How do I split a string on a delimiter in Bash?

Asked 14 years, 10 months ago Modified 9 days ago Viewed 3.6m times

2912

I have this string stored in a variable:

```
IN="bla@some.com;john@home.com"
```

Now I would like to split the strings by `;` delimiter so that I have:

```
ADDR1="bla@some.com"
ADDR2="john@home.com"
```

I don't necessarily need the `ADDR1` and `ADDR2` variables. If they are elements of an array that's even better.

After suggestions from the answers below, I ended up with the following which is what I was after:

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

IN="bla@some.com;john@home.com"

mails=$(echo $IN | tr ";" "\n")

for addr in $mails
do
    echo "> [$addr]"
done
```

Output:

```
> [bla@some.com]
> [john@home.com]
```

There was a solution involving setting [Internal_field_separator](#) (IFS) to `;`. I am not sure what happened with that answer, how do you reset `IFS` back to default?

RE: `IFS` solution, I tried this and it works, I keep the old `IFS` and then restore it:

```
IN="bla@some.com;john@home.com"

OIFS=$IFS
IFS=';'
mails2=$IN
for x in $mails2
do
    echo "> [$x]"
done
```

```
IFS=$OIFS
```

BTW, when I tried

```
mails2=($IN)
```

I only got the first string when printing it in loop, without brackets around `$IN` it works.

bash shell split scripting

Share Edit Follow

edited Oct 22, 2018 at 21:20
codeforester
41.1k 18 115 148

asked May 28, 2009 at 2:03
stefanB
78.5k 27 117 142

29 With regards to your "Edit2": You can simply "unset IFS" and it will return to the default state. There's no need to save and restore it explicitly unless you have some reason to expect that it's already been set to a non-default value. Moreover, if you're doing this inside a function (and, if you aren't, why not?), you can set IFS as a local variable and it will return to its previous value once you exit the function. – Brooks Moses May 1, 2012 at 1:26

30 @BrooksMoses: (a) +1 for using `local IFS=...` where possible; (b) -1 for `unset IFS`, this doesn't exactly reset IFS to its default value, though I believe an unset IFS behaves the same as the default value of IFS ($' \t\n'), however it seems bad practice to be assuming blindly that your code will never be invoked with IFS set to a custom value; (c) another idea is to invoke a subshell: `(IFS=$custom; ...)` when the subshell exits IFS will return to whatever it was originally. – dubiousjim May 31, 2012 at 5:21

13 `for x in $(IFS=';';echo $IN); do echo "> [$x]"; done` – user2037659 Apr 26, 2018 at 20:15

3 In order to save it as an array I had to place another set of parenthesis and change the `\n` for just a space. So the final line is `mails=($(echo $IN | tr ";" " "))`. So now I can check the elements of `mails` by using the array notation `mails[index]` or just iterating in a loop – afranques Jul 3, 2018 at 14:08

39 Answers

Sorted by: Highest score (default)

Не нашли ответ? Задайте вопрос на Stack Overflow на русском.

1 2 Next

1677

You can set the internal field separator (IFS) variable, and then let it parse into an array. When this happens in a command, then the assignment to `IFS` only takes place to that single command's environment (to `read`). It then parses the input according to the `IFS` variable value into an array, which we can then iterate over.

This example will parse one line of items separated by `;`, pushing it into an array:

```
IFS=';' read -ra ADDR <<< "$IN"
for i in "${ADDR[@]}"; do
  # process "$i"
done
```

This other example is for processing the whole content of `$IN`, each time one line of input separated by `;`:

```
while IFS=';' read -ra ADDR; do
  for i in "${ADDR[@]}"; do
    # process "$i"
  done
done <<< "$IN"
```

Share Edit Follow

edited Mar 9, 2021 at 17:42
robe007
3,763 4 35 61

answered May 28, 2009 at 2:23
Johannes Schaub - litb
501k 131 907 1.2k

35 This is probably the best way. How long will IFS persist in it's current value, can it mess up my code by being set when it shouldn't be, and how can I reset it when I'm done with it? – Chris Lutz May 28, 2009 at 2:25

16 now after the fix applied, only within the duration of the read command :) – Johannes Schaub - litb May 28, 2009 at 3:04

21 You can read everything at once without using a while loop: read -r -d '' -a addr <<< "$in" # The -d '' is key here, it tells read not to stop at the first newline (which is the default -d) but to continue until EOF or a NULL byte (which only occur in binary data). – lhunath May 28, 2009 at 6:14

89 @LucaBorrione Setting `IFS` on the same line as the `read` with no semicolon or other separator, as opposed to in a separate command, scopes it to that command -- so it's always "restored"; you don't need to do anything manually. – Charles Duffy Jul 6, 2013 at 14:39

5 @imagineerThis There is a bug involving herestrings and local changes to IFS that requires `$IN` to be quoted. The bug is fixed in `bash` 4.3. – chepner Oct 2, 2014 at 3:50

1601

Taken from *Bash shell script split array*:

```
IN="bla@some.com;john@home.com"
arrIN=(${IN//;/ })
echo ${arrIN[1]}                  # Output: john@home.com
```

Explanation:

This construction replaces all occurrences of `';'` (the initial `//` means global replace) in the string `IN` with `' '` (a single space), then interprets the space-delimited string as an array (that's what the surrounding parentheses do).

The syntax used inside of the curly braces to replace each `';'` character with a `' '` character is called Parameter Expansion.

There are some common gotchas:

1. If the original string has spaces, you will need to use IFS:

- `IFS=':'; arrIN=($IN); unset IFS;`

2. If the original string has spaces *and* the delimiter is a new line, you can set IFS with:

- `IFS=$'\n'; arrIN=($IN); unset IFS;`

Share Edit Follow

edited Dec 30, 2020 at 10:21
amo-ej1
3,297 27 35

answered Mar 10, 2011 at 9:00
palindrom
18.6k 1 22 39

121 I just want to add: this is the simplest of all, you can access array elements with ${arrIN[1]} (starting from zeros of course) – oz123 Mar 21, 2011 at 18:50

36 Found it: the technique of modifying a variable within a ${} is known as 'parameter expansion'. – KomodoDave Jan 5, 2012 at 15:13

32 No, I don't think this works when there are also spaces present... it's converting the ',' to ' ' and then building a space-separated array. – Ethan Apr 12, 2013 at 22:47

18 Very concise, but there are *caveats for general use*: the shell applies *word splitting* and *expansions* to the string, which may be undesired; just try it with. `IN="bla@some.com;john@home.com;*;broken apart"`. In short: this approach will break, if your tokens contain embedded spaces and/or chars. such as `*` that happen to make a token match filenames in the current folder. – mklement0 Apr 24, 2013 at 14:08

70 This is a bad approach for other reasons: For instance, if your string contains `;*;`, then the `*` will be expanded to a list of filenames in the current directory. -1 – Charles Duffy Jul 6, 2013 at 14:39

547

I've seen a couple of answers referencing the `cut` command, but they've all been deleted. It's a little odd that nobody has elaborated on that, because I think it's one of the more useful commands for doing this type of thing, especially for parsing delimited log files.

In the case of splitting this specific example into a bash script array, `tr` is probably more efficient, but `cut` can be used, and is more effective if you want to pull specific fields from the middle.

Example:

```
$ echo "bla@some.com;john@home.com" | cut -d ";" -f 1
bla@some.com
$ echo "bla@some.com;john@home.com" | cut -d ";" -f 2
john@home.com
```

You can obviously put that into a loop, and iterate the -f parameter to pull each field independently.

This gets more useful when you have a delimited log file with rows like this:

```
2015-04-27|12345|some action|an attribute|meta data
```

`cut` is very handy to be able to `cat` this file and select a particular field for further processing.

Share Edit Follow

edited Apr 28, 2015 at 22:17

answered Apr 27, 2015 at 18:20
DougW
29.5k 18 81 108

41 Kudos for using `cut`, it's the right tool for the job! Much cleared than any of those shell hacks. – MisterMiyagi Nov 2, 2016 at 8:42

11 This approach will only work if you know the number of elements in advance; you'd need to program some more logic around it. It also runs an external tool for every element. – uli42 Sep 14, 2017 at 8:30

1 Excatly waht i was looking for trying to avoid empty string in a csv. Now i can point the exact 'column' value as well. Work with IFS already used in a loop. Better than expected for my situation. – Louis Loudog Trottier May 10, 2018 at 4:20

7 This answer is worth scrolling down over half a page :) – Gucu112 Jan 3, 2020 at 17:26

1 @uli42 Have a look at my ***Why not `cut`*** paragraph in my answer!! Here is a little *while* loop, for processing any number of fields in whole line... (But it's not really **quick""!!) – F. Hauri - Give Up GitHub Aug 15, 2022 at 7:19

385

If you don't mind processing them immediately, I like to do this:

```
for i in $(echo $IN | tr ";" "\n")
do
  # process
done
```

You could use this kind of loop to initialize an array, but there's probably an easier way to do it.

Share Edit Follow

edited Jan 26, 2022 at 0:01
Halo
1,849 1 10 31

answered May 28, 2009 at 2:09
Chris Lutz
74.3k 16 131 183

4 You could change it to echo "$IN" | tr ';' '\n' | while read -r ADDY; do # process "$ADDY"; done to make him lucky, i think :) Note that this will fork, and you can't change outer variables from within the loop (that's why i used the <<< "$IN" syntax) then – Johannes Schaub - litb May 28, 2009 at 17:00

17 To summarize the debate in the comments: *Caveats for general use*: the shell applies *word splitting* and *expansions* to the string, which may be undesired; just try it with. `IN="bla@some.com;john@home.com;*;broken apart"`. In short: this approach will break, if your tokens contain embedded spaces and/or chars. such as `*` that happen to make a token match filenames in the current folder. – mklement0 Apr 24, 2013 at 14:13

362

Compatible answer

There are a lot of different ways to do this in bash .

However, it's important to first note that `bash` has many *special* features (so-called *bashisms*) that won't work in any other shell .

In particular, *arrays*, *associative arrays*, and *pattern substitution*, which are used in the solutions in this post as well as others in the thread, are *bashisms* and may not work under other *shells* that many people use.

For instance: on my *Debian GNU/Linux*, there is a *standard* shell called dash ; I know many people who like to use another shell called ksh ; and there is also a special tool called busybox with his own shell interpreter (ash).

For posix shell compatible answer, go to last part of this answer!

Requested string

The string to be split in the above question is:

```
IN="bla@some.com;john@home.com"
```

I will use a modified version of this string to ensure that my solution is robust to strings containing whitespace, which could break other solutions:

```
IN="bla@some.com;john@home.com;Full Name <fulnam@other.org>"
```

Split string based on delimiter in bash (version >=4.2)

In *pure* `bash` , we can create an *array* with elements split by a temporary value for *IFS* (the *input field separator*). The IFS, among other things, tells `bash` which character(s) it should treat as a delimiter between elements when defining an array:

```
IN="bla@some.com;john@home.com;Full Name <fulnam@other.org>"

# save original IFS value so we can restore it later
oIFS="$IFS"
IFS=";"
declare -a fields=($IN)
IFS="$oIFS"
unset oIFS
```

In newer versions of `bash` , prefixing a command with an IFS definition changes the IFS for that command *only* and resets it to the previous value immediately afterwards. This means we can do the above in just one line:

```
IFS=\; read -a fields <<<"$IN"
# after this command, the IFS resets back to its previous value (here, the default):
set | grep ^IFS=
# IFS=$' \t\n'
```

We can see that the string `IN` has been stored into an array named `fields`, split on the semicolons:

```
set | grep ^fields=\\\|^IN=
# fields=([0]="bla@some.com" [1]="john@home.com" [2]="Full Name <fulnam@other.org>")
# IN='bla@some.com;john@home.com;Full Name <fulnam@other.org>'
```

(We can also display the contents of these variables using `declare -p`:)

```
declare -p IN fields
# declare -- IN="bla@some.com;john@home.com;Full Name <fulnam@other.org>"
# declare -a fields=([0]="bla@some.com" [1]="john@home.com" [2]="Full Name
<fulnam@other.org>")
```

Note that `read` is the *quickest* way to do the split because there are no *forks* or external resources called.

Once the array is defined, you can use a simple loop to process each field (or, rather, each element in the array you've now defined):

```
# `"${fields[@]}"` expands to return every element of `fields` array as a separate
argument
for x in "${fields[@]}" ;do
    echo "> [$x]"
    done
# > [bla@some.com]
# > [john@home.com]
# > [Full Name <fulnam@other.org>]
```

Or you could drop each field from the array after processing using a *shifting* approach, which I like:

```
while [ "$fields" ] ;do
    echo "> [$fields]"
    # slice the array
    fields=("${fields[@]:1}")
    done
# > [bla@some.com]
# > [john@home.com]
# > [Full Name <fulnam@other.org>]
```

And if you just want a simple printout of the array, you don't even need to loop over it:

```
printf "> [%s]\n" "${fields[@]}"
# > [bla@some.com]
# > [john@home.com]
# > [Full Name <fulnam@other.org>]
```

Update: recent `bash` >= 4.4

In newer versions of `bash`, you can also play with the command `mapfile`:

```
mapfile -td \; fields < <(printf "%s\0" "$IN")
```

This syntax preserve special chars, newlines and empty fields!

If you don't want to include empty fields, you could do the following:

```
mapfile -td \; fields <<<"$IN"
fields=("${fields[@]%$'\n'}")   # drop '\n' added by '<<<'
```

With `mapfile`, you can also skip declaring an array and implicitly "loop" over the delimited elements, calling a function on each:

```
myPubliMail() {
    printf "Seq: %6d: Sending mail to '%s'..." $1 "$2"
    # mail -s "This is not a spam..." "$2" </path/to/body
    printf "\e[3D, done.\n"
}

mapfile < <(printf "%s\0" "$IN") -td \; -c 1 -C myPubliMail
```

(Note: the `\0` at end of the format string is useless if you don't care about empty fields at end of the string or they're not present.)

```
mapfile < <(echo -n "$IN") -td \; -c 1 -C myPubliMail

# Seq:      0: Sending mail to 'bla@some.com', done.
# Seq:      1: Sending mail to 'john@home.com', done.
# Seq:      2: Sending mail to 'Full Name <fulnam@other.org>', done.
```

Or you could use `<<<`, and in the function body include some processing to drop the newline it adds:

```
myPubliMail() {
    local seq=$1 dest="${2%$'\n'}"
    printf "Seq: %6d: Sending mail to '%s'..." $seq "$dest"
    # mail -s "This is not a spam..." "$dest" </path/to/body
    printf "\e[3D, done.\n"
}

mapfile <<<"$IN" -td \; -c 1 -C myPubliMail

# Renders the same output:
# Seq:      0: Sending mail to 'bla@some.com', done.
# Seq:      1: Sending mail to 'john@home.com', done.
# Seq:      2: Sending mail to 'Full Name <fulnam@other.org>', done.
```

Split string based on delimiter in shell

If you can't use `bash`, or if you want to write something that can be used in many different shells, you often **can't** use *bashisms* -- and this includes the arrays we've been using in the solutions above.

However, we don't need to use arrays to loop over "elements" of a string. There is a syntax used in many shells for deleting substrings of a string from the *first* or *last* occurrence of a pattern. Note that `*` is a wildcard that stands for zero or more characters:

(The lack of this approach in any solution posted so far is the main reason I'm writing this answer ;)

```
${var#*SubStr}  # drops substring from start of string up to first occurrence of
`SubStr`
${var##*SubStr} # drops substring from start of string up to last occurrence of
`SubStr`
${var%SubStr*}  # drops substring from last occurrence of `SubStr` to end of string
${var%%SubStr*} # drops substring from first occurrence of `SubStr` to end of string
```

As explained by Score_Under:

> `#` and `%` delete the shortest possible matching substring from the *start* and *end* of the string respectively, and
>
> `##` and `%%` delete the longest possible matching substring.

Using the above syntax, we can create an approach where we extract substring "elements" from the string by deleting the substrings up to or after the delimiter.

The codeblock below works well in bash (including Mac OS's `bash`), dash , ksh , lksh , yash , zsh , and busybox 's ash :

(Thanks to Adam Katz's comment, making this loop a lot simplier!)

```
IN="bla@some.com;john@home.com;Full Name <fulnam@other.org>"
while [ "$IN" != "$iter" ] ;do
    # extract the substring from start of string up to delimiter.
    iter=${IN%%;*}
    # delete this first "element" AND next separator, from $IN.
    IN="${IN#$iter;}"
    # Print (or doing anything with) the first "element".
    printf '> [%s]\n' "$iter"
done
# > [bla@some.com]
# > [john@home.com]
# > [Full Name <fulnam@other.org>]
```

Why not `cut` ?

`cut` is useful for extracting columns in big files, but doing *forks* repetitively (`var=$(echo ... | cut ...)`) become quickly overkill!

Here is a correct syntax, tested under many posix shell using `cut` , as suggested by This other answer from DougW:

```
IN="bla@some.com;john@home.com;Full Name <fulnam@other.org>"
i=1
while iter=$(echo "$IN"|cut -d\; -f$i) ; [ -n "$iter" ] ;do
    printf '> [%s]\n' "$iter"
    i=$((i+1))
done
```

I wrote this in order to compare **execution time**.

On my raspberrypi, this look like:

```
$ export TIMEFORMAT=$'(%U + %S) / \e[1m%R\e[0m : %P  '
$ time sh splitDemo.sh >/dev/null
(0.000 + 0.019) / 0.019 : 99.63
$ time sh splitDemo_cut.sh >/dev/null
(0.051 + 0.041) / 0.188 : 48.98
```

Where overall execution time is something like 10x longer, using **1** *forks* to `cut` , **by field**!

Share Edit Follow

edited Jul 28, 2023 at 22:41
T. Marwa
3 2

answered Apr 13, 2013 at 14:20
F. Hauri - Give Up GitHub
67.1k 18 124 142

32 The `#` , `##` , `%` , and `%%` substitutions have what is IMO an easier explanation to remember (for how much they delete): `#` and `%` delete the shortest possible matching string, and `##` and `%%` delete the longest possible. – Score_Under Apr 28, 2015 at 16:58

1 The `IFS=\; read -a fields <<<"$var"` fails on newlines and add a trailing newline. The other solution removes a trailing empty field. – user8017719 Oct 26, 2016 at 4:36

1 @moo There is a lot of script using `timeStampVar=$(date -d "$dateString" +%s)` repetitively, when `date -S` could be run in background...; And yes this seem not more *readable*, but once into a function, using `dateToEpoch -v varname "$dateString"` could be acceptable! Mostly if your script become improved from 1 or two seconds to near instant. – F. Hauri - Give Up GitHub Jan 31 at 16:33

178

This worked for me:

```
string="1;2"
echo $string | cut -d';' -f1 # output is 1
echo $string | cut -d';' -f2 # output is 2
```

Share Edit Follow

edited Jan 24, 2017 at 2:33
random-forest-cat
34.8k 12 122 102

answered Aug 11, 2016 at 20:45
Steven Lizarazo
5,414 2 29 25

3 Though it only works with a single character delimiter, that's what the OP was looking for (records delimited by a semicolon). – GuyPaddock Dec 12, 2018 at 1:37

1 Answered about four years ago by @Ashok, and also, more than one year ago by @DougW, than your answer, with even more information. Please post different solution than others'. – MAChitgarha Apr 3, 2020 at 9:41

158

I think AWK is the best and efficient command to resolve your problem. AWK is included by default in almost every Linux distribution.

```
echo "bla@some.com;john@home.com" | awk -F';' '{print $1,$2}'
```

will give

```
bla@some.com john@home.com
```

Of course your can store each email address by redefining the awk print field.

Share Edit Follow

edited Jul 3, 2019 at 13:15
noamtm
12.8k 16 72 112

answered Jan 14, 2013 at 6:33
Tong
2,107 2 15 20

13 Or even simpler: echo "bla@some.com;john@home.com" | awk 'BEGIN{RS=";"} {print}' – Jaro Jan 7, 2014 at 21:30

102

How about this approach:

```
IN="bla@some.com;john@home.com"
set -- "$IN"
IFS=";"; declare -a Array=($*)
echo "${Array[@]}"
echo "${Array[0]}"
echo "${Array[1]}"
```

Source

Share Edit Follow

edited Jul 20, 2011 at 16:21
BLeB
1,716 17 24

answered May 28, 2009 at 10:31
errator

8 +1 ... but I wouldn't name the variable "Array" ... pet peev I guess. Good solution. – Yzmir Ramirez Sep 5, 2011 at 1:06

14 +1 ... but the "set" and declare -a are unnecessary. You could as well have used just `IFS";"` && `Array=($IN)` – ata Nov 3, 2011 at 22:33

6 -1: First, @ata is right that most of the commands in this do nothing. Second, it uses word-splitting to form the array, and doesn't do anything to inhibit glob-expansion when doing so (so if you have glob characters in any of the array elements, those elements are replaced with matching filenames). – Charles Duffy Jul 6, 2013 at 14:44

1 Suggest to use `$'...'` : `IN=$'bla@some.com;john@home.com;bet <d@\ns* kl.com>'` . Then `echo "${Array[2]}"` will print a string with newline. `set -- "$IN"` is also neccessary in this case. Yes, to prevent glob expansion, the solution should include `set -f` . – John_West Jan 8, 2016 at 12:29

93

```
echo "bla@some.com;john@home.com" | sed -e 's/;/\n/g'
bla@some.com
john@home.com
```

Share Edit Follow

answered May 28, 2009 at 2:12
lothar
20k 6 47 60

4 -1 **what if the string contains spaces?** for example `IN="this is first line; this is second line" arrIN=( $( echo "$IN" | sed -e 's/;/\n/g' ) )` will produce an array of 8 elements in this case (an element for each word space separated), rather than 2 (an element for each line semi colon separated) – Luca Borrione Sep 3, 2012 at 10:08

5 @Luca No the sed script creates exactly two lines. What creates the multiple entries for you is when you put it into a bash array (which splits on white space by default) – lothar Sep 3, 2012 at 17:33

3 @Luca Good point. However the array assignment was not in the initial question when I wrote up that answer. – lothar Sep 4, 2012 at 16:55

73

This also works:

```
IN="bla@some.com;john@home.com"
echo ADD1=`echo $IN | cut -d \; -f 1`
echo ADD2=`echo $IN | cut -d \; -f 2`
```

Be careful, this solution is not always correct. In case you pass "bla@some.com" only, it will assign it to both ADD1 and ADD2.

Share Edit Follow

edited Apr 17, 2014 at 1:39
Boris S.
3 2

answered Sep 8, 2012 at 5:01
Ashok
749 5 3

1 You can use -s to avoid the mentioned problem: superuser.com/questions/896800/... "-f, --fields=LIST select only these fields; also print any line that contains no delimiter character, unless the -s option is specified" – fersarr Mar 3, 2016 at 17:17

44

A different take on Darron's answer, this is how I do it:

```
IN="bla@some.com;john@home.com"
read ADDR1 ADDR2 <<<$(IFS=";"; echo $IN)
```

Share Edit Follow

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

answered Jul 5, 2011 at 13:41
nickjb
1,186 1 12 16

1 This worked REALLY well for me... I used it to itterate over an array of strings which contained comma separated DB,SERVER,PORT data to use mysqldump. – Nick Oct 28, 2011 at 14:36

5 Diagnosis: the `IFS=";"` assignment exists only in the `$(...; echo $IN)` subshell; this is why some readers (including me) initially think it won't work. I assumed that all of $IN was getting slurped up by ADDR1. But nickjb is correct; it does work. The reason is that `echo $IN` command parses its arguments using the current value of $IFS, but then echoes them to stdout using a space delimiter, regardless of the setting of $IFS. So the net effect is as though one had called `read ADDR1 ADDR2 <<< "bla@some.com john@home.com"` (note the input is space-separated not ;-separated). – dubiousjim May 31, 2012 at 5:28

1 This fails on spaces and newlines, and also expand wildcards `*` in the `echo $IN` with an unquoted variable expansion. – user8017719 Oct 26, 2016 at 4:43

41

How about this one liner, if you're not using arrays:

```
IFS=';' read ADDR1 ADDR2 <<<$IN
```

Share Edit Follow

answered Sep 13, 2010 at 20:10
Darron
21.5k 5 50 54

1 Consider using `read -r ...` to ensure that, for example, the two characters "\t" in the input end up as the same two characters in your variables (instead of a single tab char). – dubiousjim May 31, 2012 at 5:36

1 This is probably due to a bug involving `IFS` and here strings that was fixed in `bash` 4.3. Quoting `$IN` should fix it. (In theory, `$IN` is not subject to word splitting or globbing after it expands, meaning the quotes should be unnecessary. Even in 4.3, though, there's at least one bug remaining--reported and scheduled to be fixed--so quoting remains a good idea.) – chepner Sep 19, 2015 at 13:59

40

In Bash, a bullet proof way, that will work even if your variable contains newlines:

```
IFS=';' read -d '' -ra array < <(printf '%s;\0' "$in")
```

Look:

```
$ in=$'one;two three;*;there is\na newline\nin this field'
$ IFS=';' read -d '' -ra array < <(printf '%s;\0' "$in")
$ declare -p array
declare -a array='([0]="one" [1]="two three" [2]="*" [3]="there is
a newline
in this field")'
```

The trick for this to work is to use the `-d` option of `read` (delimiter) with an empty delimiter, so that `read` is forced to read everything it's fed. And we feed `read` with exactly the content of the variable `in`, with no trailing newline thanks to `printf`. Note that's we're also putting the delimiter in `printf` to ensure that the string passed to `read` has a trailing delimiter. Without it, `read` would trim potential trailing empty fields:

```
$ in='one;two;three;'    # there's an empty field
$ IFS=';' read -d '' -ra array < <(printf '%s;\0' "$in")
$ declare -p array
declare -a array='([0]="one" [1]="two" [2]="three" [3]="")'
```

the trailing empty field is preserved.

Update for Bash≥4.4

Since Bash 4.4, the builtin `mapfile` (aka `readarray`) supports the `-d` option to specify a delimiter. Hence another canonical way is:

```
mapfile -d ';' -t array < <(printf '%s;' "$in")
```

Share Edit Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 1

answered Jun 26, 2014 at 9:11
gniourf_gniourf
45.6k 9 99 108

5 I found it as the rare solution on that list that works correctly with `\n`, spaces and `*` simultaneously. Also, no loops; array variable is accessible in the shell after execution (contrary to the highest upvoted answer). Note, `in=$'...'`, it does not work with double quotes. I think, it needs more upvotes. – John_West Jan 8, 2016 at 12:10

34

Without setting the IFS

If you just have one colon you can do that:

```
a="foo:bar"
b=${a%:*}
c=${a##*:}
```

you will get:

```
b = foo
c = bar
```

Share Edit Follow

answered Aug 1, 2016 at 13:15
Emilien Brigand
10.5k 9 34 37

24

Here is a clean 3-liner:

```
in="foo@bar;bizz@buzz;fizz@buzz;buzz@woof"
IFS=';' list=($in)
for item in "${list[@]}"; do echo $item; done
```

where `IFS` delimit words based on the separator and `()` is used to create an [array](). Then `[@]` is used to return each item as a separate word.

If you've any code after that, you also need to restore `$IFS`, e.g. `unset IFS`.

Share Edit Follow

edited Oct 26, 2016 at 10:26

answered Sep 11, 2015 at 20:54
kenorb
161k 91 692 758

6 The use of `$in` unquoted allows wildcards to be expanded. – user8017719 Oct 26, 2016 at 5:03

15

The following Bash/zsh function splits its first argument on the delimiter given by the second argument:

```
split() {
    local string="$1"
    local delimiter="$2"
    if [ -n "$string" ]; then
        local part
        while read -d "$delimiter" part; do
            echo $part
        done <<< "$string"
        echo $part
    fi
}
```

For instance, the command

```
$ split 'a;b;c' ';'
```

yields

```
a
b
c
```

This output may, for instance, be piped to other commands. Example:

```
$ split 'a;b;c' ';' | cat -n
1   a
2   b
3   c
```

Compared to the other solutions given, this one has the following advantages:

- `IFS` is not overriden: Due to dynamic scoping of even local variables, overriding `IFS` over a loop causes the new value to leak into function calls performed from within the loop.
- Arrays are not used: Reading a string into an array using `read` requires the flag `-a` in Bash and `-A` in zsh.

If desired, the function may be put into a script as follows:

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

split() {
    # ...
}

split "$@"
```

Share Edit Follow

edited Jun 13, 2017 at 18:24

answered May 24, 2017 at 8:42
bisgardo
4,400 4 31 44

13

So many answers and so many complexities. Try out a simpler solution:

```
echo "string1, string2" | tr , "\n"
```

`tr` (read, translate) replaces the first argument with the second argument in the input.

So `tr` , "\n" replace the comma with new line character in the input and it becomes:

```
string1
string2
```

Share Edit Follow edited Nov 11, 2022 at 4:55 answered Nov 4, 2022 at 7:22
sxddhxrthx
677 7 13

3 This prints both tokens but does not place them in variables. (And therefore doesn't fully answer the question. You might be able to tweak your answer to make it work.) – Jay Sullivan Jan 3, 2023 at 4:19

12

There is a simple and smart way like this:

```
echo "add:sfff" | xargs -d: -i  echo {}
```

But you must use gnu xargs, BSD xargs cant support -d delim. If you use apple mac like me. You can install gnu xargs :

```
brew install findutils
```

then

```
echo "add:sfff" | gxargs -d: -i  echo {}
```

Share Edit Follow answered Sep 16, 2015 at 3:34
Victor Choy
4,188 28 37

12

you can apply awk to many situations

```
echo "bla@some.com;john@home.com"|awk -F';' '{printf "%s\n%s\n", $1, $2}'
```

also you can use this

```
echo "bla@some.com;john@home.com"|awk -F';' '{print $1,$2}' OFS="\n"
```

Share Edit Follow edited Jan 21, 2018 at 11:34 answered Jan 20, 2018 at 15:54
reeeeeeeeeeeeeeeeeeeeeeeeeegie
488 6 7

10

There are some cool answers here (errator esp.), but for something analogous to split in other languages -- which is what I took the original question to mean -- I settled on this:

```
IN="bla@some.com;john@home.com"
```

```
declare -a a="(${IN//;/ })";
```

Now `${a[0]}`, `${a[1]}`, etc, are as you would expect. Use `${#a[*]}` for number of terms. Or to iterate, of course:

```
for i in ${a[*]}; do echo $i; done
```

IMPORTANT NOTE:

This works in cases where there are no spaces to worry about, which solved my problem, but may not solve yours. Go with the `$IFS` solution(s) in that case.

Share Edit Follow

edited Feb 13, 2023 at 15:50
guenhter
11.7k 4 37 69

answered Oct 22, 2012 at 7:10
eukras
879 7 5

5

If no space, Why not this?

```
IN="bla@some.com;john@home.com"
arr=(`echo $IN | tr ';' ' '`)

echo ${arr[0]}
echo ${arr[1]}
```

Share Edit Follow

answered Apr 24, 2013 at 13:13
ghost
458 6 11

4

This is the simplest way to do it.

```
spo='one;two;three'
OIFS=$IFS
IFS=';'
spo_array=($spo)
IFS=$OIFS
echo ${spo_array[*]}
```

Share Edit Follow

edited Feb 28, 2012 at 8:18

answered Sep 25, 2011 at 1:09
Tegra Detra
25.2k 17 54 78

4

Simple answer:

```
IN="bla@some.com;john@home.com"

IFS=';' read ADDR1 ADDR2 <<< "${IN}"
```

Sample output:

```
echo "${ADDR1}"  # prints "bla@some.com"
echo "${ADDR2}"  # prints "john@home.com"
```

Share Edit Follow

answered Jan 3, 2023 at 4:32
Jay Sullivan
17.9k 11 62 87

3

Apart from the fantastic answers that were already provided, if it is just a matter of printing out the data you may consider using `awk`:

```
awk -F";" '{for (i=1;i<=NF;i++) printf("> [%s]\n", $i)}' <<< "$IN"
```

This sets the field separator to `;`, so that it can loop through the fields with a `for` loop and print accordingly.

Test

```
$ IN="bla@some.com;john@home.com"
$ awk -F";" '{for (i=1;i<=NF;i++) printf("> [%s]\n", $i)}' <<< "$IN"
> [bla@some.com]
> [john@home.com]
```

With another input:

```
$ awk -F";" '{for (i=1;i<=NF;i++) printf("> [%s]\n", $i)}' <<< "a;b;c   d;e_;f"
> [a]
> [b]
> [c   d]
> [e_]
> [f]
```

Share Edit Follow

answered Jan 8, 2015 at 10:21
fedorqui
282k 105 569 610

3

```
IN="bla@some.com;john@home.com"
IFS=';'
read -a IN_arr <<< "${IN}"
for entry in "${IN_arr[@]}"
do
    echo $entry
done
```

Output

```
bla@some.com
john@home.com
```

System : Ubuntu 12.04.1

Share Edit Follow

edited Oct 25, 2016 at 12:55

answered Oct 25, 2016 at 12:41
rashok
13.1k 17 90 101

2

Use the `set` built-in to load up the `$@` array:

```
IN="bla@some.com;john@home.com"
IFS=';'; set $IN; IFS=$' \t\n'
```

Then, let the party begin:

```
echo $#
for a; do echo $a; done
ADDR1=$1 ADDR2=$2
```

Share Edit Follow

answered Apr 30, 2013 at 3:10
jeberle
738 3 15

2

Two bourne-ish alternatives where neither require bash arrays:

Case 1: Keep it nice and simple: Use a NewLine as the Record-Separator... eg.

```
IN="bla@some.com
john@home.com"

while read i; do
  # process "$i" ... eg.
    echo "[email:$i]"
done <<< "$IN"
```

Note: in this first case no sub-process is forked to assist with list manipulation.

Idea: Maybe it is worth using NL extensively *internally*, and only converting to a different RS when generating the final result *externally*.

Case 2: Using a ";" as a record separator... eg.

```
NL="
" IRS=";" ORS=";"

conv_IRS() {
  exec tr "$1" "$NL"
}
```

```
conv_ORS() {
  exec tr "$NL" "$1"
}

IN="bla@some.com;john@home.com"
IN="$(conv_IRS ";" <<< "$IN")"

while read i; do
  # process "$i" ... eg.
    echo -n "[email:$i]$ORS"
done <<< "$IN"
```

In both cases a sub-list can be composed within the loop is persistent after the loop has completed. This is useful when manipulating lists in memory, instead storing lists in files. {p.s. keep calm and carry on B-) }

Share Edit Follow

edited Sep 2, 2013 at 6:45

answered Sep 2, 2013 at 6:30
NevilleDNZ
1,279 13 32

2

In Android shell, most of the proposed methods just do not work:

```
$ IFS=':' read -ra ADDR <<<"$PATH"
/system/bin/sh: can't create temporary file /sqlite_stmt_journals/mksh.EbNoR10629: No
such file or directory
```

What does work is:

```
$ for i in ${PATH//:/ }; do echo $i; done
/sbin
/vendor/bin
/system/sbin
/system/bin
/system/xbin
```

where `//` means global replacement.

Share Edit Follow

edited Apr 19, 2015 at 22:27
Peter Mortensen
31.1k 22 109 132

answered Feb 20, 2015 at 10:49
18446744073709551615
16.6k 4 97 130

1 Fails if any part of $PATH contains spaces (or newlines). Also expands wildcards (asterisk *, question mark ? and braces [...]). – user8017719 Oct 26, 2016 at 5:08

2

```
IN='bla@some.com;john@home.com;Charlie Brown <cbrown@acme.com;!"#$%&/()[]{}*? are no
problem;simple is beautiful :-)'
set -f
oldifs="$IFS"
IFS=';'; arrayIN=($IN)
```

```
IFS="$oldifs"
for i in "${arrayIN[@]}"; do
echo "$i"
done
set +f
```

Output:

```
bla@some.com
john@home.com
Charlie Brown <cbrown@acme.com
!"#$%&/()[]{}*? are no problem
simple is beautiful :-)
```

Explanation: Simple assignment using parenthesis () converts semicolon separated list into an array provided you have correct IFS while doing that. Standard FOR loop handles individual items in that array as usual. Notice that the list given for IN variable must be "hard" quoted, that is, with single ticks.

IFS must be saved and restored since Bash does not treat an assignment the same way as a command. An alternate workaround is to wrap the assignment inside a function and call that function with a modified IFS. In that case separate saving/restoring of IFS is not needed. Thanks for "Bize" for pointing that out.

Share Edit Follow

edited Apr 19, 2015 at 22:28
Peter Mortensen
31.1k 22 109 132

answered Oct 10, 2014 at 11:33
ajaaskel
1,699 12 12

1 @ajaaskel you didn't fully understand my comment. Go in a scratch directory and issue these commands: `mkdir '!"#$%&'; touch '!"#$%&/()[]{} got you hahahaha - are no problem'` . They will only create a directory and a file, with weird looking names, I must admit. Then run your commands with the exact `IN` you gave: `IN='bla@some.com;john@home.com;Charlie Brown <cbrown@acme.com;!"#$%&/()[]{}*? are no problem;simple is beautiful :-)'` . You'll see that you won't get the output you expect. Because you're using a method subject to pathname expansions to split your string. – gniourf_gniourf Feb 25, 2015 at 7:26

1 @gniourf_gniourf Thanks for detailed comments on globbing. I adjusted the code to have globbing off. My point was however just to show that rather simple assignment can do the splitting job. – ajaaskel Feb 26, 2015 at 15:26

2

Here's my answer!

```
DELIMITER_VAL='='

read -d '' F_ABOUT_DISTRO_R <<"EOF"
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.4 LTS"
NAME="Ubuntu"
VERSION="14.04.4 LTS, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
```

```
PRETTY_NAME="Ubuntu 14.04.4 LTS"
VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
EOF

SPLIT_NOW=$(awk -F$DELIMITER_VAL '{for(i=1;i<=NF;i++){printf "%s\n", $i}}'
<<<"${F_ABOUT_DISTRO_R}")
while read -r line; do
   SPLIT+=("$line")
done <<< "$SPLIT_NOW"
for i in "${SPLIT[@]}"; do
    echo "$i"
done
```

Why this approach is "the best" for me?

Because of two reasons:

1. You do **not need to escape** the delimiter;
2. You will not have **problem with blank spaces**. The value will be properly separated in the array.

Share Edit Follow

edited Mar 10, 2022 at 7:42
tripleee
182k 35 288 336

answered Apr 4, 2016 at 19:54
Eduardo Lucio
2,127 2 29 48

1 The Awk shenanigan is just a clumsy way to reimplement `IFS="=" read -r` – tripleee Mar 10, 2022 at 7:43

1 I have nothing against Awk but using it to clumsily attempt to replace shell built-ins is not a good idea. – tripleee Mar 10, 2022 at 16:26

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.