2. Agile SysAdmin: Networking and Systems Administration

Instructor: Julián García-Sotoca Pascual

Scripting

Índice

- Editor Vi
- Shell scripting con Bash
  - Introducción
  - Uso de variables e inputs
  - Control de flujo
  - Debug
  - Funciones
- Python
  - uso de python para administración de sistemas
- Vista rápida a Powershell

Índice

- **Editor Vi**
- Shell scripting con Bash
  - Introducción
  - Uso de variables e inputs
  - Control de flujo
  - Debug
  - Funciones
- Python
  - uso de python para administración de sistemas
- Vista rápida a Powershell

Edición básica de ficheros con vi

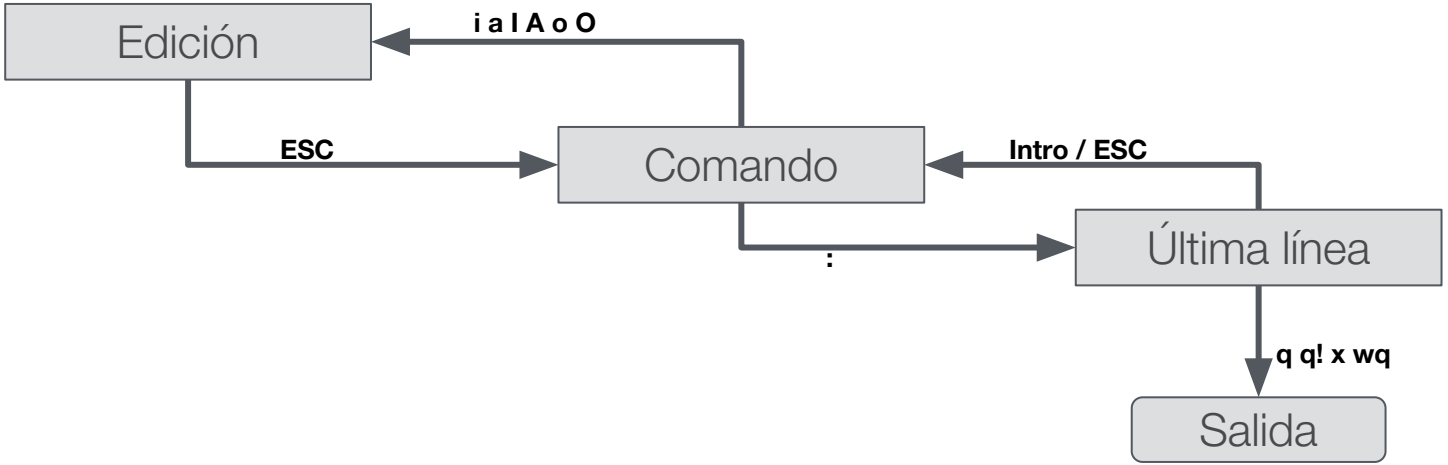
- VI es un editor de texto disponible en todas las distribuciones GNU/Linux y en la mayoría de Unix
- Proporciona muchas funcionalidades que lo hacen ser uno de los más usados por los administradores
- Realmente cuando se portó de los antiguos Unix al proyecto GNU se añadieron funcionalidades, creando el *vi improved* o *vim*
- Existen muchos otros como emacs o nano, pero Vi se ha convertido en el estándar
- En ciertas situaciones de emergencia es el único editor disponible

Edición básica de ficheros con vi

- Modos de operación:
  - **Comando**: acepta comandos en forma de letras
  - **Edición** o **Insert**: cualquier caracter introducido es insertado en el documento a excepción de la tecla esc
  - **Ex** o **Última línea**: precedido por : permite la manipulación de ficheros
- Es importante reconocer el modo en que estamos operando
- El editor arranca en modo comando

Edición básica de ficheros con vi

- Modos de operación:



Edición básica de ficheros con vi

- Formas de iniciar:
  - *vi*: abre el programa sin abrir ningún archivo
  - *vi fichero*: edita el fichero si existe y si no lo crea
  - *vi fichero1 fichero2*: edita varios archivos, para cambiar al siguiente archivo *:next* o *:n* y para el previo *:prev* o *:N*
  - *vi +[número] fichero*: edita el fichero iniciando en la línea indicada
  - *vi +/patrón fichero*: edita el fichero iniciando en la primera vez que encuentre el patrón

Edición básica de ficheros con vi

- Guía básica:

vi fichero	arranca en modo comando editando el fichero indicado
i	inserta texto a la izquierda del cursor
a	inserta texto a la derecha del cursor
I	Inserta texto al inicio de la línea
A	inserta texto al final de la línea
ESC	Vuelve al modo comando
X	borra el caracter bajo el cursor
dd	borra la línea actual
dw	borra la palabra actual
h o flecha izquierda	mueve el cursor a la izquierda
j o flecha abajo	mueve el cursor a la línea de abajo
k o flecha arriba	mueve el cursor a la línea de arriba
l o flecha derecha	mueve el cursor a la derecha
:w	guarda los cambios
:q	sale del editor

Edición básica de ficheros con vi

- Modo comando:
  - Multiplicadores: permite ejecutar un comando tantas veces como se le indica

5Y	copia 5 líneas
10dd	borra 10 líneas
3dw	borra 3 palabras
8j	mueve el cursor 8 líneas abajo

Edición básica de ficheros con vi

- Modo comando:
  - Movimiento del cursor

Flechas	mover en distintas direcciones
^ o $	mueve al inicio o al final de la línea
G	última línea
xG	mueve el cursor a la línea x
xl	mueve el cursor al carácter x de la línea

  - Movimiento de pantalla

Ctrl+f	una pantalla adelante
Ctrl+b	una pantalla atrás
Ctrl+d	media pantalla adelante
Ctrl+u	media pantalla atrás

Edición básica de ficheros con vi

- Modo comando:
  - Borrado

x	borra un carácter
dd	borra la línea
D o d$	borra desde el cursor hasta fin de línea
d0	borra desde el cursor hasta el inicio de línea
dw	borra desde el cursor hasta fin de la palabra

  - Copiar y pegar

Y o yy	copia línea
P	pega antes del cursor
p	pega después del cursor
yw	copia palabra

Edición básica de ficheros con vi

- Modo comando:
  - Búsqueda

/cadena	busca adelante la cadena
?cadena	busca atrás la cadena
n	repite el último comando de búsqueda
N	repite el último comando de búsqueda en sentido inverso
dw	borra desde el cursor hasta fin de la palabra

  - Otros

u	deshace la última acción
U	deshace todos los cambios en una línea
.	repite la acción

Edición básica de ficheros con vi

- Modo última línea:

:q	salir si no ha habido cambios
:q!	salir sin guardar
:w	guardar cambios
:w fichero1	guardar cambios en el fichero1
:wq o :x	guardar y salir
:r arch2	inserta un archivo
:e arch2	edita un archivo nuevo
:r! comando	inserta la salida de un comando

Edición básica de ficheros con vi

- Modo última línea:
  - configuración

:set number	muestra los números de línea
:set nonumber	oculta los números de línea
:set showmode	muestra el modo actual de vi
:set noshowmode	oculta el modo actual de vi
:set list	muestra caracteres ocultos
:set nolist	no muestra caracteres ocultos
:set ignorecase	ignora mayúsculas en las búsquedas
:set noignorecase	no ignora mayúsculas en las búsquedas

Se puede modificar permanentemente añadiendo las opciones en el fichero .vimrc

Editor VI

Vim Cheat Sheet

Global

`:help keyword` - open help for keyword

`:saveas file` - save file as

`:close` - close current pane

`K` - open man page for word under the cursor

Cursor movement

`h` - move cursor left

`j` - move cursor down

`k` - move cursor up

`l` - move cursor right

`H` - move to top of screen

`M` - move to middle of screen

`L` - move to bottom of screen

`w` - jump forwards to the start of a word

`W` - jump forwards to the start of a word (words can contain punctuation)

`e` - jump forwards to the end of a word

`E` - jump forwards to the end of a word (words can contain punctuation)

`b` - jump backwards to the start of a word

`B` - jump backwards to the start of a word (words can contain punctuation)

Editing

`r` - replace a single character

`J` - join line below to the current one with one space in between

`gJ` - join line below to the current one without space in between

`gwip` - reflow paragraph

`cc` - change (replace) entire line

`C` - change (replace) to the end of the line

`c$` - change (replace) to the end of the line

`ciw` - change (replace) entire word

`cw` - change (replace) to the end of the word

`s` - delete character and substitute text

`S` - delete line and substitute text (same as cc)

`xp` - transpose two letters (delete and paste)

`u` - undo

`Ctrl` + `r` - redo

`.` - repeat last command

Marking text (visual mode)

`v` - start visual mode, mark lines, then do a command (like y-yank)

`V` - start linewise visual mode

Cut and paste

`yy` - yank (copy) a line

`2yy` - yank (copy) 2 lines

`yw` - yank (copy) the characters of the word from the cursor position to the start of the next word

`y$` - yank (copy) to end of line

`p` - put (paste) the clipboard after cursor

`P` - put (paste) before cursor

`dd` - delete (cut) a line

`2dd` - delete (cut) 2 lines

`dw` - delete (cut) the characters of the word from the cursor position to the start of the next word

`D` - delete (cut) to the end of the line

`d$` - delete (cut) to the end of the line

`x` - delete (cut) character

Exiting

`:w` - write (save) the file, but don't exit

`:w !sudo tee %` - write out the current file using sudo

`:wq` or `:x` or `ZZ` - write (save) and quit

`:q` - quit (fails if there are unsaved changes)

`:q!` or `ZQ` - quit and throw away unsaved changes

Editor VI

`%` - move to matching character (default supported pairs: '()', '{}', '[]' - use `:h matchpairs` in vim for more info)

`0` - jump to the start of the line

`^` - jump to the first non-blank character of the line

`$` - jump to the end of the line

`g_` - jump to the last non-blank character of the line

`gg` - go to the first line of the document

`G` - go to the last line of the document

`5G` - go to line 5

`fx` - jump to next occurrence of character x

`tx` - jump to before next occurrence of character x

`Fx` - jump to previous occurence of character x

`Tx` - jump to after previous occurence of character x

`;` - repeat previous f, t, F or T movement

`,` - repeat previous f, t, F or T movement, backwards

`}` - jump to next paragraph (or function/block, when editing code)

`{` - jump to previous paragraph (or function/block, when editing code)

`zz` - center cursor on screen

`Ctrl` + `e` - move screen down one line (without moving cursor)

`Ctrl` + `y` - move screen up one line (without moving cursor)

`o` - move to other end of marked area

`Ctrl` + `v` - start visual block mode

`O` - move to other corner of block

`aw` - mark a word

`ab` - a block with ()

`aB` - a block with {}

`ib` - inner block with ()

`iB` - inner block with {}

`Esc` - exit visual mode

Visual commands

`>` - shift text right

`<` - shift text left

`y` - yank (copy) marked text

`d` - delete marked text

`~` - switch case

Registers

`:reg` - show registers content

`"xy` - yank into register x

`"xp` - paste contents of register x

Tip Registers are being stored in ~/.viminfo, and will be loaded again on next restart of vim.

`:wqa` - write (save) and quit on all tabs

Search and replace

`/pattern` - search for pattern

`?pattern` - search backward for pattern

`\vpattern` - 'very magic' pattern: non-alphanumeric characters are interpreted as special regex symbols (no escaping needed)

`n` - repeat search in same direction

`N` - repeat search in opposite direction

`:%s/old/new/g` - replace all old with new throughout file

`:%s/old/new/gc` - replace all old with new throughout file with confirmations

`:noh` - remove highlighting of search matches

Search in multiple files

`:vimgrep /pattern/ {file}` - search for pattern in multiple files

e.g. `:vimgrep /foo/ **/*`

`:cn` - jump to the next match

`:cp` - jump to the previous match

`:copen` - open a window containing the list of matches

Editor VI

`Ctrl` + `b` - move back one full screen

`Ctrl` + `f` - move forward one full screen

`Ctrl` + `d` - move forward 1/2 a screen

`Ctrl` + `u` - move back 1/2 a screen

Tip Prefix a cursor movement command with a number to repeat it. For example, `4j` moves down 4 lines.

Insert mode - inserting/appending text

`i` - insert before the cursor

`I` - insert at the beginning of the line

`a` - insert (append) after the cursor

`A` - insert (append) at the end of the line

`o` - append (open) a new line below the current line

`O` - append (open) a new line above the current line

`ea` - insert (append) at the end of the word

`Esc` - exit insert mode

Tip Register 0 contains always the value of the last yank command.

Marks

`:marks` - list of marks

`ma` - set current position for mark A

`` `a `` - jump to position of mark A

`y` `a` - yank text to position of mark A

Macros

`qa` - record macro a

`q` - stop recording macro

`@a` - run macro a

`@@` - rerun last run macro

Editor VI

Working with multiple files

- `:e file` - edit a file in a new buffer
- `:bnext` or `:bn` - go to the next buffer
- `:bprev` or `:bp` - go to the previous buffer
- `:bd` - delete a buffer (close a file)
- `:ls` - list all open buffers
- `:sp file` - open a file in a new buffer and split window
- `:vsp file` - open a file in a new buffer and vertically split window
- `Ctrl` + `ws` - split window
- `Ctrl` + `ww` - switch windows
- `Ctrl` + `wq` - quit a window
- `Ctrl` + `wv` - split window vertically
- `Ctrl` + `wh` - move cursor to the left window (vertical split)
- `Ctrl` + `wl` - move cursor to the right window (vertical split)
- `Ctrl` + `wj` - move cursor to the window below (horizontal split)
- `Ctrl` + `wk` - move cursor to the window above (horizontal split)

Tabs

- `:tabnew` or `:tabnew file` - open a file in a new tab
- `Ctrl` + `wT` - move the current split window into its own tab
- `gt` or `:tabnext` or `:tabn` - move to the next tab
- `gT` or `:tabprev` or `:tabp` - move to the previous tab
- `#gt` - move to tab number #
- `:tabmove #` - move current tab to the #th position (indexed from 0)
- `:tabclose` or `:tabc` - close the current tab and all its windows
- `:tabonly` or `:tabo` - close all tabs except for the current one
- `:tabdo` command - run the `command` on all tabs (e.g. `:tabdo q` - closes all opened tabs)

Índice

- Editor Vi
- **Shell scripting con Bash**
  - Introducción
  - Uso de variables e inputs
  - Control de flujo
  - Debug
  - Funciones
- Python
  - uso de python para administración de sistemas
- Vista rápida a Powershell

Scripting - Introducción

- Shell Script
  - Lista de comandos que se ejecutan secuencialmente
  - Permiten añadir cierta lógica para ejecutar ciertas partes en función de algunas condiciones
- Elementos que todo script debería tener:
  - shebang → primera línea iniciada por #! indicando que shell debe ser usada
  - comentarios → como buena práctica se deberían añadir comentarios de qué hace el script, como se debe ejecutar y el autor.
  - Declaración de salida → con exit se indica si la ejecución ha sido exitosa o no

Scripting - Variables e input

- Argumentos:
  - cualquier cosa que se ponga tras el script en la ejecución
  - $1 hace referencia al primer argumento, $2 segundo argumento, etc
  - $0 es el nombre del script
  - $@ devuelve todos los argumentos en un array
  - $# devuelve el número de argumentos

Scripting - Variables e input

- Argumentos:

```
#!/bin/bash

# Use $0 to guet the name of the script
echo "This is the name of the script: $0"

# Use $1 to get the first argument:
echo "This is the first argument: $1"

echo "The script has $# arguments"

echo "All the arguments are: $@"
```

```
This is the name of the script: ./first script.sh
This is the first argument: one
The script has 3 arguments
All the arguments are: one two three
```

Scripting - Variables e input

- Variables:
  - etiqueta para referenciar una ubicación específica de memoria que contiene un valor
  - Se pueden definir estáticamente NOMBRE=valor
  - o dinámicamente:
    - con read se pide al usuario que introduzca el valor
    - como resultado de un comando, p.e.:

      HOY=$(date +%d-%m-%y)
- Se referencian con el símbolo de $: $NOMBRE o ${NOMBRE}

Scripting - Variables e input

- Variables especiales:
  - $? → contiene el valor devuelto por el último comando ejecutado. 0 suele ser ejecución correcta
  - $$ → contiene el valor del PID de la shell actual
  - $! → contiene el PID del último proceso pasado a Background

Scripting - Variables e input

- Arrays:
  - Se pueden definir directamente con el índice:

```
:~$ foo[0]="first"
:~$ foo[1]="second"
```

  - O inicializar un array completo:

```
:~$ foo=("first" "second" "third")
```

  - Se acceden a ellos mediante el índice o con @ para devolver el array completo:

```
:~$ echo ${foo[1]}
second
:~$ echo ${foo[@]}
first second third
```

  - Obtener el número de elementos de un array:

```
:~$ echo ${#foo[@]}
3
```

Scripting - Comparaciones

- numérica: -eq, -ge, -gt, -le, -lt, -ne
- string: =, !=, -n, -z
- ficheros: -d, -f, -r, -s, -w, -x
- booleanos: !, -a, -o

Expression	Description
-d *file*	True if *file* is a directory.
-e *file*	True if *file* exists.
-f *file*	True if *file* exists and is a regular file.
-L *file*	True if *file* is a symbolic link.
-r *file*	True if *file* is a file readable by you.
-w *file*	True if *file* is a file writable by you.
-x *file*	True if *file* is a file executable by you.
file1 -nt *file2*	True if *file1* is newer than (according to modification time) *file2*.
file1 -ot *file2*	True if *file1* is older than *file2*.
-z *string*	True if *string* is empty.
-n *string*	True if *string* is not empty.
string1 = *string2*	True if *string1* equals *string2*.
string1 != *string2*	True if *string1* does not equal *string2*.

Scripting - Control de flujo

- if .. then .. else

```
#!/bin/bash
# run this script with one argument
# the goal is to find out if the argument is a file or a directory
if [ -f $1 ]
then
     echo "$1 is a file"
elif [ -d $1 ]
then
     echo "$1 is a directory"
else
     echo "I do not know what \$1 is"
fi
exit 0
```

Scripting - Control de flujo

- A veces no es necesario una declaración completa y se puede sustituir en una línea

```
[ -z $1 ] && echo no argument provided
ping -c 1 10.0.0.20 2>/dev/null || echo node is not available
```

- || es un OR lógico ejecutará la segunda parte solo si la primera no es verdadera
- && es un AND lógico y ejecutará la segunda parte si la primera es verdadera

Scripting - Control de flujo

- for

```
#!/bin/bash
#
for (( COUNTER=100; COUNTER>1; COUNTER-- )); do
        echo $COUNTER
done
exit 0
```

```
for i in {100..104}; do ping -c 1 192.168.4.$i >/dev/null && echo
192.168.4.$i is up; done
```

Scripting - Control de flujo

- while

```
#!/bin/bash
#
# usage: monitor <processname>
while ps aux | grep $1 | grep -v grep  > /dev/tty11
do
     sleep 5
done

clear
echo your process has stopped
logger $1 is no longer present
mail -s "process $1 has stopped" root < .
```

Scripting - Control de flujo

- until

```
#!/bin/bash
#
until users | grep $1 > /dev/null
do
     echo $1 is not logged in yet
     sleep 5
done
echo $1 has just logged in
mail -s "$1 has just logged in" root < .
```

Scripting - Control de flujo

- case

```
case "$1" in
  start)
          start;;
  stop)
          rm -f $lockfile
          stop;;
  restart)
          restart;;
  reload)
          reload;;
  status)
          status
          ;;
  *)
          echo "Usage: $0 (start|stop|restart|reload|status)"
          ;;
esac
```

Scripting - Debug

- Para debugear un script se puede lanzar con el parámetro -x

```
[root@server1 ~]# bash -x 319.sh
+ grep
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
+ users
+ echo is not logged in yet
is not logged in yet
+ sleep 5
```

También se puede especificar en el propio script con *set -x*
Otra opción interesante es *set -e* que hará que el script pare inmediatamente tras un error

Scripting - Funciones

- Funciones:
  - Parte de un script que realiza una subtarea
  - Se define de la siguiente forma:

```
myfn() {
    commands
}
```

  - Se puede usar la palabra clave *function* pero por portabilidad es preferible no usarlo

```
function name {
  commands
}
```

Índice

- Editor Vi
- Shell scripting con Bash
  - Introducción
  - Uso de variables e inputs
  - Control de flujo
  - Debug
  - Funciones
- **Python**
  - uso de python para administración de sistemas
- Vista rápida a Powershell

Scripting - Python

Ventajas:

- curva de aprendizaje
- código claro y fácilmente entendible
- control de errores
- orientado a objetos
- multitud de librerías e integraciones
- amplia comunidad
- estandarización
- potencia

Scripting - Python

- shebang: *#!/usr/bin/env python2, #!/usr/bin/env python3 o #!/usr/bin/env python*
- coding: *# -*- coding: utf-8 -*-*
- librerías útiles:
  - argparse: uso de argumentos
  - paramiko: conexión ssh
  - requests: enviar peticiones http
  - syslog: escribe en el syslog de la máquina
  - snmp: permite enviar consultas snmp
  - drivers de conexión a BBDD: pymssql, pymongo, psycopg2
  - elasticsearch: conexión a la API de elastic
  - sendmail: envio de correo

Scripting - Python

- argparse:

```
parser = argparse.ArgumentParser(description='Script para el analisis de logs de dispositivos de red')
parser.add_argument('-f','--file', help='Fichero a analizar',required=False)
parser.add_argument('-r','--hotel',help='Hotel por el que se quiere filtrar', required=False)
parser.add_argument('-d','--day',help='Dia que se quiere analizar', type=int, required=False, default=1)
parser.add_argument('-a','--all',help='Analiza todos los logs. Se puede filtrar por hotel', required=False, action='store_true')
parser.add_argument('-v','--verbose',help='Lanza el script en modo verbose', required=False, action='store_true')
args = parser.parse_args()
```

Scripting - Python

- sendmail:

```
def sendmail(text, h, subject):
    me = 'loganalizer@hotels.com'
    you = 'juliansotoca@hotels.com'
    msg = MIMEMultipart('alternative')
    msg['Subject'] = subject
    msg['From'] = me
    msg['To'] = you
    html= '''\
    <html>
    <head></head>
    <body> <h3>%s</h3><hr>
    <p>Numero total de errores registrados: %d</p>
    <p>Numero total de puertos flapeando: %d</p><hr>
    %s
    </body>
    </html>
    ''' % (subject, len(errList), len(flappingIF), h)
    part2 = MIMEText(html, 'html')
    msg.attach(part2)
    s = smtplib.SMTP('relay.hotels.red')
    s.sendmail(me, you, msg.as_string())
    s.quit()
```

Scripting - Python

- paramiko:

```
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
print "conectando a %s" % ap
try:
    ssh.connect(ap, username=USER, password=PASSWORD,timeout=2)
    shell = ssh.invoke_shell()
    time.sleep(1)
    res=shell.send('show interface eth0 | include MAC\n')
    time.sleep(1)
    output = shell.recv(9000)
    for l in output.splitlines():
        if l.startswith('MAC addr'):
            mac=l.split(';')[0].split('=')[1]
        if l.startswith('AH-'):
            apHostname=l.split('#')[0]
```

Scripting - Python

- syslog:

```
syslog.openlog(facility=syslog.LOG_LOCAL6)
syslog.syslog("DHCPACK on %s to %s (Unknown) via localhost" % (IP,MAC))
```

Scripting - Python

- snmp:

```
vendor=netsnmp.snmpget(netsnmp.Varbind("SNMPv2-MIB::sysDescr.0"), **parametros)[0]
```

```
hostname=netsnmp.snmpget(netsnmp.Varbind(".1.3.6.1.2.1.1.5.0"), **parametros)[0]
location=netsnmp.snmpget(netsnmp.Varbind(".1.3.6.1.2.1.1.6.0"), **parametros)[0]
```

Índice

- Editor Vi
- Shell scripting con Bash
  - Introducción
  - Uso de variables e inputs
  - Control de flujo
  - Debug
  - Funciones
- Python
  - uso de python para administración de sistemas
- **Vista rápida a Powershell**

¿Preguntas?

GRACIAS

www.keepcoding.io