

DE 0 A 100 CON BASH SHELL SCRIPTING Y AWK

ADOLFO SANZ DE DIEGO

COMMIT-CONF 2018



2 AUTOR

2.1 ADOLFO SANZ DE DIEGO

Asesor. Desarrollador. Profesor. Formador.

- Blog: asanzdiego.com
- Correo: asanzdiego@gmail.com
- GitHub: github.com/asanzdiego
- Twitter: twitter.com/asanzdiego
- LinkedIn: in/asanzdiego
- SlideShare: slideshare.net/asanzdiego

2.2 DISCLAIMER

- He intentado montar una charla útil para principiantes, pero con tips para gente con más conocimientos.
- Espero haberlo conseguido :-)

3 SHELL SCRIPT

3.1 INTRODUCCIÓN

- Un shell script es un fichero de texto con comandos.
- Para ejecutarlo no hay ni que compilar, ni tener nada **instalado**, y puedes utilizar todos los comandos del sistema.
- Usalo para **automatizar tareas del sistema y/o procesar datos** de forma rápida.
- Usalo para hacer **pequeños scripts**, no grandes programas, para eso tienes otros lenguajes.

3.2 HOLA MUNDO

```
#!/bin/bash  
  
# script showing a "Hello world!"  
echo "Hello world!"
```

3.3 PERMISOS

- Antes de ejecutar hay que **darle permisos**, pero recuerda, un gran poder conlleva una gran responsabilidad :-)

```
$ chmod +x 01_hello_world.sh
```


3.4 EJECUCIÓN

- Para ejecutar un script:
 - si está en el \$PATH, el nombre directamente
 - si no, desde la carpeta, ./nombre.sh

```
$ ./01_hello_world.sh
```

examples/01_hello_word.sh

3.5 NOMBRES

- Estas son mis **reglas de estilo** (en realidad de Google), si no te gustan tengo otras. :-)

```
ficheros_shell_scripts.sh  
VARIABLES_DE_ENTORNO  
variables_locales  
nombres_de_funciones
```

3.6 INICIO

- Es una buena práctica **empezar los shells scripts así:**

```
#!/bin/bash

# Short description of the script

set -o errexit # the script ends if a command fails
set -o pipefail # the script ends if a command fails in a pipe
set -o nounset # the script ends if it uses an undeclared variable
# set -o xtrace # if you want to debug
```

3.7 EXIT

- Es una buena práctica terminar los shells scripts con un código de retorno:
 - **mayor de 0** si ha habido un error
 - **igual a 0** si termina correctamente (si no pones 'exit')

```
#!/bin/bash

num_params=$#

if [ $num_params -lt 1 ]; then
    echo "At least one parameter must be introduced."
    exit 1 # error and exits with a return code > 0
fi

echo "All ok" # ok and exits with a return code = 0
```

3.8 PARCIALES

- Podemos guardar el resultado de la ejecución de comandos en variables con `$(codigo)`:

```
date=$(date +%Y-%m-%d %H:%M:%S '')
```

3.9 FUNCIONES

- Es una buena práctica asignar los **parámetros de la función** al principio ya sean como variables locales o, si es necesario, globales.

```
my_function() {  
    local function_param_1="$1"      # 1st param assigned as local  
    global_param_2=${2:-default}     # 2nd param assigned as global (default)  
    local function_num_params=$#     # numbers of params assigned as local  
    local all_function_params=($@)   # all params assigned as a local  
}
```

```
my_function function_param_1 function_param_2 ... function_param_N
```

3.10 MAIN

- Es una buena práctica **estructurar el código en funciones** y tener una función main que llamamos al final del script con todos los parámetros.

```
# Main function
main() {

    check "$@"
    params "$@"
    print
}

main "$@" # call the main function with all the parameters
```

3.11 PARÁMETROS

- Los parámetros los cogemos de la línea de comandos cuando ejecutamos.

```
# Default values
default_2="Commit Conf"

param_1=$1 # the first script param
param_2=${2:-${default_2}} # the second script param (with default value)
num_params=$# # the numbers of script params
all_params=($@) # all params assigned as an array
```

```
$ ./02_parameters.sh param_1 param_2 ... param_N
```

examples/02_parameters.sh

3.12 TEMPLATE

asanzdiego / commit-conf-charla-shell-script-y-awk

Unwatch

1

Star

1

Fork

0

<> Code

Issues 0

Pull requests 0

Projects 0

Wiki

Insights

Settings

Branch: master

commit-conf-charla-shell-script-y-awk / resources / template.sh

Find file

Copy path

asanzdiego

cambios menores

460b2b5 3 days ago

1 contributor

Executable File | 105 lines (89 sloc) | 1.73 KB

Raw

Blame

History

1

#!/bin/bash

2

3

small template for my bash shell scripts.

4

5

set -o errexit # the script ends if a command fails

6

set -o pipefail # the script ends if a command fails in a pipe

7

set -o nounset # the script ends if it uses an undeclared variable

8

set -o xtrace # if you want to debug

9

10

Defaults values

11

logLevel=4 #4-debug;3-info;2-success;1-warning;0-error

12

alpha=true

13

beta="beta"

14

Plantilla de Bash Shell Script

3.13 CHULETA

REDIRECCIONES		
output (salida estándar)		
tee fichero	#	output a fichero y a pantalla
> fichero	#	output a fichero
>> fichero	#	output al final del fichero
> /dev/null	#	descarta output
error		
2>&1	#	error a output
2> fichero	#	error a fichero
2>> fichero	#	error al final del fichero
2> /dev/null	#	descarta error
output y error		
2>&1 tee fichero	#	ambos a fichero y a pantalla
&2 fichero	#	ambos a fichero
&2> fichero	#	ambos al final del fichero
VARIABLES		
variables de entorno		
\$PWD	#	directorio de trabajo actual
\$OLDPWD	#	directorio de trabajo anterior
\$PPID	#	identificador del proceso padre
\$HOSTNAME	#	nombre del ordenador
\$USER	#	nombre del usuario
\$HOME	#	directorio del usuario
\$PATH	#	rutas búsqueda de comandos
\$LANG	#	idiona para los mensajes
\$FUNCNAME	#	nombre función en ejecución
\$LINENO	#	numero de línea actual (del script)
\$RANDOM	#	numero aleatorio
variables especiales		
\$0	#	nombre del script
\$N	#	parámetro N
\$*	#	identificador del proceso actual
\$*	#	identificador del último proceso
\$@	#	todos los parámetros recibidos
\$*	#	numero de parámetros recibidos
\$? # (\$normal, >\$error)	#	código de retorno del último comando
\$*	#	\$1\$2, \$2\$3, ... \$(N-1)\$N(N)
ARRAYS		
declare -a ARRAY	#	declaración array
ARRAY=(valor1 ... valorN)	#	asignación compuesta
ARRAY[N]=valorN	#	asignación simple
ARRAY=([N]=valorN valorM [P]=valorP)	#	asigna celdas N, M y P
\${ARRAY[N]}	#	valor celda N
\${ARRAY[*]}	#	todos los valores

Autor: Adolfo Sanz De Diego ([github](#) - [blog](#)) ([licencia](#)) ([licencia](#)) ([licencia](#)) Licencia: CC-BY-SA

OPERADORES		
operadores aritméticos		
+	#	suma
-	#	resta
*	#	multiplicación
/	#	división
%	#	resto
++	#	incremento
--	#	decremento
operadores comparaciones numéricas		
numero1 -eq numero2	#	numero1 igual que numero2
numero1 -ne numero2	#	numero1 distinto que numero2
numero1 -lt numero2	#	numero1 menor que numero2
numero1 -le numero2	#	numero1 menor o igual que numero2
numero1 -gt numero2	#	numero1 mayor que numero2
numero1 -ge numero2	#	numero1 mayor o igual que numero2
operadores lógicos		
!	#	NOT
&&, -a	#	AND
, -o	#	OR
operadores de ficheros		
-e fichero	#	existe
-s fichero	#	no está vacío
-f fichero	#	normal
-d fichero	#	directorio
-h fichero	#	enlace simbólico
-r fichero	#	permiso de lectura
-w fichero	#	permiso de escritura
-x fichero	#	permiso de ejecución
-O fichero	#	propietario
-G fichero	#	pertenece al grupo
f1 -ef f2	#	f1 y f2 enlaces mismo archivo
f1 -et f2	#	f1 más nuevo que f2
f1 -ot f2	#	f1 más antiguo que f2
operadores de cadenas		
-a cadena	#	no vacía
-z cadena	#	vacía
cadena1 = cadena2	#	cadena1 igual a cadena2
cadena1 == cadena2	#	cadena1 igual a cadena2
cadena1 != cadena2	#	cadena1 distinta a cadena2

Autor: Adolfo Sanz De Diego ([github](#) - [blog](#)) ([licencia](#)) ([licencia](#)) ([licencia](#)) Licencia: CC-BY-SA

ENTRECOMILLADO	
\$! RUTA	# ruta al intérprete (/bin/bash)
\$'carácter'	# valor literal del carácter
línea1 \ línea2	# para escribir en varias líneas
"cadena"	# valor literal cadena
"\$cadena"	# valor literal cadena, excepto \$ ' \
EXPANSIÓN	
[prefijo]{cad1,[...],cadN}[sufijo]	# = precadisu f ... precadisu f
\${VARIABLE:=valor}	# si VARIABLE nula, retorna valor
\${VARIABLE:=valor}	# si VARIABLE nula, asigna valor
\${VARIABLE:=mensaje}	# si VARIABLE nula, mensaje error y fin
\${VARIABLE:inicio}	# recorta desde inicio hasta el final
\${VARIABLE:inicio:longitud}	# recorta desde inicio hasta longitud
\${#prefijo}	# número de variables con prefijo
\${#VARIABLE}	# número de caracteres de VARIABLE
\${ARRAY[*]}	# elementos de ARRAY
\${VARIABLE%patrón}	# elimina mínimo patrón desde inicio
\${VARIABLE#patrón}	# elimina máximo patrón desde inicio
\${VARIABLE%patrón}	# elimina mínimo patrón desde fin
\${VARIABLE#patrón}	# elimina máximo patrón desde fin
\${VARIABLE/patrón/reemplazo}	# reemplaza primera coincidencia
\${VARIABLE//patrón/reemplazo}	# reemplaza todas las coincidencias
\${(expresión)}	# sustituye expresión por su valor
\${expresión}	# sustituye expresión por su valor
EJECUCIÓN	
/comando	# ejecuta desde directorio actual
\$RM/comando	# ejecuta desde cualquier sitio
comando	# ejecuta si está en el \$PATH
. script	# ejecuta exportando variables
\$comando param1 ... paramN	# ejecuta de forma literal
'comando param1 ... paramN'	# ejecuta sustituyendo variables
comando &	# ejecuta en segundo plano
c1 c2	# redirige salida c1 a entrada c2
c1 ; c2	# ejecuta c1 y luego c2
c1 && c2	# ejecuta c2 si c1 termina sin errores
c1 c2	# ejecuta c2 si c1 termina con errores
ARGUMENTOS DE LÍNEA DE COMANDOS	
while getopts "ht:" option ; do case "\$option" in h) DO_HELP=1 ;; t) argument=\$OPTARG ; DO_SEARCH=1 ;; *) echo "Invalid" ; return ;; esac done	# getops + "opciones disponibles" # mientras haya argumentos # seleccionamos # sin opciones # -s con opciones en \$OPTARG # error

Autor: Adolfo Sanz De Diego ([github](#) - [blog](#)) ([licencia](#)) ([licencia](#)) ([licencia](#)) Licencia: CC-BY-SA

ESTRUCTURAS DE CONTROL	
if expresión; then bloque1 elif expresión2; then bloque2 else bloque3 fi	# condicional # si expresión1 entonces # bloque1 # sino y expresión2 entonces # bloque2 # si ninguna entonces # bloque3
case VARIABLE in patrón1 ... patrónM) bloque1 ;; patrón2 ... patrónN) bloque2 ;; *) bloqueDefecto ;; esac	# selectiva # si VARIABLE coincide con patrones1 # entonces bloque1 # si VARIABLE coincide con patrones2 # entonces bloque2 # si ninguna # entonces bloqueDefecto
for VARIABLE in LISTA; do bloque done	# iterativa con lista # ejecuta bloque sustituyendo VARIABLE por cada elemento de LISTA
for ((exp1; exp2; exp3;)); do bloque done	# iterativa con contador # primero se evalúa exp1 # luego mientras exp2 sea cierta # se ejecutan el bloque y exp3
while expresión; do bloque done	# Bucle "mientras" # se ejecuta bloque # mientras expresión sea cierta
until expresión; do expresión done	# Bucle "hasta" # se ejecuta bloque # hasta que expresión sea cierta
[function] expresión () { ... { return [valor] } ... }	# función # se invoca con # nombreFunción [param1 ... paramN]
INTERACTIVIDAD	
read [-p cadena] [variable1 ...]	# input # lee teclado y asigna a variables # puede mostrarse un mensaje antes # si ninguna variable, REPLY = todo
echo cadena -e no hace salto de línea -E interpreta caracteres con \	# output # manda el valor de la cadena # a la salida estándar
printf	# output formateado (igual que C)
CONTROL DE PROCESOS	
comando &	# ejecuta en segundo plano
bg númeroProceso	# continúa ejecución en segundo plano
fg númeroProceso	# continúa ejecución en primer plano
jobs	# muestra procesos en ejecución
kill señal PID[númeroProceso1]	# mata proceso(s) indicado(s)
exit código	# salir con código de retorno # (0=normal, >0=error)
trap [comando] [código1 ...]	# ejecuta comando cuando señal(es)
wait [PID[númeroProceso1]]	# espera hasta fin proceso(s) hijo(s)
nice -n prioridad comando	# ejecuta comando con prioridad [-20/19]
renice -n prioridad comando	# modifica prioridad comando [-20/19] # -20 máxima prioridad y 19 mínima

Autor: Adolfo Sanz De Diego ([github](#) - [blog](#)) ([licencia](#)) ([licencia](#)) ([licencia](#)) Licencia: CC-BY-SA

Chuleta de Bash Shell Script

3.14 SHELLCHECK

ShellCheck - A shell script static analysis tool

ShellCheck is a GPLv3 tool that gives warnings and suggestions for bash/sh shell scripts:

```
vidar@vidarholen ~$ shellcheck myscript

In myscript line 7:
if (( $n > 3,5 ))
    ^-- Don't use $ on variables in (( )),
        ^-- (( )) doesn't support decimals, Use bc or awk.

In myscript line 16:
[[ $1 == $result ]] && mode=lookup
    ^-- Quote the rhs of = in [[ ]] to prevent glob interpretation.

vidar@vidarholen ~$
```

The goals of ShellCheck are


- To point out and clarify typical beginner's syntax issues that cause a shell to give cryptic error messages.
- To point out and clarify typical intermediate level semantic problems that cause a shell to behave strangely and counter-intuitively.
- To point out subtle caveats, corner cases and pitfalls that may cause an advanced user's otherwise working script to fail under future circumstances.

ShellCheck, el lint de Bash Shell Script

3.15 GOOGLE GUIDE

Shell Style Guide

Revision 1.26
Paul Armstrong
Too many more to mention

Each style point has a summary for which additional information is available by toggling the accompanying arrow button that looks this way: . You may toggle all summaries with the big arrow button:


 Toggle all summaries

Table of Contents

Shell Files and Interpreter Invocation	File Extensions SUID/SGID
Environment	STDOUT vs STDERR
Comments	File Header Function Comments Implementation Comments TODO Comments
Formatting	Indentation Line Length and Long Strings Pipelines Loops Case statement Variable expansion Quoting
Features and Bugs	Command Substitution Test, [and [[Testing Strings Wildcard Expansion of Filenames Eval Pipes to While
Naming Conventions	Function Names Variable Names Constants and Environment Variable Names Source Filenames Read-only Variables Use Local Variables Function Location main
Calling Commands	Checking Return Values Builtin Commands vs. External Commands
Conclusion	

Guía de estilo de Google de Bash Shell Script

3.16 TUTORIAL

Bash Scripting Tutorial for Beginners

• Lubos Rendek

• Programming & Scripting

• 27 December 2017

Bash Shell Scripting Definition

Bash

Bash is a command language interpreter. It is widely available on various operating systems and is a default command interpreter on most GNU/Linux systems. The name is an acronym for the 'Bourne-Again SHell'.

Shell

Shell is a macro processor which allows for an interactive or non-interactive command execution.

Scripting

Scripting allows for an automatic commands execution that would otherwise be executed interactively one-by-one.

Bash Shell Script Basics

Do not despair if you have not understood any of the above **Bash Shell Scripting** definitions. It is perfectly normal, in fact, this is precisely why you are reading this Bash Scripting tutorial.

Contents

- 1. Bash Shell Scripting Definition
- 2. Bash Shell Script Basics
- 3. What is Shell
- 4. What is Scripting
- 5. What is Bash
- 6. File Names and Permissions
- 7. Script Execution
- 8. Relative vs Absolute Path
- 9. Hello World Bash Shell Script
- 10. Simple Backup Bash Shell Script
- 11. Variables
- 12. Input, Output and Error Redirections
- 13. Functions
- 14. Numeric and String Comparisons
- 15. Conditional Statements
- 16. Positional Parameters
- 17. Bash Loops
 - 17.1. For Loop

Tutorial de Bash Shell Script

4 AWK

4.1 INTRODUCCIÓN

- Es una hoja de cálculo por línea de comandos.
- Tiene su propio lenguaje que es muy parecido a C.
- Muy útil para procesar datos dentro de un shell script.
- Muy útil para hacer cosas raras con datos y que con una hoja de cálculo normal es difícil de hacer.
- Muy útil cuando hay muchos datos y una hoja de cálculo se queda colgada.

4.2 EJECUCIÓN

```
awk 'awk_program' data_file
```

```
awk -f 'awk_file' data_file
```


4.3 GRADES

- Sacar las medias de los alumnos:

Pepito	4.4	3.1	5.7
Fulanito	4.2	6.5	8.8
Menganito	5.6	5.0	5.3

```
awk '{ print $1"="($2+$3+$4)/3 }' 03_grades.csv
```

examples/04_grades.sh

4.4 ROLES

- Agrupar por rol:

```
Pepito:Jefe,Sistemas  
Fulanito:Jefe,Desarrollo  
Menganito:Operario,Sistemas,Desarrollo
```

```
Sistemas -> Menganito Pepito  
Operario -> Menganito  
Jefe -> Fulanito Pepito  
Desarrollo -> Menganito Fulanito
```

4.5 SIN AWK

```
roles_file=./05_roles.csv

roles=$(cut -d : -f 2 $roles_file | sed 's/,/\n/g' | sort | uniq)

for rol in $roles; do
    echo -n "${rol} -> "
    echo $(grep -E "${rol}" "${roles_file}" | cut -d : -f 1)
done
```

examples/06_roles_sin_awk.sh

4.6 CON AWK

```
# this will run only once at first
BEGIN { FS = ",|:" }
# this will be executed for each of the lines in the file
{
    name=$1
    for (i=2; i<=NF; i++) {
        roles[$i]=roles[$i]" "old_names
    }
}
# This will only run once at the end
END {
    for (rol in roles) {
        print rol" -> " roles[rol]
    }
}
```

examples/08_roles.awk

4.7 TUTORIAL

Grymoire
Navigation

Unix/Linux

Quotes

Bourne Shell

C Shell

File Permissions

Regular Expressions

grep

awk UPDATED

sed UPDATED

find

tar

inodes

Security

IPv6

Wireless

Hardware

spam

Deception

PostScript

Halftones

Privacy

Bill of Rights

References

Top 10 reasons to avoid CSH

sed Chart PDF

awk Reference HTML

Magic

Search

About

Donate NEW

Google+: [Bruce Barnett](#)

Twitter: [@grymoire](#)

Blog: [Wordpress Blog](#)

Awk

Last modified: Thu Apr 23 16:37:47 EDT 2015

Part of the [Unix tutorials](#) And then there's [My blog](#)

Table of Contents

Why learn AWK?

Basic Structure

Executing an AWK script

Which shell to use with AWK?

Dynamic Variables

The Essential Syntax of AWK

Arithmetic Expressions

Summary of AWK Commands

AWK Built-in Variables

Associative Arrays

Picture Perfect PRINTF Output

Flow Control with next and exit

AWK Numerical Functions

String Functions

User Defined Functions

AWK patterns

Formatting AWK programs

Environment Variables

AWK, NAWK, GAWK, or PERL

Copyright 1994,1995 Bruce Barnett and General Electric Company

Copyright 2001, 2004, 2013, 2014 Bruce Barnett

All rights reserved

Tutorial de AWK

5 ACERCA DE

5.1 LICENCIA

Creative Commons Reconocimiento-CompartirIgual 3.0

5.2 FUENTES

github.com/asanzdiego/commit-conf-charla-shell-script-y-awk

5.3 SLIDES

Las slides están hechas con **MarkdownSlides**.

6 PREGUNTAS

7 GRACIAS