Variables in bash

here’s a model that may help nail down the behavior of variables in bash. Recall that there are two kinds of variables: local variables, and environment variables (also called global variables). The idea is an environment variable will propagate itself to a subshell, while a local variable is, as the name indicates, purely local to a given shell.

You can picture things this way. Each shell has a list of variables and their values. Some of the variables marked as environment variables. Two separate facts contribute to the behavior of variables:

1. In a given shell, a variable is either local or environment, but not both, i.e. it it either marked or it is not.
2. When a new shell is created, the variables that were marked in the calling shell are passed down, still marked to the new shell.

In a given shell, you can list all the variables by typing `set`. To specifically see the ones that are marked as environment variables, type `export`. (The process of passing a variable from a shell to a subshell is called exporting.)

If you define a variable as `FOO=10`, it is initially unmarked. To mark a variable for export, i.e., to make it an environment variable, you simply write `export FOO`. You can also define and mark a variable for export at the same time: `export FOO=10`. To unmark a variable, you can use `export -n FOO`. The key thing to notice here is that the fact that a variable is an environment variable or not is simply a mark; changing the value of an existing variable does not affect the mark. Hence, consider the following sample interaction (comments in italics):

```
[cs214-sp02]$ FOO=tarzan       # define FOO
[cs214-sp02]$ echo $FOO        # tarzan
[cs214-sp02]$ export FOO       # mark for export
[cs214-sp02]$ FOO=jane         # change its value
[cs214-sp02]$ bash             # invoke subshell
[cs214-sp02]$ echo $FOO
```
(I indicate a subshell by appropriate indentation.) In the above, we define a variable \textit{FOO}, mark it for export, and then change its value. The change in value does not affect the mark. So, when you invoke a new shell, the variable \textit{FOO}, marked for export, will be passed to the new shell, with its current value.

The second fact mentioned earlier helps explain why changes to an environment variable in a subshell does not affect the environment variables in calling shells. This is because each subshell gets a \textit{copy} of the environment variables of its parent. Hence, the interaction below:

\begin{verbatim}
[cs214-sp02]$ BAR=tarzan
[cs214-sp02]$ echo $BAR
tarzan
[cs214-sp02]$ export BAR
[cs214-sp02]$ echo $BAR
export BAR
[cs214-sp02]$ bash
invoke subshell
[cs214-sp02]$ echo $BAR
tarzan
sanity check...
[cs214-sp02]$ BAR=jane
[cs214-sp02]$ echo $BAR
jane
[cs214-sp02]$ declare -x BAR="jane"
Note that BAR is still marked
[cs214-sp02]$ export | grep BAR
[cs214-sp02]$ echo $BAR
Original value
[cs214-sp02]$ exit
[cs214-sp02]$
\end{verbatim}

Specifically, since the subshell is acting only on a copy of the \textit{BAR} variable, any changes it makes only affects its copy. The variable in the calling shell is unaffected.

Remembering the two facts above help answer most variable-related questions: the fact that whether or not a variable is an environment variable is simply a mark attached to the variable, and the fact that a copy of those variables that are marked is passed to subshells.