What is the most elegant way to remove a path from the $PATH variable in Bash?

Asked 16 years ago Modified 9 months ago Viewed 92k times

139

Or more generally, how do I remove an item from a colon-separated list in a Bash environment variable?

I thought I had seen a simple way to do this years ago, using the more advanced forms of Bash variable expansion, but if so I've lost track of it. A quick search of Google turned up surprisingly few relevant results and none that I would call "simple" or "elegant". For example, two methods using sed and awk, respectively:

```
PATH=$(echo $PATH | sed -e 's;:\?/home/user/bin;;' -e 's;/home/user/bin:\?;;')
PATH=!(awk -F: '{for(i=1;i<=NF;i++){if(!($i in a)){a[$i];printf
s$i;s=":"}}}'<<<$PATH)
```

Does nothing straightforward exist? Is there anything analogous to a split() function in Bash?

Update:
It looks like I need to apologize for my intentionally-vague question; I was less interested in solving a specific use-case than in provoking good discussion. Fortunately, I got it!

There are some very clever techniques here. In the end, I've added the following three functions to my toolbox. The magic happens in path_remove, which is based largely on Martin York's clever use of `awk`'s RS variable.

```
path_append () { path_remove $1; export PATH="$PATH:$1"; }
path_prepend () { path_remove $1; export PATH="$1:$PATH"; }
path_remove () { export PATH=`echo -n $PATH | awk -v RS=: -v ORS=: '$0 !=
"'$1'"' | sed 's/:$//'`; }
```

The only real cruft in there is the use of `sed` to remove the trailing colon. Considering how straightforward the rest of Martin's solution is, though, I'm quite willing to live with it!

Related question: How do I manipulate $PATH elements in shell scripts?

bash shell path variable-expansion list-processing

Share

Improve this question

Follow

edited Apr 24, 2019 at 16:16

cxw

17k ● 2 ● 48 ● 83

asked Dec 15, 2008 at 23:19

Ben Blank

56.5k ● 28 ● 131 ● 163

1 For any variable: `WORK=`echo -n ${1} | awk -v RS=: -v ORS=: '$0 != "'${3}'"' | sed 's/:$//'`; eval "export ${2}=${WORK}"` but you must call it as `func $VAR VAR pattern` (based on @martin-york and @andrew-aylett) – vesperto Nov 12, 2019 at 17:08

1 I stumbled on this question whilst looking for a way to update `PATH`, `LD_LIBRARY_PATH` etc., however, after a couple of happy hours bash scripting, it struck me, that we might all be better off using Environment Modules – Jonathan Watmough Mar 10, 2022 at 19:47

35 Answers

Sorted by: Highest score (default)

1 2 Next

70

My dirty hack:

```
echo ${PATH} > t1
vi t1
export PATH=$(cat t1)
```

Share

Improve this answer

Follow

edited Mar 9, 2016 at 16:30

answered Dec 16, 2008 at 0:05

Loki Astari

264k ● 86 ● 342 ● 571

28 It's never a good sign when the most obvious solution is to *de*-automate the process. :-D – Ben Blank Dec 17, 2008 at 17:21

2 SO much safer than the "elegant" alternatives. – Jortstek Mar 5, 2020 at 5:03

5 How did something as useless as this get so many upvotes? – Mehdi Oct 11, 2022 at 15:27

`:%s/:/\n/g` and `:%s/\n/:/` are useful here – JonnyRaa Sep 9 at 10:51

61

A minute with awk:

```
# Strip all paths with SDE in them.
#
export PATH=`echo ${PATH} | awk -v RS=: -v ORS=: '/SDE/ {next} {print}'`
```

Edit: It response to comments below:

```
$ export a="/a/b/c/d/e:/a/b/c/d/g/k/i:/a/b/c/d/f:/a/b/c/g:/a/b/c/d/g/i"
$ echo ${a}
/a/b/c/d/e:/a/b/c/d/f:/a/b/c/g:/a/b/c/d/g/i

## Remove multiple (any directory with a: all of them)
$ echo ${a} | awk -v RS=: -v ORS=: '/a/ {next} {print}'
## Works fine all removed

## Remove multiple including last two: (any directory with g)
$ echo ${a} | awk -v RS=: -v ORS=: '/g/ {next} {print}'
/a/b/c/d/e:/a/b/c/d/f:
## Works fine: Again!
```

Edit in response to security problem: (that is not relevant to the question)

```
export PATH=$(echo ${PATH} | awk -v RS=: -v ORS=: '/SDE/ {next} {print}' | sed 's/:*$//')
```

This removes any trailing colons left by deleting the last entries, which would effectively add `.` to your path.

Share

Improve this answer

Follow

edited Feb 24, 2016 at 6:41

Danica

28.8k ● 6 ● 93 ● 125

answered Dec 16, 2008 at 0:40

Loki Astari

264k ● 86 ● 342 ● 571

1 Fails when trying to remove the last element or multiple elements: in the first case, it adds the current dir (as the empty string; a potential security hole), in the second case it adds `` as a path element. – Fred Foo Mar 13, 2011 at 17:26

1 @larsmans: Works fine for me. Note: Empty is not the same as current directory which is "./" – Loki Astari Mar 13, 2011 at 18:01

3 An empty string as a "member" of the `PATH` variable *does*, as a special rule, denote current directory in all Unix shells since at least V7 Unix of 1979. It still does in `bash`. Check the manual or try for yourself. – Fred Foo Mar 13, 2011 at 22:58

2 @Martin: POSIX does not require this behavior, but does document and allow it: pubs.opengroup.org/onlinepubs/9699919799/basedefs/… – Fred Foo Mar 13, 2011 at 23:04

2 There's an even more subtle issue when removing the last element with this: awk seems to add a null byte to the end of the string. Meaning that if you append another directory to PATH later, it will in fact not be searched. – sschuberth Oct 24, 2012 at 20:36

53

Since the big issue with substitution is the end cases, how about making the end cases no different to the other cases? If the path already had colons at the start and end, we could simply search for our desired string wrapped with colons. As it is, we can easily add those colons and remove them afterwards.

```
# PATH => /bin:/opt/a dir/bin:/sbin
WORK=:$PATH:
# WORK => :/bin:/opt/a dir/bin:/sbin:
REMOVE='/opt/a dir/bin'
WORK=${WORK/:$REMOVE:/:}
# WORK => :/bin:/sbin:
WORK=${WORK%:}
WORK=${WORK#:}
PATH=$WORK
# PATH => /bin:/sbin
```

Pure bash :).

Share

Improve this answer

Follow

edited Dec 9, 2013 at 23:03

answered Jan 21, 2010 at 10:48

Andrew Aylett

40.6k ● 6 ● 70 ● 99

2 I'd add this tutorial section for some extra frosting: tldp.org/LDP/abs/html/string-manipulation.html – Cyber Oliveira Oct 8, 2011 at 18:16

1 I used this because it looked like the simplest solution. It was super fast and easy, and you can easily check your work with echo $WORK right before the last line where you actually change the PATH variable. – Phil Gran Aug 21, 2012 at 19:15

2 Absolutely a little gem. Exactly what I was trying to do when I found this post. -Thank you Andrew! BTW: Maybe you'd like to add double-quotes around ":$PATH:", just in case it should contain spaces (same about "/usr/bin") and last line "$WORK". – user1985657 Dec 9, 2013 at 15:33

3 Thanks, @PacMan-- :). I'm pretty sure (just tried it) you don't need spaces for the assignments to `WORK` and `PATH` as the variable expansion happens after the line is parsed into sections for variable assignment and command execution. `REMOVE` might need to be quoted, or you could just put your string straight into the replacement if it's a constant. – Andrew Aylett Dec 9, 2013 at 23:00

I always got confused over when strings were preserved, thus I started always to double-quote strings. Thanks again for clarifying this. :) – user1985657 Dec 11, 2013 at 19:12

27

Here's the simplest solution i can devise:

```
#!/bin/bash
IFS=:
# convert it to an array
t=($PATH)
unset IFS
# perform any array operations to remove elements from the array
t=(${t[@]%%*usr*})
IFS=:
# output the new array
echo "${t[*]}"
```

The above example will remove any element in $PATH that contains "usr". You can replace "*usr*" with "/home/user/bin" to remove just that element.

update per sschuberth

Even though i think spaces in a `$PATH` are a **horrible** idea, here's a solution that handles it:

```
PATH=$(IFS=':';t=($PATH);n=${#t[*]};a=();for ((i=0;i<n;i++)); do
p="${t[i]%%*usr*}"; [ "${p}" ] && a[i]="${p}"; done;echo "${a[*]}");
```

or

```
IFS=':'
t=($PATH)
n=${#t[*]}
a=()
for ((i=0;i<n;i++)); do
  p="${t[i]%%*usr*}"
  [ "${p}" ] && a[i]="${p}"
done
echo "${a[*]}"
```

Share

Improve this answer

Follow

edited May 23, 2017 at 12:18

Community Bot

1 • 1

answered Dec 16, 2008 at 1:27

nicerobot

9,235 • 6 • 44 • 44

2 As one liner: PATH=$(IFS=':';t=($PATH);unset IFS;t=(${t[@]%%*usr*});IFS=':';echo "${t[*]}"); – nicerobot Dec 16, 2008 at 4:10

1 This solution does not work with paths in PATH that contain spaces; it replaces them by colons. – sschuberth Oct 27, 2012 at 21:01

Will replace partial matches. Will incorrectly filter names with pattern metacharacters. – ivan_pozdeev Apr 29, 2022 at 6:28

16

Here's a one-liner that, despite the current accepted and highest rated answers, does not add invisible characters to PATH and can cope with paths that contain spaces:

```
export PATH=$(p=$(echo $PATH | tr ":" "\n" | grep -v "/cygwin/" | tr "\n" ":");
echo ${p%:})
```

Personally, I also find this easy to read / understand, and it only involves common commands instead of using awk.

Share

edited May 23, 2017 at 12:18

answered Oct 30, 2012 at 8:39

Improve this answer

Follow

Community Bot
1 • 1

sschuberth
29.8k • 8 • 105 • 151

3 ... and if you want something that can cope even with newlines in filenames, you could use this: `export PATH=$(p=$(echo $PATH | tr ":" "\0" | grep -v -z "/cygwin/" | tr "\0" ":"); echo ${p%:})` (though arguably, you might want to ask yourself why you need this, if you do :)) – Eric Hansander Jan 14, 2015 at 10:47

This will remove partial matches, which is probably not what you want; I would use `grep -v "^/path/to/remove\$"` or `grep -v -x "/path/to/remove"` – ShadSterling Jan 23, 2016 at 3:17

Fine solution, but do you really think `tr` is more common than `awk`? ;) – K.-Michael Aye May 24, 2016 at 23:52

1 Absolutely. Light-weight environments, like Git Bash on Windows, rather come with a simple tool like `tr` rather than an interpreter like `awk`. – sschuberth May 25, 2016 at 6:04

2 @ehdr: You need to replace `echo "..."` with `printf "%s" "..."` for it to work on paths like `-e` and similar. See stackoverflow.com/a/49418406/102441 – Eric Oct 8, 2018 at 14:16

12

Here is a solution that:

- is pure Bash,
- does not invoke other processes (like 'sed' or 'awk'),
- does not change `IFS`,
- does not fork a sub-shell,
- handles paths with spaces, and
- removes all occurrences of the argument in `PATH`.

```
removeFromPath() {
   local p d
   p=":$1:"
   d=":$PATH:"
   d=${d//$p/:}
   d=${d/#:/}
   PATH=${d/%:/}
}
```

Share Improve this answer Follow

answered Mar 20, 2015 at 4:22

robinbb
231 • 2 • 4

4 I like this solution. Maybe make the variable names more descriptive? – Anukool Jun 11, 2015 at 20:01

2 very nice. However, does not work for path segments containing an asterisk (*). Sometimes they get there accidentally. – Jörg Feb 14, 2019 at 18:00

Best / simplest answer IMO. Slightly shorter (not that it was too long however ;-)): `rmpath() { local d; d=:$PATH:; d=${d//:$1:/:}; d=${d#:}; PATH=${d%:}; }` – Fuujuhi Oct 26, 2021 at 8:52

1 Cannot delete two identical entries that come one after another. – ivan_pozdeev Apr 29, 2022 at 6:18

this solution is the best. – OfusJK May 26, 2022 at 8:51

7

The best pure bash option I have found so far is the following:

```
function path_remove {
  # Delete path by parts so we can never accidentally remove sub paths
  PATH=${PATH//":$1:"/":"} # delete any instances in the middle
  PATH=${PATH/#"$1:"/} # delete any instance at the beginning
  PATH=${PATH/%":$1"/} # delete any instance at the end
}
```

This is based on the not quite correct answer to Add directory to $PATH if it's not already there over on Superuser, fixing issues mentioned in comments.

Obviously this can be made into a single line function if you don't want the explanatory comments.

Share

Improve this answer

Follow

edited Apr 29, 2022 at 10:15

answered Oct 25, 2013 at 10:59

Mark Booth

7,894 ● 2 ● 76 ● 97

This is quite good too. I tested it. If there is a duplicate path (eg. two that are exactly the same) in PATH, then only one of them is removed. You can also make it into a one-liner: `removePath () { PATH=${PATH/":$1"/}; PATH=${PATH/"$1:"/}; }` – user1985657 Dec 11, 2013 at 19:26

2 This solution fails when the `$PATH` contains a sub-folder of the target (i.e. to be deleted) path. For example: `a:abc/def/bin:b` -> `a/bin:b`, when `abc/def` is to be deleted. – Robin Hsu Apr 14, 2017 at 4:06

6

I've just been using the functions in the bash distribution, that have been there apparently since 1991. These are still in the bash-docs package on Fedora, and used to be used in `/etc/profile`, but no more...

```
$ rpm -ql bash-doc |grep pathfunc
/usr/share/doc/bash-4.2.20/examples/functions/pathfuncs
$ cat $(!!)
```

```
cat $(rpm -ql bash-doc |grep pathfunc)
#From: "Simon J. Gerraty" <sjg@zen.void.oz.au>
#Message-Id: <199510091130.VAA01188@zen.void.oz.au>
#Subject: Re: a shell idea?
#Date: Mon, 09 Oct 1995 21:30:20 +1000


# NAME:
#       add_path.sh - add dir to path
#
# DESCRIPTION:
#       These functions originated in /etc/profile and ksh.kshrc, but
#       are more useful in a separate file.
#
# SEE ALSO:
#       /etc/profile
#
# AUTHOR:
#       Simon J. Gerraty <sjg@zen.void.oz.au>

#       @(#)Copyright (c) 1991 Simon J. Gerraty
#
#       This file is provided in the hope that it will
#       be of use.  There is absolutely NO WARRANTY.
#       Permission to copy, redistribute or otherwise
#       use this file is hereby granted provided that
#       the above copyright notice and this notice are
#       left intact.

# is $1 missing from $2 (or PATH) ?
no_path() {
        eval "case :\$${2-PATH}: in *:$1:*) return 1;; *) return 0;; esac"
}
# if $1 exists and is not in path, append it
add_path () {
  [ -d ${1:-.} ] && no_path $* && eval ${2:-PATH}="\$${2:-PATH}:$1"
}
# if $1 exists and is not in path, prepend it
pre_path () {
  [ -d ${1:-.} ] && no_path $* && eval ${2:-PATH}="$1:\$${2:-PATH}"
}
# if $1 is in path, remove it
del_path () {
  no_path $* || eval ${2:-PATH}=`eval echo :'$'${2:-PATH}: |
    sed -e "s;:$1:;:;g" -e "s;^:;;" -e "s;:\$;;"`
}
```

Share Improve this answer Follow

answered Mar 8, 2012 at 23:09

Mr. Wacky

83 ● 1 ● 4

6

```
function __path_remove(){
    local D=":${PATH}:";
    [ "${D/:$1:/:}" != "$D" ] && PATH="${D/:$1:/:}";
    PATH="${PATH/#:/}";
```

```
    export PATH="${PATH/%:/}";
}
```

Dug it out from my .bashrc file. When you play around with PATH, and it gets lost, awk/sed/grep becomes unavailable :-)

Share

Improve this answer

Follow

edited Oct 27, 2021 at 10:03

icedwater

4,877 ● 3 ● 38 ● 53

answered Aug 13, 2012 at 0:28

GreenFox

1,122 ● 10 ● 5

2 That's a very good point. (I never was fond of executing external utilities for simple things like this). – user1985657 Dec 11, 2013 at 19:33

5

Linux from Scratch defines three Bash functions in `/etc/profile`:

```
# Functions to help us manage paths.  Second argument is the name of the
# path variable to be modified (default: PATH)
pathremove () {
        local IFS=':'
        local NEWPATH
        local DIR
        local PATHVARIABLE=${2:-PATH}
        for DIR in ${!PATHVARIABLE} ; do
                if [ "$DIR" != "$1" ] ; then
                  NEWPATH=${NEWPATH:+$NEWPATH:}$DIR
                fi
        done
        export $PATHVARIABLE="$NEWPATH"
}

pathprepend () {
        pathremove $1 $2
        local PATHVARIABLE=${2:-PATH}
        export $PATHVARIABLE="$1${!PATHVARIABLE:+:${!PATHVARIABLE}}"
}

pathappend () {
        pathremove $1 $2
        local PATHVARIABLE=${2:-PATH}
        export $PATHVARIABLE="${!PATHVARIABLE:+${!PATHVARIABLE}:}$1"
}

export -f pathremove pathprepend pathappend
```

Ref: http://www.linuxfromscratch.org/blfs/view/svn/postlfs/profile.html

Share Improve this answer Follow

answered Nov 22, 2015 at 10:00

kevinarpe

21.3k ● 28 ● 132 ● 164

3

I did write an answer to this here (using awk too). But i'm not sure that's what you are looking for? It at least looks clear to me what it does, instead of trying to fit into one line. For a simple one liner, though, that only removes stuff, i recommend

```
echo $PATH | tr ':' '\n' | awk '$0 != "/bin"' | paste -sd:
```

Replacing is

```
echo $PATH | tr ':' '\n' |
    awk '$0 != "/bin"; $0 == "/bin" { print "/bar" }' | paste -sd:
```

or (shorter but less readable)

```
echo $PATH | tr ':' '\n' | awk '$0 == "/bin" { print "/bar"; next } 1' | paste
-sd:
```

Anyway, for the same question, and a whole lot of useful answers, see here.

Share

Improve this answer

Follow

edited May 23, 2017 at 12:34

Community Bot

1 • 1

answered Dec 16, 2008 at 0:13

Johannes Schaub - litb

506k • 131 • 917 • 1.2k

> And if you want to remove a lines which contains a partial string use `awk '$0 !~ "/bin"'`. I.e. keep lines that do not contain '/bin' with the awk operator `!~`. – thoni56 Feb 5, 2019 at 7:37

3

What is the most elegant way to remove a path from the $PATH variable in Bash?

What's more elegant than awk?

> ```
> path_remove ()  { export PATH=`echo -n $PATH | awk -v RS=: -v ORS=: '$0 !=
> "'$1'"' | sed 's/:$//'`;
> ```

Python! It's a more readable and maintainable solution, and it is easy to inspect to see that it's really doing what you want.

Say you want to remove the first path element?

```
PATH="$(echo "$PATH" | python -c "import sys; path =
sys.stdin.read().split(':'); del path[0]; print(':'.join(path))")"
```

(Instead of piping from `echo`, `os.getenv['PATH']` would be a little shorter, and provided the same result as the above, but I'm worried that Python might do something with that environment variable, so it's probably best to pipe it directly from the environment you care about.)

Similarly to remove from the end:

```
PATH="$(echo "$PATH" | python -c "import sys; path =
sys.stdin.read().split(':'); del path[-1]; print(':'.join(path))")"
```

To make these reusable shell functions that you can, for example, stick in your .bashrc file:

```
strip_path_first () {
    PATH="$(echo "$PATH" |
    python -c "import sys; path = sys.stdin.read().split(':'); del path[0];
print(':'.join(path))")"
}

strip_path_last () {
    PATH="$(echo "$PATH" |
    python -c "import sys; path = sys.stdin.read().split(':'); del path[-1];
print(':'.join(path))")"
}
```

Share
Improve this answer
Follow

edited Feb 15, 2017 at 22:42

answered Jul 10, 2016 at 13:08

Aaron Hall ♦
394k ● 91 ● 412 ● 339

2

Well, in bash, as it supports regular expression, I would simply do :

```
PATH=${PATH/:\/home\/user\/bin/}
```

Share Improve this answer Follow

answered Dec 15, 2008 at 23:23

mat
13.3k ● 5 ● 44 ● 45

1 Isn't it only pathname expansion, not regular expressions? – dreamlax Dec 16, 2008 at 0:07

3 While bash does support regular expressions (as of bash 3), this is not an example of it, this is a variable substitution. – Robert Gamble Dec 16, 2008 at 0:11

4 This is pattern variable expantion and the solution has several problems. 1) it will not match the first element. 2) it will match anything that starts with "/home/user/bin", not just "/home/user/bin". 3) it requires escaping special characters. At best, i'd say this is an incomplete example. – nicerobot Dec 16, 2008 at 0:27

2

I like the three functions shown in @BenBlank's update to his original question. To generalize them, I use a 2-argument form, which allows me to set PATH or any other environment variable I want:

```
path_append ()  { path_remove $1 $2; export $1="${!1}:$2"; }
path_prepend () { path_remove $1 $2; export $1="$2:${!1}"; }
path_remove ()  { export $1="`echo -n ${!1} | awk -v RS=: -v ORS=: '$1 !=
"'$2'"' | sed 's/:$//'`"; }
```

Examples of use:

```
path_prepend PATH /usr/local/bin
path_append PERL5LIB "$DEVELOPMENT_HOME/p5/src/perlmods"
```

Note that I also added some quotation marks to allow for the proper processing of pathnames that contain spaces.

Share Improve this answer Follow

answered Jan 5, 2016 at 20:54

Cary Millsap
822 ● 6 ● 17

1

What makes this problem annoying are the fencepost cases among first and last elements. The problem can be elegantly solved by changing IFS and using an array, but I don't know how to re-introduce the colon once the path is converted to array form.

Here is a slightly less elegant version that removes one directory from `$PATH` using string manipulation only. I have tested it.

```
#!/bin/bash
#
#   remove_from_path dirname
#
#   removes $1 from user's $PATH

if [ $# -ne 1 ]; then
  echo "Usage: $0 pathname" 1>&2; exit 1;
fi
```

```
delendum="$1"
NEWPATH=
xxx="$IFS"
IFS=":"
for i in $PATH ; do
  IFS="$xxx"
  case "$i" in
    "$delendum") ;; # do nothing
    *) [ -z "$NEWPATH" ] && NEWPATH="$i" || NEWPATH="$NEWPATH:$i" ;;
  esac
done

PATH="$NEWPATH"
echo "$PATH"
```

Share Improve this answer Follow

answered Dec 16, 2008 at 2:53

Norman Ramsey

202k ● 62 ● 371 ● 541

1

Yes, putting a colon at the end of PATH, for example, makes removing a path a bit less clumsy & error-prone.

```
path_remove ()  {
   declare i newPATH
   newPATH="${PATH}:"
   for ((i=1; i<=${#@}; i++ )); do
      #echo ${@:${i}:1}
      newPATH="${newPATH//${@:${i}:1}:/}"
   done
   export PATH="${newPATH%:}"
   return 0;
}

path_remove_all ()  {
   declare i newPATH
   shopt -s extglob
   newPATH="${PATH}:"
   for ((i=1; i<=${#@}; i++ )); do
      newPATH="${newPATH//+(${@:${i}:1})*([^:]):/}"
      #newPATH="${newPATH//+(${@:${i}:1})*([^:])+(:)/}"
   done
   shopt -u extglob
   export PATH="${newPATH%:}"
   return 0
}

path_remove /opt/local/bin /usr/local/bin

path_remove_all /opt/local /usr/local
```

Share Improve this answer Follow

answered Jan 21, 2010 at 19:58

cyrill

27 ● 1

1

If you are concerned about removing **duplicates** in $PATH, the most elegant way, IMHO, would be not to add them in the first place. In 1 line:

```
if ! $( echo "$PATH" | tr ":" "\n" | grep -qx "$folder" ) ; then
PATH=$PATH:$folder ; fi
```

$folder can be be replaced by anything, and may contain spaces ("/home/user/my documents")

Share Improve this answer Follow

answered Feb 19, 2011 at 5:11

MestreLion

13.6k ● 8 ● 70 ● 65

1

The most elegant pure bash solution I've found to date:

```
pathrm () {
  local IFS=':'
  local newpath
  local dir
  local pathvar=${2:-PATH}
  for dir in ${!pathvar} ; do
    if [ "$dir" != "$1" ] ; then
      newpath=${newpath:+$newpath:}$dir
    fi
  done
  export $pathvar="$newpath"
}

pathprepend () {
  pathrm $1 $2
  local pathvar=${2:-PATH}
  export $pathvar="$1${!pathvar:+:${!pathvar}}"
}

pathappend () {
  pathrm $1 $2
  local pathvar=${2:-PATH}
  export $pathvar="${!pathvar:+${!pathvar}:}$1"
}
```

Share Improve this answer Follow

answered Apr 7, 2012 at 5:29

TriangleTodd

49 ● 5

Most of the other suggested solutions rely only on string matching and don't take into account path segments containing special names like `.`, `..`, or `~`. The bash function below resolves directory strings in its argument and in path segments to find logical directory matches as well as string matches.

```
rm_from_path() {
  pattern="${1}"
  dir=''
  [ -d "${pattern}" ] && dir="$(cd ${pattern} && pwd)"  # resolve to absolute
path

  new_path=''
  IFS0=${IFS}
  IFS=':'
  for segment in ${PATH}; do
    if [[ ${segment} == ${pattern} ]]; then             # string match
      continue
    elif [[ -n ${dir} && -d ${segment} ]]; then
      segment="$(cd ${segment} && pwd)"                 # resolve to absolute
path
      if [[ ${segment} == ${dir} ]]; then               # logical directory
match
        continue
      fi
    fi
    new_path="${new_path}${IFS}${segment}"
  done
  new_path="${new_path/#${IFS}/}"                       # remove leading colon,
if any
  IFS=${IFS0}

  export PATH=${new_path}
}
```

Test:

```
$ mkdir -p ~/foo/bar/baz ~/foo/bar/bif ~/foo/boo/bang
$ PATH0=${PATH}
$ PATH=~/foo/bar/baz/.././../boo/././../bar:${PATH}  # add dir with special
names
$ rm_from_path ~/foo/boo/../bar/.  # remove same dir with different special
names
$ [ ${PATH} == ${PATH0} ] && echo 'PASS' || echo 'FAIL'
```

Share

Improve this answer

Follow

edited Oct 6, 2014 at 16:14

answered Sep 22, 2014 at 23:16

jwfearn

29.5k ● 28 ● 100 ● 123

plus one for outside the box – Loki Astari Aug 23, 2017 at 19:37

1

I know this question asks about BASH, which everyone should prefer, but since I enjoy symmetry and sometimes I'm required to use "csh", I built the equivalent to the "path_prepend()", "path_append()" and "path_remove()" elegant solution above.

The gist is that "csh" doesn't have functions, so I put little shell scripts in my personal bin directory that act like the functions. I create aliases to SOURCE those scripts to make the designated environment variable changes.

~/bin/_path_remove.csh:

```
set _resolve = `eval echo $2`
setenv $1 `eval echo -n \$$1 | awk -v RS=: -v ORS=: '$1 != "'${_resolve}'"' |
sed 's/:$//'`;
unset _resolve
```

~/bin/_path_append.csh:

```
source ~/bin/_path_remove.csh $1 $2
set _base = `eval echo \$$1`
set _resolve = `eval echo $2`
setenv $1 ${_base}:${_resolve}
unset _base _resolve
```

~/bin/_path_prepend.csh:

```
source ~/bin/_path_remove.csh $1 $2
set _base = `eval echo \$$1`
set _resolve = `eval echo $2`
setenv $1 ${_resolve}:${_base}
unset _base _resolve
```

~/bin/.cshrc:

```
…
alias path_remove  "source ~/bin/_path_remove.csh  '\!:1' '\!:2'"
alias path_append  "source ~/bin/_path_append.csh  '\!:1' '\!:2'"
alias path_prepend "source ~/bin/_path_prepend.csh '\!:1' '\!:2'"
…
```

You can use them like this...

```
%(csh)> path_append MODULEPATH ${HOME}/modulefiles
```

Share Improve this answer Follow

answered Jun 8, 2018 at 1:05

Lance E.T. Compte
1,168 ● 1 ● 14 ● 37

Bash built-in oneliner (doesn't remove repetition):

```
PATH=${PATH/${PATH/#$DIR:*/$DIR:}/}${PATH/${PATH/*:$DIR*/:$DIR}/}
```

1

Share Improve this answer Follow

answered Nov 23, 2021 at 19:50

Tulcas Anathar

11 ● 1

0

Since this tends to be quite problematic, as in there IS NO elegant way, I recommend avoiding the problem by rearranging the solution: build your PATH up rather than attempt to tear it down.

I could be more specific if I knew your real problem context. In the interim, I will use a software build as the context.

A common problem with software builds is that it breaks on some machines, ultimately due to how someone has configured their default shell (PATH and other environment variables). The elegant solution is to make your build scripts immune by fully specifying the shell environment. Code your build scripts to set the PATH and other environment variables based on assembling pieces that you control, such as the location of the compiler, libraries, tools, components, etc. Make each configurable item something that you can individually set, verify, and then use appropriately in your script.

For example, I have a Maven-based WebLogic-targeted Java build that I inherited at my new employer. The build script is notorious for being fragile, and another new employee and I spent three weeks (not full time, just here and there, but still many hours) getting it to work on our machines. An essential step was that I took control of the PATH so that I knew exactly which Java, which Maven, and which WebLogic was being invoked. I created environment variables to point to each of those tools, then I calculated the PATH based on those plus a few others. Similar techniques tamed the other configurable settings, until we finally created a reproducible build.

By the way, don't use Maven, Java is okay, and only buy WebLogic if you absolutely need its clustering (but otherwise no, and especially not its proprietary features).

Best wishes.

Share Improve this answer Follow

answered Dec 16, 2008 at 0:21

Rob Williams

7,921 ● 1 ● 37 ● 42

sometimes you don't have root access and your admin manages your `PATH`. Sure, you could build you own, but every time your admin moves something you have to figure out where he put it. That sort of defeats the purpose of having an admin. – Shep Apr 24, 2012 at 18:40

0

As with @litb, I contributed an answer to the question "How do I manipulate $PATH elements in shell scripts", so my main answer is there.

The 'split' functionality in `bash` and other Bourne shell derivatives is most neatly achieved with `$IFS`, the inter-field separator. For example, to set the positional arguments (`$1`, `$2`, ...) to the elements of PATH, use:

```
set -- $(IFS=":"; echo "$PATH")
```

It will work OK as long as there are no spaces in $PATH. Making it work for path elements containing spaces is a non-trivial exercise - left for the interested reader. It is probably simpler to deal with it using a scripting language such as Perl.

I also have a script, `clnpath`, which I use extensively for setting my PATH. I documented it in the answer to "How to keep from duplicating PATH variable in csh".

Share
Improve this answer
Follow

edited May 23, 2017 at 12:34
Community Bot
1 • 1

answered Dec 16, 2008 at 0:43
Jonathan Leffler
752k • 145 • 946 • 1.3k

IFS=: a=($PATH); IFS= splitting is also nice. works if they contain spaces too. but then you got an array, and have to fiddle with for loops and such to remove the names.
– Johannes Schaub - litb Dec 16, 2008 at 0:48

Yes; it gets fiddly - as with my updated comment, it is probably simpler to use a scripting language at this point. – Jonathan Leffler Dec 16, 2008 at 0:55

0

Here's a Perl one-liner:

```
PATH=`perl -e '$a=shift;$_=$ENV{PATH};s#:$a(:)|^$a:|:$a$#$1#;print'
/home/usr/bin`
```

The `$a` variable gets the path to be removed. The `s` (substitute) and `print` commands implicitly operate on the `$_` variable.

Share Improve this answer Follow

answered Dec 16, 2008 at 19:54

J. A. Faucett
410 ● 3 ● 7

Good stuff here. I use this one to keep from adding dupes in the first place.

```
#!/bin/bash
#
################################################################################
#
# Allows a list of additions to PATH with no dupes
#
# Patch code below into your $HOME/.bashrc file or where it
# will be seen at login.
#
# Can also be made executable and run as-is.
#
################################################################################

# add2path=($HOME/bin .)                  ## uncomment space separated list
if [ $add2path ]; then                    ## skip if list empty or commented
out
for nodup in ${add2path[*]}
do
    case $PATH in                   ## case block thanks to MIKE511
    $nodup:* | *:$nodup:* | *:$nodup ) ;;    ## if found, do nothing
    *) PATH=$PATH:$nodup            ## else, add it to end of PATH or
    esac                            ## *) PATH=$nodup:$PATH   prepend to front
done
export PATH
fi
## debug add2path
echo
echo " PATH == $PATH"
echo
```

Share

Improve this answer

Follow

edited Jan 13, 2010 at 9:28

answered Jan 13, 2010 at 1:35

ongoto
9 ● 2

1 You can simplify your case statement by adding a leading and trailing colon to the PATH string: `case ":$PATH:" in (*:"$nodup":*) ;; (*) PATH="$PATH:$nodup" ;; esac` – glenn jackman Jun 23, 2011 at 2:05

With extended globbing enabled it's possible to do the following:

```
# delete all /opt/local paths in PATH
shopt -s extglob
printf "%s\n" "${PATH}" | tr ':' '\n' | nl
printf "%s\n" "${PATH//+(\/opt\/local\/)+([^:])?(:)/}" | tr ':' '\n' | nl

man bash | less -p extglob
```

Share Improve this answer Follow

answered Jan 20, 2010 at 18:28

carlo
1

0

Extended globbing one-liner (well, sort of):

```
path_remove ()  { shopt -s extglob; PATH="${PATH//+(${1})+([^:])?(:)/}"; export
PATH="${PATH%:}"; shopt -u extglob; return 0; }
```

There seems no need to escape slashes in $1.

```
path_remove ()  { shopt -s extglob; declare escArg="${1//\//\\/}";
PATH="${PATH//+(${escArg})+([^:])?(:)/}"; export PATH="${PATH%:}"; shopt -u
extglob; return 0; }
```

Share Improve this answer Follow

answered Jan 21, 2010 at 10:14

carlo
1

0

Adding colons to PATH we could also do something like:

```
path_remove ()  {
   declare i newPATH
   # put a colon at the beginning & end AND double each colon in-between
   newPATH=":${PATH//:/::}:"
   for ((i=1; i<=${#@}; i++)); do
       #echo ${@:${i}:1}
       newPATH="${newPATH//:${@:${i}:1}:/}"   # s/:\/fullpath://g
   done
   newPATH="${newPATH//::/:}"
   newPATH="${newPATH#:}"      # remove leading colon
   newPATH="${newPATH%:}"      # remove trailing colon
   unset PATH
   PATH="${newPATH}"
   export PATH
   return 0
}


path_remove_all ()  {
   declare i newPATH extglobVar
   extglobVar=0
   # enable extended globbing if necessary
   [[ ! $(shopt -q extglob) ]]  && { shopt -s extglob; extglobVar=1; }
   newPATH=":${PATH}:"
   for ((i=1; i<=${#@}; i++ )); do
      newPATH="${newPATH//:+(${@:${i}:1})*([^:])/}"     # s/:\/path[^:]*//g
   done
   newPATH="${newPATH#:}"      # remove leading colon
```

```
    newPATH="${newPATH%:}"      # remove trailing colon
    # disable extended globbing if it was enabled in this function
    [[ $extglobVar -eq 1 ]] && shopt -u extglob
    unset PATH
    PATH="${newPATH}"
    export PATH
    return 0
}

path_remove /opt/local/bin /usr/local/bin

path_remove_all /opt/local /usr/local
```

Share Improve this answer Follow

answered Jan 22, 2010 at 17:45

proxxy

1

0

In path_remove_all (by proxxy):

```
-newPATH="${newPATH//:+(${@:${i}:1})*([^:])/}"
+newPATH="${newPATH//:${@:${i}:1}*([^:])/}"        # s/:\/path[^:]*//g
```

Share Improve this answer Follow

answered Jan 22, 2010 at 18:34

marius

1

0

While this is a very old thread, I thought this solution might be of interest:

```
PATH="/usr/lib/ccache:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
REMOVE="ccache" # whole or part of a path :)
export PATH=$(IFS=':';p=($PATH);unset IFS;p=(${p[@]%%$REMOVE});IFS=':';echo
"${p[*]}";unset IFS)
echo $PATH # outputs
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

found it on this blog post. I think I like this one most :)

Share Improve this answer Follow

answered Jun 22, 2011 at 21:18

mjc

442 ● 3 ● 7