The focus *should* be on solutions or a "solution of the year" or "project" of the year. Why not pick some very valuable and important open source projects that need help and live stream working on those while working on learning the language at hand? I've met people who are literally keeping planes flying with their contributions and bug fixes. Why not that?

Do NOT do Zettelkasten

Wednesday, January 8, 2025, 11:21:21AM EST

I've written about this related to Perpetual Publishing. This short reason should be obvious: Zettelkasten is disjointed and hard for an AI to parse and digest. Hiding all your knowledge in this format will guarantee your AI will never be able to help you use it usefully. You have been warned.

This was a very hard realization. It meant that all my work on KEG and `kn` tool was for not. I spend hundreds of hours on those projects and locking away knowledge in the most useful, simple format (Markdown) possible. And yet I could *never* have anticipated the impact and immediate knowledge indexing capabilities of AI—specifically ChatGPT. By putting the content in plain old web docs and blog posts my AI (Chloe) can *immediately* parse and summarize it. This is the new workflow for *all knowledge workers*. You can ask an AI to summarize any long video providing markers to key points, you can have an AI summarize an entire article, you can have an AI summarize an entire body of work or book for that matter. The AI *cannot* summarize an Obsidian knowledge base nearly as well. Just ask it. It'll explain the entire reason much better than I ever can.

Perl pae, ppae, ppie

Monday, January 6, 2025, 6:31:06PM EST

> POSIX or Perl!

A lot of people don't know that `sed -i` and `awk -i` are *not* POSIX compliant and should never be used in any scripts, ever. Besides these stupid commands use up more than three times the RAM of `perl` and don't come close to providing the same functionality. Here's how you never use horrible `sed`, `awk`, `tr` and non-POSIX inplace replacements ever again.

I have created three similarly named shell scripts that call `perl` to progressively built up whatever it is I want to transform about a file or text pipeline.

pae

```
#!/bin/sh
exec perl -aE "$@"
```

This one maps over each line of input splitting the fields into `@F` but not printing the output without adding a `say` or `print`:

```
pae print <file> ...                      # cat equivalent
pae 'say $F[2]' <file> ...                # awk '{print $3}'
pae 'END{say $x} $x += $F[2]' <file> ...  # total all third fields
pae 'print unless $.==1' <file>           # trim first line (one file only)
pae 'print if $.==3' <file>               # print third line
pae 'print if eof' <file>                 # print last line
```

This is particularly useful for doing things with the line without printing the line, the sort of *reduce* operations from functional programming. Something similar to this can be done in `awk` but with only a tiny fraction of the power and more than triple the RAM cost.

Remember, when multiple files are passed as arguments with these options Perl treats them as one continuous stream of data so the special line number variable `$.` is not reset for each file. You can, however, reset it by closing the special `ARGV` file handle when you reach the end of a line. This isn't as beautiful as a one-liner, but makes for a great two liner like this `trimfirst` script:

```
#!/usr/bin/env perl -i -n
print unless $. == 1;
close ARGV if eof;
```

ppae

This is exactly the same as `pae` but always prints the line after it has been transformed by whatever code is passed as the first argument. Notice that it is capital `E` instead of lower case. This is important because it enables all the optional features like the `say` command, which you *definitely* want:

```
#!/bin/sh
exec perl -paE "$@"
```

To start a transformation I start with a no-op (`_`) which effectively cats the file to see what I'm dealing with and then change to a transformation:

```
ppae _ <file>
ppae s,/home,/other/home,g <file>
```

Notice I don't even need to quote the regex most of the time.

This is great for complex filter transformations from within Vim or Neovim when the limited builtin "enhanced" regular expressions just won't cut it (literally).

ppie

Then we have the in-place edit version:

```
#!/bin/sh
exec perl -pi -E "$@"
```

Here's where the *real* power kicks in. If I change the `a` to an `i` it will *commit the change to the file* rather than printing it out:

```
ppie s,/home,/other/home,g <file>
```

Keep in mind *this is destructive with no backups*. If you are paranoid you can alter the `ppie` program and add a suffix to the `-i` option—perhaps even the process ID—so that you have a backup but I find cleaning up the backups annoying and using the `ppae` first ensures I can preview exactly what the transformation will be in advance. Plus, there's always `git restore`.

Also know that if you enable expanded globbing with `shopt -s globstar` you can use `**/*.go` and such to find and replace that *in every single file in the current directory and below* (yeah, you definitely want that saved with `git commit` first).

Conclusion

Perl pipelines and filters like these are a *testament to the raw power of text processing in the*

terminal and specifically to Perl. Larry Wall regularly talked about how these transformations were the very reason he created Perl in the first place, not to be better than C and POSIX shell for backend web development, the only options at the time. Forget about bloated extensions and plugins—these one-liners replace them all. And remember, this power *is* Perl's sweetest of spots. Python, Ruby, Node, Bash, none of them can touch it (and yet all of them use Perl's original regular expressions).

Join fields with bullet

Monday, January 6, 2025, 5:24:20PM EST

Yeah, `awk` isn't anywhere near as amazing as this 42 character `bullets` Perl script:

```
#!/usr/bin/perl -pa
$_ = join " • ", @F
```

What does it do?

This little script will take every whitespace-separated word on a line and join them with bullets.

- `-p` → Read each line, process it, and print it automatically.
- `-a` → Auto-split the line into the `@F` array using whitespace as the delimiter.
- `$_ = join " • ", @F` → Reconstruct the line by joining each field with a fancy bullet •.

```
foo bar baz
hello world perl
```

Becomes:

```
foo • bar • baz
hello • world • perl
```

Using it in Vim/Neovim

While I could do something like this in Vim:

```
:.!perl -pa -E '$_ = join " • ", @F'
```

Putting it into a `bullets` executable script is better since it integrates perfectly into Vim or Neovim without any unnecessary plugins. Run it on a line (`!!bullets`), section (`!}bullets` or `!apbullets`) or whole file (`:%!bullets`).

If you are going to do a lot of this, map to a Vim/Neovim keybinding:

```
vim.api.nvim_set_keymap('n', '<leader>jb', ':.!bullets <CR>', { noremap = true, silent
= true })
```

Now, every time you hit `<leader>jb`, the current line gets bulletized. For me, I just use the other filter commands because they work everywhere and I can create my own pipelines. But maybe you are

doing *a lot* of bulleted joining.

Editor independence

A filter command doesn't care whether you're in:

- Vim
- Neovim
- Nano
- Emacs
- A plain shell session

It just *works*. Lua plugins and other editor extensions, on the other hand, are trapped by their editors. They can't participate in shell pipelines or be reused across different tools.

Not just for editing

When deciding between a Lua plugin for Neovim and a command-line filter, the real power of any command executable filter comes down to *composability*.

Filters like our bullet-joining one-liner are not locked into *one editor* or *one use case*. They live in the shell, where they can:

- Be combined with other commands in pipelines
- Process input from files, pipes, or other scripts
- Run in batch mode across thousands of files

For example, you could easily chain the bullet filter with `grep`, `sort`, or any other Unix tool:

```
grep '^foo' file.txt | bullets | sort -r > output.txt
```

This pipeline:

1. Filters lines starting with `foo`
2. Applies the bullet transformation
3. Sorts the output in reverse
4. Writes the result to a file

No Lua plugin can do this.

A philosophy, not just a tool

When you use a command-line filter one-liner:

- You’re embracing the Unix philosophy: "Do one thing, and do it well."
- You’re keeping your tools *composable* and *interoperable* across any system, shell, or editor.

Lua plugins are great for in-editor enhancements such as syntax highlighting, linting, fixing, or displaying keystrokes during a live stream, but when it comes to raw power and flexibility, commands win every time.

Pygments can't `m,foo,bar,`

Monday, January 6, 2025, 12:18:10PM EST

No surprise that that the Python `pygments` tool for AsciiDoc syntax highlighting gets `m,foo,bar,` wrong. Oh well. I swear by this construct wherever regex is supported because I'm usually doing regex to a path with slashes in it and frankly its easier to type. I just have to use a backslash escape instead, which is the entire reason I prefer Perl for such things because there is so much less escaping going on.

Power of Perl

Monday, January 6, 2025, 11:37:52AM EST

The power of Perl comes from context—the very thing people complain about the most. This allows a ton of work to be packed into very few characters, which is what Perl was created to do, create the most powerful one-liners on the planet. I'm happy to report Perl still reigns supreme for such things. Take this relatively simple `afk` example. It's a simple script I threw together in a few minutes to look up all the scripts I have with `banner*` and then randomly pick one and run it with my "back in a bit" screen using TMUX.

```
#!/usr/bin/env perl
use v5.36;
use File::Basename;
my $what = join ' ', @ARGV;
$what ||= 'in a bit';
`prepend ~/.currently "  back $what"`;
my @paths   = grep { -f && /\/banner\S.*$/ } glob( dirname($0) . '/*' );
my @scripts = map  { basename($_) } @paths;
my $script  = $scripts[ int( rand( scalar(@scripts) ) ) ];
($script) ? exec "$script back $what" : exec "banner back $what";
```

These few lines do things that would take ten times as long to write in any other language, plus you get the power of Perl regular expressions, the most powerful in the world.

I'm really love Perl this year.

Have a look at the rest of the scripts in my https://github.com/rwxrob/dot files.

`gh pr create --draft`

Sunday, January 5, 2025, 9:45:58PM EST

I just learned (thanks to Chloe) that draft PRs do *not* send any notifications or emails to followers of the repo. This makes remembering to make a few PR a draft with `--draft` all the more important. Not doing so means something like my dot files repo would inundate them when doing a reorganization even on a separate branch.

Ghostty broke tview

Sunday, January 5, 2025, 4:10:39PM EST

https://github.com/gdamore/tcell/pull/763

I have to laugh a little at Ghostty's complete inability to impress me. It continues to demonstrate poor design and development decisions (compared to WezTerm, for example). The latest I discovered is that *for absolutely no reason* Ghostty has decided to come up with its own terminal marker type.

It's no surprise to me that the Ghostty devs didn't even think to test the largest terminal library on the planet (`tcell` which I sponsor) before releasing it. I know it is a passion project, but my God. At the very best we can assume that the developer either doesn't know about Go or just doesn't care about it. Maybe he'll get to it "one day" like me. If that is the case, then why not say it in the changelog or release notes before the whole freaking Internet explodes promoting it.

These things speak for themselves. Astute developers and technologists recognize horrible over-engineering when they see it. When a project doesn't provide a solution to a problem and—more importantly—demonstrates a complete and utter lack of testing against existing tools and libraries for compatibility they know to just ignore it. Maybe Ghostty will get better. Probably not. It will be *years* before Ghostty is half as good as WezTerm for everything that matters to me and most engineers focused on getting work done instead of playing with macOS only things and shaders—especially if I cannot even use it with the leading terminal library on the planet.

Suitable for work

Sunday, January 5, 2025, 1:00:30PM EST

I've decide to *attempt* to make this channel 100% suitable for work from now on, even when writing in my blog about stuff that is really hard to write about without really going off on. I have a bunch of scrubbing of the blog posts I have made so far. This way if someone wants to put the stream up on their TV at home or in the office they shouldn't have to worry about it, well, more than the average YouTube channel that is.

Terminal ASMR + chat

Sunday, January 5, 2025, 11:37:24AM EST

Going back to the beginning: just a terminal with something going on—usually coding when I can—and a bunch of people in the chat over IRC, YouTube, Facebook, and Twitch. I want to make my terminal into something that is just interesting to have on a side screen all day long—and I'm not talking *just* fishies. I want to create the Bloomberg terminal of *Terminal Velocity* fans, a place for them to come after and while reading the book to get ideas, share new discoveries, and just keep informed on what's going on.

- Live coding and terminal tweaking
- Chat via IRC on the screen
- Whatever searches and AI queries I'm doing
- Rotating ASCII art of all different kinds (some that I will create)
- Quotes and dad jokes on the screen
- Current event summaries from some reliable source
- Weather dictated by commands in the chat
- Maybe a live chat using ntalk

I'm already getting negative feedback about people "missing my voice on these live streams" and it's my own fault. Over the holidays I built up the expectation that I would live stream all day and code and talk through the whole thing. It was never intended to last beyond the holidays.

This plays nice with the original idea of having the chat—from whatever service—able to dictate what happens in the terminal when I'm not able to do something interactive. In fact, since all my writing is here and in AsciiDoc I can make commands that pull up the help docs on the screen, or activate terminal rendering of my AI queries.

Fewer talking streams

Sunday, January 5, 2025, 10:59:49AM EST

> Let all your efforts be directed to something, let it keep that end in view. It's not activity that disturbs people, but false conceptions of things that drive them mad." —SENECA, ON TRANQUILITY OF MIND, 12.5

Yesterday I streamed for 14 hours, most of it sitting next to my wife doing her thing. I got a ton of stuff done. This morning I woke up refreshed and so happy for some reason. I'm pretty sure it is because of how much stuff I got done:

- 14 blog posts
- Most of my dotfiles scripts cleaned up
- Relearning Perl
- Absolutely zero unrelated rabbit holes
- Able to stay on task and focused even when chat was flowing

Still just as many viewers

It blows my mind that I averaged about 30 viewers the entire day, without saying a single thing. And I was actually able to engage and see what they had to say even more because their chat is on the screen.

Better viewers, doing their own thing

People who keep me company on my streams often had their own projects going. The pace of a screencast stream is naturally slower, encouraging viewers to multitask—sometimes even watching other streamers. I think that's ideal. I've always been bothered by the idea that, by promoting constant engagement, I might be stealing time from my viewers—time they could spend focused on something meaningful.

Screencasting allows me to motivate others **by doing**, answering relevant questions in real-time, and capturing those answers in a way that's immediately searchable. I can even turn them into reusable commands for future viewers with the same questions. Honestly, this feels like the ideal setup.

The focus is on building a *community of quality* — people who are genuinely interested in learning and working on something important—rather than entertaining an audience with surface-level energy and antics. As I've always said, it's about *quality* of community over *quantity* of viewers.

I have one of the very best communities on the Internet, and I love keeping it that way. The long-running nature of my screencast streams acts as a natural filter, keeping away the fleeting, mediocre masses I have no desire to know or interact with.

That might sound elitist, but honestly, I don't care. I would much rather invest in **the elite**—the focused, the motivated—and help them develop the skills to change the world, rather than entertaining intellectually lazy livestream wanderers who stumbled in by accident while aimlessly scrolling for distractions.

Better physical shape

I don't think people realize the serious tax on the body it takes to live stream. Even a short live stream where I am talking and responding is crazy hard on my body. In contrast, screen casting yesterday for 14 hours actually left me rested and fulfilled. I was able to move all over, take breaks, even talk to my wife on occasion while also still streaming and following what was happening.

The single biggest factor in physical live streaming is my voice. Substantial streamers have actually lost their voice on occasion. For some reason it is speaking that is harder to control and throttle. This is the same when I go biking. When I'm just on the bike and enjoying it the experience—for me—is much better.

Writing is therapeutic

Writing has always been a much better release for me than speaking. I do love to talk, but writing doesn't trigger me as much. I'm able to think about the topic, edit it, revise it.

Shell + Perl + Go = Bliss

Saturday, January 4, 2025, 6:13:17PM EST

While cleaning up all my dot files as the holidays wrap up I have reached the perfect state of coding bliss in the terminal, the perfect combination of tools that overlap exactly the way it should be. I don't know why I ever changed. The coverage that simple Shell, Perl, and Go provide is absolutely amazing. I'm definitely covering all three for terminal development in my *Terminal Velocity* book (eventually). I had to go through this migration to fully realize just how right that decision is.

POSIX shell is ubiquitous

No not `ksh` (which is *not* POSIX), not `zsh`, not `bash`, not even `dash`, but pure POSIX shell. POSIX shell scripts will run anywhere. Usually they are gluing together a bunch of other existing things on the system. That's its purpose, to glue stuff together at a minimal level and take the place of functions and aliases that are anti-patterns for those wanting a pure Unix filter experience. Here are a few examples from plain POSIX shell.

This first one called simply `what` is very typical one-liner—often a pipe—that can be called anywhere and combined with other commands in Unix pipelines:

```
#!/bin/sh
head -1 "$HOME/.currently" | wee
```

A big more complicated but the same idea, `pre` is a typical Vim filter script:

```
#!/bin/sh
while IFS= read -r line; do echo "${1:-#} $line"; done
```

Here's the `wee` command that dumps whatever is passed as an argument or on standard input to the WeeChat IRC socket:

```
#!/bin/sh
buf="$*"
if test -n "$buf"; then
    echo "*$buf" | toemoji >"$HOME/.weechat/fifo"
    exit
fi
while IFS= read -r line; do
    echo "*$line" | toemoji >"$HOME/.weechat/fifo"
done
```

I can literally cut and paste anything, anywhere to my chat on Twitch and in IRC and indirectly to YouTube using this method. Someone wants an interesting block or URL? I just `!!wee` from within Vim to send it to them.

Scripts this short don't need any extra white space for blank lines. Don't bother and you'll save the compiler just that much extra work. (Although a `;` is the same as a single `\n` so no sense smashing it all onto one line unless you have a lot of indentation. And *never* indent with spaces. Tabs are the official method of indentation and if you use `shfmt` as your Vim/Ale fixer you will never have to worry about it.

Here's `liveicon` that is embedded in my TMUX configuration file so it appears on the screen:

```
#!/bin/sh
sec=$(date +%s)
test -e "$HOME/.state/muted" && echo "" && exit
! test -e "$HOME/.state/recording" && exit
if [ $((sec % 2)) -eq 0 ]; then
    echo 
else
    echo "  "
fi
```

Then I just toggle the status with `recording` and `muted`:

```
#!/bin/sh
file="$HOME/.state/recording"
if test -e "$file"; then
    rm "$file"
else
    touch "$file"
fi
```

Perl is a "miniature Linux"

> "Perl is, by and large, a digested and simplified version of Unix. Perl is the Cliff Notes of Unix. I could never really get myself to learn sed, awk, zsh, grep, find, m4, pipes, xargs, tee, cut, paste, yacc, lex, various IPC or even C for that matter. I ought to. In practice, in almost all cases I use Perl." - Larry Wall

Every other GNU/Linux utility can be rather easily replaced with Perl—especially `awk`. Perl's drop-dead simple handling of signals and forking as well as a simple backtick subprocess syntax make it the idea glue between simple shell scripts and other compiled programs. Just look at this example:

```
#!/usr/bin/env perl
use v5.36;
use File::Basename;
```

```
not `command -v figl` and say 'figl not found' and exit 1;
not `command -v lolcat` and say 'lolcat not found' and exit 1;

my $clear  = "[2J" ;
my $curoff = "[?25l" ;
my $curon  = "[?25h" ;
my $top    = "[H" ;

my $what = join ' ', @ARGV;
if ( not $what ) {
    print "Text: ";
    $what = <STDIN>;
    chomp $what;
}

sub cleanup {
    print "$curon$clear";
    exit 0;
}
$SIG{'INT'}  = \&cleanup;
$SIG{'TERM'} = \&cleanup;

print $clear. $curoff;
while (1) {
    print $top. `echo "$what" | figl | lolcat -f`;
    sleep 1;
}
```

That example has a little of everything, besides maybe `pack` and system calls (which I wrote about earlier). This is where Perl really shines. It absolutely destroys bash for this sort of thing. I realized that after three years bashing my head against a wall. Don't be me. Learn Perl today.

Do *not* create Perl code with *any* dependency on CPAN libraries. That defeats the entire core value proposition of Perl. At that point use Go (or *anything* that statically cross-compiles into a binary that has no dependencies). Trust me on this one.

Here's another example (`fields`) that puts `awk` to shame:

```
#!/usr/bin/env perl
use v5.36;

my $input = shift;
my $delim = shift || '\s+';
my @nums;

if ($input) {
    for my $chunk ( split /,/, $input ) {
```

```
        if ( $chunk =~ /^(\d+)\.\.(\d+)$/ ) {
            push @nums, $1 .. $2;
        }
        elsif ( $chunk =~ /^\d+$/ ) {
            push @nums, $chunk;
        }
        else {
            warn "Skipping invalid chunk: $chunk\n";
        }
    }
}

while (<STDIN>) {
    my @fields = split /$delim/;
    if ( not @nums ) {
        push @nums, 1 .. scalar(@fields);
    }
    my @out = ();
    for my $i (@nums) {
        if ( $i <= scalar(@fields) ) {
            push @out, $fields[ $i - 1 ];
        }
    }
    print join( " ", @out ), "\n";
}
```

I use this all the time as a Vim filter to eliminate columns in stuff with just `fields 2..4,5` type of syntax.

Go for the big stuff

As soon as something grows beyond the rational limits of Perl, which is far earlier than most developers who have made Perl into an entire middleware layer should have done, there is the Go language sitting in that ideal spot.

It is super fast to write, simple to grok, powerful, and has every battery included you could ever need for terminal applications—especially terminal interface apps. The Charmbracelet stuff is absolutely gorgeous and CLI frameworks like Bonzai make Go the obvious favorite for a terminal junky who wants to stay productive producing useful, reusable, shareable tools.

Here's an example of my Go YouTube API command line tool (`yt`) that uses Bonzai. You see that there is variable persistence with environment variable overrides, embedded documentation in Markdown rendered to the terminal, tab self-completion, and more. (This `cmd.go` file actually uses a high-level set of functions in another file as well.)

```
package yt

import (
    "context"
```

```
    "os"
    "text/template"

    "github.com/rwxrob/bonzai"
    "github.com/rwxrob/bonzai/cmds/help"
    "github.com/rwxrob/bonzai/comp"
    "github.com/rwxrob/bonzai/json"
    "github.com/rwxrob/bonzai/persisters/inprops"
    "github.com/rwxrob/bonzai/term"
    "google.golang.org/api/option"
    "google.golang.org/api/youtube/v3"
)

var yt *youtube.Service
var chanid string
var apikey string
var nextpage string

// ------------------------------ Cmd ------------------------------

var Cmd = &bonzai.Cmd{
    Name:  `yt`,
    Short: `interact with YouTube API`,
    Comp:  comp.CmdsAliases,
    Def:   help.Cmd,

    Cmds: []*bonzai.Cmd{
        help.Cmd, startCmd, videoCmd, messagesCmd, detailsCmd,
        chatidCmd, nextpageCmd, chanidCmd,
    },

    Persist: inprops.NewUserConfig(`ytwee`, `properties`),
    Vars: bonzai.Vars{
        {K: `yt-channel-id`, E: `YTCHANNELID`, P: true, G: &chanid},
        {K: `yt-api-key`, E: `YTAPIKEY`, P: true, R: true, G: &apikey},
        {K: `yt-chat-next-page`, P: true, G: &nextpage},
    },

    Init: func(x *bonzai.Cmd, _ ...string) error {
        ctx := context.Background()
        service, err := youtube.NewService(ctx, option.WithAPIKey(apikey))
        if err != nil {
            return err
        }
        yt = service
        return nil
    },
}

// --------------------------- messagesCmd ---------------------------
```

```
var messagesCmd = &bonzai.Cmd{
    Name:  `messages`,
    Short: `print up to 200 messages`,
    Vars:  bonzai.Vars{{I: `yt-chat-next-page`}},
    Do: func(x *bonzai.Cmd, _ ...string) error {
        vidid := FetchVideoId(yt, chanid)
        chatid := FetchChatId(yt, vidid)
        messages, nextpage, err := FetchMessages(yt, chatid, nextpage)
        if err != nil {
            return err
        }
        x.Set(`yt-chat-next-page`, nextpage)
        tmpl, _ := template.New("message").Parse(`{{.Author}} {{.Text}}` + "\n")
        for _, message := range messages {
            tmpl.Execute(os.Stdout, message)
        }
        return nil
    },
}

// ---------------------------- startCmd ----------------------------

var startCmd = &bonzai.Cmd{
    Name:  `start`,
    Short: `start relaying messages`,
    Do: func(x *bonzai.Cmd, _ ...string) error {
        println(`would start relaying`)
        return nil
    },
}

// ---------------------------- videoCmd ----------------------------

var videoCmd = &bonzai.Cmd{
    Name:  `video`,
    Short: `unique id of current live stream video`,
    Long: `
        This is the **most** _expensive_ operation available so ***use with
        caution*** and generally only once a stream to cache it someplace
        and reuse it.
    `,
    Do: func(*bonzai.Cmd, ...string) error {
        vidid := FetchVideoId(yt, chanid)
        if vidid != "" {
            term.Print(vidid)
        }
        return nil
    },
}

var chanidCmd = &bonzai.Cmd{
```

```
    Name:    `chanid`,
    Short:   `set or get the channel ID`,
    Vars:    bonzai.Vars{{I: `yt-channel-id`}},
    MaxArgs: 1,
    Do: func(x *bonzai.Cmd, args ...string) error {
        if len(args) > 0 {
            x.Set(`yt-channel-id`, args[0])
            return nil
        }
        term.Print(x.Get(`yt-channel-id`))
        return nil
    },
}

// ---------------------------- detailsCmd ----------------------------

var detailsCmd = &bonzai.Cmd{
    Name:  `details`,
    Short: `live stream details`,
    Do: func(*bonzai.Cmd, ...string) error {
        vidid := FetchVideoId(yt, chanid)
        details, err := FetchStreamDetails(yt, vidid)
        if err != nil {
            return err
        }
        if vidid != "" {
            term.Print(json.This{details})
        }
        return nil
    },
}

// ---------------------------- chatidCmd ----------------------------

var chatidCmd = &bonzai.Cmd{
    Name:  `chatid`,
    Short: `live stream chat unique identifier`,
    Do: func(*bonzai.Cmd, ...string) error {
        vidid := FetchVideoId(yt, chanid)
        chatid := FetchChatId(yt, vidid)
        if chatid != "" {
            term.Print(chatid)
        }
        return nil
    },
}

// ---------------------------- nextpageCmd ----------------------------

var nextpageCmd = &bonzai.Cmd{
    Name:  `nextpage`,
```

```
    Short: `print the next page token that has been cached`,
    Do: func(*bonzai.Cmd, ...string) error {
        term.Print(nextpage)
        return nil
    },
}

var isliveCmd = &bonzai.Cmd{
    Name:  `islive`,
    Short: `check if channal is live`,
    Do: func(*bonzai.Cmd, ...string) error {
        term.Print(nextpage)
        return nil
    },
}
```

Nothing wrong with other languages

Please do not misunderstand. This is not me saying those are the only languages out there or even that they are the only tools for the job be maintaining a personal terminal development environment the best. I know people who accomplish the entire thing using nothing but Emacs and Lisp. That's just not me. I also use a Rust terminal (WezTerm) and C when required to do low-level stuff that Go doesn't cover. It's just that Shell + Perl + Go covers every major need there is for coding utilities and tools for my personal use in the terminal. Everything else is highly specialized.

Why no Python?

> Python gets a special mention since I ported all my tools to python in 2014 and seriously regret having done that.

Python has very specific uses—especially machine learning (for some stupid reason even though C/C++ is doing all the work). The terminal is *not* one of Python's strong suits and it never has been. Just compare a simple Perl one-liner to Python. There's no contest.

The worst part about Python is how heavy and bloated it is compared to Perl. Most Python programs should be Go programs and a few small ones would be fine as Perl. There is simply never a good reason to willingly force yourself to do the Python virtual environment dance every time you want to use your dotfiles scripts. While it is true that most systems will not have Perl v5.36 it is true that those scripts that require it would be better served by installing Perl than Python.

But Rob, all these hacker tools exist in Python?

Yep, they do, and many are absolutely amazing. Pretty much every single one of them would have been a whole lot better in Go but Go wasn't really around when they where made.

The hacker community is fickle and divided. The leading web pwning toolkit was written in Java for years. Metasploit had to apologize for "Ruby only" and eventually allowed anything into it even though they made a big deal about picking Ruby instead of Perl—that's right—*not* Python. The point

is hackers are always using whatever their favorite flavor of the day is. A few tools stick around and get big enough to maintain their inertia.

If I'm hacking and needing a tool that is written in Python then I'll either port it to Go or use it as it just like I do with AsciiDoctor in Ruby.

`type` or `which` or `command`

Saturday, January 4, 2025, 4:03:37PM EST

I was pleased to see that `type` works in a Perl `backtick` pipeline if the underlying shell is `bash`, `ksh`, `zsh`, even `/bin/sh` on macOS, which is close to `ksh`. But is it good for POSIX shell scripts? Definitely not. In fact, `type` is amazing for use interactively but should probably never be used in a script, even though I had forgotten why until just now when I did the research.

Here are the options as a reminder:

Task	`type`	`command -v`	`which`
Check built-in/alias/function	✅ Yes	✅ Yes	❌ No
Check shell built-in	✅ Yes	✅ Yes	❌ No
Check shell alias	✅ Yes	✅ Yes	❌ No
Check shell function	✅ Yes	✅ Yes	❌ No
Check executable in `$PATH`	✅ Yes	✅ Yes	✅ Yes
Portability	❌ No	✅ POSIX	⚠️ Varies
Performance	✅ Fast	✅ Fast	⚠️ Slow

Interactive command line usage: `type`

I was burned pretty hard recently by using `which` out of habit. (Let's be honest, it's a lot faster to type than `command -v`.) I have several decades of muscle memory that were wrongly programmed to type `which` instead of `type`. On this one occasion I had recently added a new script to take the place of another and change the location so that PATH resolved differently. (I see you cringing already.) I ran it and still got the old thing even though it didn't even exist. It was as if the old one was gone and the new one—in my path—was not being seen. The fix was simply `hash -r` but had I used `type` I would have seen that immediately.

Keep in mind that `type` *is not POSIX* no matter what shell supports it.

Shell	`type` Support	`command -v` Support
bash	✅ Yes	✅ Yes
zsh	✅ Yes	✅ Yes
ksh	✅ Yes	✅ Yes
dash	⚠️ Partial	✅ Yes
sh (POSIX)	❌ Not guaranteed	✅ Yes

The `type` command also requires annoyingly to redirect the output every time.

Call to know which is in PATH: `which`

The `which` command returns the full path as well as whether or not it exists. Most people have used this all their terminal-using lives. It does have one *major* weakness. If the goal is to check what would happen in a user's experience when that thing is added to a pipeline potentially called from within a calling executable then the response would be drastically wrong. This can lead to all kinds of obscure but very serious bugs. It's also the main reason I only use shell functions and aliases in a very limited way so they don't potentially conflict with code someone has written that assumes `which` is a safe way to determine if that thing they are typing or calling is safe. I don't have to tell you how easy such a program would be to own by adding a function that is the same name as that executable binary the program is expecting. There. I gave you yet another attack vector: which injection.

Call to know if *anything* callable exists: `command -v`

This is the only safe way to ensure that something can be called or is not already callable by the parent shell. This is why you will see this method used in most textbooks and courses about Unix shell scripting. Hopefully, this memory jog will help you understand why.

Scripts without suffixes

Saturday, January 4, 2025, 3:08:59PM EST

Very few things communicate "I'm a noob" more that simply adding `.py` or `.pl` or `.sh` to the end of your *executable* scripts (not libraries)—even more if you hard-code them into pipelines used by other scripts. If I were interviewing a person for a SysDE position (and I've interviewed dozens) and they claimed they did this for all their tools I would immediate reject them. It demonstrates that they have no fundamental understand of Unix at all—at least not as a tool smith and developer (which is what a SysDE requires).

```
foo.py | bar.pl | mysort.sh
```

The reason no one should willingly add suffixes to any executable that could ever be used in a pipeline at all—including installation logs—should be glaringly obvious to anyone who truly understands the Unix philosophy and the very thing that makes using the terminal so powerful in the first place. Unfortunately, it isn't. I mean, it *really* isn't.

Who started it?

That idiotic practice of suffixing *everything* started in the probably 90s because of Windows systems requiring them to know what type of script it is because Windows can't do shebang lines. Why would we want a Windows practice to persist into Unix land?

Some of us have noticed that web developers and "full stack developers" tend to do this as well, which tracks because as a community they are the most likely to use a terminal without having a clue about what Unix even is, you know, the type who ask me "what's your distro?" These developers have suffixes on everything they make all the time because the web requires it, so, to them, it follows that it must be true for the scripts in their dot files scripts directory—if they even have one.

> Being a vendor who does this does not justify it. My friends and I bitch all the time about vendors who just do not understand this and yet are multi-million dollar companies. But these same companies have no problem corrupting my `~/.bashrc` without my confirmation at all either. Being a vendor of any size doesn't mean anything—in fact—it means you should probably be *more* cautious when dealing with them—especially IBM.

I see professionals argue about this as if it is defensible all the time. I just laugh every single time. Do I look down on them? Absolutely. They instantly lose every shred of respect in my eyes because it means *they do not understand Unix at all*. Wait, I already said that. Oh well, it cannot be said enough.

What about symlinks?

Some have suggested that links take care of this problem. That the original can have the suffix but

the symbolic or hard link can not have it. While this is a good hack to *fix* whatever someone did to add suffixes that you cannot now avoid, links are never a defense of doing it wrong to begin with.

We aren't talking libraries

We aren't talking about *libraries* and source files, anything that is *not* executable without explicitly providing the interpreter on the command line first. Those tend to make a lot of sense because it makes no sense to put a shebang line in them and no editor on the planet is smart enough to guess based on the content alone. In fact, using suffixes for this sorts of things should probably be a mandatory best practice in order to disambiguate them and keep editors happy.

The only exception

There are *extreme* exceptions to this like for `install` when there is a `.bat` and a `.sh` and a `.py` of the same thing. Vendors love to so this and they are counting on their script getting executed once or so and not really being joined in pipeline permanently. But even then it is really not that justified—especially if there is only one install script. People use this exception to justify creating all their home scripts with suffixes because "all interpreted scripts should have a suffix" and they are monstrously *wrong*.

Always script over function

Saturday, January 4, 2025, 2:46:10PM EST

Moving back to Perl has reminded me of a best practice that I started to seriously violate by doing everything in Bash: the use of functions and aliases for things that should really be stand-alone executables—even if just one line. **You cannot use functions and aliases as filters in Vim and Neovim**.

Yes, I know you can force functions to be visible to shell scripts provided they are also in the same language, but that is not the same thing and doesn't begin to address the problem.)

I hit this problem when I went to port my use of the following shell idiom to check for the existence of something that ends up running:

```
! type "foo" >/dev/null && echo "foo not found" && exit 1
```

While it is true that `which` and `command -a` do similar things, they only return true if an executable is in the `PATH`. Only `type` ensures that you are detecting aliases and functions as well. Only `type` catches if `foo` is still hashed to an old executable location in the `PATH` instead of the new one or worse *even after it has been removed from the system entirely* and not yet been rehashed with `hash -r`. (I promise, if you haven't hit this problem you will eventually.)

But why would you even want that? Having an alias mask a command makes sense for interactive things that you just don't want to have to type out in long form. Masking an executable with a function might allow you to wrap it and call with the full path (like I do with `lynx`). But at the end of the day, you really just want to get in the habit of creating executable scripts that can be changed into whatever language later. This is why you **should never, ever add a suffix to the end of an executable program of any kind**.

Packing a syscall in perl

Saturday, January 4, 2025, 2:22:40PM EST

I have been completely freaking out about how amazing Perl is. I am really kicking myself for moving anything over to Python and Bash all those years ago. I was so much more efficient and able to create useful code so much more quickly with Perl and I can feel that power returning as I remember it. One of those complete *oh my god* moments was remembering that I can create and send syscalls directly and pack and unpack binary data. Here's a sample:

```
#!/usr/bin/env perl
use v5.36;

use constant TIOCGWINSZ => 0x5413;

my $winsize = pack('S4', 0, 0, 0, 0);
if (ioctl(STDOUT, TIOCGWINSZ, $winsize)) {
    my ($rows, $cols, $xpix, $ypix) = unpack('S4', $winsize);
    print "Number of terminal lines: $rows\n";
} else {
    warn "Unable to get terminal size: $!";
}
```

This is just a simple call to get the number of rows, and yeah it's overkill when `tput lines` does exactly the same thing, but just the fact that you can do it blows my mind. If you consider yourself a systems operations specialist or hacker you have to appreciate how succinct and amazing that is.

Awk is never worth it

Saturday, January 4, 2025, 12:29:40PM EST

Given the size of the `awk` binary versus even the latest `perl` it makes no sense to use *any* POSIX shell script that calls out to `awk` if you are sure `perl` is going to be on the system.

```
-rwxr-xr-x  1 root  wheel   295K Aug  4 06:31 /usr/bin/awk
-rwxr-xr-x  1 root  wheel   134K Aug  4 06:31 /usr/bin/perl
```

In fact, as soon as you reach for `awk` or `tr` or `cut` or `sed` there is a good chance you have crossed over into `perl` territory and are no longer using the best tool for the job. You are likely not only going to invoke one subprocess for `awk` but another one later for one of these other POSIX requirements. Even if you use the `/usr/bin/env perl` shebang line you might as well write the entire script in `perl` at that point which ***runs in a single process*** once it is started and has ridiculously better text processing functionality for *half* the amount of RAM consumption.

Even today most systems that have `awk` installed will have a version of `perl` installed that will do what you want more cleanly and use less resources.

#/usr/bin/env perl

Saturday, January 4, 2025, 10:07:30AM EST

Normally I really hate using `/usr/bin/env` for anything if for no other reason than being completely unnecessary extra process. I also come from a time when such things were considered extremely insecure and we set `PATH` explicitly at the top of all our shell scripts.

God bless macOS for not sucking, except for `brew`. I absolutely love it but having everything in `/opt/homebrew` really screws up depending on any sort of reliable location for expected binaries in *any* interpreted language shebang line, which, if I'm completely honest, was a significant consideration in my decision to switch to `/bin/sh` instead of `/usr/bin/env bash` which is mandated if you want to use the latest bash on macOS. It just made everything that much slower to execute unnecessarily.

For what it's worth, this practice is explicitly recommended in the current official man page for Perl itself so I guess I'll get over it.

Perl chomp is dead

Saturday, January 4, 2025, 9:50:12AM EST

After a very frustrating fight with new Perl having lost a lot of knowledge about it I learned that the age-old `chomp` in Perl is still there but for some reason is not as dependable. I'm sure it has something to do with the `$/` the `$INPUT_RECORD_SEPARATOR` which is normally `\n` and now that I'm on several different types of systems (macOS and Linux) it just doesn't behave correctly. I specifically hit this in my `buildadoc` script (posted earlier).

Rather than use it I use the Unicode white space `\R` instead. That's right, good 'ol `\S` has been replaced with one that is Unicode aware.

```
#!/usr/bin/env perl
use v5.36;
my $pwd = `pwd`;
$pwd =~ s/\R$//;    # better than chomp
my ( $parent, $name ) = $pwd =~ m,/([^/]+)/([^/]+)$,m;
say "$parent-$name.txt"
```

Building AsciiDoc

Saturday, January 4, 2025, 9:37:10AM EST

Here's the latest script that I use to build this site and content.

```
#!/usr/bin/env perl
use v5.36;

# total the hours from the main index.adoc page
open my $fh, '<', './docs/index.adoc' or die "Could not open file: $!";
local $/;
my $buf          = <$fh>;
my $hours_mentor = 0;
map { $hours_mentor += $_ } $buf =~ /hours="(\d+)"/g;
my $hours_self  = $hours_mentor * 2;
my $hours_total = $hours_self + $hours_mentor;
my $hours_week  = $hours_total / 15;

# set the exact date and time of the build
my ( $sec, $min, $hour, $mday, $mon, $year ) = localtime();
$year += 1900;
$mon  += 1;
my $date = sprintf "%04d-%02d-%02d %02d:%02d:%02d",
  $year, $mon, $mday, $hour, $min, $sec;

# HTML
qx{ asciidoctor \\
  -D docs \\
  -a date="$date" \\
  -a allow-uri-read \\
  -a hours_mentor="$hours_mentor" \\
  -a hours_self="$hours_self" \\
  -a hours_total="$hours_total" \\
  -a hours_week="$hours_week" \\
  -o index.html \\
  ./docs/index.adoc
};

# HTML without table of contents
qx{ asciidoctor \\
  -D docs \\
  -a date="$date" \\
  -a toc! \\
  -a notoc=true \\
  -a allow-uri-read \\
  -a hours_mentor="$hours_mentor" \\
  -a hours_self="$hours_self" \\
  -a hours_total="$hours_total" \\
  -a hours_week="$hours_week" \\
```

```
  -o notoc.html \\
  ./docs/index.adoc
};

my $pwd = `pwd`;
$pwd =~ s/\R$//;
my ( $parent, $name ) = $pwd =~ m,/([^/]+)/([^/]+)$,m;

# PDF
qx{ asciidoctor-pdf \\
  -D docs \\
  -a date="$date" \\
  -a allow-uri-read \\
  -a hours_mentor="$hours_mentor" \\
  -a hours_self="$hours_self" \\
  -a hours_total="$hours_total" \\
  -a hours_week="$hours_week" \\
  -o "$parent-$name.pdf" \\
  ./docs/index.adoc
};

#EPUB
qx{ asciidoctor-epub3 \\
  -D docs \\
  -a date="$date" \\
  -a allow-uri-read \\
  -a hours_mentor="$hours_mentor" \\
  -a hours_self="$hours_self" \\
  -a hours_total="$hours_total" \\
  -a hours_week="$hours_week" \\
  -o "$parent-$name.epub" \\
  ./docs/index.adoc
};
```

The script is updated every time I build so might not reflect the date of this blog post.

Different formats

You'll see that `asciidoc` gets run four times:

1. HTML site with TOC
2. HTML site without TOC
3. PDF
4. EPUB

I could add any other formats that I need.

Calculate hours

I have hours calculation built into the current default. I'll probably remove that later since not everything has hours in it. That was specifically for *Coding from the Get Go*. It is nice to have though for other things. As the volume of text gets larger though it might have to go.

Subshell fail

Saturday, January 4, 2025, 9:00:00AM EST

I used to think that by encapsulating everything in any of my shell scripts into a function and calling it at the end of the script was a good thing even though it meant there would be a forced subshell execution. Here's an example:

```
#!/usr/bin/env bash

# Outputs the seconds since the thing identified by the first argument
# was last modified (not created).

ageof() {
    local path="$1"
    if [[ -z "$path" ]]; then
        echo 'usage: ageof <path>'
        return 1
    fi
    echo $(($(date +%s) - $(date -r "$path" +%s)))
}

ageof "$@" # OMG WHY THE SUBSHELL FOR NO REASON!!
```

I'm mortified with embarrassment for doing this.

My logic was that I could easily `daf` the function into other stuff later. But by just having it as a single command I can simply remove the shebang line and include it just as easily and give it a name that is better fitted for whatever I'm putting it in.

Honestly, however, if it is reusable code should either stay a single command or be written in Go in a more reusable way.

MkDocs fails writers

Saturday, January 4, 2025, 8:13:38AM EST

Even though MkDocs is beautiful and still probably good for technical documentation it fails to fulfill the needs of a writer who wants more.

No official EPUB or PDF support

While it can be done, it requires a bunch of Pandoc integration and adaptation. AsciiDoc has it built in.

Not for actual book publishing

More books have been officially published with AsciiDoc than anything else. Pandoc might be a close second. MkDocs is not for actual book publishing and never was. Use the right tool for the job. AsciiDoc was created above all to publish books as well as web content.

Vastly inferior template language

At first the AsciiDoc format is harder to grok than Markdown. Once you learn it though, you begin to realise how much better it is. Just tables are one example.

Cannot include remote URIs (without plugin)

Even though it has to be enabled (for good reason), remote URI inclusion is perhaps one of the most powerful features for writers who maintain multiple copies of books that sometimes share the same content.

```
include::https://raw.githubusercontent.com/asciidoctor/asciidoctor/main/README.adoc[]
```

As far as I am aware, this cannot be done from any other writing framework (without modification or extension).

Much harder metadata manipulation

Because AsciiDoc is designed to create documents and not searchable documentation it has much better metadata handling and tagging. This makes my content easier for search engines to process and my AI able to find the right thing.

Cannot handle large volume

After testing MkDocs with 2000 Zettelkasten nodes earlier it fell on its face. The design of how it indexes keywords is downright idiotic. I did discover some convoluted ways to get around it but it is much nicer not to have to bother with any of that.

Poor keyword searching

It turns out that keyword searching is actually a bad thing for most content. This is not intuitive at all.

In order for keyword searching to be of value it has to load the entire keyword index—often as a full-site PWA—but at that point why not just put the entire document on the screen and let the reader's powerful builtin browser searching with regular expressions be used instead?

Perl: `use v5.26`

Friday, January 10, 2025, 3:02:40PM EST

Update: Discovered that version 5.26 is the defacto on most systems including my multi-user system at work. Also just learned today that in Q1 2025 the Git project will add dependency on 5.26 (currently 5.8) which means any system that has the `git` command also must have `perl` even if the path might be weird if not installed for the whole system.

The only thing that I care about that we lose is `use warnings` but that is not a big deal since I have Ale linters that catch all the things that `use warnings` would tell me anyway.

Saturday, January 4, 2025, 7:51:24AM EST

Now that I have returned to my beloved Perl in 2025 (disposing of bash for all personal scripting after being really burned by it) I have been discovering a bunch of new changes that are very convenient for modern Perl development. Keep in mind that these are designed to make use of Perl *on a personal workstation* where its primary purpose is to create really amazing text processing filter chains that can be easily incorporated into all versions of `vim` and `vi` (as opposed to brittle, bloated, non-transportable plugins to do the same thing).

No more `use strict`

Yep. Any `use v5.12` or higher gets that by default.

No more `use warnings`

Yep. Any `use v5.36` or higher gets that by default.

No more `-E` (`say` just works)

Unfortunately, the amazing `say` keyword just doesn't work by default. I've always found this annoying given how important it is as an alternative to `print "some\n"` with the line returns every time. This is included by default since it is activated with `use v5.10` or higher.

Frequent, disposable commits

Saturday, January 4, 2025, 6:55:14AM EST

Squash and merge with disposable commit messages that really don't matter because they are all thrown away at the end in exchange for one PR merge commit that includes an extensive change log are really the only viable git PR management method if you want to be able to use draft PRs as a method of sharing ongoing work while it is happening—and you definitely should.

Who is the commit message for?

Who are all those messages for anyway? Are they really for future generations to read and understand the history of the project? I don't think so. That is why the change log exists.

The use of commit messages are to convey the current work being done to others following the draft PR during its execution.

In a way, commit messages are like an ongoing chat about the work on the project that is happening in real time. Everyone hates that guy who writes a ton in a chat room forcing everyone to read it. Shorter is better.

Conventional commits are a waste of time

This is going to really anger some people but it has to be said. All that meticulous attention to "conventional commits" is a monumental waste of time. No one cares how pretty the commit messages are.

"But Rob, don't you think having automated change logs is worth it?"

No, no I do not. Because that assumes you can automate something that should never be automated.

Besides, the idea of automating change logs is a huge pipe dream. You can *never* guarantee that everyone uses commit messages the same way. Therefore, the wasted time attempting to enforce it just stifles and slows development. Why berate the developers who have painfully decided to work on your project just because they didn't pick the exact perfect works to describe the commit.

Not everyone speaks English

The assumption that commit messages are going to be beautiful also assumes you have perfect English speakers on the project—which describes absolutely zero projects worth following. These faithful committers are a cherished commodity to be encouraged and made to feel welcome and if you hit them with push back on their "bad commit messages" they will just walk and they would be right to do so. You are being a jerk about something that absolutely, positively does not matter. Requiring "good commit" messages just proves you have no idea how to run a project. At least good developers know how to identify that early in the project and stay the fuck away.

Change logs should *never* be automated

The change log is a human document for human consumption and attempting to automate their generation *never* works. Just look around. You will find hundreds of projects with massive changelogs that are completely unintelligible *because* they are automated instead of having a human—or better yet an AI—summarize the changes in a way that can be grokked easily by the humans bold enough to open it and read it. Huge, automated change logs cause readers to avoid them like the plague they are. Only engineers who don't work with people think automating this is a good idea. Anyone responsible for the marketing and documentation of a project hates that and refuses to accept their automatic creation.

Commit early, commit often

The mantra *commit early, commit often* is far more important than any convention about the content of the message itself. By reducing the friction on the commit message to something that is reasonably distinct but short and sweet you encourage yourself and your project developers to commit all the time making the visibility of the work on the project—as well as the possibility of lost work—even higher. I want this more than anything. I want to see the small decisions about how the code is evolving while it is happening over some big, meticulously crafted chunk of code on a massive commit. It makes the commit each much easier to monitor by looking at the diffs quickly and frequently by the others on the team.

Perpetual publishing best

Friday, January 3, 2025, 6:07:50PM EST

I have only been using the *perpetual publishing* model (a term I coined) for knowledge management for a few days and have already seen *tremendous* benefit from it.

Chloe (my AI) sees my writing immediately

The single biggest advantage is that by making my writing all into plain ol' web docs on the public Internet that my ChatGPT AI *immediately* has access to my entire writing. This means I can immediately leverage AI for all sorts of things related to the writing:

- Have her check the punctuation and grammar dynamically and report
- Create IEEE bibliographic entries pointing to it for other documents
- Have her comment on what it is about and expound
- Catch legal considerations—especially as I write about sensitive Mormon topics

This means that anyone else using ChatGPT automatically can look at it as well by pointing their AIs at it and asking them to read it and digest it for them according to their preferences for tone and contextual history they have with their personal AI assistants. Ultimately my content will be added to the collective knowledge base that will populate *all* AIs within a few months. ***None of that happens unless I publish a web document that is not a Zettelkasten.***

Let me restate that because it is *so* massive. I can tell people in my community to summarize my content into digestible ways *according to their preferences and learning styles for them* by using their personal AIs. The entire fiasco of finding a compatible author with a voice and rhetorical style is *dead* in the age of AI LLMs with personalized assistants.

This discovery is so important for everyone to grok immediately—techies and non-technies—I'm going to focus on the new book Chloe suggested: *A Knowledge Warrior's Guide to Perpetual Publishing*.

Flesh out ideas and migrate to other "books"

I can blog about things and work them out and if they develop into something more I can simply move them to another repo containing the other book. I have done this a lot with stuff between my blog and faq, for example, and sometimes directly into other books like *Terminal Velocity* and *Coding from the Get Go*. For example, that list of advantages that I just discovered will get copied directly into my other book. What's even better is that I can use exactly the same text in both places by using remote URL `includes::` with AsciiDoc. I was a *fool* for not using AsciiDoc from the very beginning. ***No other writing framework allows for remote URL includes without modification, period.***

Sense of urgency abated

I get really freaked out when I make amazing discoveries like this and don't have a way to immediately capture them for myself later—but more importantly—*share* them with other people who can immediately benefit. Writing this down is always going to be more sustainable that a bunch of videos someone would have to slog through because my knowledge will become part of the AI global collective. My videos will not. I can always make a video later about the stuff that has been written and that video content will be many orders of magnitude better in quality because it will have been worked out enough in words first—plus it's more inclusive.

Easier to open and edit

A simple program like this one opens up a new blog post from within a GitHub repo, one repo per book:

```
#!/bin/sh
mark='// NEW POST HERE'
markfound=$(grep "$mark" docs/index.adoc)
if [ -z "$markfound" ]; then
    echo "no docs/index.adoc with '$mark' found"
    exit 1
fi
file="$(slug "$*").adoc"
hasfile=$(grep 'include::./'"$file"'\[\]' docs/index.adoc)
if [ -z "$hasfile" ]; then
    perl -p -i -e "s,$mark,$mark\n\ninclude::./$file\[\]," docs/index.adoc
fi
if [ ! -e docs/"$file" ]; then
    printf "= %s\n\n%s\n\n\n" "$*" "$(inow)" >"docs/$file"
fi
exec "$EDITOR" "docs/$file"
```

That script was dynamically added during my `build` command just by including the following line in my `blog` repo at a reliable location relative to my `dot` repo (`include::../../dot/scripts/blog[]`).

I also can just swap out the verb `blog` from `save` using the following script on my command line and have it push to GitHub and post a message to my life IRC Twitch chat at the same time:

```
#!/usr/bin/env bash

in-repo() {
    git rev-parse --count HEAD >/dev/null 2>&1
    return $?
}

has-local-changes() {
```

```
    test -n "$(git status --porcelain)"
    return $?
}

gitsave() {
    git pull
    git add -A .
    local message="$*"
    message=${message,,}
    if test -n "$message"; then
        git commit -a -m "$message"
    elif test -e /tmp/commitmsg; then
        git commit -a -F /tmp/commitmsg
        mv /tmp/commitmsg "/tmp/commitmsg.$(date +%s)"
    else
        git commit -a -m "save"
    fi
    if type push; then
        push
    else
        git push
    fi
}

main() {
    if ! in-repo; then
        echo "Not in a repo."
        return 1
    fi
    if ! has-local-changes; then
        echo "Already at the latest."
        return 0
    fi
    gitsave "$@"
    return $?
}

main "$@"
```

EPUB and PDF as a bonus

The reason I don't use MkDocs anymore is because it has almost zero support for EPUB and PDF. AsciiDoc has all this covered and has done for more than two decades. Sometimes mature is better than the latest trends. This is one of those cases. Sure I give up a search box from MkDocs but it is rather poorly designed anyway and doesn't catch stuff that I can catch using regular expressions from a simple web browser. This is the reason I keep my stuff all in a single, long text file with no images to screw things up.

Databases are unnecessary

Raw document data is *better* than anything in a database—especially for blogging. Storing blog posts deep in some database is actually monumentally stupid and unnecessary in the age of AI. Just a simple, static web page will get automatically indexed in ways no database engine would ever begin to touch in usefulness.

The names of the files in PP approach are easy to read and grep making them the identifiers.

My kn tool was a complete waste of time even if there was no way of knowing that then. AI has changed *everything*.

Looking forward to more

I'm really looking forward to having more discoveries because of the new format. I feel rather stupid now allowing myself to dump so much time into developing my own system based on Zettelkasten when clearly it is simply stupid to use a Zettelkasten model for anything in the age of AI.

Changelog over PR body

Friday, January 3, 2025, 5:15:52PM EST

Just got burned by the following command that wiped out my meticulously created body of my GitHub pull request:

```
gh pr edit --body # DO NOT DO THIS
```

It blew it away without asking because it was expecting a string or something. In fact, pretty sure it was waiting on stdin. Suffice it to say, I was pissed, pissed enough to find a safer alternative. Imagine if I had done that someplace else? There is really no easy way to get it back, even though the entire GitHub offering is about recovery.

Solution

The solution is what I should have been doing all along: a changelog. [1]

I simply added `- [x]` to everything instead of just `-` so that I can add stuff and commit it as I go. The approach already takes into account that a team is going to be frequently updating the file as the central TODO list for the project.

By the way, Chloe told me about it. Took me about five minutes to find and get up to speed thanks to my AI assistant. Oh, and she also wrote this perfect IEEE bibliographic reference for that site in 10 seconds.

References

[1] O. Behrens, “Keep a Changelog,” keepachangelog.com. [Online]. Available: https://keepachangelog.com/en/1.0.0/. [Accessed: Jan. 3, 2025].

More Mormon insanity

Friday, January 3, 2025, 1:21:56PM EST

Thanks to the amazing couple over at https://www.youtube.com/@alyssadgrenfell I have both been triggered into a massive panic attack around Christmas and had a cathartic laugh at a near 90-year-old "Prophet" attempt to explain his face in a hat while "translating" the Book of Mormon.

Mormon cultists lie about translations of book of Mormon

When I read this in Rough Rolling Stone it sent me on my path of doubt/reality check that led me to leave the church. I now see what good company I was in. Tom Phillips, a life-long faithful member who was chosen to have his "calling and election made sure" did a full interview chronicling his story. It was heart-wrenching to read all 102 pages of the transcript PDF—especially the absolutely inexcusable, shameful response from Jeffery Holland, a man I have come to strongly and deeply despise for his regular referral to anyone *not* believing exactly as he does as unintelligent, evil, or just plain stupid.

All Tom did is ask honest, simple questions that were met with the same hostility that caused Joseph Smith to illegally destroy a printing press publishing *facts* about Joseph that the church to this day calls "lies".

Secret ceremony I never knew

I was pretty high up in the church, or so I thought. Turns out over 20,000 people since the time of Joseph Smith have had an "apostle" wash their feet and basically tell them they cannot do anything that would keep them out of heaven except murder someone. If I hadn't known about a single other thing I would have left the church just over that absolute bullshit.

Gross mutilation of own body in covenants

The temple ceremony where I promised to remove my bowels and innards and slit my throat rather than talk about the temple ceremony wasn't even the most gruesome. They used to really have to add horrible language. Thank GOD for the Internet. The penalties have been completely removed from the ceremony now. I guess God changes to meet the times.

I can only take so much

The more I keep learning about just how horribly the Mormon cult is, the more I wish I had left it without collapsing and falling in love with another woman who loved me for me and not because I was Mormon. Tom's story *proves* that you can be absolutely the most righteous Mormon on planet Earth and *still* have your wife leave you and think you are scum in Satan's power. The Mormon church is thankfully dying slowly but surely behind the scenes and it could not bring me more joy.

References

- https://www.mormonstories.org/tom-phillips-and-the-second-anointing/
- https://www.youtube.com/@alyssadgrenfell
- https://youtu.be/W_eSubCKmGo
- https://youtu.be/8tqLad2Jse4
- https://youtu.be/N3c33WKjYV0
- https://www.wsoctv.com/news/local/police-mooresville-man-busted-child-sex-sting/334803464/

Pumping up Perl

Friday, January 3, 2025, 12:48:44PM EST

> "Perl is, by and large, a digested and simplified version of Unix. Perl is the Cliff Notes of Unix. I could never really get myself to learn sed, awk, zsh, grep, find, m4, pipes, xargs, tee, cut, paste, yacc, lex, various IPC or even C for that matter. I ought to. In practice, in almost all cases I use perl." - Larry Wall

I keep coming back to Perl.[1] There is nothing better than Perl for text processing, period. It destroys all the things it targets—especially awk. People who argue that awk is universal have to accept that it cannot do most of the things people doing text processing want to do from their local workstations doing whatever.

Better to create a perl filter than a Vim plugin

Keeping everything in a Unix self-contained filter script—even if just one line—is *always* better than a Vim plugin that does nothing more than transform content. This way these things can be chained together. You just cannot do that with filters.

References

[1] R. S. Muhlestein, “Why are you still using Perl?,” rwxrob.github.io. [Online]. Available: https://rwxrob.github.io/faq#_why_are_you_still_using_perl. [Accessed: Jan. 3, 2025].

Banishing bash for scripting

Friday, January 3, 2025, 12:25:20PM EST

Bash has burned me one too many times in 2024—mostly its horrible regular expression syntax or just not being on a system that I regularly use.

It's not as ubiquitous as people say

I know, right? Here I have been saying bash is everywhere. It's not. Container images that I love notably do not have it installed at all (Alpine, BusyBox, Kali, Arch) plus I have been doing a lot more with BSD and, well, bash is just not there without installing it. This has made it problematic when I wanted to just copy over my bashrc file. Nope. Can't do it.

I'll never willingly use GPLv3 stuff for anything

Perhaps the single biggest reason I do not want to use it ever again for scripting is the license. I despise GPLv3 and the people who push it. Most of them have no idea why it is so monstrously horrible for the OpenSource ecosystem and goes *against* all the founding principles of FOSS by *forcing* hardware manufacturers to comply with their Menshevik-esque ideas of "freedom". You are free so long as you comply with their idea of what "freedom" actually means. It's downright unethical and definitely untenable. This is why Linus has actively campaigned against it and refused to move the Linux kernel to it.[1] I personally will not willingly use anything that is GPLv3 if I can help it, and bash is one of those things.

Whatever interactive shell is fine

I'll use whatever interactive shell is on the host system from now on and will be converting my bashrc to a universal shell only so that I can just rename it. Any shell that I want has all the stuff I need.

What about completion?

It's a pain point for sure. Only bash supports self-completion. Zsh has to be modified with a couple scripts to make completion work with `complete -C foo foo` syntax. To me it's worth it even if all the other failures of zsh are something I cannot avoid.

Other reasons

Summary of other reasons I'm too lazy to write about right now:

- POSIX shell works *everywhere*
- Not as flexible or ubiquitous as POSIX
- Safer than POSIX but not as safe as `perl -T`
- Bloated with completely unnecessary stuff (100x > perl)

- `-r-xr-xr-x 1 root wheel 1.2M Aug 4 06:31 /bin/bash`
- `-rwxr-xr-x 1 root wheel 134K Aug 4 06:31 /usr/bin/perl`
- Sloooow, really slow (one reason `dash` was invented)
- Only integer math support
- Dependence on other external tools for text processing
- Bashisms aren't really that great
- Extended regular expressions are absolutely grotesque
- Using `/bin/sh` *is* `/bin/bash` on some systems but POSIX limited

References

[1] Open Source Stack Exchange, "What exactly is Tivoization and why didn't Linus Torvalds like it in GPLv3?," Open Source Stack Exchange. [Online]. Available: https://opensource.stackexchange.com/questions/259/what-exactly-is-tivoization-and-why-didnt-linus-torvalds-like-it-in-gplv3. [Accessed: Jan. 3, 2025].

AI or die

Friday, January 3, 2025, 12:52:05PM EST

For 10 years I used some form of Zettelkasten or knowledge base instead of a blog but since the advent of AI and large language models (LLMs) writing anything that isn't narrative prose is just stupid.

An AI assistant can find answers faster than any search I could ever do before.

These things favor stories, articles, and blogs over random note-taking and are sourced from the Internet.

Very few people will realize why this matters. Most of mediocrity will continue to chase the pointless trends and play with idiotic wastes of time like Obsidian and other flashy note-taking systems rather than incorporating AI into their workflows to get *actual* knowledge work done. Not me. While they are making 40 minute videos about how to get Excalidraw to render from your highly specific note taking with a stupid Lua plugin I will use by time for other things:

- Ongoing learning and research in real-time, anytime
- Interactive language instruction in Russian and French
- Automatic bibliographic reference creation down to the page number
- Automated YouTube descriptions and chapter markers derived from transcripts
- Full bike trip planning *while moving on the bike*
- Early medical issue diagnosis and training plans
- Breaking writers block for anything I'd ever write
- Perfect grammar and punctuation evaluation of my writing
- Automatic code comment generation from the code itself
- Mundane code creation

All I can say to the many skeptics out there is *try it*. Get ChatGPT and begin using it for stuff for which you would use Google search. Then, increase that, have it make you a plan for the day. Ask it what to eat for the next meal based on your day and current fitness goals. Show it your favorite painting and ask it to comment on it and provide resources. This isn't the kind of thing you will believe until it happens to you.

Plans for 2025

Wednesday, January 1, 2025, 12:02:14PM EST

Here we are in a new year. I'm a sucker for setting goals and reviewing them after the fact. Sipping coffee and eating a waffle with my wife while she writes in her shiny new 2025 journal. Sam is lounging on the patio in the sun. Seems like 2025 is going to be a great year.

- YouTube update videos
- Finish book: *Terminal Velocity*
- Finish book: *Coding from the Get Go*
- Books now available in PDF and EPUB automatically
- Son's college graduation in Idaho
- Son's stay with us in apartment
- Reward readers over viewers
- IRC (Twitch chat) all day
- Customize and improve cloudbot
- What about discord?

Mode plans:

- SCREEN: Highest percentage of content
- BIKE: Daily cycling at sunset
- COWORKING: Rebuilding Kubernetes cluster
- COWORKING: Automating Vault secrets extraction from K8SAPPs
- COWORKING: Perl over bash for small scripts
- COWORKING: Perl to Go migrations
- COWORKING: Cleaning up dot files

Time budget

Eating	10
Hygene	7
Bike (BIKE)	14
Work (COWORKING)	32
Sleep	56
Cleaning (IRL)	7
Writing (SCREEN)	14
Homelab/Coding	8

Yoga/Meditation	4
Strength	2
Chill/Doris	14

Knocked my teeth out

Friday, August 11, 2023

I've been fighting for the last few years—but especially the last week—with a frustration and deep depression about the monstrous human condition: the lack of charity, of **real** love and lives filled with dedication to improving and lifting all of us; the obsession with riches and gain; the "I got mine" mentality and "look at me" focus; the end of informed dialog and debate; the race to destroy Mother Earth as fast as we can; the demonization of education and extreme, dogmatic fundamentalism; the cults, gangs, religions, and political parties that prey on sincere, simple people turning them into human carcass batteries powering their mechanized, stinking bowels to produce the mind-worm larvae these viral parasites require to paralyze and consume others becoming hosts whose guts eventually explode infecting all around them. If I think about it at all I get very depressed. It's terrifying. Most of humanity is absolute shit, operating at a base level barely above pond scum, and we all let it get this way.

Then Friday I literally got some sense knocked into my head (and four front teeth knocked out of it). God had a painful message for me. [Disclaimer: I don't really believe in some narcissistic white-bearded dick in the sky who promises multiple wives if I do what he says (although once upon a time I did). It's a metaphor.]

Thursday night I had an amazing ride around Jetton Park and met some amazing people along the way. I sat on the point watching the sunset with my Twitch friends just realizing the contrast from the day before (at Trump golf course). It was overwhelming. I felt like it was a turning point, but the Universe apparently had even more in store for me.

Friday night I covered the interactive art where we had our feet washed by the artist, then ate a meal where each of us had to feed the person across the table. We were bound and our primary hand made useless. I joked and laughed with Nicole, one of my wife Doris' new BFFLs. Nicole's an artist and a teacher and just as demented and fun as Doris. I felt very fortunate to be paired with her.

Of course, I streamed the whole thing.

I was on a huge high after the dinner at sunset and seeing my wife in her new position as Project Manager of the McColl Center having only been a renting local resident just two years ago. Now, in many ways, she's running the place (with others of course). She's found so much fulfillment and seeing it just makes me overwhelmed with joy (a cliche word, but apt).

I set out after that on the bike to discover whatever I could find in Charlotte. Cycling is about exploration (for me anyway). We (the chat and I even though I was the only on biking) biked through the different party zones following/stalking the drunk people doing different things.

Then I was inspired to go to The Common Market, my favorite place to hang out in Charlotte because of the amazing people who choose to hang there. I was hoping to find Kevin and Jo there, the bike messengers who took me in a few weeks earlier.

It was no loss. Drinking a PBR I met some of the most amazing videographers in the region. One was actually filming at the McColl earlier and recognized me (since we were capturing around each

other the whole time). Turns out this was the place he chose to crash after the other events as well, the "after party" as it were. I had such a great time talking to Kevin (same name as messenger) and Surf about their creative and life experiences. They were really into the streaming thing and I later asked Doris about it and, since she is Program Manager now, she offered that I do a workshop for all the live streaming I do and how to setup a rig to do it. The idea of approaching IRL streaming as a legit videographer just makes me very excited. I hadn't thought of IRL streaming as an art form, but it really is, it's a live form of video/photo journalism. Thinking of IRL streaming as art wasn't something that really hit me until that night. I left with even more appreciation for it and all the people I've been fortunate to meet and "interview" through it.

I only had three beers, but my stomach was pretty empty. So I grabbed a slice at Fuel Pizza where I got to capture people randomly dancing to the DJ tunes there. It was a great slice. I could see the place where Doris and I first starting doing Yoga together, the place where I was when I **knew** I would eventually marry her. I was filled with a high like few I could have hoped for. Then I started my way home.

Not a block from pizza place I found myself on a well-lit downhill. I had been turning my front light on and off all night and had not thought to turn it on in this part of town. A woman left-turned from a dead stop in the parking on the side of the street stopping in the middle of the road. I had no where to go. I almost made it around, but I didn't and ended up leaving my teeth in the street.

https://youtu.be/uI2NCjcDQCQ?si=oJ8yOpP6YqSCpqif

After reviewing the time stamps in the video between when I was collided and when I first sent a chat message, I'm pretty sure I got knocked out for about two minutes. I hit at 6:34:40 in the video and first message at 6:36:51 "missing my front teeth". I don't remember getting myself to the grass, only sitting there. So either I was completely incoherent, or the two women carried me out of the street.

The woman who I collided with who was in some sort of party dress I think and I think had blonde hair (I never got a good look) helped me to the side of the road where I sat on the grass and just remember blood spewing everywhere from my face. She was very polite and called an ambulance. Must have been freaked out for sure. I got enough blood on my pants that the paramedics cut them off to see if I hurt my leg anywhere (thankfully I didn't).

Another very kind woman, a brunette, bent over in front of me, crouched down, just looking into my eyes the whole time telling me I'd be okay. I blabbered something about my teeth missing and she walked up and found them returning them to me. Just before the firetruck and ambulance and cop car arrived she disappeared. She was just such a nice person. I must have been a very ghastly site. I mean, the photos I posted should probably have a warning or something because they look so awful. But this amazing young woman just stayed with me until help arrived. I get all emotional just thinking of her. It was her Friday night. There was no reason for her to stop and get involved, but she did. I still don't know where she came from. She might have been one of the people in the car. I'll never know her name, but I will **never** forget her. THAT is humanity, Rob. I had forgotten.

In fact, this entire night was God tripping me all over again, reminding me what a fucking dumb ass I can be. I can just imaging God up there, "How can I get through to this idiot that all people don't suck? Oh I know …" But that wasn't the end of it. There were still plenty more amazing people to come on my bloodied path.

Maria and Ben were my paramedics. Ben was so bubbling and amazing. You would think that picking up broken people all day would eventually wear you down, but not Ben, and not Maria. They were smiling the whole time helping me out. Keeping me smiling through my toothless, bloody face. These amazing people **choose** this life. Every single day they take part in a broken system to help people in need despite all the corporate greed and political fuck-up that is American medical system. Ben and Maria don't care. They know it's the best we have and they have committed their lives to living in the broken system even if means all they can do is what they can **in spite** of the system. Do they languish in anguish over the state of humanity? Do they do tik-tok and Insta all day? Maybe, but not when they were patching my old face up and cutting off my pants. Two more people I will never forget.

While waiting for my turn to be seen by the ER doc I realized I was still streaming, just not from my IRL rig. I checked my phone. And as if God or Doris' Pixie knocked it out of my bag, my backup battery for my phone fell out of my bag onto the floor. An orderly saw it and gave it to me just as I realized my phone was at 2% and about to die. I plugged it in and had my entire streaming community there worried and wishing me the best. I always say that my friends on Twitch **are** my friends, and boy did they seem like it then. I know I've never met some of them, and may never meet them, but my God, they are **real** friends to me. While the chills started hitting me from shock of staring at the ceiling wondering about everything I found them and they kept me laughing all the way through it. Seriously, I think it helped keep me from going deeper into shock. People sent me private messages with their personal contact information and everything. These people, as much as they love to troll me, really cared. I've always known that, but it was a great reminder.

While I was laying there with the neck brace on wondering if I'd broken my neck (which I didn't, thank God) I had the unfortunately opportunity to hear what the ER goes through on a Friday night: people vomiting all over; drunk/autistic dude wandering around almost busting his head open falling having the hospital cops lock him up, then watching him break out and them call an emergency to get all the staff to tackle the guy and put him back into the room; the elderly woman and her husband having breathing problems; and all the doctors and nurses and orderlies and cops trying to keep the place from devolving into whatever it would become otherwise. I gained such an appreciation for every one of those strangers, including those who were there for whatever reason. The drunk autistic guy was particularly hard for me to take in.

Doris arrived to watch all of that as well. We both were very aware that could happen to any or our sons as well given a different circumstance? Where were that guys parents? Friends? He had no one. I had Doris. Her worried German eyes never teared up. I got her to grin a few times. She packed me up a bag of clothes not even knowing where I was or if I would be staying in the hospital. She was obviously worried, but didn't show it. She's such a rock, for everyone. That's what everyone keeps telling her at the McColl. I'm the lucky bastard who gets to call her my wife. Doris and I have been going through some trouble recently, stupid shit really, and there's never been a moment where my stupidity was more fully realized. I could hear God again, "Hey dumb-ass, remember this woman I hand delivered to you in a way that you could never forget I exist? Yeah, treat her right, you fucking moron." (That's how God talks in my head.)

They took me away for CT scans. What a zen experience that was, all the pretty colors. I think I might have been drunk a little still which made it all the more intense. It was like I was in some scene from Charlie and the Chocolate Factory. I was mesmerized. I wanted more.

Another doc took a bunch of X-rays. Nothing broken. Amazing sense of humor on that guy.

Another male nurse cleaned up my gross beard getting as much blood out as possible. He was like "oh shit" when he uncovered my lip laceration (that I got 3 stitched for later).

Later the ER doc would come in. My wife called him a "TV doctor" (meaning she thought he was hot, and he was). Strangely, while making small conversation as he stitched up my face we ended up talking about the state of the human condition and the lack of dialog. We both got legitimately into it. He was clearly from a more conservative place. We even talked about America and Europe and everything. In fact, I wonder how long we were there, because we had a very stimulating conversation while he sewed up my bloody face. When he left he said, "thank you" in a way that was very clearly sincere. He was all excited about talking about that stuff, almost like he doesn't get to talk to people much about that stuff in his line of work. I would never have imagined such a cool conversation could have happened in those conditions, but it did. Again, humans rock.

It was Doris' night and I hate that I had to take it all over with this crap. She never once said anything about it, just happy I was okay. I joked about upping my life insurance payout and she said, "Well then I'd just have more reason to kill you." Yeah, our sense of humor is really fucking demented. We laughed enough to hurt my face and made it home watching the sunrise in the final few miles. We went in, Sam sniffed me a lot but never got worried, "What, no eggs tonight?" I had a shower, and collapsed. So did they.

Earlier that night at the pub I pulled up my phone to exchange Instagrams with Surf (the Videographer hired by Apple, etc.) and the time was 11:11. To numerologists this means "a new beginning". I never got into that shit, but my wife likes to play with it. I hate to say it, but I feel like all of this has been a very distinct starting point for something very new, a new Rob perhaps, but maybe more.

Every time I give into the urge to run my tongue over my missing teeth I'll remember all of these amazing people and **know** that humans are awesome and amazing, that **these** people working a late Friday-night shift, that few people ever see, that they that matter. You won't hear about them. They don't do social media much. They're too busy being awesome. But they are there and they matter. I can never forget that, again.

Copyright