Funny mistakes and The Toyota Way

W. Adam Koszek

Koszek ORG

wojciech@koszek.com

2022-10-05T17:47:30Z

This article has a strange mix of UNIX and methodology and mental problems of post's author, related with inspirations from the The Toyota Way. I read The Toyota Way, since I tried to find something related to couple of things I'm interested in: (1) how companies work (2) Japanese culture (3) computers. (3) I involved myself. The way I think about stuff in "The Toyota Way" is that I'll not only get to know how Toyota solves problems, but I'll bring experiences from the book to my problem solving, mental toolset and I'll transmute them to my daily job with UNIX systems.

It is very funny how sometimes I get trapped with simple stuff.

===== ~/.cshrc =====

alias commbak tar cjf ~/backup/comments-`date '+%Y%m%d-%s'`.tar.bz2 /home/wojciec/comments

(challenge)

Things often get overlooked, since, yes, we all do mistakes. I do too. But it's entertaining when a person who knows what this stuff is all about gets trapped in his own mistakes. Each time I see a mistake in a script or a piece of code, I try to:

1. solve it properly: you have enough time to prepare a correct patch/fix to your scripts which correctly solve the problem. By correctly I mean: they preserve abstractions and proper code's conventions (you use some conventions, don't you?) and basically, are consistent with whatever you have there.
2. solve it: you don't have enough time, you must generate Excel/HTML based report that SIMPLY MUST BE DONE BEFORE 3pm meeting and no matter how obscure, obfuscated and ugly method you use, it doesn't really matter, since the results simply must be present by 3pm. So basically think "GM production" line – no matter what, we keep process going (this

is not really true for my day job and we only start "flow" once we get it right and we produce only as much data as we're able to analyze, but not more).

If you haven't ever done (2), it means you work for an excellent company, and I'd love to know what company that is. You're not a real hack0r.

So anyway – 2nd point is only a necessity and I'm not too ashamed, since I think it's just a part of daily job. Once again: this is preferred choice only when you're pushed to finish stuff on time and you know upfront you won't make it.

if ($g_fn_hack) { # HACK!! # Filename generation routine creates a *base_name*.log of a # file and we register *base_name* to prevent us from # overwriting files accidentally. But since I didn't # encompass in advance that we'll be repeating experiments # with different parameters, we need to bring a bit of # randomness to filenames, to be sure all files' names are # unique $fn += sprintf "%.0f", rand() * 1000; }

So this is an example of necessarily the smartest thing, but since the surrounding code wasn't trivial, this is the fastest way I could solve the problem. Randomness almost always helps – if you need ugly and incredibly dark solution that, of course, will produce darker result, use randomness. Now: this is not as evil as you may think. I just produce files with known "base name" (before extension), but add some unique factor. So not too bad.

I trapped myself not on generation, but parsing this stuff. So instead of:

ls -la

giving me a decent, nicely sorted result, I have to use:

ls -lar

disaster to figure out which files got created first (1st experiment run) and later plot HTML report based on that.

So for this particular case (and 2 others, not mentioned here) I created a bug database case with "5 why's" – notes how to not let it happen next time (***Finding the root cause, instead of solving/preventing unwanted effect***) and I'll get back to this ASAP. I know, I know: according to The Toyota Way ***problems should be solved right away***. But ... oh well. I haven't yet mastered this. For now I can tell you it's pretty darn hard to convince everybody around to fix problems correctly in this very moment, so fixing is postponed sometimes. Especially if results must be available the very next day.

(Solving problems RIGHT AWAY isn't always possible in my case, since my "cell" – the box of responsibilities – isn't shared with anybody, so it often involves lots of other stuff and I have to prioritize work; solving existing problems isn't always PRIO.1 job – I know, I know. Against The Toyota Way, but still - I've just finished 1/2 of the book)

Lets get back to automation. I solve lots of stuff by automating stuff and proper error reporting. I use cron(1) extensively, but lots of stuff is still done semi-automatically. Getting the process (getting the "flow", in Toyota's nomenclature) is painful first, but later it works pretty flawlessly.

You don't run everything from cron. You suck man, you REALLY SUCK

Well...

Stuff which I still happen to prefer to run by hand is defined by "Mental willingness to watch important process happening interactively due to the strong responsibility and need of knowing that it **just worked fine**". It's not that I can stuff everything from cron(1), because I can. I don't do it, since I don't feel too comfortable.

Not being comfortable involves:

0. Removing stuff
1. Working as a different user (shared account) and typing commands which lie in the history, which you happen to type by accident etc..
2. Moving large batches of important files
3. Copying stuff from one place to another
4. Managing processes

(0) deserves a different section I think, thus it's (0) here.

(1) I try to stay away from. For me shared accounts are a bad idea. So I just try to work as "me", and later copy results of commands/processing. (2) and (3) somehow fall into the same bucket. For me this is a standard:

/bin/ls -1

Then everything follows:

/bin/ls -1 | xargs -n 1

Every step I check just in case. So normally you write a script, but since now I'm a smart a*s in the middle of a book "The Toyota Way":

I can say I'm getting "visual inspection" done on every step. And I polish process till it's right. It gives me more experience and knowledge what I can expect in the next stages too. So finally it becomes:

/bin/ls -1 | xargs -n 1 -i# echo mv # #.old

Which even after many intermediate steps isn't doing anything, but once you type it, you'll understand how helpful it is.

"Visual Control so no Problems are Hidden"

So I have a habit of making lots of aliases. Aliases tend to be completely dark and you **simply can't understand them** unless you're me:

alias alljobs `bjobs | grep -v JOB | cut -d " " -f 1 | xargs -n 1 echo CMD`

What does this useless command do?

Huh?

Well, not too much. It just prints a line for each LSF job scheduled to run at the certain time. Why to do it in a such a complex way? Well, if you manage relatively important processes, which JUST CAN'T CRASH, you must be certain to do exactly what you want.

So the way I work is that I create myself a intermediate commands that automate lots of boring stuff (like the above: printing all running jobs) and have them generate something, that can be trivially modified:

alljobs | sed 's/CMD/bstop/g'

and suddenly you have something to feed your copy&paste buffer with. Once you inspect the output of a command, you can feed your shell with it too:

alljobs | sed 's/CMD/bstop/g' | sh

Still with me? Asleep? Yet another quiz to wake you up:

sh script1.sh script2.sh

What does this do? Hint: it is against the logic.

Yet, I stick to aliases. For example, "cd" is the most aliased command by me. If I have to type:

cd /long/path/name/to/dir

twice, it almost always lands in my:

~/.aliases

and earns it's only line there. Once again, it can get messy, since if you do:

cdt

especially when you're projecting your session, people don't really know where are you and what are you trying to accomplish.

Something useful at the end of this boring post:

alias fp '/bin/ls -1 | xargs -n 1 readlink -f'

I really can't imagine working without shortcuts.

So methodology works pretty well: do automatic stuff first, get it right, but leave the very last decision to the "human operator" (you) when possible; inspect stuff visually; double check random lines of sequences of commands which you're

going to run; if multiple (>1000) lines are involved, redirect output to a file and edit it, cross check with your assumptions. The decide what to do.

Believe me, this methodology saved me couple of times, since some critical problems (e.g.: lack of disk space, lack of RAM, machine rebooting due to cable problem in an expensive RAID shelf your company is using) happen, but happen rarely, and due to the Murphy's Laws, if your script has a potential to breaking in the situation that ***CAN'T*** happen typically, it'll do it; especially if there's potential of causing massive destruction and losing data which have been generated for the last week.

Be careful!