

Set environment variables from file of key/pair values

TL;DR: How do I export a set of key/value pairs from a text file into the shell environment?

306

For the record, below is the original version of the question, with examples.

I'm writing a script in bash which parses files with 3 variables in a certain folder, this is one of them:

118

```
MINIENTREGA_FECHALIMITE="2011-03-31"
MINIENTREGA_FICHEROS="informe.txt programa.c"
MINIENTREGA_DESTINO="./destino/entrega-prac1"
```

This file is stored in `./conf/prac1`

My script `minientrega.sh` then parses the file using this code:

```
cat ./conf/$1 | while read line; do
    export $line
done
```

But when I execute `minientrega.sh prac1` in the command line it doesn't set the environment variables

I also tried using `source ./conf/$1` but the same problem still applies

Maybe there is some other way to do this, I just need to use the environment variables of the file I pass as the argument of my script.

bash variables environment-variables

▲ Same on unix: unix.stackexchange.com/questions/31797/... – [Ciro Santilli](#) 新疆改造中心 六四事件 法轮功 Nov 19 '14 at 14:13

▲ Same with Ruby: stackoverflow.com/questions/2139080/..., a gem that does it: github.com/bkeepers/dotenv – [Ciro Santilli](#) 新疆改造中心 六四事件 法轮功 Nov 19 '14 at 14:24

▲ This is a great question but is phrased way too specifically, with particular variable names ("MINIENTREGA_FECHALIMITE"? what does that mean?) and numbers (3). The general question is simply, "How do I export a set of key/value pairs from a text file into the shell environment". – [Dan Dascalescu](#) Nov 14 '18 at 5:05

▲ Also, this has already been answered on [unix.SE](https://unix.stackexchange.com/questions/31797/...) and is arguably more on-topic there. – [Dan Dascalescu](#) Dec 25 '18 at 3:12

128

Problem with your approach is the `export` in the `while` loop is happening in a sub shell, and those variable will not be available in current shell (parent shell of while loop).

Add `export` command in the file itself:





```
export MINIENTREGA_FECHALIMITE="2011-03-31"
export MINIENTREGA_FICHEROS="informe.txt programa.c"
export MINIENTREGA_DESTINO="./destino/entrega-prac1"
```


Then you need to source in the file in current shell using:

```
. ./conf/prac1
```

OR


```
source ./conf/prac1
```

-
- 4  Although reading the file line-by-line and passing each line to `export` is not ideal, the problem can also be fixed by simply using input redirection on the loop: `while read line; do ... ; done < ./conf/$1` — chepner Sep 2 '14 at 14:00
-
- 2  And if it's not from a file, use `< <(commands that generate output)` — o11c Aug 31 '17 at 0:10
-
- 2  You have a more [clean solution](#), I have a preference for `set -o allexport` — heralight Oct 28 '18 at 9:51
-
-  If using this `.env` file between systems, inserting `export` would break it for things like Java, SystemD, or other tools — Pred Feb 15 at 17:52
-

 This might be helpful:

559

```
export $(cat .env | xargs) && rails c
```

 Reason why I use this is if I want to test `.env` stuff in my rails console.

[gabrielf](#) came up with a good way to keep the variables local. This solves the potential problem when going from project to project.

```
env $(cat .env | xargs) rails
```

I've tested this with `bash 3.2.51(1)-release`

Update:

To ignore lines that start with `#`, use this (thanks to [Pete's comment](#)):

```
export $(grep -v '^#' .env | xargs)
```

And if you want to `unset` all of the variables defined in the file, use this:

```
unset $(grep -v '^#' .env | sed -E 's/(.*)=.*\/1/' | xargs)
```

Update:


To also handle values with spaces, use:

```
export $(grep -v '^#' .env | xargs -d '\n')
```

on GNU systems or:

```
export $(grep -v '^#' .env | xargs -0)
```

on BSD systems.

-
- 4  Thanks, I like that this doesn't require prepending anything to the file — allows for compatibility with Foreman (Procfile) `.env` format. — natevw Jan 6 '14 at 22:00
-

- 18 ▲ I came up with the solution: `env $(cat .env | xargs) rails` – gabrielf May 9 '14 at 12:02
- 3 ▲ This seems not to work if any of the env values have spaces, although I'm not actually sure what the best/desired way to specify values with spaces is. github.com/ddollar/foreman/issues/56 says it should work like `export $(cat .env)` but I don't know how to make that deal with spaces. – Dan Benamy Jan 3 '15 at 17:46
- 5 ▲ @BenjaminWheeler GNU linux has `-d` for setting the delimiter, so I'm trying `env $(cat .env | xargs -d '\n') rails`, but it still errors with a file not found if `.env` has spaces. Any idea why this doesn't work? – Bailey Parker Apr 17 '15 at 6:08
- 13 ▲ Here's a shorter variation `eval $(cat .env) rails` – manalang Apr 26 '16 at 15:57
- |

247 ▲ `-o allexport` enables all following variable definitions to be exported. `+o allexport` disables this feature.

```
set -o allexport
source conf-file
set +o allexport
```

- 7 ▲ Works like a charm! Even if `.env` file has comments in it. Thanks! – Slava Fomin II Nov 15 '16 at 9:10
- 6 ▲ And in one line `set -o allexport; source conf-file; set +o allexport` – HarlemSquirrel Dec 15 '16 at 2:21
- 1 ▲ This is a great way to read in a properties file, when the Jenkins EnvInject plug-in doesn't work. Thanks! – Teresa Peters Jan 31 '17 at 2:49
- 5 ▲ @CMCDragonkai, for POSIX, it would be `set -a; . conf-file; set +a`. – Charles Duffy Feb 27 '18 at 23:21
- 2 ▲ This method works if the environment variables has spaces in it. Many of the others do not. While the `eval()` method does, I also get a little weirded out by using `eval` – CommandZ Apr 5 '18 at 14:57
- |

60 ▲ `set -a`
`./env.txt`
`set +a`

▼ If `env.txt` is like:

```
VAR1=1
VAR2=2
VAR3=3
...
```

- ▲ Can you explain the `-a` and `+a`? – Otto Jun 24 '18 at 14:03
- 3 ▲ @Otto `-a` is equivalent to `allexport`. In other words, every variable assignment in the shell is `export` ed into the environment (to be used by multiple child processes). Also see this article gnu.org/software/bash/manual/html_node/The-Set-Builtin.html – Dan Kowalczyk Jun 25 '18 at 16:42

25

The `allexport` option is mentioned in a couple of other answers here, for which `set -a` is the shortcut. Sourcing the `.env` really is better than looping over lines and exporting because it allows for comments, blank lines, and even environment variables generated by commands. My `.bashrc` includes the following:

```
# .env Loading in the shell
dotenv () {
    set -a
    [ -f .env ] && . .env
    set +a
}

# Run dotenv on Login
dotenv

# Run dotenv on every new directory
cd () {
    builtin cd $@
    dotenv
}
```

▲ This looks nice, but you do you unload environment variables when you leave the directory? – Bastian Venthur
 🚩 Aug 1 '17 at 7:37

▲ I don't unset variables, and it's never been a problem. My apps tend to use variable names that are distinct, and if there is overlap, I'll set them to blank in that `.env` with `VAR=` . – gsf Aug 2 '17 at 14:41 ✎

20

```
eval $(cat .env | sed 's/^/export /')
```

1 ▲ Using `eval $(cat .env | sed 's/^[$]/export /')` allows you to have empty lines for better readability. – Mario Uher Jul 25 '15 at 11:09
 🚩

2 ▲ I find that `cat .env | sed 's/^[^$]/export /'` strips off the initial character. I.e. for a file `A=foo\nB=bar\n` I get `export =foo\nexport =bar\n`. This works better for skipping blank lines: `cat .env | sed '/^$/!s/^/export /'` . – Owen S. Mar 2 '17 at 18:26 ✎

▲ (I also note for the sake of UNIX code golfers that you don't need `cat` in either case: `eval $(sed 's/^/export /' .env)` works just as well.) – Owen S. Mar 2 '17 at 18:28 ✎

Here is another `sed` solution, which does not run `eval` or require `ruby`:

18

```
source <(sed -E -n 's/[^#]+/export &/ p' ~/.env)
```

This adds `export`, keeping comments on lines starting with a comment.

.env contents

```
A=1
#B=2
```

sample run

```
$ sed -E -n 's/^[^#]+/export &/ p' ~/.env
export A=1
#export B=2
```

I found this especially useful when constructing such a file for loading in a systemd unit file, with EnvironmentFile .

▲ does not support multiple lines in OSX – Abdenmour TOUMI Dec 3 '17 at 5:55 ✎
▼

12

I have upvoted user4040650's answer because it's both simple, and it allows comments in the file (i.e. lines starting with #), which is highly desirable for me, as comments explaining the variables can be added. Just rewriting in the context of the original question.

▼ If the script is called as indicated: `minientrega.sh prac1` , then minientrega.sh could have:

```
set -a # export all variables created next
source $1
set +a # stop exporting

# test that it works
echo "Ficheros: $MINIENTREGA_FICHEROS"
```

The following was extracted from the set documentation:

This builtin is so complicated that it deserves its own section. set allows you to change the values of shell options and set the positional parameters, or to display the names and values of shell variables.

`set [--abefhkmnptuvxBCEHPT] [-o option-name] [argument ...]` `set [+abefhkmnptuvxBCEHPT] [+o option-name] [argument ...]`

If no options or arguments are supplied, set displays the names and values of all shell variables and functions, sorted according to the current locale, in a format that may be reused as input for setting or resetting the currently-set variables. Read-only variables cannot be reset. In POSIX mode, only shell variables are listed.

When options are supplied, they set or unset shell attributes. Options, if specified, have the following meanings:

-a Each variable or function that is created or modified is given the export attribute and marked for export to the environment of subsequent commands.

And this as well:

Using '+' rather than '-' causes these options to be turned off. The options can also be used upon invocation of the shell. The current set of options may be found in \$-.

12

SAVE=\$(set +o) && set -o allexport && . .env; eval "\$SAVE"

▼ This will save/restore your original options, whatever they may be.

Using `set -o allexport` has the advantage of properly skipping comments without a regex.

`set +o` by itself outputs all your current options in a format that bash can later execute. Also handy:
`set -o` by itself, outputs all your current options in human-friendly format.

- 2 ▲ I would probably `exec env -i bash` to clear the existing environment before calling `eval` if you need to
■ unset variables that are only set within `.env`. – b4hand Sep 11 '15 at 19:46

Improving on Silas Paul's answer

- 9 exporting the variables on a subshell makes them local to the command.

▼ `(export $(cat .env | xargs) && rails c)`

Simpler:

- 8
1. grab the content of the file
 2. remove any blank lines (just incase you separated some stuff)
 3. remove any comments (just incase you added some...)
 4. add `export` to all the lines
 5. `eval` the whole thing

▼ `eval $(cat .env | sed -e /^$/d -e /^#/d -e 's/^/export /')`

Another option (you don't have to run `eval` (thanks to @Jaydeep)):

`export $(cat .env | sed -e /^$/d -e /^#/d | xargs)`

Lastly, if you want to make your life REALLY easy, add this to your `~/.bash_profile`:

`function source_envfile() { export $(cat $1 | sed -e /^$/d -e /^#/d | xargs); }`

(MAKE SURE YOU RELOAD YOUR BASH SETTINGS!!! `source ~/.bash_profile` or.. just make a new tab/window and problem solved) you call it like this: `source_envfile .env`

- 1 ▲ I had to read .env text from gitlab secret variable for a pipeline: Based on your solution this command worked
■ for me: `source <(echo $(sed -E -n 's/[^\#]+/ &/ p' <(echo "${2}" | tr -d '\r'))); .` Somehow
gitlab saves the secret variable with a windows carriage return, so I had to trim that with `tr -d '\r'`. –
metanerd Nov 24 '17 at 11:21 ✎

You can use your original script to set the variables, but you need to call it the following way (with stand-alone dot):

5
▼ `./minientrega.sh`

Also there might be an issue with `cat | while read` approach. I would recommend to use the approach `while read line; do done < $FILE`.

Here is a working example:

```
> cat test.conf
VARIABLE_TMP1=some_value

> cat run_test.sh
#!/bin/bash
while read line; do export "$line";
done < test.conf
echo "done"

> . ./run_test.sh
done

> echo $VARIABLE_TMP1
some_value
```

Building on other answers, here is a way to export only a subset of lines in a file, including values with spaces like `PREFIX_ONE="a word"` :

5

```
set -a
. <(grep '^[ ]*PREFIX_' conf-file)
set +a
```

White spaces in the value

1

There are many great answers here, but I found them all lacking support for white space in the value:

```
DATABASE_CLIENT_HOST=host db-name db-user 0.0.0.0/0 md5
```

I have found 2 solutions that work with such values with support for empty lines and comments.






One based on `sed` and [@javier-buzzi answer](#):

```
source <(sed -e /^$/d -e /^#/d -e 's/./declare -x "&"/g' .env)
```

And one with `read line` in a loop based on [@john1024 answer](#)

```
while read -r line; do declare -x "$line"; done < <(egrep -v "^(#|^$|^$)" .env)
```

The key here is in using `declare -x` and putting line in double quotes. I don't know why but when you reformat the loop code to multiple lines it won't work — I'm no bash programmer, I just gobbled together these, it's still magic to me :)

- 1  I had to modify the `sed` solution to get it to work. But first some explanation: `-e` is short for `--expression`, which just tells `sed` what operations to take. `-e /^$/d` deletes the empty lines from the output (not the file). `-e /^#/d` deletes the bash comments (lines that start with `#`) from the output. `'s/./declare -x "&"/g'` replaces (substitutes) the remaining lines with `declare -x "ENV_VAR=VALUE"`. When you source this, at least for me, it didn't work. Instead, I had to use `source <(sed -e /^$/d -e /^#/d -e 's/./declare -x &/g' .env)`, to remove the extra `"` wrapper. — jcasner Apr 10 '18 at 20:49 
-  I don't use `ENV_VAR="lorem ipsum"`, I have `ENV_VAR=lorem ipsum`, without quotes in the `.env` file. Now I'm not sure why, but this was probably problematic in other tools that parse this file. And instead of `lorem ipsum` I have ended with `"lorem ipsum"` value — with quotes. Thx for the explanations :) — Janusz Skonieczny Apr 11 '18 at 6:51 
- 1  If it was my choice, I wouldn't use `ENV_VAR="lorem ipsum"` either. In my use case, my hosting provider generates this file based on some configuration options I have set, and they insert the double quotes. So, I am forced to work around it. Thanks for your help here - saved me a lot of time trying to work out the correct `sed` options myself! — jcasner Apr 13 '18 at 13:53

I have issues with the earlier suggested solutions:

1

- @anubhava's solution makes writing bash friendly configuration files very annoying very fast, and also - you may not want to always export your configuration.
- @Silas Paul solution breaks when you have variables that have spaces or other characters that work well in quoted values, but `$()` makes a mess out of.

Here is my solution, which is still pretty terrible IMO - and doesn't solve the "export only to one child" problem addressed by Silas (though you can probably run it in a sub-shell to limit the scope):

```
source .conf-file
export $(cut -d= -f1 < .conf-file)
```

I came across this thread when I was trying reuse Docker `--env-file` s in a shell. Their format is not bash compatible but it is simple: `name=value` , no quoting, no substitution. They also ignore blank lines and `#` comments.

0

I couldn't quite get it posix compatible, but here's one that should work in bash-like shells (tested in zsh on OSX 10.12.5 and bash on Ubuntu 14.04):

```
while read -r l; do export "$(sed 's/=.*/' <<<$l)'="$(sed -E 's/^[^=]+=/' <<<$l)";
done < <(grep -E -v '^s*(#|$)' your-env-file)
```

It *will not* handle three cases in the example from the docs linked above:

- `bash: export: `123qwe=bar': not a valid identifier`
- `bash: export: `org.spring.config=something': not a valid identifier`
- and it will not handle the passthrough syntax (a bare `FOO`)

If you're getting an error because one of your variables contains a value that contains white spaces you can try to reset bash's `IFS` (Internal Field Separator) to `\n` to let bash interpret `cat .env` result as a list of parameters for the `env` executable.

0

Example:

```
IFS=$'\n'; env $(cat .env) rails c
```

See also:

- <http://tldp.org/LDP/abs/html/internalvariables.html#IFSREF>
- <https://unix.stackexchange.com/a/196761>

My .env:

-3

```
#!/bin/bash
set -a # export all variables

#comments as usual, this is a bash script
USER=foo
```



```
PASS=bar
```

```
set +a #stop exporting variables
```

Invoking:

```
source .env; echo $USER; echo $PASS
```

Reference <https://unix.stackexchange.com/questions/79068/how-to-export-variables-that-are-set-all-at-once>

4 ▲ These env variables will not be passed down to child processes and hence this will not solve op's problem. –
🚩 nicodjimenez Aug 7 '17 at 17:31

▲ you are right. I modified my answer to include `set -a` as seen here
🚩 unix.stackexchange.com/questions/79068/... – Tudor Ilisoi May 19 '18 at 19:41 ✎
