Mini-Curso de Expressões Regulares

II SDSL - Campinas, 12 de Dezembro de 2003

Aurélio Marinho Jargas
http://aurelio.net

Índice

Introdução

Expressões Regulares. Um assunto que muitos torcem o nariz ao ouvir falar, mas que sempre acaba aparecendo na resolução dos mais diversos problemas.

Para quem não conhece ou não domina o assunto, é difícil perceber a utilidade de saber escrever todos aqueles símbolos estranhos. Mas à medida que vai se aprendendo, aplicando, tudo começa a clarear.

Este mini−curso apresentará os componentes utilizados nas expressões e ensinará como utilizá−los.

Uma Expressão Regular (ER) é um método formal de se especificar um padrão de texto.

É uma composição de símbolos, caracteres com funções especiais, chamados "metacaracteres" que, agrupados entre si e com caracteres literais, formam uma seqüência, uma expressão. Essa expressão é testada em textos e retorna sucesso caso esse texto obedeça exatamente a todas as suas condições. Diz−se que o texto "casou" com a expressão.

A ERs servem para se dizer algo abrangente de forma específica. Definido o padrão de busca, tem−se uma lista (finita ou não) de possibilidades de casamento. Em um exemplo rápido, `[rgp]ato` pode casar "rato", "gato" e "pato".

As ERs são úteis para buscar ou validar textos variáveis como:

- data
- horário
- número IP
- endereço de e−mail
- endereço de Internet
- declaração de uma função()
- dados na coluna N de um texto
- dados que estão entre <tags></tags>
- número de telefone, RG, CPF, cartão de crédito

Vários editores de texto e linguagens de programação têm suporte a ERs, então o tempo investido em seu aprendizado é recompensado pela larga variedade de aplicativos onde ele pode ser praticado.

O ambiente de testes

Para conhecer e desvendar os segredos das Expressões Regulares, usaremos o comando `grep`, que serve para "pescar" linhas dentro de um texto. Você informa uma palavra e o `grep` retorna todas as linhas do texto em que ele encontrar essa palavra.

Conhecendo o arquivo /etc/passwd

O arquivo de dados que usaremos nos testes será o `/etc/passwd`, a base de usuários de um sistema UNIX/Linux. Então, primeiro vamos conhecer o que tem dentro desse tal passwd:

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

São várias linhas, uma para cada usuário. O separador de informações é o caractere dois−pontos ":" e cada linha possui sete campos, no seguinte formato:

```
login : senha : UID : GID : Nome Completo : Diretório $HOME : shell
```

Todas as senhas estão protegidas em outro arquivo, por isso há somente uma letra "x" em seu lugar. Analisando a primeira linha, sabemos que ela é referente ao administrador do sistema ("root"). Seu número de usuário (UID) e seu número de grupo são iguais: zero. Seu nome é "root" mesmo, seu diretório padrão é o "/root" e seu shell de login é "bash".

Conhecendo o comando grep

Para não precisar listar o conteúdo do arquivo todo para saber os dados do usuário "root", podemos usar o `grep` para pesquisar e nos retornar somente a linha dele. O formato do comando grep é o seguinte:

```
grep PALAVRA ARQUIVO
```

É simples, basta passar a palavra desejada e o nome do arquivo aonde procurar por ela. Como queremos "pescar" somente a linha do usuário root, basta fazer:

```
$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
$
```

Epa! Além da linha do root, o grep retornou também a linha do usuário "operator". Mas por que isso se a pesquisa era somente por "root"?

Olhando mais atentamente para a linha do usuário operator, vemos que o seu diretório $HOME é "/root". Como dissemos ao grep "*procure por root*", as duas linhas são resultados válidos.

Para obter somente a linha desejada, seria preciso haver uma maneira de dizer ao grep "*procure por root no início da linha*". É neste momento que as Expressões Regulares entram em cena.

Além de palavras, o grep também entende ERs. Uma expressão é formada por caracteres normais que formam palavras (letras e números) como "root", mas também por símbolos especais, os chamados metacaracteres.

Conhecendo cada um dos metacaracteres

Cada metacaractere é uma ferramenta que tem uma função específica. Eles servem para dar mais poder às pesquisas, informando padrões e posições impossíveis de se especificar usando somente caracteres normais.

Os metacaracteres são pequenos pedacinhos simples que agrupados entre si, ou com caracteres normais, formam algo maior, uma expressão. O importante é compreender bem cada um individualmente, e depois apenas lê−los em seqüência.

O circunflexo ^

O primeiro metacaractere que veremos é o circunflexo "^" (ou chapéuzinho), que simboliza o início de uma linha. Podemos usá−lo para resolver o problema anterior de obter somente a linha do usuário root:

```
$ grep ^root /etc/passwd
root:x:0:0:root:/root:/bin/bash
$
```

Funcionou! Reveja o que acabamos de fazer: passamos ao grep uma Expressão Regular composta pelo metacaractere circunflexo, seguido da palavra "root".

Não confunda, não procuramos pelo texto "^root", mas sim pela palavra "root" no início da linha. Que diferença faz um simples ^ hein?

O cifrão $

O circunflexo é chamado de metacaractere de posicionamento, pois representa uma posição específica da linha. Seu primo é o cifrão "$", que representa o fim de uma linha.

Lembra que o último campo de cada linha do arquivo "passwd" é o shell de login do usuário? Agora com o metacaractere cifrão podemos pesquisar todos os usuários, além do root, que usam o "bash" como shell:

```
$ grep bash$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

A Expressão Regular "`bash$`" procura pela palavra "bash" no final da linha. Ou ainda, a palavra "bash" seguida de um fim de linha.

Esse cifrão é o mesmo caractere que é utilizado para identificar as variáveis do shell, como `$PWD` e `$HOME`. Para evitar possíveis problemas com a expansão de variáveis, é preciso "proteger" a Expressão Regular passada ao grep. A proteção é feita se colocando a ER entre 'aspas simples'. Isso evitará muitas dores de cabeça.

Dica nº1: No shell, sempre proteja as ERs com 'aspas simples'

O comando anterior, agora 'aspeado', fica:

```
$ grep 'bash$' /etc/passwd
```

Juntando os dois metacaracteres já vistos, temos na mão uma ER muito útil que serve para encontrar linhas em branco:

```
^$
```

O que é uma linha em branco senão um começo de linha seguido de um fim de linha? Guarde essa ER num cantinho especial de sua memória, pois é comum precisar dela nos problemas do dia a dia.

Mas como nem só de começo e fim de linha são compostos os problemas, precisamos conhecer os outros metacaracteres disponíveis.

A lista []

Mudando um pouco de tarefa, que tal pesquisar os usuários que possuem o nome "Carlos"? No arquivo de exemplo há dois, o "carlos" e o "acs". Vamos tentar obter ambos com o grep:

```
$ grep 'carlos' /etc/passwd
carlos:x:500:500:carlos:/home/carlos:/bin/bash
$ grep 'Carlos' /etc/passwd
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Puxa vida... Um está em minúsculas e o outro com a primeira letra maiúscula, sendo preciso usar o grep duas vezes para extraí−los. Para resolver este problema com apenas um comando, novamente precisaremos dos poderes das Expressões Regulares!

O terceiro metacaractere que veremos é a "lista". Basta colocar entre colchetes "`[]`" todos os caracteres que podem aparecer em uma determinada posição. Podem ser tantos quantos se precise, mas em nosso caso apenas dois bastam, "C" e "c":

```
$ grep '[Cc]arlos' /etc/passwd
carlos:x:500:500:carlos:/home/carlos:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Repare bem no detalhe, toda a "lista" (os colchetes e seu conteúdo) vale para **apenas uma posição**, um caractere. A ER "`[Cc]arlos`" serve para pesquisar por "Carlos" e "carlos" ao mesmo tempo.

Dica nº2: A lista, por maior que seja, representa apenas um caractere

Podemos combinar a lista com um dos metacaracteres já vistos, produzindo uma ER mais poderosa:

```
$ grep '^[aeiou]' /etc/passwd
adm:x:3:4:adm:/var/adm:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

```
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Este comando procura por usuários cujo login comece com uma vogal! Note o uso do circunflexo para "amarrar" a pesquisa no início da linha. Traduzindo essa ER, fica: busque por linhas que começam com a letra "a", ou "e", ou "i" ou "o" ou "u".

Para fazer o inverso e buscar as linhas que começam com consoantes, a ER fica bem mais comprida, pois é necessário especificar todas as letras:

```
$ grep '^[bcdfghjklmnpqrstvwxyz]' /etc/passwd
```

O ponto .

Às vezes, é necessário permitir "qualquer" caractere numa certa posição. Por exemplo, para procurar usuários onde a segunda letra do login seja uma vogal, como "mario", e "jose", mas não "ana". Não importa qual é a primeira letra, pode ser qualquer uma, mas a segunda deve ser uma vogal.

O ponto é o metacaractere que significa "qualquer letra". E mais, na verdade ele significa "qualquer caractere", pois além de letras ele pode representar um número, um símbolo, um TAB, enfim: qualquer caractere.

```
$ grep '^.[aeiou]' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
carlos:x:500:500:carlos:/home/carlos:/bin/bash
$
```

Esta expressão diz: "À partir do começo da linha, procure qualquer caractere seguido de uma vogal".

O ponto também pode ser usado para se procurar por linhas de um tamanho fixo, independente do seu conteúdo. Que tal se obter apenas as linhas que possuam exatamente 27 caracteres?

```
$ grep '^...........................$' /etc/passwd
news:x:9:13:news:/etc/news:
$
```

Será que ali tem 27 pontos mesmo? Como saber ao certo? Estranho isso de ficar colocando vários pontinhos né?

As chaves {}

Como Expressões Regulares são bacanas, elas facilitam este tipo de tarefa também. Colocando um número entre chaves "`{}`", indica−se uma quantidade de repetições do caractere (ou metacaractere)

anterior:

```
$ egrep '^.{27}$' passwd
news:x:9:13:news:/etc/news:
$
```

Essa expressão faz o mesmo que a anterior: procura por linhas que tenham 27 caracteres, quaisquer que sejam eles. A vantagem é que basta olhar o número de repetições, não precisa ficar contando os pontinhos.

Dica nº3: .{27} é igual a 27 pontos!

Note que foi usado o egrep e não o grep. É porque as chaves fazem parte de um conjunto avançado de Expressões Regulares ("*extended*"), então o egrep lida melhor com elas. Se fosse para usar o grep normal, teria que "escapar" as chaves:

```
$ grep '^.\{27\}$' /etc/passwd
```

Feio! Vamos usar somente o egrep daqui pra frente para evitar tais escapes.

As chaves ainda aceitam intervalos, sendo possível informar uma faixa de repetições válidas, com mínimo e máximo. Para procurar por linhas que tenham de 20 a 40 caracteres:

```
$ egrep '^.{20,40}$' /etc/passwd
```

Se omitir o segundo número e manter a vírgula, fica uma faixa aberta, sem fim. Assim pode se informar repetições de "no mínimo N vezes". Para obter as linhas que possuem 40 caracteres ou mais:

```
$ egrep '^.{40,}$' /etc/passwd
```

E claro, essa repetição também pode ser usada para caracteres comuns ou outros metacaracteres além do ponto. Que tal repetir uma lista? Um bom exemplo é procurar os usuários que têm um UID ou GID de três dígitos ou mais:

```
$ egrep '[0123456789]{3,}' /etc/passwd
games:x:12:100:games:/usr/games:/sbin/nologin
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Nesse exemplo a ER pesquisou por "qualquer número repetido no mínimo três vezes", ou seja, com um mínimo de três dígitos.

O curinga .* (AND)

Quando se procura por dois trechos específicos numa mesma linha, não importando o que há entre eles, usa−se o curinga "`.*`" para significar "qualquer coisa".

Um exemplo é procurar os usuários que começam com vogais e usam o shell bash:

```
$ egrep '^[aeiou].*bash$' /etc/passwd
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Ou seja, procuramos por uma linha que comece com uma vogal e termine com a palavra "bash", não importando o que tem no meio, pode ser qualquer coisa. Essa ER funcionou mais ou menos como um AND lógico, onde só casa se tiver as duas pesquisas.

Dica nº4: .* é qualquer coisa, inclusive nada

O ou | (OR)

Para fazer o OR lógico, onde se procura por uma coisa ou outra, deve−se usar o pipe "|" e delimitar as opções com os parênteses "( )":

```
$ egrep '^(ana|carlos|acs):' /etc/passwd
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Essa ER casa apenas as linhas dos três usuários citados. Ela começa procurando no início da linha "^", depois procura ou a palavra "ana", ou a palavra "carlos", ou a palavra "acs", seguido pelo dois−pontos.

Os outros repetidores: ? * +

Outros metacaracteres que podem ser usados são o asterisco, o mais e a interrogação (chamado opcional). Eles definem quantidades e funcionam como as chaves, porém com uma sintaxe mais prática:

Meta	Nome	Equivalente	Descrição
?	opcional	{0,1}	Pode aparecer ou não (opcional)
*	asterisco	{0,}	Pode aparecer em qualquer quantidade
+	mais	{1,}	Deve aparecer no mínimo uma vez

A lista negada [^]

Ainda temos mais detalhes para ver nas listas!

Lembra do comando para pesquisar os usuários que iniciam o login com consoantes?

```
$ grep '^[bcdfghjklmnpqrstvwxyz]' /etc/passwd
```

Como é incômodo ter que colocar todas essas letras, também é possível "negar" uma lista, especificando quais os caracteres que **não são válidos** em uma posição. Assim, para buscar as consoantes, podemos dizer "busque por linhas que não comecem com vogais":

```
$ grep '^[^aeiou]' /etc/passwd
```

Veja como negar uma lista é fácil! Caso o primeiro caractere dentro dos colchetes seja um circunflexo, ele inverterá o sentido dessa lista, tornando−a uma "lista negada". Ela casará qualquer caractere, EXCETO os listados após o "^".

Mas tenha cuidado, use a lista negada com cautela. Negar alguns caracteres significa permitir **todos os outros**! Números, símbolos, TAB e espaço em branco também fazem parte de "qualquer caractere exceto vogais". Em nosso exemplo atual isso não atrapalha, mas tenha sempre em mente que essa negação é bem abrangente.

Dica nº5: Atenção, negar uma lista significa permitir "todo o resto"

O intervalo em listas [−]

Outra facilidade da lista é a possibilidade de se indicar intervalos, faixas. Basta colocar um hífen entre duas letras que ele será expandido para todas as letras existentes no intervalo. Por exemplo "`a−f`" é interpretado como "todas as letras entre a e f, inclusive", ou seja "abcdef".

Reescrevendo o exemplo de procurar por números de três dígitos ou mais, que tinha sido:

```
$ egrep '[0123456789]{3,}' /etc/passwd
```

Agora fica:

```
$ egrep '[0−9]{3,}' /etc/passwd
```

Além de letras e números, o intervalo funciona para símbolos e outros caracteres. A ordem "oficial" é a ordem que os caracteres ocupam na tabela ASCII.

Dica nº6: [0−9] é o mesmo que [0123456789]

A tabela ASCII

32		64	@	96	`	162	¢	194	Â	226	â
33	!	65	A	97	a	163	£	195	Ã	227	ã
34	"	66	B	98	b	164	¤	196	Ä	228	ä
35	#	67	C	99	c	165	¥	197	Å	229	å
36	$	68	D	100	d	166	¦	198	Æ	230	æ
37	%	69	E	101	e	167	§	199	Ç	231	ç
38	&	70	F	102	f	168	¨	200	È	232	è
39	'	71	G	103	g	169	©	201	É	233	é
40	(	72	H	104	h	170	ª	202	Ê	234	ê
41	)	73	I	105	i	171	«	203	Ë	235	ë
42	*	74	J	106	j	172	¬	204	Ì	236	ì
43	+	75	K	107	k	173	−	205	Í	237	í
44	,	76	L	108	l	174	®	206	Î	238	î
45	−	77	M	109	m	175	¯	207	Ï	239	ï
46	.	78	N	110	n	176	°	208	Ð	240	ð
47	/	79	O	111	o	177	±	209	Ñ	241	ñ
48	0	80	P	112	p	178	²	210	Ò	242	ò
49	1	81	Q	113	q	179	³	211	Ó	243	ó
50	2	82	R	114	r	180	´	212	Ô	244	ô
51	3	83	S	115	s	181	µ	213	Õ	245	õ
52	4	84	T	116	t	182	¶	214	Ö	246	ö
53	5	85	U	117	u	183	o	215	×	247	÷
54	6	86	V	118	v	184	¸	216	Ø	248	ø
55	7	87	W	119	w	185	¹	217	Ù	249	ù
56	8	88	X	120	x	186	º	218	Ú	250	ú
57	9	89	Y	121	y	187	»	219	Û	251	û
58	:	90	Z	122	z	188	¼	220	Ü	252	ü
59	;	91	[	123	{	189	½	221	Ý	253	ý
60	<	92	\	124	|	190	¾	222	Þ	254	þ
61	=	93	]	125	}	191	¿	223	ß	255	ÿ
62	>	94	^	126	~	192	À	224	à		
63	?	95	_	161	¡	193	Á	225	á		

Na dúvida, basta consultar essa tabela para ver quais são os caracteres de determinado intervalo. Por exemplo, o intervalo "`[:−@]`" engloba os caracteres `: ; < = > ? @`.

Na tabela, também podemos ver que o intervalo "a−z" não inclui os caracteres acentuados! Para procurar por letras minúsculas num texto em português, deve−se fazer "`[a−záéíóúàâêôãõç]`".

A tabela dos metacaracteres

Meta	Nome	Posicionamento
^	circunflexo	Representa o começo da linha
$	cifrão	Representa o fim da linha
		Texto
[abc]	lista	Casa as letras 'a' ou 'b' ou 'c'
[a−d]	lista	Casa as letras 'a' ou 'b' ou 'c' ou 'd'
[^abc]	lista negada	Casa qualquer caractere, exceto 'a', 'b' e 'c'
(esse|aquele)	ou	Casa as palavras 'esse' ou 'aquele'
		Quantidade I
a{2}	chaves	Casa a letra 'a' duas vezes
a{2,4}	chaves	Casa a letra 'a' de duas a quatro vezes
a{2,}	chaves	Casa a letra 'a' no mínimo duas vezes
		Quantidade II
a?	opcional	Casa a letra 'a' zero ou uma vezes
a*	asterisco	Casa a letra 'a' zero ou mais vezes
a+	mais	Casa a letra 'a' uma ou mais vezes
		Curingas
.	ponto	Casa um caractere qualquer
.*	curinga	Casa qualquer coisa, é o tudo e o nada

Atenção: Não confundir os metacaracteres com os curingas do shell. Na linha de comando usa−se "`*.txt`", "`arquivo−??.txt`" e "`arquivo.{txt,html}`" para expressar mais de um arquivo. Estes são curingas, e não metacaracteres.

Há uma similaridade e equivalência entre as ações, mas ERs são usadas em programas como `grep`, `sed` e `awk`, e os curingas são usados na linha de comando, para expandir nomes de arquivos.

Equivalência Entre Curingas do Shell e ERs

Shell	ERs
*	.*
?	.
{,}	(|)

Escapar ou não escapar?

Dependendo do aplicativo, a sintaxe das ERs pode ser diferente da do egrep. No sed e awk por exemplo (bem como no grep), as chaves e o "ou" devem ser escapados para serem especiais.

Infelizmente a implementação de Expressões Regulares pelos programas é uma selva, e não tem como fugir do decoreba dessa tabela:

Programa	**opc**	**mais**	**chaves**	**borda**	**ou**	**grupo**
awk	?	+	−	−	|	()
egrep	?	+	{,}	\b	|	()
emacs	?	+	−	\b	|	\(\)
find	?	+	−	\b	|	\(\)
gawk	?	+	{,}	\<\>	|	()
grep	\?	\+	\{,\}	\b	|	\(\)
sed	\?	\+	\{,\}	\<\>	|	\(\)
vim	\=	\+	\{,}	\<\>	|	\(\)

Para saber mais

Há programas para auxiliar o aprendizado de Expressões Regulares. O txt2regex (http://txt2regex.sourceforge.net) é um programa modo texto que constrói ERs apenas mostrando menus (em português) ao usuário, que escolhe o que quer. Já para a interface gráfica, tem o Regex Coach (http://www.weitz.de/regex−coach/), com versões para Linux e Windows.

Para aprofundar−se no assunto, leia o livro "Expressões Regulares", disponível online em http://aurelio.net/er. A lista de discussão sed−br é um lugar onde dúvidas podem ser tiradas: http://br.groups.yahoo.com/group/sed−br.