# DE O 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

ejemplos/01\_hola\_mundo.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 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:

```
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 PARÁMETROS

• Los parámetros los cogemos de la linea de comandos cuando ejecutamos.

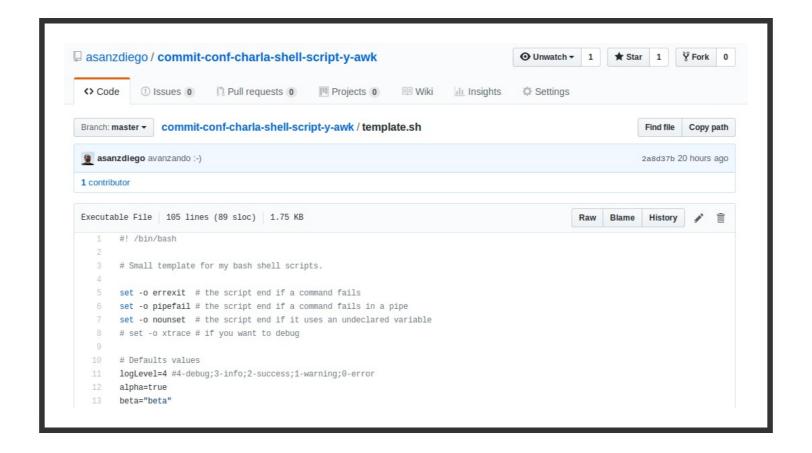
```
# Default values
default_2="Commit Conf"

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
```

ejemplos/02\_parametros.sh

#### 3.11 TEMPLATE



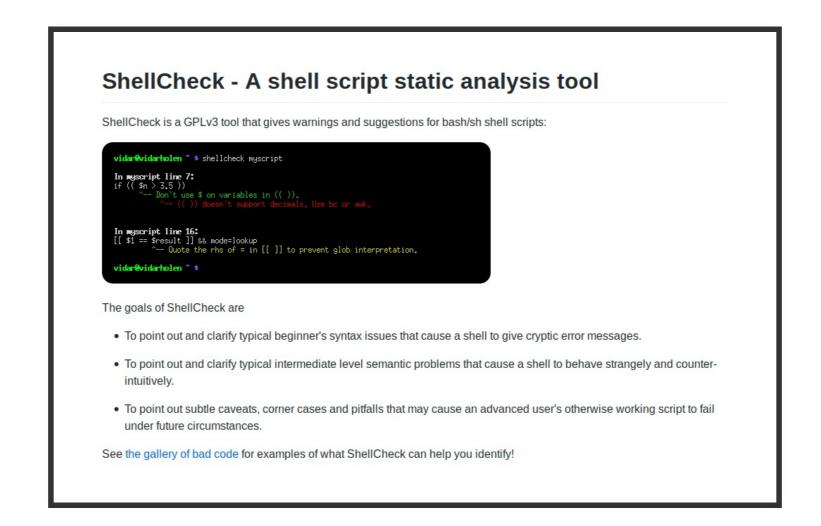
template.sh

#### 3.12 CHULETA

				ENTRECOMILLADO		ESTRUCTURAS DE CONTROL	
REDIRECCIONES		OPERADORES		#1 RUTA	# ruta al interprete (/bin/bash)	if expresion; then bloquei	# condicional
output (salida estándar)		operadores aritméticos		Noaracter	# valor literal del carácter	elif expresion2; them	<pre># si expresion1 entonces # bloque1</pre>
tee fichero	# output a fichero y a pantalla	•	# suns	lineal \ linea2	# para escribir en varias lineas	bloque2	# sino y expresión2 entonces # bloque2
> fichero	# output a fichero		# resta # multiplicación	"cadena"	# valor literal cadena	bloque3 # si ninguna entonces fi # bloque3	
>> fichero	# output al final del fichero	-	# division	"cadera"	# valor literal cadena, excepto \$ ' \		# bloque2
> /dev/null	# descarta output	7	# casto	EXPANSIÓN		case VARIABLE in patronil[[patroniN]	# selectiva
error				[prefijo]{cadi,[,],cadN}[sufijo]	# = precadisuf precadisuf	bloque1 ::	# si VARIABLE coincide con patrone # entonces bloquei
241	# error a output	**	# incremento # decremento	S(VARIABLE:-valor)	# si WARIABLE nula, retorna valor	patron2:] patron2N  # il WELFARLE coincide con patro "bloque2:] # entoncos bloque2 entoncos bloque2 entoncos bloque0efecto; # entoncos bloque0efecto;	# si VARIABLE coincide con patrone
2> fichero	# error a fichero			S(VARIABLE:=valor)	# si WARIABLE nula, asigna valor		
2>> fichero	# error al final del fichero	operadores comparaciones nume numero1 -eq numero2	# numerol iqual que numero2	\$[VARIABLE:Pmensaje]	W Si WARIABLE nula, mensaje error y fin		# entonces bloqueDefecto
2> /dev/null	# descarta error	numero1 -me numero2	# numeroi igual que numeroi # numeroi distinto que numeroi	\$[VARIABLE:inicio]	W recorta desde inicio hasta el final	esac	
output y error	In case of the control of	numero1 -Ne numero2		<pre>\${VARIABLE:inicio:longitud}</pre>	# recorta desde inicio hasta longitud	for VARIABLE in LISTA; do bloque	# iterativa com lista # ejecuta bloque sustituyendo
2>41   tee fichero 4> fichero	# ambos a fichero y a pantalla	numero1 -16 numero2	# numeroi menor que numero2 # numeroi menor o igual que numero2	S[iprefijo*]	W nombres de variables con prefijo	done	# VARIABLE por cada elemento de LIST
App Fighers	# ambos a fichero	numero1 -de numero2 numero1 -gt numero2	# numerol mayor que numerol	S(EVARIABLE)	if número de caracteres de VARDABLE	for ({expr1; expr2; expr3; }); do bloque dome	# iterativa con contador # primero se evalúa expl # luego mientras exp2 sea cierta
	# ambos al final del fichero	numero1 -ge numero2	# numerol mayor que numerol # numerol mayor o igual que numerol	\$(mrssec.[.])	W elementos de ARRAY		
VARIABLES		operadores lógicos	Financia major o agust que nancia	S[VARIABLESpatron]	W elimina minimo patrôn desde inicio		# se ejecutan el bloque y expr3
variables de entorno	# directorio de trabajo actual	operatores logicos	# NOT	\$[VARIABLEWpatron]	W elimina máximo patrôn desde inicio	while expresion; do	# bucle "mientras"
502740	# directorio de trabajo actual	46 , -a	# AND	\$(VARIABLENpatron)	V elimina minimo patrôn desde fin	bloque done until expresion; do	# se ejecuta bloque # mientras expresión sea cierta # bucle "hasta" # se ejecuta bloque
9910	# identificador del proceso padre	110	# OR	\${VARIABLENKpatron}	W elimina máximo patrôn desde fin		
SHOS THURS	E nombre del ordenador	operadores de ficheros		\$[VARIABLE/patron/reemplazo]	W reemplaza primera coincidencia	expression	
SUSER	# nombre del usuario	-e fichero	# existe	\$[VARIABLE//patron/reomplaze]	W reemplaza todas las coincidencias	[function] expression [] = # function   selection   se	# hasta que expresión sea cierta
SHORE	# directorio del usuario	-s fichero	# no està vacio	\$((expresion))	W sustituye expresión por su valor		
SPATH	g rutas bisqueda de conandos	-f fichero	# normal	S[expresion]	W sustituye expresion por su valor		# nombreFunción [parani parani]
SLANG	# idioma para los mensates	-d fichero	# directorio	EJECUCIÓN	F		
SFUNCIAME	# nombre función en ejecución	-h fichero	# enlace simbolico	./comando SRUTA/comando	# ejecuta desde directorio actual # ejecuta desde cualquier sitio		N imput
SLINENO	# número de linea actual (del script)	-r fichero	# permiso de lectura	conando	W e jecuta desde cualquier sitio		# puede nostrarse un mensaje antes
SRANDOM	# número alestorio	-w fichero	# permiso de escritura	script	W elecuta si esta en el sexim W elecuta exportando variables		# si minguna variable, REPLY = todo
variables especiales		-x fichero	# permiso de ejecución	S(comando parant paranN)	W ejecuta exportando variables W ejecuta de forma literal	echo cadena -m no hace salto de linea	# manda el valor de la cadena
50	# nombre del script	-0 fichero	# propietario	'conando parant paranti	# ejecuta sustituyendo variables	-e interpreta caracteres con \	# manda el valor de la cadena # a la salida estándar
S(N)	# parámetro N	-6 fichero	# pertenece al grupo	comando &	# ejecuta sustitujendo variables # ejecuta en segundo plano	print f	# output formateado (igual que C)
95	# identificador del proceso actual	f1 -ef f2	# f1 y f2 enlaces mismo archivo	c1   c2		CONTROL DE PROCESOS	
\$1	# identificador del último proceso	f1 -mt f2	# f1 mas nuevo que f2	a ; a	# redirige salida c1 a entrada c2 # ejecuta c1 y luego c2	conando A	N ejecuta en segundo plano
Se S	# todos los parâmetros recibidos	f1 -et f2	# fi más antiguo que f2	G 44 G	# ejecuta c1 y laego c2 # ejecuta c2 s1 c1 termina sin errores	bg númeroProceso	# continúa ejecución en segundo plan
Sir	# número de parâmetros recibidos	operadores de cadenas		d 11 d	# ejecuta c2 si ci termina sin errores	fg númeroProceso	# continúa ejecución en primer plano
\$7 # (G:normal, >Grerror)	# código de retorno del último comando	-n cadena	# no vacia	ARGUMENTOS DE LÍNEA DE COMANDOS	le element et se et matte con sum es	jobs	# muest ra procesos en ejecución
shift	# \$1=\$2, \$2=\$3, \$(N-1)=\$(N)	-z cadena	# vacia	while getopts "hs:" option ; do	= getops + "opciones disponibles"	kill sefal PIDI nümer dProceso1	# mata proceso(s) indicado(s)
ARRAYS		cadena1 = cadena2	# cadena1 igual a cadena2	case "Soption" in	W mientras haya argumentos	exit código	# salir con còdigo de retorno
declare -a ARRAY	# declaración array	cadena1 == cadena2	# cadenai igual a cadena2	h) DO_MELP=1 ;; s) argument=80PTARG ; DO_SEARCH=1 ;;	W selectionamos W -h sin optiones		# (6=normal, >6=error)
ARRAYm[valor1 valorN]	# asignación compuesta	cadena1 != cadena2	# cadenai distinta a cadena2	") echo "Invalid" ; return ;;	# -s can opciones en \$OPTARG	trap [comando] [código1]	# ejecuta comando cuando señal(es)
ARRAY[8] =valorN	# asignación simple			dose	" error	wait [PID1]nimeroProceso1]	# espera hasta fin proceso(s) hijo(s
ARRAY#([N]#valorN valorM [P]#valorP)	# asigna celdas N, M y P			-		mice -n prioridad comando remice -n prioridad comando	# ejecuta comando con prioridad [-20 # modifica prioridad comando [-20/19
S(ARRAY[N])	# valor celds N					remark on priceriosas continuo	# -20 maxima prioridad y 19 minima
\${ARRAY[*]} Autor: Adolfo Seno De Chego ( <u>augustiona</u> - <u>Ston) Schill</u>	# todos los valores (Lamedo/Lime/Dary/Taller) Limenia (7.57-5)	Asteri Addite Sanz be Singe ( <u>populátes</u> ) - <u>Q</u>	ing Strate Laboration Classics Course (C. St. St.	Autor: Addito Sens Se Siege ( <u>sourchings</u> - <u>Sing Cittle)</u>	LiderStr(StateStare(Suitter) Literatus SC-20-13	Autor: Adulfo Serv De Diego ( <u>averdiego</u> - <u>Singl Sto</u>	had Linkelin ( Linkeline ( Tal Her) Lineville )

chuleta-shell-script.pdf

#### 3.13 SHELLCHECK



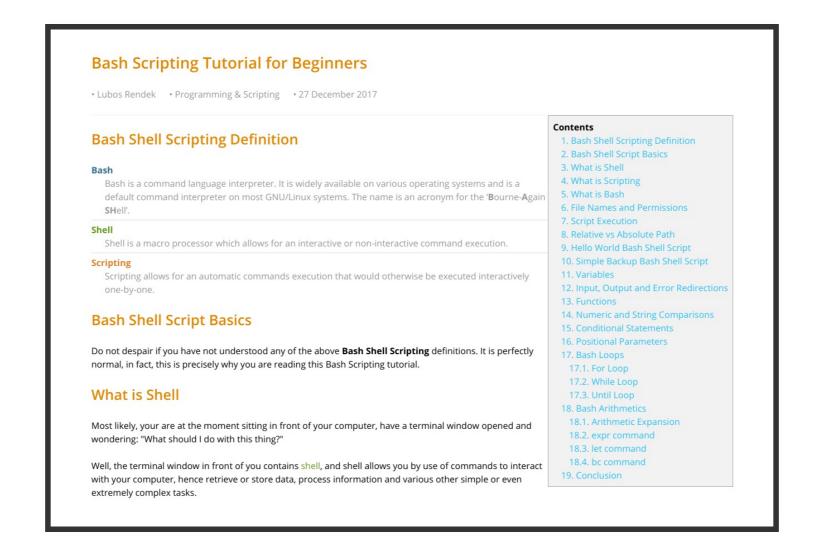
https://github.com/koalaman/shellcheck

#### 3.14 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
Shell Files and Interpreter	File Extensions SUID/SGID
Invocation Environment	STDOUT vs STDERR
Comments	File Header Function Comments Implementation Comments TODO Comments
Farmattina	Indentation Line Length and Long Strings Pipelines Loops Case statement Variable expansion Quoting
Formatting	Command Substitution Test, [ and [ Testing Strings Wildcard Expansion of Filenames Eval Pipes to While
Features and Bugs	
	Function Names Variable Names Constants and Environment Variable Names Source Filenames Read-only Variables Use Local Variables Function Location main
Features and Bugs	

https://google.github.io/styleguide/shell.xml

#### 3.15 TUTORIAL



https://linuxconfig.org/bash-scripting-tutorial-for-beginners

## 4 AWK

#### 4.1 INTRODUCCIÓN

- Es una hoja de cálculo por línea de comandos.
- Tiene su propio penguaje 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 dificil 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\_program' 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
```

ejemplos/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
```

ejemplos/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]
    }
}
done
```

ejemplos/07\_roles\_con\_awk.sh

#### 4.7 TUTORIAL

Grymoire Navigation	<u>Awk</u>
Navigation	
Unix/Linux	
Quotes	
Bourne Shell	
C Shell	
File Permissions	Last modified: Thu Apr 23 16:37:47 EDT 2015
Regular Expressions	
awk worth	Part of the Unix tutorials And then there's My blog
sed UPDATED	
find	Table of Contents
tar	
inodes	Why learn AWK?
Security	Basic Structure
IPv6	Executing an AWK script
Wireless	Which shell to use with AWK?
Hardware	Dynamic Variables
spam	The Essential Syntax of AWK
Deception	Arithmetic Expressions Summary of AWK Commands
PostScript Halftones	Summary or Awc Commands AWK Built-in Variables
Privacy	Associative Arrays
Bill of Rights	Picture Perfect PRINTF Output
References	Flow Control with next and exit
Top 10 reasons to avoid CSH	AWK Numerical Functions
sed Chart PDF	String Functions
awk Reference HTML	User Defined Functions
Magic	AWK patterns
Search	Formatting AWK programs
About	Environment Variables AWK, NAWK, GAWK, or PERL
Donate KW	AWK, NAWK, GAWK, OF PERL
Google+: Bruce Barnett	Copyright 1994,1995 Bruce Barnett and General Electric Company
Twitter: @grymoire	Copyright 2001, 2004, 2013, 2014 Bruce Barnett
Blog: Wordpress Blog	All rights reserved

http://www.grymoire.com/Unix/Awk.html

## 5 ACERCA DE

#### 5.1 LICENCIA

- Creative Commons Reconocimiento-Compartir Igual 3.0
  - http://creativecommons.org/licenses/by-sa/3.0/es/

#### 5.2 FUENTES

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

#### 5.3 SLIDES

- Las slides están hechas con MarkdownSlides
  - https://github.com/markdownslides/markdownslides

## 6 PREGUNTAS

## 7 GRACIAS