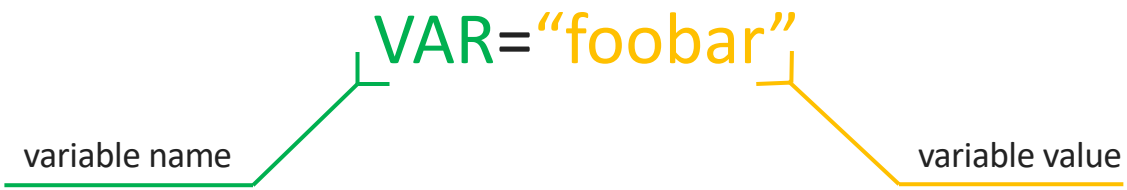
Variables

- What is a variable
- Variable types
- Sourcing variables

What is a variable

Variable is a name, that have some value associated with it.
Important trait of a variable is that the associated value can be changed.

*A name, which associated value cannot be changed is usually referred as constant.



Variable tells your language (Bash, in our case) where to look for it's value in memory. Bash does the heavy lifting, so you don't have to think about the memory organization while coding in it.

Variable types

The main caveat is that Bash doesn't have var types as a concept. All of the variables in Bash are **character strings.**

BUT!

Bash is able to determine the variable "type" by it's value and do certain operations depending on the context, e.g. you can do arithmetic operations on variables that contain only digits.

For the sake of convenience we're going to call variables by the type of their value, e.g.

FOO=123 - integer variable

BAR="baz" - string variable

Variables comparison and tests

To compare variables or test them, use [ ] or [[ ]] blocks. They're almost identical, but have differences.

[ ] block – A built-in command (the opening bracket is the command's name). Lacks functionalty, but is portable to any other shell.

[[ ]] block - A **bash** syntax construct (a keyword). Will not work in other Bourne-compatible shells (zsh, ksh, etc.), but has more powers, like regexp. comparison, wildcards, etc. Also it is considered safer to use.

General rule: Use [ ] if you don't need complex comparison, else use [[ ]].

String comparison operators

Let's assume that **FOO="pizza",** then:

Operator	Description	Example
== or =	comparing strings on equality	[[$FOO == "pizza"]] (true)
!=	comparing strings on inequality	[[$FOO != "bar"]] (true)
<	lexicographic comparison (before)	[[$FOO < "eggs"]] (false)
>	lexicographic comparison (after)	[[$FOO > "ham"]] (true)
=~	Regexp match (Bash 3.x only)	[[$FOO =~ .*zz.*]] (true)
-z	testing string for having zero length	[[-z $FOO]] (false)
-n	testing string for having non-zero length	[[-n $FOO]] (true)

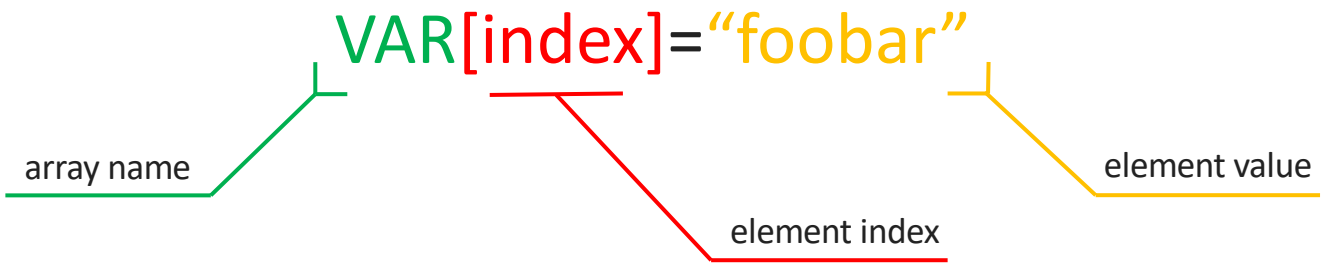
Integer comparison operators

Let's assume that **FOO=2020,** then:

Operator	Description	Example
-eq	comparing integers on equality	[[$FOO –eq 3]] (false)
-ne	comparing integers on inequality	[[$FOO –ne 3]] (true)
-lt	integer less than	[[$FOO –lt 2020]] (false)
-le	integer less or equal to	[[$FOO –le 2020]] (true)
-gt	integer greater than	[[$FOO –gt 2020]] (false)
-ge	integer greater or equal to	[[$FOO –ge 2020]] (true)

Arrays

In Bash, arrays are created by declaring a variable like this:



This will create array with name VAR with one element “foobar” on the [index] position. Index is any positive integer or zero.

You can also initialize the whole array at once like this:

```
VAR=(“foo” “bar” “baz”)
```

Arrays usage

Command	Description
${VAR[1]}	Get first element of an array
${VAR[@]}	Get all elements of an array
${#VAR[1]}	Get the length of the first element of an array
${#VAR[@]}	Get the number of elements in an array
${VAR[@]:3:2}	Get 2 elements of an array starting from element with index 3
${VAR[@]/foo/bar}	Search and replace element “foo” with “bar”
unset VAR[2]	Unset element with index 3 in an array
for i in "${VAR[@]}"	Iterate over an array

Sourcing variables

If you need to set a lot of variables in bulk in current shell or want to have separate file with variables for your script, you should use the **source** command.

From **man**:

source filename [arguments]
Read and execute commands from filename in the current shell environment and return the exit status of the last command executed from filename.

Also note that **source** and **.** are the same command.

source filename [arguments]
. filename [arguments]

Thanks for watching!