educación IT
CODE YOUR FUTURE

Bootcamp DevOps Engineer

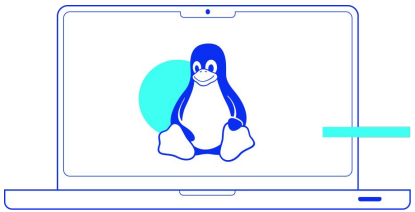
Fase 1 - SysAdmin

Módulo 1

Introducción

En las ***expresiones regulares*** se utilizan **cadenas de texto o patrones**. Estos patrones pueden ser de dos tipos:

- **Literales** (texto plano).
- **Metacaracteres** (símbolos con un significado especial).



Uso de Grep

El comando **grep** se utiliza para **buscar patrones en archivos o por STDIN**.

Sintaxis:

```
grep [opciones]  patrón archivo
```

El patrón puede ser una expresión regular.

Opciones:

`i`	No diferencia mayúsculas de minúsculas.
`c`	Cuenta la cantidad de coincidencias.
`v`	Muestra el resultado inverso.
`E`	Utiliza expresiones regulares extendidas.
`r`	Búsqueda recursiva.
`n`	Muestra el número de línea.
`A [numero]`	Muestra “número” de líneas después del patrón encontrado.
`B [numero]`	Muestra “número” de líneas antes del patrón encontrado.

Buscar la palabra “**error**” en el archivo *messages:*

```
# grep  error /var/log/messages
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
```

Contar la **cantidad de coincidencias** de la palabra “**error**” en el archivo *messages:*

```
# grep  -c error /var/log/messages
4
```

Buscar la palabra “**error**” en el archivo *messages* y marcarla con **color:**

```
# grep --color  error /var/log/messages
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
Aug 23 21:33:03 localhost NetworkManager[1216]:    ifcfg-rh:     error: Couldn't parse file
'/etc/sysconfig/network-scripts/ifcfg-lo;4c730325'
```

Contar la **cantidad de coincidencias** de la palabra "**error**" en todos los archivos que comienzan con "**a**":

```
[root@localhost ~]# grep  -c error /var/log/a*
/var/log/agns:0
/var/log/anaconda.log:1
/var/log/anaconda.program.log:0
/var/log/anaconda.storage.log:0
/var/log/anaconda.syslog:6
```

Contar la **cantidad de coincidencias** de la palabra "**error**" **sin distinguir mayúsculas de minúsculas:**

```
[root@localhost ~]# grep  -ci error /var/log/messages
5
```

Muestra solo lo que contenga la palabra "**root**":

```
# cat /etc/passwd|grep root
root:x:0:0:root:/root:/bin/bash
```

Muestra lo que no contenga la palabra "**bash**":

```
# cat /etc/passwd|grep -v bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
```

Busca la palabra "**link**" y muestra el **número de línea:**

```
# grep -n link /var/log/messages
3329:Aug 23 14:42:46 restauracion kernel: ADDRCONF(NETDEV_UP): eth0: link is not ready
3334:Aug 23 14:42:46 restauracion kernel: ADDRCONF(NETDEV_UP): ath0: link is not ready
3345:Aug 23 14:42:59 restauracion kernel: ADDRCONF(NETDEV_UP): peth0: link is not ready
3388:Aug 23 14:44:55 restauracion kernel: ADDRCONF(NETDEV_CHANGE): peth0: link becomes ready
```

Busca la palabra "**error**" en el directorio */var/log* y todos sus subdirectorios:

```
# grep -rn error /var/log/*
```

Busca la palabra "**DHCPACK**" y muestra **una línea antes y una después** del resultado:

```
# grep -A 1 -B 1 DHCPACK /var/log/messages
Nov 13 03:55:36 oc6127656113 dhclient[2991]: DHCPREQUEST on em1 to 9.0.132.30 port 67
Nov 13 03:55:36 oc6127656113 dhclient[2991]: DHCPACK from 9.0.132.30
Nov 13 03:55:36 oc6127656113 dhclient[2991]: bound to 9.6.162.69 -- renewal in 2880 seconds.
```

Veamos algunos ejemplos.

La contrabarra "\" es un carácter de "escape", esto significa que el carácter que este a continuación de este tendrá un significado en especial.

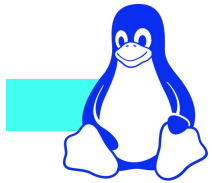
Expresiones regulares

Posicionales (position anchor)

^	Indica que es al inicio de la línea.
$	Al final de la línea.
\< \>	Exactamente lo que está entre \< \> Ejemplo: \< palabra\>

Buscar el patrón "**bin**" dentro del archivo */etc/passwd*

```
# grep 'bin' /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
(...salida cortada...)
```



Buscar la línea que comienza con “**bin**”:

```
# grep '^bin' /etc/passwd
bin:x:2:2:bin:/bin:/bin/sh
```

Buscar lo que finaliza con “**sh**”:

```
# grep 'sh$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

Buscar exactamente “**bin**”:

```
# grep '\<bin\>' /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

Mostrar todas las líneas no vacías:

```
# grep -v '^$' archivo.txt
```

Conjuntos de caracteres *(character sets)*

`[abc]`	Grupo de caracteres. Esto identifica una de esas tres letras.
`[a-z]`	Identifica cualquier letra de la **a** a la **z** en minúscula.
`[0-9]`	Identifica cualquier número.
`[^abc]`	Identifica cualquier letra que no sea **a, b** o **c** en minúscula.
`[^a-z]`	Identifica cualquier carácter que no sea de la **a** la **z** en minúscula.
`.`	Este signo es un comodín para cualquier carácter, excepto nueva línea.

Clases de caracteres de la expresión regular POSIX

`[:alnum:]`	Alfanumérico [a-zA-Z0-9]
`[:alpha:]`	Alfabético [a-zA-Z]
`[:blank:]`	Espacios o tabs
`[:cntrl:]`	Caracteres de control
`[:digit:]`	Dígitos numéricos[0-9]
`[:graph:]`	Cualquier carácter visible
`[:lower:]`	Minúsculas [a-z]
`[:print:]`	Caracteres que no son de control
`[:punct:]`	Caracteres de puntuación
`[:space:]`	Espacios en blanco
`[:upper:]`	Mayúsculas [A-Z]
`[:xdigit:]`	Dígitos hex [0-9a-fA-F]

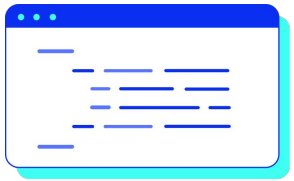
Ejemplos:

Los caracteres pueden ser situados en **grupos y rangos** para realizar expresiones regulares más eficientes.

```
[root@localhost ~]# cat prueba.txt
Villa Dalmine
Restaurador de Leyes de Argentina
JMR
Argentina
argentina
124Argentina
12JMR
12345JMR
evil5
```

Buscar "**Argentina**" o "**argentina**":

```
[root@localhost ~]# grep '[Aa]rgentina'
prueba.txt
Restaurador de Leyes de Argentina
Argentina
argentina
124Argentina
```



Buscar exactamente “**Argentina**” o “**argentina**”:

```
[root@localhost ~]# grep '\<[Aa]rgentina\>'
prueba.txt
Restaurador de Leyes de Argentina
Argentina
argentina
```

Buscar tres **números** consecutivos del **0** al **9**:

```
[root@localhost ~]# grep ‘[0-9][0-9][0-9]‘
prueba.txt
124Argentina
12345JMR
```

Buscar todo lo que **no comience con un número:**

```
[root@localhost ~]# grep ‘^[^0-9]‘ test
Villa Dalmine
Restaurador de Leyes de Argentina
JMR
Argentina
argentina
evil5
```

Modificadores *(Quantity modifiers)*

Existen dos tipos de expresiones regulares: básicas y extendidas.

- Las **expresiones regulares extendidas** consideran ciertos caracteres como especiales.
- En las **expresiones regulares básicas** para que dicho carácter tenga un sentido especial es necesario anteponer una contra barra, tal como se muestra a continuación:

Básicas	Extendidas	Descripción
`*`	`*`	Identifica 0 o más veces un único carácter
`\?`	`?`	Identifica 0 o una vez la expresión regular que antecede
`\+`	`+`	Identifica 1 o más veces la expresión regular que antecede
`\{n,m}`	`{n,m}`	Identifica un rango de ocurrencias (un carácter o una expresión regular) que antecede. Debe identificar al menos **n** hasta **m** ocurrencias
`|`	`|`	Identifica una u otra. Función lógica **OR**
`\(regex\)`	`(regex)`	Agrupa. Identifica grupo de expresiones regulares

Ejemplos:

```
# cat file1
ab
abc
abcc
abccc
asaabsde
ffda4
```

Buscar **ab**, y que pueda o no contener 1 o más **c**:

```
# grep ‘abc*’ file1
ab
abc
abcc
abccc
asaabsde

# cat file1
123456
abc12asf1245
a1gb1sd2
12456
```

Buscar al menos dos números:

```
# grep ‘[0-9][0-9][0-9]*’ file1
123456
abc12asf1245
12456
# cat prueba2.txt
abc
abcabc
abcabcabc
abcabcabcabc
```

Buscar exactamente **abc** y que aparezca sólo de 2 a 4 veces:

```
# grep --color '\<\(abc\)\{2,4\}\>' prueba2.txt
abcabc
abcabcabc
abcabcabcabc
```

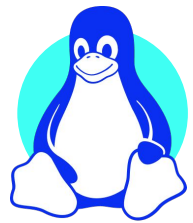
Buscar exactamente **abc** y que aparezca 2 o más veces:

```
# grep --color '\<\(abc\)\{2,\}\>' prueba2.txt
abcabc
abcabcabc
abcabcabcabc

# cat file1
file
file1
file2
file22
fil1
```

Buscar **file** que puede o no tener el número **1** o el **2**:

```
# grep 'file[12]\?' file1
file
file1
file2
file22
```



Buscar **file** que debe tener el número **1** o el **2** al menos una vez:

```
# grep 'file[12]\+' file1
file1
file2
file22

# cat file1
1
11
111
1111
11111
1010110
```

Buscar que comience con el número **1** y se repita entre 3 y 5 veces:

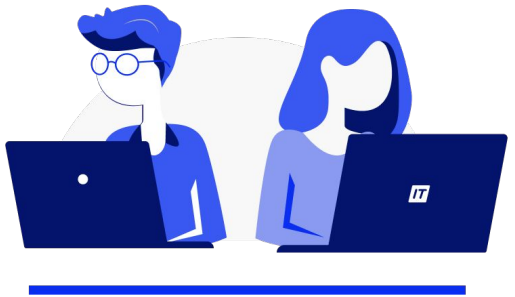
```
# grep '^1\{3,5\}$' file1
111
1111
11111
```

Otro ejemplo:

```
# cat file1
124
1
1234
12345
1234567
124567889
```

Buscar un número de **2 a 5 dígitos:**

```
# grep '\<[0-9]\{2,5\}\>' file1
12
1234
12345
124
```



¡Sigamos trabajando!