

A tool to discover unintended variable shadowing in your bash code

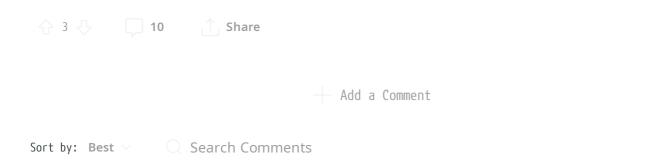
submission

Hey guys. I've been writing a complex script and encountered some problems passing variables by name as functions args. The problem was unintended variable shadowing.

Here is an example. Lets make a function with three args. It should sum \$2+\$3 and assign the result to the variable with name \$1. I know the code below is not optimal: it is that way to demonstrate the problem.

```
sum2 () {
local sum
```

Read more Y





Summary: don't use *sh for complex scripts, use python/perl/et al.

For complex scripts and hundreds of line scripts with a lot of functions is not recommended to use bash/sh/etc. You should use python/perl/other-real-script-lang. I don't want to demeaner bash/sh, but the scripting capabilities are more limited because thing like this. Also performance/memory usage and post execution shell stability could be a problem.

Love that you found a hack. But still is a hack that under different circumstances may fail.

Also not sure what are you writting but checkout things like ansible/puppet for alternatives to simplify much more the problem that you want to solve.

Maybe not the response that you wanna hear, but I think that probably is the one that you need to hear.



Skip to main content

Log In

You could laugh but I'm porting it from some of those to bash. I want it to work without any deps.

Also not sure what are you writting but checkout things like ansible/puppet for alternatives to simplify much more the problem that you want to solve.

It is a pure shell stuff, no external binaries used in 1k+ lines of code.





Limit variables scope to functions where ever possible, and declare constants as read only. You can even fake out block scope as follows:

```
declare foo='a'
echo "$foo"

_(){
    unset -f _
    declare foo='v'
    echo "$foo"

};__
echo "$foo"
```



Your sample code is reverse of what I try to accomplish. If I want to pass a var by its name to a function it is essential to NOT shadow it with a local var declared in the function. VARR detects shadowing.







Over my career I've had to write scripts that could be used in the latest version of bash, or it could be used in a shell that doesn't support local. The simple solution that I came up with for this scenario is to simply prepend vars that I want to be 'local' vars.

That is to say: I don't use the following approach all the time, just when I need to. It looks like this:

```
$SUM # This is an environment/global var
$sum # This is a script level var
$_sum # This is a local var
```

When I use this approach, I also make a point of unset ting any "local" vars at the end of a function.

Would a simple habitual-soft-scopes approach like this not work for your scenario? Am I misunderstanding the issue?



Skip to main content



Log In

But functions can call functions. Your _sum var in some function doesnt differ from a local var with the same name in a function it calls.

Am I misunderstanding the issue?

Let me provide another example. | count_zero | is a function to count number of zero elements in an array named \$2 and save the result to a var named \$1.

```
count_zero () {
    [[ $1 == result ]] || {
        local -n result
        result=$1
    [[ $2 == list ]] || {
        local -n list
        list=$2
    local el n # <=== 'n' local to 'count_zero'</pre>
    n=0
    for el in "${list[@]}"; do
       ! ((el == 0)) || ((++n))
    done
    result=$n
}
main () {
    local -a list=(1 0 2 3 0 4 5 6)
    local n # <=== 'n' local to 'main'</pre>
   count_zero n list
   declare -p n
}
main
```

Output:

```
declare -- n
```

If I use [m] instead of [n] in [main()], the outpus becomes

The problem is count_zero is not just what it does (set var named \$1 to number of zero els in array named \$2), but also which local var names it declares internally, which should not matter (since it is just a function-scope var declared in that function) but it Trying the should not matter unith assemply the world in the state of the three condenses are either nor a upvote el 1 result and list local vars are out of question since they are declared taking shadowing into account.

The relianding the your re written such function which works with variables passed by name long time Refactorately the department of the remover of zerougels in array named \$2. So you think number of els? Let me name the var n'. Run it and it doesn't work.

With VARR you can start count_zero with varr "\$1" "\$2" to prohibit local vars with r/swifines \$1, \$2 and it would catch the problem for you:

Suggestions for a Swift bosioner how to improve this gods

7 upvotes varr on 18: 'n' could be shadowed; call chain: main > count_zero

r/ProgrammingLanguages
In the example I've followed VARR rules for protected functions (local vars are declared Anybody which local retwritement a necessity of the restrictions of the state of the restriction of the local retwritement of the restriction of the local retwritement of the restriction of the local retwritement of the restriction of the rest

r/adventofcode

Is this solution understandable ? (Day 4 - Javascript)

3 upvotes ⋅ 8 comments

r/PowerBI

How to loop call a web API - Need Help

10 upvotes · 11 comments

r/golang

Custom error capable of wrapping another error?

4 upvotes · 3 comments

r/neovim

[Help needed] SegFault Error because of lua async? And how to detect if a buffer is a file.

3 upvotes · 4 comments

TOP POSTS



reReddit: Top posts of July 3, 2021





Log In

reReddit: Top posts of 2021