Variables

- Variable names
  - Must start with a letter or underscore
  - Number can be used anywhere else
  - Do not use special characters such as `@,#,%,$`
  - Case sensitive
  - Allowed: `VARIABLE, VAR1234able, var_name, _VAR`
  - Not allowed: `1var, %name, $myvar, var@NAME, myvar-1`
- To reference a variable, prepend **`$`** to the name of the variable
- Example: `$PATH, $LD_LIBRARY_PATH, $myvar` etc.

Global and Local Variables

- Two types of variables:
  - Global (Environmental) variables
    - Inherited by subshells (child process, see next slide)
    - provide a simple way to share configuration settings between multiple applications and processes in Linux
    - Using all uppercase letters by convention
    - Example: `PATH, LD_LIBRARY_PATH, DISPLAY` etc.
    - `printenv/env` list the current environmental variables in your system.
  - Local (shell) variables
    - Only visible to the current shell
    - Not inherited by subshells

Editing Variables

Local (Shell) variables is only valid within the current shell, while environment variables are valid for all subsequently opened shells.

Type	sh/ksh/bash	csh/tcsh
Shell (local)	`name=value`	`set name=value`
Environment (global)	`export name=value`	`setenv name value`

With export	Without export

`$ export v1=one`
`$ bash`
`$ echo $v1`
→`one`
	`$ v1=one`
	`$ bash`
	`$ echo $v1`
	→

Script Example (~/.bashrc)

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
Fi

# User specific aliases and functions
export PATH=$HOME/packages/bin:$PATH
export LD_LIBRARY_PATH=$HOME/packages/lib:$LD_LIBRARY_PATH
alias qsubI="qsub -I -X -l nodes=1:ppn=20 -l walltime=01:00:00
–A my_allocation"
alias lh="ls -altrh"
```

List of Some Environment Variables

PATH	A list of directory paths which will be searched when a command is issued
LD_LIBRARY_PATH	colon-separated set of directories where libraries should be searched for first
HOME	indicate where a user's home directory is located in the file system.
PWD	contains path to current working directory.
OLDPWD	contains path to previous working directory.
TERM	specifies the type of computer terminal or terminal emulator being used
SHELL	contains name of the running, interactive shell.
PS1	default command prompt
PS2	Secondary command prompt
HOSTNAME	The systems host name
USER	Current logged in user's name

Quotations

Single quotation

- Enclosing characters in single quotes `( ' )` preserves the literal value of each character within the quotes. A single quote may not occur between single quotes, even when preceded by a backslash.

Double quotation

- Enclosing characters in double quotes `( " )` preserves the literal value of all characters within the quotes, with the exception of `'$', '`'`, `'\'`

Back “quotation?”

- Command substitution ``( ` ` )`` allows the output of a command to replace the command itself, enclosed string is executed as a command, almost the same as `$()`

Special Characters (1)

#	Start a comment line.
$	Indicate the name of a variable.
\	Escape character to display next character literally
{ }	Enclose name of variable
;	Command separator. Permits putting two or more commands on the same line.
;;	Terminator in a case option
.	"dot" command, equivalent to `source` (for bash only)
|	Pipe: use the output of a command as the input of another one
> <	Redirections (`0<:` standard input; `1>:` standard out; `2>:` standard error)

Special Characters (2)

`$?`	Exit status for the last command, 0 is success, failure otherwise
`$$`	Process ID variable.
`[ ]`	Test expression, eg. if condition
`[[ ]]`	Extended test expression, more flexible than [ ]
`$[ ]`, `$(( ))`	Integer expansion
`||`, `&&`, `!`	Logical OR, AND and NOT

Integer Arithmetic Operations

Operation	Operator
Addition	`+`
Subtraction	`–`
Multiplication	`*`
Division	`/`
Exponentiation	`** (bash only)`
Modulo	`%`

String Comparisons

Operation	bash
Equal to	`if [ $a == $b ]`
Not equal to	`if [ $a != $b ]`
Zero length or null	`if [ -z $a ]`
Non zero length	`if [ -n $a ]`

Logical Operators

Operation	Example
! (NOT)	if [! —e test]
&& (AND)	if [-f test] && [-s test]
	if [[-f test && -s test]]
	if (-e test && ! —z test)
|| (OR)	if [-f test1] || [-f test2]
	if [[-f test1 || -f test2]]

File Operations

Operation	bash
File exists	`if [ -e test ]`
File is a regular file	`if [ -f test]`
File is a directory	`if [ -d /home ]`
File is not zero size	`if [ -s test ]`
File has read permission	`if [ -r test ]`
File has write permission	`if [ -w test ]`
File has execute permission	`if [ -x test ]`

Functions

- A function is a code block that implements a set of operations. Code reuse by passing parameters,
- Syntax:

```
function_name ()
    {
        command...
    }
```

- By default all variables are global.
- Modifying a variable in a function changes it in the whole script.
- Create a local variables using the local command, which is invisible outside the function `local var=value local varName`

Pass Arguments to Bash Scripts

- Note the difference between the arguments passed to the script and the function.
- All parameters can be passed at runtime and accessed via `$1, $2, $3…, add {} when >=10`
- `$0:` the shell script name
- Array variable called `FUNCNAME` contains the names of all shell functions currently in the execution call stack.
- `$* or $@:` all parameters passed to a function
- `$#:` number of positional parameters passed to the function
- $?: exist code of last command
- $$: PID of current process